Building a workbench to improve sharing and reproducibility of MOOC experiments

Master's Thesis

JLM de Goede

Building a workbench to improve sharing and reproducibility of MOOC experiments

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE
TRACK INFORMATION ARCHITECTURE

by

JLM de Goede
born in Nuenen c.a.

Web Information Systems
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
http://wis.ewi.tudelft.nl

Building a workbench to improve sharing and reproducibility of MOOC experiments

Author: JLM de Goede
Student id: 4091876
Email: J.L.M.deGoede@student.tudelft.nl

Abstract

This thesis improves sharing of code and reproducibility (S&R) in research for massive open online courses (MOOCs). Reproducibility is recreating an experiment by a different researcher. Science in general struggles with reproducibility. MOOC experiments often contain useful code that could be used by other researchers, but that code is oftentimes not shared with others. To improve S&R in MOOC research, this thesis first identifies the challenges MOOC scientists encounter when trying to share code and when making their experiments reproducible. Then, user interviews are performed to further understand MOOC research and to better understand the challenges identified earlier. A conceptual experimental workflow is designed and implemented in the form of a workbench. The workbench is then evaluated.

The identified challenges based on literature with regards to S&R are: (1) Difficulty in selecting the right tools, (2) lack of a standardized workflow that enables reprodubility and (3) manual work required to enable reproducibility without a clear incentive to perform that work. From the user interviews, researchers indicate experiencing these same challenges. Based on these challenges, I propose an experimental workflow that focuses on making research reproducible and on sharing code. This workflow is implemented in the form of a workbench, where researchers can create and manage their MOOC experiments. This workbench allows for sharing code between researchers and is evaluated using a real-world data science task by three actual large-scale learning analytics researchers and a visiting volunteer researcher. The evaluation finds that the workbench needs more work to be suitable for actual MOOC research use and that more researcher education is needed to improve sharing and reproducibility in MOOC research.

Thesis Committee:

Chair:	Prof. dr. ir. G.J.P.M. Houben, Faculty EEMCS, TU Delft
University supervisor:	Dr. ir. Claudia Hauff, Faculty EEMCS, TU Delft
University co-supervisor:	Dr. ir. Christoph Lofi, Faculty EEMCS, TU Delft
Committee Member:	Dr. Sebastian Erdweg, Faculty EEMCS, TU Delft

Preface

I would like to thank my thesis supervisor Claudia and co-supervisor Christoph, for providing me with support and feedback during this thesis. Also, I would like to thank all the PhD researchers from the large-scale learning analytics department of WIS for their input and participation in this process. Finally, I would like to thank my boyfriend, my parents, sisters and my friends for their support, not only during this thesis but also during the complete time of my study.

JLM de Goede
Delft, the Netherlands
July 14, 2017

Contents

List of Figures

Chapter 1

Introduction

1.1 Context and motivation

MOOCs, also known as *massive open online courses*, have become a very popular learning platform for many students around the world since 2011 when Stanford University launched three online courses for free[1]. A MOOC is an online course that is accessible via the web where anyone can participate [22]. Their goal is to provide education at a university-grade level for free. Millions of students follow a variety of MOOCs[2] and MOOCs have the advantage over traditional college since students can set their own pace, by for example rewinding or fast-forwarding the lecture for a topic that is harder or easier for the student to understand [39]. MOOCs also have some drawbacks: They predominately reach already privileged learners[3], most MOOCs are not accredited, and if they are, it is often not recognized [29] and student assessment is sometimes difficult when a large number of MOOC students are enrolled [29].

The large-scale and online nature of MOOCs provide ample research opportunities[4]. Research into MOOCs can help the science of learning by providing new large-scale experiments and new sources of data [38]. Students from many different educational backgrounds and all parts of the world are following MOOCs. This allows for statistically valid research, as the population is so diverse. The first generation of MOOC research was mainly concerned with case studies and the educational theory behind MOOCs [13]. The sudden influx of MOOCs led to a huge amount of data being generated that can provide insight into student behavior and can provide better support to students [13].

One of the domains in MOOC research is Learning Analytics and Knowledge (LAK). The term Learning Analytics is used 'for studies aimed at understanding and supporting the behavior of learners based on large datasets' [13]. The focus of this thesis is on learning analytics researchers.

Many computer scientists from different research fields struggle with making their source code available and creating reproducible experiments. For example, in a study

[1] http://www.nytimes.com/2012/07/18/education/top-universities-test-the-online-appeal-of-free.html

[2] https://twitter.com/edXOnline/status/631844606964035588

[3] http://www.chronicle.com/blogs/wiredcampus/moocs-are-reaching-only-privileged-learners-survey-finds/48567

[4] http://conference.oeconsortium.org/2015/presentation/mooc-research-what-can-we-do-with-big-data/

done by the University of Arizona, less than half of the software created in research papers could be successfully built or installed [30], others reported similar difficulties [4].

The focus of this thesis is to improve the reproducibility of a MOOC experiment. Reproducing an experiment means duplicating it, either by the same researcher or by someone else with either the same or a different dataset. This thesis focuses on reproducing an experiment by a different researcher with a different dataset and uses the following definition of an experiment: 'A scientific procedure undertaken to make a discovery, test a hypothesis, or demonstrate a known fact'[5]. Furthermore, for a MOOC experiment, this thesis only considers experiments that use programming to achieve their result. This means that, for example, a thought experiment is not in the scope of this thesis.

