Reliability Analysis
in
Mechanical Engineering Design

G.C. Avontuur
Reliability Analysis
in
Mechanical Engineering Design

(Betrouwbaarheidsanalyse bij Werktuigbouwkundig Ontwerpen)

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. ir. K.F. Wakker,
in het openbaar te verdedigen ten overstaan van een commissie,
door het College voor Promoties aangewezen,

op dinsdag 27 juni 2000 te 10.30 uur
door Gerardus Cornelius AVON TuUR
werktuigkundig ingenieur
geboren te 's-Gravenhage.
Dit proefschrift is goedgekeurd door de promotor:
Prof. dr. ir. K. van der Werff

Samenstelling promotiecommissie:
Rector Magnificus, voorzitter
Prof. dr. ir. K. van der Werff, Technische Universiteit Delft, promotor
Prof. ir. J.L. Spoormaker, Technische Universiteit Delft
Prof. drs. ir. J.K. Vrijling, Technische Universiteit Delft
Prof. dr. ir. H.A.J. de Ridder, Technische Universiteit Delft
Prof. dr. A.R. Hale, Technische Universiteit Delft
Dr. T.J. Bedford, Technische Universiteit Delft

Dit proefschrift is het resultaat van een samenwerkingsverband tussen de afdelingen Werktuigbouw en Bouwinformatica van de Bouwdienst Rijkswaterstaat en de sectie Ontwerpkunde van de Faculteit Werktuigbouwkunde van de TU Delft.


Published and distributed by:
Delft University Press
P.O. Box 98
2600 MG Delft
The Netherlands
Telephone: +31 15 278 3254
Telefax: +31 15 278 1661
e-mail: DUP@Library.TUDelft.NL

© 2000 by G.C. Avontuur

All rights reserved. No part of the material protected by this copyright notice may be reproduced or utilised in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without written permission from the publisher: Delft University Press.

Printed in The Netherlands
Acknowledgements

Zoetermeer, April 2000

In 1993 the Mechanical Engineering department of the Construction Division of the Ministry of Public Works, Transport, and Water Management and Delft University of Technology started a research activity with two projects: reliability analysis in mechanical engineering design, and dynamic calculations in mechanical engineering design. I took part in both projects learning a great deal from them. This thesis is the product of the first part of this research program. From the second part of the program, I learned how to manage a project, how to design and implement software, and how groups of people react to innovation. I would like to thank Klaas van der Werff, Frans Remery, Frans van Dam, and Hans Jongedijk, for making this research program possible, for giving me the opportunity to work on these projects, for supporting me in difficult periods, and for everything I learned from them.

Furthermore, I would like to thank Frank Pluim, Ion Paraschiv, and Jan van Duin, who worked with me on the dynamic calculations project, for delivering a good piece of craftsmanship, for being patient and not giving up, for the good times we had, and for being my source of inspiration. I am grateful to the members of the steering group, and the users who worked with the software and responded positively. I would like to thank Bartele Kiestra, Luc Claes, Ron Beem, and Jeroen van Geijlswijk for working with me on the reliability analysis project, for supporting me, for being critical, for helping me with the reliability analysis course, and for making the project a success. For the success of the project I am also grateful to the members of the steering group, and the many colleagues who attended the reliability analysis course and responded so enthusiastically. I am grateful to Cindy Turner for reading the preceding versions of this thesis, and improving my English.
Last, but not least I would like to thank the 'home front', my parents and my wife Berna, for giving me the opportunity to study, for supporting me, and for pushing me to go to work to finish this thesis.

Gerard Avontuur
Summary

The task of a designer of a mechanical system is to determine the constructive variables, such as the type and dimensions of the components. Furthermore, it is necessary to analyse the system design in order to optimise it, and to verify if it can fulfil its functions. Normally, a designer uses a deterministic method to analyse a design. In many cases this method is satisfactory. However, in some cases this design method is not adequate, because it results in a system design that is too expensive or not reliable enough. In these cases a probabilistic approach or risk analysis may be helpful. Risk analysis experts support the designers to guarantee the costs and safety of complex systems. They analyse whether the system can perform its major functions, such as carrying load and executing motion, using a probabilistic model. Both the load carrying capacity of a system as well as the external loads can show a stochastic behaviour. This thesis concentrates on modelling the stochastic behaviour of the system; minor attention is given to the stochastic character of the loads.

The most frequently used risk analysis techniques are: Failure Modes and Effects Analysis (FMEA), Fault Tree Analysis (FTA), and event tree analysis. Fault tree analysis is the most commonly used technique, because the results of the analysis quantify the reliability, making it possible to compare design solutions, and localise the critical spots in a structure. This technique, however, is complex and labour intensive.

Therefore, the designer usually does not execute the fault tree analysis himself, but a risk analysis expert assists him in this task. Risk analysis is executed at the end of the design process, when the layout of the structure is more or less definite. In this stage of the design process, it is not desirable to introduce major changes in the structure, because they would increase the costs. In this way the results of the reliability analysis scarcely influence the layout and the quality of the design. Changes are restricted to adaptations of details.
Reliability analysis would have a major influence on the design, if it were to be applied during the conceptual design phase. This would result in more reliable and less expensive structures; a structure that is reliable in concept is less expensive than a structure that is not reliable in concept, but was improved in a later phase of the design process.

The introduction of reliability analysis in the conceptual design phase would have consequences on the design process. To achieve this, the designer and risk analyst would have to work closely together. The risk analyst would have to make fault tree analyses for many design solutions, increasing the costs of the design.

Automation can make the analysis less complex, can reduce the time of an analysis, and can prevent errors. Can automation support the reliability analysis in a way that the designer can execute the analysis himself? In this way the designer would not have to depend on the availability of a risk analysis expert. He would not have to wait for the results of the analysis, because he could execute the analysis himself, and immediately decide whether the design should be improved or not. Also, the designer could optimise the reliability of a structure by determining the reliability for a number of concepts. Thus, reliability analysis could be applied in many more cases than before, which should lead to better designs.

To verify this idea, this thesis will answer the following questions:

- Can the reliability analysis process be automated, and are the results produced by the automated method satisfactory?
- Can the automated method be integrated in the conceptual design process, and can the designer execute the reliability analysis himself?
- Does the integrated reliability analysis method improve the designs?

To answer these questions a design and reliability analysis method was developed, and implemented into software. This software package contains two parts: a component modeller that helps the designer compose a structure, and an analysis program that calculates the reliability of the structure.

The first part, the modeller, enables the designer to think in terms of mechanical components, rather than in terms of reliability analysis. The modeller stores the design as components instead of plain geometry, which makes it possible to automate several types of analysis, such as fault tree analysis. The second part, the analysis program, automatically produces the result of a fault tree analysis.

To automate the fault tree analysis it is necessary to make a more abstract description of the functions of a structure. Chapter 2 demon-
strates, that the major functions of a drive system can be decomposed into a function *carry load* and a function *execute motion*. Thus, an abstract description for only two functions is necessary. It is possible to describe these two functions with a specially adapted finite element theory. The equilibrium equations, \( \mathbf{D}^T \mathbf{\sigma} = \mathbf{f} \), describe the function *carry load*, and the continuum equations, \( \mathbf{\varepsilon} = \mathbf{D} \mathbf{u} \), describe the function *execute motion*.

The analysis program uses these equations for the reliability analysis. Assume that a particular combination of components has failed. It is possible to express this in the finite element equations. When it is not possible to find a permissible stress distribution \( \mathbf{\sigma} \) that can carry the load, or kinematically permissible displacement field \( \mathbf{u} \) that realises the desired motion, the combination of failing components is a failure mode. All failure modes are found by trying all combinations. Finally, the probability of failure for all failure modes is calculated.

**Conclusion**

It appears to be possible to automate reliability analysis. This automated method was implemented into software, that can be integrated into the design process, producing valid results. Though the software supports the designer in such a way, that he can execute the reliability analysis himself, it is still wise to consult a risk analyst for a final judgement of the results.

The automated method analyses the reliability of a structure in two steps: first, it determines the failure modes of the structure, and second it quantifies the probability of occurrence of these modes. This thesis describes two models to quantify the probability of failure: failure of discontinuous processes and failure of continuous processes. The division in continuous and discontinuous processes is not accurate enough, to describe the behaviour of various types of structures. Therefore a sub-division was made: Continuous processes can be divided into rest and action, and discontinuous processes into start and stop. In each of these sub-processes different failure mechanisms take place. Thus, it is necessary to apply different failure data in each sub-process.

The reliability analysis software was imbedded in a design system. Social aspects have a major influence on the success of the introduction for such a system. Software that is developed *bottom up* will be accepted easier than software that is developed *top down*. However, it is possible to successfully introduce software, that is developed top down. A factor for success is, that the reliability analysis tool does not change any
existing work methods. This makes the introduction of the system easier, and allows the users to accept the reliability analysis software.

The reliability analysis software has not been applied in design practice for a long period of time. Therefore, it is difficult to see whether newly designed structures change, and whether designs are improved due to the application of the reliability analysis software. To conclude this, it is necessary to monitor the application of the software over a longer period of time.
Samenvatting

De taak van een ontwerper van een werktuigbouwkundige installatie bestaat uit het bepalen van constructieve variabelen, zoals het type en de afmetingen van de componenten. Daarnaast is het noodzakelijk om de constructie te analyseren om hem te optimaliseren en om vast te stellen of hij zijn functies kan vervullen. Een ontwerper gebruikt gewoonlijk een deterministiche methode om het model te analyseren. In veel gevallen is deze methode afdoende. Echter, is sommige gevallen is deze ontwerpmethode niet toereikend, omdat zij resulteert in een ontwerp, dat te kostbaar of niet betrouwbaar genoeg is. In deze gevallen kan een probabilistische aanpak uitkomst bieden. Risico analyse experts ondersteunen de ontwerpers bij het bewaken van de kosten en veiligheid van complexe installaties. Zij analyseren, gebruik makend van een probabilistisch model, of het systeem haar hoofdfuncties, zoals het dragen van belasting en het uitvoeren van een beweging, kan uitoefenen. Zowel de draagkracht van een constructie als de externe belastingen kunnen een stochastisch karakter vertonen. Dit proefschrift legt de nadruk op het modelleren het stochastische gedrag van de installatie; het stochastische gedrag van de belasting krijgt minder aandacht.

De meest gebruikte risico analyse technieken zijn: Faalvormanalyse (Failure Modes and Effects Analysis, FMEA), Foutenboomanalyse (Fault Tree Analysis, FTA) en Gebeurtenissenboomanalyse (Event tree analysis). De Foutenboomanalyse is de meest gebruikte techniek, omdat de resultaten van de analyse de betrouwbaarheid kwantificeren, waardoor het mogelijk is om varianten te vergelijken en de kritieke plekken in een constructie aan te wijzen. Helaas is deze techniek erg complex en arbeidsintensief.

Daarom voert de ontwerper meestal de foutenboomanalyse niet zelf uit, maar hij laat zich bijstaan door een risico analyse expert. De risico analyse wordt aan het einde van het ontwerpproces uitgevoerd, als de lay-out van de installatie al min of meer definitief is. In dit stadium van het ontwerpproces is het erg lastig om grote wijzigingen door
te voeren, omdat ze de kosten zouden verhogen. Op deze manier beïnvloeden de resultaten van de betrouwbaarheidsanalyse de lay-out en de kwaliteit van het ontwerp nauwelijks. Wijzigingen worden beperkt tot aanpassingen van details.

Betrouwbaarheidsanalyse zou van grote invloed op het ontwerp zijn als zij tijdens de voorontwerp fase werd toegepast. Dit zou resulteren in betrouwbardere en minder dure constructies; een constructie die goedkoop in concept is, is minder duur dan een constructie die niet betrouwbaar in concept is, maar in een latere fase van het ontwerpproces verbeterd werd.

De introductie van betrouwbaarheidsanalyse in de voorontwerp fase zou consequenties hebben voor het ontwerpproces. Om dit te bereiken zouden de ontwerper en de risico analist nauw samen moeten werken. De risico analyst zou voor veel varianten een foutenboom analyse moeten maken, hetgeen de kosten van het ontwerp zou opdrijven.

Automatisering kan de analyse minder complex maken, kan de analysetijd bekorten en kan fouten voorkomen. Kan automatisering de betrouwbaarheidsanalyse zodanig ondersteunen, dat de ontwerper haar zelf uit kan voeren? Op die manier zou de ontwerper niet afhankelijk zijn van de beschikbaarheid van een risico analyse expert. Hij zou niet hoeven wachten op de resultaten van de analyse, omdat hij zelf de analyse kan uitvoeren en onmiddellijk kan beslissen of het ontwerp verbetering behoeft of niet. De ontwerper zou ook de betrouwbaarheid van de constructie kunnen optimaliseren door de betrouwbaarheid voor een aantal concepten te bepalen. Hierdoor zou betrouwbaarheidsanalyse in meer gevallen toegepast kunnen worden dan tot nu toe, hetgeen zou moeten leiden tot betere ontwerpen.

Om dit idee te verifiëren zal dit proefschrift de volgende vragen beantwoorden:

- Kan het betrouwbaarheidsanalyse proces geautomatiseerd worden en zijn de resultaten van de geautomatiseerde methode bevredigend?

- Kan de geautomatiseerde methode geïntegreerd worden in de voorontwerp fase en kan de ontwerper zelf de betrouwbaarheidsanalyse uitvoeren?

- Leidt de geïntegreerde betrouwbaarheidsanalyse tot betere ontwerpen?

Om deze vragen te beantwoorden is een ontwerp- en betrouwbaarheidsanalysemethode ontwikkeld en geïmplementeerd in software. Dit software pakket bestaat uit twee delen: een componenten modeller, die de ontwerper helpt om een constructie samen te stellen, en een analyse programma, dat de betrouwbaarheid van een constructie berekent.
Het eerste deel, de modeller, biedt de ontwerper de mogelijkheid om in termen van werkbuigkundige componenten te denken, in plaats van in termen van betrouwbaarheidsanalyse. De modeller slaat het ontwerp op in componenten in plaats van betekenisloze geometrie. Dit maakt het mogelijk om verschillende soorten analyse, zoals foutenboomanalyse, te automatiseren. Het tweede deel, het analyseprogramma, generereert automatisch het resultaat van een foutenboomanalyse.

Om de foutenboomanalyse te automatiseren is het noodzakelijk om een abstracte beschrijving van de functies van een installatie te maken. Hoofdstuk 2 laat zien, dat de hoofdfscties van een aandrijfssysteem gedecomponeerd kunnen worden in de functies belasting dragen en beweging uitvoeren. Hierdoor is slechts een abstracte beschrijving van twee functies nodig. Het is mogelijk om deze twee functies te beschrijven met een speciaal aangepaste eindige elementen theorie. De evenwichtsvergelijkingen, \( \mathbf{D}^T \mathbf{\sigma} = f \), beschrijven de functie belasting dragen en de continuïteitsvergelijkingen, \( \mathbf{\varepsilon} = \mathbf{D} \mathbf{\alpha} \), beschrijven de functie beweging uitvoeren.

Het analyse programma gebruikt deze vergelijkingen voor de betrouwbaarheidsanalyse. Neem aan dat een bepaalde combinatie van componenten gefaald heeft. Het is mogelijk om dit in de eindige elementen vergelijkingen uit te drukken. Als het niet mogelijk is om een toelaatbare spanningsverdeling \( \mathbf{\sigma} \), die de belasting kan dragen, of kinematisch toelaatbaar verplaatsingsveld \( \mathbf{u} \), dat de uit te voeren beweging realiseert, te vinden, dan is de combinatie van falende componenten een faalvorm. Alle faalvormen worden gevonden door alle combinaties te proberen. Tenslotte wordt de faalkans voor alle faalvormen berekend.

**Conclusie**

Het blijkt mogelijk te zijn om betrouwbaarheidsanalyse te automatiseren. De geautomatiseerde methode is geïmplementeerd in software, die in het ontwerpprocess geïntegreerd kan worden en juiste resultaten produceert. Ofschoon de software de ontwerper op zo'n manier ondersteunt, dat hij zelf de betrouwbaarheidsanalyse uit kan voeren, is het toch verstandig om een risico analist te raadplegen voor een laatste beoordeling van de resultaten.

De geautomatiseerde methode analyseert de betrouwbaarheid van een constructie in twee stappen: ten eerste bepaalt zij de faalvormen van de constructie en ten tweede kwantificeert zij de kans van optreden van deze faalvormen. Dit proefschrift beschrijft twee modellen om de faalkans te kwantificeren: het falen van discontinue processen en het falen van continue processen. De verdeling in continue processen en discon-
tinue processen is onvoldoende om het falen van verschillende typen constructies te beschrijven. Daarom is er een onderverdeling aangebracht: Continue processen kunnen verdeeld worden in rust en actie en discontinue processen in start en stop. In elk van deze subprocessen vinden andere faalmechanismen plaats. Daarom is het noodzakelijk om in ieder subprocess andere data toe te passen.

De betrouwbaarheidsanalyse software is ingebed in een ontwerpsysteem. Sociale aspecten zijn van cruciaal belang voor het succes van de introductie van zo'n systeem. Software, die bottom up ontwikkeld is, zal gemakkelijker geaccepteerd worden dan software, die top down ontwikkeld is. Echter, het is mogelijk om software die top down ontwikkeld is succesvol te introduceren. Een factor voor succes is, dat de betrouwbaarheidsanalyse software geen bestaande werkmethoden verandert. Dit maakt de introductie van het systeem gemakkelijker en helpt de gebruikers om de betrouwbaarheidsanalyse software te accepteren.

De betrouwbaarheidsanalyse software is nog niet voor langere tijd toegepast in het ontwerpproces. Daarom is het moeilijk om te zien of nieuw ontworpen constructies veranderen en of de ontwerpen verbeteren ten gevolge van de toepassing van de betrouwbaarheidsanalyse software. Om deze conclusie te trekken is het noodzakelijk om het toepassen van de software langere tijd te volgen.
# Contents

<table>
<thead>
<tr>
<th></th>
<th>Introduction</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Reliability analysis</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>Problem statement</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>Approach</td>
<td>6</td>
</tr>
<tr>
<td>1.4</td>
<td>Outline</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>Reliability in the design process</td>
<td>11</td>
</tr>
<tr>
<td>2.1</td>
<td>Design process</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>Reliability analysis techniques</td>
<td>16</td>
</tr>
<tr>
<td>2.3</td>
<td>Functions of a drive system</td>
<td>20</td>
</tr>
<tr>
<td>2.4</td>
<td>Automated reliability analysis</td>
<td>24</td>
</tr>
<tr>
<td>2.5</td>
<td>Demonstration</td>
<td>28</td>
</tr>
<tr>
<td>2.6</td>
<td>Conclusion</td>
<td>36</td>
</tr>
<tr>
<td>3</td>
<td>Finite element theory</td>
<td>39</td>
</tr>
<tr>
<td>3.1</td>
<td>Mechanical domain</td>
<td>41</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Static analysis</td>
<td>41</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Dynamics with undeformable elements</td>
<td>42</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Dynamics with deformable elements</td>
<td>43</td>
</tr>
<tr>
<td>3.2</td>
<td>Hydraulic domain</td>
<td>43</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Example: pipe and riser element</td>
<td>45</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Analytical solution</td>
<td>48</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Numerical solution</td>
<td>49</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Test program, incompressible fluid</td>
<td>50</td>
</tr>
<tr>
<td>3.2.5</td>
<td>Test program, compressible fluid</td>
<td>50</td>
</tr>
<tr>
<td>3.3</td>
<td>Multi domain systems</td>
<td>51</td>
</tr>
<tr>
<td>3.4</td>
<td>Conclusion</td>
<td>52</td>
</tr>
<tr>
<td>4</td>
<td>Description of the software package</td>
<td>53</td>
</tr>
<tr>
<td>4.1</td>
<td>Architecture</td>
<td>53</td>
</tr>
<tr>
<td>4.2</td>
<td>Component modeller</td>
<td>55</td>
</tr>
<tr>
<td>4.3</td>
<td>Finite element program and minimal cut set algorithm</td>
<td>59</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Data structures</td>
<td>60</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Classic algorithm</td>
<td>62</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Schwab's algorithm</td>
<td>65</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Physically non linear analysis</td>
<td>67</td>
</tr>
<tr>
<td>4.3.5</td>
<td>Reliability analysis, finding minimal cut sets</td>
<td>70</td>
</tr>
<tr>
<td>4.3.6</td>
<td>Reliability analysis, carrying load</td>
<td>72</td>
</tr>
<tr>
<td>4.3.7</td>
<td>Reliability analysis, executing motion</td>
<td>73</td>
</tr>
<tr>
<td>4.3.8</td>
<td>Reliability analysis, one-out-of-two, and two-out-of-two systems</td>
<td>74</td>
</tr>
<tr>
<td>4.4</td>
<td>Conclusion</td>
<td>75</td>
</tr>
<tr>
<td>5</td>
<td>Calculation of the probability of failure</td>
<td>77</td>
</tr>
<tr>
<td>5.1</td>
<td>Failure per demand versus failure rate</td>
<td>77</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Failure per demand</td>
<td>78</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Failure rate</td>
<td>79</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Unavailability</td>
<td>81</td>
</tr>
<tr>
<td>5.1.4</td>
<td>Application of the theory</td>
<td>84</td>
</tr>
<tr>
<td>5.1.5</td>
<td>A set of components</td>
<td>88</td>
</tr>
<tr>
<td>5.1.6</td>
<td>Physical model for data</td>
<td>91</td>
</tr>
<tr>
<td>5.2</td>
<td>Uncertainty</td>
<td>95</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Uncertainties, an overview</td>
<td>95</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Uncertainty in data</td>
<td>97</td>
</tr>
<tr>
<td>5.3</td>
<td>Dependent failure</td>
<td>100</td>
</tr>
<tr>
<td>5.4</td>
<td>Conclusion</td>
<td>103</td>
</tr>
<tr>
<td>6</td>
<td>Qualification, verification, validation, and evaluation</td>
<td>105</td>
</tr>
<tr>
<td>6.1</td>
<td>Qualification</td>
<td>108</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Five trusses</td>
<td>109</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Six trusses</td>
<td>109</td>
</tr>
<tr>
<td>6.1.3</td>
<td>One hydraulic cylinder</td>
<td>111</td>
</tr>
<tr>
<td>6.1.4</td>
<td>Two hydraulic cylinders</td>
<td>112</td>
</tr>
<tr>
<td>6.1.5</td>
<td>Open non return valve</td>
<td>114</td>
</tr>
<tr>
<td>6.1.6</td>
<td>Closed non return valve</td>
<td>114</td>
</tr>
<tr>
<td>6.1.7</td>
<td>Two hydraulic cylinders and a prescribed displacement</td>
<td>115</td>
</tr>
<tr>
<td>6.1.8</td>
<td>Two hydraulic cylinders, a non return valve, and a prescribed displacement</td>
<td>116</td>
</tr>
<tr>
<td>6.2</td>
<td>Verification</td>
<td>117</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Double roll-bascule bridges in Terneuzen</td>
<td>117</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Storm surge barrier in the Oosterschelde</td>
<td>121</td>
</tr>
<tr>
<td>6.3</td>
<td>Evaluation</td>
<td>122</td>
</tr>
<tr>
<td>6.4</td>
<td>Conclusion</td>
<td>125</td>
</tr>
<tr>
<td>7</td>
<td>The introduction of design systems</td>
<td>127</td>
</tr>
<tr>
<td>7.1</td>
<td>Project organisation</td>
<td>129</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>7.2</td>
<td>System architecture</td>
<td>132</td>
</tr>
<tr>
<td>7.3</td>
<td>Experiences: Strength analysis tool</td>
<td>135</td>
</tr>
<tr>
<td>7.4</td>
<td>Experiences: Reliability analysis tool</td>
<td>138</td>
</tr>
<tr>
<td>7.5</td>
<td>Conclusion</td>
<td>144</td>
</tr>
<tr>
<td>8</td>
<td>Conclusion</td>
<td>145</td>
</tr>
<tr>
<td>9</td>
<td>References</td>
<td>149</td>
</tr>
<tr>
<td>A</td>
<td>Finite element program</td>
<td>155</td>
</tr>
<tr>
<td>A.1</td>
<td>Data structures</td>
<td>155</td>
</tr>
<tr>
<td>A.1.1</td>
<td>Definition of recipe</td>
<td>155</td>
</tr>
<tr>
<td>A.1.2</td>
<td>Definition of element</td>
<td>158</td>
</tr>
<tr>
<td>A.1.3</td>
<td>Definition of node</td>
<td>159</td>
</tr>
<tr>
<td>A.1.4</td>
<td>Definition of bound, load, and stress</td>
<td>160</td>
</tr>
<tr>
<td>A.2</td>
<td>Algorithms</td>
<td>161</td>
</tr>
<tr>
<td>A.2.1</td>
<td>Main program</td>
<td>162</td>
</tr>
<tr>
<td>A.2.2</td>
<td>Calculate deformation numbers</td>
<td>162</td>
</tr>
<tr>
<td>A.2.3</td>
<td>Calculate degree of freedom numbers</td>
<td>163</td>
</tr>
<tr>
<td>A.2.4</td>
<td>Renumber</td>
<td>165</td>
</tr>
<tr>
<td>A.2.5</td>
<td>Build element matrices</td>
<td>165</td>
</tr>
<tr>
<td>A.2.6</td>
<td>Build system matrices</td>
<td>166</td>
</tr>
<tr>
<td>A.3</td>
<td>Syntax of recipe file</td>
<td>166</td>
</tr>
<tr>
<td>A.4</td>
<td>Syntax of model file</td>
<td>172</td>
</tr>
<tr>
<td>A.5</td>
<td>Syntax of initialisation file</td>
<td>175</td>
</tr>
<tr>
<td>A.6</td>
<td>Commands</td>
<td>176</td>
</tr>
<tr>
<td>B</td>
<td>Component modeller</td>
<td>179</td>
</tr>
<tr>
<td>B.1</td>
<td>Data structures</td>
<td>179</td>
</tr>
<tr>
<td>B.1.1</td>
<td>Definition of recipe</td>
<td>179</td>
</tr>
<tr>
<td>B.1.2</td>
<td>Definition of component</td>
<td>180</td>
</tr>
<tr>
<td>B.1.3</td>
<td>Definition of node</td>
<td>182</td>
</tr>
<tr>
<td>B.2</td>
<td>Syntax of recipe file</td>
<td>182</td>
</tr>
<tr>
<td>B.3</td>
<td>Syntax of model file</td>
<td>185</td>
</tr>
<tr>
<td>B.4</td>
<td>From components to elements</td>
<td>189</td>
</tr>
<tr>
<td>B.4.1</td>
<td>Mapping file</td>
<td>189</td>
</tr>
<tr>
<td>B.4.2</td>
<td>Explanation file</td>
<td>191</td>
</tr>
<tr>
<td>B.5</td>
<td>Syntax of initialisation file</td>
<td>194</td>
</tr>
<tr>
<td>B.6</td>
<td>Commands</td>
<td>194</td>
</tr>
<tr>
<td>C</td>
<td>Program architecture</td>
<td>199</td>
</tr>
<tr>
<td>C.1</td>
<td>Matrix operations</td>
<td>200</td>
</tr>
<tr>
<td>C.1.1</td>
<td>Initialising and releasing a matrix</td>
<td>200</td>
</tr>
<tr>
<td>C.1.2</td>
<td>Accessing data in a matrix</td>
<td>201</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>C.1.3</td>
<td>Matrix functions</td>
<td>201</td>
</tr>
<tr>
<td>C.2</td>
<td>2D and 3D graphics</td>
<td>203</td>
</tr>
<tr>
<td>C.2.1</td>
<td>Initialising graphics</td>
<td>203</td>
</tr>
<tr>
<td>C.2.2</td>
<td>2D graphics</td>
<td>205</td>
</tr>
<tr>
<td>C.2.3</td>
<td>3D graphics</td>
<td>207</td>
</tr>
<tr>
<td>C.3</td>
<td>User interface functions</td>
<td>210</td>
</tr>
<tr>
<td>C.3.1</td>
<td>Command line</td>
<td>210</td>
</tr>
<tr>
<td>C.3.2</td>
<td>Menus</td>
<td>213</td>
</tr>
<tr>
<td>C.3.3</td>
<td>Modes</td>
<td>215</td>
</tr>
<tr>
<td>C.4</td>
<td>Standard commands</td>
<td>215</td>
</tr>
<tr>
<td>D</td>
<td>Reliability analysis course manual for mechanical engineers</td>
<td>221</td>
</tr>
<tr>
<td></td>
<td>Curriculum vitae</td>
<td>251</td>
</tr>
</tbody>
</table>
# Table of symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Number of minimal cut sets, or number of demands per unit time.</td>
</tr>
<tr>
<td>a</td>
<td>Base of motion, or base of fluid motion.</td>
</tr>
<tr>
<td>\ddot{a}</td>
<td>Base of acceleration, or base of fluid acceleration.</td>
</tr>
<tr>
<td>A</td>
<td>Cut set, or availability.</td>
</tr>
<tr>
<td>A_i</td>
<td>Pipe or riser area.</td>
</tr>
<tr>
<td>B</td>
<td>Cut set.</td>
</tr>
<tr>
<td>B</td>
<td>Damping matrix, or deformation matrix of a multi domain system.</td>
</tr>
<tr>
<td>B_{cc}</td>
<td>Upper left corner of the deformation matrix of a multi domain system.</td>
</tr>
<tr>
<td>C</td>
<td>Set of all cut sets.</td>
</tr>
<tr>
<td>C</td>
<td>Connection matrix of two domains.</td>
</tr>
<tr>
<td>C_{fu}</td>
<td>Failure and unavailability costs.</td>
</tr>
<tr>
<td>C_h</td>
<td>Hydraulic part of the connection matrix of two domains.</td>
</tr>
<tr>
<td>C_i</td>
<td>Investment costs.</td>
</tr>
<tr>
<td>C_{imr}</td>
<td>Inspection, maintenance, and replacement costs.</td>
</tr>
<tr>
<td>C_L</td>
<td>Life cycle costs.</td>
</tr>
<tr>
<td>C_m</td>
<td>Mechanical part of the connection matrix of two domains.</td>
</tr>
<tr>
<td>C_n</td>
<td>Connection matrix of nodes.</td>
</tr>
<tr>
<td>C_o</td>
<td>Operational costs.</td>
</tr>
<tr>
<td>C_{tot}</td>
<td>Total costs.</td>
</tr>
<tr>
<td>D</td>
<td>Deformation matrix.</td>
</tr>
<tr>
<td>D_c</td>
<td>Part of the deformation matrix, that corresponds with the calculable nodal displacement vector ( \mathbf{u}_c ), or the external nodal flows ( \mathbf{V}_c ).</td>
</tr>
<tr>
<td>D_{cc}</td>
<td>Upper left corner of ( D ).</td>
</tr>
<tr>
<td>D_{co}</td>
<td>Upper right corner of ( D ).</td>
</tr>
<tr>
<td>D_o</td>
<td>Part of the deformation matrix vector, that corresponds with prescribed nodal displacement vector ( \mathbf{u}_o ), or the calculable nodal flows ( \mathbf{V}_o ).</td>
</tr>
</tbody>
</table>
$D_{bc}$ Lower left corner of $D$.
$D_{rc}$ Lower right corner of $D$.
$i$ Annual inflation.
$f$ Nodal load vector.
$f_c$ Part of the nodal load vector, that corresponds with calculable nodal displacement vector $u_c$.
$f_0$ Part of the nodal load vector, that corresponds with the prescribed nodal displacement vector $u_0$.
$f(t)$ Probability density function.
$F(t)$ Probability distribution function.
$F(t)_{comp}$ Probability distribution function of a component.
g Annual growth, or gravity acceleration.
g Load vector of a multi domain system.
$H$ System (damping) matrix.
i Number of failed components.
i(t) Number of components, that has failed in a time interval $(0, t)$.
$j$ Size of a subset of components
$K$ System (stiffness) matrix.
$L$ Element load.
l Pipe or riser length.
m Element mass.
$M$ A set of minimal cut sets.
$M$ Mass matrix.
n Number of deformations, or population size of a set of components.
$N$ Number of load cycles before yield.
n$_e$ Number of load cycles of endurance strength.
n$_l$ Maximum number of load cycles of low cycle fatigue.
p Probability of failure due to a shock, that causes dependent failure.
P Probability of failure.
$\hat{P}$ Estimator for the probability of failure.
p Nodal pressure vector.
p$_c$ Calculable nodal pressure vector.
P$_c$ Probability of failure of a minimal cut set.
P$_{cut\ set}$ Probability of failure of a minimal cut set.
P$_{comp}$ Probability of failure of a component.
P$_d$ Dependent probability of failure.
P$_f$ Probability of failure.
P$_{failure}$ Probability of failure.
P$_i$ Independent probability of failure.
\( P_{i, \text{comp}} \) Independent probability of failure of a component.
\( P_{k, \text{syst}} \) Independent probability of failure of a system of components.
\( P_{\text{tot}} \) Probability of failure of a system of components.
\( p_0 \) Prescribed nodal pressure vector.
\( q \) Probability of survival of a shock, that causes dependent failure.
\( Q \) Probability of failure per demand (query).
\( Q_{\text{comp}} \) Probability of failure per demand (query) of a component.
\( Q_{\text{start}} \) Probability of failure per demand (query) during start.
\( Q_{\text{stop}} \) Probability of failure per demand (query) during stop.
\( r \) Annual interest.
\( R \) Reduction matrix.
\( R_e \) Part of the reduction matrix that corresponds with the calculable element pressures.
\( S \) Element strength.
\( S \) Element stiffness matrix.
\( S_e \) Part of the element stiffness matrix of the mechanical domain, that corresponds with the calculable element stresses \( \sigma_e \).
\( S_{eh} \) Part of the element stiffness matrix of the hydraulic domain, that corresponds with the calculable element stresses \( \sigma_e \).
\( S_e \) Endurance strength.
\( S_{nh} \) Element stiffness matrix of a multi domain system.
\( S_y \) Yield strength.
\( t \) Time.
\( T \) Period of dynamic flow.
\( T_m \) Maintenance interval.
\( T_r \) Repair time.
\( T_t \) Test interval.
\( t_t \) Test time.
\( \ldots \) Transposed of a matrix or vector.
\( u \) Nodal displacement vector.
\( U \) Unavailability.
\( u_e \) Calculable part of the nodal displacement vector.
\( u_f \) Calculable part of the nodal velocity vector.
\( U_{\text{cut set, } f} \) Total unavailability of a minimal cut set due to failure.
\( U_f \) Unavailability due to failure.
\( U_m \) Unavailability due to maintenance.
\( U_{\text{tot, } f} \) Total unavailability of a structure due to failure.
\( u_g \) Prescribed part of the nodal displacement vector.
\( u' \) Nodal velocity vector.
\( \ddot{u} \) Nodal acceleration vector.
\( \dddot{u} \) Calculable part of the nodal acceleration vector.
$v$  Nodal volume flow vector.
$\mathbf{V}$  External nodal volume flow vector. The subscript ‘c’ denotes, that the corresponding nodal pressures can be calculated.
$\mathbf{v}_d$  Virtual displacement.
$\mathbf{v}_f$  Virtual force.
$\mathbf{V}_0$  Calculable nodal volume flow vector.
$\mathbf{V}$  Flow through an element.
$\mathbf{V}'$  Flow into an element.
$\mathbf{V}''$  Vector with flows into an element.
$\mathbf{w}$  Displacement vector of a multi domain system.
$\mathbf{w}_c$  Calculable part of the displacement vector of a multi domain system.
$x$  Transmission ratio of pump.
$y$  Year number.
$Z$  Stochastic variable.
$\alpha_j$  Alfa factor for dependent failure of a set of J components, or nodal rotation of pump shaft.
$\beta$  Beta factor for dependent failure.
$\mathbf{e}$  Element deformation vector.
$\mathbf{e}_c$  Part of the element deformation vector, that corresponds with the calculable element stresses $\sigma_c$.
$\mathbf{e}_0$  Part of the element deformation vector, that corresponds with the prescribed element stresses $\sigma_0$.
$\mathbf{e}'$  Element deformation velocity vector.
$\delta$  Variation of a vector.
$\Delta h_0$  Initial fluid level difference.
$\Delta V$  Volume change of an element.
$\mathbf{q}$  Element flow vector.
$\mathbf{q}'$  Element flow acceleration vector.
$\lambda$  Failure rate.
$\lambda_{action}$  Failure rate during action.
$\lambda_c$  Failure rate of a component.
$\lambda_d$  Dependent failure rate of a component.
$\lambda_i$  Independent failure rate of a component.
$\lambda_j$  Failure rate of any subset of size j out of a set components.
$\lambda_j''$  Failure rate of a particular subset of size j out of a set components.
$\lambda_s$  Failure rate of a subset of components.
$\lambda_{rest}$  Failure rate during rest.
$\lambda_{tot}$  Total failure rate.
\( \lambda(t) \)  
Failure rate.

\( \mu \)  
Rate at which shocks occur, that cause dependent failure.

\( \mu_L \)  
Mean value of element load.

\( \mu_N(L) \)  
Mean value of maximum number of load cycles.

\( \mu_s \)  
Mean value of element strength.

\( \mu_Z \)  
Mean value of stochastic variable \( Z \).

\( \pi \)  
Element pressure vector.

\( \pi_c \)  
Calculable part of the element pressure vector.

\( \pi_0 \)  
Prescribed part of the element pressure vector.

\( \theta \)  
Repair time.

\( \sigma \)  
Element stress vector.

\( \sigma_c \)  
Calculable part of the element stress vector.

\( \sigma_L \)  
Standard deviation of element load.

\( \sigma_N \)  
Standard deviation of maximum number of load cycles.

\( \sigma_s \)  
Standard deviation of element strength.

\( \sigma_Z \)  
Standard deviation of stochastic variable \( Z \).

\( \sigma_0 \)  
Prescribed part of the element stress vector.

\( \tau \)  
Failure time.
1 Introduction

This chapter describes the position of reliability analysis in mechanical engineering design. It outlines the most frequently used reliability analysis techniques, and describes the problem statement, the research approach, and the outline of this thesis.

1.1 Reliability analysis

A designer usually uses a deterministic method to design a structure. Standards prescribe discrete values of the loads on the structure and of the allowable material stresses in the structure. The standards guarantee, that the structure will not fail, if the loads on the structure do not exceed the allowable stresses. More precisely: the probability that a structure will fail is acceptably small, if the designer follows the standards. (The reader will know, that the loads and the allowable stresses in these standards are determined with a probabilistic method, that assumes a probability distribution of the load and of the material strength.) In many cases this method is satisfactory.

However, in a growing number of cases this design method is not adequate. Since the second world war, mankind creates structures that are far more complex than ever before, such as space ships, and nuclear power plants. It appears, that when a large number of reliable components are combined into a large structure, the result is not necessarily a reliable structure. The effect of failure of these structures can be extremely large. Failure would lead to loss of human lives, or large economic damage. Because of these effects, it is very important to secure the reliability of these structures. Therefore, risk analysis experts help the designers to guarantee the safety of these complex structures. They analyse whether the system can perform its major functions, such as carrying load and executing motion using a probabilistic model. Both the load carrying capacity of a system and the external loads can show a stochastic behaviour. This thesis concentrates on modelling the stochas-
tic behaviour of the system; minor attention is given to the stochastic character of the loads.

The risk analysis techniques evolved from analysis techniques for hazardous situations into techniques to economically optimise maintenance. Nuclear and rocket scientists first applied the techniques in the United States. The development of nuclear power plants and the storm surge barriers in the Delta Works introduced risk analysis in The Netherlands. The experts, who worked on these projects, now teach students at Delft University of Technology how to use risk analysis [41], and use the techniques to economically optimise the maintenance of the Dutch conventional power plants. They also give lectures [23] to transfer their knowledge to a larger group of people, and to stimulate the application of risk analysis. Originally, the techniques were only applied in cases which could lead to hazardous situations. Now, they are applied more and more in everyday cases.

The most frequently used risk analysis techniques are: Failure Modes and Effects Analysis (FMEA), Fault Tree Analysis (FTA), and event tree analysis. Appendix D, Beem and others [8], Henley [11], Rao [12], Knezevic [13], Van Gestel and others [22], O’Connor [44], Carter [45], Zacks [55], Lewis [56], Ushakov [57], Vrijling [60], and Ansell [64] describe these techniques in detail. This section only gives a brief description of the techniques:

- **FMEA** is a technique to make an inventory of all possible failure modes. The risk analyst divides the structure into components. Then, he enumerates the failure modes and effects of failure for each component in tables. The analyst can also make an inventory of the criticality for each failure mode. This variant of FMEA is known as Failure Modes Effects and Criticality Analysis (FMECA).

- **Fault Tree Analysis (FTA)** is a technique used to analyse one single failure mode. The technique shows which components of a structure contribute to this failure mode. The risk analyst can use this technique to quantify the probability of failure.

- **Event tree analysis** is a technique that defines all states of a structure, including the working state. A risk analyst uses this technique, when a structure can partly fail and function at the same time. In this case, he can not use a fault tree analysis, because that technique only analyses total failure of a structure. A disadvantage of event tree analysis is, that it can only be applied to small sets of components.
1.2 Problem statement

Fault tree analysis is the most commonly used technique, because the results of the analysis quantify the reliability, making it possible to compare design solutions, and localise the critical spots in a structure.

This technique, however, is complex and labour intensive. First, to construct the fault tree it is necessary to understand both the behaviour of the structure to be analysed as well as the fault tree analysis method. Second, to analyse the fault tree it is necessary to understand the rules of Boolean algebra. Third, to quantify the probability of failure, it is necessary to understand probabilistic mathematics and know the appropriate failure data of the components of the structure. Yamashina [35] says: 'However, the fault tree construction itself is a tedious, error prone, and time-consuming task.'

Therefore, the designer usually does not execute the fault tree analysis himself. A risk analysis expert assists him in this task. The expert analyses the structure at the end of the design process, when the lay out of the structure has been determined. However, in this stage of the design process, it is not possible to introduce major changes to the structure. Therefore, the results of the analysis have little influence on the design.

The reliability analysis would have a major influence on the design, if it were to be applied during the conceptual design phase. This would result in more reliable and less expensive structures; a structure that is reliable in concept is less expensive than a structure that is not reliable in concept, but was improved in a later phase of the design process.

The introduction of reliability analysis in the conceptual design phase would have consequences on the design process. To achieve this, the designer and risk analyst would have to work closely together. The risk analyst would have to make fault tree analyses for many variants, increasing the costs of the design.

Automation can make the analysis less complex, can reduce the time of an analysis, and can prevent errors. Therefore, attempts are made to automate parts of this process. Software that analyses the fault tree and quantifies the probability of failure, was implemented successfully (IsoGraph Ltd. [69], and many others). Kocza [28], Robitaile [32], O'Hern [32b], Sacks [33], Matsuoka [34], Yamashina [35], Kohda [36], and Takahashi [37] also attempt to automate the construction of the fault tree. They all use some sort of flow diagram to model the structure being analysed. Their software automatically generates a fault tree from the flow diagrams and analyses it.
This thesis moves one step further: can automation support the reliability analysis in such a way that the designer can execute the analysis himself? In this way the designer would not have to depend on the availability of a risk analysis expert. He would not have to wait for the results of the analysis. He could execute the analysis himself, and immediately decide whether the design should be improved or not. The designer could optimise the reliability of a structure by determining the reliability for a number of concepts. Thus, reliability analysis could be applied in many more cases than before. This should lead to better designs.

To verify this idea, this thesis will answer the following questions:

- Can the reliability analysis process be automated, and are the results produced by the automated method satisfactory?
- Can the automated method be integrated in the conceptual design process, and can the designer execute the reliability analysis himself?
- Does the integrated reliability analysis method improve the designs?

**Restriction**

This thesis focuses on the analysis of the construction only, but it should be clear to the reader that failure of the structure is only a (small) part of the total risk considered by a risk analyst. The paragraphs below give a global overview of the environment of a structure. The structure and its environment form a system. Vrijling [42], [43] divides this system into five layers:

1. *the natural system*: the sea, the wind;
2. *the technical system*: the structure, the bridge, the storm surge barrier, the power plant;
3. *the professional system*: the people who operate the structure, open and close the bridge or storm surge barrier or operate the power plant;
4. *the users*: the people who have benefit from the structure, people who use a bridge, use electricity;
5. and *the bystanders*: the people who do not operate nor use the structure but live near it, people who live near a factory but do not use its products.

The first two layers of this system cause risk to the next three layers of the system. Examples of risks caused by the natural system are storm, extremely high tide, and flooding. Examples of risks caused by the
technical system are the collapse of a structure, or the explosion of a structure, such as a chemical plant.

The amount of risk that is acceptable depends on the benefits, that people receive from taking the risk. The people in the fifth layer, the bystanders, receive no benefits. Thus, the acceptable risk for them is very low. The people in the fourth layer, the users, receive some benefits from using the structure, the bridge, electricity. Thus, the acceptable risk for them is higher. The people in the third layer, the professional system, receive benefits in the form of salaries. They receive the highest benefits. Thus, the acceptable risk is the highest.

The paragraph above describes the acceptable risk for an individual. A different approach is necessary for a group of individuals. If a large group of people is exposed to the same source of risk, failure could lead to a large number of casualties. Society does not accept large numbers of casualties, although the individual risk is equal for all individuals, and might be acceptably low. Therefore, the acceptable risk is lower for risk sources that can cause a large number of casualties, than for sources that can cause smaller number of casualties.

Of course it is technically possible to reduce the risk, caused by a structure, to a very low level. However, to reduce the risk, extra investments in the structure are necessary. Section 2.1 discusses the economically optimal acceptable risk.

This thesis discusses the analysis of the second layer, the technical system, the structure. Within the analysis of the structure, the analysis of the drive train is only a small part. Berenbak [63] divides the reliability analysis of the storm surge barrier in the New Waterway into many sub-analyses of subsystems, such as the civil structure, the mechanical installation, electrical installation, the software. The TAW [24], [25] also demonstrates, that the drive train is a small part within the total risk analysis. It recognises three sub-systems within the analysis: First, the water barrier - the dike. Second, the artefact in the barrier - the navigation lock, the storm surge barrier. Third, the closing operation of the artefact. However, the drive train plays an important role in the total safety, because the closing operation, in which the drive train plays an important role, consumes a large part of the allowable probability of failure.

The third and fourth layers do not only determine the allowable probability of failure, they also influence the actual probability of failure of a structure. The way a structure is maintained and used can have both a positive and negative influence on the performance of the structure. This thesis does not discuss this subject, and assumes that the cir-
cumstances for the analysed structures are all equal and comparable with the circumstances in civil structures in The Netherlands.

1.3 Approach

The term Computer Aided Design system suggests, that such a system actively supports the designer in the design process. The classic CAD systems, however, are more or less automated drawing boards; these programs support drafting, not designing. A CAD-drawing consists of geometry, and the designer, not the system, knows the interpretation of the geometry. Thus, a classic CAD system offers only limited design support.

The drafting programs do not use the extra possibilities of a computer, which can not only store the geometry, but also the semantics (meaning of objects and relationship between objects) of a design. The extra knowledge, that is stored in this manner, offers the possibility to automate design analysis. Since analysing design solutions plays an important role in the design process, integration of this task in a design system can improve the support of the designer. The system can present the results of expert calculations to the designer directly, which increases the speed of the design process. Thus, more variants of designs can be made and compared, design will take less time, so more alternatives can be studied, and the quality of the designs will be higher.

The individual components of a design and the way they are connected determine the analysis results. Therefore, a design should not consist of geometry only, but should be built up from components with meaning, elements, or objects, that are recognisable to both the computer and the designer. In this way, a design will not be built from geometric entities, such as lines, circles, and arcs, but from meaningful construction components, such as: gears, roller bearings, shafts, rack and pinions, and hydraulic cylinders. The computer can interpret the design. Schwab and Van der Werff [1] call this method Design with Discrete Components, while Wouters [2] calls this method Design with Design Elements. More generally, it is the method of Primitive Instancing (Taylor [68]).

Figure 1.1-1: Gears, a roller bearing, and a shaft are examples of discrete components.
This research project implemented this idea in software. The software contains a component modeller that helps the designer to compose a structure, and an analysis program that calculates the reliability of the structure. The modeller enables the designer to think in terms of mechanical components, rather than in terms of reliability analysis. The modeller stores the design as components instead of plain geometry. It also stores non-geometrical data, such as the probabilities of failure, with the components. The modeller is coupled directly with the fault tree analysis program, which determines the failure modes, and quantifies the (un)reliability of a design. Furthermore, it designates which components of a structure influence the reliability the most.

To automate the fault tree analysis, it is necessary to make a more abstract description of the functions of a structure. It is too ambitious to describe all functions of every type of structure. Therefore, this research project applies the theory in design practice, which was found at the Mechanical Engineering Department of the Construction Division of the Ministry of Transport, Public Works, and Water Management. This department designs drive trains of moveable bridges, lock gates, and other structures. A reliability analysis is part of the design process of these structures. The law and standards demand such an analysis of structures in water barriers. The customers also ask for a reliability and availability analysis of other structures. Until now, external experts executed the analyses. However, the design department wishes to integrate reliability analysis into the design process to improve their designs.

The major functions of a drive system can be decomposed into a function carry load and a function execute motion. Thus, an abstract description for only two functions is necessary. It is possible to describe these two functions with a specially adapted finite element theory. The equilibrium equations, \( \mathbf{D}^T \sigma = f \), describe the function carry load, and the continuum equations, \( \mathbf{\varepsilon} = \mathbf{D} \mathbf{u} \), describe the function execute motion. This idea was inspired by Besseling [10] and Van der Werff [17].

The analysis program uses these equations for the reliability analysis. Assume that a particular combination of components has failed. It is possible to express this in the finite element equations. When it is not possible to find a permissible stress distribution \( \sigma \) that can carry the load, or kinematically permissible displacement field \( \mathbf{u} \) that realises the desired motion, the combination of failing components is a failure mode. All failure modes are found by trying all combinations. Finally, the probability of failure for all failure modes is calculated.
A fault tree analysis can only describe the reliability concerning one function of a system at a time for one specific geometrical configuration and load case. A reliability analysis should consider all significant functions of a structure. Therefore, the designer should make a separate fault tree analysis for each function.

1.4 Outline

Chapter 2 describes the design process, various reliability analysis techniques, and the functions of a drive train. It ends with an overview of the automated reliability analysis method.

Chapter 3 discusses the finite element method and an adaptation to the method to describe the behaviour of hydraulic components in drive systems. The classic finite element theory is appropriate to describe the behaviour of mechanical components, such as a shaft, roller bearings, and a pair of gears. However, the theory is not suitable to model hydraulic components, such as pipes and valves. To model these components, this thesis uses a specially adapted finite element theory, that is analogue to the theory for mechanical components. The theory for mechanical components describes the mechanical domain, while the theory for hydraulic components describes the hydraulic domain. Schwab and Van der Werff [1] combine both domains into a multi domain system. This thesis also combines both domains, but with a slightly different method.

