Enabling domain experts to participate in the process of improving software quality using change impact analysis

Master's Thesis

Tim Nederveen

Enabling domain experts to participate in the process of improving software quality using change impact analysis

THESIS

submitted in partial fulfilment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Tim Nederveen
born in Dordrecht, the Netherlands

Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

Exact
Molengraaffsingel 33
Delft, the Netherlands
www.exact.com

Enabling domain experts to participate in the process of improving software quality using change impact analysis

Author: Tim Nederveen
Student id: 4575849
Email: ik@tim365.nl

Abstract

Software engineers often lack the domain knowledge needed to validate context specific parts of software. Domain experts do have this knowledge needed to validate the software, but often lack the expertise and tools to apply this knowledge in a way that tests the software product. Based on a case study at business-software company Exact, this study proposes a method of change impact analysis to help domain experts comprehend the structure of the system and allow them to take part in the code review process by assessing whether the impact of a change is as expected. Evaluation of a developed proof of concept at Exact using common-scenarios and a user evaluation shows that the method is effective in providing insights about the impact of changes to domain experts which provides a good intuition that using change impact analysis can aid domain experts to be involved in the process of improving software quality.

Thesis Committee:

Chair:	Prof. Dr. Andy E. Zaidman, Faculty EEMCS, TU Delft
University supervisor:	Ass. Prof. Sebastian Proksch, Faculty EEMCS, TU Delft
Company supervisor:	Valentijn van de Kamp, Exact
Committee Member:	Ass. Prof. Thomas Höllt, Faculty EEMCS, TU Delft

Preface

Writing a master's thesis felt like finding my way through a maze. When I started, I knew where I was, and I knew my goal was to get to the exit: graduation. Sometimes, the path towards the exit was clear and the next step was obvious. Sometimes the road was straightforward and enjoyable. Other times, I would take a wrong turn and end up in a dead end, forcing me to take a step back. Before you lies – or more probably stands – the route from the entrance to the exit: my master's thesis.

With an interest in the overlap between finance and computer science, I reached out to Exact hoping I could do my graduation project with them. Four months later, in November 2020 I was warmly welcomed in the office at Exact to pick up my laptop. This later turned out to be one of the only two physical activities of my thesis project. Nevertheless, I enjoyed the nine months that followed being part of the team at Exact. I want to thank Exact for providing me with this opportunity and all the means to collaborate towards my graduation. Especially I want to thank Valentijn van de Kamp and Edgar Wieringa for supervising my project. I want to thank the payroll team for all their help and the fun online *vrijmibo*'s.

Furthermore, I want to thank Sebastian Proksch for his supervision through the maze of graduation. Although we have never met in person, your efforts in finding the correct path and keeping my moral up have been a great help to me. I want to thank Andy Zaidman for his feedback on the thesis report. Although receiving critical notes is never fun, this polishing has led me to a thesis of which I can be proud.

Finally, I want to thank my family and my friends for their support during my thesis and my study in general. Especially in the times where social contact was sparse, your presence has kept me going.

Tim Nederveen
Delft, the Netherlands
August 15, 2021

Contents

Chapter 1

Introduction

To develop software for a specific context, software engineers need domain specific knowledge to incorporate into the software project to meet the requirements of the end-users. When software engineers do not have this knowledge, domain experts can be used to set up requirements using requirements engineering [17]. This does create a gap between software engineers and domain experts in terms of their ability to test the software.

On the one hand software engineers have the knowledge and techniques to systematically validate software, for example using forms of code review, static analysis, unit tests and integration tests [3, 19]. They do however often lack the domain knowledge needed to validate context specific parts of the software [5]. On the other hand, domain experts have this knowledge needed to validate the software, but often lack the expertise and tools to apply this knowledge in a way that tests the software product. Transferring this domain specific knowledge from domain experts to software engineers is inefficient and can lead to mistakes and inadequate software engineering [4].

In an ideal scenario, domain experts would be given the tools needed to allow them to participate in the systematic validation of software. Some existing methods attempt to involve domain experts in the process of software testing [2, 10, 17, 18, 21]. These methods are often focused on allowing the domain experts to manually test a software product or helping them to develop automated tests using graphical user interfaces or textual representations, which still require software engineers to set up and execute the tests.

The goal of this thesis is to find a methodology to enable domain experts without software engineering background to participate in the process of improving software quality with minimal effort from software engineers with the goal to improve the quality of the software product for end-users.

This MSc thesis project is performed at Exact, a Delft-based software company focused on business software. One of the products of Exact is Exact Online Payroll as an integrated part of Exacts business software suite: Exact Online. In this product, Exact supports more than 150 CLA's (collective labour agreements) which contain legislation and agreements regarding payment, leave, pensions, etc. CLA administrators at Exact are responsible for keeping this CLA configuration up to date on regular changes.

CLA administrators are domain experts with regards to payroll calculations but do not have a technological background, which means that they do not have experience with automatic testing or test-driven development, nor with the practices of a software development workflow.

Because the CLA configuration is complex and is at the core of the payroll calculations and therefore the payroll product, an effective strategy to utilise the knowledge of the domain experts is crucial to deliver good quality software. This makes Exacts case well suited to investigate how the goal of this thesis can be reached.

Using four research questions we will investigate how domain experts can be enabled to participate in the process of improving software quality. The workflow of the domain experts at Exact will be used as a case study.

> **RQ1** What are the challenges domain experts without software engineering background run into when participating in the process of involving software quality?

Understanding what limits domain experts to participate is essential to develop a methodology to enable their participation. This is investigated using interviews with the different stakeholders at Exact. Following the interviews, the primary insight is that the domain experts have trouble comprehending the structure of the software which makes it difficult for them to assess whether changes to it are correct. To overcome this, analysis using change impact analysis is proposed.

> **RQ2** How can the software under test be formalised in a model which can be used to analyse the impact of changes?

To allow an evaluation of this proposed methodology, it is necessary to develop a proof of concept for the case of Exact. To analyse the configuration and provide insights to the domain experts, the first challenge is to formalise the configuration of Exact in a model such that it can be used in the change impact analysis.

> **RQ3** How can the model of the software be used to provide useful insights to domain experts to help them assess the quality of changes?

The created model can be used to provide insights about the quality of the software product to domain experts. Answering this question results in a working proof of concept of a methodology to involve the domain experts in the case of Exact.

> **RQ4** Does the methodology of using change impact analysis enable domain experts to participate in the process of improving software quality?

Using this proof of concept, the idea of using change impact analysis to involve software engineers can be evaluated for the specific case study. This will be done by evaluating common-scenarios and performing a user evaluation. The findings of the evaluation of the proof of concept at Exact will provide an insight in how well the methodology of using

change impact analysis helps to involve domain experts in the process of improving software quality.

This research has three main contributions:

- it introduces a method to implement change impact analysis on software artefacts targeted at domain experts without software engineering background;
- it shows how formalising software in a model can help in providing insights about the quality of the software;
- using a proof of concept, it evaluates how well the method of change impact analysis helps to involve domain experts in the process of improving software quality.

This thesis describes the steps to answer the research questions using the following structure. Chapter 2 gives a background of the case of Exact by providing information about the current workflow and the different stakeholders. Chapter 3 describes the interviews conducted with the stakeholders at Exact to determine the challenges they run into when testing the CLA configuration and proposes a methodology of using change impact analysis to allow CLA administrators to validate the quality of their changes. Chapter 4 describes the methodology of creating a prototype by first describing the conversion of the configuration under test to a model which can be analysed. Chapter 5 then explains the methodology to produce insights which can be presented to the CLA administrator. Chapter 6 describes the common-scenario evaluation and user evaluation of the created prototype and the insights found from this. Chapter 7 compares the proposed method to previous work. Chapter 8 then discusses what the insights and implications from the evaluation are for the research question as well as describing the limitations of the research and further work in this direction. Chapter 9 concludes with a summary of the thesis.

Chapter 2

Context

Exact is a Delft-based software company focused on business software. One of their products is Exact Online Payroll as an integrated part of Exacts business software suite: Exact Online. This product is responsible for calculating more than 100.000 payslips each month and managing all information needed to perform this calculation.

In this product, Exact supports more than 150 CLA's (collective labour agreements) which contain legislation and agreements regarding payment, leave, pensions, etc. The calculation of all (sub)parts of the salary is based on various intertwined components where the output of one component can be an input for another. CLA administrators at Exact are responsible for keeping these component configurations up to date on regular changes.

Exact would like to have a strategy of using automated tests to verify the correctness of the CLA configurations. These tests have several goals:

- making the calculation of payslips more reliable by reducing the number of bugs in calculations;
- reducing the time needed to configure CLA's by providing more direct feedback on changes made in configuration files;
- possibly providing additional regression tests to verify that changes in the software do not break the payroll calculations.

Since CLA administrators are domain experts with regards to payroll calculations but do not have a technological background, they do not have experience with automatic testing or test-driven development, nor with the practices of a software development workflow. This means that the testing framework should be usable without this background and needs to be understandable enough to be socially implemented in the workflow of these administrators.

2.1 Workflow

Since 2020, the workflow of CLA administrators follows a software development strategy. Instead of changing the configuration using a web-interface of Exact Online, CLA administrators change the CLA configuration in JSON files with Visual Studio Code as IDE. The

JSON files are stored in a separate Git repository which can be used with common software development principles as pull-based development and version control. Using a converter, the JSON configuration files are converted to the same XML files as in the former workflow which then get loaded into the production database in the same manner as before.

To support the CLA administrators in creating their configurations, various internal tools have been developed. Using an extension for Visual Studio Code, administrators get auto completion for the relevant files and have access to a more human-readable description of the configuration files.

2.1.1 Testing

The payroll configuration is only tested manually. To do this, the team has three methodologies to test: performing calculations manually, a daily comparison and a VS code extension. The manual calculations and comparison are used for every change, the VS code extension is more a helpful tool when developing and is used little in practice.

Performing calculations An administrator can export the configuration to XML and load the configuration in a local instance of Exact Online. The administrator can then go through the payroll process using the provided configuration and manually validate that it gives the desired results.

Comparison Every day, a comparison called the *Payroll Compare* is run on the current version of the configuration. This comparison calculates end-to-end payroll calculations for different employees and scenarios in companies with different CLA's. This takes three to four hours to complete and then points out which parts of the calculations for different CLA's have changed. A tester then manually verifies that only the desired parts have changed correctly. Depending on the amount of differences discovered by the tool,

VS code extension One of the functionalities in the internal VS code extension for CLA administrators is the ability to try calculation methods and individual expressions by specifying input parameters and seeing the output of the calculation method or expression. This can be used to manually test a single part of the payroll calculation.

2.2 Stakeholders

End user An end user uses Exact Online to calculate and process payroll slips for each of their employees. A company which falls under a certain CLA can specify this in their settings after which the calculation uses the rules defined for the specified CLA.

When an end user discovers problems with the calculation, they can contact Exact support to report a bug. An end user can also diverge from the configuration by manually adjusting components in their configuration.

Support Support maintains the contact with the customer when problems arise. Support then gathers information about the problem and the steps to reproduce the problem.