To enable reproducibility of computer science experiments, a lot of manual work is required. Due to a lack of resources and/or incentives of modern research, that work is not always performed [28]. Another difficulty for enabling reproducibility is selecting the appropriate tools to help reproducibility. For example, choosing a version control system and choosing testing, documentation and package management libraries. Furthermore, the lack of a standard workflow on what tasks should be accomplished to enable reproducibility is an issue. These are tasks such as defining dependencies in a standard manner, writing documentation and how to organize the source code.

Another part of reproducibility is the ability to share reusable parts of code of an experiment. If more code is shared and reused among MOOC researchers, creating an experiment might be easier. The definition of reuse is the use of existing software components to construct new systems [36], in this case to construct new experiments. Re-usable code means more eyes looking at the code, which will lead to more stable code and a more efficient experiment creation process, which might improve MOOC research as a whole. Sharing reusable parts of an experiment also helps reproducibility, as existing, tested, peer-reviewed code is used in the creation of an experiment. It ensures the code runs in different computing environments.

1.2 Objective of this research

This research will help improve the ability to share and reproduce experiments of MOOC research by building a workbench targeted for large-scale learning analytics researchers. The purpose of this workbench is that it allows researchers to share more easily reusable parts of their experiment and reproduce theirs and others experiments. Before such a workbench can be created, it is first necessary to understand what sharing and reproducibility exactly means for MOOC researchers and how MOOC researchers are currently performing their experiments.

The main goal of this research is to improve and understand reproducibility in MOOC research through a literature review, user interviews with a representative group of large-scale learning analytics researchers and through the creation of a workbench.

[5] https://en.oxforddictionaries.com/definition/experiment

1.3 Steps and questions

The research questions for this thesis are as follows:

1. What are the (software) requirements for learning analytics researchers to enable reproducibility and sharing of reusable parts of their experiments?
2. Having defined the requirements, how does a good experimental workflow design look like?
3. How can the designed workflow be implemented with modern day web engineering practices?
4. Having implemented the design, does the implemented solution achieve its requirements defined by RQ1?

1.4 Approach and outline

The first research question is approached by doing a literature study on MOOCs and MOOC research. This literature study looks at current reproducibility issues in science in general and specifically for MOOC research. It also looks at current approaches being employed to increase reproducibility of experiments. The end result of this literature study is a list of challenges that, when tackled, are likely to improve the reproducibility of experiments in general. This literature study is the subject of Chapter 2.

These challenges serve as input for Chapter 3, where again the first research question is under consideration. This time, user interviews are performed to further identify any new challenges and gain more insight into how MOOC experiments are currently performed.

The second research question, designing the experimental workflow, is answered by considering the input of the answer to the first question, so based on the requirements. Here, a conceptual design of the experimental workflow is proposed. This chapter also studies other workflows in other domains, which are studied for inspiration and to base off the design of a solution to this problem. This design is the subject of the fourth chapter.

In the fifth chapter, the conceptual design is implemented, by means of a use case at the Delft University of Technology. This chapter also studies modern engineering approaches for inspiration. The end result is the implementation of the workbench.

Finally, the fourth research question is considered in Chapter 6. Since the implementation is now complete, it is imperative to investigate if the workbench has the desired properties and find out if it realizes the desired goals. The outcome of this chapter is an analysis regarding the performance of the workbench as well as conclusion about that usage. In this evaluation, researchers perform real-world tasks in the workbench.

Chapter 2

Literature study

This chapter helps find the (software) requirements of learning analytics researchers for sharing and reproducing experiments, the first research question, based on literature. In the first sections of this chapter literature on MOOCs and MOOC research is discussed, to gain a better understanding of what MOOC research exactly is. The next section will review literature with regards to sharing and reproducibility in computer science and in MOOC research.

2.1 MOOCs

The purpose of this section is to give background information on MOOCs, on how MOOCs are used and on what characterizes MOOCs. Since this thesis tries to improve sharing and reproducibility of MOOC research, it is useful to gain some background information and insight of MOOCs.

A MOOC is short for 'massive open online course'. It is an online course where an unlimited number of people can take part and that is openly accessible via the web. They require on the part of the students some self-motivation, because learning can happen anywhere in the world, so also outside of the classroom [20]. MOOCs do not require students to complete the course or to have an academic qualification [20]. Although many students take advantage of a MOOC, the completion rate of a MOOC is only about 15 percent of those who enroll[6]. This does not have to be an issue though, considering that different people have different objectives for MOOCs, which might not include completing it.

Usually, tens of thousands student enroll in a MOOC. However, the 'massive' characteristic is not about numbers, but about scalability. A MOOC can in principle scale infinitely. The second characteristic is that a MOOC is open. This means that anyone can take part in a MOOC. No pre-requisite required. The third characteristic is that MOOCs are online and final characteristic is that a MOOC is a course. This means all the open educational resources are organized as a course.

MOOCs are designed based on two philosophies, namely xMOOCs or cMOOCs [42]. xMOOCs are based on a traditional university course. This is the most popular format for a MOOC course and is used on platforms such as Udacity, Coursera and edX. The cMOOCs, where the c stands for connectivist, involve a group of people

[6] http://www.katyjordan.com/MOOCproject.html

learning together. There is no individual instructor in a cMOOC: They often include blogs, learning communities and social media with learning content and promote interaction. In a cMOOC every student is also considered a teacher.

2.2 MOOC research

The purpose of this section is to find out what MOOC research is, which domains in MOOC research exist and where this kind of research is headed. This helps to provide more insight into the MOOC workbench and also ensures that the MOOC workbench is tailored towards the specific needs of MOOC researchers.