Chapter 4 describes in detail the software that was built. It introduces a new concept of program architecture for finite element software, which fully separates the implementation of the algorithms from the implementation of the elements. Usually, the programmer stores the definition of the elements in separate source files. The main program calls the routines in these files, for instance: a routine to number nodes, a routine to number deformations, a routine to build matrices, and many more. This program architecture requires the implementation of the same general theory in many different places in the source code. The programmer must adjust all source files, when changes are made to an algorithm or a new one is developed. When creating a new element, many algorithms must be copied and adjusted. Adjusting one algorithm is easily forgotten.

This research project produced both new algorithms, such as an algorithm to determine the failure modes of a structure, as well as special elements, such as a gear element (Rankers [70]). The architecture of the developed program tackles the problem described above. The architec-
ture separates the implementation of the theory and the implementation of the elements in such a way, that changing or developing a new algorithm can not introduce errors in the elements, and vice versa, changing or developing a new element can not introduce errors in the algorithms. To achieve this, the elements are defined in recipe files (see Figure 1.4-1). The program reads these files at run time. The advantages of the recipe file architecture are:

- Extendability. The recipe files make it extremely easy to add new element types to the software package. To add a new element, it is sufficient to create a new recipe file - a text file that defines the element deformation and stiffness matrices - and place it in the correct directory. It is not necessary to recompile the program.

- An open structure. Since the recipe files are text files, the designers are able to verify whether the implementation of components and elements is correct.

- Maintainability. The recipe files make it possible to modify the elements to remove bugs and implement new features, without recompiling the program.

- Software reliability. The recipe files separate the element definition from the algorithms in the core of the program. Therefore, it is not possible to introduce bugs and errors in the algorithms by maintenance on the elements, and vice versa, it is not possible to introduce bugs and errors in the elements by maintenance on the algorithms.

Chapter 5 describes methods to quantify the minimal cut sets - the failure modes - of a fault tree. These methods are not new; enough literature is available describing various models to quantify minimal cut sets. It is not always clear when and how these models should be applied. Chapter 5 gathers theory from various sources, and describes how it can be applied in the analysis of drive trains of civil structures.

Van Gestel and others [22] describe a method to calculate the unavailability from a fault tree. They consider unavailability only. This is an important parameter for the reliability of continuously operating
systems. However, for discontinuous operating systems the probability of failure is also of importance. Furthermore, Van Gestel and others only consider continuous and discontinuous processes. However, it is not sufficient to consider these two processes only (Kiestra [6] and Van Geijlswijk [9]). Failure of a system can occur during four phases: rest, start, action, and stop. Rest and action are continuous processes, while start and stop are discontinuous processes. Different failure mechanisms take place in each phase. Example: a roller bearing fails due to corrosion during the rest phase, and it fails due to fatigue during the action phase. Different data should be applied for each failure mechanism, though both are continuous processes. Chapter 5 demonstrates how to calculate the probability of failure and the unavailability in each phase. Depending on the phase different models must be used.

Chapter 6 elaborates on the quality aspects of this project. It divides the quality of the project results into three categories: the quality of the results of the algorithms that were developed, the ability to model and analyse real structures, and the influence of the software on the designs of these structures. Chapter 6 refers to the quality control of these categories as: qualification, verification, and evaluation.

Chapter 7 describes the introduction of the automated reliability analysis software in the design process. The introduction of software does not always succeed. Chapter 7 discusses the factors for success (and failure). It is only possible to successfully introduce such a design system, when the users accept the system and are willing to use it. A test group of future users can ease the acceptance of the system. Test group members should be carefully selected: they should be representative for the total group of users, they should be held in high esteem by their colleagues, and they should be willing to accept changes - young people accept changes easier.
2 Reliability in the design process

This chapter discusses the role of reliability analysis in the design process of drive systems at the Mechanical Engineering Department of the Construction Division of the Ministry of Transport, Public Works, and Water Management. It describes the design process of drive trains of civil structures, various reliability analysis techniques, and the functions of a drive system. These are steps used to find an automated reliability analysis method, which is also described and demonstrated in this chapter.

2.1 Design process

This section describes the design process for drive trains of infrastructural works, such as moveable bridges, lock gates, and storm surge barriers. First, the total realisation process, of which the design process is a part, is outlined. Then, the role of reliability analysis in the design process is discussed.

■ Realisation and design process

The realisation of infrastructural works is a complex process. To reduce the complexity, the project manager decomposes this job into phases. A phase is a bundle of tasks, that can be controlled in complexity, time, and costs. The Construction Division usually separates the realisation of infrastructural works into the phases:

- initiation;
- conceptual design;
- design;
- specification;
- construction;
- and after care.

In the initiation phase the principal and a project manager of the Construction Division define the project goal. The project manager then composes a project plan. This plan describes the project goal, approach, project risks (FMEA), activities in all project phases, use of personnel, costs, quality, information flow during the project, and project organisation. The principal accepts the plan, if he agrees with it.

In the conceptual design phase the project team creates a number of design solutions. A designer determines the main dimensions of each solution. Examples: diameter and length of shafts, ratio of the gearbox, and the power of the engine. He validates these dimensions with a static analysis, and optimises each solution to reduce the costs. The result of this phase is a rough drawing, a rough static analysis report, and a rough cost estimate for each solution. The project team evaluates the solutions and recommends to the principal which one is the most optimal solution. The principal chooses one solution for further detailing in the design phase.

In the design phase the chosen variant is developed further. A designer determines the dimensions of the details of the chosen solution. He validates these dimensions with a static analysis, and optimises the structure to reduce the costs. For some structures, a risk analysis expert executes a fault tree analysis, after the designer has determined all the details. The result of this phase is a detailed (set of) drawings, a detailed static analysis report, an accurate cost estimate, and sometimes a fault tree analysis report.

In the specification phase the project team finishes the detailing of the design, prepares the contract, and sends out requests for proposal to different contractors. The request contains both drawings and specifications in text.

In the (beginning of the) construction phase a contractor is chosen. During the construction phase, the contractor works on further detailing of the design, and builds the infrastructural work. The contractor submits the detailed drawings and calculations for the approval of the project team. At the end of this phase, the contractor delivers the structure to the project manager, and the project manager delivers the structure to the principal.

In the after care phase the project team composes a maintenance plan. This plan advises the principal about maintenance and replacement intervals of components, and a conservation schedule for steel parts.
As demonstrated above, designing takes place in various phases. It takes place in the conceptual design phase, the design phase, the specification phase, and the construction phase. However, the major design decisions are made in the first two phases: the conceptual design phase, and the design phase. Figure 2.1-1 describes the design process in these two phases.

**CONCEPTUAL DESIGN PHASE**

- **generate solutions**
  - calculate costs
  - static analysis
  - optimise

  (main) iteration loop

- choose solution

**DESIGN PHASE**

- **optimise**

- calculate costs
  - static analysis
  - optimise

  main iteration loop

- fault tree analysis

  secondary iteration loop

- accept design

---

*Figure 2.1-1: Current conceptual design and design phase.*

---

**The role of reliability analysis**

The reliability analysis takes place at the end of the design process, after the layout of the structure is determined. The role of the analysis is to verify if the reliability of the structure satisfies the demanded reliability.

However, at the end of the design process, it is costly, or there is not enough time available to introduce major changes in the structure. Therefore, the results of the analysis have little influence on the design.
The reliability analysis would have a major influence on the design, if it were to be applied during the conceptual design. This would result in more reliable and less expensive structures; a structure that is reliable in concept is less expensive than a structure that is not reliable in concept, but that was improved in a later phase of the design process.

It is known that about eighty percent of the costs of a design are determined in the first twenty percent of the time. This means that eighty percent of the costs are determined in the conceptual design phase. Therefore, the price of a concept is a major issue in pre-design. However, the most affordable design usually is not the safest solution. Therefore, a designer should not make his decisions based on costs alone. Not only should a designer produce a cost-effective design, he should also make a reliable design. Reliability analysis gives a designer the possibility to make decisions based on more aspects than costs only, since it gives him the opportunity to optimise the reliability and costs.
In conceptual design, the role of reliability analysis is to optimise the costs and the reliability. In Figure 2.1-2, the optimisation is placed in the main iteration loop of the conceptual design process.

**Life cycle costs**

Beem [19] writes about the optimisation between the costs and the reliability: 'A major issue for the selection of solutions is the price-performance ratio. Besides the building costs, the price of the design solutions is composed of all the costs during the life cycle of a structure, which are necessary to fulfil its functional demands.' The life cycle costs, $C_L$, are composed of three parts: investment costs, $C_i$, the present value of the operational costs, $C_o$ (operators, energy), and the present value of the inspection, maintenance, and replacement costs, $C_{i,m,r}$.

\[
C_L = C_i + \sum_{y=0}^{n} \frac{C_o + C_{i,m,r}}{(1 + r - i - g)^y}
\]  

(2.1-1)

In this equation $r$ is the annual interest, $i$ is the annual inflation, and $g$ is the annual growth. An usual value for $r - i - g$ is 0.05.

To compare life cycle costs and reliability, Beem also expresses failure and unavailability in costs. The theory is applied to navigation locks. Equation (2.1-1) is expanded with the expectation value of the costs for waiting ships due to the unavailability of the structure. Causes of unavailability are external circumstances, such as extremely high tide, failure and repair, and maintenance. It is also possible to expand the formula with the expectation values of other costs caused by failure or unavailability, $C_{i,u}$ such as costs from flooding. The most optimal solution is the one with the lowest total costs, $C_{tot}$.

\[
C_{tot} = C_i + \sum_{y=0}^{n} \frac{C_o + C_{i,m,r} + C_{i,u}}{(1 + r - i - g)^y}
\]  

(2.1-2)

Vrijling [42] states that the investment costs, $C_i$, the inspection, maintenance, and replacement costs, $C_{i,m,r}$, and the costs of failure or unavailability, $C_{i,u}$, are a function of the probability of failure $P_f$ and the repair time $T_r$. To reduce the costs of failure, it is necessary to reduce the probability of failure. This reduction can be achieved by improving the structure, or intensifying inspection and maintenance, which increases the investment costs and the maintenance costs. The economically optimal structure is the structure with minimal total costs:
\[
\min(C_{\text{tot}}) = \min \left( C_i(P_f) + \sum_{y=0}^{n} \frac{C_o + C_{i,m,r}(P_t) + C_{i,u}(P_f, T_r)}{(1 + r - i - g)^y} \right) \tag{2.1-3}
\]

Figure 2.1-3 is a graphical representation of equation (2.1-2) and shows the optimum, represented by minimal total costs.

![Graph showing the economically optimal structure is the structure with minimal total costs.](image)

2.2 Reliability analysis techniques

The most frequently used reliability analysis techniques are: Failure Modes and Effects Analysis (FMEA), Fault Tree Analysis (FTA), and event tree analysis. Appendix D, Beem and others [8], Henley [11], Rao [12], Knezovic [13], Van Gestel and others [22], O’Connor [44], Carter [45], Zacks [55], Lewis [56], Usakov [57], Vrijling [60], and Ansell [64] describe these techniques in detail. This section gives an overview of these techniques.

- **Failure modes and effects analysis**

FMEA is a technique to make an inventory of all possible failure modes. A risk analyst applies this technique, when he wants to answer the question: 'What can go wrong with this structure?'. With a systematic approach he tries to find the complete set of failure modes of the structure. The result of the analysis is a long list of all possible failure modes.

Method: Divide the structure into components. Then, list the failure modes of each component in a table.

Not all modes in this table are of importance. Some of them could merely cause an inconvenience, while others could cause a disaster. To separate the important failure modes from the irrelevant ones, the ana-
lyst can list the effect and criticality of each failure mode. This solution of FMEA is known as Failure Modes Effects and Criticality Analysis (FMECA). Table 2.2-1 shows the relationship between effect, probability, and criticality.

A designer should take precautions to prevent the occurrence of critical failure modes. Whether or not a failure mode is critical, depends on the effect and on the probability of occurrence. When either the effect of a failure mode is small, or probability of occurrence is low, the criticality is low. However, when both the effect is large, and the probability of occurrence is high, the criticality is high. The risk analyst can express the effect and criticality of a failure mode qualitatively with high or low. He can also express it quantitatively with a 1 for low and a 10 for high. He can then obtain a quantitative value for the criticality by multiplying the effect and the probability: effect × probability = criticality. Table 2.2-2 is an example of a table format to use in a FMECA.

<table>
<thead>
<tr>
<th>Component</th>
<th>Failure mode</th>
<th>Effect</th>
<th>Probability</th>
<th>Criticality</th>
</tr>
</thead>
</table>

Table 2.2-2: Example of the header of a table for a FMECA.

## Fault tree analysis

Fault Tree Analysis (FTA) is a technique to analyse one single failure mode. A risk analyst applies this technique, when he wants to answer the question: ‘What causes this failure mode?’. The result of the analysis is a set of combinations of failing components, which cause failure of the total structure, along with a quantification of the probability of failure. The results of the analysis also show which combination of components has the highest contribution to the total probability of failure.

Figure 2.2-1: Hydraulic system.
Method: Define the failure mode that has to be analysed. This failure mode is called the top event, because it is placed at the top of the fault tree. Examples of top events: ‘the bridge does not open’, ‘the bridge does not close’. First, trace the location or component, where the top event occurs. Failure can occur by failure in the component itself, or by failure in bounding subsystems. Example: an electric wire can fail because it is broken, or because the power supply does not work. Second, place a component or set of components, whose failure results in the top event, in the fault tree. Third, consider all bounding subsystems. Place a subsystem or a set of subsystems in the fault tree, whose failure contributes to the failure of the previous component. Repeat this algorithm for this subsystem. Systematically follow the

Figuur 2.2-2: Fault tree of the hydraulic system in Figure 2.2-1.
entire system. Quotation: 'Follow the copper road.' [R.W. van Otterloo (KEMA)], which means follow the electrical wires to find the components that must be put in the fault tree. Figure 2.2-1 shows a scheme of a hydraulic system. The fault tree in Figure 2.2-2 describes the failure of the hydraulic system.

A combination of failing components, which causes failure of the total structure, and contains as few components as possible, is called a minimal cut set. The fault tree is reduced with Boolean algebra to find these sets. Figure 2.2-3 shows a reduced fault tree; the minimal cut sets are:

- barrel D empty;
- pump A broken;
- pipe B blocked and pipe C blocked.

![Diagram showing fault tree](image)

*Figure 2.2-3: The fault tree of Figure 2.2-2 after reduction with Boolean algebra.*

It is necessary to quantify the fault tree to find the probability of failure for the total structure, and which components and minimal cut sets have the highest contribution to this probability of failure. A designer can improve parts of the structure, if he judges that the probability of failure is too high. Chapter 5 demonstrates how to quantify a fault tree. If failure of the components occurs per demand, and if failure of the components is independent, the probability of failure for the minimal cut sets can be calculated by multiplying the probabilities of failure of the components. The probability of failure for the total structure can be calculated by adding the probabilities of failure of the minimal cut sets.
Event tree analysis

Event tree analysis is a technique that defines all possible states of a structure, including the working state. A risk analyst applies this technique when a structure can partially fail and function (at a reduced level) at the same time. For more about partial mission completion this thesis refers to Bedford [61]. Example: a pumping-engine with two pumps can still produce flow when one of the pumps does not work. However, it does not produce to its full capacity. It fails, but it also works. The risk analyst can not use fault tree analysis in this situation, since fault tree analysis only analyses total failure of a structure. The result of the event tree analysis is a set of all possible states in which a structure can exist, along with the probability of occurrence for each state.

Method: Divide the structure into components. Define failure modes for each component. The occurrence of a failure mode is an event. Define the failure modes along the horizontal axis of a diagram. Then, draw a horizontal binary tree. The tree splits up into a true branch and a false branch at each event. Determine in which state the structure is for all paths through the branches, and calculate the probability of each branch at the end of the tree. Combine the probabilities of the branches that have the same state.

An event tree grows in width exponentially. Therefore, an event tree analysis can only be applied to small sets of components.

2.3 Functions of a drive system

This section derives an abstract description of the functions of a drive system. This description is necessary to automate reliability analysis.

Usually a drive system is part of a larger structure. It supports this structure to fulfil one of its functions. This thesis focuses on drive systems in infrastructural works, such as moveable bridges, navigation locks, and storm surge barriers. This section considers the functions of these structures, and how the drive trains contribute to these functions.
The main functions of a moveable bridge are to allow road traffic to pass over water, and ships to pass under the bridge. Besides these main functions the bridge has secondary functions, such as preventing people and vehicles from falling into the water, preventing collisions with ships, and being aesthetically pleasing. Table 2.3-1 shows the contribution of the drive train to these functions.

<table>
<thead>
<tr>
<th>Main functions</th>
<th>Contribution of drive train</th>
</tr>
</thead>
<tbody>
<tr>
<td>Letting road traffic pass over water.</td>
<td>If the bridge is open, the drive train closes the bridge.</td>
</tr>
<tr>
<td>Letting ships pass under the bridge.</td>
<td>The drive train opens the bridge.</td>
</tr>
<tr>
<td><strong>Secondary functions</strong></td>
<td></td>
</tr>
<tr>
<td>Preventing people from falling into the water.</td>
<td>None, barriers prevent people from falling into the water when the bridge is open (of course the barrier has a drive train also, but that is not considered here), and rails prevent people from falling into the water when the bridge is closed.</td>
</tr>
<tr>
<td>Preventing vehicles from falling into the water.</td>
<td>None, barriers prevent vehicles from falling into the water when the bridge is open (of course the barrier has a drive train also, but that is not considered here).</td>
</tr>
<tr>
<td>Preventing collisions with ships.</td>
<td>The drive train opens the bridge. Traffic lights show the captain whether he can pass through.</td>
</tr>
<tr>
<td>Being aesthetically pleasing.</td>
<td>None.</td>
</tr>
</tbody>
</table>

*Table: 2.3-1: The functions of a moveable bridge and the contribution of the drive train to these functions.*

The main functions of a navigation lock are being a water barrier, and letting ships through. The lock has secondary functions, such as a passage for people over the lock doors, preventing people from falling into the water, and preventing collisions with ships. Table 2.3-2 shows the contribution of the drive train to these functions.
<table>
<thead>
<tr>
<th>Main functions</th>
<th>Contribution of drive train</th>
</tr>
</thead>
<tbody>
<tr>
<td>Being a water barrier.</td>
<td>The drive trains close the doors.</td>
</tr>
<tr>
<td>Letting ships through.</td>
<td>The drive trains open and close the lock doors.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Secondary functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Giving passage to people over the lock doors.</td>
<td>The drive trains close the doors.</td>
</tr>
<tr>
<td>Preventing people from falling into the water.</td>
<td>None. Rails on the lock doors prevent people from falling into the water, when they walk over the doors. The shores of the navigation lock should be clean and clear; there should not be objects that could make people stumble, and fall into the water.</td>
</tr>
</tbody>
</table>

| Preventing collisions with ships.      | The drive trains open the doors. Traffic lights show the captain whether he can pass through. |

Table: 2.3-2: The functions of a navigation lock and the contribution of the drive train to these functions.

The main functions of a storm surge barrier are (of course) being a water barrier, letting water in and out, and letting ships through. The lock has secondary functions, such as preventing collisions with ships. Table 2.3-3 shows the contribution of the drive train to these functions.

<table>
<thead>
<tr>
<th>Main functions</th>
<th>Contribution of drive train</th>
</tr>
</thead>
<tbody>
<tr>
<td>Being a water barrier.</td>
<td>The drive train closes the door.</td>
</tr>
<tr>
<td>Letting water in and out.</td>
<td>The drive train opens the door, and holds it in the open position.</td>
</tr>
<tr>
<td>Letting ships through.</td>
<td>The drive train opens the door, and holds it in the open position.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Secondary functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Preventing collisions with ships.</td>
<td>The drive train opens the door, and holds it in the open position. Traffic lights show the captain whether he can pass through.</td>
</tr>
</tbody>
</table>

Table: 2.3-3: The functions of a storm surge barrier and the contribution of the drive train to these functions.

In all of these cases, the drive train supports the structure to fulfil its main functions. The function of the drive train is to move (open, close) or hold some sort of substructure. To move this substructure,
the drive train needs to execute a motion and carry a load. An example of a storm surge barrier demonstrates that the functions of a drive train can be decomposed in two functions: *carrying load*, and *executing motion*.

Figure 2.3-1 shows a model of a storm surge barrier. The drive train of the barrier has to keep it open when there is no storm. When there is a storm, the drive train closes the barrier. When the storm is over, the drive train opens the barrier.

Table 2.3-4 demonstrates how to decompose the functions of the drive train of the storm surge barrier into the functions *carrying load* and *executing motion*.

<table>
<thead>
<tr>
<th>Function</th>
<th>Direction of load</th>
<th>Direction of motion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keep barrier open</td>
<td>Down</td>
<td>-</td>
</tr>
<tr>
<td>Close barrier</td>
<td>Down</td>
<td>Down</td>
</tr>
<tr>
<td>Open barrier</td>
<td>Down</td>
<td>Up</td>
</tr>
</tbody>
</table>

*Table 2.3-4: Decomposition of the functions of the drive train of the storm surge barrier of Figure 2.3-1 into the basic functions 'carrying load' and 'executing motion'.*

Drive systems fulfil many tasks, such as opening a bridge, closing a bridge, moving a lock gate, and holding a lock gate against water pressure. It is possible to decompose the task *opening a bridge* into the functions *executing a motion* and *carrying load* (acceleration, wind). It is also possible to decompose the other examples into these functions. This leads to the assumption, that all functions of a drive system can be decomposed into: 1 *carrying load*, 2 *executing motion*.

**Assumption**

All functions of a drive system can be decomposed into:

1. carrying load;
2. executing a motion.
It is possible to generalise this assumption: the major function of a drive system is to supply energy. This generalised function can be divided into several specialised sub-functions: Load and displacement (motion) define energy. Pressure and flow also define energy. Thus, the functions of a drive system are supplying pressure, and producing flow. Temperature and heat also define energy. Thus, the functions of a drive system are maintaining a temperature, and producing heat. Voltage and current also define energy. Thus, the functions of a drive system are maintaining a voltage, and producing current. In this thesis we focus on the functions carrying load, and executing motion.

A failure modes and effects analysis (FMEA) is not useful to analyse the reliability concerning these functions. It does not describe the cause of failure for a single function. This type of analysis is useful to find failure for more functions, than the two that are described above.

An event tree may be an useful technique to describe failure of these two functions. However, this technique only produces practical results for small sets of components.

Therefore, a fault tree analysis is the most appropriate technique used to analyse the reliability of a drive train concerning these functions. Not able to execute motion, and not able to carry load are the top events of the fault trees.

2.4 Automated reliability analysis

It was concluded in the section above, that fault tree analysis is the most appropriate technique to use in the conceptual design phase when analysing the reliability of the functions of a structure. Several researchers have attempted to automate fault tree analysis. This section briefly describes their approach, followed by a discussion of the first attempt [5], and final approach used to automate the fault tree analysis for this research project.

■ Various approaches to automate fault tree analysis

Kócza [28] uses system models, a type of flow diagram, to automate fault tree analysis. The system model consists of blocks, which represent the components of a structure. The blocks are connected with arrows, that represent the signal flows. Kócza’s software automatically translates the system model into a fault tree.

Robitaille [32] and O’Hern [32b] use the Digraph Matrix Analysis technique (DMA) developed by Sacks [33], to automate fault tree analy-
sis. Digraph Matrix Analysis is based on Petri Net analysis, a type of flow diagram analysis. The diagram consists of nodes, that schematically represent the components of a structure. Arrows representing flows determine the flow direction between the nodes. Sacks' software automatically translates the Digraph into a fault tree.

Matsuoka [34] uses the Go-Flow methodology. The Go-Flow methodology gives a time dependent, probabilistic, and logic representation of a structure. A Go-Flow chart consists of signal generators, logical operators, and probabilistic operators. This methodology replaces the fault tree analysis technique, since it has extra functionality.

Yamashina [35] uses semantic flow diagrams to automate fault tree analysis. The diagram consists of nodes, that schematically represent the components of a structure, and nodes that represent the flows in the structure. Arrows represent the relationships between the nodes. Yamashina's software automatically translates the flow diagrams into a fault tree.

Similar to Robitaille and O'Hern, Kohda [36] uses Petri Nets to automate fault tree analysis. Kohda also automates event tree analysis using Petri Nets.

Takahashi [37] developed a component modeller to define a structure. The modeller automatically translates the component model into a flow diagram, and it automatically translates the flow diagram into a fault tree. The component model is an extra layer above the flow diagram. This layer gives the designer or risk analyst a more comprehensive representation of the structure.

All of the references above use some type of flow diagram to automate fault tree analysis. The flow diagrams give an abstract representation of the structure. The fault tree generation programs help prevent the risk analyst from making errors. However, they do not yet help the designer. For the designer, these programs change the problem from 'how to make a fault tree' to 'how to make a flow diagram'. Takahashi [37] solved this problem. He adds an extra shell to his software, that hides the flow diagram. This shell uses components such as pumps, and valves to model a structure. Thus, the designer does not need to have knowledge about flow diagrams to execute a fault tree analysis.

However, Takahashis' components are very schematic. They do not support the designer in defining the geometry of a structure. The next step is to introduce components with real dimensions to support the designer in defining both the geometry of a structure as well as the reliability analysis. This research project attempts to achieve this.
First attempt

The first attempt to automate fault tree analysis for this research project also used a flow diagram approach [5]. A prototype of an automated fault tree analysis program was developed to test the approach. This software contains a component modeller, which uses components such as hydraulic cylinders, pipes, valves, and pumps, and a fault tree generator.

The component modeller stores the model of a structure as components and nodes. The nodes of a component are connection points. If two components have a node in common, they are connected.

Besides the topology of the structure, the nodes also describe the function of the components: the nodes of a pipe should produce a flow, the nodes of a cylinder should produce a flow and cause a displacement. The nodes describe the interface between connected components. Figure 2.4-1 is an example of a pipe component.

Failure of a component can occur by failure in the component itself, or by failure of an adjacent component. Each component has a local fault tree, that describes failure by the failure in the component itself, and failure by the interfaces on the nodes. Figure 2.4-2 is an example of a local fault tree for a pipe component.

To create the global fault tree of a structure the program starts at the node, that describes the main function of the structure. This node is connected to a component. The local fault tree of this component is the base of the global fault tree of the structure. The local fault tree refers to the interfaces with the component nodes. The program links the local fault trees of other components on these positions in the fault tree. A recursive algorithm walks through the total structure in this way. Some structures contain loops, for instance in piping. Figure 2.4-3
demonstrates this with a simple hydraulic scheme. To prevent the algorithm from running in circles in the structure, and entering an algorithmic infinite loop, the program sets flags at each branch of pipes and other components.

Although the algorithm produces valid fault trees for all structures that were tested, it is too complex to prove that it produces valid results in all cases, since the structures often have a network architecture, and the way the algorithm navigates through the structure is not always clear. Therefore, a more robust algorithm was developed. This algorithm is described below.

**Final approach**

In section 2.3 the functions of a drive train were decomposed into the functions *carrying load* and *executing motion*. The finite element method can be adapted to mathematically describe these functions. The equilibrium equations, that describe the equilibrium of the external loads \(f\) on nodal points and the internal element stresses \(\sigma\), will be used to represent the function *carrying load*:

\[
D^T \sigma = f
\]

(2.4-1)

The continuum equations, that describe the relationships between nodal displacements \(u\) and deformations \(\varepsilon\), will be used to represent the function *executing motion*:

\[
\varepsilon = D u
\]

(2.4-2)

Each elemental stress represents a failure mode of an element that might fail to fulfil the function *carrying load*, if it is broken. In this case the finite element equations prescribe an elemental stress to be equal to zero. Each elemental deformation represents a failure mode of an element that might fail to fulfil the function *executing motion*, if it is blocked. In this case the finite element equations prescribe an elemental deformation equal to zero.
Assume that a certain combination of components has failed. It is possible to express this in the finite element equations above. When it is not possible to find a permissible stress distribution \( \sigma \), or kinematically permissible displacement field \( u \), the combination of failing components is a failure mode. Otherwise, it is not. All failure modes are found by trying all combinations. A structure fails, when equation (2.4-1) or (2.4-2) cannot be solved for the prescribed loads and displacements. An unsolvable set of equations is the mathematical representation of a failing drive train.

This theory was implemented in an algorithm, that finds the minimal cut sets of a fault tree by simply trying all combinations of failing elemental deformations and stresses. A combination is a failure mode, when the system of equations cannot be solved.

The algorithm was implemented into a software package. The design process is as follows: The designer creates design concepts, and defines the functions of the structure. Then, he decomposes these functions into the basic functions carrying load and executing motion. The software helps the designer to build a component model of the design concepts. For each function a separate model is necessary. The software automatically translates the component model into a finite element model, generates the minimal cut sets of the fault tree, and quantifies the fault tree.

Like Takahashi's [37] program, the software package of this research project has an extra component shell, that hides the finite elements model from the designer. This shell uses components, such as pumps and valves, to model a structure. Thus, the designer does not need to have knowledge about finite element theory to execute a fault tree analysis. Figure 2.4-4 shows which part of the modelling and analysis process is supported by the software package.

![Figure 2.4-4: Automated design process.](image)
2.5 Demonstration

This section demonstrates how to optimise a drive train with the software, that was developed in this project.

Four different types of drive trains for the storm surge barrier of Figure 2.5-1 are evaluated. Table 2.3-4 shows that five analyses per solution are necessary. The function *keep barrier open* is translated into a model with a downward load on the gate. The brake in the drive train carries this load. The functions *open barrier* and *close barrier* are each translated to two models per function. One model with a downward load on the gate. The engine of the drive train carries this load. In the second model, the gate has a prescribed motion downward or upward.

*Figure 2.5-1: Component model of a storm surge barrier. This model represents the function 'keep barrier open'. The more abstract description of this function is 'carrying load'.*
**Type I**

![Diagram](image1)

*Figure 2.5-2: Type I of the drive train for the storm surge barrier.*

The first type of the drive train contains a single engine, a coupling, a single brake, a coupling and a gearbox.

**Type II**

![Diagram](image2)

*Figure 2.5-3: Top view of Type I.*

*Figure 2.5-4: Type II of the drive train for the storm surge barrier.*

*Figure 2.5-5: Top view of Type II.*

The second type of drive train contains double engines, couplings and brakes.
Type III

Figure 2.5-6: Type III of the drive train for the storm surge barrier.

The third type of drive train contains double engines, an extra gearbox and two extra couplings.

Type IV

Figure 2.5-8: Type IV of the drive train for the storm surge barrier.

The fourth type of drive train contains double engines, one extra coupling and no extra gearbox.

Results

The paragraphs below show a part of the output for the analysis of Type II for the function carry load while the barrier is being closed:
Analysis of start action period.

The probability of failure is the probability, that a structure has failed in one cycle of periods (rest, start, action, or stop). The periods, that were taken into account, are mentioned above. The unavailability refers to the total time of a cycle. This is the sum of the rest and action interval, regardless if these intervals were analysed.

Rest interval: \(7.2 \times 10^2\) h,
Amount of queries per unit time: \(1.4 \times 10^3\) 1/h,
Action interval: \(1.0 \times 10^0\) h.

Probability of failure (independent failure): \(3.9 \times 10^{-6}\)
Unavailability (independent failure): \(2.6 \times 10^{-7}\)
Probability of failure (dependent failure, beta: 0.1): \(2.9 \times 10^{-5}\)
Unavailability (dependent failure, beta: 0.1): \(1.1 \times 10^{-6}\)

5%, 50%, and 95% certainty intervals.
Probability of failure (independent failure): \(3.2 \times 10^{-6} 3.8 \times 10^{-6} 4.3 \times 10^{-6}\)
Unavailability (independent failure): \(9.4 \times 10^{-8} 2.2 \times 10^{-7} 4.9 \times 10^{-7}\)
Probability of failure (dependent failure, beta: 0.1): \(1.6 \times 10^{-5} 2.7 \times 10^{-5} 4.4 \times 10^{-5}\)
Unavailability (dependent failure, beta: 0.1): \(3.8 \times 10^{-7} 9.1 \times 10^{-7} 2.0 \times 10^{-6}\)

Minimal cut set:

Probability of failure (independent failure): \(5.1 \times 10^{-8}\)
Unavailability (independent failure): \(1.7 \times 10^{-9}\)
Probability of failure (dependent failure): \(2.5 \times 10^{-5}\)
Unavailability (dependent failure): \(8.3 \times 10^{-7}\)

5%, 50%, and 95% certainty intervals.
Probability of failure (independent failure): \(1.4 \times 10^{-8} 4.0 \times 10^{-8} 9.4 \times 10^{-8}\)
Unavailability (independent failure): \(4.3 \times 10^{-10} 9.4 \times 10^{-10} 3.5 \times 10^{-9}\)
Probability of failure (dependent failure, beta: 0.1): \(1.2 \times 10^{-5} 2.3 \times 10^{-5} 4.0 \times 10^{-5}\)
Unavailability (dependent failure, beta: 0.1): \(1.7 \times 10^{-7} 6.2 \times 10^{-7} 1.7 \times 10^{-6}\)

cname rotgen cmpnr 1 ename beam elmrn 1 definr 2
Failure mode: Torque engine to low.
Lrest 7.5e-06 Qstart 1.5e-04 Laction 7.5e-05 Qstop 1.5e-05 Trepair 2.4e+01
srest 3.7e-06 sstart 7.5e-05 saction 3.8e-05 sstopt 7.5e-06 srepair 1.2e+01

cname rotgen cmpnr 10 ename beam elmrn 10 definr 2
Failure mode: Torque engine to low.
Lrest 7.5e-06 Qstart 1.5e-04 Laction 7.5e-05 Qstop 1.5e-05 Trepair 2.4e+01
srest 3.7e-06 sstart 7.5e-05 saction 3.8e-05 sstopt 7.5e-06 srepair 1.2e+01

Minimal cut set:

Probability of failure (independent failure): \(3.9 \times 10^{-6}\)
Unavailability (independent failure): \(2.6 \times 10^{-7}\)
Probability of failure (dependent failure): \(3.9 \times 10^{-6}\)
Unavailability (dependent failure): \(2.6 \times 10^{-7}\)

5%, 50%, and 95% certainty intervals.
Probability of failure (independent failure): \(3.2 \times 10^{-6} 3.8 \times 10^{-6} 4.5 \times 10^{-6}\)
Unavailability (independent failure): \(9.2 \times 10^{-8} 2.2 \times 10^{-7} 4.9 \times 10^{-7}\)
Probability of failure (dependent failure, beta: 0.1): \(3.2 \times 10^{-6} 3.8 \times 10^{-6} 4.5 \times 10^{-6}\)
Unavailability (dependent failure, beta: 0.1): \(9.2 \times 10^{-8} 2.2 \times 10^{-7} 4.9 \times 10^{-7}\)

cname gearbox2 cmpnr 13 ename gear2 elmrn 62 definr 1
Failure mode: Collapse of gears due torsional momenting going shaft.
Lrest 2.0e-09 Qstart 3.9e-06 Laction 2.0e-09 Qstop 3.9e-07 Trepair 4.8e+01
srest 1.0e-09 sstart 3.9e-07 saction 1.0e-09 sstopt 3.9e-08 srepair 2.4e+01

The output on the reliability analysis software contains the following data:

- The probability of failure and the unavailability for the total structure and for each minimal cut set.

- Since the probability data are not exactly known, the software also calculates the 5%, 50%, and 95% certainty intervals of these values.
- The output presents the minimal cut sets sorted by the probability of failure, which was calculated assuming that failure of the components is (partly) dependent. With this information, the designer can determine which components contribute the most to the probability of failure.

- The software not only calculates all values assuming that failure of the components is totally independent, but also assuming that failure is (partly) dependent.

Table 2.5-1 to 2.5-4 show the results of the analyses. The results in Table 2.5-1 and 3 take into account, that failure of the components in a minimal cut set is (partly) dependent. The results in Table 2.5-2 and 4 take into account, that the failure of components is totally independent. The differences between the two calculation methods are a factor 10.

<table>
<thead>
<tr>
<th>Type</th>
<th>function</th>
<th>P carry load</th>
<th>highest contrib.</th>
<th>P exec. motion</th>
<th>highest contrib.</th>
<th>P total</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>hold</td>
<td>1.0*10^{-3}</td>
<td>brake</td>
<td>-</td>
<td>-</td>
<td>1.0*10^{-3}</td>
</tr>
<tr>
<td>I</td>
<td>close</td>
<td>2.3*10^{-4}</td>
<td>engine</td>
<td>3.5*10^{-4}</td>
<td>brake</td>
<td>5.8*10^{-4}</td>
</tr>
<tr>
<td>I</td>
<td>open</td>
<td>2.3*10^{-4}</td>
<td>engine</td>
<td>3.5*10^{-4}</td>
<td>brake</td>
<td>5.8*10^{-4}</td>
</tr>
<tr>
<td>II</td>
<td>hold</td>
<td>1.2*10^{-4}</td>
<td>brake</td>
<td>-</td>
<td>-</td>
<td>1.2*10^{-4}</td>
</tr>
<tr>
<td>II</td>
<td>close</td>
<td>2.9*10^{-5}</td>
<td>engine</td>
<td>6.7*10^{-4}</td>
<td>brake</td>
<td>7.0*10^{-4}</td>
</tr>
<tr>
<td>II</td>
<td>open</td>
<td>2.9*10^{-5}</td>
<td>engine</td>
<td>6.7*10^{-4}</td>
<td>brake</td>
<td>7.0*10^{-4}</td>
</tr>
<tr>
<td>III</td>
<td>hold</td>
<td>1.0*10^{-3}</td>
<td>brake</td>
<td>-</td>
<td>-</td>
<td>1.0*10^{-3}</td>
</tr>
<tr>
<td>III</td>
<td>close</td>
<td>4.5*10^{-5}</td>
<td>engine</td>
<td>3.8*10^{-4}</td>
<td>brake</td>
<td>4.3*10^{-4}</td>
</tr>
<tr>
<td>III</td>
<td>open</td>
<td>4.5*10^{-5}</td>
<td>engine</td>
<td>3.8*10^{-4}</td>
<td>brake</td>
<td>4.3*10^{-4}</td>
</tr>
<tr>
<td>IV</td>
<td>hold</td>
<td>1.0*10^{-3}</td>
<td>brake</td>
<td>-</td>
<td>-</td>
<td>1.0*10^{-3}</td>
</tr>
<tr>
<td>IV</td>
<td>close</td>
<td>3.0*10^{-5}</td>
<td>engine</td>
<td>3.7*10^{-4}</td>
<td>brake</td>
<td>4.0*10^{-4}</td>
</tr>
<tr>
<td>IV</td>
<td>open</td>
<td>3.0*10^{-5}</td>
<td>engine</td>
<td>3.7*10^{-4}</td>
<td>brake</td>
<td>4.0*10^{-4}</td>
</tr>
</tbody>
</table>

Table 2.5-1: Probabilities of failure for the functions of the drive train for the storm surge barrier. The results are found by taking into account that failure of the components in all second order minimal cut sets is partly dependent. \( P_{total} = P_{carry\ load} + P_{exec\ motion} \)
<table>
<thead>
<tr>
<th>solution</th>
<th>function</th>
<th>P carry load</th>
<th>highest contrib.</th>
<th>P exec. motion</th>
<th>highest contrib.</th>
<th>P total</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>hold</td>
<td>1.0*10^3</td>
<td>brake</td>
<td>-</td>
<td>-</td>
<td>1.0*10^3</td>
</tr>
<tr>
<td>I</td>
<td>close</td>
<td>2.3*10^4</td>
<td>engine</td>
<td>3.5*10^4</td>
<td>brake</td>
<td>5.8*10^4</td>
</tr>
<tr>
<td>I</td>
<td>open</td>
<td>2.3*10^4</td>
<td>engine</td>
<td>3.5*10^4</td>
<td>brake</td>
<td>5.8*10^4</td>
</tr>
<tr>
<td>II</td>
<td>hold</td>
<td>4.3*10^6</td>
<td>brake</td>
<td>-</td>
<td>-</td>
<td>4.3*10^6</td>
</tr>
<tr>
<td>II</td>
<td>close</td>
<td>3.9*10^6</td>
<td>engine</td>
<td>6.7*10^4</td>
<td>brake</td>
<td>6.7*10^4</td>
</tr>
<tr>
<td>II</td>
<td>open</td>
<td>3.9*10^6</td>
<td>engine</td>
<td>6.7*10^4</td>
<td>brake</td>
<td>6.7*10^4</td>
</tr>
<tr>
<td>III</td>
<td>hold</td>
<td>1.0*10^3</td>
<td>brake</td>
<td>-</td>
<td>-</td>
<td>1.0*10^3</td>
</tr>
<tr>
<td>III</td>
<td>close</td>
<td>2.0*10^5</td>
<td>engine</td>
<td>3.8*10^4</td>
<td>brake</td>
<td>4.0*10^4</td>
</tr>
<tr>
<td>III</td>
<td>open</td>
<td>2.0*10^5</td>
<td>engine</td>
<td>3.8*10^4</td>
<td>brake</td>
<td>4.0*10^4</td>
</tr>
<tr>
<td>IV</td>
<td>hold</td>
<td>1.0*10^3</td>
<td>brake</td>
<td>-</td>
<td>-</td>
<td>1.0*10^3</td>
</tr>
<tr>
<td>IV</td>
<td>close</td>
<td>5.0*10^6</td>
<td>engine</td>
<td>3.7*10^4</td>
<td>brake</td>
<td>3.8*10^4</td>
</tr>
<tr>
<td>IV</td>
<td>open</td>
<td>5.0*10^6</td>
<td>engine</td>
<td>3.7*10^4</td>
<td>brake</td>
<td>3.8*10^4</td>
</tr>
</tbody>
</table>

Table 2.5-2: Probabilities of failure for the functions of the drive train for the storm surge barrier. The results are found by taking into account that failure of the components in all second order minimal cut sets is independent. $P_{\text{total}} = P_{\text{carry load}} + P_{\text{exec. motion}}$.

<table>
<thead>
<tr>
<th>solution</th>
<th>function</th>
<th>U carry load</th>
<th>highest contrib.</th>
<th>U exec. motion</th>
<th>highest contrib.</th>
<th>U total</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>hold</td>
<td>8.0*10^4</td>
<td>brake</td>
<td>-</td>
<td>-</td>
<td>8.0*10^4</td>
</tr>
<tr>
<td>I</td>
<td>close</td>
<td>7.5*10^6</td>
<td>engine</td>
<td>8.6*10^4</td>
<td>brake</td>
<td>1.6*10^5</td>
</tr>
<tr>
<td>I</td>
<td>open</td>
<td>7.5*10^6</td>
<td>engine</td>
<td>8.6*10^4</td>
<td>brake</td>
<td>1.6*10^5</td>
</tr>
<tr>
<td>II</td>
<td>hold</td>
<td>7.5*10^5</td>
<td>brake</td>
<td>-</td>
<td>-</td>
<td>7.5*10^5</td>
</tr>
<tr>
<td>II</td>
<td>close</td>
<td>1.1*10^6</td>
<td>engine</td>
<td>1.6*10^5</td>
<td>brake</td>
<td>1.7*10^5</td>
</tr>
<tr>
<td>II</td>
<td>open</td>
<td>1.1*10^6</td>
<td>engine</td>
<td>1.6*10^5</td>
<td>brake</td>
<td>1.7*10^5</td>
</tr>
<tr>
<td>III</td>
<td>hold</td>
<td>8.0*10^4</td>
<td>brake</td>
<td>-</td>
<td>-</td>
<td>8.0*10^4</td>
</tr>
<tr>
<td>III</td>
<td>close</td>
<td>2.8*10^6</td>
<td>engine</td>
<td>1.0*10^5</td>
<td>brake</td>
<td>1.3*10^5</td>
</tr>
<tr>
<td>III</td>
<td>open</td>
<td>2.8*10^6</td>
<td>engine</td>
<td>1.0*10^5</td>
<td>brake</td>
<td>1.3*10^5</td>
</tr>
<tr>
<td>IV</td>
<td>hold</td>
<td>8.0*10^4</td>
<td>brake</td>
<td>-</td>
<td>-</td>
<td>8.0*10^4</td>
</tr>
<tr>
<td>IV</td>
<td>close</td>
<td>1.2*10^6</td>
<td>engine</td>
<td>9.3*10^4</td>
<td>brake</td>
<td>1.1*10^5</td>
</tr>
<tr>
<td>IV</td>
<td>open</td>
<td>1.2*10^6</td>
<td>engine</td>
<td>9.3*10^4</td>
<td>brake</td>
<td>1.1*10^5</td>
</tr>
</tbody>
</table>

Table 2.5-3: Unavailability due to failure for the functions of the drive train for the storm surge barrier. The results are found by taking into account that failure of the components in all second order minimal cut sets is partly dependent. $U_{\text{total}} = U_{\text{carry load}} + U_{\text{exec. motion}}$.
<table>
<thead>
<tr>
<th>solution</th>
<th>function</th>
<th>$U_{\text{carry load}}$</th>
<th>highest contrib.</th>
<th>$U_{\text{exec. motion}}$</th>
<th>highest contrib.</th>
<th>$U_{\text{total}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>hold</td>
<td>$8.0 \times 10^4$</td>
<td>brake</td>
<td>-</td>
<td>-</td>
<td>$8.0 \times 10^4$</td>
</tr>
<tr>
<td>I</td>
<td>close</td>
<td>$7.5 \times 10^4$</td>
<td>engine</td>
<td>$8.6 \times 10^4$</td>
<td>brake</td>
<td>$1.6 \times 10^5$</td>
</tr>
<tr>
<td>I</td>
<td>open</td>
<td>$7.5 \times 10^4$</td>
<td>engine</td>
<td>$8.6 \times 10^4$</td>
<td>brake</td>
<td>$1.6 \times 10^5$</td>
</tr>
<tr>
<td>II</td>
<td>hold</td>
<td>$2.7 \times 10^4$</td>
<td>brake</td>
<td>-</td>
<td>-</td>
<td>$2.7 \times 10^4$</td>
</tr>
<tr>
<td>II</td>
<td>close</td>
<td>$2.6 \times 10^7$</td>
<td>engine</td>
<td>$1.6 \times 10^5$</td>
<td>brake</td>
<td>$1.6 \times 10^5$</td>
</tr>
<tr>
<td>II</td>
<td>open</td>
<td>$2.6 \times 10^7$</td>
<td>engine</td>
<td>$1.6 \times 10^5$</td>
<td>brake</td>
<td>$1.6 \times 10^5$</td>
</tr>
<tr>
<td>III</td>
<td>hold</td>
<td>$8.0 \times 10^4$</td>
<td>brake</td>
<td>-</td>
<td>-</td>
<td>$8.0 \times 10^4$</td>
</tr>
<tr>
<td>III</td>
<td>close</td>
<td>$2.0 \times 10^6$</td>
<td>engine</td>
<td>$1.0 \times 10^5$</td>
<td>brake</td>
<td>$1.2 \times 10^5$</td>
</tr>
<tr>
<td>III</td>
<td>open</td>
<td>$2.0 \times 10^6$</td>
<td>engine</td>
<td>$1.0 \times 10^5$</td>
<td>brake</td>
<td>$1.2 \times 10^5$</td>
</tr>
<tr>
<td>IV</td>
<td>hold</td>
<td>$8.0 \times 10^4$</td>
<td>brake</td>
<td>-</td>
<td>-</td>
<td>$8.0 \times 10^4$</td>
</tr>
<tr>
<td>IV</td>
<td>close</td>
<td>$3.3 \times 10^7$</td>
<td>engine</td>
<td>$9.3 \times 10^6$</td>
<td>brake</td>
<td>$9.6 \times 10^6$</td>
</tr>
<tr>
<td>IV</td>
<td>open</td>
<td>$3.3 \times 10^7$</td>
<td>engine</td>
<td>$9.3 \times 10^6$</td>
<td>brake</td>
<td>$9.6 \times 10^6$</td>
</tr>
</tbody>
</table>

Table 2.5-4: Unavailabilities due to failure for the functions of the drive train for the storm surge barrier. The results are found by taking into account that failure of the components in all second order minimal cut is independent. $U_{\text{total}} = U_{\text{carry load}} + U_{\text{exec. motion}}$.

Type I is the simplest: there are no redundant components in the drive train. The other types contain double or redundant components to improve the reliability. Type II shows, that improving the reliability of the function **hold** (keep barrier open) has a negative effect on the other functions. Double brakes improve the reliability of the function **hold**. However, they have a negative effect on the functions **close** and **open**. Types III and IV show, that it is only possible to make minor improvements to the reliability of the functions **close** and **open**. The brake has a negative effect on the reliability of these functions. However, it is not possible to remove it from the drive train, because it is necessary for the function **hold**. Type III and IV show, that removing unnecessary components improves the reliability.

Which type should the designer choose? The analysis results show that a type, that is optimal for one function, is not optimal for another function. In the case of the storm surge barrier, it is sensible to give the highest priority to the function **close**. Thus, the designer will not select Type II. Type III is more expensive and less reliable than type IV, because it contains extra components. Thus, the designer will not select Type III either. Type IV is slightly more reliable (a factor 1½) than Type I. However, it is also more expensive, since it contains an extra engine, coupling, and a more complex gearbox. It is necessary to study
the results of the analysis more closely to be able to choose between Type I and IV. Table 2.5-5 and 6 show the distribution of the probabilities of failure of these types.

<table>
<thead>
<tr>
<th>Type</th>
<th>5% boundary</th>
<th>50% boundary</th>
<th>95% boundary</th>
</tr>
</thead>
<tbody>
<tr>
<td>P carry load</td>
<td>$1.1 \times 10^4$</td>
<td>$2.0 \times 10^4$</td>
<td>$3.6 \times 10^4$</td>
</tr>
<tr>
<td>P exec. motion</td>
<td>$1.6 \times 10^4$</td>
<td>$3.1 \times 10^4$</td>
<td>$6.1 \times 10^4$</td>
</tr>
</tbody>
</table>

Table 2.5-5: Distribution of the probabilities of failure for the function 'close' of Type I. The results are found by taking into account that failure of the components in all second order minimal cut sets is partly dependent.

<table>
<thead>
<tr>
<th>Type IV</th>
<th>5% boundary</th>
<th>50% boundary</th>
<th>95% boundary</th>
</tr>
</thead>
<tbody>
<tr>
<td>P carry load</td>
<td>$1.7 \times 10^5$</td>
<td>$2.8 \times 10^5$</td>
<td>$4.6 \times 10^5$</td>
</tr>
<tr>
<td>P exec. motion</td>
<td>$1.7 \times 10^4$</td>
<td>$3.3 \times 10^4$</td>
<td>$6.3 \times 10^4$</td>
</tr>
</tbody>
</table>

Table 2.5-6: Distribution of the probabilities of failure for the function 'close' of Type IV. The results are found by taking into account that failure of the components in all second order minimal cut sets is dependent.

The distribution function of the total probability of failure is the sum of the distributions of the probability that the drive train is not able to carry the load, and execute the motion. It is not possible to calculate this function directly from Table 2.5-5 and 6. However, it is possible to conclude that the deviation of the probability of failure for Type IV is smaller than the deviation of the probability of failure for Type I. It is also possible to conclude that the 50% boundary of the probability of failure for Type IV is lower than the 50% boundary for Type I. This means, that Type IV is indeed more reliable than Type I. However, the advantage in reliability is small.