This information is passed on to the CLA administrators, which help in identifying the cause of the problem and resolving the problem by either making changes to the configuration or by identifying problems in the configuration of the end user.

CLA administrator The CLA administrators are responsible for maintaining the CLA configuration. They update the configuration to follow changes in regulations and resolve bugs reported by end users in the configurations. The CLA administrators have expertise in the field of payroll calculations and do not have a background in software engineering.

Developers Developers are responsible for the code behind Exact Online Payroll. While they are not directly involved with the development of the CLA configuration, they do work on the calculation engine which uses the configuration to calculate. They were also involved with the design of the current workflow and provide support to the CLA administrator when they run into challenges.

Quality engineers Quality engineers are responsible for testing changes to both the software and the CLA configuration. The changes to the CLA configuration are mostly reviewed by the same quality engineer, but other quality engineers are involved in this process occasionally.

Chapter 3

Interviews

To understand the challenges domain experts run into when participating in the quality process, the domain experts at Exact are asked to explain their background, workflow and challenges with regards to testing the product. This will give insight in possible approaches and their potential value. Because the goal is to get as much information as possible from a small number of participants and to get a broad sense of the challenges they run into, this is done in the form of semi-structured interviews.

The domain experts in the case of Exact are the CLA administrators. These CLA administrators have the knowledge to interpret CLA's and convert these agreements to the CLA configuration in Exacts payroll product. Additionally, the quality engineers that test changes made by the CLA administrators know the workflow of the administrators and their common pitfalls. Both the CLA administrators and the quality engineers are therefore valuable interview subjects to learn about the role of the domain experts.

This chapter describes the setup and results of interviews with the CLA administrators and quality engineers about their current workflow with regards to testing and how domain experts are involved in the development process. Also, the software engineers are asked to verify the results of these interviews and provide their insights on possible approaches. The results of these interviews will be used to find a methodology to involve domain experts in the process of improving software quality.

3.1 Interview setup

The goal of the interviews is to understand the role of domain experts at Exact in the process of maintaining the quality of the CLA configuration and to gain insight in the challenges they run into. For this, semi-structured interviews are set up with the CLA administrators and quality engineers. The interviews are conducted through video conferencing in WebEx and last around 45 minutes.

In an introduction at the start of the interview, the participants are introduced with the goal of gaining more insight into their workflow around configuring and testing CLA's. The interview primarily focuses on their current workflow but will also ask the participants about their vision on how it can be improved.

The interview consists of four parts: establishing the background of the participant, understanding the current workflow, establishing which tools are in use and which functionalities or tools are missing and closing off. The questions for each part are a guideline but follow-up questions can be asked to gain more insight. To analyse the results later, the interviews are recorded with permission from the participants and from Exact.

While there is some overlap between how administrators test and how quality engineers test, not all questions are applicable to both groups of participants because of the nature of their role. For example, some of the tools used by the CLA administrators are not used by the quality engineers and vice versa. Therefore, the questions for some parts are tailored to the specific role.

Participant background In the first part, the participants are asked to explain their background and role at Exact, as well as their role with regards to the CLA configuration. The goal of this part is to get an understanding of the background of the people working on the CLA configuration to understand how this impacts their ability to contribute to the quality of the product.

1. Can you tell me a little bit about your background and your role at Exact?
2. What is your role with regards to the CLA configuration?

Workflow To understand how the participation of CLA administrators in the quality maintenance can be improved, we first have to understand how they are involved in their current workflow. Therefore, the second part of the interview focuses on the workflow of the CLA administrators and quality engineers. The main goal for this part is to identify the existing methods for testing and understanding what information is available to test changes.

For the CLA administrators, interview questions relate to how they gather the information needed for the CLA configurations change and how they use this information to test their own changes. The administrators are also asked to elaborate on how bugs are discovered, and which parts of their process are the most challenging.

1. How do you test changes you made to the CLA configuration?
    a) Where do you get the information about a CLA?
    b) When do you consider a configuration correct?
    c) What is the most common problem in configurations?
2. What are the steps to solve a bug?
    a) How are bugs discovered?
    b) How does this differ from planned changes to a CLA?
    c) Which step of the process takes the most of your time? What is the most painful step?

For the quality engineers, the questions are related to their process when testing a change made by the CLA administrators.

1. What steps do you take to test a CLA configuration?
2. Which step of the process takes the most of your time / is the most painful?
3. When do you consider a change correct?
4. Are there regression tests? What do they cover?
5. What is the most common problem in configurations?

Tools For the third part, the administrators and quality engineers are asked to elaborate on the tools they use to verify the CLA configuration. This will give a better sense of the methodologies used by the domain experts to validate software and will expose gaps in the current tooling.

The CLA administrators are asked about which tools they use and what their opinion is, as well as how they experience the learning curve to understand the tools and language.

1. What tools do you use to create and verify the CLA configuration?
    a) What do you like about the tool?
    b) What do you not like about the tool?
    c) What do you miss in the tool?
2. How easy was it to learn to use the tools & language? What did you find difficult?
3. What tool would you like to help you in creating and verifying CLA configurations?

The quality engineers do not use the same tools as the CLA administrators and are therefore asked how their use of the tools compares to the CLA administrators. They are also asked how they think the CLA administrators can be helped with testing.

1. What tool would you like to help you in testing CLA configurations?
2. How do you test differently than the CLA team?
3. How do you think the CLA team can be helped in testing their own configurations?

Closing To wrap up the interview, the participants get the opportunity to add remarks or ask questions they might have.

1. Is there anything you would like to add?

3.2 Interview analysis and validation

The interviews with the CLA administrators and domain expert produce unstructured data. To extract insights from the interviews, the statements of the participants need to be transformed in a more formal representation.

For this, the recording of each interview is transcribed literally after the interview using the recording of the WebEx-meeting. This transcript is used for further analysis. The goal of the analysis is to extract the main challenges when validating the CLA configuration and finding directions in which the involvement of domain experts in this process can be improved.

The analysis is done using a form of open coding [11]. The first step is to extract statements about the workflow from the transcript by marking them in the transcript and converting the statement to a (digital) post-it note, categorised by subject, where the colour of the post-it reflects the participant who made the statement. This resulted in 169 statements on 15 subjects. Next, similar statements from different participants are grouped together, resulting in 74 groups of statements with the same message, still categorised by subject. Finally, by combining similar problems from different subjects, the created groups are used to extract 6 challenges where improvements could be made to the way CLA administrators are involved in the process of validating the configuration.

To ensure the privacy of the participants, the literal transcripts of the interviews were not shared with others in- or outside of the company. To validate that the method of open coding extracted the correct problems, the results of the interviews are presented to the developers that work in the payroll team. In three individual brainstorm sessions, three developers are asked if they recognise the challenges concluded from the interviews. In addition, to gain insight in the impact and priority of the challenges, they are also asked how they would spend their time if they would have to improve the testing approach.

3.3 Interview results

After analysing the unstructured data from the interviews, six challenges with regards to maintaining quality were found. This section first describes the demographics of the interviewees, then the found results and finally the validation with the developers.

In total, five employees participated in the interview. Of these, three were CLA administrators, responsible for maintaining the configuration. Two were quality engineers. This number of participants was exhaustive, since there are no more employees in the CLA team, and the subject is too specific to interview employees outside of the CLA team.

Participant 1 is a CLA administrator who has been with the payroll team for almost five years and is the lead of the CLA team. P1 gained experience with payroll administration at another company and describes themself now as a payroll specialist. P1 does not have any technical background.

Participant 2 is a CLA administrator who has been with the CLA team for a year. Before P2 was active at customer support at Exact, specialising in support for the payroll package. P2 has some technical background in self-taught basic programming.

Participant 3 started as a CLA administrator around a month before the interview. Before this, P3 worked at the second line of customer support specialising in payroll. P3 does not have any technical background.

Participant 4 is a quality engineer who has been at Exact payroll for over five years. P4 has a technical background with almost fifteen years of experience in testing. P4 is the primary tester for the CLA team.

Participant 5 is a quality engineer who has been active as quality engineer at Exact for the payroll team for in total almost five years. P5 has a background in IT.

3.3.1 Found challenges

Based on the interviews with the five members of the CLA team, six challenges with regards to validating the CLA configuration were found.

Mostly manual testing When asking about how administrators and quality engineers test changes, all participants stated that the main strategy is to create or reuse employees in the software with specific scenarios, running the calculation and looking at the result and intermediate steps. Functionality for testing single configuration entities is rarely used.

Assessing the impact of a change All administrators and quality engineers agreed that it is difficult to assess the impact of a change on other parts of the configuration. The different parts of the configuration have a high level of interdependence, which means that changing one thing can have unforeseen effects on other areas. Three participants stated that the configuration is too complex, which also contributes to the difficulty of assessing the impact of a change.

Setting up scenarios According to three of the five participants, setting up employees and calculation scenarios to manually test the configuration is the most time-consuming step of the testing process. Which scenarios need testing depends on the change, which makes setting up scenarios manual and often repetitive work. Three participants would like to have an easier way to setup scenarios.

Using the Compare as a regression test The Payroll Compare is being used as a regression test with the goal to capture unintended effects of changes. All participants stated that the scenarios in the Compare are outdated and far from complete, meaning it will not capture all problems. Also, two participants mentioned that it is unclear what scenarios are in the comparison, making it more difficult to interpret the results of the comparison. Furthermore, because of the size of the Compare, running it takes too long (three to four hours) to be used as direct feedback while developing.

Systematic validation of configurations The three CLA administrators stated that most of the bugs are discovered by end users. Testing is only done on changes to the configuration and the regression test only compares results before and after a change, meaning existing problems will not be discovered. This means that the payroll configuration is not systematically validated.

Development experience and inconveniences Various specific problems make the configuration and workflow less development friendly and cause inconveniences while developing.

- The cycle of making changes to the configuration, transferring the changes to Exact Online, checking the results and going back to change the configuration is cumbersome (mentioned by two participants).

- It is not easy to check a small portion of the configuration because this involves manually entering all parameters every time (mentioned by two participants).

- The usage of codes for components makes understanding the configuration more difficult (mentioned by two participants).

- Components have the same start and end dates for both monthly and periodic calculations making it harder to create specific enough configurations (mentioned by three participants).

3.3.2 Validation with developers

To validate that the challenges extracted from the interviews are correct, the found challenges are presented to the developers in the Payroll team. These developers are also asked to give their opinion on which challenge has the highest impact and priority. In three semi-structured interviews, three developers from the payroll team provided their insights on the found challenges from the interviews. For each of the challenges, the developers were asked to elaborate on whether they recognise the challenge and on how valuable a solution for this challenge would be to enable the participation of domain experts.

All three developers had been with the payroll team for several years and have worked on the code responsible for calculating payslips. Of the challenges found by the CLA administrators and domain experts, five challenges resulted in four possible approaches to involve CLA administrators in improving the quality of the configuration. The challenges *Mostly manual testing* and *Systematic validation of configurations* were combined in *Moving towards automated testing*. *Development experience and inconveniences* was excluded because these improvements would not be part of a new methodology.