2.2.1 What is MOOC research?

While the first MOOCs appeared in 2011 [31], papers on what we now call MOOCs began to appear in 2008 [25] and gained much interest from the academic community since. MOOC research is dominated by the education, engineering and computer science disciplines [46]. According to the literature review of Veletsianos et al. 23.1 percent of their literature corpus consisted of research done by a computer scientist [46]. Only the education discipline has a larger percentage, with 32.8 percent of the studies in their corpus done by someone from that discipline.

There are gaps in several areas of MOOC research, due to several reasons [5]. Data protection concerns are one of the reasons. These are concerns regarding personally identifiable information and ownership of the data. Other hurdles are a tendency to hoard data which results in less data sharing and ethical hurdles for qualitative research [5].

Research on MOOCs is done on a variation of topics. Topics include student experiences, cost, performance metrics and learner analytics, MOOC policy and alternative MOOC formats. In general, however, according to Eichhorn et al. there are three universal MOOC research objectives: use of MOOCs, impact of MOOCs and MOOCs and teaching and learning [14].

The ‘Use of MOOCs’ research category is concerned with who is using the MOOCs. Research in this category can consist, for example, of a survey to find out more about a MOOC audience. This also helps to understand informal learning. Based on a survey from the University of California performed in Coursera for 4,200 MOOC students, most of the MOOC population are not students enrolled in a college [14]. They often have a full-time job. Another research category is impact of MOOCs, which is concerned with policies, and MOOCs and teaching and learning, which for example looks at learning pathways, to see how students are learning [14]

Learning Analytics and Knowledge (LAK) and Educational Data Mining (EDM) are closely related to MOOC research. LAK is described as ‘the measurement, collection, analysis and reporting of data about learners and their contexts, for purposes of understanding and optimising learning and the environments in which it occurs’ [43]. The other domain is Educational Data Mining which is data mining within an educational context.

Commonly used methods of analysis in learning analytics are prediction, clustering, relationship mining, distillation of data for human judgment and discovery with models [1]. Both LAK and EDM areas are related to each other. One paper even calls

for researchers of both disciplines to work and communicate more with each other [43].

2.2.2 Where is MOOC research headed?

There are many new frontiers for MOOC research. Eichhorn et al. identify three: [14]

1. changing faculty instructional approaches;
2. educational technology development and use of data science in education;
3. understanding diversity from MOOCs.

The second challenge is especially interesting, as it is focused on computer science. According to Eichhorn, MOOCs produce very large datasets and it is difficult to use these datasets to their full potential. This is because of the lacking data science methods which are required to analyze these datasets [14].

2.2.3 Conclusion

It is clear there are many new challenges for MOOC research. Based on literature, I can conclude that a number of these challenges are especially important to learning analytics researchers, namely new developments in educational technologies, the use of data science in education and improvements in sharing data. For these and other challenges to be addressed more effectively, MOOC science being done going forward would benefit from more focus on reproducible research and research where it is easier to share reusable parts of an experiment and the sharing of data.

2.3 Sharing and reproducibility in computer science

Reproducibility is the duplication of an experiment performed by the same or different researchers. There are several levels of reproducibility in computer science defined by different researchers. One definition states that reproducibility of a computer science publication can consist of four levels, namely: Publication only, publication plus code, publication plus code and data, publication plus linked executable code and data [34].

The Association for Computing Machinery, or ACM, a large society for computer science, is working on improving reproducibility in computer science research. One of the improvements they introduced are badges with different levels, such as the Artifacts Available badge which is given when the artifacts of the paper are made available[7]. These badges might work as an incentive to produce more reproducible work.

The ACM calls for the set up of an infrastructure that consists of, among other things, standards, guidelines and best practices for reproducibility of published research[8]. Specifically, ACM notes that sharing of artifacts should be encouraged. As of July 2015, it was too early to be prescriptive with regards to specific best practices and thus no challenges or best practices are identified by the ACM.

[7] https://www.acm.org/publications/policies/artifact-review-badging
[8] https://www.acm.org/data-software-reproducibility

In a study done in 2014 by FitzJohn et al. [16], a certain research question was asked which required some data analysis. Besides finding an answer to that question, the goal of the research was also to find out what it takes to make the data analysis of this research reproducible. In a blog post, the authors describe their challenges while performing the tasks involved with enabling reproducibility [9].

The main challenges and tools to enable reproducibility are, according to FitzJohn et al. functions with long-running calculations that are poorly documented, the automated caching of dependencies, version control for archiving purposes, automated checking of the code via continuous integration and documenting dependencies. The authors note that adhering to these principles requires a lot of manual work and that new tools are needed.

One approach for reproducible research comes in the form of containers by using Docker, which makes it easier to share executable code and data, independent of the underlying system. Boettiger [3] uses the containerized technology of Docker to make research done with R more reproducbile. The author identifies four technical challenges that hinder reproducibility. The first is dependency hell, where the exact dependency of a program is often unclear and can lead to different results even on the same computational platform [3]. The second is imprecise documentation, such as missing instructions to install and run the code. Even small gaps in the documentation can be major barriers [3], especially for novices in the field. Other problems with documentation are that the required parameters are not well enough specified.

The third issue is code rot. Because software is updated often, with added features, deprecated code or changed behavior, this can effect the results of the experiment [3]. The fourth and final identified technical problem are the barriers that exist towards adoption and reuse in existing solutions [3]. What this means, is that it is difficult for researchers to set up an environment in which they can easily optimize their code for reproducibility and sharing. Existing solutions are for example virtual machines, workflow software, continuous integration and other best practices from software development. These tools have extensive manuals before they can be used effectively [3].