Whether to choose Type IV over Type I depends on the difference in costs and the demanded probability of failure. If the calculated probability of failure for Type I is much larger than the demanded probability of failure, the extra investment of Type IV is not necessary. However, if Type I is barely safe enough, it depends on the costs. If Type I is not safe enough it is necessary to choose Type IV.

2.6 Conclusion

This chapter demonstrates, that it is possible to improve a design by applying reliability analysis. It shows how to quantify failure and unavailability in costs, and how to compare them with investment costs.
to improve the reliability. A better design is not necessarily a design with a higher reliability, but a design with optimal life cycle and failure costs.

It is possible to automate reliability analysis, and to integrate the automated method into the design process. Fault tree analysis appears to be a suitable technique to analyse the reliability of the functions of a structure, and to compare the reliability of different solutions of a structure.

It is possible to give an abstract description of the most important functions of a drive train. This chapter demonstrates how all major functions of a drive train can be decomposed into the basic functions carrying load, and executing motion. The finite element theory describes these functions. The minimal cut sets of a fault tree can be derived directly from the finite elements model.

The theory was embedded in a software package, that helps a designer create design solutions. The software produces comprehensive results, and helps the designer improve the structure, and choose the most optimal solution.
3 Finite element theory

Chapter 2 demonstrates, that the finite elements theory describes the major functions of a drive train, carrying load and execution motion, and that it is possible to express failure in terms of these equations. This chapter describes the finite element method and a special adjustment to this method to describe the behaviour of hydraulic components in drive systems.

The classic finite element theory adequately describes the behaviour of mechanical components, such as a shaft, roller bearings, and a pair of gears. However, the theory is insufficient to model hydraulic components, such as pipes and valves. To model these components, this thesis describes a specially adapted finite element theory, that is analogous to the theory for mechanical components. The theory for mechanical components describes the mechanical domain, and the theory for hydraulic components describes the hydraulic domain. Schwab and Van der Werff [1] combine both domains into a multi domain system. This thesis also combines both domains, but with a slightly different method.

The mechanical domain includes structures such as a bridge and a lock gate, and components of drive trains such as a rack and pinion and a panama-wheel. An analogous description of the hydraulic domain includes structures such as pipes and channels, and components of a hydraulic drive system, such as valves. Figure 3-1 shows an example of a hydraulic drive system. It is also possible to include the simple flow of gasses, for instance in a compressor. Extra research is needed to show, whether a similar method can be found to include the electrical and heat domain.

Often, energy passes through various domains from its source to the place of use. Example: The energy in the wind is in the hydraulic domain. A windmill transfers the energy to the mechanical domain using a system of gears and shafts. This system drives a generator, which transfers the energy to the electrical domain. The electrical do-
main transports the energy to the place where it is needed. There, an electric motor transfers it back to the mechanical domain with a shaft that drives a pump. The pump transfers the energy to the hydraulic domain using a system of pipes and valves. Finally, a hydraulic cylinder transfers the energy to the mechanical domain, to move a lock gate.

A multi domain system describes the energy transitions between different domains. This thesis only describes the interconnection between the mechanical and the hydraulic domain. It applies this theory on pumps and hydraulic cylinders, as used in infrastructural works (see Figure 3-2). An interesting subject for further research would be to combine the mechanical, hydraulic, and thermal domain to describe a steam or a internal combustion engine.

Figure 3-1: The power unit for the hydraulic drive system for lock gates in Enkhuizen, The Netherlands. The pump in this unit transfers energy from the mechanical to the hydraulic domain.

Figure 3-2: Hydraulic cylinder in Enkhuizen lock, The Netherlands. The cylinder transfers energy from the hydraulic to the mechanical domain.

Section 3.1 and 3.2 describe the mechanical and hydraulic domain respectively. A pipe flow program was developed to test the theory of this domain. The output of this pipe flow program is also presented in section 3.2. Section 3.3 combines both domains into a multi domain system, using a slightly different method than Schwab.
3.1 Mechanical domain

A system is in equilibrium when the internal virtual deformation energy is equal to the externally applied virtual energy for all kinematically admissible deformations and displacements. The virtual deformation energy is described by the internal element stresses $\sigma$ and virtual element strains $\delta \varepsilon$, and the externally applied virtual energy by the external forces $f$ and virtual displacements $\delta u$. According to d'Alembert's principle the mass forces can be included using a mass matrix $M$, multiplied by the acceleration vector $\ddot{u}$.

$$<\sigma, \delta \varepsilon> = <f - M\ddot{u}, \delta u>$$  \hfill (3.1-1)

Equation (3.1-2) describes the kinematically admissible displacements $\delta u$ and deformations $\delta \varepsilon$:

$$\delta \varepsilon = D \delta u$$  \hfill (3.1-2)

Substitution of (3.1-2) in (3.1-1) gives:

$$<\sigma, D \delta u> = <f - M\ddot{u}, \delta u>$$  \hfill (3.1-3)

Now, (3.1-3) must be true for all variations of $\delta u$ only.

$$<D^T \sigma, \delta u> = <f - M\ddot{u}, \delta u>$$  \hfill (3.1-4)

$$<D^T \sigma - f + M\ddot{u}, \delta u> = 0$$  \hfill (3.1-5)

Since this equation is true for all possible variations of $\delta u$:

$$M\ddot{u} + D^T \sigma = f$$  \hfill (3.1-6)

3.1.1 Static analysis

For a static analysis all accelerations $\ddot{u}$ are zero: $\ddot{u} = 0$. For linear elasticity:

$$\sigma = S \varepsilon$$  \hfill (3.1-7)

The vector $\sigma$ contains stresses, and $\varepsilon$ contains deformations. Apply (3.1-2) to (3.1-7):

$$\sigma = S D u$$  \hfill (3.1-8)

Apply equation (3.1-8) to (3.1-6), replacing the virtual displacements $\delta u$ by real displacements $u$, and a classic finite elements equation is obtained:
\[ D^T S D u = f \]  

(3.1-9)

In most cases boundary conditions must be applied to solve this system. Separate the vector with nodal displacements \( u \) into a vector with prescribed displacements \( u_0 \), and a vector with calculable displacements \( u_c \). On the nodes with calculable nodal displacements, forces \( f_c \) can be prescribed. Apply this to (3.1-9) to obtain a non singular set of equations:

\[ D_c^T S D_c u_c = f_c \]  

(3.1-10)

### 3.1.2 Dynamics with undeformable elements

All deformations \( \varepsilon \) of undeformable elements are zero. When elements are undeformable, the vector \( u_0 \), with calculable nodal displacements, can be reduced to a (set of vectors) \( \bar{a} \), forming a new base for the system's nodal displacements:

\[ u_c = R \bar{a} \]

\[ \ddot{u}_c = R \ddot{\bar{a}} \]  

(3.1-11)

\( R \) is a reduction matrix, formed by the kernel of matrix \( D_c \); the zero vector is excluded. \( D_c \) is a part of matrix \( D \) from equation (3.1-2). \( D_c \) describes the relationship between nodal displacements, that can be calculated, and element deformations. The kernel of matrix \( D_c \) is a set of vectors, that describe nodal displacements without element deformations.

It should be clear to the reader that (3.1-11) is only true for systems, that are geometrically linear. For geometrically non linear systems \( D_c \) is a function of the nodal positions \( u_c \), which are a function of time: \( D_c = D_c(u_c(t)) \). For these systems it is not possible to determine constant bases of nodal velocities and accelerations, since the equations contain second order terms.

Apply equation (3.1-11) to (3.1-6):

\[ M R \ddot{\bar{a}} + D_c^T \sigma = f_c \]  

(3.1-12)

The vector \( f_c \) contains loads acting on the nodes with calculable nodal displacements. \( D_c R = 0 \) by definition; \( \iff R^T D_c^T = 0 \). Multiply (3.1-12) with \( R^T \):

\[ R^T M R \ddot{\bar{a}} = R^T f_c \]  

(3.1-13)

These equations can be solved by a time integration algorithm.
3.1.3 Dynamics with deformable elements

Assume a (linear) relation $S$ between stresses $\sigma$ and deformations $\varepsilon$, and a (linear) relation $B$ between stresses $\sigma$ and deformation velocities $\varepsilon'$:

$$\sigma = B \varepsilon' + S \varepsilon \tag{3.1-14}$$

Apply (3.1-2) to (3.1-14):

$$\sigma = B \mathbf{D} \mathbf{u}' + S \mathbf{D} \mathbf{u} \tag{3.1-15}$$

It should be clear to the reader that (3.1-15) is only true for systems, that are geometrically linear in time. To analyse other systems with these equations, it is necessary to analyse the system in several decisive positions. Apply (3.1-15) to (3.1-6):

$$\mathbf{M} \ddot{\mathbf{u}} + \mathbf{D}^T \mathbf{B} \mathbf{D} \mathbf{u}' + \mathbf{D}^T \mathbf{S} \mathbf{D} \mathbf{u} = \mathbf{f} \tag{3.1-16}$$

Apply boundary conditions to find equations for nodes with calculable nodal displacements:

$$\mathbf{M}_e \dddot{u}_e + \mathbf{D}_e^T \mathbf{B} \mathbf{D}_e \mathbf{u}_e' + \mathbf{D}_e^T \mathbf{S} \mathbf{D}_e \mathbf{u}_e = f_e \tag{3.1-17}$$

Substitute: $\mathbf{D}^T \mathbf{B} \mathbf{D} = \mathbf{H}$, $\mathbf{D}^T \mathbf{S} \mathbf{D} = \mathbf{K}$

$$\mathbf{M}_e \dddot{u}_e + \mathbf{H} \mathbf{u}_e' + \mathbf{K} \mathbf{u}_e = f_e \tag{3.1-18}$$

These equations can be solved by a time integration algorithm.

3.2 Hydraulic domain

The equations for the hydraulic domain are analogues to the equations for the mechanical domain. Equation (3.2-1) describes the equilibrium between external energy and internal deformation energy. The nodal pressures $p$, and virtual external nodal volume flows $\delta V$ describe the external energy. The element pressures $\pi$ minus ‘acceleration pressures’ $\varphi''$, along with the virtual internal flows and volume changes $\delta \varphi$ describe the internal energy. Equation (3.2-1) must be true for all variations of $\delta V$ and $\delta \varphi$, which satisfy (3.2-2).

$$<p, \delta V> = <\pi - M\varphi'', \delta \varphi> \tag{3.2-1}$$

$$\delta V = D\delta \varphi \tag{3.2-2}$$

Applying (3.2-2) to (3.2-1) results in (3.2-3) and (3.2-4), which must be true for all variations of $\delta \varphi$ only:
\[ \langle p, D\delta \varphi \rangle = \langle \pi - M\varphi'', \delta \varphi \rangle \]  \hspace{1cm} (3.2-3)

\[ \langle D^T p - \pi + M\varphi'', \delta \varphi \rangle = 0 \]  \hspace{1cm} (3.2-4)

Since this equation is true for all possible variations of \( \delta \varphi \):

\[ M\varphi'' + D^T p = \pi \]  \hspace{1cm} (3.2-5)

By applying the boundary conditions a solvable system is found. The volume flow vector \( \mathbf{V} \) can be separated into a vector \( \mathbf{V}_c \) with prescribed external nodal flows, and a vector \( \mathbf{V}_o \) containing calculable nodal flows (the subscript 'c' denotes that the corresponding nodal pressures can be calculated):

\[ D_c \varphi = \mathbf{V}_c \]  \hspace{1cm} (3.2-6)

\[ D_o \varphi = \mathbf{V}_o \]  \hspace{1cm} (3.2-7)

The kernel \( \mathbf{K} \) of \( D_c \) forms a set of permissible element flows \( \varphi \) (the zero vector is excluded), when the prescribed external nodal flows \( \mathbf{V}_c \) are equal to zero (no leaking). The matrix \( \mathbf{R} \) reduces \( \varphi \) to a new base of system flows:

\[ \varphi = R \bar{a} \]

\[ \varphi'' = R \bar{\bar{a}} \]  \hspace{1cm} (3.2-8)

Unlike the matrix \( \mathbf{R} \) in the mechanical domain, \( \mathbf{R} \) in the hydraulic domain is constant in time, since it is assumed that the layout of channels, pipes, and other components does not change in time. Apply this to equation (3.2-5):

\[ M\mathbf{R}\bar{a} + [D_c^T D_c^T]p = \pi \]  \hspace{1cm} (3.2-9)

Multiply with \( R^T \):

\[ R^T M\mathbf{R}\bar{a} + R^T[D_c^T D_c^T]p = R^T \pi \]  \hspace{1cm} (3.2-10)

\( D_c \mathbf{R} = 0 \), because the matrix \( \mathbf{R} \) is the kernel of \( D_c \). It can be demonstrated that \( \mathbf{A} \mathbf{B} = \mathbf{B}^T \mathbf{A}^T \). Therefore: \( R^T D_c^T = 0 \). Apply this to (3.2-10):

\[ R^T M\mathbf{R}\bar{a} + R^T D_c^T p = R^T \pi \]  \hspace{1cm} (3.2-11)

The vector \( p_0 \) contains the prescribed nodal pressures. This equation can be solved by a time integration algorithm.
3.2.1 Example: Pipe and riser elements

To test the theory for hydraulic domain systems, a pipe and riser element were developed and tested in a computer program. Below, the equations for the elements are given.

**Pipe element**

A pipe element is a two node element with two internal volume flows: the flow through the element, and the volume change of the element (see Figure 3.2-1). The nodal volume flows are flows *into* the element.

Assume, that the nodal flow \( V_i \) into a pipe element is the sum of the flow through the element \( V^* \) and the volume change \( \Delta V \) of the element: \( V_i = -V^* + \frac{1}{2} \Delta V \), and \( V_2 = V^* + \frac{1}{2} \Delta V \). The matrices and vectors from the equations above for one pipe element are given below:

\[
D = \begin{pmatrix}
-1 & \frac{1}{2} \\
1 & \frac{1}{2}
\end{pmatrix}
\quad (3.2-12)
\]

\[
\phi = \begin{pmatrix}
\tilde{V} \\
V \\
\Delta V
\end{pmatrix}
\quad (3.2-13)
\]

\( V^* \) is the flow through the element, \( \Delta V \) is the volume change of the element.

\[
V^* = \begin{pmatrix}
V_1^* \\
V_2^*
\end{pmatrix}
\quad (3.2-14)
\]

\( V_1^* \) and \( V_2^* \) are the flows into the element.

\[
M = \begin{pmatrix}
m & 0 \\
0 & \frac{1}{2} m
\end{pmatrix}
\quad (3.2-15)
\]

The variable \( m \) is the mass of the pipe divided by the area of the pipe.
\[ \pi = \begin{pmatrix} \Delta p \\ p \end{pmatrix} \]  

(3.2-16)

The variable \( \Delta p \) is the pressure loss over an element, and \( p^* \) is the average pressure in an element. In the test program, \( \Delta p \) is a quadratic function of the flow and a linear function of the height difference of the nodal points of the element. The variable \( p^* \) is a linear function of the deformation of the fluid.

### Riser element

A riser element is a one-node-element with two internal volume flows: the flow through the element, and the volume change of the element (see Figure 3.2-2). The nodal volume flow is a flow into the element.

Assume, that the nodal flow \( V_1 \) into a riser element is the sum of the flow through the element \( V^* \) and the volume change \( \Delta V \) of the element: \( V_1 = -V^* + \frac{1}{2} \Delta V \). The matrices and vectors from the equations above for one riser element:

\[ D = (-1 \ \frac{1}{2}) \]  

(3.2-17)

\[ \Phi = \begin{pmatrix} \tilde{V} \\ V \\ \Delta V \end{pmatrix} \]  

(3.2-18)

\( V^* \) is the flow through the element, \( \Delta V \) is the volume change of the element.

\[ V^* = (V_1^*) \]  

(3.2-19)

\( V_1^* \) is the flow into the element.

\[ M = \begin{pmatrix} m & 0 \\ 0 & \frac{1}{2} m \end{pmatrix} \]  

(3.2-20)

The variable \( m \) is the mass of the riser divided by the area of the riser.
\[ \pi = \begin{pmatrix} \Delta p \\ p \end{pmatrix} \]  

(3.2-21)

The variable \( \Delta p \) is the pressure loss over an element, and \( p^* \) is the average pressure in an element. In the test program, \( \Delta p \) is a quadratic function of the flow, and a linear function of the fluid height in the element. The variable \( p^* \) is a linear function of the deformation of the fluid.

**Connection matrix**

The equations above are equations for *one* element only, or for a set of elements, that each have their own nodal flows into the element. The nodal flows of the elements have to be connected with the external nodal flows (see Figure 3.2-3). The matrix \( C_n \) (connection matrix of nodes) describes the relation between the flows into the elements \( V^* \), and the (external) nodal flows \( V \):

\[ V = C_n V^* \]  

(3.2-22) for one node:

\[ (V_{\text{external}}) = \begin{pmatrix} 1 & 1 & \ldots & \ldots & \ldots & 1 \end{pmatrix} \]  

(3.2-23)

Substitute (3.2-2) in (3.2-22):

\[ V = C_n \cdot D^* \cdot \Phi \]  

(3.2-24)

From (3.2-2) and (3.2-24) follows:

\[ D = C_n \cdot D^* \]  

(3.2-25)

The hydraulic domain theory was implemented in software and tested. The test model contains two risers with different fluid levels, connected by a pipe forming two communicating reservoirs as shown in Figure

Finite element theory
3.2.4. Table 3.2-1 gives the dimensions of the test model. The results were compared with an analytical solution and the solution of existing numerical software.

<table>
<thead>
<tr>
<th>Description</th>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Height of the water column in the left riser.</td>
<td>l1</td>
<td>700 mm</td>
</tr>
<tr>
<td>Area of the cross section of the left riser.</td>
<td>A1</td>
<td>49087 mm2</td>
</tr>
<tr>
<td>Length of the connecting pipe.</td>
<td>l2</td>
<td>2500 mm</td>
</tr>
<tr>
<td>Area of cross section of the connecting pipe.</td>
<td>A2</td>
<td>1963 mm2</td>
</tr>
<tr>
<td>Height of the water column in the right riser.</td>
<td>l3</td>
<td>500 mm</td>
</tr>
<tr>
<td>Area of the cross section of the right riser.</td>
<td>A3</td>
<td>49087 mm2</td>
</tr>
</tbody>
</table>

Table 3.2-1: Dimensions of the communicating reservoirs.

![Diagram](image)

Figure 3.2-4: A finite element model in the hydraulic domain, formed by two risers with different fluid levels, and one pipe.

### 3.2.2 Analytical solution

The model in Figure 3.2-4 has one degree of freedom $V$, the volume flow through the connecting pipe. When the assumption is made, that the fluctuation of the water level in the riser elements is much smaller than the water level, equation (3.2-26) describes the behaviour of this model:
\[
\left( \frac{e_1}{A_1} + \frac{e_2}{A_2} + \frac{e_3}{A_3} \right) \ddot{V} + g \left( \frac{1}{A_1} + \frac{1}{A_3} \right) V = g \Delta h_0
\]  
(3.2-26)

In equation (3.2-26) \( g \) is the acceleration of gravity, and \( \Delta h_0 \) is the initial level difference in the risers. Equation (3.2-26) describes a recurrent flow with a period \( T \):

\[
T = 2.\pi \sqrt{\frac{\frac{e_1}{A_1} + \frac{e_2}{A_2} + \frac{e_3}{A_3}}{g \left( \frac{1}{A_1} + \frac{1}{A_3} \right)}}
\]  
(3.2-27)

Substitution of the parameters mentioned above results in a period \( T = 11.3 \text{ s} \).

### 3.2.3 Numerical solution

It is possible to find a numerical solution for the problem, when it is not accurate enough to assume that the fluctuation in water level in the riser elements is relatively small. For this purpose the computer program PSI/c is used. Figure 3.2-5 presents the results.

![Figure 3.2-5: The output of computer program PSI/c shows the same period T as the analytical calculation.](image)

---

**PSI/c model:**

\( g = 9.81 \);  
% gravity acceleration [m/s²]  
\( L10 = 0.7 \);  
% height of fluid column in left riser [m]  
\( L20 = 2.5 \);  
% length of connecting pipe [m]  
\( L30 = 0.5 \);  
% height of fluid column in right riser [m]  
\( A1 = \pi /4.0*0.25*0.25 \);  
% area of left riser [m²]  
\( A2 = \pi /4.0*0.05*0.05 \);  
% area of connecting pipe [m²]  
\( A3 = \pi /4.0*0.25*0.25 \);  
% area of right riser [m²]  
\( V = \int (dv \text{ par: 0.0}) \);  
% flow [m³/s]  
\( dv = \int (ddv \text{ par: 0.0}) \);  
\( dh0 = L10 - L30 \);  
\( L1 = L10 - V/A1 \);  
\( L2 = L20 \);  
\( L3 = L30 + V/A3 \);  
\( ddv = g^* (dh0 - V/A1 - V/A3)/(L1 - V/A1)/A1 + L2/A2 + (L3 + V/A3)/A3) \);  
\( \phi i = -dv \);
The numerical solution also shows a period \( T = 11.3 \text{ s} \).

### 3.2.4 Test program, incompressible fluid

In Figure 3.2-6, results of time integration with finite elements equations for incompressible fluid elements are presented. For incompressible fluid, \( \Delta V \) in equations (3.2-13) and (3.2-18) is prescribed to be zero. The corresponding element pressures \( \text{p} \) in equations (3.2-16) and (3.2-21) are also zero. Thus only equations for \( V \) remain.

In figure 3.2-6 the results for incompressible fluid flow are presented. The finite element model also shows a period \( T = 11.3 \text{ s} \).

### 3.2.5 Test program, compressible fluid

The finite elements model, described above, was used to test time integration with compressible fluid. Very small integration steps are necessary, when time integration with compressible fluid is executed, since the eigen frequencies are very high. The results of time integration for a very short period of time are printed in Figure 2.3-7.

---

Figure 3.2-6: Fluid flow through the pipe element of the model in figure 3.2-4, due to the difference in fluid levels in the riser elements at time \( = 0 \). Calculation with incompressible fluid.

Figure 3.2-7: Flow through the pipe element of the model in 3.2-4, due to the difference in fluid levels in the riser elements at time \( = 0 \). Calculation with compressible fluid over 0.1 seconds.
3.3 Multi domain systems

The sections above describe the theory for two domains: the mechanical domain, and the hydraulic domain. Both domains can be treated separately, however, in many cases they interfere. Examples: a hydraulic cylinder, a motor driven pump. This section describes how to integrate the equations of both domains.

It is possible to couple the deformations and stresses of two elements of different domains by adding an extra equation. The first row in equation (3.3-2) describes the relationship between internal element stresses and external nodal displacements in the mechanical domain. The second row describes the relationship between internal element pressures and external prescribed pressures in the hydraulic domain. The third row is a set of extra equations, that describe the coupling between stresses in the mechanical domain and pressures in the hydraulic domain. These extra equations behave like an extra node, a virtual node with a virtual displacement and a virtual force.

\[
\begin{pmatrix}
\varepsilon \\
\varphi
\end{pmatrix} =
\begin{pmatrix}
D & 0 & C_m \\
0 & R & C_h
\end{pmatrix}
\begin{pmatrix}
u \\
a \\
v_d
\end{pmatrix}
\tag{3.3-1}
\]

\[
\begin{pmatrix}
D^T & 0 \\
0 & R^T \\
C_m^T & C_h^T
\end{pmatrix}
\begin{pmatrix}
\sigma \\
\pi
\end{pmatrix} =
\begin{pmatrix}
f \\
-R^T D^T p_o \\
v_f
\end{pmatrix}
\tag{3.3-2}
\]

The vector \(vd\) contains displacements of the virtual nodes, and \(vf\) is a set of forces of the virtual nodes. The connection matrix \(C = [C_m^T C_h^T]\) describes the relationship between elemental stresses in the mechanical and hydraulic domain.

The virtual displacements of the virtual nodes cause changes in the definition of the deformations of the elements. Use equation (3.3-3) to calculate comprehensive deformations \(\varepsilon\) and \(\varphi\):

\[
\begin{pmatrix}
\varepsilon \\
\varphi
\end{pmatrix} =
\begin{pmatrix}
D & 0 & 0 \\
0 & R & 0
\end{pmatrix}
\begin{pmatrix}
u \\
a \\
v_d
\end{pmatrix}
\tag{3.3-3}
\]
Example: pump

Formula (3.3-4) shows how equation (3.3-1) looks for a pump. The pump is modelled as a beam, of which only the torsional deformation is show in (3.3-4), and a pipe, of which only the deformation that represents the flow through the pipe is shown in (3.3-4). The variable $\varepsilon$ is the deformation between the pump shaft and housing, $\varphi$ is the flow through the pump, $\alpha_1$ is the rotation of the housing of the pump, $\alpha_2$ is the rotation of the pump shaft, $a$ is the basis for fluid motion, and $x$ is the transmission ratio between shaft and fluid.

\[
\begin{pmatrix}
\varepsilon \\
\varphi
\end{pmatrix} =
\begin{pmatrix}
-1 & 1 & 0 & 1 \\
0 & 0 & 1 & x \\
0 & 0 & 1 & a
\end{pmatrix}
\begin{pmatrix}
\alpha_1 \\
\alpha_2 \\
x \\
v_d
\end{pmatrix}
\]  

(3.3-4)

3.4 Conclusion

A finite element description of the flow of fluids is necessary to model all important components in drive trains. The classic finite elements theory does not describe the flow of fluids. Therefore, the finite element theory was extended. The equations for solid materials and for fluids can be solved simultaneously.
4 Description of the software package

This chapter describes the reliability analysis software demonstrated in Chapters 2 and 3, in more detail. The software contains a component modeller, and a specially adapted finite element program, that determines the minimal cut sets of a fault tree. The component modeller supports the designer in creating different design solutions, while the finite element program executes the reliability analysis of the design solutions.

This chapter first describes the software architecture, which is the same for both the modeller and the finite element program. Then, it describes specific details for the component modeller, and the finite elements program.

4.1 Architecture

The same software architecture is the base for both the component modeller and the finite element program. In the modeller the architecture separates the implementation of algorithms and components, and in the finite element program it separates algorithms and elements. Since the principle is the same for the modeller and the finite element program, this section explains the architecture for the finite element program only.

 Finite element program

A finite element program contains the implementation of the algorithms and the implementation of the elements. Usually, the programmer stores the definition of the elements in separate source files. The main program then calls the routines in these files. For instance: a rou-
tine to number nodes, a routine to number deformations, a routine to build matrices, and many more.

This program architecture requires the implementation of the same general theory in many different places in the source code. The programmer has to adjust all source files, when he changes an algorithm or develops a new one. He also has to copy and adjust many algorithms, when he creates a new element. Adjusting one of the algorithms is easily forgotten.

This research project resulted in both new algorithms and new elements. The architecture of the developed program tackles the problem described above. The architecture separates the implementation of the theory and the implementation of the elements in such a way, that changing or developing a new algorithm can not introduce errors in the elements. The reverse is also true. Changing or developing a new element can not introduce errors in the algorithm. To achieve this, the elements are defined in recipe files (see Figure 4.1-1). Appendix A and B define the syntax of these recipe files.

The program reads these text files at run time, and stores the element definition in an internal data structure, called the recipe list. A recipe contains formula's, the number of degrees of freedom of the nodes, the number of deformations, and other information, that describes an element.

The program uses the recipe to define and evaluate the elements (or components). The data structures, node list and element (or component) list, contain the model. An element in the element list contains parameters, matrices, and geometry. The user defines the parameters in an input file, or with the user interface. A recipe in the recipe list contains formulas. The program substitutes the parameters of the element into the formulas of the recipe to calculate the element matrices and element geometry. It then substitutes the results of the formulas back into the element matrices and geometry.

Figure 4.1-1: The element definitions are stored in recipe files. The program reads these files at run time.
Figure 4.1-2: The parameters of an element are substituted in the recipe. Then, the formulas are evaluated, and the results are substituted in the element matrices.

Figure 4.1-3: The design system contains a graphically interactive component modeller, and a finite element program. Recipe files define the description of components and finite elements. External files also describe the mapping from components to finite elements. Explanation files describe the failure modes of the components.

**Software package**

Figure 4.1-3 describes the software package containing the component modeller and the finite element program. Both the component modeller and the finite element program use recipe files to define the components and elements. Not only the description of the components and elements, but also the coupling between the modeller and the finite element program is stored in external files. These files define the mapping from components to elements. Explanation files define the failure modes, and the explanation of the failure modes of the components. Appendix B defines the syntax of the mapping and explanation files.

### 4.2 Component modeller

Figure 4.2-1 shows the user interface of the component modeller. The modeller supports the designer in creating different design solutions in
the conceptual design phase. The modeller hides the finite element pro-
gram from the user. It is possible to build a three dimensional structure
from comprehensive components, such as an electric motor, couplings,
a brake, and a gearbox, instead of abstract finite elements. In this way
the designer focuses his attention on creating solutions instead of trans-
lating a structure to a finite element model.

External text files define the components, the mapping of the com-
ponents to finite elements, the failure modes and the explanation of the
failure modes. Appendix A and B define the syntax of these files. This
section gives a brief description. Besides the handling of the compo-
nents, external text files also define the user interface. Appendix C de-
scribes the syntax of these files.

Figure 4.2-1: Screen of the component modeller.
**Recipe**

The recipe is a type definition of a component. The recipe contains:

- number of nodes (connection points);
- nodal positions in local coordinate system;
- parameters (names and default values);
- formulas (a formula can be a function of the parameters and other formulas);
- geometry, positioned in the local coordinate system (the geometry is a function of the parameters and formulas).

**Component**

The component contains:

- name of the component;
- node numbers;
- position in the global coordinate system;
- orientation in the global coordinate system;
- parameter values;
- geometry, positioned in the global coordinate system.

The modeller creates the component with the default parameters of the recipe. It uses the algorithm described in the previous section, and shown in Figure 4.1-2. When the user changes the parameters, position, or orientation of the component, the modeller re-evaluates the recipe with the actual parameters of the component. Figure 4.2-3 shows the most important data structures of the modeller: recipe, and component.

*Figure 4.2-2: Global and local coordinate system.*
Figure 4.2-3: Overview of data structures of modeller.
- **Mapping to elements**

  The modeller automatically translates a component model to a finite element model. Each component is translated to one or more finite elements. The mapping algorithm first prints the list of component nodes to the finite element file. Then, it translates the component list. Mapping files define the mapping from components to elements. Figure 4.2-4 shows that one component can be translated in more elements. For each component a mapping file exists. The contents of this file are written (copied) to the finite element model.

- **Failure modes**

  Each elemental stress represents a failure mode. The failure data, definition, and meaning of the failure modes of a component are stored in explanation files.

4.3 **Finite element program and minimal cut set algorithm**

The finite element program is the core of the reliability analysis software. The finite element model contains an abstract description of the functions of a construction or drive train, and the software finds the minimal cut sets from this model. This section describes how the software works.

This section begins with the description of the data structures for the finite elements program. Then, it describes two algorithms for static analysis, and for physically non linear analysis. This section continuous with the algorithm used to find minimal cut sets directly from a finite element model. Finally, it describes the equations, that are used to execute reliability analysis for the functions carrying load and executing motion.
4.3.1 Data structures

External text files define the elements. Appendix A defines the syntax of these files. This section gives a brief description.

■ Recipe

The recipe is a type definition of an element. The recipe contains:
- the domain of the element (mechanical, hydraulic);
- number of nodes;
- nodal positions in local coordinate system;
- the number of degrees of freedom;
- the number of deformations;
- units of the degree of freedom and deformation parameters;
- parameters names;
- formulas (a formula can be a function of the parameters and other formulas);
- definition of the deformation and stiffness matrix ($D$, $S$);
- definition of the transformation matrix, which transforms coordinates from the global to the local coordinate system ($T$, see Figure 4.2-2);
- geometry, positioned in the local coordinate system (the geometry is a function of the parameters and formulas).

■ Element

The element contains:
- name of the element;
- node numbers;
- deformation numbers;
- orientation in the global coordinate system;
- parameter values.

To create an element, the program uses the algorithm described in section 4.1, and Figure 4.1-2.

A finite element model contains nodes, elements, and boundary conditions for nodes, loads, etc. The program reads the model from a text file and stores it in the data structures: node list, bound list (boundary conditions), load list (nodal forces and moments), stress list (prescribed element stresses), and element list. The next pages show a graphical representation of these data structures.
Figure 4.3-1: Overview of the data structures of the finite element program.
Figure 4.3.2: Overview of the data structures of the finite element program.

4.3.2 Classic algorithm

The classic algorithm calculates nodal displacements, loads, element deformations and stresses of statically loaded structures. The algorithm is useful to test newly developed elements, and is necessary to calculate the element stresses needed to determine the probabilities of failure.
Chapter 5 describes in detail how the probabilities are calculated. This section describes the classic algorithm and equations.

Section 3.1 derives equations (4.3-1) and (4.3-2).

\[ D^T \cdot \sigma = f \]  \hspace{1cm} (4.3-1)

\[ \varepsilon = D \cdot u \]  \hspace{1cm} (4.3-2)

Separate the vector with stresses into a vector with prescribed stresses \( \sigma_0 \), and a vector with calculable stresses \( \sigma_c \). The vector \( \varepsilon_0 \) is the part of the deformation vector, that corresponds to the prescribed stresses \( \sigma_0 \). The vector \( \varepsilon_c \) is the part of the deformation vector, that corresponds to the calculable stresses \( \sigma_c \). Separate the nodal displacements vector into a vector with prescribed displacements \( u_0 \), and a vector with calculable displacements \( u_c \). The vector \( f_0 \) is the part of the load vector, that corresponds to the prescribed displacements \( u_0 \), while \( f_c \) corresponds to the calculable displacements \( u_c \).

\[
[D_{cc}^T \cdot D_{cc}^T] \cdot [\varepsilon_c \cdot \sigma_0] = f_c \quad (4.3-3a)
\]

\[
[D_{cc}^T \cdot D_{cc}^T] \cdot [\varepsilon_c \cdot \sigma_0] = f_0 \quad (4.3-3b)
\]

\[
\varepsilon_c = [D_{cc} \cdot D_{cc}] \cdot [u_c \cdot u_0]^T \quad (4.3-4a)
\]

\[
\varepsilon_0 = [D_{cc} \cdot D_{cc}] \cdot [u_c \cdot u_0]^T \quad (4.3-4b)
\]

\[
\sigma_c = S_c \cdot \varepsilon_c \quad (4.3-5)
\]

Substitution of (4.3-5) and (4.3-4a) into (4.3-3a) results in (4.3-6). Use a Gauss elimination algorithm to solve \( \varepsilon_c \).

\[
D_{cc}^T \cdot S_c^T \cdot D_{cc} \cdot u_c = f_c - D_{cc}^T \cdot \sigma_0 - S_c \cdot D_{cc} \cdot u_0 \quad (4.3-6)
\]

Substitute the solution \( u_c \) and \( u_0 \) (known displacements) into equations (4.3-4a and b) to obtain the deformations \( \varepsilon_c \) and \( \varepsilon_0 \). The calculable stresses \( \sigma_c \) follow from substitution of \( \varepsilon_c \) into (4.3-5). Substitute \( \sigma_c \) and \( \sigma_0 \) (prescribed stresses) into (4.3-3b) to find the reaction forces \( f_c \).

**Multi domain systems**

It is possible to analyse multi domain systems, as described in section 3.3, with the classic finite element algorithm. Due to a special approach, the equations for mechanical and hydraulic domain systems are analogues. Therefore, it is legitimate to replace the matrices of equation (4.3-6) as follows for these systems:

Replace \( D_{cc} \) by \( B_{cc} \).
\[ B_{cc} = \begin{pmatrix} D_{cc} & 0 & C_{mc} \\ 0 & R_c & C_{hc} \end{pmatrix} \]  \hfill (4.3-7)

The matrix \( B_{cc} \) couples the deformations in the mechanical domain and the hydraulic domain. The matrix \( R_c \) corresponds to the calculable stresses of reduction matrix \( R \) described in section 3.2. The matrices \( C_{mc} \) and \( C_{hc} \) are the calculable part of the connection matrix \( C \) (see section 3.3).

Replace \( S_c \) by \( S_{mhc} \):

\[ S_{mhc} = \begin{pmatrix} S_c & 0 \\ 0 & S_{hc} \end{pmatrix} \]  \hfill (4.3-8)

\( S_{hc} \) is the stiffness matrix for the hydraulic domain.

Replace \( u_c \) by \( w_c \):

\[ w_c = \begin{pmatrix} u_c \\ a \\ v_d \end{pmatrix} \]  \hfill (4.3-9)

The vector \( a \) is a base of fluid motion, as described in section 3.2. The vector \( v_d \) contains the displacements of the virtual nodes (see section 3.3).

Replace \( f_c \) by \( g_c \):

\[ g_c = \begin{pmatrix} f_c \\ -R^T \cdot D_c \cdot p_0 \\ v_f \end{pmatrix} \]  \hfill (4.3-10)

The vector \( p_0 \) is a vector of prescribed nodal pressures, while \( D_c \) is the deformation matrix for the hydraulic domain (see section 3.2). The vector \( v_f \) contains the loads on the virtual nodes (see section 3.3).

Substitute (4.3-7 to 10) into (4.3-6), and solve the system of equations to calculate the nodal displacements \( u_c \), the fluid motion \( a \), and the virtual displacements \( v_d \).

Calculate the element deformations \( \varepsilon \) and flows \( \phi \) with equation (4.3-11):

\[
\begin{pmatrix} \varepsilon_c \\ \varepsilon_0 \\ \phi \end{pmatrix} = \begin{pmatrix} D_{cc} & D_{c0} & 0 \\ D_{0c} & D_{00} & 0 \\ 0 & 0 & R \end{pmatrix} \begin{pmatrix} u_c \\ u_0 \\ a \end{pmatrix}
\]  \hfill (4.3-11)
Calculate the element stresses $\sigma_c$ and pressures $\pi_c$ as follows:

$$
\begin{bmatrix}
\sigma_c \\
\pi_c
\end{bmatrix} =
\begin{bmatrix}
S_c & 0 \\
0 & S_{hc}
\end{bmatrix}
\begin{bmatrix}
D_{cc} & D_{c0} & 0 & C_m \\
0 & 0 & R_c & C_h
\end{bmatrix}
\begin{bmatrix}
\frac{u_c}{u_0} \\
\frac{u_0}{a} \\
\frac{v_d}{v_d}
\end{bmatrix}
$$  \hspace{1cm} (4.3-12)

Calculate the reaction forces $f_0$ as described previously in this section. Use equation (4.3-13) to find calculable nodal flows $V_c$.

$$
D_c \phi = V_c 
$$  \hspace{1cm} (4.3-13)

Solve (4.3-14) to calculate nodal pressures $p$.

$$
D^T p = \pi 
$$  \hspace{1cm} (4.3-14)

### 4.3.3 Schwab’s algorithm

Schwab’s algorithm is an alternative to the classic algorithm described in the previous sub section. An advantage of this algorithm is, that it is easier to prescribe deformations and stresses. This is very useful in the algorithms for reliability analysis, since failing elements are represented by prescribed stresses and deformations. A disadvantage of this algorithm is, that it uses more memory, and requires more time to solve the equations. The algorithm is named after its inventor, Schwab.

Schwab’s algorithm combines (4.3-1), (4.3-2), and (4.3-5) into one set of equations:

$$
\begin{bmatrix}
D & 0 & -I & 0 \\
0 & -I & 0 & D^T \\
0 & 0 & S & -I
\end{bmatrix}
\begin{bmatrix}
u \\
\varepsilon \\
\sigma
\end{bmatrix} =
\begin{bmatrix}
f \\
0 \\
0
\end{bmatrix}
$$  \hspace{1cm} (4.3-15)

Equation (4.3-16) prescribes nodal displacements $u_0$, loads $f_0$, and stresses $\sigma_0$. The second and third columns are zero columns, because the contribution of the prescribed displacements and loads is moved to the right hand side of the equation.
\[
\begin{pmatrix}
D_{cc} & 0 & 0 & 0 & -I & 0 & 0 & 0 \\
D_{0c} & 0 & 0 & 0 & 0 & -I & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & D_{cc}^T & D_{0c}^T \\
0 & 0 & 0 & -I & 0 & 0 & D_{ec}^T & D_{00}^T \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & S_e & 0 & -I & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I
\end{pmatrix}
\begin{pmatrix}
\mu_c \\
0 \\
f_0 \\
\varepsilon_c \\
0 \\
\varepsilon_0 \\
\sigma_c \\
0
\end{pmatrix}
= \begin{pmatrix}
-D_{cc0} \cdot \mu_0 \\
-D_{00} \cdot \mu_0 \\
f_c \\
0 \\
0 \\
0 \\
\sigma_0
\end{pmatrix}
\] (4.3-16)

In the classic algorithm the size of the matrices, and the degrees of freedom and deformation numbers depend on the number of prescribed displacements and stresses. It is not possible to prescribe deformations in this algorithm.

However in Schwab’s algorithm the size of the large matrix does not change, when displacements, loads, or stresses are prescribed. Deformation and degree of freedom numbers do not change either. It is also possible to describe deformations in this algorithm. To prescribe a displacement: just add \( D \cdot u_0 \) to the right hand side and place a zero column in the left hand side of (4.3-16). To prescribe a load: also place a zero column in the left hand side and add \( f_0 \) to the right hand side. To prescribe a deformation or stress: make the row representing the equation \( \sigma_i = S_i \varepsilon_i \) equal to zero, and let it represent \( \sigma_i = \sigma_{0i} \) or \( \varepsilon_i = \varepsilon_{0i} \).

Use a Gauss elimination algorithm, that skips zero columns, to solve (4.3-16).

■ Multi domain systems

It is also possible to analyse multi domain systems, as described in section 3.3, using Schwab’s algorithm. Due to a special approach the equations for mechanical and hydraulic domain systems are analogues. Therefore, it is legitimate to replace the matrices of equation (4.3-15) as follows for these systems:

Replace \( D \) by \( B \):

\[
B = \begin{pmatrix}
D & 0 & C_m \\
0 & R & C_h
\end{pmatrix}
\] (4.3-17)

The matrix \( B \) couples the deformations in the mechanical domain with the hydraulic domain. The matrix \( R \) is the reduction matrix described in section 3.2. The matrices \( C_m \) and \( C_h \) are the connection matrix (see section 3.3).
Replace $S$ by $S_{mh}$:

$$S_{mh} = \begin{pmatrix} S & 0 \\ 0 & S_h \end{pmatrix}$$  \hspace{1cm} (4.3-18)

$S_h$ is the stiffness matrix for the hydraulic domain.

Replace $u$ by $w$:

$$w = \begin{pmatrix} u \\ a \\ vd \end{pmatrix}$$  \hspace{1cm} (4.3-19)

The vector $a$ is a base of fluid motion, as described in section 3.2. The vector $vd$ contains the displacements of the virtual nodes (see section 3.3).

Replace $f$ by $g$:

$$g = \begin{pmatrix} f \\ -R^T.D_o.D_p \end{pmatrix}$$  \hspace{1cm} (4.3-20)

The vector $p_0$ contains prescribed nodal pressures, while $D_o$ is the deformation matrix for the hydraulic domain (see section 3.2). The vector $vf$ contains the loads on the virtual nodes (see section 3.3).

Substitute (4.3-17 to 20) into (4.3-15), apply the boundary conditions, and solve the set of equations to calculate the nodal displacements $u_o$, fluid motion $a$, reaction forces $f_e$, and element stresses $\sigma$ and pressures $\pi$.

Calculate the elemental deformations $g$, and flows $p$ with equation (4.3-11). Calculate the nodal flows $V_e$, and the nodal pressures $p$ using equation (4.3-13) and (4.3-14).

4.3.4 Physically non linear analysis

Cables and non return valves are commonly used components in drive systems. The behaviour of these components is non linear. A cable can withstand tension, but no compression. A non return valve behaves like a pipe for flows in one direction, and like a closed valve in the other direction. This phenomenon is called physically non linear behaviour.

This subsection describes how to extend the algorithms described in the paragraphs above, to solve physically non linear problems. This
thesis describes only a restricted implementation of this behaviour limiting the maximum and minimum stress in an element. If a maximum or minimum allowable stress is exceeded, the algorithm prescribes this stress to be zero. Of course this is arbitrary, since this behaviour is only true for brittle material. For ductile material it would be better to prescribe a value equal to the minimum or maximum allowable stress. However, the purpose of this thesis is not to exactly describe material behaviour, but to describe the behaviour of components such as non return valves and cables. For these components it is sufficient to define the maximum or minimum allowable stress in one direction to be zero, and to prescribe a value of zero if the maximum or minimum allowable stress is exceeded.

To analyse physically non linear behaviour the algorithms have to check for invalid stresses after solving the system. An invalid stress is a stress that exceeds its minimum and maximum allowable value. If the algorithm finds invalid stresses, it changes the equations to prescribe valid values for the invalid stresses.

This is easier to implement this in Schwab’s algorithm, since it solves the stresses directly. The classic algorithm first calculates the displacements, then the deformations, and finally the stresses. Changing the equations is also less difficult in Schwab’s algorithm, because the size of the matrix, and the deformations numbers do not depend on the number of prescribed stresses. The non linear analysis was first implemented in Schwab’s algorithm. To increase calculation speed, the algorithm was also implemented in the classic algorithm, which led to an improvement factor of 1½ to 2½.

### Non linear algorithm that uses classic equations

<table>
<thead>
<tr>
<th>while first time or (new invalid stresses found and system was solved)</th>
</tr>
</thead>
<tbody>
<tr>
<td>build equation (4.3-6), and prescribe stresses</td>
</tr>
<tr>
<td>solve ( u_c ) from equation (4.3-6)</td>
</tr>
<tr>
<td>solve ( \varepsilon ) from equation (4.3-2)</td>
</tr>
<tr>
<td>solve ( \sigma_c ) from equation (4.3-5)</td>
</tr>
</tbody>
</table>
Non linear algorithm that uses Schwab's equations

while first time or (new invalid stresses found and system was solved)

build equation (4.3-16), and prescribe stresses

solve $u$, $f_0$, $\varepsilon$, and $\sigma$ from equation (4.3-16)

Pressure controlled valve

Besides the modelling of simple non linear components the non linear algorithm offers the possibility to model more complex components with special functions, such as a pressure controlled valve. (This component has not yet been implemented.) This example demonstrates the modelling possibilities revealed by the non linear algorithm. Figure 4.3-3 shows the modelling of such a valve in finite elements. The model is composed of two pipes and three springs. The lowest pipe represents the valve, while the upper pipe is the control pipe, which opens and closes the valve. The springs connect the two pipes with each other, and represent a switch mechanism.

The average pressure and volume change of the control pipe are coupled with the left upper spring. The pressure difference over the pipe and flow through the pipe, representing the valve, are coupled with the spring on the right. The lower left spring is a very weak spring, that scarcely influences the stresses in the other springs.

The spring connected with the flow through the valve has a maximum allowable stress that is just below the control pressure. Therefore, the spring will break, when the control pressure is applied to the control pipe. The stress in the spring is then zero. This means that there is no pressure difference over the pipe representing the valve. The flow through the valve is free. The lower left spring is necessary to obtain equilibrium with the control pressure.

If the control pressure is zero, the spring connected with the flow through the valve will not break. The connection between the pipe and

---

Description of the software package 69
the spring reduces the difference in pressure with a factor of 100. Thus, the springs only transfer a neglectable fraction of the pressure to the control pipe.

The paragraphs above describe a valve that opens, when the control pressure is applied. The paragraphs below describe a valve that closes, when the control pressure is applied.

Replace the maximum stress in the spring, that is connected to the pipe, representing the valve, by a very large value. Choose the minimum value of this stress just below the control pressure. In this way the spring breaks when there is no control pressure, and remains intact when the control pressure is applied. Thus, the difference in pressure over the pipe is zero, and flow through the pipe is free, when no control pressure is applied. When control pressure is applied, the lower left spring creates an equilibrium with the control pressure, and the pressure difference over the valve.

4.3.5 Reliability analysis, finding minimal cut sets

Chapter 2 shows, that a fault tree analysis is the most useful reliability analysis method in the conceptual design phase. This analysis method contains three steps: 1 construct the fault tree, 2 determine the minimal cut sets, and 3 quantify the minimal cut sets. The previous sections of this chapter describe how to construct a component model, and how to translate this model into a finite element model. This sub section, and the next discuss how to find the minimal cut sets directly from the finite element model, without constructing a fault tree. The fault tree is not necessary, since the minimal cut sets contain all the information. Chapter 5 describes how to quantify the minimal cut sets.

Section 2.4 demonstrates that a failure mode exists for each elemental deformation or stress. A minimal cut set is a set of failing elemental deformations or stresses, that causes the total structure to fail. A minimal cut set contains as few deformations or stresses as possible, so that if one deformation or stress is removed, the system will not fail.

The reliability analysis algorithm finds minimal cut sets by systematically prescribing all combinations of deformations and stresses to be zero. It starts with cut sets with one stress, then with two stresses, then with three, etc. Rao [12] calls this the enumeration method.

The algorithm checks whether failure of the total structure occurs for each cut set. The finite element equations are used to prescribe the deformations or stresses in the cut set to be zero. This means, that an element is broken or blocked. Then, it tries to solve the finite element
equations. A structure fails, when the set of equations can not be solved. Sub sections 4.3.6 and 4.3.7 describe the finite element equations.

When a cut set causes failure, it is saved. From these cut sets, only the \textit{minimal} cut sets are of importance. Therefore, the algorithm checks which cut sets are \textit{minimal} cut sets. When no other cut set is a subset of the cut set, then it is a \textit{minimal} cut set. If A is a cut set needing to be checked, and B is another cut set from the set of all cut sets C, A is a minimal cut set, when the intersection of A and B is not equal to B for all sets B in C, not equal to A:

\[
M = \{ A \mid (A \cap B) \neq B; B \neq A; B \in C \} \tag{4.3-21}
\]

M is a set of minimal cut sets.

For large numbers of deformations the number of possible sets is very large. When a finite element model has \( n \) deformations, the number of possible sets of size 1 is \( n \). The number of possible sets of size two is \( n^2(n-1)/2! \). The number of sets of size \( i \) is \( n!/(i!(n-i)!i!) \). Therefore, the number of sets \( a \) is:

\[
a = \sum_{i=1}^{n} \frac{n!}{(n-i)!i!} \tag{4.3-22}
\]

For large numbers of \( n \) the number of possible sets is very large:

\[
a = O(n^{16}.2^n) \tag{4.3-23}
\]

To analyse all combinations of failing elements would lead to excessive calculation times. However, it is possible to reduce the number of combinations to be analysed. In preliminary design, it will be accurate enough to consider sets of limited size only. This will reduce the number of sets to be tested. This is legitimate for two reasons. First, a structure normally will not have large numbers of redundant components. Therefore, it will not be useful to check for cut sets with very large size. Second, when probabilities of failure are smaller than one, the contribution of a large minimal cut set to the total probability of failure is very small.
Example
- All probabilities of failure have a value of $1 \times 10^{-3}$ and are independent.
- Assume that the smallest minimal cut set consists of two elements. Its contribution to the total probability of failure is $P_1^*P_2 = (1 \times 10^{-3})^2 = 1 \times 10^{-6}$.
- The contribution of a minimal cut set of size three is $P_1^*P_2^*P_3 = (1 \times 10^{-3})^3 = 1 \times 10^{-9}$. This is only 0.1% of the contribution of the smallest minimal cut set!
- The contribution of a minimal cut set of size four is $P_1^*P_2^*P_3^*P_4 = (1 \times 10^{-3})^4 = 1 \times 10^{-12}$. This is only 0.0001% of the contribution of the smallest minimal cut set!