Moving towards automated testing All three of the developers stated that the payroll configuration can benefit from automated tests. Especially calculation methods are a good candidate for unit testing because calculation methods are independent parts with certain inputs and outputs. Testing components on a unit level does not provide much value since this would only test what is specified in the configuration of the component itself. Testing components on an integration level could be useful to determine whether they interact correctly. To prevent these tests from being high maintenance, the tests could focus on conditions with regards to the usage of components instead of asserting on often changing values.

Assessing the impact of changes The developers agree that predicting the consequences of a change can be difficult because of the unknown interdependencies between different entities in the configuration. Assessing the impact of a change can start by simply providing an overview of the components or calculation methods that are related to a file that is being changed. A more extensive method of determining the impact can definitely provide a lot of value if it is feasible.

Defining scenarios The workflow of administrators would be simplified if they could execute scenarios right from Visual Studio Code instead of exporting the configuration to Exact Online and running scenarios there. For this, scenarios could be defined as configuration, possibly based on the JSON format that is used to provide to the calculation engine. These scenarios can then be shared between developers and could be exported from Exact Online when for example support reports a bug with a specific scenario.

A side note placed by one of the developers was that it might be good to check how much of the changes to the configuration are in calculations and how much are in components to determine where to focus the effort.

Improving the Compare The speed of the Compare could be improved by only executing scenarios which are related to changes to the configuration. This could be tricky since changes to calculation methods can have unforeseen consequences to other CLA's. These could be missed when executing only the CLA that was intended to change. However, for some changes, choosing which scenarios are relevant might be possible.

To better understand the contents of the Compare, a tool which provides coverage information could be developed. Using the coverage of the Compare, an administrator would know whether or not their changes were being covered by one of the scenarios in the Compare. Another way to make the Compare more transparent, is making the scenarios part of the code repository.

3.4 Chosen approach

All of the possible approaches might have a positive influence on involving the domain experts at Exact in the process of improving software quality, but because the scope of this thesis only allows the realisation of one approach, the most promising approach has to be chosen. To do this, the opinions of the stakeholders at Exact and the ability to generalise the approach to other contexts is considered.

A common theme in all the interviews with the CLA administrators and quality engineers is not being able to comprehend the structure and interdependence in the configuration. As a consequence, administrators and quality engineers have difficulty in assessing the impact of changes, which makes it hard to assess if the change has any unintended wrong consequences for end-users. Also after talking to the developers, the idea of building a method to mitigate this problem seems to have the potential to provide a lot of value to the team in terms of helping them to improve the quality.

In addition, of the found approaches, helping domain experts assess the impact of changes is the most generalisable to other contexts. While developing an automated testing methodology for the configuration entities, helping domain experts to define scenarios to use for testing and improving the tool used to compare calculations before and after a change will most likely have positive effects on the quality of the software, this would mainly focus on fitting existing techniques to the specific case of Exact. This is quite specific for the context of Exact making it harder to transfer to other contexts. Conversely, the

concept of assessing the impact of changes is broadly applicable in other contexts by fitting the concept to the specific software context.

Therefore, this thesis chooses to focus on the idea of enabling domain experts to assess the impact of changes with the goal to prevent bugs for end-users because of unintended consequences of changes. For this, a change impact analysis tool will be implemented for the case of Exact which can then be evaluated with the CLA team.

Chapter 4

Formalising the configuration

To allow the development of a proof of concept for the method of change impact analysis as described in section 3.4, the subject under test has to be converted into a model which can be analysed. In the case of Exact, the subject under test is the configuration which defines how payroll calculations are performed.

While the different parts of the configuration have references to each other, the current form of the configuration does not provide enough structure to analyse changes to the configuration. Configuration entities refer to other entities by name, which means that connections are implicit and are only resolved at runtime. To allow for more extensive static and dynamic analysis to provide insights about the impact of changes, the configuration should be formalised in a model where the connections between different entities are explicit.

This chapter describes the current structure of the configuration in section 4.1, the design of a model in section 4.2 and the steps to generate a model from the configuration in section 4.3.

4.1 Domain model

The structure of the configuration files is custom for Exact and consists of different entities with several different dependency types. Before modelling these connections, we first need to understand this structure of the current configuration to be able to derive a model that contains all necessary information.

The configuration for the payroll calculations is stored in JSON files in a separate repository from the payroll software. These files are maintained by the CLA administrators. Changes to the configuration are converted to a representation which is included in the software platform that is used by the customers. The configuration consists of five parts: groups, components, calculation methods, formulas and tables.

Groups Groups represent different levels in the configuration with which CLA administrators can create a hierarchy of employment condition groups. Each group is represented in a file in the corresponding directory. This file contains information about the group, such as the identifier, a description and the number of hours in a full-time week. In general, there are three levels: the top-level (`NL`), the sector level (for

example `NL-0100` for agriculture) and the branch level (for example `NL-0101` for gardeners).

Components Components are the core of the configuration and are used to calculate all different parts of the salary slip and pension declaration. Each component is part of a group, is identified by an alphanumerical code and is described in a JSON file in the directory corresponding to the group. Each component has a set of inputs, which can refer to other components in the calculation, outcomes of components in historic calculations or details about the company, employee or employment. Next to the inputs, each component also has a set of outputs which can be used by other components or can be shown on the payslip. The component specifies the calculation method which is used to calculate the outputs. Each output defines which step of the calculation it should output. A component can also specify that the outcome of the calculations should be included in a number of subtotal components.

There are several mechanisms within components to allow administrators to efficiently create the CLA configuration and keep it up to date. Components can inherit properties, inputs and outputs from other components. This allows for inheritance between different employment condition groups. By default, all components on branch level are inherited from the sector level, but an administrator can choose to create an inherited component which overrides certain properties in a component with branch-specific agreements. Another mechanism is the start and end dates of components, which allows administrators to create successors of components when agreements change.

Calculation methods Calculation methods describe how the outputs of a calculation method should be calculated. Each calculation method is identified by a code and is described in a separate JSON file. A calculation method can be used by multiple components. A calculation method has a set of parameters which correspond to the inputs as defined in the component that uses the calculation method. To calculate, the calculation method follows a list of steps. Each step uses a formula to calculate an outcome based on the value of the parameters or earlier steps. These steps can then be referenced by components outputs.

Formulas Formulas are Excel-like expressions which can reference calculation method parameters and calculation method steps by name. Each formula has a single output. Formulas are defined in individual files which are referenced by filename. Formulas can be used by multiple calculation methods.

Tables Tables are CSV files which can be used as lookup tables in calculations. This can for example be used to look up the minimum wage based on the age of the employee.

4.2 Design model

To be able to determine the impact of changes to the configuration, the configuration needs to be structured in a model which has connections allowing further analysis, i.e. to deter-

mine which entities will change when one entity has been changed. The plain representation of the configuration as stored in the JSON files does not contain any links between different parts, which makes it hard to follow how changes propagate through the configuration.

The principle of entities which are dependant on other entities for their calculation is similar to a call graph. Instead of functions that call other functions, the configuration consists of entities which have dependencies on other entities. By converting the configuration to a graph where each configuration entity is a node and entities that impact each other are connected through edges, it is possible to analyse the graph to provide insights about the configuration and traverse the graph to determine the impact of changes.

The graph contains nodes for each entity in the configuration, i.e. components, inputs, outputs, calculation method steps, calculation method parameters and formulas. For inputs and outputs which are inherited from another component, nodes are only created if the inherited entities overwrite one or more properties.

Edges in the graph are directed and represent an impact relation between two nodes, i.e. if the node at the tail of an edge changes, this change will impact the node at the head of the edge. In most cases, this is directly related to the control flow of the payroll calculation, making the graph representation similar to a call graph. Edges are also created for inheritance relations. These relations do not represent the control flow but are relevant since a change in the parent will have impact on the child.

Some edges represent a connection that is only present under certain conditions. This occurs in two cases. The first is when a component is only active for a specified period. Edges to this component should therefore only represent impact in this specified period. The second is the edges from calculation method steps to component outputs. Calculation methods can be used by multiple components, but when it is used in component A and B, changes propagated from component A through the calculation method should not impact component B. Therefore, in both cases, the edges are annotated with the conditions. When traversing the graph, these conditions should be checked.

Figure 4.1 shows an example of a graph using this representation. The dots represent the nodes which represent various configuration entities, symbolised by a colour for the type and a title below the node. The white nodes represent input data. A solid edge from node A to B represents that A is used for the calculation of B. A dotted edge from A to B means that B inherits properties from A. Some edges have conditions to represent impact relations that only apply in certain periods or certain CLA's.

In the example graph, there is one calculation method with two steps. Both steps get their input from a calculation method input. `STEP2` also uses the outcome of `STEP1`. These calculation method inputs are in turn coming from a component input. The calculation steps both use a formula to calculate their outcome. STEP1 is used for a component output, but only for a specific component for a specific period. The same applies for `STEP2`, which is used in an output of a component with multiple versions.

The dotted lines between components and between components and inputs or outputs represent inheritance relations. For example, component input `PARAM2` is part of Component `NL/A`. Component `NL/NL-0000/A` inherits from `NL/A`, which means that it also inherits the

same component input. If anything changes to component `NL/A`, this will be propagated to all inherited components and related entities.

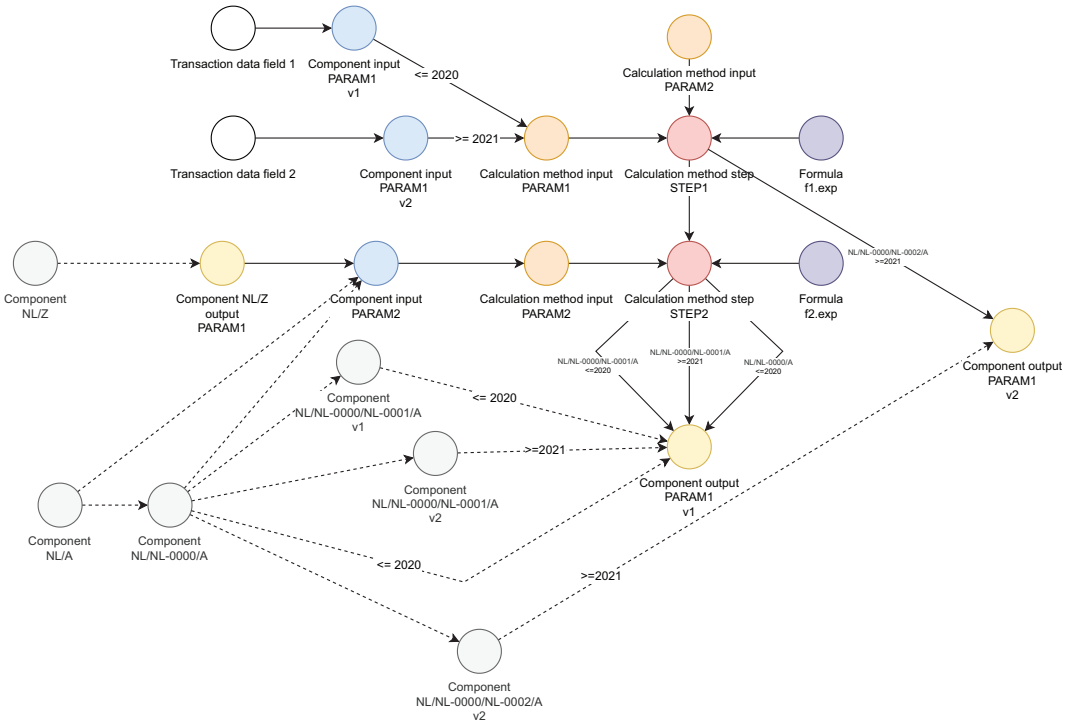

Figure 4.1: Visualisation of example model graph structure

4.3 Converting configuration to model

Now that we have a design for a model, we can convert the current configuration to the graph representation as described in section 4.2 which can be used to determine the impact of changes.