The author solves this problem with containers by using Docker. In the container the researcher can store their dependencies, frozen in time. Using Dockerfiles, the problem with imprecise documentation is solved, because Dockerfiles provide an easy to write scripting language that describes all the steps needed to set up and run the container. Code rot is prevented by allowing researchers to use specific versions of their dependencies and the saving of the binary of that dependency. Only the fourth problem, the adoption barrier, is something that is not immediately solved with containers, as it is up to the researchers to use this tool.

Collberg et al. have given several recommendations to improve repeatability in computer systems research [37]. Repeatability differs from reproducibility in that repeatability is performing the exact same experiment twice under the same circumstances, either by the same or by a different researcher. That includes using the same dataset.

They investigate reasons why code cannot be shared and find versioning problems, the programmer left, bad backup practices and more. The main concerns regarding

[9] http://ropensci.org/blog/2014/06/09/reproducibility/

enabling reproducibility and repeatability are that it is difficult to accomplish. Collberg et al. find researchers that tried to improve repeatability in their department, by, for example, creating a web site for uploading research artifacts, but failed due to lack of usage. The authors make several proposals to increase repeatability, such as creating a permanent email address, backing up code, using version controls system, create a project website, plan for longevity and plan for repeatable releases.

2.3.1 moocRP

Pardos et al. build a MOOC workbench called moocRP [32]. The authors identify that earlier work still has some problem areas. One such problem area is 'the duplication of analytics work across platforms, institutions and individual instructors and their TAs'. The other is that replication of data intensive research results are difficult, due to logistical challenges. Meanwhile, the learning analytics community has the ability to help these institutions finding out what is actionable from their data and how instructors and students can take advantage of it. The goal of moocRP is to enable institutions to easily share data with the learning analytics community, that can then analyze the data and re-share the result with the institutions.

moocRP seems to be meant for the general MOOC community, which means not only researchers but also course designers and course instructors. As such, it focuses on ease of use and data visualization. It also has many out-of-the-box analytics modules, written by the moocRP developers and the MOOC community. These modules have the purpose to provide the MOOC instructor with insights in the MOOC itself.

moocRP has analytics modules that are finished, complete and provide direct insight into a MOOC. However, MOOC research in computer science is more concerned with the development of such a module, to, for example, see if some way of data science provides new insights.

2.3.2 Conclusion: Challenges for reproducibility

In conclusion, based on the issues described earlier with regards to sharing and reproducibility, I can conclude that the following high level challenges exist for sharing and reproducibility in computer science in general based on literature:

- Difficulty in selecting and using tools that enable reproducible research.
- Lack of a workflow on what tasks should be accomplished to enable reproducibility.
- Manual work required by researchers to enable reproducibility, with a lack of resources or incentives to perform that work.

Apart from those issues for CS in general, specifically for MOOC research I am able to identify the following challenge:

- Issues with sharing data: Too large data sets and concerns regarding data protection, among other things.

Chapter 3

Requirements analysis

The purpose of this chapter is to answer the first research question: 'What are the (software) requirements for learning analytics researchers to enable reproducibility and sharing of reusable parts of their experiments?'. The previous chapter accomplished that based on literature, this chapter approaches this research question by conducting user interviews.

3.1 User interviews

The purpose of the user interviews is to investigate how researchers are currently designing, building, sharing and reproducing experiments. For this purpose, MOOC researchers at the Web Information Systems (WIS) department of the University of Technology Delft (TU Delft) were interviewed. The TU Delft serves as a use case for this thesis and, as this research community is doing cutting-edge MOOC research [11] [6] [10] [7], they can be considered a representative group of people for the larger research community.

The goals of these user interviews are to find out:

- to what degree the identified challenges with regards to sharing and reproducibility in the previous chapter apply to the MOOC research community at WIS;
- how researchers are currently creating experiments (which methods, tools, approaches);
- what steps are already being taken to ensure reproducibility of the experiments created;
- how researchers are currently reproducing experiments and the problems experienced with reproducing an experiment;
- how researchers are currently sharing experiments and code, why they are sharing and the problems experienced with sharing.

3.1.1 User interviews set-up

The user interviews are semi-structured with a mix of closed and open-ended questions. The advantage of this type of interview is that it elicits the foreseen information,

but also has enough flexibility to allow unexpected information to surface [41]. The interviews were done individually.

The questions are formulated and the interviews conducted based on the best practices of M. Patton regarding qualitative evaluation and research methods [33], such as avoiding asking 'why' questions, only asking one question at a time, avoid asking leading questions, recording the interview and using an interview guide.

Each interview used a user interview guide. This guide contained the questions, as well as a short introduction, and some topics to be mentioned, if not already touched upon. At the start of the interview, a short description of the thesis project was given. Answers given by the interviewees remain anonymous. Each interview has been recorded and was transcribed afterwards, to make analysis of the qualitative data easier.

3.1.2 Conducting the interviews and results

In total I interviewed four people with the following roles at the time the interviews were conducted:

- interviewee 1 (I1): PhD candidate in large-scale learning analytics;
- interviewee 2 (I2): Master of Science student at WIS with the thesis topic on cheating in MOOCs;
- interviewee 3 (I3): PhD candidate in large-scale learning analytics;
- interviewee 4 (I4): PhD candidate in large-scale learning analytics.

The complete list of questions and summarized answers are given in appendix A.

Background information

The researchers are most active in the domains of LAK and EDM. Research topics include learning transfer, detecting multiple accounts cheating in MOOCs, designing interventions in MOOCs, finding out how student personalities affect MOOC learning and more. One of the interviewed researchers, I4, does not have a Computer Science background.

How are researchers currently creating experiments (which methods, tools, approaches)

There is no single approach with regards to how researchers are currently creating experiments. Often, it happens that previous work is used to start a new experiment.