Conclusion: in this case cut sets of large size have very little influence on the total probability of failure.

Non-coherent systems

A system is called a coherent system if failure of a certain set of components leads to failure of the total structure, and failure of an extra component also leads to failure of the total structure for all components. For some structures, failure of a certain set of components does lead to failure of the total structure, while failure of one extra component does not lead to failure of the total structure. Such a system is called a non-coherent system. Ansell [64] describes this behaviour mathematically. The enumeration method, in principle, produces valid results for coherent and non-coherent systems, since it simply tries all combinations. However, to increase calculation speed, the implemented version of the enumeration method does not check sets of failing components containing a subset, that is a minimal cut set. In these cases the algorithm assumes that these sets also lead to failure. This is legitimate, because the structures in the application field, such as storm surge barriers, movable bridges, and navigation locks, usually do not behave in the way described above. Furthermore, if they would behave in this way, a conservative estimation of the failure behaviour of a structure is given.

4.3.6 Reliability analysis, carrying load

Chapter 2 defines reliability as the probability that a system fulfils its functions. It is stated, that it is possible to decompose any function of a drive system into the functions carrying load and executing motion. The previous sub section describes how the reliability analysis algorithm finds the minimal cut sets. This sub section describes the equations the algorithm uses in the case of an analysis of the function carrying load. The algorithm is called collapse analysis.

The collapse analysis has been implemented in three algorithms: an algorithm that uses classic finite element equations, one that uses
Schwab’s equations, and one to analyse structures with physically non linear elements.

■ Classic

The first algorithm uses the equations of sub section 4.3.2.

\[ D_{oc}^T \sigma_c = f_c - D_{oc}^T \sigma_0 \]  \hspace{1cm} (4.3-24)

All possible combinations of elemental stresses are prescribed to be zero. If it is not possible to find a permissible stress distribution \( \sigma_c \) that satisfies equation (4.3-24), a failure mode \( A \) is found. If \( A \) satisfies (4.3-21), the failure mode is a \textit{minimal} cut set.

■ Schwab’s algorithm

Collapse analysis has also been implemented in an alternative algorithm. This algorithm uses equation (4.3-16) instead of (4.3-24).

■ Non linear analysis

The last collapse analysis algorithm uses a non linear algorithm, as described in sub section 4.3.3. It is possible to analyse systems with physically non linear (but geometrically linear) elements such as non return valves, and cables.

4.3.7 Reliability analysis, executing motion

The previous sub section describes the equations that are used to find the minimal cut sets for the function \textit{carrying load}. This sub section describes the equations that the algorithm uses in case of an analysis of the function \textit{executing motion}.

The objective of motion analysis is to calculate the probability that a structure can not execute the prescribed motion. A nodal displacement \( u_0 \) represents the motion.

In motion analysis, failure of an elemental deformation is described by prescribing a deformation to be zero. Furthermore, all deformations with calculable stresses, \( \varepsilon_c \), are considered to be rigid and are also prescribed to be zero.

Similar to the previous sub section, describing collapse analysis, the theory for motion analysis has been implemented in three algorithms: an algorithm that uses classic finite element equations, one that uses Schwab’s equations, and one to analyse structures with physically non linear elements.
■ **Classic**

The classic algorithm for motion analysis uses equation (4.3-25):

\[
\begin{pmatrix}
D_{cc} \\
D_{0c, prescribed}
\end{pmatrix} \cdot (u_c) = \begin{pmatrix}
Q - D_{c0} \cdot u_0 \\
Q - D_{00, prescribed} \cdot u_0
\end{pmatrix}
\]  

(4.3-25)

\(D_{0c, prescribed}\) and \(D_{00, prescribed}\) are sub matrices of \(D_0\) corresponding with failing deformations, that are prescribed to be zero.

■ **Schwab's algorithm**

Motion analysis has also been implemented in an alternative algorithm. This algorithm uses equation (4.3-16) instead of (4.3-25).

■ **Non linear analysis**

The last algorithm uses a non linear algorithm as described in sub section 4.3.3. It is possible to analyse systems with physically non linear elements such as non return valves, and cables.

The non linear deformations are not prescribed to be zero. An extra parameter \(\text{rigid}[ij]\) is added to the non linear element deformations.

Syntax: \(\text{par rigid}[<\text{integer}\ i>]\ no\).

The integer \(i\) is the non linear deformation number.

### 4.3.8 Reliability analysis: one-out-of-two, and two-out-of-two systems

Some systems contain double, triple or even more parallel components, for instance, double electric motors. Sometimes all parallel components are necessary to let the system function, but sometimes only one or a subset of the parallel components is necessary for the system to function. A system of double components, of which only one is necessary to allow the system function, is called a one-out-of-two system. A system of double components, which are both necessary to let the system function, is called a two-out-of-two system. The same goes for one-out-of-three, two-out-of-three, and three-out-of-three systems.

The paragraphs above describe algorithms that are partly able to analyse both types of systems. The algorithms separate the reliability analysis into reliability of \(\text{carrying load}\), and \(\text{executing motion}\). They describe both one-out-of-more and more-out-of-more systems for the function \(\text{carrying load}\). However, for the function \(\text{executing motion}\) it
only describes one-out-of-more systems. This means, that two types of systems can be modelled for this function. The first type are systems where it is acceptable for them to perform at a reduced level in case of emergency. For example: a navigation lock gate that closes at half speed. In most civil systems, this is acceptable. The second type are systems containing fully redundant components. For example: two redundant pumps of which each one can produce enough flow to allow the lock gate to work at full speed. In normal conditions, the pumps have to produce half the flow, or one of the pumps waits on stand by. If one of the pumps fails, the other one has to be switched to full speed or has to be started. This causes functional dependency as described in chapter 5.

The linear algorithms for *carrying load*, described in the paragraphs above, are able to describe one-out-of-more systems. They do not check if allowable stresses are exceeded. Therefore, these algorithms assume, that only one component in a set of redundant components is sufficient to let the structure function. The non linear algorithms are able to describe more-out-of-more systems by adjusting the maximum allowable stress in a set of more components in such a way that it is exceeded if too many components fail. Example: The load on two electric motors is 50 Nm. The maximum torque on each motor is 35 Nm. In normal working condition both motors take 25 Nm, but if one motor fails, the other motor has to take 50 Nm, which is too much. The non linear algorithm will let the other motor collapse. Thus, a two-out-of-two system is modelled.

The linear and non linear algorithms for *executing motion*, described in the paragraphs above, are only able to describe one-out-of-more systems, since it is not possible to prescribe maximal and minimal allowable deformations. These algorithms assume, that only one component in a set of redundant components is sufficient to let the structure function.

### 4.4 Conclusion

This chapter describes the implementation of the theory from chapters 2 and 3 into software. The software contains a component modeller, which helps the designer in creating design solutions, and a finite element program, which analyses the variants. An open software architecture makes it possible to alter the implementation of components and elements. Various algorithms were implemented. Some algorithms
should produce equal output. It is possible to validate these algorithms by comparing their results (see chapter 6).

Only geometrically linear algorithms were implemented. However, the motion of some structures, such as a moveable bridge, is geometrically non linear. It is necessary to analyse such a structure in one or more decisive positions.
5 Calculation of the probability of failure

The previous chapters demonstrate how to find the failure modes and the minimal cut sets of a structure. These minimal cut sets describe how a structure can fail. This chapter discusses how to calculate the probability of occurrence of these failure modes.

This chapter begins by describing the theory used to calculate the probability of failure for a single component and for a set of components. Then, it discusses how to cope with uncertainty in failure data, and models for dependent failure.

5.1 Failure per demand versus failure rate

Consider a pumping station, that controls the water level of the channels in a certain area. In a period of dry weather the engine is not operating. When it starts raining, the water in the channels rises, and the engine starts pumping to reduce the water level. Fail to start is a possible failure mode of the pumping engine. During a rainy period the engine must pump continuously. Stop pumping is another possible failure mode.

This section first describes two models for the probability for failure of a single component. The first model describes failure per demand. It assumes, that failure can occur when an action is demanded, and that the probability of failure is constant per demand. This model describes failure of discontinuous processes. Failure of the pumping engine to start is best described by this first model, failure per demand, since starting is a discontinuous process.

The second model assumes that the probability of failure is a function of time. The failure rate is a measure of the speed at which the probability of failure grows in time. This model describes failure of continuous processes. Stop pumping is best described by the second
model, failure with a failure rate, because pumping is a continuous process.

This section then discusses the availability, unavailability, and application of the models and theory. Finally, the calculation of the probability of failure and the unavailability of a set of components is described.

Van Gestel [22], Schröfer [31], Zacks [55], Lewis [56], Ushakov [57], O’Connor [44], Carter [45], Gits [59], and Ansell [64] also describe the theory in this section. This thesis applies the theory to drive trains of civil structures, and partly extends it, since it is not sufficient to recognise continuous and discontinuous processes alone (Kiestra [6] and Van Geijlswijk [9]).

5.1.1 Failure per demand

This sub section describes a model for the probability of failure \( P \) for a single component. It is assumed, that failure can occur when an action is demanded, and that the probability of failure is constant per demand. However, it is reasonable to argue that the probability of failure is not constant, but increases in time, since a component degenerates by using it. On the other hand, if a component is maintained and replaced adequately, it is reasonable to assume that ageing does not influence the probability of failure, and that the probability of failure is constant in time. Hence, the model is only valid for components which are maintained, and does not describe the ageing of components.

\[
P = Q
\] (5.1-1)

\( Q \) is the probability of failure per demand (query).

It is possible to estimate the probability of failure per demand of a component as follows: Consider a very large population of identical components of size \( n \). A component can fail each time it is demanded to work. If all components of the population of size \( n \) are demanded to work, a number \( i \) will fail. Formula (5.1-2) is an estimator for the probability of failure per demand:

\[
\hat{P} = Q = i/n
\] (5.1-2)

When the probability of failure is relatively low, this method to estimate the probability of failure requires a very large population of components. Another option to estimate the probability of failure is to consider a smaller population of components of size \( n \), and make an inventory of the number of failing components, \( i \), over multiple de-
mands. When a component has failed, it is replaced by a new one. Thus, the size of the population remains equal to n. This results in an estimator \( \hat{P} = Q = i/(n^*q) \) in which q is the number of queries.

### 5.1.2 Failure rate

This sub section describes another model for the probability of failure P of a single component. In this model the probability of failure is a function of time.

\[ P = F(t) \quad (5.1-3) \]

\( F(t) \) is the probability that a component fails in a time interval from 0 to t. \( F(t) \) is called the probability distribution function.

It is possible to estimate the probability of failure of a component as follows: Consider a very large population of identical components of size n. The number of components of the population that have failed in a time interval from 0 to t is represented by \( i(t) \). Formula (5.1-4) is an estimator for the probability of failure:

\[ \hat{P} = \hat{F}(t) = i(t)/n \quad (5.1-4) \]

\( f(t) \) is the probability density function:

\[ f(t) = \frac{dF(t)}{dt} \quad (5.1-5) \]

\( \lambda(t) \) is the failure rate:

\[ \lambda(t) = \frac{f(t)}{1 - F(t)} \quad (5.1-6) \]

The density function \( f(t) \) refers to the original population of components, while the failure rate function \( \lambda(t) \) refers to the remaining population of components.

---

Figure 5.1-1: The probability distribution function \( F(t) \) is a function of time.

Figure 5.1-2: The probability density function \( f(t) \).

Figure 5.1-3: The failure rate function. This form of \( \lambda(t) \) is called a bathtub curve.
The variable \( \tau \) is the failure time of a component. The variable \( P \) is the probability that a component fails in a time interval from \( a \) to \( b \).

\[
P = P(a \leq \tau \leq b) = \int_a^b f(\tau) \, d\tau \tag{5.1-7}
\]

For the rest of this section the probability of failure \( P \) of a component is defined as the probability, that it has failed in a time interval from \( 0 \) to \( \tau \). Therefore, \( a = 0 \) and \( b = \tau \):

\[
P = P(0 \leq \tau \leq t) = \int_0^t f(\tau) \, d\tau = F(t) \tag{5.1-8}
\]

### Bath-tub-curve

Figure (5.1-3) shows a failure rate function. The special form of this function is called the bath-tub-curve. The left side of the curve represents failure due to production and installation faults. The right side represents failure due to ageing. The middle of the curve represents coincidental failure. The theory below describes the middle of the bath-tub-curve: Interval \( T \) in figure (5.1-1) to (5.1-3).

Assume that the failure time of a component is exponentially distributed:

\[
F(t) = 1 - e^{-\lambda t} \tag{5.1-9}
\]

\[
f(t) = \lambda e^{-\lambda t} \tag{5.1-10}
\]

\[
\lambda(t) = \lambda \tag{5.1-11}
\]

Figure 5.1-4 and 5.1-5 show the distribution function and density function of these formula’s. An exponential distribution has a constant failure rate \( \lambda \). If \( \lambda \cdot t < < 1 \):

\[
F(t) \approx \lambda \cdot t \tag{5.1-12}
\]

\[
f(t) \approx \lambda \tag{5.1-13}
\]
5.1.3 Unavailability

The probability of failure is a measure of the chance, that a system fails. After a system has failed, it is unavailable. Two reasons for unavailability exist: first, because failure remains undetected for a while, and second, because the system has to be repaired. The probability of failure is not a good measure to judge the unavailability of a system. Besides the probability of failure, it is interesting to consider the fraction of time that a system is in working condition, the availability $A$, and the fraction of time that it is not in working condition, the unavailability $U$. The relationship between availability and unavailability is:

$$A = 1 - U$$ \hfill (5.1-14)

Failure and repair are not the only reasons for unavailability. Another cause of unavailability is maintenance.

- Maintenance

Unavailability from maintenance is caused by the maintenance time $\tau$. Assume that the probability of failure is reduced to zero after maintenance. The unavailability by maintenance of a single component is:

$$U_m = \tau / (T_m + \tau) \approx \tau / T_m$$ \hfill (5.1-15)

$T_m$ is the maintenance interval.

Application of redundant components can reduce the system unavailability due to maintenance to zero. While one component is in maintenance, the other can still fulfil its task.

In some cases it is possible to execute maintenance in periods, when a system is out of operation. In these cases the unavailability due to maintenance is also zero.

To calculate the unavailability of a system, it is necessary to distinguish between revealed versus unrevealed failure, and failure per demand versus failure with a failure rate. The paragraphs below give a general formulation for the unavailability of a system of components, and then evaluate this formulation for one single component.
## Revealed failure per demand

The time $\theta$, necessary to repair a system, and the time that the failure remains undetected cause unavailability by failure. Failure detected immediately, is called *revealed* failure. The unavailability of a system due to revealed failure per demand depends on the repair time $\theta$, the number of queries per unit time $a$, and the probability of failure $P$, as follows:

$$U_f = \theta.a.P \quad (5.1-16)$$

The unavailability due to revealed failure per demand of one single component is similar to the unavailability of a system:

$$U_f = \theta.a.Q \quad (5.1-17)$$

$Q$ is the probability of failure per demand of one single component.

## Revealed failure with a failure rate

The probability of failure of a system in a time interval $(t, t + dt)$ is equal to $f(t).dt$. If failure occurs, the fraction of time that repair takes in a maintenance interval $T_m$ is equal to $\theta/T_m$. Failure is revealed and the system is repaired immediately. Thus, the expectation value for the fraction of time that a system is unavailable due to failure in the time interval $(t, t + dt)$, is equal to $f(t).dt.\theta/T_m$. The total unavailability in the maintenance interval $T_m$ is the sum of the unavailabilities of all time intervals $(0 < t \leq T_m)$. The unavailability of a system due to revealed failure with a failure rate is:

$$U_f = 1/T_{mnt} = \int_0^{T_m} \theta.f(t).dt \quad (5.1-18)$$

The unavailability due to revealed failure of one single component with a failure rate $\lambda$ is:

$$U_f = \lambda.\theta \quad (5.1-19)$$

## Unrevealed failure per demand

Failure not detected immediately, is called *unrevealed* failure. The time interval, in which a system is unavailable due to unrevealed failure, can be reduced by testing. Thus, a system will never be unavailable longer than the test interval $T_t$ plus the repair time.

Assume that the duration of the test does not contribute to the unavailability. If a system is active, a test checks to see if the system still fulfils its task. Assume that it is not necessary to stop the process for
the check. A system in rest is not required to be available. Thus, testing in a rest period does not influence the unavailability either.

The unavailability of a system due to unrevealed failure is the sum of the unavailability due to the time $T_i$, that failure remains undetected, and the unavailability due to repair. For a system, that fails per demand, this results in:

$$U_t = T_i.aP + \theta.aP = (T_i + \theta).aP$$  \hspace{1cm} (5.1-20)

The unavailability due to unrevealed failure per demand of one single component is similar to that of a system:

$$U_t = T_i.aQ + \theta.aQ = (T_i + \theta).aQ$$  \hspace{1cm} (5.1-21)

### Unrevealed failure with a failure rate

The calculation for the unavailability due to unrevealed failure of a system with a failure rate is more complex. The influence of testing on the probability of failure is discussed below. Then, the length of the time interval in which the system is unavailable is determined.

If a test shows, that a system has failed, it will be repaired. Assume that after repair the system is as good as new. Thus, the probability of failure is equal to zero after repair. On the other hand, if a test shows that a system is in working condition, the probability that it has failed before the test is apparently zero. Consequently, the probability of failure of a system is always zero after a test.

The probability of failure in a time interval $(t, t + dt)$ is equal to $f(t)dt$. If failure occurs at time equal to $t$, $(t_1 < t \leq t_1 + T_i)$, the interval that the system is unavailable is equal to the time that failure remains undiscovered plus the repair time $(t_1 + T_i - t + \theta)$. The fraction of time that failure remains undiscovered plus the repair time in a test interval $T_i$ is equal to $(t_1 + T_i - t + \theta)/T_i$. Thus, the expectation value for the fraction of time, that a system is unavailable due to failure in the time interval $(t, t + dt)$, is equal to $f(t)dt.(t_1 + T_i - t + \theta)/T_i$. The total unavailability in the test interval $T_i$ is the sum of the unavailabilities of all time intervals.
intervals \( t_i < t \leq t_i + T_i \). The unavailability due to unrevealed failure of a system in a test interval \( T_i \), starting at time equal to \( t_i \), is:

\[
U_f = \frac{1}{T_i - t_i} \int_{t_i}^{t_i + T_i} (t_i + T_i - t + \theta) f(t) \, dt \tag{5.1-22}
\]

Evaluation of (5.1-22) results in the unavailability due to unrevealed failure of one single component:

\[
U_f = \frac{1}{2} \lambda T_i + \lambda \theta \tag{5.1-23}
\]

### 5.1.4 Application of the theory

The paragraphs above describe the theory of failure per demand, failure rate, and unavailability of systems and components with revealed and unrevealed failure. This sub section discusses the application of this theory.

It is possible to recognise intervals of rest and action in the utilisation of mechanical systems. Between these intervals transitions are defined. The transition from rest to action is called starting. The transition from action to rest is called stopping.

- **Rest**

Failure in a rest period is modelled as a continuous process. Therefore, it is best described by the failure rate theory. Failure in a rest period is unrevealed, because the mechanical system does not stop functioning. Failure during this period is detected, when the system fails to start.

The probability of failure is a useful parameter to describe failure during the rest period. It is possible to reduce the probability of failure by testing if a system can be repaired before action is demanded. Therefore, testing is only useful if the test interval is smaller than the rest period.

Since the system is not required to be available in a rest period, availability is not a useful parameter to describe failure. However, failure at the end of the rest interval can influence the availability in the action period, if repair has not been finished before action is demanded.
Maintenance executed during the rest period does not contribute to the unavailability of the system, since the system is not required to be available in this period. However, it does reduce the probability of failure.

An example of a failure mechanism, that takes place during a rest interval, is corrosion of bearings.

Start

Starting is not a continuous process. Therefore, it is best described by a probability of failure per demand. Besides the probability of failure, the unavailability is a useful parameter to describe failure during the start of a system, since failure during start causes unavailability in the action period.

It is assumed, that the probability of failure per demand is constant in time. Maintenance does not influence the probability of failure and unavailability during start, because of this assumption. When failure is unrevealed, testing does reduce the unavailability. However, testing does not reduce the probability of failure. When failure is revealed, testing does not have any influence, because it is assumed that a system will be repaired immediately after detection of failure.

An example of a failure mechanism during start is a short circuit due to degeneration or melting of the insulation material in an electric motor.

Action

Failure in an action period is modelled as a continuous process, and is best described with the failure rate theory. Failure in an action period can be both revealed and unrevealed. An example of a system with revealed failure during an action period is a motorcar. The driver notices immediately, when the car stops while it is driving. An example of a system with unrevealed failure is an unmanned pumping station. When the pumps have stopped, it will take some time before failure is detected.

Since unavailability is a measure for the fraction of time that a system is in working condition, it is the most useful parameter to describe failure during action. The probability of failure is less useful, because it is not constant in time. However, it is possible to calculate the probability, that a system fails in an action period.

It is not possible to (positively) influence the unavailability of a revealed failing system by testing or maintenance during the action interval. Even worse, maintenance during the action period enlarges the

Calculation of the probability of failure
unavailability, because it is assumed that the system has to be stopped. Since failure is detected and the system is repaired immediately, testing does not influence the unavailability of a system with revealed failure. On the other hand, it is possible to reduce the unavailability of an unrevealed failing system by testing and maintenance. Testing reduces the period that failure remains undetected, and the system is assumed to be as good as new after maintenance.

An example of a failure mechanism during action is the fatigue of bearings.

**Stop**

Stopping, like starting, is not a continuous process. Therefore, it is best described by a probability of failure per demand. Failure during stopping must not be mistaken for failure to stop. The first includes all failure mechanisms that are caused by stopping, but may cause failure during another interval. The latter means failure to end the action interval. However, failure to stop is presumably best described by starting, since stopping usually requires a braking device to become active.

Since the rest interval follows the stop interval, and failure in the rest interval is unrevealed, failure during stop is also unrevealed. The probability of failure is a useful parameter to describe failure during stopping. It is assumed, that the probability of failure per demand is constant in time. Maintenance does not influence the probability of failure during stopping, because of this assumption.

Since the system is not required to be available in the rest period following the stop, availability is not a useful parameter to describe failure during stopping. However, failure caused by stopping influences the availability in the action period, because failure is detected and the system then has to be repaired.

An example of a failure mechanism during stopping is the sticking of a electrical relay.
### Summary

<table>
<thead>
<tr>
<th>parameters revealed failure</th>
<th>positively influenced by</th>
<th>parameters unrevealed failure</th>
<th>positively influenced by</th>
</tr>
</thead>
<tbody>
<tr>
<td>rest</td>
<td>-</td>
<td>-</td>
<td>F(T₀)</td>
</tr>
<tr>
<td>start</td>
<td>P, U</td>
<td>-</td>
<td>P, U</td>
</tr>
<tr>
<td>action</td>
<td>U</td>
<td>-</td>
<td>U</td>
</tr>
<tr>
<td>stop</td>
<td>P</td>
<td>-</td>
<td>P</td>
</tr>
</tbody>
</table>

*Table 5.1-1: F(T₀) is the probability of failure at the end of a test interval, P is the probability of failure, U is the unavailability.*

### Different system types

In some systems the action period is dominant. An example of a system with a dominant action period is an electric power plant. In other systems the rest period is dominant. An example of a system with a dominant rest period is a storm surge barrier.

Most failure data is obtained from systems with a dominant action period. Data is available for the probability of failure to start, Q_{start} and the failure rate in action intervals, \( \lambda_{action} \).

The techniques offered by Van Gestel [22] are also based on systems with a dominant action interval. For these systems the unavailability is more important than the probability of failure.

Mechanical systems in infrastructural works, such as storm surge barriers, moveable bridges, and lock gates, have a dominant rest period. This means that \( \lambda_{action} \) is less significant than \( Q_{start} \) and \( \lambda_{rest} \).

*Figure 5.1-9: In some systems the action period is dominant.*

*Figure 5.1-10: In other systems the rest period is dominant.*
Unfortunately, not enough data for the rest period is available. Since the mechanisms for failure in the rest and action period are different, $\lambda_{\text{action}}$ cannot be applied in the rest period. Example: The major failure mechanism of a roller bearing is corrosion in the rest period and fatigue in the action period.

### 5.1.5 A set of components

The paragraphs above generally describe how to calculate the probability of failure and the unavailability of a system, and apply the theory to calculate these values for a single component. This sub section describes in more detail how to calculate the probability of failure and unavailability for a set of components and a total structure.

A sub-result of a fault tree analysis is a set of minimal cut sets. A minimal cut set is a set of failure modes for one or more components, that is as small as possible, and causes failure of the total structure. The structure will not fail if one of the failure modes is removed from the minimal cut set. If one failure mode is added to a minimal cut set, it is no longer a minimal cut set.

If the probabilities of failure for the minimal cut sets are small, the probability of failure for a structure is the sum of the probabilities of failure of the minimal cut sets:

$$P_{\text{tot}} = \sum P_{\text{cut set}}$$  \hspace{1cm} (5.1-24)

When failure occurs per demand, and the probabilities of failure of the components in the minimal cut set are independent, the probability of failure for a minimal cut set is the product of the probabilities of failure for the components:

$$P_{\text{cut set}} = \prod P_{\text{comp}} = \prod Q_{\text{comp}}$$  \hspace{1cm} (5.1-25)

When the components have a failure rate, the probability of failure is a function of time. When the probabilities of failure of the components in the minimal cut set are independent, the probability of a minimal cut set is the product of the probability distribution functions of the components:

$$P(0 \leq \tau \leq t)_{\text{cut set}} = \prod F(t)_{\text{comp}}$$  \hspace{1cm} (5.1-26)

If the unavailabilities due to failure of the minimal cut sets are small, the unavailability of a structure is the sum of the unavailabilities of the minimal cut sets:
\[ U_{\text{tot},f} = \sum U_{\text{cut set, } f} \]  

(5.1-27)

When failure occurs per demand, the unavailability of a minimal cut set is given by formulas (5.1-28) and (5.1-29):

- **Revealed failure**

\[ U_{\text{cut set, } f} = \theta.a.P_{\text{cut set}} \]  

(5.1-28)

- **Unrevealed failure**

\[ U_{\text{cut set, } f} = (T_t + \theta).a.P_{\text{cut set}} \]  

(5.1-29)

The longest repair time for the components in the minimal cut set is \( \theta \), while \( a \) is the number of queries per time unit and \( T_t \) is the test interval for unrevealed failure.

When the components have a failure rate, the unavailability due to failure of a minimal cut set is a function of the probability density function \( f(t)_{\text{cut set}} \) of the minimal cut set. This function can be obtained by differentiation of the probability distribution function for the minimal cut set. This distribution function is a function of time and is the product of the distribution functions of the components in the minimal cut set. Formula (5.1-31) and (5.1-32) give the unavailability of a minimal cut set.

\[ f(t)_{\text{cut set}} = d(\Pi F(t)_{\text{comp}})/dt \]  

(5.1-30)

- **Revealed failure**

\[ U_{\text{cut set, } f} = 1/T_{m:v} - \int_{T_m}^{T_t} \theta.f(t).dt = \theta/T_m.\Pi F(T_m)_{\text{comp}} \]  

(5.1-31)

Formula (5.1-30) is true if \( F(0)_{\text{comp}} \) is equal to zero. This is true in most cases. If \( F(0)_{\text{comp}} \neq 0 \), then \( U_{\text{cut set, } f} = \theta/T_m.\Pi (F(T_m)_{\text{comp}} - F(0)_{\text{comp}}) \).

- **Unrevealed failure**

\[ U_{\text{cut set, } f} = 1/T_{m:v} - \int_{T_t}^{T_m} (T_t - t + \theta).f(t)_{\text{cut set}}.dt \]  

(5.1-32)

The maintenance interval is \( T_m \), while \( T_t \) is the test interval.

If the probability distribution function of a component is equal to \( F(t)_{\text{comp}} = \lambda_{ti}t \), and a minimal cut set exists of \( n \) failure modes of components, evaluation of formula (5.1-32) results in:

---

Calculation of the probability of failure
\[ U_{\text{cut.set},f} = \prod_{i=1}^{n} \lambda_i \cdot \left( \frac{1}{n+1} \right) T^n + \frac{T^{n-1}}{n} \]  

(5.1-33)

If a minimal cut set contains \( n \) components, that fail per demand, with a probability of failure \( Q_i \), and \( m \) components with probability distribution function equal to \( F(t)_{\text{comp}} = \lambda_j t \), evaluation of formula (5.1-32) results in:

\[ U_{\text{cut.set},f} = \prod_{i=1}^{n} Q_i \cdot \prod_{j=1}^{m} \lambda_j \cdot \left( \frac{1}{m+1} \right) T^m + \frac{T^{m-1}}{m} \]  

(5.1-34)

The next part of this section combines the formulas above with the theory of sub section 5.1.4, showing that failure can occur in four phases of employment of a system: during rest, start, action, or stop. Assume that failure during rest and action have a failure rate, and that failure during start and stop is per demand. Furthermore, assume that failure during rest and stop is unrevealed, and that failure during start and action is revealed. (See Table 5.1-2.)

<table>
<thead>
<tr>
<th>interval</th>
<th>failure rate or failure per demand</th>
<th>revealed or unrevealed failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>rest</td>
<td>failure rate</td>
<td>unrevealed</td>
</tr>
<tr>
<td>start</td>
<td>failure per demand</td>
<td>revealed</td>
</tr>
<tr>
<td>action</td>
<td>failure rate</td>
<td>revealed</td>
</tr>
<tr>
<td>stop</td>
<td>failure per demand</td>
<td>unrevealed</td>
</tr>
</tbody>
</table>

> Table 5.1-2: Application of the failure models.

Thus, each component can fail in four phases. Chapter 2 defines failure in terms of not able to carry load and not able to execute motion, regardless of the phase. Therefore, for a minimal cut set with one single component it is necessary to add the probabilities of failure and the unavailabilities of the phases to find the total probability of failure and unavailability. For a minimal cut set with multiple components it is more complex to find the total probability of failure and unavailability. It is necessary to add the probabilities and unavailabilities of the components in all possible combinations of phases.

Example: In a minimal cut set of two components the first component can fail during the rest phase, and the second during the rest, start, action, or stop phase. Then, the first component can fail during the start phase, and the second again during the rest, start, action, or stop phase. etc.
If all components in a minimal cut set fail, unrevealed formulas (5.1-33) and (5.1-34) are applied. In the other cases formula (5.1-31) is applied.

5.1.6 Physical model for data

This sub section describes a physical model to calculate the probability of failure and the failure rate for a component. This model describes failure due to external loads. However, for other causes of failure no physical model is available. In these cases, the failure data has to be obtained from databases.

The theory in the paragraphs above describes failure per demand and by failure rate. This sub section discusses failure due to static and dynamic load. Failure due to a static load is described by failure per demand, while failure due to dynamic load is described with a failure rate.

As mentioned in chapter 1, this thesis concentrates on modelling the stochastic behaviour of the system. Minor attention is given to the stochastic character of the loads. Therefore, only simple load models are given in this sub section.

Failure due to static load

When the probability density functions of the load $L$ on a component, and the available element strength $S$ are known, it is possible to calculate the probability of failure for a component.

The load on the elements can be calculated by solving displacements $u_c$ from the classical system of finite elements equations (see chapter 3):

$$D^{T}_{cc}S_cD_{cc}u_c = f_c \quad (5.1-35)$$

$$\sigma_c = S_cD_{cc}u_c \quad (5.1-36)$$

The element stiffness matrix is represented by $S_c$, while $D_{cc}$ is the deformation matrix, $f_c$ is a vector with (external) nodal loads, and $\sigma_c$ a vector with internal element loads (stresses).

Spoormaker [14], and Brinkman [15] describe how to

Figure 5.1-11: A component fails when the load is larger than the strength.
calculate the probability of failure for a single element:

It is assumed that the load on the elements can be described by a stochastic variable $L$, with a Gauss-Normal probability density. The average values of the load $\mu_L$, can be obtained by normalising the elemental stresses from the vector $\sigma_e$: $|\sigma_{ei}| = \mu_L$. The standard deviation of $L$ is $\sigma_L$.

The strength of the elements, $S$, can also be described by a stochastic variable with a Gauss-Normal probability density with parameters $\mu_S$ and $\sigma_S$.

A component fails when the external load exceeds the available strength:

$$P_{\text{failure}} = P(L > S) = P(L - S > 0) \quad (5.1-37)$$

A new stochastic variable $Z$ is introduced, defined by:

$$Z = L - S \quad (5.1-38)$$

Failure of a component occurs when $Z$ is larger than zero:

$$P_{\text{failure}} = P(Z > 0) \quad (5.1-39)$$

$Z$ is a stochastic variable with a Gauss-Normal probability density, since $L$ and $S$ are Gauss-Normal stochastic variables. The parameters of $Z$ can be calculated, applying the next equations:

$$\mu_Z = \mu_L - \mu_S \quad (5.1-40)$$

$$\sigma_Z = \sqrt{(\sigma_L^2 + \sigma_S^2)} \quad (5.1-41)$$

**Failure due to dynamic load**

During its life cycle a structure is subjected to a spectrum of dynamic loads, that varies in amplitude, average value, as well as in frequency. It is important to model this load spectrum properly to obtain reliable results. However, to make an exact and accurate description of the loads on a structure is a complex matter, which is not within the scope of this thesis. Therefore, a less complex model was chosen. The actual load spectrum on a structure was replaced by a representative dynamic load with constant frequency and amplitude.

The paragraphs below describe the strength of components subjected to dynamic loads. The Palmgren and Miner diagram of figure 5.1-12 describes the relation between the maximum number of load cycles $N$ and the material strength $S$ for loads with a constant amplitude.
The horizontal line on the left side of the diagram describes the behaviour of components subjected to low cycle fatigue or static loads as discussed in the paragraphs above.

The horizontal line on the right side of the diagram describes behaviour of components subjected to dynamic loads that are below the endurance strength $S_e$.

The behaviour of components subjected to loads between the yield strength $S_y$ and the endurance strength $S_e$ is described below:

$$\log(S) = a \cdot \log(N) + b \Leftrightarrow$$

$$\Leftrightarrow N = 10^{\frac{\log(S) - b}{a}}$$

$$a = \frac{\log\left(\frac{S_e}{S_y}\right)}{\log\left(\frac{n_e}{n_1}\right)}$$

$$b = \frac{\log(S_y) \cdot \log(n_e) - \log(S_e) \cdot \log(n_1)}{\log\left(\frac{n_e}{n_1}\right)}$$

If $n_1 = 10^3$, $n_e = 10^6$, $S_e = 0.45S_y$, then $a = -0.116$, $b = \log(S_y/0.45)$.

The probability of failure as a function of time must be calculated to calculate the failure rate. However, the Palmgren and Miner diagram does not describe the variation of strength in time. It describes how many load cycles are allowed for a certain applied load.

The maximum number of load cycles varies from component to component. Thus, as time proceeds, the number of failed components from a certain population increases. The probability density function of the maximum number of load cycles, $f_{\text{max}}$, describes this process. A

Calculation of the probability of failure
Gauss-Normal probability density function is assumed for both the material strength and the maximum number of load cycles. Figure 5.1-13 describes the relationship between the standard deviations $\sigma$ of both functions.

The number of load cycles at which a certain component fails is represented by $n$. The probability that a component has failed between 0 and $N$ cycles, is:

$$P(n \leq N) = \int_{-\infty}^{+\infty} f_{L}(L).f_{N}(n, L).dn.dL =$$

$$= \int_{-\infty}^{+\infty} f_{L}(L).(F_{N}(N, L) - F_{N}(0, L)).dL \quad (5.1-45)$$

The failure rate $\lambda$ is proportional to $dP/dN$ (for small numbers of $N$). For $t$ is small:

$$\lambda(t) \approx dP/dt \quad (5.1-46)$$

$$N = \alpha.t \quad (5.1-47)$$

The number of load cycles per unit time is $\alpha$.

$$dP/dt = (dP/dN).(dn/dt) = \alpha.dP/dN \quad (5.1-48)$$

O'Connor [44] observes, that the number of load cycles until failure $N$ is log normal or Weibull distributed. However, in formula (5.1-49) a normal distribution is substituted for two reasons: First, because this formula describes the implementation of the software described in chapter 4. Second, because the form of the log normal, Weibull, and normal distribution is comparable, when the mean value of $N$ is large and the standard deviation of $N$ is relatively small.

$$dP/dN = \int_{-\infty}^{+\infty} f_{L}(L).f_{N}(N, L).dL =$$

$$= \int_{-\infty}^{\infty} \frac{1}{\sigma_{L}.\sqrt{2\pi}} . e^{\frac{(L-\mu_{L})^{2}}{2\sigma_{L}^{2}}} \cdot \frac{1}{\sigma_{N}.\sqrt{2\pi}} . e^{\frac{(N-\mu_{N}(L))^{2}}{2\sigma_{N}^{2}}} . dL \quad (5.1-49)$$

Calculating the probability of failure
The average load is \( \mu_L \), while \( \sigma_N \) is the standard deviation of the load, \( \mu_N(L) \) is the average of the maximum number of load cycles, and \( \sigma_N \) is the standard deviation of the maximum number of load cycles. Note that \( \mu_N(L) \) is a function of the load \( L \!\!\!. \)

For \( N << \mu_N \):

\[
\frac{dP}{dN} \approx \int_{L=\infty}^{\infty} \frac{1}{\sigma_L \cdot \sqrt{2\pi}} . e^{-\frac{(L-\mu_L)^2}{2\sigma_L^2}} \cdot \frac{1}{\sigma_N \cdot \sqrt{2\pi}} . e^{-\frac{\mu_N(L)^2}{2\sigma_N^2}} . dL \tag{5.1-50}
\]

### 5.2 Uncertainty

The exact value of the probability of failure for a component depends on the circumstances in which the component is manufactured and used. Usually, a designer can not predict these circumstances. He does not know the circumstances in which the failure data were gathered, nor does he know whether application under different circumstances influences the failure data. Uncertainty in failure data is an important issue in reliability analysis.

This section begins with an overview of all uncertainties, including uncertainty in failure data, that are of importance to a designer. Then, it discusses how to take uncertainty in failure data into account.

#### 5.2.1 Uncertainties, an overview

During the design process a designer executes various analyses. The first step in each analysis is to make a physical model of the reality. Sometimes, it is useful to convert the physical model into a finite element model or another model, such as a flow diagram. The second step is to make a mathematical description of the model. The third and last step is to substitute data into the mathematical model and analyse it. Each modelling step is a simplification to make an understandable description of reality. The model does not describe certain parts of reality due to this simplification. Example: the assumption of a linear relationship between nodal displacements and element deformations causes inaccuracy for large displacements. Each modelling step also introduces errors; behaviour that does not exist in reality, but is described by the model. Example: instability of a numerical integration algorithm causes results, resembling vibrations in reality. Thus, each step introduces uncertainty in the model.

Calculation of the probability of failure
Step 1a: Physical model

Uncertainty: Does the physical model sufficiently describe the behaviour of the structure? The designer decomposes the structure into components. In the decomposition he simplifies the structure by neglecting details and grouping several small components into larger ones. Example: though a pump exists of bolts, bearings, seals and many other parts, it is modelled as only one component. The designer is uncertain if the simplifications affect the description of the behaviour of the structure, and if the components fully describe the behaviour of the components in reality.

Uncertainty: Is it sufficient to consider the structure only? In the simplifications of the system the designer usually leaves out the most important source of failure: human failure. Furthermore, he leaves out all other sources of failure that originate outside the system. Therefore, the analysis only predicts failure invoked by the system itself. The designer is uncertain about causes of failure from outside of the structure.

Uncertainty: Does the physical model fully and sufficiently describe the behaviour of the environment of the structure? The designer models the environment of the structure with loads. He applies the loads on the nodes connecting the components. The designer is uncertain about the size of the loads, and the place where they should be attached to the structure. Furthermore, he is uncertain if loads are the only part of the environment, that should be modelled. Is the structure exposed to heat, chemicals, or salt water? Does this influence the strength of the structure?
■ Step 1b: Finite element model

Uncertainty: Are linear equations sufficient to describe the behaviour of the structure? It is not possible to model large displacements, dynamics, and buckling with linear equations. The designer is uncertain if it is permissible to neglect this behaviour.

Uncertainty: Do the boundary conditions and loads sufficiently model the environment of the structure? The finite element model describes the environment only with loads and prescribed displacements. It does not describe heat, chemicals, or salt water. The designer is uncertain if these factors have influence on the structure.

■ Step 2: Mathematical model

Uncertainty: Does the equation solving algorithm produce valid results? The algorithm can produce invalid results due to limited accuracy or numerical instability. The designer can be uncertain if the results are valid.

■ Step 3: Substitution of data

Uncertainty: Can the data substituted in the mathematical equations be applied for the structure in specific circumstances? Loads, material strength, and failure data from databases are obtained in specific circumstances, and even within these circumstances the values vary. The designer is uncertain if he can apply the data from the databases for the structure in his environment.

The designer resolves the uncertainties about most of the above aspects by assuming, that they are outside of the system he is analysing. Thus, he isolates a structure he is reasonably certain about from the environment he is not certain about. However, he remains uncertain about the interface between the structure and its environment - the loads and the boundary conditions - and if he should take in account other interactions with the environment.

Nevertheless, he can not move the uncertainty about the failure data of the components out of the system. The next sub section discusses how to take this uncertainty into account.

5.2.2 Uncertainty in data

Section 5.1 describes formulas to calculate the probability of failure and unavailability. These formulas are all functions of the probability of failure per demand, and failure rate of components. However, the exact
values of these parameters are not always known. Various data sources give different values for the same failure mode of a component. In many cases the maximum and minimum values vary by more than a factor of 100. Thus, the calculated probabilities and unavailabilities are strongly dependent on the chosen values of the failure data.

This section describes how to take the uncertainty in failure data into account. The paragraphs below describe how to model the uncertain parameters as stochastic variables, and how to evaluate the formulas of section 5.1 with stochastic parameters instead of discrete parameters.

Data sources often give a maximum and minimum value for the failure data, they give a mean value and a standard deviation, or they give a mean value and an error factor describing the 5 % lower and the 95 % upper boundary of the data. With this information it is possible to model the uncertainty of the failure data by estimating a distribution function representing the certainty for a range of values. In this way the failure data parameters are modelled as stochastic variables.

Since the probability of failure per demand and the failure rate always have positive values, the domain of the certainty distribution function must contain positive values only. The normal distribution function is inappropriate, because its domain contains both positive and negative values. Therefore, the log-normal distribution function, which has a positive domain, was chosen to model the certainty distribution of the failure data.

Sub section 5.1.5 shows how to describe the probability of failure, and the unavailability as the sum and product of variables. Some of these variables are certain, some are not. Example: in formula (5.1-33) the variable $n$, describing the number of components in a minimal cut set, is certain, and $\lambda_i$, describing the failure rates of the components, is not certain. The paragraphs below demonstrate how to evaluate the formulas from subsection 5.1.5 by adding and multiplying certain and uncertain variables.

Sometimes, only a maximum and minimum value of the variable is known or estimated. This may be insufficient information to estimate a distribution function. In this case a deterministic model can be used to give lower and upper bounds of the probabilities and unavailabilities.

When a distribution function of the variables is known or can be estimated, a probabilistic model can be used. In order to determine the effect of uncertainty in data a probabilistic model of the joint uncertainty has to be built. Bedford [29] and Kuipers [39] describe this technique.

98 Calculation of the probability of failure
- **Deterministic model**
  
  If $x$ is certain, zero can be substituted for $\delta$.
  
  $$x = p \pm \delta$$  \hspace{1cm} (5.2-1)

  $$y = q \pm \varepsilon$$  \hspace{1cm} (5.2-2)

  - Sum of two variables:
    
    $$z = x + y = (p + q) \pm (\delta + \varepsilon)$$  \hspace{1cm} (5.2-3)

  - Product of two variables:
    
    $$z = x^*y = p.q \pm (|p.\varepsilon| + |\delta.q| + |\delta.\varepsilon|)$$  \hspace{1cm} (5.2-4)

- **Probabilistic model**

  - Sum of a stochastic and a non stochastic variable; suppose that $x$ is a stochastic variable and $b$ is a non stochastic variable. $z = x + b$:
    
    $$h(z) = f(z - b)$$  \hspace{1cm} (5.2-5)

  - Product of a stochastic and a non stochastic variable; suppose that $x$ is a stochastic variable and $b$ is a non stochastic variable. $z = x^*b$:
    
    $$h(z) = f(z/b)$$  \hspace{1cm} (5.2-6)

  - Sum of two independent stochastic variables. $z = x + y$:
    
    $$H(z) = P(z \leq z) = \int_{x+y}^{\infty} f(x).g(y).dx.dy =$$
    
    $$= \int_{-\infty}^{\infty} f(x).dx \int_{-\infty}^{z-x} g(y).dy =$$
    
    $$= \int_{-\infty}^{\infty} f(x).G(z-x).dx \Leftrightarrow$$
    
    $$\Leftrightarrow h(z) = dH(z)/dz = \int_{-\infty}^{\infty} f(x).g(z-x).dx$$  \hspace{1cm} (5.2-7)

  - Product of two independent stochastic variables. $z = x^*y$:
    
    $$H(z) = P(z \leq z) = \int_{x+y}^{\infty} f(x).g(y).dx.dy =$$
    
    $$= \int_{-\infty}^{\infty} f(x).dx \int_{-\infty}^{y/x} g(y).dy =$$
    
    $$= \int_{-\infty}^{\infty} f(x).G(z/x).dx \Leftrightarrow$$
    
    $$\Leftrightarrow h(z) = dH(z)/dz = \int_{-\infty}^{\infty} f(x).g(z/x).d(z/x)/dz =$$
    
    $$= \int_{-\infty}^{\infty} 1/x.f(x).g(z/x).dx$$  \hspace{1cm} (5.2-8)
5.3 Dependent failure

Sub section 5.1.5 describes how to calculate the probability of failure and the unavailability for independently failing systems. In many cases the assumption, that the failure of components in a minimal cut set is independent, is to optimistic. This section begins by describing the causes of dependent failure. It ends by discussing various models used to calculate more realistic probabilities of failure and unavailabilities.

Van Gestel [22] describes eight categories of dependent failure:

1. Common support systems: Several systems or components depending on the same support system, such as the power or water supply. Failure of the power or water supply will cause all depending systems to fail.

2. Functional dependency: A back up system depending on another system to be activated, such as an emergency power supply. The emergency power supply depends on the system, that uncouples the public electricity net from the local net, before it can start.

3. Common cause start events: The failure of a support system obstructs safety systems from operating. Failure of the support system does not lead to failure of the total system directly, but causes a hazardous situation, in which safety systems can not prevent accidents.

4. Physical interaction dependency: Exposure of system components to hot or corrosive gases or fluids that are accidentally released (from the system). This can lead to failure of several redundant components.

5. Operator interaction dependency: Failure of several redundant systems due to exploitation faults.

6. Common cause failure: Failure of two or more identical components due to a common cause, that is not one of the five causes mentioned above. Common cause failure can be invoked by design, manufacturing, assembly, test and maintenance faults and exposure to extreme environmental conditions such as heat or salt water.

7. Dependency in human action: To prevent human mistakes systems often require multiple action from an operator, such as software that requires a conformation by the user before it takes certain action. However, it is possible, that an operator makes successive mistakes, and causes failure.
External events: Events from outside of the system, such as flooding, impact of a crashing aeroplane, explosions, and earthquakes causing failure.

It is possible to describe the first five of these categories of dependency in a fault tree. The paragraphs below describe various techniques to take the 6th category, common cause failure, into account. Van Gestel [22] and Bedford [29] also describe these techniques. It would be too ambitious to describe how to take the 7th and 8th category into account in this thesis. Therefore, for human failure this thesis refers to Kirwan [46]. For failure due to external events this thesis refers to Vrijling [42], [43] and the Ministry of Housing, Land use planning, and Environment [47]. For more about dependent - common cause - failure this thesis refers to Bedford [61], and Ansell [64].

■ Square root bounding model

Bedford [29] refers to the Rasmussen report WASH-1400 [48], which describes the square root bounding model. This model assumes, that the total probability of failure $P_{tot}$ is the square root of the product of the independent probability of failure $P_i$ and the maximum dependent probability of failure $P_d$:

$$P_{tot} = \sqrt{(P_i \cdot P_d)} \tag{5.3-1}$$

However, Bedford [29] refers to Lewis [49], who reviewed the Rasmussen Report [48], to explain that this model is not valid and should not be used.

■ Marshall-Olkin model

This model assumes that a system of components is subjected to shocks, which can cause failure of the components. A shock can cause failure of different subsets of components in the system. Assume that the probability of failure of each subset $s$ fails according to a Poisson process with failure rate $\lambda_s$. The (dependent) failure rate of a component, $\lambda_c$, is the sum of the failure rates of all subsets, that it is part of.

$$\lambda_c = \Sigma \lambda_s, \text{ if component } i \text{ is a part of the subset} \tag{5.3-2}$$

■ Beta factor model

The beta factor model assumes that the failure rate of a component, $\lambda_{tot}$, consists of an independent part, $\lambda_i$, and a dependent part, $\lambda_d$:
\[ \lambda_{\text{tot}} = \lambda_i + \lambda_d \quad (5.3-3) \]

\( \beta \) is the fraction of the dependent failure rate and the total failure rate:

\[ \beta = \frac{\lambda_d}{(\lambda_i + \lambda_d)} \iff \quad (5.3-4) \]

\[ \iff \lambda_d = \beta/(1 - \beta) \lambda_i \Rightarrow \quad (5.3-4) \]

Assume, that a common cause makes all components in a system fail with certainty. Thus, the dependent probability of failure for a system of components is equal to the dependent probability of failure for a single component. The total probability of failure for a system is equal to the sum of the dependent probability of failure \( P_d = \beta/(1 - \beta) P_{i,\text{comp}} \) and the independent probability of failure of the system \( P_{i,\text{syst}} \).

Three approaches exist to calculate the total probability of failure for a system of components: Bedford [29] multiplies the independent probability of failure of the system by the probability, that the dependent failure not occurs: \( 1 - P_d = (1 - \beta \cdot \beta P_{i,\text{comp}})/(1 - \beta) \). Thus, the total probability \( P_{\text{tot}} \) is:

\[ P_{\text{tot}} = (1 - \beta - \beta P_{i,\text{comp}})/(1 - \beta) P_{i,\text{syst}} + \beta/(1 - \beta) P_{i,\text{comp}} \approx \beta/(1 - \beta) P_{i,\text{comp}} \quad (5.3-5a) \]

Ansell [64] does not multiply the independent probability of failure of the system by an extra factor. Thus the total probability \( P_{\text{tot}} \) is:

\[ P_{\text{tot}} = P_{i,\text{syst}} + \beta/(1 - \beta) P_{i,\text{comp}} \approx \beta/(1 - \beta) P_{i,\text{comp}} \quad (5.3-5b) \]

Van Gestel [22] totally neglects the independent probability of failure for the system. Thus the total probability \( P_{\text{tot}} \) is:

\[ P_{\text{tot}} = \beta/(1 - \beta) P_{i,\text{comp}} \quad (5.3-5c) \]

All approaches are legitimate, since in most cases the contribution of the independent probability of failure \( P_{i,\text{syst}} \) to the total probability of failure \( P_{\text{tot}} \) can be neglected.