Parsing To parse the configuration, the JSON files are read in and stored in an in-memory representation of all the configuration entities. Because the configuration is stored in JSON files, these entities can be read in trivially into classes following the schema of the JSON files. Each entity is stored in a list of all entities of this kind. In this step, there are no relations between the entities created yet.

Nodes Next, nodes are generated from the parsed entities. Every entity is converted to one node. The nodes are added to a graph with a unique key based on the type and name. This key can later be used to find the node in the graph.

Edges After creating the nodes, edges between the nodes are generated. For most edges, it is trivial to find the referenced node. For example, a reference to another step of a calculation method can only refer to one other node. Similarly, every component input can be linked to the corresponding calculation method parameter and every output can be linked to the defined calculation method step.

For some relations, the calculation engine does not resolve the reference until runtime because resolving these requires information about the calculation period and selected employment condition group. Since components can be overwritten on multiple levels and can have start and end dates, a reference to a component can point to multiple components. While building the graph representation, the graph builder takes all possible impact relations into account. The created edges are annotated with information about when the edge should be followed, i.e. in which periods the link is active and for which employment condition groups the component at the head of the edge is valid.

To achieve this, the graph builder creates a data structure which stores for each component code which components could be referenced and for which periods it would be active. When indexing a new component on a deeper level, the period for which this component is valid is subtracted from components with the same code on a higher level. When the graph builder wants to resolve a component code, it creates edges to every node that could be active in the period of the source component and is reachable from the level of the source component.

The result is a graph with edges which define when each node has impact on which other nodes.

Chapter 5

Implementing a change impact analysis tool

In the interviews, not being able to comprehend the impact of changes made to the configuration was one of the main challenges domain experts identified. After we have translated the configuration into a model in the form of a graph, we can develop a proof of concept implementation of a tool that provides domain experts with more insight about the impact of changes to the configuration. This proof of concept is necessary to evaluate the use of change impact analysis to involve domain experts in the process of improving software quality.

The configuration of CLA's is at the core of calculations in Exact Online Payroll. It is essential that the maintainers of this configuration understand the impact of changes to ensure the correctness of payroll calculations for end users. By providing a solution which provides insight in the impact of changes made to the configuration, CLA administrators and testers will gain more understanding of the working of the software. Because of this, CLA administrators will be more assured of the impact of their work and bugs will be caught more often.

5.1 Vision

To achieve this, a tool should be created which determines the impact of changes made to the configuration. This tool would be integrated in the workflow of the CLA administrator. Since analysing the entire configuration will take a few minutes, using it as part of the continuous integration pipeline would provide the least interference with the work of the user. The tool would run on changes made in the branch and then report under which conditions the calculation outcomes will change and what this change is. This is presented to the user, for example as attachment in a pull request, who can then manually verify whether the changed outcomes are expected and correct, allowing the domain expert to take part in the code review process.

The following steps illustrate how the tool would be used in the workflow of the CLA administrators to provide insight about the impact of changes:

1. A CLA administrator wants to make a change in one specific CLA. For this, the administrator changes a formula which contained a wrong operation. The administrator commits the changes in a separate branch.

2. The CLA administrator creates a pull request for the created branch which triggers the change impact tool to run on the changed branch.

3. The change impact tool compares the selected branch with the master branch and finds the changed formula. The tool then finds two calculation methods which use this formula. Next, the tool determines for which CLA's and under which conditions these calculation methods are used. Finally, the tool determines which component outputs will change because of this formula change.

4. The CLA administrator receives a report which states that the changes in this branch have impact on two groups. The first group is all the employees in the CLA where the intended change was, which is expected. The second group is hourly paid employees from a different CLA. This should not have changed.

5. The CLA administrator looks at the report to determine why the second group is affected. The report shows that the formula is used in a different place. The administrator changes the configuration to only use the changed formula in the correct CLA.

6. The change is committed, and the change impact tool is run again. It now only shows the expected change for group 1.

7. The administrator now requests review from a different administrator or tester. The tester looks at the report attached to the pull request and sees the one expected change. The tester verifies that this change is correct and complete and approves the pull request.

5.2 Implementation

To determine what has changed, the tool analyses differences between two git branches, i.e. the main branch and the branch with the changes that need to be analysed. Both branches are checked out and converted to a model. Next, for each node in the new graph, the properties of the node are compared to the equivalent node in the old graph (i.e. the node with the same key) are compared. If there is a difference in any of the properties, the node is marked as changed by adding the key to a list. New nodes in the model of the changed branch are added to this list to handle parts added to the configuration. The tool does not yet cope with deleted parts of the configuration. Deletion of configuration entities does not occur often and the impact of deleting configuration entities is often already clear by the checks that are already performed to check the references between configuration entities.

After determining the changed nodes, each of the changed nodes is the starting point of a traversal through the graph. The traversal is performed using a depth first search. At every node, all outgoing edges are considered for traversal. Information about the traversal

through different nodes is stored in a trace. This trace also contains information needed to make a decision at conditional edges like the period for which the trace is active or for which component the trace is going through a calculation method.

At every node, the trace up until that point is stored in a list which in its turn is stored in a dictionary with the node key as key. After having traversed the entire graph, this dictionary is used to determine all component outputs that are possibly affected by the changes.

Since the model of the configuration contains loops, the traversal needs a mechanism to detect and prevent looping around indefinitely in the graph. To do this, a dictionary stores for each node for which CLA's and periods it has been traversed. When a trace reaches a node which has already been traversed for all the CLA's and periods in the trace, the trace will end, since there cannot be any new effects when the part of the graph is traversed again with the same conditions. If a trace reaches a node with different conditions, that part of the graph is traversed again with the new conditions until there are no more new CLA's or periods for which nodes can be traversed.

Finally, the found impact needs to be presented to the user. For this, the tool generates a report. The report has two goals for CLA administrators. Firstly, it should provide a clear overview of what impact their change has on the entire system. This allows CLA administrators to assess whether their change is correct and does not have unintended consequences. Secondly, it should provide enough information for CLA administrators to dig deeper into the found consequences to allow them to find where unintended effects originate and how they can be solved.

The tool generates a report in the form of an HTML file. Using JavaScript, dynamic functionality is added in the report to allow the users to click between tabs and filter the results on CLA and period.

To give users a quick overview of the impact of their change, the report first shows a list of CLA's which are affected. Next, the report shows timelines for the years around the impacted periods for monthly, four weekly and weekly periods. Periods that are affected are coloured orange. By clicking on the CLA's or periods, user can filter the effects to only show effects in a certain CLA or a certain period.

Next, the report shows a list of all component outputs which can be affected by the change. For each output, the CLA's and periods for which the change applies are shown, together with a list of traces which show the path from the changed configuration entity to the given output.

The result is a proof of concept of a tool that takes reads in the configuration, converts it to a model, looks at a change using version control, computes what the impact of this change on other parts of the software is and presents these insights to domain experts. Figure 5.1 shows an example of a part of a generated report.

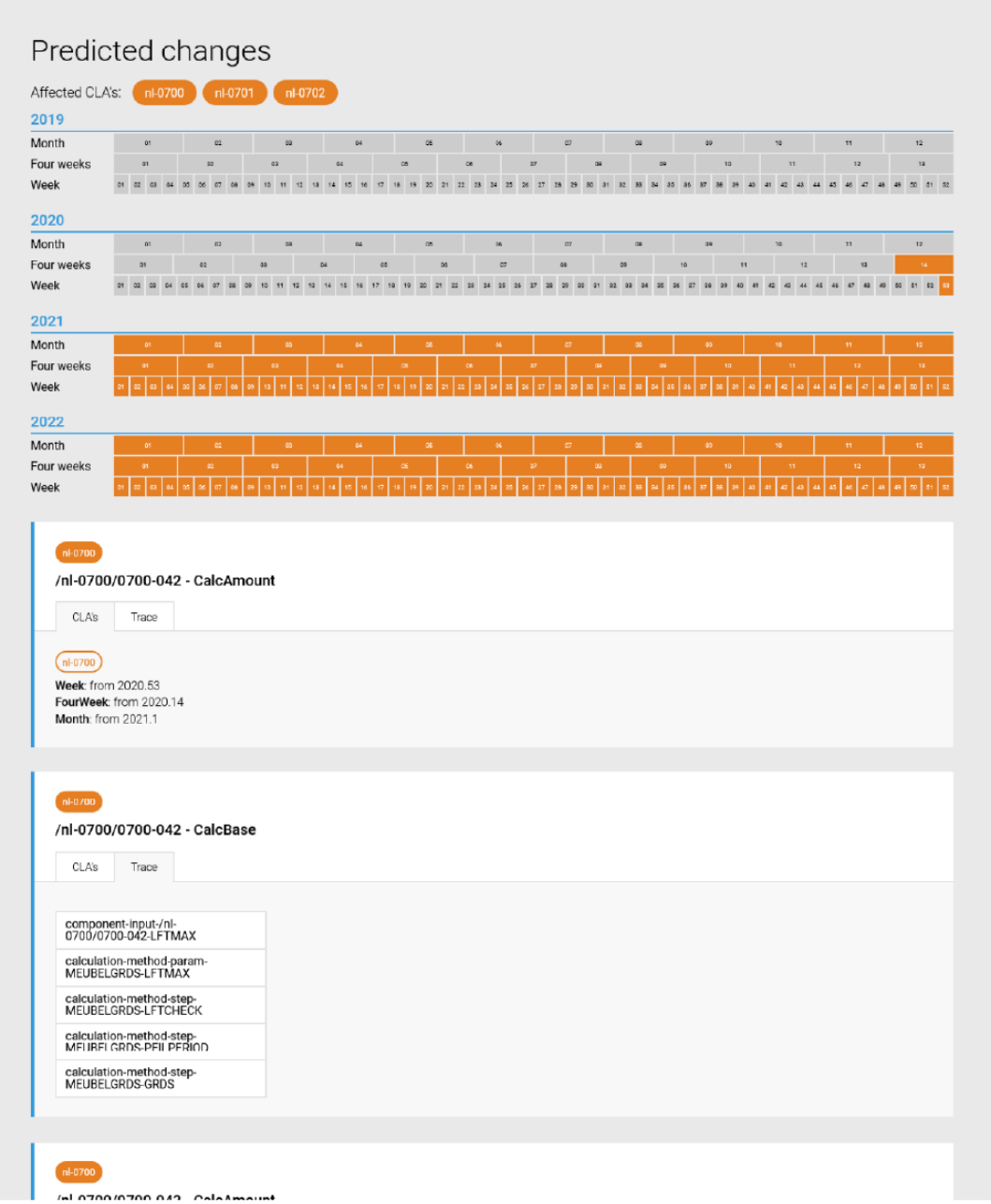

Figure 5.1: Example of part of generated report

Chapter 6

Evaluation

The proof of concept implementation of the change impact analysis tool has the potential to enable domain experts to be more involved with improving the quality of the software. To investigate the advances, the model and tool will be evaluated from two different angles. Section 6.1 describes a common-scenario evaluation to evaluate the completeness of the model and the correctness of the change impact analysis algorithm. Section 6.2 then describes a user evaluation where the tool is presented to the targeted users to evaluate the usability of the tool and the value for their workflow.