If an experiment is based on previous work, it holds that code from previous work is often still applicable to the new work. Code is then carried over from one experiment to another, which happens quite often. This means that code from previous experiments is used in a new experiment and is modified to fit some new purpose. What kind of code is reused, varies. Sometimes, old code from data analysis work is reused in an experiment. For example, interviewee I3 states:

> *...I just use that code to build a big giant database, which can deal with data from every MOOC. So, every time if I need that experiment or a new*

> *MOOC, I just dump that data into the database. So, you can just regard it like very frequently.*

In terms of programming languages, researchers I1, I2 and I3 use Python to develop their experiments. Researcher I4 uses R.

Issues experienced with reusing own code are present. It happens that researchers no longer know what the code does or how it should be used. For example, researcher I1:

> *...sometimes when I go back [to previous written code, auth.] I find that even though I have some comment on that, I still need a lot of time to remind myself to think about what I am trying to do in that code.*

Researcher I3, when asked about difficulties understanding code, notes:

> *Yes, yes, issues like this happen a lot.*

Researcher I4 does not have issues with reusing code:

> *...since I have only been writing code for over a year now, it is not so far back I do not remember, so once I revisit my own code, I can pretty much get it.*

Researcher I2 does not have experience with reusing their own code.

In general, it can be said that MOOC research largely consists of two phases: a data gathering phase and a data analysis phase. In the data analysis phase, an important part is visualizing the data. In terms of tools, all researchers indicate that they use special data visualization tools for Python or R.

One of the researchers indicated the use of Jupyter [10]. This tool allows the writing of Python scripts in the form of a notebook, with parts of text explaining what the code does, followed by parts of code. The researcher indicates that this kind of workflow helps the understanding of the code, both for themselves and for other researchers. When asked about limitations, researcher I1 notes that code written in Jupyter cannot be used in other Python scripts and instead copies the code from Jupyter.

Which actions are already being taken to ensure reproducibility of experiments

Researcher I3 mentions that sometimes a deadline approaches and the focus is mainly placed on completing the project, instead of focusing on ensuring the code is able to be understood and run by others:

> *Honestly, I haven't thought about this, sometimes when the deadline is approaching, you are just... I quickly finish it.*

With regards to dependencies, researchers have no specific or default way of managing them, as indicated by researcher I1. Researchers install their Python dependencies system-wide and do not record version numbers. Researcher I1, for example, notes:

[10] http://jupyter.org

> *For my own work, I put all of the dependencies, libraries in the Python library of my computer.*

Researcher I4 uses R, where it is possible to install dependencies directly from the script:

> *When you run my code, what happens is, in my code I would include the installation and the loading of those libraries.*

Although three of the four of the researchers are familiar with a version control system such as git, use of it is limited. For example, researcher I2 notes with regards to pushing and pulling daily or weekly:

> *I just do [commit code with git, auth.]... not regularly, it just seems like, it is not a good habit, I think.*

Researcher I3 uses the version control system only as a back-up system:

> *I just use it like a place to store the code, in case my laptop is stolen by somebody else.*

Researcher I1 only commits and pushes code they consider to be complete and instead makes private backups, and notes:

> *After I think, okay, it's a version, this looks like complete work, sometimes I will update it on GitHub.*

The researcher indicates this to be a personal preference.

Written documentation by researchers comes in the form of source code comments. A README file is sometimes written when the experiment is going to be made public.

No testing is being done, apart from running the code until the code works, as indicated by all the researchers. Researcher I1:

> *I do not use specific testing tools or specific code for testing this part.*

Instead, they are familiar with the dataset, are able to extract minimum and maximum values and can recognize strange results. They also check the logic behind their algorithm, and if it works as designed. Researcher I1 further notes:

> *Actually, we do not spend a lot of time on the verification and on the unit tests. So, it's not a good habit, but I know that, it's that I am already familiar with this way to check the code, so that is why I do not use that.*

Sometimes testing is difficult or impossible, due to the lack of a testing or training set. Researcher I2:

> *I only have user logs from MOOCs created by TU Delft, I do not have something like, in machine learning you maybe have a training set or a tester set, something to verify it. (...) It is impossible in my master thesis.*

Researchers occasionally publish their source code on GitHub or on the Open Science Framework (OSF). Before they make their source code public, researcher I3 focuses on making the code more readable and erases any sensitive information such as passwords embedded in the source code. Researcher I1 writes a README before public release, containing dependency information, what the code does and how to run the code and some source code comments. For publication on the OSF, a manuscript was published in advance that outlined how the data was going to be analyzed. Researcher I4 follows the steps of the OSF to publish source code. Researcher I2 has not made source code available to the public yet, but plans on releasing it on GitHub after graduation.

When asked about how MOOC research could be made more reproducible, researchers mostly mention that they need a set of common tools. Researcher I1 states that code should be made more reusable in general, but that researchers only sometimes account for re-usability of their experiment:

> *I think for the most of the PhD students, if your topic is not focusing on the reusable part, maybe you would like to make your code run correctly and make sure that you can run the whole program fast and then get a result and then maybe you run to another topic, and this code is just there.*

How are researchers currently reproducing experiments and what problems do they experience when reproducing experiments

Researchers do not have much experience reproducing experiments from other researchers. Only researcher I2 has experience reproducing other people's work. In general, there are no pre-defined steps on how to approach the reproduction of a MOOC experiment and the researchers solve the challenges involved with reproducing an experiment as they appear.