Van Gestel’s approach of the beta factor model was implemented in the software described in chapters 2 and 4.

The beta factor model does not offer the possibility to differentiate between systems with two redundant components and more dependent components. It is also not possible to take \( n \) out of \( i \) systems into account. These are systems with \( n \) redundant components from which \( i \) components are necessary to allow the system to function. If this model is applied, a designer is not rewarded for introducing extra redundancy in his design. With the binomial failure rate model it is possible to take extra redundancy as well as \( n \) out of \( j \) systems into account.
**Binomial failure rate model**

Like the beta factor model, the binomial failure rate model assumes that the failure rate of a component consists of an independent and a dependent part, \( \lambda_d \). The dependent part of the failure rate for one single component is:

\[
\lambda_d = \mu.p.q^{(n-1)} \tag{5.3-6}
\]

The rate of the shocks introducing common cause failure is described by \( \mu \), while \( p \) is the probability of failure of one component due to a shock. The variable \( q \) is the probability that all other components in a set of size \( n \) did not fail; \( q = 1 - p \).

The dependent part of the failure rate for one component out of a set of \( n \) components is:

\[
\lambda_d = n.\mu.p.q^{(n-1)} \tag{5.3-7}
\]

The dependent part of the failure rate for any subset of size \( j \) out of a set of \( n \) components is:

\[
\lambda_d = \mu.p^j.q^{n-j} \tag{5.3-8}
\]

Thus, the dependent part of the failure rate for all subsets of size \( j \) out of a set of \( n \) components is:

\[
\lambda_d = \binom{n}{j}.\mu.p^j.q^{n-j} \tag{5.3-9}
\]

**Alfa factor model**

The variable \( \lambda_j^* \) is the failure rate for a particular set of components of size \( j \). The failure rate, \( \lambda_j \), for any set of size \( j \) out of a set of size \( n \) is:

\[
\lambda_j = \binom{n}{j}\lambda_j^* \tag{5.3-10}
\]

\[
\alpha_j = \lambda_j/(\lambda_1 + ... + \lambda_n) \Leftrightarrow \tag{5.3-11}
\]

\[
\Leftrightarrow \lambda_j = \alpha_j(\lambda_1 + ... + \lambda_n) \tag{5.3-12}
\]
5.4 Conclusion

This chapter describes two models used to calculate the probability of failure. The first model, failure per demand, describes failure of discontinuous processes. The second model, the failure rate model, describes failure of continuous processes. The division in continuous and discontinuous processes is very accurate. Therefore, it is necessary to divide continuous processes into the sub-processes rest and action, and discontinuous processes into start and stop.

In each of these sub-processes different failure mechanisms take place. Thus, it is necessary to apply different failure data in each sub-process. Unfortunately, the failure data bases do not provide data for all sub-processes. Therefore, for many processes failure data must be estimated.

Furthermore, this chapter shows that uncertainty in failure data, and dependent failure can be taken in account.
6 Qualification, verification, validation, and evaluation

This chapter demonstrates that the software, described in the preceding chapters, produces satisfying results. The results consist of more than just the output of the software. It is possible to recognise various types of results: the analysis results of the software, the influence of the software on the design process, and the influence of the software on the designs themselves. This chapter briefly considers quality control to explain the various types of results that exist. To define these types the realisation process of a bridge is first considered. The realisation of the software is then considered, followed by a discussion of the results of the software described in this thesis.

Consider the realisation process of a bridge. The first step in this process is the problem definition. Examples of problems: a part of the country that is economically underdeveloped, and traffic congestion. The problem leads to a need for a solution. Examples of needs: need for economic development, need for better road connections, and a need to improve the traffic flow. Examples of solutions: a tunnel, a bridge, and a ferry. The next step is to list the requirements of the solution/bridge: how should the bridge perform? Examples of requirements: the number of vehicles that will use the bridge each year, the number of ships that will cross under the bridge, and the requirement for a moveable part in the bridge. The next step is to design the bridge and to define how it should be built in the specifications. As mentioned in Chapter 2, the design phase is separated into a conceptual design phase and a detailed engineering phase. Examples of specifications: the materials that must be used, and the exact dimensions of the bridge. The next step is to actually build the bridge, while the last step is to open the bridge for traffic.

Quality control requires various checks on the results of several steps. Figure 6-1 gives a schematic overview of the steps and quality

105
checks in the realisation process of a bridge. It is possible to recognise three types of checks:

The first type of check is *verification* - compare the results with the requirements. The first verification possibility of the bridge is after the conceptual design phase, the second time after the detailed engineering phase, and the third time after the construction of the bridge has been completed.

The second type of check is *qualification* - compare the results with the design, the *specifications*. The first qualification possibility of the bridge is after the detailed engineering phase, when it is possible to compare the detailed design with the conceptual design. The second

---

**Figure 6.1:** Qualification, verification, validation, and evaluation.
qualification is after the bridge has been completed, when it is possible to compare the bridge with the design. A qualification compares results with specifications, that have been derived from the requirements.

The third type of check is evaluation - record how the structure behaves after it has been finished, and see if it solves the problem, that it was built for. It is possible to learn how to improve designs from an evaluation. Deming [50] states, that this is an important step to improve quality of future designs.

The results of the qualification and verification checks are the basis of the decision to proceed to the next phase. The decision that the results of a phase are satisfying, and to proceed to the next phase is called validation of a design or structure.

The realisation process of software is different from that of a bridge. In software development it is not possible to separate the realisation of software into a conceptual design phase, a detailed engineering phase and a construction phase. However, there are similarities in both processes. It is possible to compare the development of the software with the conceptual design and detailed engineering phases of a bridge. The difference between software and bridge design is that in software development the design is the final product, and in the realisation of a bridge it is just a sub product.

The bridge example illustrates, the three types of quality checks: verification, qualification, and evaluation. This chapter applies these checks to the reliability analysis software described in this thesis.

The preceding chapters of this thesis specify the reliability analysis software. Section 6.1 shows, that the software works according to the specifications. It describes the analysis of simple structures, that can be analysed by hand. The results of the hand analysis are compared with the results of the software. Besides this quality check, section 6.1 also compares the results of different algorithms, that should produce the same results. The algorithms are based on the classic finite element theory and Schwab’s theory (see section 4.3). This section describes the qualification of the software, by checking whether the software works according to specifications.

Section 6.2 demonstrates, that it is possible to model real structures, and shows that the software finds valid minimal cut sets by comparing the results of a manual analysis with the results of the software. This section describes the verification of the software, by checking whether it is possible to analyse the type of structures that it is designed for.

Section 6.3 discusses the results of regular fault tree analyses, and fault tree analysis with the software. Furthermore, it shows how future users of the software were trained. This section describes the evaluation
of the software, by checking whether the introduction of the software leads to an improved design process and better results.

For more about quality control this thesis refers to Deming [50] and Maas and Bollen [51].

## 6.1 Qualification

This section qualifies the algorithms in the reliability analysis software. Table 6.1-1 shows which subsections give the qualification results of which algorithms. Subsection 6.1.1 to 6.1.6 demonstrate, that the developed software produces correct results in the static analysis, and that it finds the minimal cut sets for reliability analysis of the function *carrying load*. Subsection 6.1.7 and 6.1.8 show, that the software finds valid minimal cut sets for reliability analysis of the function *executing motion*.

This section qualifies all algorithms in the software. The first three algorithms in Table 6.1-1 are algorithms for static analysis; the third algorithm is physically non linear. The fourth to the sixth algorithm are algorithms for reliability analysis of the function ‘carrying load’. The last three algorithms are algorithms for reliability analysis of the function ‘executing motion’. The letter ‘m’ denotes that the mechanical domain was tested, while ‘h’ shows that the hydraulic domain was tested.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>subsection: 6.1.1</th>
<th>6.1.2</th>
<th>6.1.3</th>
<th>6.1.4</th>
<th>6.1.5</th>
<th>6.1.6</th>
<th>6.1.7</th>
<th>6.1.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>static</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>classic</td>
<td>m</td>
<td>m</td>
<td>m, h</td>
<td>m, h</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Schwab</td>
<td>m</td>
<td>m</td>
<td>m, h</td>
<td>m, h</td>
<td>h</td>
<td></td>
<td>h</td>
<td></td>
</tr>
<tr>
<td>nlinSchwab</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>carry load</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>collclassic</td>
<td>m</td>
<td>m</td>
<td>m, h</td>
<td>m, h</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>collSchwab</td>
<td>m</td>
<td>m</td>
<td>m, h</td>
<td>m, h</td>
<td>h</td>
<td></td>
<td>h</td>
<td></td>
</tr>
<tr>
<td>nlincoll</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>exec. motion</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nomoclassic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>m, h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nomoSchwab</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>m, h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nlinnomo</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>m, h</td>
<td></td>
</tr>
</tbody>
</table>

*Table 6.1-1: Relation between tests and subsections.*

108 Qualification, verification, validation, and evaluation
6.1.1 Five trusses

This subsection describes the analysis of a model with five trusses. Four of the trusses form a square, while the fifth truss is the diagonal of the square.

A horizontal load of 10,000 N is applied to the upper right corner of the square. This load causes stresses in the diagonal truss (number 5), and the vertical truss on the right (number 3). The other trusses do not carry any load. The stress in the diagonal truss (number 5) = \(\sqrt{2} \times 10,000 = 1.4142\times10^4\) N, while the stress in the vertical truss on the right (number 3) = \(-1.000\times10^4\) N.

The analysis of the function carrying load only results in minimal cut sets of one element, since there are no redundant elements in the model. The algorithms only find minimal cut sets with the loaded components (number 3, and 5).

<table>
<thead>
<tr>
<th>comp. nr.</th>
<th>elem. nr.</th>
<th>def. nr.</th>
<th>expected v.</th>
<th>Classic</th>
<th>Schwab</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>(-1.000\times10^4)</td>
<td>(-1.000\times10^4)</td>
<td>(-1.000\times10^4)</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1</td>
<td>(1.414\times10^4)</td>
<td>(1.414\times10^4)</td>
<td>(1.414\times10^4)</td>
</tr>
</tbody>
</table>

*Table 6.1-2: Static analysis, expected and calculated values.*

6.1.2 Six trusses

This subsection describes the analysis of a model with six trusses. Four of the trusses form a square, while the fifth and sixth trusses are the diagonals of the square.
A horizontal load of 10,000 N is applied to the upper right corner of the square. This load causes stresses in all trusses.

All minimal cut sets of the analysis of the function carrying load contain two elements, because there are redundant elements in the model.

All components carry the load, when component number 1 fails. This results in the minimal cut sets: \( \{1, 2\} \), \( \{1, 3\} \), \( \{1, 4\} \), \( \{1, 5\} \), and \( \{1, 11\} \).

Components number 4 and 11 carry no load, when component 2 fails. This results in the minimal cut sets: \( \{1, 2\} \), \( \{2, 3\} \), and \( \{2, 5\} \).

All components carry the load, when component number 3 fails. This results in the minimal cut sets: \( \{1, 3\} \), \( \{2, 3\} \), \( \{3, 4\} \), \( \{3, 5\} \), and \( \{3, 11\} \).

Component numbers 2 and 11 carry no load, when component 4 fails. This results in the minimal cut sets: \( \{1, 4\} \), \( \{3, 4\} \), and \( \{4, 5\} \).

All components carry the load, when component number 5 fails. This results in the minimal cut sets: \( \{1, 5\} \), \( \{2, 5\} \), \( \{3, 5\} \), \( \{4, 5\} \), and \( \{5, 11\} \).

Component numbers 1, 2, and 4 carry no load, when component number 11 fails. (This results in the minimal cut sets: \( \{3, 11\} \) and \( \{5, 11\} \).)

**Comment on results**

The software does not find the minimal cut sets \( \{1, 2\} \), \( \{1, 4\} \), \( \{1, 11\} \), and \( \{3, 5\} \), because the program uses a linear model, that does not describe instability.
<table>
<thead>
<tr>
<th>comp. nr.</th>
<th>elem. nr.</th>
<th>def. nr.</th>
<th>expected v.</th>
<th>Classic</th>
<th>Schwab</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td>3.964*10³</td>
<td>3.964*10³</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
<td>3.964*10³</td>
<td>3.964*10³</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td></td>
<td>-6.036*10³</td>
<td>-6.036*10³</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
<td></td>
<td>3.964*10³</td>
<td>3.964*10³</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1</td>
<td></td>
<td>8.536*10³</td>
<td>8.536*10³</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>1</td>
<td></td>
<td>5.607*10³</td>
<td>5.607*10³</td>
</tr>
</tbody>
</table>

Table 6.1-3: Static analysis: comparison of the results of the algorithms ‘classic’ and ‘Schwab’.

6.1.3 One hydraulic cylinder

The model of subsection 6.1.1 was extended with a hydraulic cylinder. The cylinder was attached to the upper right corner of the model. The rest of the model contains five trusses. Four of the trusses form a square, while the fifth truss is the diagonal of the square.

A horizontal load of 10,000 N is applied to the upper right corner of the square. The load is obtained by prescribing a pressure of 0.7278 N/mm² below the piston (at the left), and a pressure 0.0 N/mm² above the piston (at the right). The ring cross section of the piston is 1.374*10⁴ mm². This load causes stresses in the diagonal truss (number 5), and the vertical truss on the right (number 3). The other trusses do not carry any load.

The stress in the diagonal truss (number 5) = \( \sqrt{2*10,000} = 1.4142*10^4 \) N, while the stress in the vertical truss on the right (number 3) = -1.000*10⁴ N.
Besides the minimal cut sets found by the collapse analysis of subsection 6.1.1, the algorithm finds minimal cut sets with failure modes of the hydraulic cylinder.

<table>
<thead>
<tr>
<th>comp. nr.</th>
<th>elem. nr.</th>
<th>def. nr.</th>
<th>expected v.</th>
<th>Classic</th>
<th>Schwab</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>-1.000*10^4</td>
<td>-1.000*10^4</td>
<td>-1.000*10^4</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1</td>
<td>1.414*10^4</td>
<td>1.415*10^4</td>
<td>1.415*10^4</td>
</tr>
<tr>
<td>11</td>
<td>16</td>
<td>1</td>
<td>-1.000*10^4</td>
<td>-1.000*10^4</td>
<td>-1.000*10^4</td>
</tr>
<tr>
<td>11</td>
<td>17</td>
<td>2</td>
<td>7.278*10^3</td>
<td>7.278*10^3</td>
<td>7.278*10^3</td>
</tr>
<tr>
<td>11</td>
<td>20</td>
<td>1</td>
<td>-7.278*10^3</td>
<td>-7.278*10^3</td>
<td>-7.278*10^3</td>
</tr>
<tr>
<td>11</td>
<td>20</td>
<td>2</td>
<td>3.639*10^3</td>
<td>3.639*10^3</td>
<td>3.639*10^3</td>
</tr>
<tr>
<td>11</td>
<td>21</td>
<td>1</td>
<td>1.000*10^4</td>
<td>1.000*10^4</td>
<td>1.000*10^4</td>
</tr>
</tbody>
</table>

*Table 6.1-4: Static analysis: expected and calculated values.*

### 6.1.4 Two hydraulic cylinders

The model in this subsection is a symmetric model with two levers, that are connected by two hydraulic cylinders, and a pipe. A vertical load of -10,000 N is applied to the lever on the left. The lever on the right is fixed to the ground. A pressure 0.0 N/mm² is prescribed to the lower sides of the cylinders.

The ring cross sections of the pistons are 1.374*10⁴ mm². The vertical load on the left cylinder is 10,000 N. This load causes a pressure of 0.7278 N/mm² in the cylinders and the pipe. The reaction force on node number 2 is 10,000 N.

The results of the reliability analysis of the function *carrying load* are symmetrical. Each cut set with an element of the right side of the model corresponds to a cut set with an element of the left side. The probabilities of failure of left and right side elements are identical.
Figure 6.1-7: Component model with five trusses and a hydraulic cylinder.

Figure 6.1-8: Finite element model in undeformed and deformed position.

<table>
<thead>
<tr>
<th>comp. nr.</th>
<th>elem. nr.</th>
<th>def. nr.</th>
<th>expected v.</th>
<th>Classic</th>
<th>Schwab</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>4</td>
<td>1.000*10^7</td>
<td>1.000*10^7</td>
<td>1.000*10^7</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>1</td>
<td>1.000*10^9</td>
<td>1.000*10^9</td>
<td>1.000*10^9</td>
</tr>
<tr>
<td>10</td>
<td>34</td>
<td>1</td>
<td>-1.000*10^4</td>
<td>-1.000*10^4</td>
<td>-1.000*10^4</td>
</tr>
<tr>
<td>10</td>
<td>35</td>
<td>2</td>
<td>7.278*10^1</td>
<td>7.278*10^1</td>
<td>7.276*10^1</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>1</td>
<td>1.000*10^4</td>
<td>1.000*10^4</td>
<td>1.000*10^4</td>
</tr>
<tr>
<td>11</td>
<td>24</td>
<td>1</td>
<td>-1.000*10^4</td>
<td>-1.000*10^4</td>
<td>-1.000*10^4</td>
</tr>
<tr>
<td>11</td>
<td>25</td>
<td>2</td>
<td>7.278*10^1</td>
<td>7.275*10^1</td>
<td>7.276*10^1</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>2</td>
<td>7.278*10^1</td>
<td>7.278*10^1</td>
<td>7.276*10^1</td>
</tr>
</tbody>
</table>

Table 6.1-5: Static analysis: expected and calculated values.
6.1.5 Open non return valve

This subsection describes the analysis of a model with two pipes, and one non return valve. On the left side of the model a pressure of 15 N/mm² is prescribed. The pressure opens the non return valve. The pressure in all pipes is 15 N/mm², since there is no pressure loss in the system.

The reliability analysis of the function carrying load only finds minimal cut sets of one element, because there are no redundant elements in the model. Expected cut sets: {elem. nr. 1, def. nr. 2}, {elem. nr. 2, def. nr. 2}, {elem. nr. 3, def. nr. 2}.

<table>
<thead>
<tr>
<th>comp. nr.</th>
<th>elem. nr.</th>
<th>def. nr.</th>
<th>expected v.</th>
<th>Nlinschw</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.500*10¹</td>
<td>1.500*10¹</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1.500*10¹</td>
<td>1.500*10¹</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1.500*10¹</td>
<td>1.500*10¹</td>
</tr>
</tbody>
</table>

Table 6.1-6: Static analysis: expected and calculated values.

6.1.6 Closed non return valve

This subsection describes the analysis of a model with two pipes, and one non return valve. On the left side of the model a pressure of 15 N/mm² is prescribed. The non return valve is orientated in the opposite direction of the non return valve in model Nonlin01.mod.

The pressure closes the non return valve. The pressure in pipe number 1 is 15 N/mm², while the pressure in pipe number 2 is zero.
Expected cut sets: \{elem. nr. 1, def. nr. 2\}, \{elem. nr. 3, def. nr. 2\}, \\
\{elem. nr. 3, def. nr. 1; elem. nr. 2, def. nr. 2\}.

<table>
<thead>
<tr>
<th>comp. nr.</th>
<th>elem. nr.</th>
<th>def. nr.</th>
<th>expected v.</th>
<th>Nlinschw</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.500*10^1</td>
<td>1.500*10^1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0.000</td>
<td>2.455*10^0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1.500*10^1</td>
<td>1.255*10^1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>7.500*10^0</td>
<td>8.727*10^0</td>
</tr>
</tbody>
</table>

*Table 6.1-7: Static analysis: expected and calculated values.*

- **Comment on results**

The differences between the expected and calculated values can be explained by the stiffness of the non return valve.

**6.1.7 Two hydraulic cylinders and a prescribed displacement**

The model in this subsection is a symmetric model with two levers, that are connected by two hydraulic cylinders, and a pipe. A vertical displacement is applied to the lever on the right. A pressure 0.0 N/mm² is prescribed to the lower sides of the cylinders.

The minimal cut sets found with the classic algorithm are equal to the minimal cut sets found with Schwab's algorithm for no-motion analysis.
6.1.8 Two hydraulic cylinders, a non return valve, and a prescribed displacement

The model in this subsection is a symmetric model with two levers, that are connected by two hydraulic cylinders, and a pipe with a non return valve. A vertical displacement is applied to the lever on the right. The non return valve will open due to this displacement. A pressure of 0.0 N/mm² is prescribed to the lower sides of the cylinders.

The minimal cut sets found with the classic algorithm and Schwab’s algorithm were equal to the minimal cut sets found with the non linear algorithm for no-motion analysis, except for the connecting pipe and the non return valve. Extra cut sets with an extra pipe, and a non return valve are found.
6.2 Verification

The previous paragraphs demonstrate, that the algorithms of the software produce correct results. This section goes one step further by checking if it is possible to use the software to analyse a real structure. Two examples are given: the Double roll-bascule bridges in Terneuzen, and the Storm surge barrier in the Oosterschelde.

6.2.1 Double roll-bascule bridges in Terneuzen

In the beginning of 1998, a ship collided with the double roll-bascule bridges over one of the locks in Terneuzen, The Netherlands. The bridges were damaged in such a way, that they had to be rebuilt. The Mechanical Engineering Department of the Construction Division of

Kiestra [6] modelled and analysed these three alternatives to verify, whether the automated reliability analysis method can be applied to regular structures. He analysed the design alternatives by hand, and compared the results of the manual analysis with those of the automated analysis, to check whether the automated method produces correct results. The software produced satisfying results.

- **Alternative I: roll-bascule bridge with mechanical drive train.**

![Component model of a roll-bascule bridge with a mechanical drive train.](image)

*Figure 6.2-1: Component model of a roll-bascule bridge with a mechanical drive train.*

![Cross section of a roll-bascule bridge with a mechanical drive train.](image)

*Figure 6.2-2: Cross section of a roll-bascule bridge with a mechanical drive train.*
Alternative II: bascule bridge with a mechanical drive train.

Figure 6.2-3: Component model of a bascule bridge with a mechanical drive train.

Figure 6.2-4: Cross section of a bascule bridge with a mechanical drive train.
Alternative III: bascule bridge with a hydraulic drive train.
6.2.2 Storm surge barrier in the Oosterschelde

For maintenance it is desirable to permanently remove one of the hydraulic tubes for a part of the drive train of the storm surge barrier in the Oosterschelde. If the tube is removed, some parts of the structure can be reached easier for painting. Avontuur [53] modelled a part of the structure to consider whether this is allowable, regarding the probability of failure.

Figures 6.2-9 to 6.2-11 show the three solutions, that were studied: the original situation, the structure with only one tube removed, and with one tube and two valves removed. The analysis results in Table 6.2-1 show that the reliability of the structure slightly decreases when the tube is removed.
6.3 Evaluation

As mentioned before, it is possible to separate the results of the automated reliability method into three parts: First, the results of the software itself, second, the influence on the design process, and third, the influence on the designs. After the software has been released, introduced, and used for a while in the design process, it is possible to evaluate the results. This process will take several years. Even though this process has not been completed, it is possible to predict some of the results.

This section compares the manual reliability analysis method with the automated reliability analysis method, and demonstrates that the automated method is better, improves the design process, and even improves the designs (see also subsection 6.2.2).

This chapter covers the technical results, the software and the possible influence of the software on its environment - the design process.
and the designs. The success of the software depends on the acceptance by the future users, which can also been seen as a result. Chapter 7 discusses this part of the results.

- Manual reliability analysis

Kiestra [6] studied the manual reliability analysis of several structures including the Storm surge barrier in the Hartel Canal, and the locks between the North Sea Channel and the IJsselmeer - the Nieuwe Oranjesluizen. From these analyses he found, that the designers were not able to execute the reliability analysis themselves. In most cases an external expert executed the reliability analysis. This section describes Kiestras conclusions. In the analyses he found the following shortcomings:

The construction of a fault tree is usually an important part of the reliability analysis. In the analysis of the storm surge barrier in the Hartel Channel the risk analyst did not apply the Boolean algebra rules to simplify the fault tree to find the minimal cut sets. He substituted the failure data in the fault tree to calculate the probability of failure. It is possible, that this leads to a too optimistic probability of failure.

In the manual analysis of the Nieuwe Oranjesluizen the risk analyst formulated the failure events in the fault tree inaccurately, such as: 'component unavailable', or 'component fails'. This is inaccurate, because a component can have multiple failure modes. A component can appear on several places in the fault tree with different failure modes. If several failure modes have the same description, the simplification of the fault tree with Boolean algebra leads to errors in the minimal cut sets. Furthermore, it is possible, that this results in errors in the quantification of the fault tree, since the risk analyst has to choose between data of different failure modes. The risk analyst of the Nieuwe Oranjesluizen also did not use the rules of Boolean algebra to simplify the fault tree and find the minimal cut sets. The analysis took 200 hours, instead of the 80 hours scheduled. The design contained two redundant pump units. The risk analysis demonstrated, that one pump unit with redundant non return valves would have been sufficient. However, the analysis was executed while the structure was already under construction. Therefore, it was too late to alter the design. If the analysis had been executed in an earlier stage of the design process, it would have been possible to save one pump unit.
Automated reliability analysis

Kiestra [6] also studied the automated reliability analysis method. He demonstrates, that the reliability analysis software is an adequate tool to generate design alternatives. He estimates, that an experienced user can build a model of a design alternative of equal complexity as alternative I, II, or III from section 6.2.1 in less than two hours. The analysis time of alternative III was the longest. It took the non linear algorithm over 4 hours on a Pentium 200 MHz to analyse the model. However, it is much faster than the manual analysis, which took about two weeks. Kiestra did another comparison between the manual and the automated method. In this comparison the manual method took 20 hours, while the automated method took 3 - 5 hours. A non linear algorithm, based on the classic instead of Schwab’s equations, was implemented to solve the problem of long calculation times. It reduces the analysis time with a factor 1½ to 2½.

The version of the software tested by Kiestra did not distinguish failure during rest, start, action, and stop. Furthermore, it did not calculate unavailabilities, nor take the uncertainty of failure data into account. It only calculated the probability of failure during start. Therefore, it did not find all minimal cut sets, the user had to estimate the unavailabilities by hand, and the boundaries of the results were unclear. The current version of the software features: failure during rest, start, action, and stop, unavailabilities, and gives 5 % and 95 % boundaries of the analysis results.

<table>
<thead>
<tr>
<th>aspect</th>
<th>manual</th>
<th>automated</th>
</tr>
</thead>
<tbody>
<tr>
<td>analysis time</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>analysis by designer possible</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>improvement to structures</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td>accurate description of failure modes</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>valid minimal cut sets</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>unavailabilities</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>uncertainty in data</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>rest, start, action, stop phases</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>number of plusses</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>number of minuses</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6.3-1: Advantages and disadvantages of the manual and automated reliability analysis method.

124 Qualification, verification, validation, and evaluation
Subsection 6.2.2 shows that it is possible to improve a design by re-analysing it with the reliability analysis software. The discussion about the manual analysis of the Nieuwe Oranjesluis in this section confirms, that it is possible to improve a design when a reliability analysis is executed in a preliminary stage of the design process. The software could be helpful to achieve this.

6.4 Conclusion

This chapter shows that the results of a (design) software package consist of more than just the results of the software. The quality of the results depends on the algorithms that produce correct results, the ability to model and analyse real structures, and the impact of the results on the designs and structures themselves. The algorithms of the automated reliability analysis method produce correct results by analysing various structures, which can be checked manually. Furthermore, the automated analysis of complex structures produces valid results too. Finally, the application of the software would lead to improvement in the reliability analysis methods of structures, along with changes and improvements to the structures themselves. The changes to structures would not lead to more reliable structures, but to less expensive structures with an equal reliability.

The automated reliability analysis method produces more reliable results than a manual method, since the automated method does not make mistakes, and the approach is equal for all problems, independent of the designer. Thus, the results of reliability analyses of different structures are more comparable.
The author of this thesis was involved in the development and introduction of two design systems for the Mechanical Engineering Department of the Engineering and Construction Division of the Ministry of Transport, Public Works, and Water Management. The reliability analysis software, which is subject of this thesis, is one of these systems. The other system was developed for (deterministic) structural analysis of drive trains for bridges, lock gates, storm surge barriers, etc., and has proved its worthiness for the analysis of other structures too. Both the reliability analysis software and the structural analysis software are based on the same design philosophy, as described in section 1.3.

During the development and introduction of a design system both technical and social problems have to be solved. All technical problems faced during the development process of both design systems appeared to be solvable. The social problems, however, were more difficult to deal with. They even appeared to be of major influence to the success of the development projects. This chapter discusses these social aspects, and various attempts and approaches to deal with them. Stark [51] describes acceptance problems with the introduction of CAD/CAM systems, and how to tackle them. For more information about the introduction of CAD/CAM this thesis refers to Preston [54]. The acceptance problems have social, as well as financial, technical and organizational grounds.

The social problems are the difficulties in convincing people of the necessity for the introduction of the design system. People doubt the benefits of the system, and see various obstacles, that hinder the introduction of the system. The designer fears loosing his job, and is unsure whether he is able to learn how to work with the design system. The manager of the design department is unsure whether it is technically achievable to increase production with the system. In fact, the intro-
duction of the design system will initially decrease the production, because the designers have to learn how to work with it. The financial department doubts if the investment in the design system will pay off. And finally, the top management of the company is worried about all the commotion connected with the design system.

To the designer, the introduction of a design system can be a radical change to his working environment. The computer takes over (some of) the tasks, that are part of his skills. Wagemans [3] says: 'To the designer-calculator the computer is a straight threat, and sometimes a dangerous weapon. The computer calculates much faster and more reliable than himself, threatening his social security. It is a dangerous weapon, if it is used by designers who do not have a feel (or do not want to have), for the behaviour of their structures. They use the computer as a black box, with all risks included, as I recently experienced in a large Dutch project.' Furthermore, the computer integrates and adds new tasks - that were executed by others before - to the job of the designer. For instance, it can integrate drafting and structural analysis, and drafting and cost analysis. Thus, the function of the designer becomes more complex, while the function and position of others becomes simpler. Therefore, the introduction of a design system can cause emotions. As a result, the designer will not be able to accept the system immediately.

An example of the financial problems are the difficulties in calculating if the system will pay off. The investments in a design system are very high, and it is difficult to express all benefits of a design system in money. Thus, the pay back time is very large; Stark [51] mentions pay back times of at least 5 years. However, Stark also mentions that CAD/CAM is not about pay back, but about staying alive: 'Among managers most likely to feel uncomfortable in the new climate, there are often those who believe that traditional methods are best and that computers and high technology create more problems than they solve. Unfortunately, the future is bleak for people who maintain such attitudes, since companies only have two choices - adapt to new circumstances or, like dinosaurs, go out of business.'

The technical problems are the difficulties in the selection of the system, and the uncertainty whether the promised benefits of the system will be reached. As mentioned above, the author of this thesis has experienced, that technical problems are much easier to solve than the other problems.

The organisational problems are the difficulties in involving everybody in the organisation in the introduction of the design system. Van Asch [40] advises the involvement of the future users in the develop-
ment of an automated system. It appears, that users accept a new system easier, if they are involved in the development. Furthermore, their advice can improve the systems functionality, and implementation in the organisation. The users can introduce their skills and knowledge in the development project, and foresee what parts of a process can be automated and not. Stark [51] even mentions, that the successful implementation of a design system depends not only on the involvement of the users, but on the involvement of representatives of all levels in the company. He insists on the involvement of the top management, middle management, future direct and indirect users of the design system.

This chapter begins by describing the influence of the organisation of a software development/automation project on the acceptance of the software. Then, it discusses the influence of the system architecture. Finally, it describes the experiences with the two development projects, mentioned above.

7.1 Project organisation

The involvement of all levels of the organisation is crucial to successfully develop and implement a design system. Therefore, the project manager running the development project should select people from all levels in the project organisation. Their contribution to the project and the part of the project organisation in which they operate, is described below.

This section describes the organisation of a development project. Besides the project manager and a development team, the organisation consists of a principal and a steering group, a user group, and contractors.

Principal, steering group

Figure 7.1-1 shows an organisation scheme of a development project. In the cases described in this thesis, the principal of the development project is a manager of a design department. A steering group advises the principal. The supervisor to the project manager can be a member of the steering group. Other members can be: a man-

Figure 7.1-1: Project organisation.
ager of one of the contractors, external experts.

To optimise the design process in his department the principal chose to automate a part of the process with a design system. The introduction of the chapter shows, that the introduction of a design system is a complex matter. Therefore, the position of the principal should be assured:

First, the principal should have full support from the rest of the management team of the design department. As demonstrated, new developments are often subject to discussion. The management should have a unanimous and positive opinion about the necessity for the development. If the opinions in the management team are divided, the opponents of the development will use their indecisiveness to obtain supporters for their point of view within the management team. This results in a weakened position of the principal.

Second, the principal should have full support of the higher management in the company. Development projects often require little budgets per year, but run over several years. Thus, the total costs can be high. The higher management should be well informed about the total required budget for the project. If they are not informed, they will not support the project. This also weakens the position of the principal.

- **User group**

The acceptance of a design system by future users is of major influence for the success of a software development project. Therefore, it is wise to install a group of future users to advise the development team. Van Asch [40] presents 9 rules for successful user participation:

1. Define the word *user* clearly. It is possible to recognise three types of users: Policy defining users (managers), functional users (superintendents, who manage a small group of employees), and operational users (they use the system). Van Ash advises to involve especially the last group of users, the operational users, in the development project, because they will have to use and accept the system in the end.

2. Form a user group, organise the future users.

3. Define the aspects of the system, that are to be judged by the user group. For instance: recognisable functions and possibilities, the user interface, implementation in the organisation, reliability.

4. Offer the users the possibility to really influence the development process.
Clearly define the moments of consultation of the company advisory board.

Clearly define the moments of consultation of the user group.

Train the users. The users should have enough knowledge - technically, about the project, and the organisation - to communicate with the developers.

The users of the user group should communicate with their colleagues about the project. The management should support them.

All users - not only the users of the user group - should inform the management about the changes in the organisation invoked by the automation. The management often does not know the need for training. The company advisory board can play a role in this process.

The two development projects, described in this section, show that the composition of the user group is very important:

It is important to select participants, who are enthusiastic for the development. New developments always encounter resistance. Therefore, it is very important that some people outside the development group support them.

Also select user group members, who are representatives of their department. They should be held in esteem by their colleagues. If they approve of the design system, their colleagues should embrace their opinion.

Select participants, who are willing to change. A new design system can change the design process. The user group should be able to accept this. Therefore, user group members are preferred to be young, since young people usually accept changes more easily than older people.

Despite all good advise in the paragraphs above Van Asch [40] recognises a number of limitations to the participation of users in automation projects:

- Not all future users of a department can participate in the development project. The users who participate will accept the development, while the rest of the department will hesitate. The participants should communicate with their colleagues to promote the automation project. However, the rest of the department may accuse the participants of betrayal, because of their participation in the project.

- Sometimes, the users do not have enough knowledge about automation to communicate with the developers. Then, the users are not able to steer the developers. In this case the developers will produce a system
on their own, without giving the users the opportunity to influence the development process.

- Sometimes, the users get 'carried away' with the technical and economical opportunities of the development. They take the role of developer, instead of user. Thus, the users do not represent their department.

### Development team and contractors

Usually a company does not develop its own design system from scratch. Sometimes, a design system is bought and implemented 'out of the box' with no further developments and adjustments. In most cases, however, a design system is bought and adjusted, tailored, or developed to some extent. It is possible to allow the designers to develop the system themselves. An advantage of this approach is that the designers know the design process and know the desired development best. A disadvantage is that the designers might not have enough expertise to develop a system, or do not have enough time available. Another option is to contract experts do develop the system. It is possible to determine two sub-options for contracting experts. The first option is a product based contract, that specifies the result of the development. An advantage of such a contract is a fixed price. A disadvantage is that it is very difficult to specify the result of the system development in detail at the beginning of the project. The second option, an effort based contract, is a solution for the difficulty in specifying the result. The development team of the company and the contractors form one team. The project manager instructs the software engineers of the contractors directly. A disadvantage is that there is no fixed price, and there are no guarantees for the result.

### 7.2 System architecture

Along with the role of the future users and the management in the development, the system architecture also has influence on the acceptance of the automation. First, this section generally considers the development and introduction of new products. Then it focuses on the development of design systems.

Deming [50] discusses the role of consumers and producers in product development: 'The producer is in a far better position than the consumer to invent new design and new service. Would anyone that
owned an automobile in 1905 express a desire for pneumatic tires, had you asked him what he needed? Would I, carrying a precise pocket watch, have suggested a tiny calculator and quartz time piece? Though the producer is in a better position to develop new products, the success of a product depends on the acceptance by the consumer. There are many examples of improved products that were not successful, because the consumer did not accept or buy them. An example is the Philips Video 2000 system, which was technically better than the VHS video system, but did not sell enough to stay on the market. The Video 2000 system is no longer available.

The same holds for design systems: a system developer is in a far better position than a designer to invent and improve a new design system, but the success of the system depends on the acceptance by the users. A technically advanced system is worthless if the users will not use it. Therefore, a system developer should work both on the system itself, and on the acceptance of the system. The acceptance of a design system will be higher if the users are counselled.

Therefore, the first step in design automation is the analysis of the existing process. This analysis involves the designers in the development process. A system analyst executes the analysis by interviewing experienced designers to determine the old design process. He describes the functions and data transfer moments in the design process. Occasionally, the designers can suggest improvements to the process.

After the process analysis, the system analyst can choose from two options to automate the design process. The first approach is a \textit{bottom up}, and the second a \textit{top down} approach.

\section*{Bottom up}

The bottom up approach automates existing parts of the design process. The system analyst and the designer choose which functions, and data transfer moments will be automated. The automation will result in an improved design process that strongly resembles the old process. The designers will prefer this approach, because it does not radically change the design process.

However, this type of approach does not lead to an optimal situation. It will be diffi-
cult for a designer to consider parts of the design process, that are not his tasks. Thus, the results are several separate automated sub-processes - islands - such as drafting, and structural analysis. The bottom up approach leads to island automation.

- **Top down**

The top down approach redesigns the design process. The system analyst considers the input and output of the design process, and designs a new, optimised process. This optimised process does not resemble the old design process. The designers may not approve of this approach, because it alters their jobs.

In the paragraphs above, Deming [50] suggests the top down approach. Although he places the initiative of new development on the producer, he also recognises the role of the consumer and user. Deming divides the development of new products into four steps: 1 design the new product, 2 build it, or make it, 3 sell it, or introduce it to the consumer, 4 evaluate why the user bought it and the nonuser did not, or test it in service. The last step shows the importance of the user and how the producer can improve his product or service.

Both the reliability analysis system, which is the subject of this thesis, and the structural analysis system were developed top down. The paragraphs below describe the experiences with the development and introduction of these systems. They show, that top down development can be both successful and unsuccessful.

### 7.3 Experiences: Structural analysis tool

This section reports about the development and implementation of a structural analysis tool. The project was technically successful, but the introduction failed, because the designers did not accept the system.
This section begins by explaining the objective of the development project. Then, the approach, results, and experiences of the project are discussed.

The development of the structural analysis tool started in 1993. The Mechanical Engineering Department of the Ministry of Transport, Public Works, and Water Management wanted to develop a design tool to analyse drive trains of structures such as bridges, lock gates, and storm surge barriers.

The drafting and analysis tasks of the department were strongly separated. Drafters defined the geometry of the structures while analysts checked the strength. A drafter first initiated a design. He made a drawing, and when it was finished, the analysts checked the strength of the structure. Often the drawing had to be altered to fulfil the demands in the calculation standards.

The objective of the development project was to optimise this process by combining the tasks drafting and analysing into designing. This new task had to be supported by a new tool, a design tool, instead of separate drafting and analysis tools.

The development team spent the first year on feasibility research of the modelling area of the design system. Several software packages for the calculation kernel were evaluated. The second year resulted in a prototype of the modeller. The modeller was developed as an application in the CAD package Dimension III of the design department. SPACAR, a finite elements package for spatial mechanisms from Delft university of Technology, was selected for the analysis kernel. Special finite elements, such as a gear box, and a rack and pinion, were developed. The modeller and the analysis kernel were integrated in the third year. A post processor was developed the fourth, and fifth year. The sixth year was spent on debugging, and user introduction of the system.

## Technical problems

During the development of the system the project team encountered four technical problems. Three of them were solved or are solvable with limited effort.

The first major technical problem encountered was the coupling of the modeller with the analysis kernel. The modeller and the analysis kernel were developed simultaneously. It is not possible to realise a coupling between two software packages that are being changed constantly.
The second major technical problem was the modelling of shafts with high revolution speed. The transmission ratio in a movable bridge is over one hundred. This results in high differences in rotation velocities of shafts. The shaft with the highest rotation velocity determines the integration step size. This lead to very large calculation times (several hours). The problem has been solved by separating the motion of the shafts in a geometrically linear, and a non linear part. The linear part of the motion is the rotation of the shaft around its axis. The non linear part of the motion are the changes of position and orientation in space. It is possible to increase the integration step size with a factor of one hundred with this method. Thus, the calculation times were reduced to several minutes.

**Figure 7.3-1:** Schematic drawing of a bascule bridge.

**Figure 7.3-2:** Component model of a bascule bridge, that was generated with the design system.

**Figure 7.3-3:** Load on the bridge as a function of the opening angle.

**Figure 7.3-4:** Output of the post processor of the design system.
The third technical problem was the generation of 2D geometry from the 3D model. The design tool supports the definition of the topology and geometry of the structure. The design is stored in a central data model. This model provides data for applications for structural analysis and calculation of costs. These functions have been successfully implemented. Furthermore, the model should be the foundation of 2D drawings. This functionality has not been realised, but implementation is within reach. Generation of 2D drawings from 3D geometry is possible with the current generation of solid modellers.

The fourth technical problem was the coupling of the UNIX and PC environment. The system was developed on two different software platforms. The input module is based on Dimension III, the current CAD system of the principal. Dimension III ran on VAX VMS, and now runs on a UNIX platform. The analysis software was developed on a PC platform. In the beginning of the project it was assumed, that a transparent coupling could be generated between the two platforms. In practice, however, the coupling is not transparent.

- Social problems

During the development of the system, the project team encountered four social problems, that appeared to be unsolvable.

The first social problem is the limited portability of the system. Besides the lack of transparency, the UNIX platform also hinders the distribution of the design system. The CAD users are not educated to judge the results of the structural analysis. Potential, higher educated users do not have direct access to a UNIX system. Unfortunately, they are PC users.

The second social problem is the acceptance by future users. The design system alters the design process. The system brings structural analysis within reach of people who did not make the calculations before. This was a threat for the group of analysts, who could not accept this.

Another change to the design process is the moment in which the drawings are made. In the old process, the drawings were made at the beginning of the design process. In the altered process they are drawn at the end. A drawing is just a projection of the model in the new process. The model is more important than the drawing, since it contains more information. However, this is not recognised by the drafters.

The third social problem is the insufficient representation of the future users in the project organisation. The user group contained both analysts and drafters. However, the calculator representative was not
able to convince his colleagues of the advantages of the design system. Instead, his colleagues blamed him for taking part in the user group. Furthermore, the representative drafter did not fully represent the group of future users, since the future users are designers, not drafters.

The system was tested several times by people with the future user profile. Despite satisfactory test results, the design system remained with a bad image, because of the commotion from the group of analysts. This bad image caused discussion about the desirability of the system in the management team of the design department. The bad image also withholds project managers from applying the design system in their projects.

The fourth social problem is the lack of support by the higher management. The development of the design system took six years. Despite the fact that the investments per year were relatively low, the overall development costs were reasonably high. Therefore, the higher management accused the management of the design department of spending too much money on the design system.

7.4 Experiences: Reliability analysis tool

This section reports about the development and implementation of the reliability analysis tool. Both technical implementation and the introduction with the designers were successful. This section describes the approach, results, and experiences of the project.

Just like the development of the structural analysis tool, the development of the reliability analysis tool started in 1993. The Mechanical Engineering department of the Ministry of Transport, Public Works, and Water Management received an increasing number of requests for reliability analysis of drive trains. Therefore, the department wanted to develop more knowledge on this subject. A development project developed software to implement this knowledge in the design process. The software supports the designer with the reliability analysis.

The software has been developed until the end of 1997, before the implementation trajectory started. During this development process the system was not shown to future users. Thus, the project did not suffer from criticism, until the design system had taken its final form. Consequently, at the beginning of the implementation trajectory the software could be demonstrated to future users with almost full functionality. This made it easy to communicate with the users about wishes and demands.
The portability of the design system has been very important during its development. The system has been migrated from DOS to Windows 3.11, and then to a Windows 95 and NT, and a HP UNIX platform. The environment of the system has changed more rapidly than the velocity of development.

An implementation project was started at the end of 1997. A member of the management team of the Mechanical Engineering department of the Ministry was the principal of the implementation project.

The steering group contained members of the Mechanical Engineering department, a research department, the Electrical Engineering department, and Delft University of Technology. Two project managers, two risk analysis experts, a design expert, and the principal formed the steering group.

The user group for the reliability analysis tool was selected with more care than the user group for the structural analysis tool. The group was small. There were only two members, both young, and one being a student. They were willing to accept changes in the design process, and to explore new technology, while giving constructive criticism, which was important to the improvement of the design system.

The field of reliability analysis is fairly unexplored in mechanical engineering design practice. Therefore, the reliability analysis tool does not change any existing work methods. This made the introduction of the system easier.

The system was introduced to a larger group of users in the end of 1998 by a risk analysis course. The course considered allowable probability of failure for a structure, failure modes and effects analysis, fault

Figure 7.4.1: Steering group meeting.
tree analysis, and event tree analysis, and finally introduced the software. The course was developed and taught by three lecturers. One of these lecturers - a member of the user group - introduced the design system. The response to the course was (positively) overwhelming. The steering group estimated, that 12 designers would be interested in the course, but finally 25 people attended the course. Therefore, larger accommodations had to be arranged, and several sessions of the course were taught twice, since even the larger accommodations appeared to be too small. All participants wanted to learn to work with the software, and successfully used it during the course. The paragraphs below show the evaluation results of the course:

**Functional design, average evaluation results for day 1**

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Did you have foreknowledge about this subject?</td>
<td>yes</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Did you hear about this subject before this course?</td>
<td>yes</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Did you study the course manual before the course?</td>
<td>yes</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>If so, is the course manual clear?</td>
<td>yes</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Did the lecturer speak loud enough?</td>
<td>yes</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Did the lecturer explain the subject-matter clearly?</td>
<td>yes</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Was this session of the course difficult?</td>
<td>yes</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Was the subject-matter of this session comprehensible?</td>
<td>yes</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Did you get a better understanding of functional design?</td>
<td>yes</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Did this part of the course fulfil your expectations?</td>
<td>yes</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Are you satisfied with the accommodations?</td>
<td>yes</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Would you recommend this session to a colleague?</td>
<td>yes</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### FMEA, FTA, and event trees, average evaluation results for day 2 and 3

<table>
<thead>
<tr>
<th>Question</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Did you have foreknowledge about this subject?</td>
<td>☐</td>
<td>☒</td>
</tr>
<tr>
<td>Did you hear about this subject before this course?</td>
<td>☐</td>
<td>☒</td>
</tr>
<tr>
<td>Did you study the course manual before the course?</td>
<td>☒</td>
<td>☐</td>
</tr>
<tr>
<td>Did the lecturer speak loud enough?</td>
<td>☒</td>
<td>☐</td>
</tr>
<tr>
<td>Did the lecturer explain the subject-matter clearly?</td>
<td>☐</td>
<td>☒</td>
</tr>
<tr>
<td>Was this session of the course difficult?</td>
<td>☒</td>
<td>☐</td>
</tr>
<tr>
<td>Was the subject-matter of this session comprehensible?</td>
<td>☒</td>
<td>☐</td>
</tr>
<tr>
<td>Was the homework for day 2 and 3 too much?</td>
<td>☒</td>
<td>☐</td>
</tr>
<tr>
<td>Was the homework for day 2 and 3 difficult?</td>
<td>☒</td>
<td>☐</td>
</tr>
<tr>
<td>Did this session of the course fulfil your expectations?</td>
<td>☒</td>
<td>☐</td>
</tr>
<tr>
<td>Can you apply the subject-matter in your job?</td>
<td>☒</td>
<td>☐</td>
</tr>
<tr>
<td>Would you recommend this session to a colleague?</td>
<td>☒</td>
<td>☐</td>
</tr>
<tr>
<td>Was the homework for day 1 too much?</td>
<td>☒</td>
<td>☐</td>
</tr>
<tr>
<td>Was the homework for day 1 difficult?</td>
<td>☒</td>
<td>☐</td>
</tr>
</tbody>
</table>

### Reliability analysis software, average evaluation results for day 4 and 5

<table>
<thead>
<tr>
<th>Question</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do you have experience using software programs?</td>
<td>☒</td>
<td>☐</td>
</tr>
<tr>
<td>Did you use the reliability analysis software before?</td>
<td>☒</td>
<td>☐</td>
</tr>
<tr>
<td>Did you study the course manual before the course?</td>
<td>☒</td>
<td>☐</td>
</tr>
<tr>
<td>Did the lecturer speak loud enough?</td>
<td>☒</td>
<td>☐</td>
</tr>
<tr>
<td>Did the lecturer explain the subject-matter clearly?</td>
<td>☒</td>
<td>☐</td>
</tr>
<tr>
<td>Is the menu lay out of the software user friendly?</td>
<td>☒</td>
<td>☐</td>
</tr>
<tr>
<td>Is the software in the current state usable?</td>
<td>☒</td>
<td>☐</td>
</tr>
<tr>
<td>Were you capable of completing the homework yourself?</td>
<td>☒</td>
<td>☐</td>
</tr>
<tr>
<td>Was the homework for day 4 and 5 too much?</td>
<td>☒</td>
<td>☐</td>
</tr>
<tr>
<td>Was the homework for day 4 and 5 difficult?</td>
<td>☒</td>
<td>☐</td>
</tr>
<tr>
<td>Did this session of the course fulfil your expectations?</td>
<td>☒</td>
<td>☐</td>
</tr>
<tr>
<td>Could you apply the software in your job?</td>
<td>☒</td>
<td>☐</td>
</tr>
<tr>
<td>Would you apply the software in your job?</td>
<td>☒</td>
<td>☐</td>
</tr>
<tr>
<td>Would you recommend this session to a colleague?</td>
<td>☒</td>
<td>☐</td>
</tr>
</tbody>
</table>
The evaluation results show that the designers, who attended the
course, found the subject-matter comprehensible and the explanation of
the lecturer clear. They claim to be able to apply the subject-matter in
their jobs, and even would recommend the course to a colleague. The
same applies for the software: they also claim to be able to use the
software in their job.