6.1 Common-scenario evaluation

A requirement for a tool that determines what the impact of changes will be is that it is as complete as possible to not miss unintended effects. To understand how complete the model behind the tool is and how correct the algorithm for change impact analysis is, both should be evaluated by verifying that the tool is capable of providing actionable feedback on every configuration change. Unfortunately, performing a quantitative evaluation on changes from the past is infeasible, as it would require a lot more data about what types of bugs were solved by each change and would take a lot of time, which was not possible within this thesis project. To still show how the model and algorithm perform on real-world scenarios, both are evaluated using scenarios of common problems in the configuration based on the experience of the CLA administrators and quality engineers. This will show to which extent the tool is capable of providing actionable feedback on relevant and often problematic configuration changes.

To gather these common scenarios, three CLA administrators and a quality engineer were interviewed. Before the user evaluation, the participants were asked to describe categories of failures that can occur when changing the configuration. The answers from these interviews were combined in a list which was then coded to find a complete as possible set of scenarios. This resulted in an unordered list of eight categories. Two categories of failures were discarded: *syntax errors*, which is irrelevant for the working of the tool since these are already filtered out before the code is ready to be reviewed, and *readability and maintainability*, which was more a cause of problems than a category of failures. The

remaining six scenarios are as follows:

1. Unintended effects in other CLA's
2. Unintended effects in other periods
3. Unintended side-effects in calculation methods
4. Creating a new entity but not using it
5. Bug fixes on the wrong level
6. Unsolved bugs for other employee types

The scenarios are explained in more detail later. Using these common scenarios, the model and algorithm can be evaluated. This consists of two steps. The first step is to evaluate how the tool handles a hypothetical case of this scenario by reasoning about the working of the model and algorithm. The second step if the scenario is captured by the tool is to construct an example of the scenario in the configuration and show the output of the tool.

Results

After evaluating the model and algorithm for each of the given scenarios, these are the results per scenario.

1. Unintended effects in other CLA's Configuration entities can be used in calculations for multiple CLA's. When a component or calculation method is changed with the intention to change a calculation for one CLA, it could also have unforeseen effects on the calculation of other CLA's.

Components and calculation methods and any part of them are integrated as nodes in the generated model of the configuration. From these nodes, edges are created to any other node which is dependant on the outcome of this node. When a CLA administrator updates a component, calculation method or any part of that in a specific CLA, the tool will mark the corresponding node as changed. Using traversal, all edges following from this changed node are followed. This results in a list of nodes which are dependant on the changed nodes. For each component in the configuration, the tool determines which CLA's use the component in the calculation. This is done by determining the location of the component and checking whether the component is overwritten by a component with the same code on a lower, more specific level.

By combining this information, the tool can determine the CLA's that use any of the nodes that were dependant on changed entities in the configuration. A list of these CLA's is presented in the report that is presented to the user, which allows them to judge whether there are any effects in CLA's that were not intended to change.

Example
Suppose an administrator wants to change a part of the calculation for CLA `nl-0301` (construction and infrastructure). The administrator changes the calculation method `CORDGRDSL` which is used for this calculation, assuming it will only have impact on this CLA and its descendants `nl-0301a` and `nl-0301b`. After running the tool and

opening the report, the administrator finds that the change has impact on a lot more CLA's than intended. This is shown at the top of the report with a list of all affected CLA's, as shown in figure 6.1.

Figure 6.1: Indication of changes in unintended CLA's

2. Unintended effects in other periods When a configuration entity is used in calculations for multiple payroll years, changes to this entity or related entities can have unintended effects on other periods. This also occurs when a component has a wrongly configured start date, which causes it to be used in the calculations for a wrong payroll year.

The scope of periods on which a change has effect is determined by the start and end dates of components that are changed or are dependant on a changed entity. For each component, the tool determines for which payroll periods (monthly, four-weekly and weekly) the component is calculated.

After having created a list of components which have changed or are dependant on a change to the configuration, the tool collects all periods for which these components are calculated. These periods are presented visually in the report, making it easy for users to spot unforeseen effects in other periods.

Example
Suppose an administrator wants to make a change to a component to update the payroll calculations for the furniture industry CLA in 2021. For this, the administrator updates one of the inputs of the component `0700-042`. After running the tool and opening the report, it becomes apparent that the change also has effect on the last period of the four-weekly and weekly calculations of 2020. This is shown at the top of the report with the coloured periods which indicate a change in that period, as shown in figure 6.2. This is correct, since the component has a start date of `2021-01-01`. This date is part of week 53 of 2020, meaning that the component will also be used to calculate this last week and the last four-weekly period of 2020.

3. Unintended side-effects in calculation methods Changes to alter one output of a calculation method might also have impact on other outputs of the calculation method. Especially since not every parameter of calculation methods is well documented, changes to the calculation method can have unintended effects on the calculation.

Calculation method parameters are incorporated in the model of the configuration by creating edges from the parameters to the calculation steps with formulas that reference a particular calculation method parameter. This means that changes to configurable parameters of a calculation method will be reflected in the report of the change

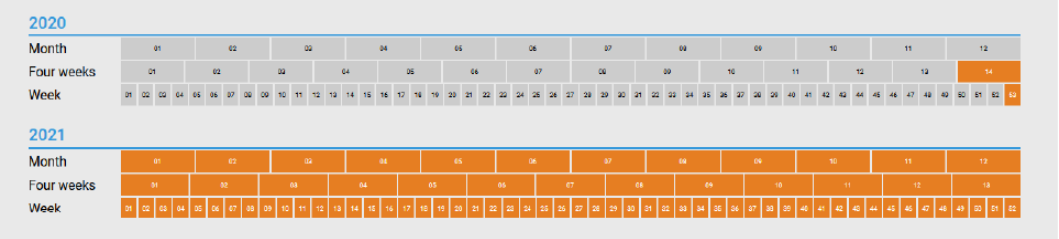

Figure 6.2: Indication of change in unintended period

impact analysis tool. This only indicates that the parameter is used in a step of the calculation method and does not necessarily show any unexpected results. Changes to outputs are also represented in the report, meaning that if a change to an output causes another output to unexpectedly change this will be shown in the report.

Example
Suppose an administrator wants to change one specific step of the calculation method `CORDGRDSL: BTERUURLN` to update the output `CalcMaximum` in component `5601-023`. For this, the administrator updates the formula that is responsible for this step, `/CORDGRDSL/bteruurln.expression`. The assumption is that this will only change the output `CalcMaximum`. After running the tool and opening the report, the administrator finds that also the output `CalcAmount` has been changed, as can be seen in figure 6.3. Looking at the trace shows that this is because the step `BTERUURLN` is also used in `RESULTAAT` which is responsible for the `CalcAmount` output.

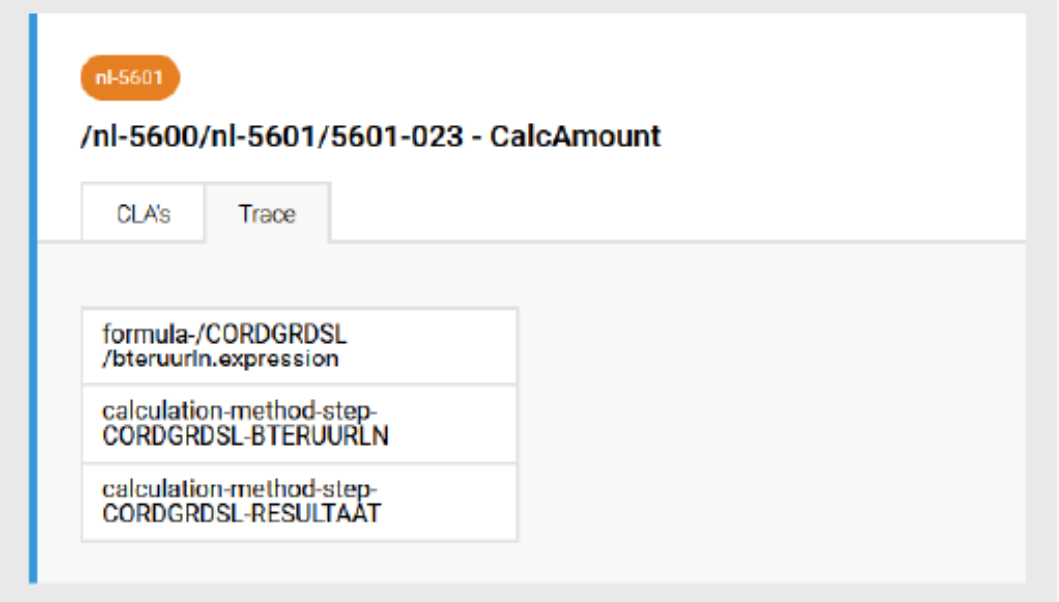

Figure 6.3: Affected output because of unintended side-effect in calculation method

4. Creating a new entity but not using it A CLA administrator can create a new configuration entity like a formula or calculation method. Until the new entity is referenced by an existing entity, the new entity is never used. Therefore, if the CLA administrator forgets to change this reference, the calculation will not change.

New configuration entities will be incorporated in the model and outputs in the new configuration entity will be shown in the report. However, if the new configuration entity is never used, it will not have impact on any component output and will therefore not be shown in the report. Unless the CLA administrator has made more changes, this will give a clear signal that the change has no effect on the calculation.

Example
Suppose an administrator wants to add an extra output to the component `6001-041` with the goal to use this output in another component. The administrator adds the output to the component but forgets to use the output in the other component. After running the tool, the report only shows one change which is the new output, as shown in figure 6.4. The rest of the report is empty, which is a clear indication that the change did not have the intended effect.