Researcher I4 is working with another university to run their experiments in other MOOCs:

> *...maybe within a couple of years (...) we will have policies in place for reproducing other people's studies and MOOCs in TU Delft MOOCs. But for now, it is figuring out as we go, making sure all parties involved are happy and everything is ethically sound.*

Issues experienced with reproducing others work are the reluctance of researchers cooperating. Sharing happened via email and was required to be bidirectional, meaning the other party also wanted to receive source code. The researcher then prepared the source code for release, by adding a README file.

The main problems experienced with reproducing seem to be that often there is no source code present in a publication. Researcher I2:

> *On the basis of my experience, about the academic publication in MOOCs, I rarely see some people paste the code or the link to the code on their publication.*

When asked about how MOOC reproducibility could be improved, researcher I2 said:

I think open source is of course a good choice.

Also, researcher I2 found issues with differing data formats and the quality of the source code, or lack thereof.

How researchers are currently sharing experiments and code, why they are sharing and the problems they experience with sharing

Researchers share via different ways, sometimes via email, sometimes via a private Dropbox shares and sometimes via GitHub. Sharing code among researchers does not happen often. When it does happen, it usually means that the received code has to be rewritten because it is not well enough documented, does not fit the data or is unreadable.

The sharing problem is multifaceted. On the one hand, researchers are not aware of the existence of common MOOC tools, because no central 'marketplace' exists where they can find these tools. Another issue is that when they do find the code, it lacks documentation, is not readable or is of insufficient quality. In the stories from researchers, it has happened that code from other researchers ended up not being used because of one of these reasons. Another problem is that researchers move from one project to another and do not consider finding reusable parts in their experiment and publishing them publicly.

One other suggestion is that MOOC code should be independent of the data. Researcher I2:

> *I mean, your code is an implementation of the algorithm, and your algorithm should not focus on any special data. The reason why I cannot reproduce the author's code is because the data format is quite awkward from my perspective. So, you should pay attention to the code and to the design of the algorithm at the same time.*

When code is shared between researchers, experiences are not always positive. Researcher I3, regarding their experience using code from another university:

> *Inaccurate and their code was not complete. And usually you need to spend a lot of time to figure it out. Like a week or two.*

Researcher I3 argues that documentation is a big problem when reusing code from others, as well as code quality. Researcher I1 also has negative experiences regarding documentation in the two occasions code was shared:

> *I think the documentation of them, both of them was not really good.*

3.2 Takeaways and conclusions

The envisioned workbench fulfills two purposes: Improving reproducibility for MOOC experiments and enabling code reuse among researchers.

For improving reproducibility, based on literature and the user interviews, the following takeaways and conclusion can be drawn:

3.2.1 How are researchers currently creating experiments (which methods, tools, approaches)

No standard workflow or approach exists for the creation of an experiment. The MOOC workbench should create such a standard workflow. This standard experimental workflow should ensure reproducibility of research.

3.2.2 Which actions are already being taken to ensure reproducibility of experiments

Not much steps are being taken to ensure reproducibility of the created experiments by researchers. As identified in the literature, reproducible computer science research means the use of some dependency management system that ensures code can be run in the future, a version control system so that the evolution of the code can be tracked and extensive documentation so it is clear what the code does.

As is clear from the interviews, researchers have no standard way of defining dependencies. Researchers do not write extensive documentation for their experiments. Documentation is limited to writing source code comments and the occasional README file. Also, researchers are not performing formal testing or verification of their experiments, do not use version control frequently and find source code sometimes to be difficult to understand.

The issue of reproducible research seems to go beyond tools. Researchers are sometimes aware of their lack of attention to reproducible research or of their bad habit with the use of these systems. Sometimes, a deadline is approaching and thus the focus is put on the end result instead of other aspects, such as reproducibility. This issue was also identified in the previous chapter, namely that researchers lack incentives to perform the work to improve reproducibility. In order to improve reproducibility, the workbench should help to shift the focus of the experimental process more towards reproducibility.

Thus, merely providing these tools for easier dependency management and others, might not be enough, as they can then still be ignored when a deadline is approaching. A workbench should make it visible how reproducible an experiment is to try and change this behavior.

3.2.3 How are researchers currently reproducing experiments and what problems do they experience when reproducing experiments

In general, experience with reproducing experiments by other researchers are limited. Issues encountered when an experiment is being reproduced are the lack of source code in publications and the lack of documentation and quality of the source code.

3.2.4 How are researchers currently sharing experiments and code, why they are sharing and the problems they experience with sharing

Code is shared via different mediums, such as via email, Dropbox, GitHub or via a personal project page of the author. The researchers are sharing code or using shared code because they need a common piece of functionality. Issues experienced with sharing are lack of documentation or inaccurate documentation or lack of code quality.

Researchers are focused on the end-result and then immediately move on to the next experiment. They do not consider or do not have the possibility, due to a lack of tools or lack of time, to review the code and find reusable parts, extract those parts and put them in a separate tool for everyone to use.

3.3 Requirements

Based on these takeaways and conclusions, the following requirements can be established.

The MOOC workbench:

- should give insight into how reproducible each experiment is;
- should give useful actions on how to improve reproducibility of an experiment;
- should be a MOOC tool marketplace, where researchers can add, find, use, run and install common MOOC tools and functions;
- should facilitate dependency management;
- should facilitate version control;
- should facilitate writing documentation throughout the experiment;
- should make it easier to test and verify source code;
- should enable researchers to easily share and publish experiments;
- should support the different phases of MOOC research;
- could facilitate the separation of the code from the data.

Chapter 4

Experimental workflow design

This chapter will study other relevant systems, such as scientific workflow systems, to give a proper basis for the conceptual design of the workbench. This chapter defines the key properties of a good design for experimental workflows for MOOC researchers and gives the proposal for the conceptual design of the experimental workflow that the MOOC workbench should support.