**Software development and maintenance**

The project team contained the author of this thesis, a student, and a
system developer. During 1997, '98, and '99, the project team develop-
oped the software to make it applicable in the design process. The
changes in the software were relatively large. The software was not used
in the design process during this period. By the end of 1999, the devel-
opment was finished, and the software had to be implemented in the
design process. After this implementation the user support and soft-
ware maintenance have to be arranged. A project is not a suitable or-
ganisation for this type of work.

The project team for the reliability analysis software was a typical
development organisation; it realised a brand new product, it did not
have concerns about support for customers or users, and it did not have
conscerns about configuration management. The project team worked
according to a project plan, that defined the desired development, and
that was agreed by the principal and the steering group. Since the soft-
ware was not operational in the design process, the team did not have
to worry about support for users, configuration management, valida-
tion, and new releases.

Software in an operational environment requires a different type of
organisation. It requires an organisation that is able to support users,
take care or configuration management, and validate new versions of
the software before it is released. As mentioned above, project organisa-
tion is not suitable for this type of work. Project Consult [58] states,
that a project has the following properties:

- A project is finite. The project is finished after the final result has been
  accomplished.
- The result of a project is a discrete product, such as a bridge, a report,
or a CD-rom with software.
- A project organisation is temporary. After the project is finished, the
  project organisation splits up. Therefore, a department or other regular
  organisation that creates discrete products is not a project.
Maintenance and user support are a continuous task. Thus, a temporary organisation is not satisfactory. Furthermore, support and maintenance are not discrete products. This demonstrates that a project is not a suitable organisation for these tasks.

Maintenance and support require a permanent organisation, that is not plan driven, but event and procedure driven. A project is a plan driven organisation, that focuses on finishing a job described in a plan. A procedure driven organisation focuses on events, and responds to these events according to procedures. The world wide adapted ITIL [62], [65], [66], [67] concept describes such an event driven organisation for software maintenance and support.

ITIL describes how to handle small problems (incidents), larger problems, changes to the system, and the management of the process. Incidents are simple, small problems, such as user support, that can be solved at once. This section focuses on larger problems: all problems that can not be solved at once, such as removal of bugs, requests for changes in the user interface, and changes in failure data. These changes are too small to make it worthwhile to write a project plan. However, they can have a significant influence on the results of calculations from the software. Hence, for these problems a special procedure was agreed.

Changes are proposed to a problem resolving team. If the changes concern the algorithms, a software development expert gives advise. If the changes concern failure data, a risk analysis expert gives advise. The chairman of the problem resolving team decides whether the changes will be accepted. Then, the development team implements, and tests the changes. The user group also tests the changed software. Finally, the adjusted software is submitted to the maintenance team for technical

Figure 7.4-4: Process to implement changes into the design software.
analysis software, which supplies the software to the users via the network.

7.5 Conclusion

Social aspects have a major influence on the success of the introduction of a design system. The introduction of the structural analysis tool failed, while the introduction of the reliability analysis tool was successful, although both are based on the same system architecture. The failure of the introduction of the structural analysis tool is not caused by technical problems. Social issues, such as an altered design process, the choice of the participants in the user group, and the role of the management caused the failure. Below, the causes of success with the implementation of the reliability analysis tool are described.

It is wise not to demonstrate software, that does not have a reasonable functionality, to future users. It is difficult to explain the functionality of a design system just in words. It is necessary to demonstrate a system to make people enthusiastic. Therefore, develop a system in silence, and only demonstrate it, when it is ready.

Software that is developed bottom up will be accepted easier, than software that is developed top down. However, the reliability analysis tool is an example, that it is possible to successfully introduce software, that is developed top down. A factor for success is, that the reliability analysis tool does not change any existing work methods. This made the introduction of the system easier.

The response to the reliability analysis course shows, that the users accepted the reliability analysis software. Apparently, there is a need for support of reliability analysis in the design process*. It also shows, that the software helps the designer in executing reliability analysis himself.

A technical aspect: it is easier to introduce a system that runs on one, wide spread platform. The structural analysis tool runs on a UNIX and a PC platform. This made it more difficult to introduce the system, since the future users do not have direct access to a UNIX system.

*) This is a circular argument, since this research project started because of the need for support of reliability analysis in the design process. The result of the project can not be one of the starting points.
8 Conclusion

This chapter answers the three research questions from section 1.2. To answer these questions a design and reliability analysis method was developed, and implemented in software. Chapter 2 gives an overview of the method and the software. Chapter 3 to 5 describe the method in detail. Chapter 6 qualifies, verifies, and evaluates the method and software. The software was introduced in a design environment. Chapter 7 describes the implementation of the design system in the design environment.

All these activities contribute to the answer of the research questions. The paragraphs below answer these questions as much as possible. The answers are built up from the conclusions in the previous chapters.

- Can the reliability process be automated, and does the automated method produce satisfying results?

Chapter 2 shows, that it is possible to automate reliability analysis, and to integrate the automated method into the design process. Fault tree analysis appears to be the most suitable technique to use in the design process. This technique analyses the reliability of one function of a structure.

It is possible to give an abstract description of the most important functions of a drive train. All major functions of a drive train can be decomposed into the basic functions carrying load, and executing motion. The finite element theory describes these functions. The minimal cut sets of a fault tree can be derived directly from the finite elements model.

Chapter 3 discusses a finite element description of the flow of fluids, which is necessary to model all important components in drive trains. The classic finite elements theory does not describe the flow of fluids. Therefore, the finite element theory was extended. The equations for solid materials, and for fluids can be solved simultaneously.
Chapter 4 describes the successful implementation of the theory from chapter 2 and 3 into software. The software consists of a component modeller, that helps the designer create different design solutions, and a finite element program, that analyses the solutions. An open software architecture makes it possible to alter the implementation of components and elements. Various algorithms were implemented. Some algorithms should produce equal outputs. It is possible to validate these algorithms by comparing their results.

Chapter 5 discusses two models used to quantify the probability of failure. The first model, failure per demand, describes failure of discontinuous processes. The second model, the failure rate model, describes failure of continuous processes. However, the division into continuous and discontinuous processes is not accurate enough. Continuous processes can be divided into rest and action, and discontinuous processes into start and stop. In each of these sub-processes different failure mechanisms take place. Thus, it is necessary to apply different failure data in each sub-process.

Chapter 6 demonstrates, that the automated method produces valid results. It gives the analysis results of various structures, which can be verified manually. Furthermore, it refers to the analysis of complex structures. An improved automated reliability analysis method also produces valid results for these structures.

Can the automated method be integrated into the conceptual design process, and can the designer execute the reliability analysis himself?

Chapter 2 describes the software package, that embeds the theory, and helps a designer in creating different design solutions. The software produces comprehensive results, and helps the designer in improving the structure, and choosing the most optimal solution. Though the automated method supports the designer in such a way, that he can execute the reliability analysis himself, it is still wise to consult a risk analyst for a final judgement of the results.

Chapter 7 shows, that social aspects have a major influence on the success of the introduction of a design system. The introduction of a strength analysis tool, and of a reliability analysis tool is also discussed. The introduction of the strength analysis tool failed, while the introduction of the reliability analysis tool was successful, although both are based on the same system architecture. The failure of the introduction of the strength analysis tool is not caused by technical problems. Social issues, such as an altered design process, the choice of the participants in the user group, and the role of the management caused the failure. Be-
low, the causes of success with the implementation of the reliability analysis tool are described.

It is wise not to demonstrate software, that does not have a reasonable functionality, to future users. It is difficult to explain the functionality of a design system just in words. It is necessary to demonstrate a system to make people enthusiastic. Therefore, develop a system in silence, and only demonstrate it, when it is ready.

Software that is developed **bottom up** will be accepted easier, than software that is developed **top down**. However, it is possible to successfully introduce software, that is developed top down. A factor for success is, that the reliability analysis tool does not change any existing work methods. This makes the introduction of the system easier.

The training of future users can ease the introduction of a design system. The response to the reliability analysis course demonstrates, that the users accepted the reliability analysis software. Apparently, there is a need for support of reliability analysis in the design process. It also shows, that the software enables the designer to execute reliability analysis himself.

A technical aspect: it is easier to introduce a system that runs on one, wide spread platform. The strength analysis tool runs on an UNIX and a PC platform. This made it more difficult to introduce the system, since the future users do not have direct access to an UNIX system.

- Does the integrated reliability analysis improve the designs?

To answer this question another question rises: What is a better design? Is a better design a more reliable design? Is it a less expensive design? Is it a structure designed with a more controlled design method? Is the acceptance of a design method important to the quality of the designs? Chapters 2, 6, and 7 of this thesis answer these questions to some extent.

Chapter 2 shows, that it is possible to improve a design by applying reliability analysis. It demonstrates how to quantify failure and unavailability in costs, and how to compare them with investment costs to improve the reliability. A better design is not necessarily a design with a higher reliability, but a design with optimal life cycle and failure costs.

Chapter 6 shows that application of the software would lead to improvements in the reliability analysis methods of structures, and in the structures themselves. The changes to structures would not lead to more reliable structures, but to less expensive structures with an equal reliability.

Conclusion
A valid automated reliability analysis method produces more reliable results, than a manual method. The automated method does not make mistakes, and the approach is equal for all problems, independent of the designer. Thus, the results of reliability analyses for different structures are better comparable.

However, it is not possible to fully answer this question, since the reliability analysis tool has not been applied in the design practice for a long period. Therefore, it is necessary to monitor the application of the tool over a longer period of time. It is then possible to see whether newly designed structures change and designs are improved due to the application of the reliability analysis tool. The results of chapter 6 are promising while chapter 7 shows that the system is accepted by the future users. Therefore, the expectations are optimistic.

- **Recommendations for further research**

The results of a reliability analysis strongly depend on the accuracy of the reliability data that are used. The appendices of this thesis contain a list of reliability data form various sources. This list shows that the deviation within the failure data of one failure mode of one single component is sometimes a factor 100. Chapter 5 discusses a method to take this uncertainty in data into account. However, if this method is applied, but different data is used in various reliability analyses, it is still not possible to compare the results. Therefore, it is necessary to develop standards for the data and the deviation in data that should be used.

Chapter 3 combines the mechanical and hydraulic domain into a multi domain system. The mechanical domain describes the behaviour of mechanical components, while the hydraulic domain describes the behaviour of hydraulic components in drive trains. It is possible to describe the simple flow of gasses with the hydraulic domain. Further research is necessary to verify whether other domains, such as the heat domain, can be described with a similar method. The combination of the mechanical, hydraulic, and heat domain would open the possibility to model a steam engine or a internal combustion engine.
9 References


[3] L.A.G. Wagemans; The incalculable constructive design / Het onberekenbare constructief ontwerp (inaugural speech, in Dutch); Delft University of Technology, Civil Engineering Faculty; February 1996

[4] G.C. Avontuur; Reliability analysis in mechanical engineering design; Ministry of Transport, Public Works, and Water Management, RIJKSWATERSTAAT, Engineering and Construction Division, Mechanical Engineering Department; Delft University of Technology, Faculty of Mechanical Engineering and Marine Technology, Section Design; April 1995


[6] B.H. Kiestra; Reliability analysis in the design process of drive trains / Betrouwbaarheidsanalyse in het ontwerpproces van bewegingswerken (in Dutch); Delft University of Technology, Mechanical Engineering and Marine Technology Faculty, Section Design; 1998

[7] B.H. Kiestra; Component book / Componentenboek (in Dutch); Delft University of Technology, Mechanical Engineering and Marine Technology Faculty, Section Design; 1998

[8] R.C.A. Beem, G.C. Avontuur, L.M.A. Claes; Risk analysis for mechanical engineers / Cursusdiktaat Risico analyse voor werktuigbouwkundige ontwerpers (in Dutch); Ministry of Transport, Public Works, and Water Management, RIJKSWATERSTAAT, Engineering and
Construction Division, Research, Design Informatics, and Mechanical Engineering Department; 1998

[9] J.J.M. van Geijlswijk; *Risk analysis / Risico analyse* (in Dutch); Delft University of Technology, Mechanical Engineering and Marine Technology Faculty, Section Design; 1998


[14] J.L. Spoormaker; *Reliability-based design / Ontwerpen op bedrijfszekerheid* (in Dutch); Delft University of Technology; Division Industrial Design; Section Construction; The Netherlands; 1989


[16] H. Lammerse; *Reliability data / Bedrijfszekerheidsegevens* (in Dutch); Kluwer; 1993

[17] K. van der Werff; *Kinematic and dynamic analysis of mechanisms, a finite element approach*; Delft University Press; 1977


[20] S.J. de Lange; *Probabilistic analysis and statistics / Kansrekening en statistiek* (in Dutch); Delftse Uitgeversmaatschappij; 1988


[23] P.J. van Gestel; *Reliability data / Cursusdiktaat Bedrijfszekerheidsgegevens*; KEMA; 1994
[24] Baseline storm surge barriers and special structures / Basisrapport waterkerende kunstwerken en bijzondere constructies (in Dutch); TAW; 1997

[25] Handbook storm surge barriers and special structures / Leidraad waterkerende kunstwerken en bijzondere constructies (in Dutch); TAW; 1997

[26] H.C. Wels; Probabilistic data of hydraulic components / Faaldata hydraulische componenten (in Dutch); KEMA; 1992

[27] Reliability analysis of the lock doors of 'the Krammer locks' / Betrouwbaarheidsanalyse van de sluisdeuren van de Krammersluizen (in Dutch); KEMA; 1985

[28] G. Kocza; Models and algorithms for automatic reliability assessment of complex systems; Delft University Press; 1997

[29] T. Bedford, R.M. Cooke; Mathematical tools for probabilistic risk analysis; Delft University of Technology, Department of Mathematics; 1998

[30] R.M. Cooke; Unicorn r3.0, Unicorn: methods and code for uncertainty analysis; Delft University of Technology; 1995

[31] E. Schrurer; Zuverlässigkeit von mes- und Automatisierungs- einrichtungen; Carl Hanser Verlag München Wien; 1984


[32b] Eugene A. O'Hern; Space Shuttle main propulsion system fault tolerance model; Rockwell International, Space Systems Division, Downey, Ca. 90241; Elsevier Science Publishing Co., Inc.; Probabilistic Safety Assessment and Management; 1991


[34] Takeshi Matsuoka, Michiyuki Kobayashi; Go-flow: A system reliability analysis methodology; System Engineering Division, Ship Research Institute, Ministry of Transport, 6-38-1, Shinkawa, Mitaka, Tokyo, 181 Japan; Elsevier Science Publishing Co., Inc.; Probabilistic Safety Assessment and Management; 1991

[35] Hajime Yamashina, Hiromitsu Kumamoto, Susumu Okumura; Plant failure diagnosis by an automated fault tree construction consistent with boundary conditions; Department of Precision Mechanics, Faculty of Engineering, Kyoto University, Kyoto 606, Japan; Elsevier Science
[36] Takehisa Kohda, Koichi Inoue; *A petri net approach to probabilistic safety assessment for obtaining event sequences from component models*; Department of Aeronautical Engineering, Faculty of Engineering, Kyoto University, Kyoto 606, Japan; Elsevier Science Publishing Co., Inc.; Probabilistic Safety Assessment and Management; 1991

[37] Hideaki Takahashi, Yutaka Kitamura, Ako Takahashi; *Development of expert systems to support system reliability analysis*; Mitsubishi Research Institute Inc., Time&Life Bldg., 3-6, Otemachi 2-Chome, Chiyoda-ku, Tokyo 100, Japan; Elsevier Science Publishing Co., Inc.; Probabilistic Safety Assessment and Management; 1991

[38] Richard E. Barlow, Frank Proschan; *Importance of system components and fault tree events*; North Holland Publishing Company; Stochastic Processes and their Applications 3 (1975) 153-173


[40] Anton van Asch, Toon Jansen; *To an improvement of the participation opportunities of company advisory boards with automation / Naar een verbetering van de participatiemogelijkheden van ondernemingsraden bij automatisering* (in Dutch); Ministry of Education, and Science; Labour and new technology foundation; Amsterdam; 1986

[41] J.K. Vrijling; *Risk and safety / Risico en veiligheid* (in Dutch); Delft University of Technology, Civil Engineering Faculty; 1980


[43] J.K. Vrijling, J Stoop; *Water magicians, Delft's ideas for another 200 years of Rijkswaterstaat, Ch. 17: Rijkswaterstaat and probabilistic calculations / Watertovenaren, Delfse ideeën voor nog 200 jaar Rijkswaterstaat, H 17: Rijkswaterstaat en de kansberekening* (in Dutch); Bêta Imaginations Publishers; 1998


[46] Barry Kirwan; *A guide to practical human reliability assessment*; Taylor & Francis; 1994

[47] Ministry of Housing, Land use planning, and Environment / Ministerie van VROM; *Relating to risks / Omgaan met risico's* (in Dutch); The Hague / Den Haag; 1992

[48] Nuclear Regulatory Commission; *Reactor safety study*; WASH-1400, NUREG - 751014; 1975

[50] W. Edwards Deming; *Out of the crisis, quality productivity and competitive position; Cambridge University Press; 1992

[51] John Stark; *Managing CAD/CAM, implementation, organisation, and integration; McGraw-Hill Book company; 1988

[52] J.G.V. Maas and J.J.G. Bollen; *Quality for development and design / Kwaliteit bij ontwikkeling en ontwerp (in Dutch); Kluwer; 1992

[53] G.C. Avontuur; *Simplification of the hydraulic unit of the drive train of the storm surge barrier in the Oosterschelde, analysis with FEMFault / Vereenvoudiging van de hydraulische installatie van de bewegingswerken van de Stormvloedkering in de Oosterschelde, analyse met FEMFault (in Dutch); Ministry of Transport, Public Works, and Water Management, Rijkswaterstaat, Engineering and Construction Division, Mechanical Engineering Department; Delft University of Technology, Faculty of Mechanical Engineering and Marine Technology, Section Design; December 9th 1998

[54] Edward J. Preston, George W. Crawford, Mark E. Coticchia; *CAD/CAM systems, justification implementation productivity measurement; Marcel Dekker, Inc., New York and Basel; 1984

[55] Sheleyyahu Zacks; *Introduction to reliability analysis, probability models and statistical methods; Springer Verlag; 1992


[57] Igor A. Ushakov, Robert A. Harrison; *Handbook of reliability engineering; John Wiley & Sons, Inc.; 1994

[58] Project Consult; *Course project management / diktaat Cursus projectmanagement (in Dutch); Amersfoort; 1995

[59] C.W. Gits; *On the maintenance concept for a technical system, a framework for design; Dissertation drukkerij Wibro, Helmond; 1984

[60] J.K. Vrijling et al.; *Probabilities in civil engineering, theory of probabilistic design / Kansen in de civiele techniek, deel 1 Probabilistisch ontwerpen in theorie (in Dutch); CUR, Civielttechnisch Centrum Uitvoering Research en Regelgeving; 1997


[62] CCTA; *The IT Infrastructure Library, An introduction; HMSO Publications Centre; London; 1990

[63] J. Berenbak; *The methodical development of the design of the Maeslantkering, the storm surge barrier in the New Waterway / De methodische ontwikkeling van het ontwerp van de Maeslantkering, de stormstuw in de

References 153
Nieuwe Waterweg (in Dutch); seminar Ontwerpmethodologie / seminar Design Methodology; Delft University of Technology, Faculty of Architecture; may 28th and 29th 1998


[65] CCTA; The IT Infrastructure Library, Configuration management; HMSO Publications Centre; London; 1990

[66] CCTA; The IT Infrastructure Library, Problem management; HMSO Publications Centre; London; 1990

[67] CCTA; The IT Infrastructure Library, Software control and distribution; HMSO Publications Centre; London; 1990


[69] Isograph Ltd., Television House, 10 Mount St, M2 5NT, United Kingdom; Software package FaultTree + Version 8.0; 1999

[70] A.M. Rankers, K. van der Werff; Stress and vibration analysis of mechanical transmissions based on the finite element method; The Theory of Machines and Mechanisms, 7th World Congress, Vol. 2, p. 737 - 740; Sevilla, Spain; 17 - 22 September 1987
A Finite element program

This appendix gives a more detailed description of the finite elements program that is described in chapter 4. The first section describes the data structures of the program. The second section describes some of the software algorithms. The last third, fourth, and fifth section describe the syntax of the recipe, model, and initialisation files, and the last section describes the commands.

A.1 Data structures

This section describes the data structures recipe, element, node, bound, load, and stress.

A.1.1 Definition of recipe

Recipes are stored in the global variable recipe list. The data in the structure recipe defines an element type. All fields or records of the recipe structure are explained in this chapter. Sometimes the names of the fields are slightly changed, because the original names in the source file are to cryptically.

- name

The name of the recipe is an element type. The number of characters in this field is ELEMENT_NAME_SIZE + 1 (NULL byte for end of string marker included). ELEMENT_NAME_SIZE is defined to be 8 positions. Example: ‘truss’, ‘beam’.
- **domain**
  The name of the domain of the recipe. At the moment the mechanical and hydraulic domain are implemented. Possible values: 'mech', 'hydr'.

- **number of nodes (original name: amountnodes)**
  The number (amount) of nodes of an element of type <name>. Possible values: 0, 1, 2, 3, ..., etc.

- **degrees of freedom (original name: dof)**
  The total number of degrees of freedom: the sum of the degrees of freedom of all nodes. Possible values: 1, 2, 3, ..., etc.

- **degrees of freedom per node (original name: dofpnode)**
  An array with the number of degrees of freedom of each single node. Possible indices of the array: 0, 1, 2, ..., (number of nodes - 1). *degrees of freedom per node[0]* is the number of degrees of freedom of the first node, *degrees of freedom per node[1]* is the number of degrees of freedom of the second node.
  Possible values for *degrees of freedom per node[i]*: 0, 1, 2, ..., etc. The sum of all *degrees of freedom per node[i]* is equal to the value of *degrees of freedom*.

- **deformations (original name: def)**
  Number of deformations. Possible values: 1, 2, ..., etc.

- **set of equations (original name: eq)**
  Pointer to a set of equations. The entries of the element deformation, stiffness and transformation matrix are calculated by evaluating the equations. \( D[i][j] \) is an entry of the local element deformation matrix, \( S[i][j] \) is an entry of the element stiffness matrix, and \( T[i][j] \) is an entry of the nodal transformation matrix. Entries of the inverted orientation matrix \( oriinv[i][j] \) are available for use in the transformation matrix. The coordinates of the nodes \( x[n] \), \( y[n] \), \( z[n] \) are also available.

- **D**
  Pointer to the element deformation matrix in the local coordinate system. Equation (A.1-1) defines this matrix for the mechanical and (A.1-2) for the hydraulic domain.
\[ \varepsilon = D_{\text{loc}} \psi_{\text{loc}} \]  \hspace{1cm} (A.1-1)

\( \varepsilon \) is a vector with element deformations, \( \psi_{\text{loc}} \) is a vector with nodal displacements in the local coordinate system. The size of \( D_{\text{loc}} \) in the equation above is: rows = deformations, and columns = degrees of freedom

\[ \psi_{\text{loc}} = D_{\text{loc}} \phi \]  \hspace{1cm} (A.1-2)

\( \psi_{\text{loc}} \) is a vector with local nodal flows into the element. \( \phi \) is a vector with element deformations. The size of \( D_{\text{loc}} \) in the equation above is: rows = degrees of freedom, and columns = deformations.

**D**

Pointer to the element deformation matrix in the global coordinate system. In the mechanical domain:

\[ \varepsilon = D_{\text{glob}} \phi_{\text{glob}} \]  \hspace{1cm} (A.1-3)

\( \phi_{\text{glob}} \) is a vector with nodal displacements in the global coordinate system. See equation (A.1-5) for definition of \( D_{\text{glob}} \).

The local and global deformation matrix in the hydraulic domain are equal, because the degrees of freedom and deformations in the hydraulic domain are not orientation dependent.

The dimensions of global deformation matrix are equal to the dimensions of the local matrix for both domains.

**S**

Pointer to the element stiffness matrix. Size: number of rows = number of columns = number of deformations.

**T**

Pointer to the nodal transformation matrix. \( T \) transforms all degrees of freedom of an element from the global to the local coordinate system. In the mechanical domain:

\[ u_{\text{loc}} = T u_{\text{glob}} \]  \hspace{1cm} (A.1-4)

Note: \( u_{\text{loc}} \) and \( u_{\text{glob}} \) are vectors with the degrees of freedom of all nodes of an element. Substitution of equation (A.1-4) in (A.1-1) combined with (A.1-3) results in:

\[ D_{\text{glob}} = D_{\text{loc}} T \]  \hspace{1cm} (A.1-5)

Usually, in the hydraulic domain the transformation matrix is equal to the unit matrix: \( T = I \). See also chapters \( D_{\text{loc}} \) and \( D_{\text{glob}} \).
The size of $T$: number of rows = number of columns = degrees of freedom.

- **next**

The global variable recipe list is implemented as a linked list of recipes. `next` is a pointer to the next recipe in a list of recipes or NULL if the recipe is the last recipe in the list.

**Figure A.1.1:** Nodal displacements that are expressed in the global coordinate system $xyz$ can be transformed to the local coordinate system $uvw$ with matrix $T$. This matrix can contain entries of the inverse of the orientation matrix $[u, v, w]$. $r$

### A.1.2 Definition of element

Elements are stored in the global variable element list. All fields or records of the element data structure are explained in this subsection. Sometimes the names of the fields are slightly changed, because the original names in the source file are too cryptically.

- **number**

Each element has a unique number. Possible values: 0, 1, 2, ..., etc.

- **name**

The name of an element type (recipe). The number of characters in this field is `ELEMENT_NAME_SIZE` + 1 (NULL byte for end of string marker included). `ELEMENT_NAME_SIZE` is defined to be 8 positions. Example: ‘truss’, ‘beam’.

- **node numbers (original name: nodenrs)**

Array of node numbers. The size of the array is defined by the field `number of nodes` of the corresponding recipe. Possible values: 0, 1, 2, ..., etc.

- **deformation numbers (original name: defnrs)**

Array of deformation numbers. The size of the array is defined by the field `deformations` of the corresponding recipe. Possible values: 1, 2, 3, ..., etc.
- **orientation matrix (original name: orient)**
  
  Pointer to a $3 \times 3$ matrix with element orientation. See figure A.1-1.

- **inverse of orientation matrix (original name: oriinv)**
  
  Pointer to a $3 \times 3$ matrix with inverse of orientation matrix. See figure A.1-5, chapter A.1.1 - set of equations, chapter A.1.1 - T.

- **list of parameters (original name: par)**
  
  Pointer to a (list of) parameters. The parameters are substituted in the equations of the corresponding recipe when they are evaluated. See figure A.1-2.

- **next**

  The global variable element list is implemented as a linked list of elements. next is a pointer to the next element in a list of elements or NULL if the element is the last element in the list.

### A.1.3 Definition of node

Nodes are stored in the global variable node list. All fields or records of the node data structure are explained in this subsection. Sometimes the names of the fields are slightly changed, because the original names in the source file are to criptical.

- **number**

  Each node has a unique number. Possible values: 0, 1, 2, ..., etc.

- **domain**

  The name of the domain of the node. At the moment the mechanical and hydraulic domain are implemented. Possible values: ‘mech’, ‘hydr’.

- **x, y, z**

  Coordinates of the node in global coordinate system. Example: $x = 4.311 \times 10^2$, $y = 1.001 \times 10^4$, $z = 9.588 \times 10^3$.

- **number of degrees of freedom (original name: dof)**

  The number of degrees of freedom of the node. Possible values: 1, 2, 3, ..., etc.
degree of freedom numbers (original name: dofnrs)

Array with numbers of degrees of freedom. Size of the array: number of degrees of freedom.

next

The global variable node list is implemented as a linked list of nodes. next is a pointer to the next node in a list of nodes or NULL if the node is the last node in the list.

A.1.4 Definition of bound, load, and stress

Nodal displacements and pressures can be prescribed by boundary conditions. Nodal forces and bending moments can be prescribed using the data structure load. To prescribe element stresses and pressures the data structure stress can be used.

Bounds, loads, and stresses are stored in linked lists called bound list, load list, and stress list. The bound, load, and stress data structure are very similar. All data structures have almost identical records or fields and field names. The data structures are described in this chapter.

For bound and load: node number (original name: nodenr)

The number of the node to which the bound or load is applied. Possible values: 0, 1, 2, ..., etc.

For stress: element number (original name: elementnr)

The number of the element to which the or stress is applied. Possible values: 0, 1, 2, ..., etc.

For bound and load: degree of freedom (original name: dof)

'Local' degree of freedom number to which the bound or load is applied. Index of degree of freedom number array of data structure node. Possible values: 1, 2, 3, ..., number of degrees of freedom of node.

For stress: deformation (original name: def)

'Local' deformation number to which the stress is applied. Index of deformation number array of data structure element. Possible values: 1, 2, 3, ..., deformations of corresponding recipe.
value

Prescribed value of bound, load, or stress.
- Use a bound to prescribe a nodal displacement.
- Use a bound to prescribe a nodal pressure.
- Use a load to prescribe a nodal force.
- Use a load to prescribe a nodal bending moment.
- Use a stress to prescribe an element stress.
- Use a stress to prescribe an element pressure.

next

Bound, load, and stress list are implemented as linked lists. next is a pointer to the next bound, load, or stress in the list. If no next entry exists, next is NULL.

A.2 Algorithms

In this paragraph describes the algorithms for calculating deformation and degree of freedom numbers, and algorithms for building matrices are described. No algorithms for solving equations are described.
A.2.1 Main program

read model
scale model
calculate deformation numbers for mechanical and hydraulic domain
calculate degree of freedom numbers for mechanical and hydraulic domain
renumber
build deformation matrix for mechanical domain $D_{\text{mech}}$ and for hydraulic domain $D_{\text{hydr}}$
build connection matrix $C$  
This matrix connects the mechanical and hydraulic domain.
build stiffness matrix for mechanical domain $S_{\text{mech}}$ and for hydraulic domain $S_{\text{hydr}}$
solve equations using $D_{\text{mech}}$, $D_{\text{hydr}}$, $C$, $S_{\text{mech}}$, and $S_{\text{hydr}}$
unscale model  
This algorithm also unscales the results of ‘solve equation’.
write output

A.2.2 Calculate deformation numbers

First, this algorithm is used to calculate the deformation numbers of the elements in the mechanical domain, then it is used to calculate the numbers for the hydraulic domain. To calculate deformation numbers in the hydraulic domain the variable ‘number of deformations of model’ is not set to 0.
number of deformations of model = 0
get first element

while an element was found

    get corresponding recipe
    i = 0

    while i < number of deformations of recipe

        number of deformations of model = number of deformations of model + 1
        deformation numbers[i] of element = number of deformations of model
        i = i + 1

    get next element

A.2.3 Calculate degree of freedom numbers

First, this algorithm is used to calculate the degree of freedom numbers of the nodes in the mechanical domain, then it is used to calculate the numbers for the hydraulic domain. To calculate degree of freedom numbers in the hydraulic domain the variable ‘number of degrees of freedom of model’ is not set to 0.

The number of degrees of freedom per node is calculated for each node:
get first element

while an element was found

  get corresponding recipe
  i = 0

  while i < number of nodes of recipe

    get node with number node numbers[i] of element
    domain of node = domain of recipe

    if number of degrees of freedom of the node < number of degrees of freedom per node[i] of the recipe

      number of degrees of freedom of the node = number of degrees of freedom per node[i] of the recipe

    i = i + 1

get next element

The numbers of the degrees of freedom are calculated:

number of degrees of freedom of model = 0

get first node

while a node was found

  allocate memory for array of degree of freedom numbers of the node

  while i < number of degrees of freedom of the node

    number of degrees of freedom of model = number of degrees of freedom of model + 1
    degree of freedom numbers[i] of the node = number of degrees of freedom of model

    i = i + 1

get next node
A.2.4 Renumber

Renumber is used after deformations and degrees of freedom numbers are calculated. Renumber gives prescribed deformations and degrees of freedom the highest and free deformations and degrees of freedom the lowest numbers in the corresponding domain. Deformations and degrees of freedom in the mechanical domain have the lowest numbers, degrees of freedom and deformations in the hydraulic domain have the highest numbers.

- **Order of deformation numbers from low to high:**
  1. numbers of free deformations in mechanical domain
  2. numbers of prescribed deformations in mechanical domain
  3. numbers of free deformations in hydraulic domain
  4. numbers of prescribed deformations in hydraulic domain

- **Order of degree of freedom numbers from low to high:**
  1. numbers of free degrees of freedom in mechanical domain
  2. numbers of prescribed degrees of freedom in mechanical domain
  3. numbers of free degrees of freedom in hydraulic domain
  4. numbers of prescribed degrees of freedom in hydraulic domain

A.2.5 Build element matrices

<table>
<thead>
<tr>
<th>get recipe of element</th>
</tr>
</thead>
<tbody>
<tr>
<td>get first parameter of element</td>
</tr>
</tbody>
</table>

while a parameter was found

<table>
<thead>
<tr>
<th>substitute parameter of element in equations of recipe</th>
</tr>
</thead>
<tbody>
<tr>
<td>get next parameter of element</td>
</tr>
</tbody>
</table>

evaluate equations of recipe

<table>
<thead>
<tr>
<th>substitute values of equations $D_{ij}$, $S_{ij}$, and $T_{ij}$ in matrices of recipe</th>
</tr>
</thead>
<tbody>
<tr>
<td>calculate $D_{\text{glob, element}} = D_{\text{loc, element}} \cdot T$</td>
</tr>
</tbody>
</table>
A.2.6 Build system matrices

Below the algorithm for building the system deformation matrix for the mechanical domain is described. The algorithms for building the deformation matrix for the hydraulic domain and the system stiffness and transformation matrices are similar to this algorithm.

<table>
<thead>
<tr>
<th>get first element</th>
</tr>
</thead>
<tbody>
<tr>
<td>while an element was found</td>
</tr>
<tr>
<td>build element matrices</td>
</tr>
<tr>
<td>get corresponding recipe</td>
</tr>
<tr>
<td>n = 0</td>
</tr>
<tr>
<td>while n &lt; number of nodes recipe</td>
</tr>
<tr>
<td>get node with number node numbers[n] of element</td>
</tr>
<tr>
<td>j = 0</td>
</tr>
<tr>
<td>while j &lt; degrees of freedom per node of recipe</td>
</tr>
<tr>
<td>row = degree of freedom numbers[j] of node</td>
</tr>
<tr>
<td>i = 0</td>
</tr>
<tr>
<td>while i &lt; number of deformations of recipe</td>
</tr>
<tr>
<td>col = deformation numbers[i] of element</td>
</tr>
<tr>
<td>[ D_{\text{sys}}[\text{row}][\text{col}] = D_{\text{glob, element}}[i][j] ]</td>
</tr>
<tr>
<td>i = i + 1</td>
</tr>
<tr>
<td>j = j + 1</td>
</tr>
<tr>
<td>n = n + 1</td>
</tr>
<tr>
<td>get next element</td>
</tr>
</tbody>
</table>

A.3 Syntax of recipe file

The elements of the finite elements program are stored in recipe files. The name of the recipe file should be "*.rec"; "*" is the name of the recipe. In this section the syntax of the recipe file is described and an example is given.
A recipe exists of a set of keywords and parameters. The order of keywords is not important, except for the keywords `amountnodes`, `dof[i]`, and `formula`. The keyword `amountnodes` must be used before `dof[i]`. `formula`'s must be given in order of evaluation.

A recipe should define the `domain` of an element, the local deformation matrix `D`, the transformation matrix `T`, and the stiffness matrix `S`, see chapter 0. Indices of matrices and vectors start with index 1.

Matrix and vector entries must be multiplied by its units. The unit force `[F]` and unit length `[L]` are available. The units are used for internal scaling of the model. The scaling improves the accuracy of the results of the finite elements program.

The end of a recipe has to be marked with `end`.

- **amountnodes**
  Syntax: `amountnodes <integer n>`.
  Number of nodes of an element.

- **def**
  Syntax: `def <integer n>`.
  Number of deformations an element.

- **dof[i]**
  Syntax: `dof <integer i> <integer n>`.
  Number of degrees of freedom of node `i`. Index `i = 1, 2, ..., amountnodes`.

- **domain**
  Syntax: `domain <mech/hydr>`.
  Domain of an element.

- **D[i][j]**
  Syntax: `formula D[<integer i>][<integer j>] = <a formula>`.
  Index `i = 1, 2, ..., def`. Index `j = 1, 2, ..., (sum of degrees of freedom of all nodes)`. The formula must include the units of `D[i][j]`. For the definition of the deformation matrix, see chapter 0.

- **end**
  The end of recipe file has to be masked by this keyword.

- **eps[i]**
  Syntax: `formula eps[<integer i>] = <units>`.
Index $i = 1, 2, ..., \text{def}$. This statement is used to define the units of the deformations of an element in the mechanical domain.

- **[F]**
  
  Unit force. This keyword can be used in formula's. Units are used to scale the finite elements model to obtain more accurate results.

- **formula**
  
  Syntax: `formula <name> = <operators, real numbers, functions, parameters, results of previous formula's>;`

  Available operators: +, -, *, /, ^. Available functions: sin, cos, tan, asin, acos, atan, log, log10. A formula can contain parameters, results of previous formula's, node positions $x[n]$, $y[n]$, $z[n]$, entries from the inverse of the orientation matrix $oriinv[i][j]$ (see chapter 0), units [F], [L].

- **f[i]**
  
  Syntax: `formula f[i <integer i> ] = <units> `.  

  Index $i = 1, 2, ..., \text{(sum of the number of degrees of freedom of all nodes)}$. This statement is used to define the units of the load on a degree of freedom of an element in the mechanical domain.

- **[L]**
  
  Unit length. This keyword can be used in formula's. Units are used to scale the finite elements model to obtain more accurate results.

- **oriinv[i][j]**
  
  Entry of the inverse of the orientation matrix of an element. This keyword can be used in formula's and is predefined.

- **par**
  
  Syntax: `par <name> <default value>`.

  Parameters can be used in formula's. The value of the parameter is defined in the element data structure or in the model file. The parameter names exclude, exclude[i], rigid[i], sigmamin[i], and sigmamax[i] are reserved.

  `par exclude yes` excludes an element from the reliability analysis and `par exclude no` does not. exclude[i] can be used to exclude a single deformation $i$ form the analysis.
For reliability analysis of executing motion of physically non linear elements like non return valves, and cables an extra parameter \textit{rigid[i]} has to be added for the non linear element deformations, syntax: \textit{par rigid[i] < integer i >/ no}. \textit{i} is the non linear deformation number.

\textit{sigmamin[i]} and \textit{sigmamax[i]} define the maximum and minimum allowable stress of an element. They are used in the non linear analyses and to calculate the probability of failure of a component.

\section*{\textbf{phi[i]}}

Syntax: \textit{formula phi[i] < integer i > = < units >}.

Index \textit{i} = 1, 2, ..., def. This statement is used to define the units of the deformations of an element in the hydraulic domain.

\section*{\textbf{pi[i]}}

Syntax: \textit{formula pi[i] < integer i > = < units >}.

Index \textit{i} = 1, 2, ..., def. This statement is used to define the units of the pressures of an element in the mechanical domain.

\section*{\textbf{p[i]}}

Syntax: \textit{formula p[i] < integer i > = < units >}.

Index \textit{i} = 1, 2, ..., (sum of the number of degrees of freedom of all nodes). This statement is used to define the units of the pressure in a degree of freedom of an element in the hydraulic domain.

\section*{\textbf{rem}}

Syntax: \textit{rem < text >}. Remark. Comment until end of line.

\section*{\textbf{sigma[i]}}

Syntax: \textit{formula sigma[i] < integer i > = < units >}.

Index \textit{i} = 1, 2, ..., def. This statement is used to define the units of the stresses of an element in the mechanical domain.

\section*{\textbf{S[i][j]}}

Syntax: \textit{formula S[i][j] < integer i >/ < integer j > = < a formula >}.

Index \textit{i} = 1, 2, ..., def. Index \textit{j} = 1, 2, ..., def. This statement is used to define the stiffness matrix. The formula must include the units of \textit{S[i][j]}.

---

Finite element program

169
- \( T[i][j] \)
  Syntax: ‘formula \( T[i][\text{integer } i][\text{integer } j] \) = \(<\text{a formula}>\)’.
  Index \( i = 1, 2, \ldots \), (sum of degrees of freedom of all nodes). Index \( j = 1, 2, \ldots \), (sum of degrees of freedom of all nodes). For the definition of the transformation matrix, see chapter 0.

- \( u[i] \)
  Syntax: ‘formula \( u[i] \) = \(<\text{units}>\)’.
  Index \( i = 1, 2, \ldots \), (sum of the number of degrees of freedom of all nodes). This statement is used to define the units of the displacements of degrees of freedom of an element in the mechanical domain.

- \( V[i] \)
  Syntax: ‘formula \( V[i] \) = \(<\text{units}>\)’.
  Index \( i = 1, 2, \ldots \), (sum of the number of degrees of freedom of all nodes). This statement is used to define the units of the volume flows of degrees of freedom of an element in the hydraulic domain.

- \( x[n], y[n], z[n] \)
  Position parameters of node \( n \). These keywords can be used in formula’s and are predefined.

- **Example: recipe file of a two dimensional beam element**

```plaintext
domain mech
amountnodes 2
dof[1] 3
dof[2] 3
def 3
par Emod
par area
par Izz

rem units of deformations:
formula eps[1] = [L];
rem eps[2] and eps [3] have no units

rem units of stresses:
formula sigma[1] = [F];
formula sigma[2] = [F]*[L];
formula sigma[3] = [F]*[L];

rem units of nodal dispalcements:
formula u[1][1] = [L];
formula u[1][2] = [L];
rem u[1][3] has no units
formula u[2][1] = [L];
formula u[2][2] = [L];
rem u[2][3] has no units
```

![Figure A.3-1: Two dimensional beam element.](image-url)
rem units of nodal forces:

formula f[1][1] = [F];
formula f[1][2] = [F];
formula f[1][3] = [F]*[L];
formula f[2][1] = [F];
formula f[2][2] = [F];
formula f[2][3] = [F]*[L];

formula length = \((x[2]-x[1])^2 + (y[2]-y[1])^2)^{0.5};

\[
D = \begin{pmatrix}
-1 & 0 & 0 & 1 & 0 & 0 \\
0 & -\frac{1}{\ell} & 1 & 0 & \frac{1}{\ell} & 0 \\
0 & \frac{1}{\ell} & 0 & 0 & -\frac{1}{\ell} & -1
\end{pmatrix}
\]  \hspace{1cm} (A.3-1)

formula D[1][1] = -1;
formula D[1][4] = 1;

formula D[2][2] = -1/length * 1/[L];
formula D[2][5] = 1/length * 1/[L];

formula D[3][2] = 1/length * 1/[L];
formula D[3][5] = -1/length * 1/[L];

\[
S = \begin{pmatrix}
\frac{E.A}{\ell} & 0 & 0 \\
0 & \frac{4E.I}{\ell} & -2\frac{E.I}{\ell} \\
0 & -2\frac{E.I}{\ell} & \frac{4E.I}{\ell}
\end{pmatrix}
\]  \hspace{1cm} (A.3-2)

formula S[1][1] = Emod*area/length * [F]/[L];
formula S[2][2] = 4*Emod*Izz/length * [F]*[L];
formula S[3][3] = 4*Emod*Izz/length * [F]*[L];
formula S[2][3] = -2*Emod*Izz/length * [F]*[L];
formula S[3][2] = -2*Emod*Izz/length * [F]*[L];

\[
T = \begin{pmatrix}
uvw_{1,1}^{-1} & uvw_{1,2}^{-1} & 0 & 0 & 0 \\
uvw_{2,1}^{-1} & uvw_{2,2}^{-1} & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & uvw_{1,1}^{-1} & uvw_{1,2}^{-1} \\
0 & 0 & 0 & uvw_{2,1}^{-1} & uvw_{2,2}^{-1} \\
0 & 0 & 0 & 0 & 1
\end{pmatrix}
\]  \hspace{1cm} (A.3-3)

formula T[1][1] = oriinv[1][1];
formula T[2][1] = oriinv[2][1];
formula T[1][2] = oriinv[1][2];
formula T[2][2] = oriinv[2][2];
formula T[3][3] = 1;
formula T[4][4] = oriinv[1][1];
A.4 Syntax of model file

A model file exists of a list of nodes, elements, boundary conditions, prescribed stresses, and loads. These lists can be mixed and the order of the entries is not important. Each entry starts with a keyword node, element, bound, or stress, and ends with the keyword end. The end of the model has to be marked with end.

**bound**

Syntax: `bound nodenr <integer n> dof <integer d> <real v / free> end`.

The order of the sub-keywords nodenr and dof is not important. It is possible to prescribe more degrees of freedom of a node in one bound statement. The keyword dof can be used more than once. nodenr is a node number. Possible values: 0, 1, ..., etc. dof is a degree of freedom of the node. The degree of freedom number d is a relative/local degree of freedom number. d is the index in the degree of freedom array of the node (see figure A.1-4). Possible values: 1, 2, ..., number of degrees of freedom. v is the prescribed displacement, rotation or pressure of the degree of freedom. The value 'free' can be used if a degree of freedom is not prescribed.

**element**

Syntax: `element name <string s> nr <integer n> nodenrs <integer numbers i> orient <real u_1> <real v_1> <real w_1> <real u_2> <real v_2> <real w_2> <real u_3> <real v_3> <real w_3> par <string p> <string v> end`.

The sub-keyword name must be used before the other sub-keywords. The order of use of the other sub-keywords is not important. name is the name (recipe name) of the element. nr is an unique element number. Possible values: 0, 1, 2, etc. nodenrs are the node numbers of the element. This keyword is usually followed by more than one value. Possible values: 0, 1, 2, etc. orient is the orientation matrix [u v w], that
is described in chapter A.1-1 and in figure A.1-5. The sub-keyword *par* can be used more than once in an element definition. The keyword is followed by the parameter name and the parameter value.

**load**

Syntax: ‘load nodenr <integer n> dof <integer d> <real v> end’.

The order of the sub-keywords *nodenr* and *dof* is not important. It is possible to prescribe the loads on more degrees of freedom of a node in one *load* statement. The keyword *dof* can be used more than once. *nodenr* is a node number. Possible values: 0, 1, ..., etc. *dof* is a degree of freedom of the node. The degree of freedom number *d* is a relative/local degree of freedom number. *d* is the index in the degree of freedom array of the node (see figure A.1-4). Possible values: 1, 2, ..., number of *degrees of freedom*. *v* is the prescribed load or bending moment on the degree of freedom.

**node**

Syntax: ‘node nr <integer n> xyz <real x> <real y> <real z> end’.

The order of use of the sub-keywords *nr* and *xyz* is not important. *nr* is a unique node number. *xyz* are the global coordinates of the node.

**stress**

Syntax: ‘stress elementnr <integer n> def <integer d> <real v> end’.

The order of the sub-keywords *elementnr* and *def* is not important. It is possible to prescribe more stresses of an element in one *stress* statement. The keyword *def* can be used more than once. *elementnr* is an element number. Possible values: 0, 1, ..., etc. *def* is a deformation of the element. The deformation number *d* is a relative/local deformation number. *d* is the index in the deformation array of the element (see figure A.1-3). Possible values: 1, 2, ..., number of *deformations*. *v* is the prescribed stress or pressure.
Example: model with five 2D beam elements

![Example diagram with node and load specifications]

Figure A.4.1: Two dimensional finite elements model.

node
  nr 1
  xyz 0.000000e+00 0.000000e+00 0.000000e+00
end

node
  nr 2
  xyz 1.000000e+03 0.000000e+00 0.000000e+00
end

node
  nr 3
  xyz 1.000000e+03 1.000000e+03 0.000000e+00
end

node
  nr 4
  xyz 0.000000e+00 1.000000e+03 0.000000e+00
end

load
  nodenr 3
dof 1 5000
end

bound
  nodenr 1
dof 1 0
dof 2 0
dof 3 free
end

bound
  nodenr 2
dof 1 0
dof 2 free
dof 3 free
end

element
  name p1beam
  nr 1
  nodens 1 2
  orient 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00
  0.000000e+00 0.000000e+00 1.000000e+00
  par Emod 2.1e5
  par area 1.125000e+04
  par lzz 5.273438e+06
end

element
A.5 Syntax of initialisation file

It is necessary to define the program install path of the finite elements program, and the path of the model, recipe, mapping, and explanation files in the initialisation file, that is named *globvars.dat* in Appendix C. The scaling algorithm, that scales the finite elements before the calculation uses the parameters *unitlength* and *unitforce*. It scales the model in such a way, that the average element length and average force are equal to the values, that are defined in the ini-file. In the current version of the software the name of the ini-file is *ff2_32.ini*. Below, an example of an ini-file is given:
A.6 Commands

`autocoll` Auto select the algorithm for reliability analysis of the function carrying load.

`autonomo` Auto select the algorithm for reliability analysis of the function executing motion.

`chdata` Open menu to change size of rest, and action interval.

`check` Check if model has enough constraints, and show possible motions if it has not.

`classic` Classic static analysis.

`clearmodel` Clear model.

`collclassic` Reliability analysis of the function carrying load with classic equations.

`collschwab` Reliability analysis of the function carrying load with Schwabs algorithm.

`comparefiles <path> <path>` Compare files in the two paths, and write the differences to the file testproc.out.

`deformed` Draw model in deformed state.

`drawmodel` Draw model.

`drelmnrts or drawelmnrts` Draw element numbers.

`drnodes or drawnodes` Draw nodes.

`prepare` Prepare finite element equations.

`nomoclassic` Reliability analysis of the function executing motion with classic equations.
nomoschwab  Reliability analysis of the function executing motion with Schwabs algorithm.
nonlinclassic  Non linear static analysis with classic equations.
nonlincoll  Non linear reliability analysis of the function carrying load with Schwabs algorithm.
nonlincollclassic  Non linear reliability analysis of the function carrying load with classic equations.
nonlinnomo  Non linear reliability analysis of the function executing motion with Schwabs algorithm.
nonlinschwab  Non linear static analysis with Schwabs algorithm.
redraw  Redraw.
runtest <path>  The path defines a file, that contains commands. The finite elements program executes these commands.
schwab  Static analysis with Schwabs algorithm.
scale  Change scale to enlarge deformations.
wzmall [<screen point>]  Zoom all, for all windows, or for the window, that is defined by the screen point.
2win  Close all windows, and open two windows: a top view, and a side view.
4win  Close all windows, and open four windows: a top view, a front view, a side view, and an isometric view.
B Component modeller

Section B.1 to B.3 describe the modeller, B.4 describes the communication of the modeller and the finite elements program, B5 describes the initialisation file, and B6 the commands.

B.1 Data structures

The data structures of the modeller are very similar to the data structures of the finite elements program, that is described in chapter 4 and appendix A. This section describes the recipe, component and node data structure.

B.1.1 Definition of recipe

Recipes are stored in the global variable recipe list. The data in the structure recipe defines a component type. All fields or records of the recipe structure are explained in this subsection. Sometimes the names of the fields are slightly changed, because the original names in the source file are to cryptical.

- **name**

  The name of the recipe is a component type. The number of characters in this field is COMP_NAME_SIZE + 1 (NULL byte for end of string marker included). COMP_NAME_SIZE is defined to be 8 positions. Example: ‘hcyl’, ‘pipe’.