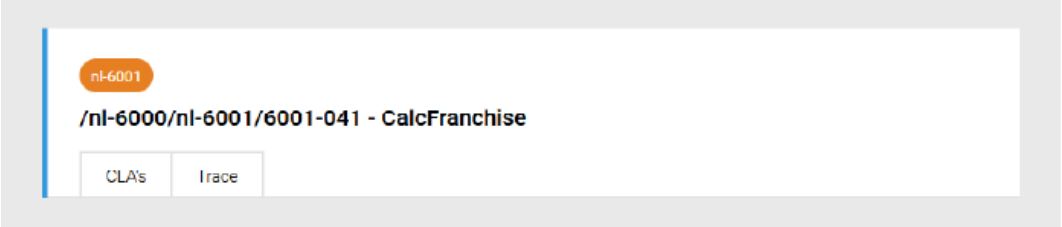

Figure 6.4: Only changed output because of not using new entity

5. Bug fixes on the wrong level When a bug is solved for a specific CLA, problems can still occur in related CLA's because of code duplication causing a form of late propagation [1]. This can also occur when a fix is applied to a specific CLA when the problem was in a more generic configuration entity.

If a bug is solved on CLA level, the report will show that outputs in this CLA have been changed. Similarly, if a bug is solved in a configuration entity that is shared between different CLA's, the report will present all the CLA's where the change has effect. This means that if a CLA administrator does not know that the bug occurs in multiple CLA's, showing only the CLA's that has changed because of the fix will not point the user to the problem in other CLA's.

Detecting this scenario would only be possible if the tool pointed users to configuration entities that are similar to the changed entities to let the user decide whether they should be updated as well.

6. Unsolved bugs for other employee types If a problem is solved for a specific type of employee (full-timer, part-timer, etc.) while the bug was caused by a more generic bug, the problem persists for the other employee types.

The distinction between full-time, part-time and hourly employees is made in conditional expressions in the formulas of calculation methods where the value of parameters containing the type of employees are checked. While the tool does analyse which parameters are referenced in a formula, it does not perform any further analysis on the contents of the formulas. This means that the tool cannot distinguish between different types of employees when determining the impact of a change. A change for a specific type of employee will show up in the report as a change for everyone in the CLA.

For this to be more specific, formulas would have to be further expanded in the graph to for example a syntax tree, which would allow the tool to determine which parts of the formula are affected by a change and reporting under which conditions the effects take place. Because of the complexity, this was not yet part of the scope of this project.

As shown in table 6.1, four of the six scenarios are well represented in the model and will be detected by the tool in most scenarios. For the other scenarios, the tool does provide information which can help in discovering the bug, but not provide enough detail to always detect the problem. Fully covering these scenarios would require an extension to the tool.

Scenario	**Result**
1. Unintended effects in other CLA's	Covered
2. Unintended effects in other periods	Covered
3. Unintended side-effects in calculation methods	Covered
4. Creating a new entity but not using it	Covered
5. Bug fixes on the wrong level	Needs extension
6. Unsolved bugs for other employee types	Needs extension

Table 6.1: Result per common scenario

While these scenarios are not proof of the correctness of the tool and the list of scenarios might not be complete, the scenarios are illustrative cases for how the change impact analysis tool is able to provide insights to the domain experts about the quality of the software. The model can be extended to incorporate more cases if scenarios are found which are not yet represented.

6.2 User evaluation

To determine whether change impact analysis provides the desired value to the domain experts, the ultimate test is to evaluate the proof of concept with the targeted users in the form of a user evaluation. This means determining whether the users have the confidence that the solution will help them to improve the quality of the software.

6.2.1 Methodology

As part of this evaluation, users give their opinion on the usefulness of the concept of change impact analysis and the user interface of the prototype. This is done using the method of heuristic evaluation as described by Nielsen and Molich [16]. The idea of heuristic evaluation is that a small group of expert evaluators inspect the application and rate a set of usability principles. Commonly, ten heuristics devised by Nielsen and Molich are used to evaluate the usability. Since our evaluation is not purely focused on the user interface itself but also on the usefulness of the results provided by the change impact analysis, specific heuristics in line with the goal of the solution are used instead:

1. The tool provides useful results.
    a) **Complete**: there are no effects missing from the report.
    b) **Correct**: presented effects are a consequence of the changes.
    c) **Non-trivial**: the presented effects were not clear before running the tool.
    d) **Insightful**: the report provides insights needed to fix unforeseen effects.
    e) **Valuable**: the time it takes to look through the report is worth it.
2. The tool is easy to use
    a) **Logical**: navigation through the report does not require outside assistance.
    b) **Clear**: the meaning of each effect becomes clear from the given information.
    c) **Clean**: the report does not provide too much information.

The experts used in the evaluation are three CLA administrators and a CLA tester. Since they work with the payroll solution daily and are experts in the field of payroll configuration, they have the expertise to describe the potential value of the tool for their workflow.

The evaluation is performed in the form of semi-structured individual interviews which last an hour. The experts are first introduced to the concept behind the tool with an explanation how the change impact analysis would be performed on pull requests and would give insight in the effects of changes to the configuration. The experts are given an explanation of the heuristics which they will use to evaluate the tool. For this, they also receive a PDF containing the heuristics with brief introductions. The experts are given the instruction that they are going to look at three reports, after which they are asked to rate each of the eight heuristics on a scale from 1 (completely disagree) to 5 (completely agree). In addition, they are asked to share their opinion about the working of the tool already while looking at the reports and to ask questions if they have any.

Next, the evaluators receive a zip file containing three generated reports from the built prototype. By using three different reports, the evaluators get a broad picture of the functionality of the tool which allows them to also evaluate the concept of the tool instead of an individual report. Each report emphasises different parts of the tool. The first report is about a change in one CLA, which results in a small set of changes (7 changed outputs). The second report is about a change in a configuration entity on the top-level. This change

has a large impact on the entire configuration, leading to a very large set of changes (699 changed outputs), requiring the user to use filters to make sense of the information. This report also contains an example of a misconfigured period which can be seen in the period filters at the top of the report. The third report is about a change in references within configuration entities which has effect on several CLA's. This results in a set of 142 changed outputs.

The experts are first introduced with the change in the configuration by showing the git diff of the changed configuration entities. After this, they can open the report and inspect the presented information by clicking around in the report. At the first report, the participants are introduced to the structure of the report and the possibilities with regards to navigation and filtering. If the evaluators have any questions with regards to the working of the tool, they receive an explanation about this part. After each report, the participants are asked what they think about the example report and the working of the tool.

After having seen all three reports, they are asked to give their opinion on each of the given heuristics. For each heuristic, they are asked to give a score using a Likert scale [13] of 1 to 5 and explain why they chose that score, preferably using examples from the tool itself. Afterwards, the experts give their opinion on whether the tool would be a good addition to their utilities and what value the tool would provide in their workflow.

To analyse interviews with the experts, the scores are the starting point. For each of the heuristics, the given scores are explained based on the explanations from the evaluators. In addition, for each heuristic, the implications of the perception of the evaluators on the usability of the tool are discussed. The results are concluded with an analysis of the sentiment regarding what the administrators and tester think about the value of the tool in the process of maintaining the quality of the configuration.

6.2.2 Results

In the interviews, the experts gave their opinion on each of the eight heuristics. Their findings are summarised by the scores they gave for each heuristic as shown in table 6.2. As well as giving scores for each heuristic, the experts described what they liked and disliked about the tool for each of the heuristics.

	Expert 1	**Expert 2**	**Expert 3**	**Expert 4**	Avg
Complete	2	5	5	5	4,25
Correct	5	5	5	5	5
Non-trivial	5	4	4	5	4,5
Insightful	5	3	5	5	4,5
Valuable	5	3	5	4	4,25
Logical	4	5	5	4	4,5
Clear	4	2	5	4	3,75
Clean	5	1	5	5	4
Avg	4,375	3,5	4,875	4,625	

Table 6.2: Scores per expert per heuristic, higher is better

Except for the clearness of the report, each heuristic has an average score of 4 or higher, meaning overall the experts were quite positive about the working of the tool. When looking at the average scores of the experts, expert 2 stands out with an average of 3,5 where the others are all 4,375 or above. This is in line with the tone during the evaluation, where expert 2 expressed their concerns about interpreting the results given by the tool.

Complete Three of the four experts fully agreed with the statement that there are no effects missing from the report. Expert 1 disagreed and gave a score of 2. All experts mentioned that they had reservations with stating that the reports were complete because the reports were too large to manually check everything. However, looking at the report, they could not come up with effects that were missed, which gave them the confidence to give a score of 5. Expert 1 discovered a problem in one of the reports, where the report indicated that one of the effects was applicable for one year for the weekly and four-weekly calculations, but indefinitely for monthly calculations, as shown in figure 6.5. This is impossible with the structure of the configuration and the export therefore identified this as a flaw immediately and scored this heuristic with a 2. Analysis showed that this problem was caused by a bug in the way periods are combined before they are presented to the user.

Overall, the report provided a complete picture of the effects of a change, with the exception for the presentation of the periods. The fact that the expert who found the bug lowered its score to 2 shows the importance of accuracy in the report: any flaws will lower the trust of users in the tool. After fixing this bug, the report would have to prove itself in practice to be more confident about the completeness of the results.

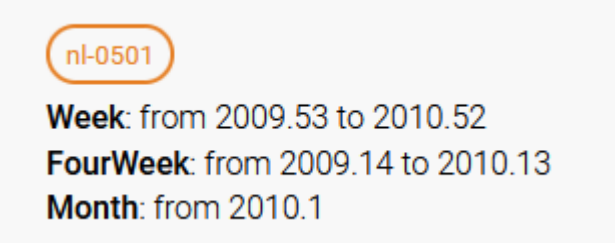

Figure 6.5: Bug in periods

Correct The correctness of the tool is rated with a 5 unanimously, meaning all experts fully agreed with that the presented effects are in fact a consequence of the changes to the configuration. Like with the completeness, the experts could not state with certainty that all the results are correct because of the scale of the report. The experts mention that this is something that will become apparent when the tool is used in practice. From the current reports, the experts conclude that the presented effects are correct. For this, they mainly refer to the traces which show the path through the configuration which made sense to them.

Non-trivial Two experts fully agreed with the statement that the presented effects were not clear before running the tool with a score of 5. The other experts agreed with a score

of 4. All experts mention that some of the presented effects are clear beforehand. The administrators and testers know the structure of the configuration and can therefore to a certain extent predict the impact of a change. The magnitude of effects on other configuration entities was however experienced as non-trivial. Also, one of the experts mentioned that the list of affected CLA's at the top of the report provides a quick realisation of the impact of a change. One expert praised the information given by the traces which provides insights that were not apparent before using the tool. Another expert was happy with the level of detail the report provides about changes to specific configuration entities, but also mentioned it could sometimes be too much.

The ratings about the non-triviality show that the tool is a welcome addition to the set of tools of the administrators and testers. It provides insights which could not be obtained easily before and helps users to determine the full impact of a change.

Insightful The insightfulness of the tool is rated with three 5's and one 3 from an expert who did neither agree nor disagree with the statement that the report provides insights needed to fix unforeseen effects. The experts who fully agreed with the statement mentioned that the report provides clear insight in what changes and the traces provide a lot of information about where a specific effect originates. Using these traces, the experts feel like they have enough information to fix problems that may be found. The same holds for the affected CLA's and periods, one expert mentions, since the report provides a clear overview of this allowing the user to quickly discover flaws in the configuration. Expert 2 who gave a score of 3 was mainly overwhelmed by the size of the reports. The report provides a lot of information which this expert finds hard to interpret and convert to action to fix the configuration.