4.1 Scientific workflow systems

A scientific workflow system enables the composition and execution of complex analysis, often run on distributed resources [12]. These workflows often consist of directed graphs where nodes represent computation and the edges of the nodes represent the flow of data. Scientific workflow systems have the advantage of improving reproducibility, because a workflow created in such a system can easily be understood and re-executed by others. The main focus of scientific workflow systems seems to be the natural sciences, but this is expanding towards other domains. [24] gives examples of domains where scientific workflow systems are used, namely astronomy, biology and computational engineering, but also oceanography, seismology and neuroscience. Of the workflow systems identified in [2], most are built for natural sciences.

Many scientific workflow systems exist and the purpose of this section is to study these systems and understand how they work. Examples of scientific workflow systems are Kepler[11], Triana[12] and JS4Cloud[13].

Most scientific workflow systems provide abstractions for computational constructs and data by providing a graphical user interface or a domain specific language that allow researchers to create an experiment by dragging and dropping components. For example, Triana uses units and cables and Kepler actors in combination with directors [8].

I believe that to abstract away from the core competency of computer scientists seems counter-intuitive. These kind of graphical user interfaces in a workflow system seem to be mostly useful for domain-expert analysts and high-level users [26].

[11] https://kepler-project.org/
[12] http://www.trianacode.org
[13] [26]

As such, I believe the traditional scientific workflow system might not be applicable to this problem domain. Converting existing pieces of code to such a scientific workflow systems requires lots of effort by the researchers and I believe the kind of 'visual programming' offered by scientific workflow systems might be too limiting for computer scientists.

I believe the main problem right now though is that the domains for which most of these systems are created, are domains where computer scientists work to support this domain, for example bioinformatics or geoinformatics. The computer scientists can then work actively to maintain these systems and add modules, while other, less technical researchers, use the systems for their experiments. In MOOC research, this is not the case as computer scientists themselves are performing experiments, instead of providing support similar as is done in other domains where scientific workflow systems exist.

Only once a number of these modules have been created by researchers, I believe it starts to make sense to think about a workflow system. Instead, it makes more sense to first lay the groundwork for such a system that might be used in the future and be more suitable for non-computer scientists.

A system that might support this kind of groundwork is YesWorkflow, that allows adding useful annotations to visualize the workflow of experiments [27]. It allows researchers to annotate their source code by using simple commands to recover workflow information. These annotations represent program blocks, ports or channels. By adding these annotations, YesWorkflow can automatically generate a visualization of the workflow. Users can then click on nodes in the graph and see the corresponding code blocks. For future work, I believe this system might be useful for the workbench, but it should be studied better before a decision is made to integrate it.

4.2 Reproducibility enhancing systems

Reproducibility enhancing systems are not complete experimental workflows, but often are meant to be used after the creation of an experiment to improve the reproducibility of created experiments. It is useful to discuss these kinds of systems and see how they might fit into the experimental workflow of MOOC researchers.

4.2.1 RunMyCode

RunMyCode[14] is an international academic project that provides a computational infrastructure allowing authors to publish code and data related to their papers [44]. It was founded by economics and statistics professors and its main purpose is to improve research-reproducibility for the computational sciences. RunMyCode allows users to execute the uploaded code in the cloud. The uptake of RunMyCode is not very high. In November 2014, 82 papers have used the website to upload their code and data. Of those 82, none are from computer science departments.

The authors of [44] believe that research reproducibility can greatly be improved by opening data and code to the public. RunMyCode works by allowing the creators of the experiment to create a website, which serves as the public information page for

[14] http://www.runmycode.org/

the experiment. This process consists of six steps, namely providing information about the experiment and publication, describing the source code, such as input of the data and preview the page to make sure that everything is in order.

Researchers can then upload their code and data, if they wish. Others are then able to download the code and/or run it in the cloud. RunMyCode supports C++, Fortran, MatLab, R and RATS. After the code is added and some standard checks are performed, the results are automatically summarized in a PDF file and the website enters the validation stage.

In this stage, the authors and coders validate the created website. After that, an editorial team from RunMyCode checks whether the created website complies with the editorial policy of the website. What this editorial review exactly entails, is left unspecified. Finally, a technical validation of the code is done, focusing on robustness, security and CPU requirements.

This section considers the approach of RunMyCode because it tackles one aspect identified in both literature and the requirements analysis, namely that of making code and data more publicly available. In terms of experimental workflow, it has little to offer, because RunMyCode only requires the researchers to perform certain actions, such as uploading the code, after their experiment is complete. During the development and design of the experiment, it does not require any actions of the researcher.

4.2.2 SHARE

Another system comparable to RunMyCode is SHARE, described by Gorp et al. [45]. SHARE is a platform where authors can share virtual machines. These VMs contain all the code and data required to run the experiment. They even contain the original paper. Using a virtual machine to enable reproducibility is an interesting concept, because it solves many of the earlier described problems, such as dependency issues and system compatibility.

However, virtual machines also have some downsides, such as the limited life-span of a virtual machine, the use of a proprietary platform and security [37]. Also, virtual machines are not very suitable for large-scale use, because even small Linux virtual machines take up at least one gigabyte and cannot be run on every platform.

4.2.3 Open Science Framework

One of the researchers interviewed during the requirements analysis used the Open Science Framework (OSF)[15]. OSF is a project set up by the Center for Open Science and aims to enable and improve collaboration in science research.

The OSF allows researchers to upload data and code and maintain a wiki. For each file, version control is used and it is possible to review earlier versions. Researchers can divide projects into components. OSF describes components as 'sub-projects that help you organize different parts of your research' [16]. Each component has their own privacy settings, contributors, wiki, add-ons and files. OSF offers support for several add-ons, such as Amazon S3, GitHub, Dropbox and Google Drive.