- **number of nodes (original name: amountnodes)**

  The number of nodes of an element of type <name>. Possible values: 0, 1, 2, 3, ..., etc.
- default parameter values (original name: defparvalues)
  Pointer to a list of default parameter values.

- set of equations (original name: eq)
  Pointer to a set of equations. The parameters of the component are also added to the set of equations.

- list of geometry basics (original name: basiclist)
  Pointer to a list of geometry elements. Possible geometry: line, circle, cylinder, cone, and ball. First, the parameters of a component are substituted in the set of equations of the recipe. Then, the co-ordinates and parameters of the geometry are calculated in the local co-ordinate system of the recipe. Finally the list of geometry basics of the recipe is transformed to global co-ordinates and copied to the component.

- next
  The global variable recipe list is implemented as a linked list of recipes. next is a pointer to the next recipe in a list of recipes or NULL if the recipe is the last recipe in the list.

B.1.2 Definition of component

Components are stored in the global variable component list (original name: complist). All fields or records of the component data structure are explained in this subsection. Sometimes the names of the fields are slightly changed, because the original names in the source file are too criptical.

- number
  Each component has a unique number. Possible values: 0, 1, 2, ..., etc.

- name
  The name of a component type. The number of characters in this field is COMP_NAME_SIZE + 1 (NULL byte for end of string marker included). COMP_NAME_SIZE is defined to be 8 positions. Example: 'hcyl', 'pipe'.
- **node numbers (original name: nodenrs)**

  Array of node numbers. The size of the array is defined by the field *number of nodes* of the corresponding recipe. Possible values: 0, 1, 2, 3, ..., etc.

- **position**

  Pointer to a $3 \times 1$ matrix with the position of the origin of the component. See figure B.1-2.

- **orientation (original name: orient)**

  Pointer to a $3 \times 3$ matrix with the orientation of the component. See figure B.1-1.

- **list of parameters (original name: par)**

  Pointer to a list of parameters. The parameters are substituted in the equations of the corresponding recipe when they are evaluated.

- **list of geometry basics (original name: basiclist)**

  Pointer to a list of geometry elements. Possible geometry: line, circle, cylinder, cone, and ball. The corresponding recipe is used to calculate the *list of geometry basics* of a component. First, the parameters of a component are substituted in the *set of equations* of the recipe. Then, the co-ordinates and parameters of the geometry are calculated in the local co-ordinate system of the recipe. Finally the list of geometry basics of the recipe is transformed to global co-ordinates and copied to the component.

- **has moved**

  Marker that can be set to TRUE if a component has been moved during an operation.

- **next**

  The global variable component list is implemented as a linked list. *next* is a pointer to the next component in a list of components or NULL if the component is the last component in the list.
B.1.3 Definition of node

Nodes are stored in the global variable node list. All fields or records are described in this subsection.

- **number**
  
  Each node has a unique number. Possible values: 0, 1, 2, ..., etc.

- **x, y, z**
  
  Coordinates of the node in the global co-ordinate system. Example: \( x = 4.311 \times 10^3, \ y = 1.001 \times 10^9, \ z = 9.588 \times 10^3 \).

- **next**
  
  The global variable nodelist is implemented as a linked list of nodes. *next* is a pointer to the next node in a list of nodes or NULL if the node is the last node in the list.

B.2 Syntax of recipe file

The components of the modeller are stored in recipe files. The recipe file describes the geometry of the component. The name of the recipe file should be "*.rec"; "*" is the name of the recipe. In this section the syntax of the recipe file is described and an example is given.

A recipe exists of a set of keywords and parameters. The order of keywords is not important. *Formula’s* must be given in order of evaluation.

- **amountnodes**

  Syntax: ‘*amountnodes* <integer n>’.

  Number (amount) of nodes of an element.

- **basic**

  Syntax: ‘*basic* <name> <node positions> <parameters> end’.

  Syntax of node position:

  ‘par x[i] <value x> par y[i] <value y> par z[i] <value z>’.

  Syntax of parameter: ‘par <parameter name> <value>’.
Geometry basic. Possible values for name: line, cir, ball, cyl, cone. The basics line, cir, cyl, and cone have two nodes. A ball has one node. A cone has two parameters: r[1], and r[2] (radius). The basics cir, ball, and cyl have one parameter: r (radius). A line has no parameters.

- **end**

  The end of the recipe file has to be marked by this keyword.

- **formula**

  Syntax: ‘formula <name> = <operators, real numbers, functions, parameters, results of previous formula’s>;’

  Available operators: +, -, *, /, ^. Available functions: sin, cos, tan, asin, acos, atan, log, log10. A formula can contain parameters, results of previous formula’s, and node positions x[n], y[n], z[n].

- **par**

  Syntax: ‘par <name> <default value>’.

  Parameters can be used in formula’s.

- **rem**


### Example: Recipe file of a beam

```plaintext
amountnodes 2
par height 150
par width 75
par length 500
par Emod 2.1e5

rem parameter for probabilistic calculations:
par Sigmanax 140
par taumax 70

formula x[1] = 0;
formula y[1] = 0;
formula z[1] = 0;
formula x[2] = length;
formula y[2] = 0;
formula z[2] = 0;
formula ymax = width/2;
formula ymin = -width/2;
formula zmax = height/2;
formula zmin = -height/2;
formula area = width*height;
formula Iyy = 1/12*width*height^3;
formula Izz = 1/12*height*width^3;
formula Ip = Iyy + Izz;

rem formulas for probabilistic calculations:
formula Nmax = sigmanax*area;
```

Component modeller 183
formula \( N_{\text{min}} = -E\text{mod}^2 Izz/\text{length}^2 \);
formula \( T_{\text{max}} = \tau_{\text{aumax}}^2 I_{\text{p/height}} \);
formula \( T_{\text{min}} = -\tau_{\text{aumax}}^2 I_{\text{p/height}} \);
formula \( M_{\text{yymax}} = \sigma_{\text{gmax}}^2 I_{\text{y/height}} \);
formula \( M_{\text{yymin}} = -\sigma_{\text{gmax}}^2 I_{\text{y/height}} \);
formula \( M_{\text{zzmax}} = \sigma_{\text{gmax}}^2 I_{zz/width} \);
formula \( M_{\text{zzmin}} = -\sigma_{\text{gmax}}^2 I_{zz/width} \);

```
basic
  name line
  par x[1] 0
  par y[1] ymin
  par z[1] zmin
  par x[2] length
  par y[2] ymin
  par z[2] zmin
end
```

```
basic
  name line
  par x[1] length
  par y[1] ymin
  par z[1] zmin
  par x[2] length
  par y[2] ymax
  par z[2] zmin
end
```

```
basic
  name line
  par x[1] length
  par y[1] ymax
  par z[1] zmin
  par x[2] 0
  par y[2] ymax
  par z[2] zmin
end
```

```
basic
  name line
  par x[1] 0
  par y[1] ymax
  par z[1] zmin
  par x[2] 0
  par y[2] ymin
  par z[2] zmin
end
```

```
basic
  name line
  par x[1] length
  par y[1] ymin
  par z[1] zmax
  par x[2] length
  par y[2] ymin
  par z[2] zmax
end
```

```
basic
  name line
  par x[1] length
  par y[1] ymax
  par z[1] zmax
  par x[2] 0
end
```

184  Component modeller
par y[2] ymax
par z[2] zmax
end

basic
name line
par x[1] 0
par y[1] ymax
par z[1] zmax
par x[2] 0
par y[2] ymin
par z[2] zmax
end

basic
name line
par x[1] 0
par y[1] ymin
par z[1] zmin
par x[2] 0
par y[2] ymin
par z[2] zmax
end

basic
name line
par x[1] length
par y[1] ymin
par z[1] zmin
par x[2] length
par y[2] ymin
par z[2] zmax
end

basic
name line
par x[1] length
par y[1] ymax
par z[1] zmin
par x[2] length
par y[2] ymax
par z[2] zmax
end

basic
name line
par x[1] 0
par y[1] ymax
par z[1] zmin
par x[2] 0
par y[2] ymax
par z[2] zmax
end

end

B.3 Syntax of model file

A model file exists of a list of nodes and components. This list can be mixed and the order of the entries is not important. Each entry starts with a keyword node or comp and ends with the keyword end. The end of the model has to be marked with end.
**comp**

Syntax: ‘comp name <string s> nr <integer n> nodenrs <integer numbers i> position <real x> <real y> <real z> orient <real u1> <real v1> <real w1> <real u2> <real v2> <real w2> <real u3> <real v3> <real w3> par <string p> <string v> end’.

The sub-keyword name must be used before the other sub-keywords. The order of the other sub-keywords is not important. name is the name (recipe name) of the component. nr is an unique component number. Possible values: 0, 1, 2, etc. nodenrs are the node3 numbers of the component. This keyword is usually followed by more than one value. Possible values: 0, 1, 2, etc. position is the position of the origin of the component. orient is the orientation matrix [u v w], that is described in chapter B.1.2. The sub-keyword par can be used more than once in an component definition. The keyword is followed by the parameter name and the parameter value.

**end**

The end of the recipe file has to be marked by this keyword.

**node**

Syntax: ‘node nr <integer n> xyz <real x> <real y> <real z> end’.

The order of the sub-keywords nr and xyz is not important. nr is a unique node number. Possible values: 0, 1, 2, etc. xyz are the global coordinates of the node. Example: 1.034*10^3 2,000*10^2 -3,000*10^2.
Example of model file

node
  nr 1
  xyz 9.999996e+02 1.000000e+03 0.000000e+00
end

node
  nr 3
  xyz 9.999996e+02 0.000000e+00 0.000000e+00
end

node
  nr 2
  xyz -3.662109e-04 0.000000e+00 0.000000e+00
end

node
  nr 10
  xyz -3.662109e-04 1.000000e+03 0.000000e+00
end

comp
  name plforce
  nr 8
  nodenrs 1
  position 1.000000e+03 9.999998e+02 0.000000e+00
  orient 1.000000e+00 0.000000e+00 0.000000e+00
          0.000000e+00 1.000000e+00 0.000000e+00
          0.000000e+00 0.000000e+00 1.000000e+00
  par fx 1000
  par fy 0
  par scale 0.5
end

comp
  name plbound
  nr 7
  nodenrs 3
  position 1.000000e+03 -1.831055e-04 0.000000e+00
  orient 1.000000e+00 0.000000e+00 0.000000e+00
          0.000000e+00 1.000000e+00 0.000000e+00
          0.000000e+00 0.000000e+00 1.000000e+00
  par nx free
  par ny 0
  par gamma free
  par scale 100

Component modeller
end

comp
ame plbound
nr 6
nodens 2
position -3.662109e-04 0.000000e+00 0.000000e+00
orient 1.000000e+00 0.000000e+00 0.000000e+00
0.000000e+00 1.000000e+00 0.000000e+00
0.000000e+00 0.000000e+00 1.000000e+00
par nx 0
par ny 0
par gamma free
par scale 100
end

comp
name pltruss
nr 5
nodens 3 10
position 1.000000e+03 -1.831055e-04 0.000000e+00
orient -7.071068e-01 -7.071068e-01 0.000000e+00
7.071068e-01 -7.071068e-01 0.000000e+00
0.000000e+00 0.000000e+00 1.000000e+00
par diameter 25
par length 1.414214e+03
par Emod 2.1e5
par sigmamax 140
end

comp
name pltruss
nr 4
nodens 2 10
position -3.662109e-04 0.000000e+00 0.000000e+00
orient -4.371139e-08 -1.000000e+00 0.000000e+00
1.000000e+00 -4.371139e-08 0.000000e+00
0.000000e+00 0.000000e+00 1.000000e+00
par diameter 25
par length 1000
par Emod 2.1e5
par sigmamax 140
end

comp
name pltruss
nr 3
nodens 3 1
position 1.000000e+03 -1.831055e-04 0.000000e+00
orient -4.371139e-08 -1.000000e+00 0.000000e+00
1.000000e+00 -4.371139e-08 0.000000e+00
0.000000e+00 0.000000e+00 1.000000e+00
par diameter 25
par length 1000
par Emod 2.1e5
par sigmamax 140
end

comp
name pltruss
nr 2
nodens 10 1
position -3.662109e-04 1.000000e+03 0.000000e+00
orient 1.000000e+00 0.000000e+00 0.000000e+00
0.000000e+00 1.000000e+00 0.000000e+00
0.000000e+00 0.000000e+00 1.000000e+00
par diameter 25
par length 1000
par Emod 2.1e5
par sigmamax 140
end

comp
name pltruss
B.4 From components to elements

This section describes the translation form components to finite elements. The first subsection describes the mapping file, that defines the translation. The second subsection describes the file that defines the meaning of the failure modes.

B.4.1 Mapping file

The mapping algorithm first prints the list of component nodes to the finite elements file. Then the list of components is translated. A mapping file, which describes the translation, exists for each component. The contents of this file are written to the finite element model.

All characters in the mapping file are copied to the finite element file. Except for words between ‘#’ signs. They are parameters or keywords and are replaced by their values.

- **#compnr[i]#**
  Prints the component number in the finite elements file, when i is equal to 1. A new component number is created if i is equal to 2 or larger.

- **#nodenr[i]#**
  Prints node number i in the finite elements file. Possible values for i: 1, 2, ..., etc.

- **#orient[i][j]#**
  Prints entry with indices i and j of the component orientation matrix (see figure B.1-2) in the finite elements file. Possible values for i: 1, 2, and 3. Possible values for j: 1, 2, and 3.
Example: part of the mapping file of a hydraulic cylinder

![Screen dump of component modeller with a hydraulic cylinder component.](image)

```plaintext
rem start mapping of hcyl:
  element
    name truss
    nr #compr[1]#
    nodens #noden[1]# #noden[3]#
    orient #orient[1][1]# #orient[1][2]# #orient[1][3]#
    #orient[2][1]# #orient[2][2]# #orient[2][3]#
    #orient[3][1]# #orient[3][2]# #orient[3][3]#
    par Emod #Esteel#
    par area #Arod#
  rem parameters for probabilistic calculations:
    par sigmamax[1] #Nrodmax#
    par sigmamin[1] #Nrodmin#
    rem extra parameters:
    par compr[2]#
    par moden[1] 1
    par exclude #exclude#
  end

element
  name beam
  nr #compr[2]#
  nodens #noden[1]# #noden[2]#
  orient #orient[1][1]# #orient[1][2]# #orient[1][3]#
    #orient[2][1]# #orient[2][2]# #orient[2][3]#
    #orient[3][1]# #orient[3][2]# #orient[3][3]#
  par Emod #Esteel#
  par area #Arod#
  par Iyy #Iyyrod#
  par Izz #Izzrod#
  par Ip #Iprod#
  rem parameters for probabilistic calculations:
    par Prob[1] #Pclogged#
    par sigmamax[2] 1.0e+03 rem small
    par sigmamin[2] -1.0e+03 rem small
    par sigmamax[3] #Myyrodmax#
    par sigmamin[3] #Myyrodmin#
    par sigmamax[4] #Myyrodmax#
    par sigmamin[4] #Myyrodmin#
    par sigmamax[5] #Mzzrodmax#
    par sigmamin[5] #Mzzrodmin#
    par sigmamax[6] #Mzzrodmax#
    par sigmamin[6] #Mzzrodmin#
  rem extra parameters:
```

Component modeller
par compname hcy1
par compnr #compr[1]#
par modenr[1] 2
par modenr[2] 3
par modenr[3] 4
par modenr[4] 4
par modenr[5] 4
par modenr[6] 4
par exclude #exclude#
end

rem this part of the rod doesn't take longitudinal stress:
stress
  elementnr #compr[2]#
def 1 0.0
end

B.4.2 Explanation file

Each element deformation represents a failure mode. The physical meaning of an element deformation is determined by the component from which the element originates. The explanation file contains a list of component failure modes of a component type. The failure mode numbers and the originating component types and can be added to the parameter list in the finite elements model. The keyword par is used for this purpose:

par compname hcy1
par compnr #compr[1]#
par modenr[1] 2

The statements above tell the finite elements program that an element originates from a ‘hcy1’ (= hydraulic cylinder) component with number #compr[1]# and that deformation number 1 of the element represents failure mode number 2 of the hydraulic cylinder.

Syntax of the explanation file:

'mode <failure mode number i> "<description of failure mode>”
Trepair <mean repair time> srepair <standard deviation of repair time>
[  model ]
[   Lrest <mean value of lambda rest>
srest <value of standard deviation of lambda rest>
Qstart <mean value of Q start>
ssstart <value of standard deviation of Q start>
Laction <mean value of lambda action>
saction <value of standard deviation of lambda action>
Qstop <mean value of Q stop>
ssstop <value of standard deviation of Q stop>  ]'
The explanation file gives two options for the definition of the failure data: The first option, *model*, means that the failure data are defined by a probabilistic model, that calculates the probability of failure with the load on the elements and the maximum and minimum allowable stresses $\text{sigmamin}[i]$ and $\text{sigmamax}[i]$. The second option, *Lrest* etc., defines the failure data in the explanation file.

**Example: explanation file of hydraulic cylinder**

mode 1 "Collapse of rod due to longitudinal stress."
    Trepair 96.0  srepair 48.0
    model

mode 2 "Obstruction of movement of part of rod outside of the cylinder."
    Lrest 1.0e-8  srest 0.5e-8
    Qstart 0.5e-7  sstart 0.25e-7
    Laction 1.0e-7  saction 0.5e-7
    Qstop 0.5e-7  sstop 0.5e-7
    Trepair 96.0  srepair 48.0
    rem Laction are data by RWS

mode 3 "Collapse due to torsional moment of part of rod outside of the cylinder."
    Trepair 96.0  srepair 48.0
    model

mode 4 "Collapse due to bending moment of part of rod outside of the cylinder."
    Trepair 96.0  srepair 48.0
    model

mode 5 "Obstruction of movement of part of rod inside of the cylinder."
    Lrest 1.0e-8  srest 0.5e-8
    Qstart 0.5e-7  sstart 0.25e-7
    Laction 1.0e-7  saction 0.5e-7
    Qstop 0.5e-7  sstop 0.5e-7
    Trepair 96.0  srepair 48.0
    rem Laction are data by RWS

mode 6 "Collapse due to bending moment of part of rod inside of the cylinder."
    Trepair 96.0  srepair 48.0
    model

mode 7 "Leaking of oil through seal below the piston."
    Trepair 96.0  srepair 48.0
    model

mode 8 "Clogging of rod and piston in mantle due to obstruction in oil below the piston."
    Trepair 96.0  srepair 48.0
    Lrest 1.0e-8  srest 0.5e-8
    Qstart 0.5e-7  sstart 0.25e-7
    Laction 1.0e-7  saction 0.5e-7
    Qstop 0.5e-7  sstop 0.5e-7
    rem Laction are data by RWS
mode 9 "Leaking of oil through pipe connection below the piston."
    Trepair 96.0  srepair 48.0
mode

mode 10 "Leaking of oil through seal above the piston."
    Trepair 96.0  srepair 48.0
mode

mode 11 "Clogging of rod and piston in mantle due to obstruction in oil
     above the piston."
    Lrest 1.0e-8 srest 0.5e-8
    Qstart 0.5e-7 sstart 0.25e-7
    Laction 1.0e-7 saction 0.5e-7
    Qstop 0.5e-7 sstop 0.5e-7
    Trepair 96.0  srepair 48.0
    rem Laction are data by RWS
mode 12 "Leaking of oil through pipe connection above the piston."
    Trepair 96.0  srepair 48.0
mode

mode 13 "Leaking of oil along the piston."
    Trepair 96.0  srepair 48.0
mode

mode 14 "Leaking of the mantle."
    Trepair 96.0  srepair 48.0
mode

mode 15 "Collapse of mantle due to longitudinal stress."
    Trepair 96.0  srepair 48.0
mode

mode 16 "Obstruction of movement below the piston."
    Lrest 1.0e-8 srest 0.5e-8
    Qstart 0.5e-7 sstart 0.25e-7
    Laction 1.0e-7 saction 0.5e-7
    Qstop 0.5e-7 sstop 0.5e-7
    Trepair 96.0  srepair 48.0
    rem Laction are data by RWS
mode 17 "Collapse due to bending moment of part of mantle below the
     piston."
    Trepair 96.0  srepair 48.0
mode

mode 18 "Obstruction of movement above the piston."
    Lrest 1.0e-8 srest 0.5e-8
    Qstart 0.5e-7 sstart 0.25e-7
    Laction 1.0e-7 saction 0.5e-7
    Qstop 0.5e-7 sstop 0.5e-7
    Trepair 96.0  srepair 48.0
    rem Laction are data by RWS
mode 19 "Collapse due to torsional moment of part of mantle above the
     piston."
    Trepair 96.0  srepair 48.0
mode

mode 20 "Collapse due to bending moment of part of mantle above the
     piston."

Component modeller
B.5 Syntax of initialisation file

It is necessary to define the program install path of the component modeller, the program install path of the finite elements program, and the path of the model, recipe, mapping, and explanation files in the initialisation file, that is named globvars.dat in Appendix C. In the current version of the software the name of the ini-file is gedit_32.ini. Below, an example of an ini-file is given:

```
[PATHS]
programinstallpath= .\ 
ff2installpath= ..\ff32\ 
recipepath= ..\ge2_rec\ 
mappingpath= ..\ff2_map_new\ 
modelpath= ..\projecten\ 
```

B.6 Commands

`ang <3D point> <3D point>`

Calculate the angle between the line through the first and the second, and the first and the third point, and place the result on the command line.

`autoconn`

Autoconnect. Connect all nodes of all selected components.

`clearmodel`

Clear model.

`comp <component name> <3D point>`

Create a component of type ‘name’ and position it on the 3D point.

`conn <3D point>`

Connect the node near the 3D point of a selected component.

`copy <3D vector>`

Copy selected items and translate the copied components along the 3D vector.

`del`

Delete selected components.

`delyes`

Delete selected components without confirmation.
\textit{disconn} <3D point> \quad \text{Disconnect the node near the 3D point of the last selected component.}

\textit{dist} <3D point> <3D point> \quad \text{Calculate the distance between the points, and put the result on the command line.}

\textit{dmovedown} <screen point> \quad \text{Mouse down command for dynamically moving - drag dropping - components.}

\textit{dmovemove} <screen point> \quad \text{Mouse move command for dynamically moving - drag dropping - components.}

\textit{dmoveup} <screen point> \quad \text{Mouse up command for dynamically moving - drag dropping - components.}

\textit{drcmpnrs} \quad \text{Draw component numbers.}

\textit{drlnodes} \quad \text{Draw nodes.}

\textit{dshiftdown} <screen point> \quad \text{Mouse down command for dynamic shift (pan).}

\textit{dshiftmove} <screen point> \quad \text{Mouse move command for dynamic shift (pan).}

\textit{dshiftup} <screen point> \quad \text{Mouse up command for dynamic shift (pan).}

\textit{dzoomdown} <screen point> \quad \text{Mouse down command for dynamic zoom.}

\textit{dzoommove} <screen point> \quad \text{Mouse move command for dynamic zoom.}

\textit{dzoomup} <screen point> \quad \text{Mouse up command for dynamic zoom.}

\textit{ff2dump} \quad \text{Translates the current model file to a finite element file.}

\textit{ff2dumpno} \quad \text{Show pop up menu with the message: 'Model not translated.'}

\textit{ff2dumpyes <path>} \quad \text{Translate the current model file to a finite element file. The name of the file is defined in the path.}

\textit{ff2start} \quad \text{Translate the current model file to a finite elements file, and start the finite elements program.}

\textit{ff2startyes <path>} \quad \text{Translate the current model file to a finite element file, and start the finite}
elements program. The name of the file is defined in the path.

`getnode <3D point>` Put coordinates of the nearest node on the command line.

`givedir` Put the command `makedir` plus the path of current directory on the command line.

`makedir <directory name>` Create directory.

`move <3D point>` Move selected components along the vector that is defined by the 3D points.

`<3D point>`

`par <name> <value>` Change the parameter values of all selected components. For values with spaces or values that are also commands like “yes”, and “no” use double quotation marks: “”.

`redraw` Redraw.

`rot <3D point>` Rotate selected components around the vector, that is defined by the 3D points.

`<3D point>` `<angle>`

`rot2 <3D vector>` Rotate selected components around the 3D vector, that is located in the current position of the 3D cursor.

`<angle>`

`sel <3D point>` Select a component near the 3D point.

`sgetnode <screen point>` Smart get node. Put coordinates of the nearest node on the command line.

`showform` Show the formula results of the last selected component.

`showpars` Show the parameters of the last selected component.

`sset <screen point>` Smart select. Select a component near the screen point.

`unsset <screen point>` Smart unselect. Unselect a component near the screen point.

`unsall` Unselect all components.
unsel <3D point>

Unselect a component near the 3D point.

valid

Move all components in such a way, that the connected nodes are coincident.

vector <3D point>
<3D point>

Put the vector, that is defined by the two points, on the command line.

wzmall [<screen point>]

Zoom all, for all windows, or for the window, that is defined by the screen point.

2win

Close all windows and open two windows: a top view, and a side view.

4win

Close all windows and open four windows: a top view, a front view, a side view, and an isometric view.
C Program architecture

It is very easy to create software and user interfaces with modern programming tools. These user interfaces are system dependent. The user interface has to be adjusted to port it to a different operating system. It is even possible that the total program has to be rewritten, when no attention is paid to the program architecture.

![Diagram showing program architecture](image)

*Figure C.1: Program Lay out. The boxes with a grey background represent system dependent software. The other boxes represent system independent software. This software architecture makes it possible to port the program to many different platforms.*

The software that has been written for this research project has been build according to the scheme that is printed above. The system dependent and the system independent parts of the software are separated. The program kernel communicates with the user interface via a command line. The user interface is separated in a system dependent
event handler and system independent menu functions. System depend- 
ent user interfaces, that are generated by a programming tool, can also 
communicate with the program kernel via the command line. How- 
ever, no system dependent user interfaces have been implemented yet.

This appendix describes the implementation in C of the user inter-
face. The first section describes matrix operations. The second section 
explains how the graphical functions are implemented. Then the for-
formula evaluator is described, and finally the user interface functions are 
described.

C.1 Matrix operations

This section describes the matrix operations that are defined in the C 
files matrixop.c and matrixop.h. The functions use scroll.c and scroll.h to 
display matrices and error messages on the screen.

C.1.1 Initialising and releasing a matrix

The function initmat initialises a matrix. The function allocates memory 
for the matrix and initiates the matrix values.

MATRIX *M; /* declares a pointer to a matrix */ 
M = initmat (2, 3, 0.0);

Result: M is a pointer to a matrix with 2 rows and 3 columns. All ma-
trix entries have the value 0.0.

The matrix has to be discarded with the function trash. This func-
tion releases all memory.

trash (M);

Result: All memory that was occupied by matrix M is released.

Some functions return a matrix. It is not necessary to initialise the 
matrix that is returned:

MATRIX *A, *B, *C; /* declares three pointers to matrices */ 
A = initmat (2, 3, 1.0); /* initialise matrix A with value 1.0 */ 
B = initmat (2, 3, 2.0); /* initialise matrix B with value 2.0 */ 
C = addmatCAB (A, B); /* initialise matrix C and store the */ 
/* result of A + B in it. */

addmatCAB adds two matrices and returns the result in a matrix. The 
function initialises the matrix and allocates the memory of a matrix. It
is not possible to initialise a matrix twice. The memory space that was allocated the first time becomes unreachable if a matrix is initialised twice. To store the result of and addition in C for the second time, use the function addmatABC:

```c
MATRIX *D, *E; /* declares two pointers to matrices */
/* C has been declared elsewhere */
D = initmat (2, 3, 2.0); /* initialise matrix D with value 2.0 */
E = initmat (2, 3, 3.0); /* initialise matrix E with value 3.0 */
C = addmatABC (D, E, C); /* initialise matrix C and store the */
/* result of D + E in it. */
```

C.1.2 Accessing data in a matrix

Use the function pmat to put data in a matrix:

```
pmat (M, 2, 3, 1.0);
```

Result: $M_{2,3} = 1.0$.

Use gmat to retrieve (get) data from a matrix:

```
float x;
x = gmat (M, 2, 3);
```

Result: x has the value of $M_{2,3}$.

`showmat` prints the entries of a matrix on screen:

```
showmat (M, "%.8.1e ");
```

Result: M was printed on the screen in floating point format taking 8 places with 1 digit behind the point, the numbers are separated by a space. Other format specifiers are allowed too. Example: "%.f " or "%.9.2e ".

C.1.3 Matrix functions

- Initialising and releasing a matrix

Initialise a matrix:

```
MATRIX *initmat (int rows, int cols, float initval);
```

Initialise a square matrix and initialise its diagonal entries with ‘initval’:

```
MATRIX *diagmat (int size, float initval);
```
Release a matrix:

```c
void trash (MATRIX * mat);
```

**Accessing data in a matrix**

Get an entry form a matrix:

```c
float gmat (MATRIX * mat, int row, int col);
```

Get a part of a matrix and store it in another matrix:

```c
MATRIX * partmatA (MATRIX * A, int i, int j, int k, int l);
void partmatAB (MATRIX * A, MATRIX * B, int i, int j, int k, int l);
```

Put a value in an entry of a matrix:

```c
void pmat (MATRIX * mat, int row, int col, float val);
```

Put zeros in a matrix:

```c
void clearmat (MATRIX * M);
```

Print a matrix on the screen; ‘format’ is a format specifier like in `printf`:

```c
void showmat (MATRIX * mat, char format[80]);
```

**Add matrices**

```c
void addmatAB (MATRIX * A, MATRIX * B, int row, int col);
void addmatABC (MATRIX * A, MATRIX * B, MATRIX * C);
MATRIX * addmatCAB (MATRIX * A, MATRIX * B);
void adddiagmatAB (MATRIX * A, MATRIX * B);
```

**Transpose matrices**

```c
MATRIX * transpM (MATRIX * M);
void transpMMT (MATRIX * M, MATRIX * MT);
```

**Multiply matrices**

Multiply two matrices:

```c
MATRIX * multiCAB (MATRIX * A, MATRIX * B);
void multiABC (MATRIX * A, MATRIX * B, MATRIX * C);
void diagmatmultiAD (MATRIX * A, MATRIX * D);
void diagmatmultiADB (MATRIX * A, MATRIX * D, MATRIX * B);
MATRIX * diagmatmultiBAD (MATRIX * A, MATRIX * D);
```

Multiply a matrix and a floating point number:
**Inversion and solving a set of equations**

LU-decomposition:

`void ludecomp (MATRIX * M);`

Solve a set of equations that has been decomposed with `ludecomp`:

`MATRIX *lusolv (MATRIX * LU, MATRIX * b);`
`MATRIX *gauss (MATRIX * M, MATRIX * b);`
`MATRIX *inverse (MATRIX * M);`
`void invert (MATRIX * M);`

---

**Normate**

`void normateM (MATRIX * M);`
`MATRIX *normateNM (MATRIX * M);`

---

### C.2 2D and 3D graphics

#### C.2.1 Initialising graphics

Declare a global variable (for MS Windows):

`HDC a_device_context;`

At the start of the program, initialise variables:

`clear_errorlog ();`  /* reset error log file */
`/* initialise global variable with screen device (for MS Windows) */`
`a_device_context = GetDC (Form1->WindowHandle);`
`initdevinf ();`  /* initialise screen device information */
`deviceopen ();`  /* open device */
`graphon ("globvars.dat");`  /* read variables for 2D and 3D */
`/* graphics */
`readbitf ("bitfont.bit");`  /* read bit map font */
`readmenulist ("menu.dat");`  /* read user interface */
`drawmenulist ();`  /* draw user interface */
The file names "globvars.dat", "bitfont.bit", and "menu.dat" should be adjusted to the names of the actual files that the application uses.

At the end of the program, release memory and data:

```c
destroymenulist (); /* release memory allocated in */
    /* readmenulist */
graphof (); /* release memory allocated in graphon */
deviceclose (); /* close screen device */
/* Release handle to device context (for MS Windows) */
ReleaseDC (Form1->WindowHandle, a_device_context);
```

The function `graphon` reads data from a file. An example that shows the syntax of this file is given in this sub-paragraph. Syntax:

```
<keyword> = <value>
```

All keywords that are read by `graphon` are given below. Besides these keywords other, application dependent keywords, can be defined in "globvars.dat".

### Example of "globvars.dat"

[DOS]
bgipath= .\n
[FRONT]
fontfile= linefont.dat
fontsize= 0.900000

[LINES]
dottedline= 0

[GRID]
groidon= 0
gridsize= 50.00000

[COLORS]
foregroundcolor= 0
backgroundcolor= 7
highlightcolor= 19
shadowcolor= 12
alarmcolor= 15
popupcolor= 9

### Syntax of "bitfont.bit"

The size of the characters is defined by the keywords `horibitmap` and `vertbitmap`.

```
horibitmap <horizontal amount of pixels per character>
vertbitmap <horizontal amount of pixels per character>
<ascii code> <enter> <x> <y> <x> <y>...... <enter>
```

The value of `<x>` is the horizontal position of a pixel and `<y>` is the vertical position of a pixel. The values vary from 1 to the horizon-
tal/vertical amount of pixels and take only one character. There are no spaces between the x and y values. Higher values than 9 can be defined by the next character in the range of ascii characters.

- **Syntax of “menu.dat”**

The syntax of the menu file is described in subsection C.3.2.

- **Syntax of the line font file**

The name of the line font file is defined in “globvars.dat” with the keyword `fontfile=`. The syntax of the file is:

```
<ascii code> <enter>
/l<x> <y> <x> <y>/l<x> <y> <x> <y>......<enter>
```

The value of `<x>` is the horizontal position of an endpoint of a line and `<y>` is the vertical position. The values vary from 0 to 9. There are no spaces between the x and y values.

The size of the line font can be changed by the keyword `fontsize=`. `Fontsize = 1.0` results in a font size with a maximum amount of 80 x 25 characters on the screen. The size is stored in the global variable `fontsize`, that can be changed in the application code too.

**C.2.2 2D graphics**

The `BOX` is the main data structure in the graphics library. A box can be created and is drawn on the screen by the function `makebox`.

```
BOX box; /* declare variable with type BOX */
box = makebox (100, 200, 300, 250);
```

Result: A box was drawn on the screen and was stored in the variable ‘box’. The upper left corner of the box has coordinates `x = 100` and `y = 200`, the lower right corner of the box has coordinates `x = 300` and `y = 250`.

The box will remain on the screen in a single tasking operating system. However, in a multi tasking environment it may disappear when the application window is closed and re-opened. Therefore, an application in a multi tasking environment should be able to redraw itself. The application programmer has to build a redraw routine in a standard programming environment.

This graphics library offers automatic redraw functionality. Thus, the programmer work like in a single tasking environment. Things that
have been drawn on the screen remain on the screen. All boxes and text and lines that are drawn in the boxes are stored automatically. They are redrawn when required. The programmer should initialise the global variable boxstoremode to be TRUE to activate this functionality:

boxstoremode = TRUE;

/* automatically store all drawn items */

\section*{Drawing in a box}

As mentioned before: the box is the main data structure. All items are drawn in a box. To draw a line in a box:

\begin{verbatim}
lineib (0, 0, 500, 500, b);    /* draws a line in box variable b */
\end{verbatim}

or

\begin{verbatim}
addbuffbox (0, 0, 500, 500, b);    /* adds a line to the buffer */
emptybuffbox ();                 /* draws all lines in the buffer */
\end{verbatim}

lineib draws the line in the box immediately. addbuffbox adds the line to a memory buffer. The lines are drawn on the screen when the buffer is full or when emptybuffbox is called. The last method is faster when large amount of lines have to be drawn.

The coordinate system of a box is different from the screen coordinate system. The vertical axis points upwards instead of downwards and the coordinates are in floating point numbers instead of integers.

The default x-axis scale of a box is 1000 to 1000. The default scale of the y-axis is adjusted to the height of the box. The scale of the axes can be adjusted with the function setscale:

\begin{verbatim}
setscale (width, height);
\end{verbatim}
setscale (0, 700, -3000, 3000, b);

Result: the x-scale of box b is 0 to 700 and the y-scale is -3000 to 3000.

• Text

A special scale is required to draw text in a box. The function maketextbox creates a box and automatically adjust the scale for use of text functions.

To draw text in a box:

texbitl (1, "Hello world!", b); /* draws text on line 1 in box */
/* variable b */

or:

texline (1, "Hello world!", b); /* draws text on line 1 in box */
/* variable b */

The function texbitl uses a bit map font and texline uses a stroked font.

• Erasing the contents of a box

The function clearspace erases the contents of a box. Also the boxes that were drawn (totally) inside the box are erased.

clearspace (b); /* erase everything inside box b */

C.2.3 3D graphics

The WINDOW is the main data structure for 3D graphics. This structure was derived from the BOX.

The function makewindow creates a window. addtowindowlist stores the window in the global variable windowlist. The application automatically redraws windows that are in the window list. The example below defines the position and orientation of the window in the global coordinate system:

---

Program architecture 207
WINDOW win; /* declare window */
win = makewindow (100, 100, 350, 400, */ define box */
0.0, 0.0, 0.0, /* 3D position of window origin */
1.0, 0.0, 0.0, /* orientation of window X-axis */
0.0, 1.0, 0.0, /* orientation of window Y-axis */
0.0, 0.0, 1.0, /* orientation of window Z-axis */
1.0e+09); /* distance between window and eye */
addtowindowlist (win); /* add win to global variable */
/* windowlist */

## Drawing in a window

Use a window instead of a box to draw 3D geometry. The graphics library translates spatial geometry to a window, then to a box, and finally draws it on the screen. A box on the screen represents the window. The window X-axis and Y-axis coincide with the axes of the box coordinate system. The library has four types of coordinate systems:

- the screen coordinate system to define the upper left and lower right corner of the box,
- the box coordinate system to draw 2D geometry,
- the window coordinate system,
- and the global or world coordinate system to draw 3D geometry.

To draw a line in a window:

-line3d (0, 0, 0, 50, 50); /* draws a 3D line */
or

addtobuff3d (0, 0, 0, 50, 50); /* adds a line to the buffer */
emptybuff3d (); /* draws all lines in the buffer */

line3d draws the spatial line immediately. addtobuff3d adds the line to a memory buffer. The lines are drawn on the screen when the buffer is full or when emptybuff3d is called. The last method is faster when large amount of lines have to be drawn.

The library has three memory buffers for geometry: a buffer for 3D geometry, a buffer for 2D geometry in box coordinates, and a buffer for 2D geometry in screen coordinates. The buffers are used to store the geometry while it is translated from one coordinate system to another.

The function windowshift moves the origin position of the window:

windowshift (100, 0, 0, win); /* move window origin */

208 Program architecture
The translation vector is defined in the window coordinate system.

*windowzoom* changes the scale of the window and moves the origin. The window origin always remains in the centre of the window an box. This means that the minimum and maximum X-scale are equal in length. The same applies to the Y-scale.

*windowzoom* (-750, 250, 0, 0, win);

Result: The X-scale ranges from -500 to 500 and the window origin was moved -250 along the window X-axis.

### Colors

The colors of the boxes and the background are stored in global variables. The function *graphon* reads the initial values of these variables from an initialisation file (see paragraph C.2.1, initialising graphics). The programmer can change these colors in his application.

*foregroundcolor* /* color of lines, circles, etc. */
*backgroundcolor* /* background color, used by *clearspace* */
*popupcolor* /* extra (menu) color */
*alarmcolor* /* color of message boxes */
*highlightcolor* /* color of upper side of boxes */
*shadowcolor* /* color of lower side of boxes */

Twenty standard colors are available in MS Windows. Below is a list of values and the color they represent:

0   black
1   dark red
2   dark green
3   tan/dark yellow
4   dark blue
5   deep purple
6   dark turquoise
7   light grey
8   greenish-white
9   light blue/sky blue
10  off-white
11  middle grey
12  dark grey
13  red
14  green
15  yellow
16  middle blue
The global variable `dottedline` determines whether graphic items are
drawn in dotted lines or solid lines. `dottedline = TRUE` or `dottedline =
1` results in dotted lines, circles, etc. `dottedline = FALSE` or `dottedline =
0` results in solid lines.

C.3 User interface functions

The application user interface exists of three levels:

The first and lowest level is the command line. The user types
commands on the keyboard, and the application sends the keyboard
events to the command line. The application executes the commands
when the command line receives an 'enter'.

The second level are the menus. The user pushes the menu buttons
with the mouse and the buttons send (a series of) commands to the
command line. The user has to type less commands in this manner. He
or she doesn’t have to remember all commands either, because the
menus guide him.

The third and highest level are the modes. The user clicks and drags
the mouse in the windows and boxes. When a mode is switched on, the
mouse directly sends a command (or a series of commands) to the
command line. In this way the user can manipulate the contents of the
boxes and windows directly, without having to bother about com-
mands; the application understands what the user means.

C.3.1 Command line

The programmer has to create a file `usercomm.c`. This file `must` contain
all functions that are described in this paragraph.

The items on the command line can either be arguments or com-
mands. The next function checks if an item is a command or not. The
library puts an item in the argument list if an item is not a command.
First, `is_command` calls the function `is_standard_command` to check if
an item is a command that is defined in the library. Then it checks if it
is a user defined command.
Figure C.3-1: Algorithm to execute commands.

```c
BOOLEAN is_command (char *pc) {
    BOOLEAN returnvalue = FALSE;

    if (is_standard_command (pc))
        returnvalue = TRUE;
    if (strcmp ("usercommand1", pc) == 0)
        returnvalue = TRUE;
    if (strcmp ("usercommand2", pc) == 0)
        returnvalue = TRUE;
    ...
    ...
    return returnvalue;
}
```
The next function executes commands. First, it calls `exe_standard_command` to execute commands from the standard library. Then it executes user defined commands. The functions that execute a command can obtain arguments by calling the function `get_arglist (char *pc)`. The functions `getscreenpoint`, `getfloatpoint`, and `get3dpoint` can be used to get coordinates from the argument list.

```c
void exe_command (char *pc)
{
    exe_standard_command (pc);
    if (strncmp ("usercommand1", pc) == 0)
        userfunction1 ();
    if (strncmp ("usercommand2", pc) == 0)
        userfunction2 ();
    ...
    ...
}
```

The next function returns `TRUE` if a command has to be added to the undo-list:

```c
BOOLEAN is_undocommand (char *pc)
```

The library calls the next function to initialise the undo-file. The function should write the command ‘open’ and the filename of the current model to the undo-file.

```c
void user_initundo (FILE *undofile)
{
    /* filename is a global variable */
    fprintf (undofile, "open %s\n", filename);
}
```

The function `user_initredo` is empty.

```c
void user_initredo (void)
```

The next functions should contain or call the applications load and save routines.

```c
void user_load (char *path)
void user_save (char *path)
void user_savecurrentfile (void)
```

The next function should redraw the model. The function that writes the model to a *.DXF file calls this function.

```c
void user_printindxfile (void)
```
C.3.2 Menus

The function `readmenulist` reads menus from a file.

```c
readmenulist ("menu.dat"); /* read user interface */
```

The menu file has to be terminated with the keyword `end`. An example of a part of a menu file:

```c
menu main
  formula b = 20 + vertbitmap + 2*vertoffset + 2;
  formula c = maxscr;
  formula a = c - 12*horibitmap - 2*horoffset - 4;
  formula d = b + 10*(vertbitmap + 2*vertoffset + 2) + 2;
  visible 1
  active 1
  border down
  box a b c d
  button 1
      text "redraw"
      command "redraw"

  "

  end
  end
```

The keyword `menu` in the menu file has to be terminated by the keyword `end`. The menu name directly follows the keyword. Other keywords within the menu definition statement define the properties of the menu.

The keyword `box` defines the size and position of the menu. Four arguments follow the `box` statement. The first pair of arguments are the position of the upper left corner, the second pair are the position of the lower right corner. The coordinates of the box can be

Figure C.3-2: Global variables that are available to define menus.
defined by formulas, starting with the keyword *formula*, then the name of the formula followed by an is equal to sign, then the formula, and ending with a semicolon. The global variables *maxxscr*, *maxyscr*, *horibitmap*, *vertbitmap*, *horioffset*, and *vertoffset* can be used in the formulas. They define the size of the screen, of the bitmap font and of the amount of pixels between the characters of the bitmap font and the button borders. The buttons are separated from each other and from the menu border by one pixel.

The box statement is the only obligatory statement within the menu definition. The rest of the statements are optional.

The keyword *button* defines buttons in the menu. A *button* also has to be terminated by the keyword *end*. A number directly behind the *button* statement defines the position of the button in the menu. 1 stands for first button in the menu, 2 for second button, etc. The keyword *text* followed by double quotes and a text defines the text on the button. The keyword *command* followed by double quotes and a text defines the command that the button puts on the command line when it is pushed. These commands can be used to open or close other menus. The commands “activatemenu *<menuname>*”, “hidemenu *<menuname>*”, and “disablemenu *<menuname>*” can be used to open a menu, hide a menu, and disable the buttons of a menu. The keyword *color* is optional. The default color of the button is the menu background color. *color* followed by any of the global color variable names or an integer number defines the color of the button.

Other keywords to define a menu:

*visible* 0/1, default: 0. The menu is visible if the keyword *visible* is 1.

*active* 0/1, default: 0. The menu buttons work if active is 1.

*autobide* 0/1, default: 0. If *autobide* is 1, the menu automatically closes itself when the mouse is clicked outside the menu.

*border up/down/double/noborder*, default: *up*.

The default color of the menu is the background color. *color* followed by any of the global color variable names or an integer number defines the color of the menu.

*text* *<line number>* “type your menu text here”
C.3.3 Modes

The modes are defined in the menu file. Below an example of a mode definition is given:

```
mode 2dmode
down "2d getxys
"
end
```

The definition of a mode starts with the keyword `mode`, followed by the name of the mode. An `end` keyword has to close the mode definition. The keywords `up`, `down`, and `move` are optional. Double quotes and the command that has to be put on the command line at a mouse up, mouse down, or mouse move event follows these keywords. A mode can be switched on by the command “mode <modename>”.

C.4 Standard commands

The paragraphs below give a list of all standard commands:

- **File commands**

  - `newfilename` Create new file name.
  - `[load][open] <filename>` Open file.
  - `saveasmenu` Create and open the save_as menu.
  - `save <filename>` Save to file.
  - `saveyes <filename>` Save and overwrite.
  - `savecurrentfile` Save current file.
  - `openmenu` Create and open the open menu.
  - `chdir <argument>` Change directory.
  - `chfilename <filename>` Change global variable `file_func_filename`.
  - `drive <drive name>` Change disk drive.
  - `quit` Leave program.
  - `quityes` Leave program, no confirmation.
no, yes, ok
   Same as buttons in pop up boxes.

chdxfilename <filename>
   Change global variable filefunc_dxffilename.

saveasdxfmenu
   Create and open the save_as_DXF menu.

savedxf <filename>
   Save to DXF file.

savedxfyes <filename>
   Save and overwrite DXF file.

opendxf <screen point>
   Open DXF mode for a window. Everything, that is drawn in the window, is saved to DXF file in DXF mode.

closedxf
   Close DXF mode.

■ Menu commands

activatemenu <menu name>
   Show and activate menu.

hidemenu <menu name>
   Hide menu.

disablenuimenu <menu name>
   Switch buttons in menu off.

■ Mouse mode commands

mode <mode name>
   Set mouse mode.

getxsys
   get current position of mouse cursor in screen coordinates.

dumpbox <screen point>
   Start to dynamically draw a rectangle.

updddb <screen point>
   Continue to dynamically draw a rectangle.

cleardumpbox
   Stop to dynamically draw a rectangle.

■ Point operator commands

2D <screen point>
   Convert screen point to box/window coordinates.

3D <screen point>
   Convert screen point to box/window coordinates.
**get3Dp**  
Put the coordinates of the cursor in the global coordinate system on the command line.

-  
Apply behind an operand to convert from positive to negative and vice versa.

**x <value>**  
Apply behind an operand to change the x value of the operand, and change the position of the cursor in the global coordinate system.

**y <value>**  
Apply behind an operand to change the y value of the operand, and change the position of the cursor in the global coordinate system.

**z <value>**  
Apply behind an operand to change the z value of the operand, and change the position of the cursor in the global coordinate system.

**dx <value>**  
Apply behind an operand to change the x value of the operand, and change the position of the cursor in the global coordinate system.

**dy <value>**  
Apply behind an operand to change the y value of the operand, and change the position of the cursor in the global coordinate system.

**dz <value>**  
Apply behind an operand to change the z value of the operand, and change the position of the cursor in the global coordinate system.

**Screen function commands**

**scroff**  
Close scroll box.

**paint**  
Repaint user interface.
**Undo commands**

*undo [integer i]*

Undo one command, or undo *i* commands.

*redo [integer i]*

Redo one command, or redo *i* commands.

*initredo*

Initialise redo file.

*initundo*

Initialise undo file.

**Window commands**

*open <screen point>*

Opens a window.

*close <screen point>*

Closes a window.

*wres <screen point>*

Resize a window. The first point points out the window, the second and third point give the new size of the window.

*wshift <screen point>*

<2D point> <2D point>

Translate a window in the local coordinate system of the window. The first point points out the window, the second and third point are the translation vector.

*wzoom <screen point>*

<2D point> <2D point>

Zoom. The first point points out the window, the second and third point define the zoom box.

*wmin <screen point>*

Zoom in.

*wmax <screen point>*

Zoom out.

*origin <screen point>*

<3D point>

Define origin position of the window in the global coordinate system.

*wtop <screen point>*

Give window a top view.

*wfront <screen point>*

Give window a front view.
\textit{ws}ide \textless \text{screen point} > \quad \text{Give window a side view.}
\textit{wiso} \textless \text{screen point} > \quad \text{Give window a isometric view.}
\textit{wrotx} \textless \text{screen point} > \quad \textless \text{angle} > \quad \text{Rotate window around local x-axis.}
\textit{wroty} \textless \text{screen point} > \quad \textless \text{angle} > \quad \text{Rotate window around local y-axis.}
\textit{wrotx} \textless \text{screen point} > \quad \textless \text{angle} > \quad \text{Rotate window around local z-axis.}
\textit{weye} \textless \text{screen point} > \quad \textless \text{distance} > \quad \text{Define distance from window to eye.}

\textit{draxes} \quad \text{Draw the axes of the global coordinate system.}

\textit{drawgrid} \quad \text{Draw grid.}
\textit{gridon} \quad \text{Switch grid on.}
\textit{gridoff} \quad \text{Switch grid off.}
\textit{gridsize} \textless \text{value} > \quad \text{Define size of grid.}
In October 1998 the author of this thesis contributed to a reliability analysis course [viii] for the Mechanical Engineering Department of the Engineering and Construction Division of the Ministry of Transport, Public Works, and Water Management. This appendix contains his contribution to this course.
Cursus risico analyse voor werktuigbouwkundige ontwerpers

Voorwoord

Zoetermeer, oktober 1998

Beste lezer,

Dit diktaat maakt deel uit van een cursus risico analyse voor ontwerpers van de afdeling Werktuigbouw van de Bouwdienst Rijkswaterstaat. Ik heb getracht om dit boekje zo op te stellen, dat het ook voor niet-werktuigbouwkundigen interessant kan zijn. Ik denk daarbij aan collega’s van de afdeling Electrotechniek.