The scores show a division in the way experts conceive the information in the report. On the one hand, three of the experts are clearly convinced of the insightfulness of the results and have confidence that the tool will help them to discover and solve problems in the configuration leading to a higher quality product. On the other hand, the expert who was overwhelmed by the size of the report shows that the presentation of the results might not be perfect for every domain expert. This could possibly be solved with training in interpreting the report but might also require a different approach to presenting the results.

Valuable Three experts agreed that the report is worth the time it takes to look through it, of which two experts even fully agreed. Expert 2 neither agreed nor disagreed. For two of the experts, the value of the tool was clear, and they fully agreed with the statement, which they emphasised by stating that they already would like to use the tool in their daily work. A third expert mentioned that the value of the tool is dependant on the situation. The more complex a change is, the more value the report can provide to understand the effects of the change. The complexity of the change however also increases the time needed to inspect the report. The last expert saw the value of the information at the top of the report: the list of CLA's and periods that are affected by the change. This can be checked in a glance to see whether the change

had unintended consequences. This expert however said the list of changed outputs with details like the traces are too complex and time consuming to inspect.

The scores for this heuristic provide a good understanding of how willing the administrators and testers are to use this tool in their daily work. The experts think the tool can provide valuable insights which help them to determine the correctness of their change. Like with the ratings for the insightfulness, the value depends on the understanding of the tool, whereas not being able to interpret the result efficiently will lower the value and will hold back the users from using the tool in their workflow.

Logical With to regards to how logical the user interface of the tool is, two experts rated the heuristic with a 5 and two with a 4. At the start of the interview, the experts received a short explanation of the functionality within the report. With this explanation, all experts could navigate through the report without any problems. One expert commented that the font was clear, and the report was easy to navigate. One expert added a feature suggestion to allow users to filter traces on CLA's.

While improvements in terms of user experience can always be made, the experts were positive about how logical the user interface is. This provides the confidence that change impact analysis results can be presented to the users in an understandable way.

Clear Three experts agreed the meaning of each effect becomes clear from the given information, one of which fully agreed with the statement. Expert 2 disagreed with the statement. The experts who agreed with the statement stated that the report provides a lot of information to understand what each effect means. One expert mention that an explanation of what all information meant was necessary to fully understand the results, while another expert mentioned that a lot of presented effects will not be clear immediately but provide enough details to allow the user to investigate what the meaning of a certain effect is. Expert 4 did not understand where the presented effects originated, which also made it difficult to assess whether the presented effects needed any further action. One of the experts added the suggestion to create a visual representation of the traces to help with understanding the meaning.

The report provides a lot of detail which can help users investigate problems. This can however overwhelm users leaving them unsure what to do with the information. An extension of the tool could be a more sophisticated analysis of the found effects to point the users to possible issues, providing a more steered approach. Overall, the experts where however positive about the information provided allowing them to perform the analysis themselves.

Clean The 'clean' heuristic had a similar evaluation as the 'clear' heuristic where three experts fully agree with the statement that the report does not provide too much information and one expert fully disagrees. Also for this heuristic, the experts who agreed with the statement stated that the liked the amount of information which allowed them to dive into the results themselves. They mention it does provide a lot of information, but do not think it is too much because of the way it is presented and because of the

possibility to filter results. Expert 2 again mentions that there is too much information in the report to go through the report.

About the methodology of using change impact analysis, one expert mentioned that currently they have to hope that they have tested every important scenario and have not looked over a bug in a specific case. According to the expert, the tool provides a lot of information from which every little insight is welcomed and will most likely help to reduce the risk of failures in the software. The expert who was more critical about the amount of information still sees the value of using change impact analysis in their work but would like a more steered report where it already performs analysis and leaves out irrelevant information before presenting it to the user. All experts agree that using this report as an addition to the code review for the pull requests is a good addition to their workflow.

Using the results from the common scenario evaluation and the insights from the user evaluation, we see that using change impact analysis can benefit domain experts to be more involved in the quality process. The common scenario evaluation showed that the most relevant scenarios which cause problems are captured by the tool and others can be incorporated by extending the model. In addition, the user evaluation showed that experts are positive about the value of the approach for their workflow, while there are some improvements to be made to make the tool more user friendly for all users. Both evaluations provide a good intuition that change impact analysis is an effective methodology to enable domain experts to participate in the process of improving software quality.

Chapter 7

Related work

This chapter presents previous work related to involving domain experts in the process of improving software quality. In section 7.1, different methods to involve non-software engineers with software quality are considered. Section 7.2 then continues by describing the field of change impact analysis and its connection to involving non-software engineers.

7.1 Involving non-software engineers in software quality

In an empiric evaluation at three companies, Mäntylä et al. [15] found that testing software is not an action that is solely performed by specialists. A lot of defects in software are discovered by employees in different areas than the software development or testing. The knowledge of domain experts helps them to design more effective test cases and allows them to recognise more failures in software [9]. This is in line with the goal of Exact, where they want to benefit from the knowledge of domain experts to test context-specific configurations.

During the development of software, non-software engineers play a large role in defining the requirements and assessing whether the requirements have been met. Using requirements engineering [17], domain experts can be involved in determining the requirements for a software product. These requirements can then be used to steer the development. The requirements can also be formalised in stories which can be used to test the software [18]. Instead of needing a software developer to build the tests, domain experts can define the requirements in natural language which can then be translated to test cases [10].

Another way to involve domain experts in the testing of software is to build an interface which allows them to specify automated tests without the need for formal modelling or programming skills [2]. Bärisch found that their approach of Model-Driven Test Case Construction which allowed non-specialised testers to create automated tests was understandable and usable by domain experts given minimal training and preparation.

Instead of manually expressing the requirements, tests can also be created by recording the interactions of a user with the interface and replaying the recording to test the expected functionality [21].

While some of these methods to let domain experts create tests were considered for the

case of Exact, this was not the chosen approach. This is mainly due to the nature of the subject under test. Since the configuration only specifies the method of calculation, using behaviour driven tests is cumbersome and will not achieve the desired goal. Also using other forms of testing like replay testing results in a lot of work for the domain experts, since a small change to the configuration would have a large impact on the number of tests that need modification.

7.2 Using change impact analysis

Change impact analysis can be applied to software engineering with several goals [12]. One of the main goals of change impact analysis is to help developers comprehend a software system to determine the scope of a change before the change is made [23]. Also, it can provide insight in change propagation where other entities in a software system need to be updated after a change to be consistent with the changed entity [8].

Change impact analysis is used in software testing to determine the impact of a change in the software on the tests. In this way, regression testing can be performed faster because only tests that cover impacted areas have to be run [20]. Conversely, once a test fails, change impact analysis can help by providing insight in which code change had impact on the test case and caused it to fail [7].

Change impact analysis can also provide metrics about the state of the software, for example to describe the maintainability of different parts of a system in terms of the impact changes in this part have had in the past [6].

Change impact analysis can also play a role in code review. This mainly helps developers understand the change and the impact of the change, but also provides a focus on what should be reviewed more thoroughly. An example of a tool that aids developers with their code reviews using change impact analysis is *BLIMP Tracer* [22]. This tool looks at the impact of changes on the software build and determines which changes need a more careful review. Another example is *TaintImpact* [14], which uses a git commit to compute an impact set of a change using dynamic taint analysis with the goal to help developer focus their attention on parts of the code that are not obviously impacted by a change and finding bugs in changes.

The proposed method for the case of Exact combines several of these goals. The primary goal is to create comprehension of the configuration to non-software engineers to let them understand the effects of their change. The tool will be used in the review phase to determine whether intended effects are correct, but also to provide a focus on which parts of the configuration might need more manual testing.

Chapter 8

Discussion

The goal of this thesis was to investigate how domain experts without software engineering background can be involved in the process of improving software quality. In the thesis, the method of change impact analysis has been proposed to bridge the gap between software development and domain experts' knowledge and has been implemented and evaluated for the case of Exact. This chapter discusses the found results and the implications of these findings in section 8.1, the limitations of the method and threats to the validity of this research in section 8.2 and future work in section 8.3.

8.1 Results and implications

The steps in this research were focused on answering four research questions to understand the challenges domain experts run into, to formalise the software into a model, to use the model to provide useful insights and to evaluate whether this methodology enables domain experts to participate in improving software quality. This section describes the answers to each of the research questions and discusses the implications of these results.

> **RQ1** What are the challenges domain experts without software engineering background run into when participating in the process of involving software quality?

In the interviews with the different stakeholders at Exact as described in chapter 3, several challenges which make it more difficult for domain experts to participate in the process of improving software quality were identified. One of the main insights was that they had trouble comprehending the system they are validating. Because the domain experts do not understand how the system operates, they have difficulties with predicting which areas of the software will be affected by a change. Because of this, unintended effects of the changes are not discovered in time.

The domain experts at Exact examine the source code when reviewing a change to the configuration. This means that they have more information to determine which areas of the software will be affected. The problems with comprehending the structure of the system and the implications of changes will likely be worse for domain experts who do not have access to this information from the source code.

RQ2 How can the software under test be formalised in a model which can be used to analyse the impact of changes?

Chapter 4 shows how the software under test can be modelled as a graph to allow change impact analysis. The graph contains nodes for each entity in the software and edges annotated with information to traverse the graph to determine impact.

Already during development, the potential usages of such a model became apparent. Next to change impact analysis, the model could already be used for various analysis purposes. By formalising parts of software that are not yet covered by static or semantic analysis, new analysis can be done to increase the quality of the software.

The case of Exact showed a domain specific language, developed for the use case of payroll calculations. Implementing the conversion from the configuration files to the model required a dedicated parser and converter. For every change to the structure of the configuration, the code that reads the configuration and builds the model has to be changed as well. When developing a new product using a domain specific language, the step of converting the DSL to a model can be shared between the software and the change impact analysis tool, causing less overhead.

RQ3 How can the model of the software be used to provide useful insights to domain experts to help them assess the quality of changes?

In chapter 5, the model created from the configuration is used to develop a change impact analysis tool which detects changes using git and traverses the graph to determine the configuration entities that are possibly affected. These results are presented to the user in a report which can be attached to a pull request. The evaluation using common-scenarios as described in section 6.1 showed that converting the software artefact into a model and using this model to perform analysis on software changes is an adequate method to detect common problems in terms of software quality.

This methodology could also be applied outside of the context of enabling domain experts to participate. By modelling the software and generating insights about the impact of changes, also software engineers could be alerted to identified problems.

This concept of creating a model to perform change impact analysis could also be generalised to broader contexts, for example for a specific programming language. At Exact the change impact analysis was performed on a domain specific language in the form of JSON configuration files which was very specific for the case of Exact. If it would be possible to build a tool that instead of analysing one specific software context would predict the impact of changes to source code based on constructs in programming languages, the concept of change impact analysis could be applied more broadly for both software engineers and domain experts.

RQ4 Does the methodology of using change impact analysis enable domain experts to participate in the process of improving software quality?

The evaluation with users of the tool pointed out that the majority of the users is positive about the concept of using change impact analysis to validate the changes. The evaluation

showed that the method provides useful information to the domain experts which helps them to comprehend the impact of the change on the software product. These insights allow them to discover flaws in the change and act on these problems, which increases the quality of the software.

The evaluation also showed that the value of change impact analysis differs per scenario and per individual. Some introduced bugs are not apparent from a change impact report and might need further analysis to be detected. Also, while most users saw the value in the information provided by the report, not all users were able to interpret the given information and use this to assess the correctness of the software. This might be something that can be improved with training and experience but might also be an indication that this methodology of change impact analysis is not fit for every domain expert.

The evaluation also showed the need for accurate results from the change impact analysis tool. When an evaluator discovered a flaw in the reports, their trust in the results of the tool decreased significantly. Not knowing whether the results presented by the tool are complete will discourage domain experts from using the tool in their workflow.

Overall, the implementation of change impact analysis in the workflow of the CLA team at Exact can be seen as a proof of concept of how change impact analysis can enable domain experts to better assist in validating the quality of the product. Letting domain experts assess whether the changes have the intended effect serves as a form of code review, allowing them to be directly involved in the development process. Because the current methodology creates an over-approximation of the possible impact and does not run the software with concrete values, the tool cannot yet replace existing testing methodologies such as regression testing and manual testing in the workflow of domain experts. The tool can however provide more confidence to developers and domain experts when delivering their work and provide testers with guidance on what to test more thoroughly.

8.2 Threads to validity

While the proposed method as described in this thesis provides a good intuition that change impact analysis can help domain experts to participate in improving software quality, there are some limitations which can influence how valid the results are for this context and how well results transfer to other contexts.

Internal validity This research did not perform a long-term user evaluation to determine the effect of using this methodology in practice. While the common-scenario evaluation and user evaluation attempted to gather enough data to show the practical value of the methodology, this is not a guarantee that the methodology will in fact increase the quality of the software. Factors like user adaptation, long-term maintenance and the discovery of other problems in the concept or the tool might hinder the achievement of this goal in the long-term. In addition, the long-term success of the change impact analysis depends a lot on the maintainability of the tool itself. During the user evaluation, the need for highly accurate results became apparent. If the report shows wrong or too trivial results, the user will not see the value in examining the report on every change.

The method of using heuristic evaluation for a user evaluation is not uncommon. However, because the user evaluation focused on different aspects than usability only, custom heuristics were used to evaluate the value of the report for the domain experts. These heuristics have not been proven or evaluated before, which means they do not guarantee correct results. Nevertheless, as the evaluators were the target group for the tool and can as such be seen as experts to determine the value of the solution, the result from the heuristic evaluation do provide an insight in how well the method of change impact analysis helps the domain experts in their workflow.

The developed proof of concept does not cover all scenarios, since this was too broad for the scope of this research. Also, because the tool has not been evaluated for a longer period of time, there might be bugs in the implementation, as was seen during the user evaluation. The evaluation focused on the value of change impact analysis in general, so although having bugs in the implementation could have influenced the perception of the tool by the evaluators, it should not have a significant impact on the results of this research.

External validity For Exact, the domain experts themselves are responsible for maintaining the part of the software that they test. This means that they have more technical knowledge about the structure of the software than a domain expert without any technical background, which most likely allows them to better understand the change impact analysis report since they can connect presented effects with concepts from the code. There might still be value in the methodology of change impact analysis for domain experts without any technical background, but for this, the report should abstract away any technical details and only give insights that can be interpreted without the technical background.

Similarly, the effectiveness of change impact analysis might depend heavily on the context in which it is applied. The expertise of users, the complexity of the software under test and the domain context might influence how well the impact of changes can be analysed and how well domain expert can interpret findings by a tool and convert these findings to actions that improve the quality of the software. This research only evaluated the methodology with one team within one company with a specialised workflow and domain context, which means that results in other teams and companies may vary.

8.3 Future work

While the work at Exact provides a first idea of how change impact analysis can help domain experts participate in improving the quality of software, more work can be done to stimulate their participation.

During the interviews, several challenges were identified that degrade the participation of domain experts in the quality process. In this research, only one of these challenges - assessing the impact of changes - was explored. Future work can investigate how a way to create automated tests, a library of scenarios and an improved regression test can enable domain experts to participate.

As discussed in the threads to validity, the value of change impact analysis for domain experts has to be evaluated more thoroughly. To properly evaluate the value of the method

in the long run, users should be using it in practice and the adaptation by domain experts and the effects on the quality should be studied. In addition to a more proper form of evaluation, the method should be evaluated in other contexts. As described, the context in which the tool was evaluated is quite specialised, which means results might not be transferable to other teams. By implementing the methodology in other companies with different contexts, more insight about the merit of the method can be gained.

The user evaluation showed that not every user was able to extract the information in the same way, which could be improved by providing a more steered approach to presenting the found effects. More work can be done to determine the best way to present change impact analysis information to domain experts. Improving the way the results are presented will most likely also improve the efficacy of the method.

Chapter 9

Summary

Software engineers often lack the domain knowledge needed to validate context specific parts of software. Domain experts do have this knowledge, but often lack the expertise and tools to apply this knowledge in a way that tests the software product. Transferring this domain knowledge is inefficient and can lead to bugs for end-users. This research investigates how domain experts can be enabled to validate the context specific parts of the software themselves with minimal effort from software engineers. For this, the payroll team at business-software company Exact where domain experts are responsible for maintaining the configuration of collective labour agreements (CLA's) used in the calculation of payslips is used as a case study.

To understand the challenges domain experts run into when participating in improving quality, interviews with the domain experts at Exact are conducted to identify several challenges. One challenge was that they had trouble assessing the impact of changes to the configuration. Because the configuration is complex and domain experts do not comprehend the structure of the system, they have difficulties with predicting which areas of the software will be affected by a change, leading to unforeseen behaviour for end-users. This research proposes the use of change impact analysis to overcome this problem. If a tool can give insights to domain experts about what the impact of a change is, the domain experts can use their knowledge to verify the correctness of this change. By creating a proof of concept for the case of Exact, this idea is evaluated.

To be able to analyse the configuration, the configuration first had to be formalised into a model. By converting the configuration to a graph representation, the configuration becomes traversable to determine which areas will be affected by a change. The graph contains nodes for all configuration entities and edges between entities that have impact on each other. An evaluation using common-scenarios shows that this model is effective in identifying scenarios which are known to cause problems. The model can be extended to identify more cases.

Using the constructed model, a tool is created that – based on changes in git – generates a report that presents the impact of the changes. A user evaluation shows that the method is effective in aiding domain experts to understand the impact of changes which provides a good intuition that change impact analysis is an effective method to enable domain experts to participate in the process of improving quality.

Bibliography

[1] Liliane Barbour, Foutse Khomh, and Ying Zou. Late propagation in software clones. In *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, pages 273–282. IEEE, 2011.

[2] Stefan Bärisch. *Model-driven test case construction by domain experts in the context of software system families*. PhD thesis, Uni Kiel, 2009.

[3] V.R. Basili and R.W. Selby. Comparing the effectiveness of software testing strategies. *IEEE Transactions on Software Engineering*, SE-13(12):1278–1296, 1987. doi: 10.1109/TSE.1987.232881.

[4] M Bialy, V Pantelic, J Jaskolka, A Schaap, L Patcas, M Lawford, and A Wassyng. Software engineering for model-based development by domain experts. In *Handbook of System Safety and Security*, pages 39–64. Elsevier, 2017.

[5] Dines Bjørner. Domain engineering. In *Formal Methods: State of the Art and New Directions*, pages 1–41. Springer, 2010.

[6] M Ajmal Chaumun, Hind Kabaili, Rudolf K Keller, and François Lustman. A change impact model for changeability assessment in object-oriented software systems. *Science of Computer Programming*, 45(2-3):155–174, 2002.

[7] Ophelia C Chesley, Xiaoxia Ren, and Barbara G Ryder. Crisp: A debugging tool for java programs. In *21st IEEE International Conference on Software Maintenance (ICSM'05)*, pages 401–410. IEEE, 2005.

[8] Ahmed E Hassan and Richard C Holt. Replaying development history to assess the effectiveness of change propagation tools. *Empirical Software Engineering*, 11(3): 335–367, 2006.

[9] Juha Itkonen, Mika V Mäntylä, and Casper Lassenius. The role of the tester's knowledge in exploratory software testing. *IEEE Transactions on Software Engineering*, 39 (5):707–724, 2012.

[10] Sunil Kamalakar, Stephen H Edwards, and Tung M Dao. Automatically generating tests from natural language descriptions of software behavior. In *ENASE*, pages 238–245, 2013.

[11] Shahedul Huq Khandkar. Open coding. *University of Calgary*, 23:2009, 2009.

[12] Bixin Li, Xiaobing Sun, Hareton Leung, and Sai Zhang. A survey of code-based change impact analysis techniques. *Software Testing, Verification and Reliability*, 23 (8):613–646, 2013.

[13] Rensis Likert, Sydney Roslow, and Gardner Murphy. A simple and reliable method of scoring the thurstone attitude scales. *The Journal of Social Psychology*, 5(2):228–238, 1934.

[14] Tobias Lüscher. Taintimpact: Taint-based change impact analysis. B.S. thesis, ETH Zurich, 2021.

[15] Mika V Mäntylä, Juha Itkonen, and Joonas Iivonen. Who tested my software? testing as an organizationally cross-cutting activity. *Software Quality Journal*, 20(1):145–172, 2012.

[16] Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '90, page 249–256, New York, NY, USA, 1990. Association for Computing Machinery. ISBN 0201509326. doi: 10.1145/97243.97281.

[17] Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, page 35–46, New York, NY, USA, 2000. Association for Computing Machinery. ISBN 1581132530. doi: 10.1145/336512.336523.

[18] Shelly Park and Frank Maurer. A literature review on story test driven development. In *International Conference on Agile Software Development*, pages 208–213. Springer, 2010.

[19] Reinhold Plösch, Harald Gruber, Christian Körner, and Matthias Saft. A method for continuous code quality management using static analysis. In *2010 Seventh International Conference on the Quality of Information and Communications Technology*, pages 370–375. IEEE, 2010.

[20] Xiaoxia Ren, Fenil Shah, Frank Tip, Barbara G Ryder, and Ophelia Chesley. Chianti: a tool for change impact analysis of java programs. In *Proceedings of the 19th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 432–448, 2004.

[21] John Steven, Pravir Chandra, Bob Fleck, and Andy Podgurski. jrapture: A capture/replay tool for observation-based testing. In *Proceedings of the 2000 ACM SIGSOFT international symposium on Software testing and analysis*, pages 158–167, 2000.

[22] Ruiyin Wen, David Gilbert, Michael G Roche, and Shane McIntosh. Blimp tracer: Integrating build impact analysis with code review. In *2018 IEEE International conference on software maintenance and evolution (ICSME)*, pages 685–694. IEEE, 2018.

[23] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, 31(6):429–445, 2005. doi: 10.1109/TSE.2005.72.