Over het algemeen beginnen boeken over risico analyse met één of meer hoofdstukken statistiek. Ik heb dat achterwege gelaten. Naar mijn mening is voor het uitvoeren van ‘ons soort’ risico analyses nauwelijks kennis van statistiek noodzakelijk. Het opnemen van een hoofdstuk hierover zou hoogstens kunnen bewerkstelligen dat de lezer voortijdig afhaakt vanwege de taatje materie.

Wel worden de drie belangrijkste risico analyse technieken behandeld: faalvorm-, foutenboom- en gebeurtenissenboomanalyse. Tenslotte wordt aangegeven hoe met de voor de afdeling Werktuigbouw ontwikkelde software een betrouwbaarheidsanalyse uitgevoerd kan worden.

Dit diktaat staat niet op zichzelf. Het moet gezien worden in samenhang met het werk van J.T. de Vries [iii], R.C.A. Beem [iv] en de TAW [xi], [xii]. En L. Claes heeft een programma-handleiding geschreven bij de software.

Ik hoop dat met deze cursushandleiding een bijdrage kan leveren aan het verder invoeren van risico analyse in het ontwerpproces bij de Bouwdienst.

met vriendelijke groeten,

Gerard Avontuur
Inhoud

1 Inleiding 225

2 Failure modes analysis 227
   Inventarisatie 227
   Voorbeeld 227
   Kwaliteitsplan 229
   Welke faalvormen zijn ernstig? 229

3 Fault tree analysis 230
   3.1 Foutenboom opstellen 230
      Topgebeurtenis 230
      Voorbeeld 231
      Basisgebeurtenis 232
      Symbolen 234
      Booleaanse algebra 234
      Minimale deelverzameling 236
   3.2 Faalkans berekenen 236
      Afhankelijk falen, Common cause failure 237
      Niet beschikbaarheid 238

4 Event tree analysis 239
   Voorbeeld 240

5 Software voor foutenboomanalyse 241
   Hypothese 242
   Voorbeeld 243

6 Literatuur 245

A Faaldata componenten 246
A.1 Hydraulische componenten 246
A.2 Mechanische componenten 248
A.3 Electrische componenten 248
1 Inleiding

Het goed functioneren van bruggen, sluizen, stuwen en beweegbare waterkeringen is van groot maatschappelijk belang. Ondanks alle aandacht, die aan het ontwerpen van deze kunstwerken geschonken wordt, kan er wel eens iets mis gaan. Dat illustreert het kranteartikel hieronder.


In het verleden werd een risico analyse toegepast als een toets achteraf, als het ontwerp al bijna vastgelegd was. De analyse werd uitgevoerd door externe experts. Bijvoorbeeld van Kema. In dit stadium van het proces was het niet meer mogelijk om verregaande veranderingen in het ontwerp aan te brengen. Als een constructie niet voldeed aan de faalkans-eisen, werd niet het ontwerp, maar de risico analyse aangepast, zodat het ontwerp binnen de norm viel.

Deze gang van zaken is wel begrijpelijk. Uit de hoofdstukken hierna zal blijken, dat een risico analyse een zeer arbeidsintensieve aangelegenheid is. Daarnaast ontbrak het de ontwerper aan kennis om een risico analyse uit te voeren.
Echter, een risico analyse leent zich juist goed om keuzen te maken in het begin van het ontwerpproces. De ontwerper kan daardoor zijn beslissingen baseren op andere criteria dan sterkte en kosten alleen. De prijs-kwaliteit verhouding van verschillende ontwerpvarianten kan in beeld gebracht worden. Het resultaat van een risico analyse is niet zo zeer de faalkans van een constructie. Het is veel belangrijker om aan te geven hoe met geringe kosten een grote verbetering in het ontwerp doorgevoerd kan worden.

Deze cursus dient om de hierboven beschreven hindernissen voor het toepassen van een risico analyse weg te nemen. Door de ontwerper op te leiden heeft hij in ieder geval genoeg kennis om een risico analyse uit te voeren.

In deze cursus komen de volgende technieken aan de orde:

1 Faalvormanalyse of ‘failure modes analysis’, een techniek om te inventariseren welke ongewenste gebeurtenissen kunnen optreden.

2 Foutenboomanalyse of ‘fault tree analysis’, om na te gaan hoe het falen van deelsystemen en componenten kan leiden tot één bepaalde ongewenste gebeurtenis.

3 Gebeurtenissenboomanalyse of ‘event tree analysis’, om de toestanden waarin een systeem terecht kan komen op te sporen.

Daarnaast is er een computergereedschap ontwikkeld, waarmee de benodigde tijd om een foutenboomanalyse uit te voeren gereduceerd kan worden. Het gebruik van deze programmatuur wordt in deze cursus toegelicht.

2 Failure modes analysis

Het vakgebied risico analyse is doorspekt met Engelse termen. De verschillende analyse technieken hebben vaak Engelse namen. Eén ervan is Failure Modes Analysis (FMA). In het Nederlands: Faalvormanalyse.

■ Inventarisatie

Het doel van deze analyse is een inventarisatie van alle faalmogelijkheden van een installatie. Recept: verdeel een constructie in onderdelen. Som vervolgens van alle onderdelen de mogelijkheden van falen op. Het resultaat is een lijst met onderdelen en faalvormen.

FMA wordt vooral toegepast op grote, complexe constructies. Denk hierbij aan een sluizencomplex, een chemische fabriek, een kerncentrale.

■ Voorbeeld

Figuur 2-1 geeft een overzicht van de sluizen in Terneuzen. Op een sluizencomplex kan veel misgaan. Een failure modes analysis heeft tot doel om een zo volledig mogelijke opsomming van alle misser te maken. Er mogen geen zaken over het hoofd gezien worden. Een FMA is systematische aanpak, die daarbij helpt.

Het sluizencomplex wordt opgedeeld in onderdelen: Westsluis, Middensluis, Oostsluis, bedieningsgebouw, etc. Soms kunnen de onderdelen verder opgedeeld worden in subonderdelen. De Westsluis wordt in dit voorbeeld opgedeeld in een

Figuur 2-1: Sluizencomplex in Terneuzen.

buitenhoofd, een tussenhoofd, een binnenhoofd, basculebruggen over het buitenhoofd en binnenhoofd, luchtbellenschermen, etc. Eén en ander wordt weergegeven in lijsten (een spreadsheet-programma kan hierbij een belangrijk hulpmiddel zijn).

<table>
<thead>
<tr>
<th>Onderdeel</th>
<th>Sub-onderdeel</th>
<th>Faalvorm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Westsluis</td>
<td>Buitenhoofd</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tussenhoofd</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Binnenhoofd</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Basculebrug over het buitenhoofd</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Basculebrug over het binnenhoofd</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Luchtbellenschermen</td>
<td></td>
</tr>
<tr>
<td>Middensluis</td>
<td>.........</td>
<td></td>
</tr>
<tr>
<td>Oostsluis</td>
<td>.........</td>
<td></td>
</tr>
<tr>
<td>Bedieningsgebouw</td>
<td>.........</td>
<td></td>
</tr>
<tr>
<td>.........</td>
<td>.........</td>
<td></td>
</tr>
</tbody>
</table>

_Tabel 2-1: (Deel van) een lijst met onderdelen van het sluizencomplex._

De hierboven afgedrukte lijst is slechts een voorbeeld. Naar wens kunnen kolommen toegevoegd of weggelaten worden. Zo is het denkbaar om nog een kolom met sub-sub-onderdelen toe te voegen. De sluishaofden bestaan namelijk uit meerdere onderdelen. Zo hebben het binnen- en buitenhoofd elk twee() roldeuren. Aan iedere roldeur zit een bewegingswerk met een elektrische installatie. Het is ook mogelijk om kolommen weg te laten. Voor een FMA van een eenvoudige constructie is wellicht geen kolom met sub-onderdelen noodzakelijk.

<table>
<thead>
<tr>
<th>Onderdeel</th>
<th>Sub-onderdeel</th>
<th>Faalvorm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Westsluis</td>
<td>Buitenhoofd</td>
<td>sluisdeur opent niet</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sluisdeur sluit niet</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sluisdeur opent spontaan</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sluisdeur sluit spontaan</td>
</tr>
<tr>
<td></td>
<td></td>
<td>er kan niet genivelleerd worden</td>
</tr>
<tr>
<td></td>
<td></td>
<td>er wordt spontaan genivelleerd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.........</td>
</tr>
</tbody>
</table>

_Tabel 2-2: (Deel van) een lijst met faalvormen van een onderdeel._
Per onderdeel kan nu naar faalvormen gezocht worden:

1 Een onderdeel faalt als het onverwacht zijn functie niet meer vervult. Voorbeelden: sluisdeur sluit niet, sluisdeur opent niet.

2 Of als het spontaan zijn functie wel vervult, op hol slaat. Voorbeelden: sluisdeur sluit spontaan, sluisdeur opent spontaan.

### Kwaliteitsplan

Het resultaat van een volledige FMA is een zeer grote lijst met faalvormen. Tegen sommige vormen kunnen maatregelen genomen worden om de kans van optreden te minimaliseren. Missers die vaak voorkomen en die grote gevolgen hebben, moeten het eerst aangepakt worden. Bij het opstellen van het kwaliteitsplan, een lijst met kritieke eigenschappen en onderdelen kan een FMA een belangrijke rol spelen.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Westsluis</td>
<td>Buitenhoof</td>
<td>sluisdeur</td>
<td>grote zeeschappen kunnen niet geschut worden: economische schade</td>
<td>7</td>
<td>5</td>
<td>4.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>operent niet</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>sluisdeur</td>
<td>grote zeeschappen kunnen niet geschut worden: economische schade, Er is kans dat de waterstand op het kanaal te hoog of te laag wordt: economische schade, verzakking van de waterkering: kans op grote economische schade en het verlies van mensenlevens</td>
<td>9</td>
<td>5</td>
<td>5.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sluit niet</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


Reliability analysis course manual for mechanical engineers 229
Welke faalvormen zijn ernstig?

Uit tabel 2-2 blijkt nog niet welke faalvormen ernstig zijn en welke niet. Ook blijkt niet hoe vaak bepaalde zaken voorkomen. Om dit in kaart te brengen, kan de lijst uitgebreid worden met extra kolommen. Bijvoorbeeld kolommen om het effect van het falen te omschrijven en hoe vaak een bepaalde gebeurtenis optreedt. Het effect en de frequentie van optreden kunnen in woorden omschreven worden, maar het is ook mogelijk om er een numerieke waarde aan te geven. Bijvoorbeeld 0 voor geen effect en 9 voor een calamiteit met grote economische schade, die mensenlevens kost. En waarde 1 voor een faalvorm die weinig voorkomt en 10 voor één die veel voorkomt. In de literatuur worden voor dergelijke analyses de termen FMEA (Failure Modes and Effects Analysis) en FMECA (Failure Modes, Effects and Critcallity Analysis) gebruikt.

3 Fault tree analysis

Een Fault Tree Analysis (FTA) of foutenboomanalyse beantwoordt de vraag: ‘hoe groot is de kans dat een installatie één bepaalde functie niet kan vervullen?’. Een FTA behandelt één bepaalde faalvorm van een installatie. Deze analyse geeft inzicht in de (on)betrouwbareheid van de installatie voor wat betreft die ene faalvorm. De resultaten geven aan welke componenten van een systeem de grootste bijdrage aan de faalkans leveren.

In dit hoofdstuk wordt eerst aangegeven hoe een foutenboom opgesteld wordt. Daarna wordt uitgelegd hoe met behulp van de foutenboom de faalkans van een constructie berekend kan worden.

3.1 Foutenboom opstellen

Topgebeurtenis

Recept: Traceer de plaats van het ongemak. Beschouw vervolgens alle aangrenzende deelsystemen, die een bijdrage leveren aan de juiste werking van het systeem. Componen-
ten, waarvan het falen leidt tot het optreden van de topgebeur-
tenis, krijgen een plaats in de foutenboom. Bedenk dat een component kan falen door een storing in zichzelf of door het falen van een aangrenzend deel-
systeem. Voorbeeld: een elek-
triciteitsleiding kan falen door-
dat hij gebroken is, maar ook doordat er door falen elders in het systeem geen spanning op staat. Doorloop systematisch het hele systeem. Citaat: ‘Volg de koperen weg.’ [R.W. van Otterloo (KEMA)].

**Voorbeeld**

Het systeem in figuur 3-2 pompt via twee redundante leidingen olie van vat D in vat E. Er is sprake van falen als er geen vloeistof in vat E stroomt.

De topgebeurtenis wordt bij het opstellen van de fouten-
boom gedefinieerd als ‘geen vloeistofstroom in vat E’. Hiervoor moet er zowel geen olie uit leiding B als uit C komen. De foutenboom begint daarom met een ‘en’-poort met als linkertak ‘geen olie uit lei-
ding B’ en als rechtertak ‘geen olie uit leiding C’.

Figuur 3-2: Hydraulisch systeem.

Figuur 3-3a: Stap 1 bij het construeren van de foutenboom van het systeem in figuur 3-1.
Figuur 3.3b: Stap 2.

Figuur 3.3c: Stap 3.
Figuur 3.3d: Stap 4, de foutenboom is voltooid.

■ Basisgebeurtenis


De gebeurtenis 'geen toevoer naar leiding B' kan ontstaan doordat de pomp faalt of doordat er geen toevoer aan de pomp is, omdat vat D leeg is. Onder de gebeurtenis 'geen toevoer naar leiding B' wordt een
'of'-poort opgenomen, die de basisgebeurtenissen 'pomp stuk' en 'vat D leeg' verbindt.

Analoog aan de hierboven beschreven methode kan het deel van de foutenboom onder 'geen olie uit leiding C' opgebouwd worden.

■ Symbolen

Bij het tekenen van foutenbomen wordt gebruik gemaakt van symbolen. De symbolen en hun betekenis zijn in de figuur hiernaast aangegeven.

![Symbolen](image)

*Figuur 3-4: In foutenbomen gebruikte symbolen.*

■ Booleaanse algebra

Door de foutenboom te vereenvoudigen kan een beter inzicht in het falen van een systeem verkregen worden.

![Booleaanse algebra](image)

*Figuur 3-5: Vertaling van een 'of'-poort naar een vergelijking.*

*Figuur 3-6: Vertaling van een 'en'-poort naar een vergelijking.*

*Figuur 3-7: Voorbeeld foutenboom.*
worden. De boom wordt hiervoor eerst omgezet in een Gebeurtenissen, die met een ‘of’-poort verbonden zijn, worden in de vergelijking met een ‘+’ teken geschreven.

Gebeurtenissen die met een ‘en’-poort verbonden zijn, worden in de vergelijking met een ‘*’ teken geschreven.

Voorbeeld: figuur 3-7 geeft schematisch de foutenboom uit figuur 3-3d weer. Hieronder is deze boom als een vergelijking te geschreven:

Topgebeurtenis = ((D + A) + B)*((D + A) + C) \hspace{1cm} (3-1)

De vergelijking, die op deze manier ontstaat, kan vereenvoudigd worden met de volgende rekenregels:

\[
\begin{align*}
A + A &= A \hspace{1cm} (3-2a) \\
A*A &= A \hspace{1cm} (3-2b) \\
A + A*B &= A \hspace{1cm} (3-2c)
\end{align*}
\]


Een vergelijking moet zover mogelijk uitgewerkt (vermenigvuldigd) worden om deze regels toe te kunnen passen. De vergelijking van het voorbeeld (3-1) kan geschreven worden als:

Topgebeurtenis = D*D + A*D + C*D + 
+ A*D + A*A + A*C + 
+ B*D + A*B + B*C \hspace{1cm} (3-3)

Hieronder wordt vergelijking (3-3) met behulp van van rekenregels (3-2a t/m c) vereenvoudigd. De termen ‘A*A’ en ‘D*D’ in (3-3) mogen volgens rekenregel (3-2a) vervangen worden door ‘A’ en ‘D’:

Topgebeurtenis = D + A*D + C*D + 
+ A*D + A + A*C + 
+ B*D + A*B + B*C \hspace{1cm} (3-4a)

De termen ‘D + A*D’ en ‘A + A*D’ in (3.4a) mogen volgens rekenregel (3-2c) vervangen worden door ‘D’ en ‘A’:
Topgebeurtenis = D + C*D + 
+ A + A*C + 
+ B*D + A*B + B*C

(3-4b)

De termen ‘D + C*D’ en ‘A + A*C’ in (3.4b) mogen volgens rekenregel (3-2c) vervangen worden door ‘D’ en ‘A’:

Topgebeurtenis = D + A + B*D + A*B + B*C = 
= D + B*D + A + A*B + B*C

(3-4c)

De termen ‘D + B*D’ en ‘A + A*B’ in (3.4c) mogen volgens rekenregel (3-2c) vervangen worden door ‘D’ en ‘A’:

Topgebeurtenis = D + A + B*C

(3-4d)

geen vloeistofstrom
in vat E

of

of

en

vat D
leeg

pomp A
stuk

leiding B
verstopt

leiding C
verstopt

Figuur 3-8: Vereenvoudigde foutenboom voor de constructie in figuur 3-2.

■ Minimale deelverzameling

Het resultaat van deze exercitie is een aantal minimale deelverzamelingen of ‘minimal cut sets’ Een minimale deelverzameling bevat een zo klein mogelijk aantal componenten en veroorzaakt falen van het totale systeem. In het voorbeeld zijn \{A\}, \{D\}, en \{B, C\} minimale deelverzamelingen.

3.2 Faalkans berekenen

alsof (3-4d) een gewone vergelijking is (d.w.z. ‘+’ staat voor optellen en ‘×’ voor vermenigvuldigen):

1. De vergelijking is geschreven in de meest vereenvoudigde vorm.
2. De faalkansen zijn klein.
3. Het falen van de componenten is onafhankelijk.

Let op! Faalkansen van componenten worden in de literatuur in twee vormen gegeven: als faalkans per vraag en als faaltempo. De faalkans per vraag is de kans dat een component weigert, als er een beroep op gedaan wordt. Het kansgetal is eenheidsloos. Een faaltempo geeft de faalkans per bedrijfsuur aan. De dimensie is dus ‘per tijdseenheid’. Beide kansgetallen zijn in elkaar om te rekenen. Reken in een foutenboom altijd met eenheidsloze grootheden (= faalkans per vraag)!

\[ P_{\text{per vraag}} = P_{\text{faaltempo}} / n \]  
(3-5)

\( n \) is het aantal vragen per tijdseenheid.

### Afhankelijk falen, Common cause failure

Het falen van twee componenten is onafhankelijk als het falen van de ene component geen invloed heeft op het falen van de ander. Vaak is dit niet zo.

Bijvoorbeeld bij de (vier) bouten waarmee een wiel aan een auto vastzit. Als een bout bezwijkt wordt de belasting op de andere bouten hoger. Daardoor neemt de kans dat één van de andere bouten bezwijkt toe.

Componenten kunnen ook falen door één gemeenschappelijke oorzaak. Bijvoorbeeld brand of een constructieonderhouds-, of montagefout die consequent in alle onderdelen is doorgevoerd. Falen door een gemeenschappelijke oorzaak wordt ‘common cause failure’ genoemd.

Bij afhankelijk falende componenten mogen de kansen bij een ‘en’-poort niet meer vermenigvuldigd worden. Om afhankelijk falen in rekening te brengen kan het beta factor model gebruikt worden. Eigenlijk geldt het beta factor model alleen voor identieke componenten, die afhankelijk falen door een gemeenschappelijke oorzaak. Maar bij gebrek aan beter kan het ook in andere situaties toegepast worden.

<table>
<thead>
<tr>
<th>Type component</th>
<th>( \beta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dieselgeneratoren</td>
<td>0.05</td>
</tr>
<tr>
<td>Motor gestuurde kleppen</td>
<td>0.08</td>
</tr>
<tr>
<td>Terugslagkleppen</td>
<td>0.06</td>
</tr>
<tr>
<td>Koelwaterpompen</td>
<td>0.03</td>
</tr>
<tr>
<td>Koelmachines</td>
<td>0.11</td>
</tr>
<tr>
<td>Ventilatoren</td>
<td>0.13</td>
</tr>
<tr>
<td>150 kV circuits</td>
<td>0.18</td>
</tr>
<tr>
<td>150 kV rail</td>
<td>0.36</td>
</tr>
</tbody>
</table>

*Tabel 3-1: \( \beta \) - factoren voor verschillende componenten.*
\[ \beta = \frac{P_{cc}}{P_{cc} + P_{comp}} \quad (3-6) \]

\( P_{cc} \) is de gemeenschappelijke faalkans van twee componenten en \( P_{comp} \) is de individuele faalkans van een component. \( P_{comp}/(1 + P_{comp}) \leq \beta < 1 \). Uit (3-5) volgt:

\[ P_{cc} = \frac{\beta}{1 - \beta} \cdot P_{comp} \quad (3-7) \]

Aanname: de kansen B en C uit formule (3-4d) zijn afhankelijk. Dan volgt uit (3-4d) en (3-6):

\[ \text{Topgebeurtens} = D + A + \beta/(1 - \beta) \cdot B \quad (3-8) \]

Tabel 3-1 van de KEMA [xi] geeft waarden voor \( \beta \) van enkele componenten. Kema adviseert om een waarde 0.1 te nemen, als geen factoren voor \( \beta \) bekend zijn.

**Niet beschikbaarheid**

Uit een foutenboom volgt uiteindelijk de faalkans van een systeem voor één bepaalde faalvorm. Ook wordt duidelijk welke componenten de grootste bijdrage aan de faalkans leveren.

Nadat een systeem gefaald heeft, moet het gerepareerd worden. Daardoor is het niet beschikbaar voor gebruik. Daarnaast kan een systeem door onderhoud niet beschikbaar zijn. De beschikbaarheid of 'availability' is voor de beheerder van een kunstwerk van belang. Met behulp van een foutenboom kan de verwachtingswaarde van de niet beschikbaarheid ten gevolge van falen berekend worden.

De verwachtingswaarde van de niet beschikbaarheid ten gevolge van falen \( E_{cut\ set} \) van een minimale deelverzameling kan berekend worden met de kans van optreden per vraag \( P_{cut\ set} \), het aantal vragen per tijdseenheid \( n \) en de reparatietijd \( t_{reparatie} \):

\[ E_{cut\ set} = P_{cut\ set} \cdot n \cdot t_{reparatie} \quad (3-9) \]

Voorbeeld:

\[ E_{cut\ set} = 1.0 \times 10^3 [1/\text{vraag}] \times 0.0799 [\text{vragen}/\text{h}] \times 24 [\text{h}] = 1.9 \times 10^3. \]

De verwachtingswaarde van een systeem \( E_{systeem} \) is de som van de verwachtingswaarden van de minimale deelverzamelingen.

\[ E_{systeem} = \sum E_{cut\ set} \quad (3-10) \]
Voorbeeld:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pomp A stuk</td>
<td>1.0*10^-3</td>
<td>0.0799</td>
<td>24</td>
<td>1.9*10^-3</td>
</tr>
<tr>
<td>Vat D leeg</td>
<td>1.0*10^-6</td>
<td>0.0799</td>
<td>2</td>
<td>1.6*10^-7</td>
</tr>
<tr>
<td>Leiding B en leiding C verstopt</td>
<td>1.1*10^-6</td>
<td>0.0799</td>
<td>24</td>
<td>2.1*10^-6</td>
</tr>
<tr>
<td>Totaal</td>
<td></td>
<td></td>
<td></td>
<td>1.9*10^-3</td>
</tr>
</tbody>
</table>

*Tabel 3-2: Voorbeeld berekening van de verwachtingswaarde van de niet beschikbaarheid.*

4 Event tree analysis

Een event tree analysis of gebeurtenissenboomanalyse laat zien hoe een systeem blijft functioneren na het falen van deelsystemen of componenten. Het resultaat is een aantal toestanden waarin het systeem terecht kan komen. De analyse is vooral geschikt als er meerdere toestanden zijn. Bijvoorbeeld: 'werkt volledig', 'werkt een beetje', of 'werkt helemaal niet meer'.

Recept: Verdeel de constructie in deelsystemen of componenten. Bedenk van alle componenten faalvormen. Het optreden van een faalvorm is een gebeurtenis. Zet de gebeurtenissen langs een horizontale as. Teken horizontaal een binaire boom. Bij iedere gebeurtenis splitst de boom zich in een

*Figuur 4-1: Centrifugaalpomp in gemaal Leemans. (Oude situatie.)*

*Figuur 4-2: Model van een gemaal.*
tak met ‘ja’ en een tak met ‘nee’. Uiteindelijk leiden de takken tot een aantal toestanden. Hiervan kan de kans van optreden berekend worden. Eén en ander wordt met een voorbeeld duidelijk gemaakt.

**Voorbeeld**

Een gemaal kan op verschillende manieren falen. De eerste functie van een gemaal is het pompen van water. Een gemaal kan falen om deze functie te vervullen, doordat het door een storing slechts een gedeelte van het maximale debiet kan opbrengen. Ook geheel niet kunnen pompen is een faalvorm. Daarnaast heeft een gemaal een functie als waterkering. Lekkage betekent falen met betrekking tot deze functie.

In figuur 4-2 is een model van een gemaal gegeven. Met een gebeurtenissenboomanalyse kunnen de toestanden, waarin het gemaal terecht kan komen, opgespoord worden.

Figuur 4-3 geeft de gebeurtenissenboom van dit gemaal weer. Bovenaan de figuur staan de faalvormen van de componenten. Het falen van een component wordt als gebeurtenis aangemerkt. In een boom wordt systematisch het wel en het niet optreden van de gebeurtenissen beschouwd. Wel optreden is een tak naar boven, niet naar beneden.

Het blijkt dat na het optreden van de gebeurtenis ‘afsluiter dicht’ de andere gebeurtenissen geen invloed meer hebben op de toestand waarin het gemaal komt. Door deze gebeurtenis als eerste in op te nemen wordt de grootte van de boom beperkt (ga dit na).

Met behulp van de gebeurtenissenboom kan de kans uitgerekend worden, dat een systeem in een bepaalde toestand terechtkomt. De kans dat het gemaal de helft van het maximale debiet levert, is de som van de kansen die horen bij alle takken, waar ‘half debiet’ bij staat. De kans van de tak met het vette label is gelijk aan:

\[
P_{vet\ label} = (1 - P_{afsluiter\ dicht}) \times (1 - P_{pomp\ A\ start\ niet}) \times (1 - P_{pomp\ B\ start\ niet}) \times (1 - P_{terugslagklep\ A\ open\ niet}) \times (1 - P_{terugslagklep\ B\ open\ niet})
\]

De som van de kansen van alle takken is gelijk aan 1.0.
Figuur 4-3: Gebeurtenissenboom van het model in figuur 4-2.

5 Software voor foutenboomanalyse

In de voorgaande hoofdstukken zijn drie risico analyse technieken behandeld. De foutenboomanalyse is de meest toegepaste techniek. Het opstellen van de foutenboom en het uitwerken ervan met behulp van booleaanse algebra is zeer tijdrovend. Daarnaast is het met de hand uitvoeren van deze taak foutgevoelig. Daarom is voor de afdeling Werk-
tuigbouw van de Bouwdienst software ontwikkeld, die de ontwerper hierbij helpt.

![Diagram](image)

*Figuur 5-1: Model van een bewegingswerk van een sluisdeur.*

De programma's *G-Edit* en *FemFault* ondersteunen de ontwerper bij het uitvoeren van de betrouwbaarheids-analyse van bewegingswerken. *G-Edit* is een Grafische componenten Editor. Dat is een programma, waarmee een bewegingswerk gemodelleerd kan worden door het aan elkaar 'klikken' van componenten. Denk hierbij aan: motoren, assen, koppelingen, pomp en terugslagkleppen, etc. *FemFault* kan een betrouwbaarheidsanalyse uitvoeren van een constructie, die met *G-Edit* gemodelleerd is.

### Hypothese

<table>
<thead>
<tr>
<th>Een bewegingswerk heeft twee functies:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  Het dragen van een belasting.</td>
</tr>
<tr>
<td>2  Het uitvoeren van een beweging.</td>
</tr>
</tbody>
</table>

Voor beide functies moeten in *G-Edit* aparte modellen gemaakt worden. In het model moet de functie van de constructie aangegeven worden. Een functie van een bewegingswerk is het overwinnen van de belasting die op een sluisdeur staat. Deze functie wordt aangegeven door een belasting op de deur te plaatsen. Een andere functie is het bewegen van de deur. Dit wordt gemodelleerd door de verplaatsing van de deur voor te schrijven.

Voor de functie 'het dragen van een belasting' wordt een model gemaakt met een kracht op de aan te drijven constructie. De motor of hydrauliekpomp worden voorgesteld als starre elementen. Een motor wordt voorgesteld als een rotatiegenerator en een pomp als debietgenerator.

Voor de functie 'het uitvoeren van een beweging' wordt er een verplaatsing van de aan te drijven constructie voorgeschreven. De motor of
hydrauliekpomp worden voorgesteld als elementen, die een kracht kunnen uitoefenen of druk kunnen genereren. Een motor wordt voorgesteld als een koppelgenerator en een pomp als drukgenerator.

![Bewegingswerk van een sluisdeur](image)

**Figuur 5-2:** Bovenaanzicht van een model van een bewegingswerk van een sluisdeur. Dit model is bedoeld voor analyse van de betrouwbaarheid van de functie 'het dragen van een belasting'.

![Model voor analyse](image)

**Figuur 5-3:** Model voor analyse van de betrouwbaarheid van de functie 'het uitvoeren van een beweging'.

![3D aanzicht](image)

**Figuur 5-4:** 3D aanzicht van hydraulische variant 1.

**Voorbeeld**

De figuren in dit hoofdstuk geven vier varianten van een bewegingswerk van een sluisdeur weer. De eerste variant is een elektro-
mechanische aandrijving met een heugel. Vervolgens worden drie hydraulische varianten getoond.

Figuur 5-5: Bovenaanzicht van hydraulische variant 1.

Figuur 5-6: Hydraulische variant 2.

Figuur 5-7: Hydraulische variant 3.

<table>
<thead>
<tr>
<th>variant</th>
<th>collapse analysis</th>
<th>motion analysis</th>
<th>totaal</th>
</tr>
</thead>
<tbody>
<tr>
<td>heugel</td>
<td>3.1*10^4</td>
<td>5.5*10^3</td>
<td>8.6*10^4</td>
</tr>
<tr>
<td>hydraulisch 1</td>
<td>6.7*10^3</td>
<td>2.7*10^3</td>
<td>2.8*10^3</td>
</tr>
<tr>
<td>hydraulisch 2</td>
<td>1.3*10^4</td>
<td>2.2*10^3</td>
<td>2.3*10^3</td>
</tr>
<tr>
<td>hydraulisch 3</td>
<td>1.3*10^4</td>
<td>2.1*10^3</td>
<td>2.2*10^3</td>
</tr>
</tbody>
</table>

Tabel 5-1: Faalkansen per vraag van de varianten die hierboven zijn weergegeven.
Uit de analyse blijkt, dat de elektro-mechanische variant het meest betrouwbaar is.

Verder blijken de hydraulische varianten met dubbele pompen betrouwbaarder te zijn, dan die met een enkele pomp.

Door het dubbel uitvoeren van hydraulische componenten neemt de betrouwbaarheid van het dragen van de belasting af. Dit komt doordat er meer componenten zijn, die kunnen gaan lekken. De bewegingsanalyse blijkt echter maatgevend te zijn voor de faalkans. Daardoor neemt de totale betrouwbaarheid toe.

In tegenstelling tot wat men zou verwachten, leidt het toepassen van dubbele terugslagkleppen tot een lichte toename van de faalkans (bij de bewegingsanalyse)! Dit is te wijten aan het feit, dat er bij hydraulische variant 2 meer combinaties van falende componenten zijn, die tot falen van het totale systeem leiden (bewegingsanalyse).

6 Literatuur

[i] G.C. Avontuur; *Tussenverslag Geautomatiseerde Faalkansanalyse*; Bouwdienst Rijkswaterstaat, Afdeling Werktuigbouw; februari 1993

[ii] G.C. Avontuur; *Reliability analysis in mechanical engineering design*; Ministry of Transport, Public Works, and Water Management, RIJKSWATERSTAAT, Engineering and Construction Division, Mechanical Engineering Department; Delft University of Technology, Faculty of Mechanical Engineering and Marine Technology, Section Design; April 1995

[iii] J.T. de Vries; *Handboek onderhoud kunstwerken NL*; Bouwdienst Rijkswaterstaat, Afdeling Ontwikkeling Technieken Natte Infrastructuur; 1998


[vii] J.L. Spoormaker; *Ontwerpen op bedrijfszekerheid*; Technische Universiteit Delft; Faculteit Industrieel Ontwerpen; Sectie Constructie-leer; 1989

[viii] S.J. de Lange; *Kansrekening en statistiek*; Delftse Uitgeversmaatschappij; 1988


Reliability analysis course manual for mechanical engineers 245
A **Faaldata componenten**

In deze appendix zijn faaldata opgenomen. De gegevens komen uit de bronnen [iii], [xiii] en [xiv].

### A.1 **Hydraulische componenten**

<table>
<thead>
<tr>
<th>Component</th>
<th>Faalvorm</th>
<th>Kans per vraag</th>
<th>Faaltempo</th>
<th>Bron</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer (hydraulisch)</td>
<td>lek</td>
<td>-</td>
<td>5.0*10⁵/h</td>
<td>TNO</td>
</tr>
<tr>
<td>cilinder</td>
<td>lek</td>
<td>-</td>
<td>7.5*10⁵/h</td>
<td>RWS</td>
</tr>
<tr>
<td></td>
<td>lek</td>
<td>-</td>
<td>1.0*10⁵/h</td>
<td>RWS</td>
</tr>
<tr>
<td></td>
<td>vast</td>
<td>-</td>
<td>1.0*10⁴/h</td>
<td>RWS</td>
</tr>
<tr>
<td>drukschakelaar</td>
<td>uittwendige lek</td>
<td>-</td>
<td>1.0*10⁴/h</td>
<td>RWS</td>
</tr>
<tr>
<td></td>
<td>voortijdig ingrijpen</td>
<td>4.1*10⁵</td>
<td>1.3<em>10⁷/h - 1.3</em>10⁵/h</td>
<td>RWS</td>
</tr>
<tr>
<td>filter</td>
<td>uittwendige lekkage</td>
<td>-</td>
<td>1.0<em>10⁴/h - 2.0</em>10⁴/h</td>
<td>RWS, TNO</td>
</tr>
<tr>
<td></td>
<td>verstopping</td>
<td>7.5*10⁵</td>
<td>7.5*10⁵/h</td>
<td>RWS</td>
</tr>
<tr>
<td></td>
<td>verstopping</td>
<td>-</td>
<td>2.0*10⁴/h</td>
<td>RWS</td>
</tr>
<tr>
<td>klep (motor gestuurd)</td>
<td>opent niet</td>
<td>5.1*10³</td>
<td>-</td>
<td>TNO</td>
</tr>
<tr>
<td></td>
<td>sluit niet</td>
<td>6.0*10³</td>
<td>-</td>
<td>TNO</td>
</tr>
<tr>
<td></td>
<td>voortijdig open dicht</td>
<td>-</td>
<td>5.0<em>10⁷/h - 1.0</em>10⁴/h</td>
<td>RWS</td>
</tr>
<tr>
<td>klep (regel-, stuur-klep)</td>
<td>uittwendige lekkage</td>
<td>-</td>
<td>1.0<em>10⁴/h - 1.0</em>10⁴/h</td>
<td>RWS</td>
</tr>
</tbody>
</table>

246 **Reliability analysis course manual for mechanical engineers**
<table>
<thead>
<tr>
<th>Component</th>
<th>Faalvorm</th>
<th>Kans per vraag</th>
<th>Faaltempo</th>
<th>Bron</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>open voortijdig</td>
<td>-</td>
<td>1.0*10^4/h bedrijf</td>
<td>RWS</td>
</tr>
<tr>
<td></td>
<td>schakelt niet</td>
<td>3.0*10^5</td>
<td>-</td>
<td>Kema, RWS</td>
</tr>
<tr>
<td></td>
<td>open, sluit niet</td>
<td>1.0*10^4</td>
<td>1.0*10^7/h bedrijf</td>
<td>RWS</td>
</tr>
<tr>
<td></td>
<td>verstopt</td>
<td>1.0*10^4</td>
<td>4.0*10^9/h</td>
<td>RWS</td>
</tr>
<tr>
<td>(smoor-, rem-)</td>
<td></td>
<td>-</td>
<td>1.5*10^9/h</td>
<td>RWS</td>
</tr>
<tr>
<td>klep (terugslag)</td>
<td>uitwendige lekka-</td>
<td>-</td>
<td>1.0*10^9/h</td>
<td>RWS</td>
</tr>
<tr>
<td></td>
<td>open voortijdig</td>
<td>-</td>
<td>1.0*10^5/h bedrijf</td>
<td>RWS</td>
</tr>
<tr>
<td></td>
<td>schakelt niet</td>
<td>1.0*10^5</td>
<td>-</td>
<td>Kema</td>
</tr>
<tr>
<td></td>
<td>open, sluit niet</td>
<td>1.0*10^4</td>
<td>1.0*10^7/h bedrijf</td>
<td>RWS, TNO</td>
</tr>
<tr>
<td></td>
<td>open, verstopt</td>
<td>1.0*10^4</td>
<td>4.0*10^9/h</td>
<td>RWS, TNO</td>
</tr>
<tr>
<td>(veiligheid)</td>
<td>uitwendige lekka-</td>
<td>-</td>
<td>1.0*10^9/h</td>
<td>RWS</td>
</tr>
<tr>
<td></td>
<td>open voortijdig</td>
<td>1.0*10^4</td>
<td>1.0*10^5/h</td>
<td>Kema, RWS</td>
</tr>
<tr>
<td></td>
<td>open voortijdig</td>
<td>-</td>
<td>1.0*10^6/h bedrijf</td>
<td>RWS</td>
</tr>
<tr>
<td></td>
<td>sluit niet</td>
<td>-</td>
<td>1.0*10^7/h bedrijf</td>
<td>RWS</td>
</tr>
<tr>
<td></td>
<td>verstopt</td>
<td>-</td>
<td>4.0*10^8/h</td>
<td>RWS</td>
</tr>
<tr>
<td>kogelkranen</td>
<td>gesloten</td>
<td>7.5*10^6</td>
<td>7.2<em>10^8/h - 8.0</em>10^6/h</td>
<td>RWS</td>
</tr>
<tr>
<td></td>
<td>verstopt</td>
<td>-</td>
<td>5.0*10^7/h</td>
<td>RWS</td>
</tr>
<tr>
<td></td>
<td>uitwendige lekka</td>
<td>-</td>
<td>1.0*10^8/h</td>
<td>RWS</td>
</tr>
<tr>
<td>pakking</td>
<td>lekkage</td>
<td>-</td>
<td>5.0*10^7/h</td>
<td>TNO</td>
</tr>
<tr>
<td>plunjer</td>
<td>lekkage langs cilinder</td>
<td>-</td>
<td>5.0*10^7/h</td>
<td>TNO</td>
</tr>
<tr>
<td>pomp-motor</td>
<td>valt uit</td>
<td>-</td>
<td>3.0<em>10^5/h - 8.5</em>10^5/h</td>
<td>RWS, TNO</td>
</tr>
<tr>
<td>Component</td>
<td>Faalvorm</td>
<td>Kans per vraag</td>
<td>Faaltempo</td>
<td>Bron</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------</td>
<td>----------------</td>
<td>---------------------</td>
<td>-------</td>
</tr>
<tr>
<td>slang</td>
<td>valt uit</td>
<td>-</td>
<td>1.0*10⁶/h bedrijf</td>
<td>RWS</td>
</tr>
<tr>
<td></td>
<td>start niet</td>
<td>1.0<em>10⁻¹ - 4.8</em>10⁻⁵</td>
<td>-</td>
<td>RWS,</td>
</tr>
<tr>
<td></td>
<td>uitwendige lekkage, breuk</td>
<td>7.5*10⁴</td>
<td>1.0<em>10⁴/h - 4.0</em>10⁷/h</td>
<td>Kema,</td>
</tr>
<tr>
<td></td>
<td>uitwendige lekkage, breuk</td>
<td>-</td>
<td>5.0*10⁸/h bedrijf</td>
<td>RWS,</td>
</tr>
<tr>
<td></td>
<td>uitwendige lekkage</td>
<td>-</td>
<td>5.0*10⁷/h</td>
<td>TNO</td>
</tr>
</tbody>
</table>

### A.2 Mechanische componenten

<table>
<thead>
<tr>
<th>Component</th>
<th>Faalvorm</th>
<th>Kans per vraag</th>
<th>Faaltempo</th>
<th>Bron</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer (mechanisch)</td>
<td>veerbreuk</td>
<td>3.7*10⁶</td>
<td>-</td>
<td>RWS</td>
</tr>
<tr>
<td>heugel</td>
<td>tandbreuk</td>
<td>1.0*10⁷???</td>
<td>-</td>
<td>RWS</td>
</tr>
<tr>
<td>koppeling (flexibel)</td>
<td>breuk</td>
<td>3.7*10⁶</td>
<td>-</td>
<td>RWS</td>
</tr>
<tr>
<td>koppeling (star, vast)</td>
<td>breuk</td>
<td>7.4*10⁶</td>
<td>-</td>
<td>RWS</td>
</tr>
<tr>
<td>lier</td>
<td></td>
<td>-</td>
<td>7.9*10⁵/h</td>
<td>TNO</td>
</tr>
<tr>
<td>rondsels</td>
<td>tandbreuk</td>
<td>1.0*10⁷???</td>
<td>-</td>
<td>RWS</td>
</tr>
<tr>
<td>tandwielkast</td>
<td>blokkeert</td>
<td>6.3*10⁶</td>
<td>1.5*10⁷/h</td>
<td>Stork, TNO</td>
</tr>
<tr>
<td></td>
<td>tandbreuk</td>
<td>-</td>
<td>1.9*10⁴/h bedrijf</td>
<td>Stork</td>
</tr>
<tr>
<td></td>
<td>tandbreuk</td>
<td>-</td>
<td>1.3*10⁴/h</td>
<td>TNO</td>
</tr>
</tbody>
</table>

### A.3 Electrische componenten

<table>
<thead>
<tr>
<th>Component</th>
<th>Faalvorm</th>
<th>Kans per vraag</th>
<th>Faaltempo</th>
<th>Bron</th>
</tr>
</thead>
<tbody>
<tr>
<td>accubatterij</td>
<td>levert geen span-</td>
<td>6.6*10³</td>
<td>-</td>
<td>TNO</td>
</tr>
<tr>
<td></td>
<td>ning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>stopt voortijdig</td>
<td>-</td>
<td>1.9*10⁴/h</td>
<td>TNO</td>
</tr>
<tr>
<td>coax verbinding</td>
<td>maakt geen con-</td>
<td>-</td>
<td>1.7*10⁷/h</td>
<td>TNO</td>
</tr>
<tr>
<td></td>
<td>tact</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dieselgenerator</td>
<td>start niet</td>
<td>1.0*10²</td>
<td>-</td>
<td>TNO</td>
</tr>
<tr>
<td>Component</td>
<td>Faalvorm</td>
<td>Kans per vraag</td>
<td>Faaltempo</td>
<td>Bron</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------------------</td>
<td>----------------</td>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td>(gecombineerd)</td>
<td>1.7*10^2</td>
<td>-</td>
<td>TNO</td>
</tr>
<tr>
<td></td>
<td>start niet (zonder repeteer)</td>
<td></td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>stopt voortijdig</td>
<td></td>
<td>3.0*10^3/h</td>
<td>TNO</td>
</tr>
<tr>
<td></td>
<td>start niet (met repeteer)</td>
<td>1.8*10^3</td>
<td>-</td>
<td>TNO</td>
</tr>
<tr>
<td>drukknop</td>
<td>maakt geen contact</td>
<td>-</td>
<td>1.0*10^5/h</td>
<td>TNO</td>
</tr>
<tr>
<td>electromotor</td>
<td>start niet</td>
<td>3.0<em>10^4 - 3.0</em>10^4</td>
<td>-</td>
<td>RWS</td>
</tr>
<tr>
<td>GEB/LS aansluiting</td>
<td>valt uit</td>
<td>-</td>
<td>4.0*10^5/h</td>
<td>TNO</td>
</tr>
<tr>
<td>gelijkrichter</td>
<td>geeft geen gelijkspanning</td>
<td>-</td>
<td>1.8*10^6/h</td>
<td>TNO</td>
</tr>
<tr>
<td>(gest.vdg)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hoogspannings-lijn</td>
<td>110 - 220 kV valt uit</td>
<td>-</td>
<td>8.0*10^7/h</td>
<td>TNO</td>
</tr>
<tr>
<td></td>
<td>380 kV valt uit</td>
<td>-</td>
<td>3.0*10^7/h</td>
<td>TNO</td>
</tr>
<tr>
<td></td>
<td>50 kV valt uit</td>
<td>-</td>
<td>9.0*10^7/h</td>
<td>TNO</td>
</tr>
<tr>
<td>kabel (begraven)</td>
<td>breekt</td>
<td>-</td>
<td>1.0*10^6/km/h</td>
<td>TNO</td>
</tr>
<tr>
<td>kabel (bovengronds)</td>
<td>breekt</td>
<td>-</td>
<td>3.0*10^6/km/h</td>
<td>TNO</td>
</tr>
<tr>
<td>kabel (in goot)</td>
<td>breekt</td>
<td>-</td>
<td>1.2*10^7/km/h</td>
<td>TNO</td>
</tr>
<tr>
<td>kabel (signal)</td>
<td>breekt</td>
<td>-</td>
<td>1.0*10^5/km/h</td>
<td>TNO</td>
</tr>
<tr>
<td>luchtkoeler</td>
<td>start niet</td>
<td>2.1*10^4</td>
<td>-</td>
<td>TNO</td>
</tr>
<tr>
<td></td>
<td>stopt voortijdig</td>
<td>-</td>
<td>1.1*10^6/h</td>
<td>TNO</td>
</tr>
<tr>
<td>modem</td>
<td>komt niet on-line</td>
<td>-</td>
<td>3.0*10^6/h</td>
<td>TNO</td>
</tr>
<tr>
<td>omvormer</td>
<td>geeft geen wisselspanning</td>
<td>-</td>
<td>1.2*10^5/h</td>
<td>TNO</td>
</tr>
<tr>
<td>PLC</td>
<td>CPU valt uit</td>
<td>-</td>
<td>5.1*10^5/h</td>
<td>TNO</td>
</tr>
<tr>
<td></td>
<td>I/O kaart valt uit</td>
<td>-</td>
<td>9.0*10^6/h</td>
<td>TNO</td>
</tr>
<tr>
<td></td>
<td>geheugenmodule valt uit</td>
<td>-</td>
<td>1.0*10^5/h</td>
<td>TNO</td>
</tr>
<tr>
<td></td>
<td>voeding/spanning valt uit</td>
<td>-</td>
<td>2.9*10^5/h</td>
<td>TNO</td>
</tr>
<tr>
<td>relais</td>
<td>schakelt niet</td>
<td>7.6<em>10^4 - 3.0</em>10^4</td>
<td>-</td>
<td>Kema,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RWS,</td>
</tr>
</tbody>
</table>

Reliability analysis course manual for mechanical engineers 249
<table>
<thead>
<tr>
<th>Component</th>
<th>Faalvorm</th>
<th>Kans per vraag</th>
<th>Faaltempo</th>
<th>Bron</th>
</tr>
</thead>
<tbody>
<tr>
<td>schakelaar (druk)</td>
<td>schakelt niet</td>
<td>1.0<em>10^3 - 1.0</em>10^4</td>
<td>4.0<em>10^5/h - 6.0</em>10^3/h</td>
<td>Kema, RWS</td>
</tr>
<tr>
<td></td>
<td>lek</td>
<td></td>
<td>1.0*10^9/h</td>
<td>RWS</td>
</tr>
<tr>
<td></td>
<td>schakelt voortijdig</td>
<td></td>
<td>1.0*10^7/h bedrijf</td>
<td>RWS</td>
</tr>
<tr>
<td>schakelaar (eind)</td>
<td>open niet</td>
<td>3.0*10^6</td>
<td></td>
<td>TNO</td>
</tr>
<tr>
<td></td>
<td>open spontaan</td>
<td></td>
<td>1.0*10^5/h</td>
<td>TNO</td>
</tr>
<tr>
<td></td>
<td>schakelt niet</td>
<td>1.0<em>10^3 - 2.2</em>10^5</td>
<td></td>
<td>RWS</td>
</tr>
<tr>
<td></td>
<td>sluit niet</td>
<td>3.0*10^6</td>
<td></td>
<td>TNO</td>
</tr>
<tr>
<td>schakelaar (hand)</td>
<td>schakelt niet</td>
<td>3.0*10^6</td>
<td></td>
<td>TNO</td>
</tr>
<tr>
<td>schakelaar (spil-eindschakelaar)</td>
<td>open voortijdig</td>
<td></td>
<td>9.0*10^4/h</td>
<td>TNO</td>
</tr>
<tr>
<td>schakelaar (vermogen)</td>
<td>sluit niet</td>
<td>1.0*10^4</td>
<td></td>
<td>TNO</td>
</tr>
<tr>
<td></td>
<td>open spontaan</td>
<td></td>
<td>5.0*10^7/h</td>
<td>TNO</td>
</tr>
<tr>
<td>termische beveiliging</td>
<td>schakelt niet</td>
<td>3.0*10^1^3</td>
<td></td>
<td>TNO</td>
</tr>
<tr>
<td></td>
<td>open voortijdig</td>
<td></td>
<td>3.0*10^4/h</td>
<td>TNO</td>
</tr>
<tr>
<td></td>
<td>schakelt niet in</td>
<td>3.0*10^7</td>
<td></td>
<td>TNO</td>
</tr>
<tr>
<td>transformator</td>
<td>valt uit</td>
<td></td>
<td>1.0*10^4/h</td>
<td>TNO</td>
</tr>
<tr>
<td>veiligheid (smelt)</td>
<td>open niet</td>
<td>1.0*10^5</td>
<td></td>
<td>TNO</td>
</tr>
<tr>
<td></td>
<td>open spontaan</td>
<td></td>
<td>1.0*10^7/h</td>
<td>TNO</td>
</tr>
<tr>
<td>veiligheid (veerbelast)</td>
<td>open/sluit niet</td>
<td>1.5*10^4</td>
<td></td>
<td>TNO</td>
</tr>
<tr>
<td></td>
<td>sluit niet na openen</td>
<td>1.0*10^2</td>
<td></td>
<td>TNO</td>
</tr>
</tbody>
</table>
Curriculum vitae

Gerardus Cornelius Avontuur was born on May 30th 1968 in The Hague. In 1987 he finished primary education at the Interconfessioneel Westland College in Naaldwijk, the Netherlands, and started his study Mechanical Engineering at Delft University of Technology. He specialised in the field of Design Automation under the supervision of prof. Van der Werff and graduated in 1992. Since his graduation Avontuur is working at the Construction Division of the Ministry of Public Works, Transport, and Water Management (Bouwdienst Rijkswaterstaat). From 1992 to 1996 he worked at the Mechanical Engineering Department, and from 1996 until now he works at the Design Automation Department (Bouwinformatica).