[15] https://osf.io/

[16] http://help.osf.io/m/gettingstarted/l/481998-creating-components

OSF offers researchers the possibility to cite work on the platform by creating a unique ID. Each file, project and component already has a unique URL that can be used to cite the work. Researchers can also obtain a DOI, or a digital object identifier, which is an identifier that can be used by others to reference the work on OSF. Researchers are given insight into how often each file of the project is downloaded and how often the project's page is visited.

Other researchers are given the option to fork a project, which enables reuse [17].

4.2.4 Conclusion

The problem with RunMyCode, OSF and SHARE seem to be that they allow researchers to more easily publish their code and data, but they do not do anything to help improve the quality of the code and data, which is what seems to be required to improve reproducibility. RunMyCode addresses these problems more, though, than SHARE, because if the code is able to run in the cloud, then it can also be ran by other researchers, but it does not improve other aspects of the code, such as documentation or testing standards. Also, for MOOC research, code can often not be rerun by others, because of the lack of data, a problem that these systems do not address. Furthermore, both do not contain a marketplace for researchers to exchange common pieces of code. Researchers can only fork complete experiments.

SHARE, OSF and RunMyCode are not complete experimental workflows, but only work after the creation of the experiment. Instead, based on the challenges required for reproducible research, I think it is required to change the complete workflow of a researcher, something that these systems do not address.

4.3 Other tools and considerations

4.3.1 Literate programming

Literate programming is a form of programming where the natural language is the central point [19]. The documentation contains chunks of code and is written in such a way that it follows the 'human thought process rather than the usual flow of a program' [19]. Literate programming can benefit reproducibility, because of the fact that it follows human thought, as such these kinds of tools are very useful to use for researchers.

4.3.2 knitr

knitr [50] is an example of a literate programming tool. It is an R package that improves reproducible research as it allows researchers to write their scripts directly into LaTeX or Markdown. When the script is then run, either a HTML or PDF page is generated, complete with the script, results and graphical results embedded in the document as a whole. knitr can also format code to optimize readability. There are several libraries like knitr for R, but knitr has the advantage that it is easy to use for beginners because of the support for Markdown. LaTeX typically has a steep learning curve for beginners, which does not hold for Markdown [50].

[17] https://osf.io/4znzp/wiki/Measuring%20impact%20using%20the%20OSF/

4.3.3 Jupyter (IPython Notebook)

Perez et al. have looked into IPython, nowadays called Jupyter, a system for interactive scientific computing and present properties of a good interactive computing environment [35] for this purpose. The first property is access to all of the session state. IPython remembers the session state of all previous runs. Another property is a control system, which IPython provides and is designed to improve Python in an interactive shell. For example, using this control system it is possible to activate logging. Another property is access to the operating system, which allows for reading data and loading other programs, as well as dynamic introspection.

IPython refers to the shell and IPython Notebook refers to the web-based application that connects the IPython shell to a user-friendly front-end. The notebook was developed in 2011 and revolves around two ideas, namely an open protocol that controls the computational engine and an open format that records the interactions between user and the computational engine [28]. IPython started out as a project specifically for Python, but it evolved into Jupyter, that contains the language-agnostic aspects of the notebook system [28]. IPython Notebook is a system that 'provides an end-to-end environment for the creation of reproducible research'. It does so by capturing any computational session and adding version control, re-execution and conversion into other outputs. Useful is that notebooks can easily be shared, either in their native form or in HTML, LaTeX or PDF. Drawbacks of Jupyter are that extra work is required to import notebooks into generic Python files, since Jupyter Notebooks are not plain Python. Furthermore, using Jupyter with a version control system like git offers difficulties, since Jupyter files can contain binary data. The workbench should first support Python, since that is the main language the interviewed researchers use. Only then is it possible to support Jupyter, since Jupyter notebooks can contain Python source code. This makes it more suitable to look at in future work. Also, not all researchers might be inclined to use Jupyter instead of plain Python. In the interviews, only one researcher indicated using it.

4.4 Software engineering practices

The lack of proper software engineering practices in MOOC research might hurt reproducibility and reuse of experiments. Scientific researchers think that software engineering practices could increase their ability to develop quality software [18]. As a result, I believe it to be important that the workbench seeks to improve and enable these practices more among scientific developers.

4.4.1 Testing

The testing of MOOC experiments is limited to identifying if the program returns the correct results. This is not very different from how most scientific software is created [18]. Given reasons for this kind of testing are lack of management support and applications that are not large or complex enough to test [18]. Many studies agree that scientific software developers can benefit from testing.

As described by one of the researchers during the user interviewers, testing is often impossible, due to the lack of an oracle to verify the results against. Overcoming

this problem is difficult. In one survey, a total of twelve techniques are proposed to overcome oracle problems, such as using a pseudo oracle, using solutions obtained analytically, using results obtained experimentally and using simplified data to determine the correctness of the algorithm [21]. Other feasible options are using assertion based checking and automated clone checking. The latter is useful for when an issue is fixed in one place, but not in the other.

The workbench should provide testing tools and leave it up to the researcher to find the best way to overcome the oracle problem, if that is applicable to their research. Integrating testing in the experimental workflow is difficult, as everyone has their own preference when to write tests, either before or after writing code. I want the entry barrier of the workbench to be as low as possible, because if researchers use only a fraction of the tools instead of not using the workbench at all, then the former helps reproducibility more. For this reason, it is best to leave it up to the researcher to decide which testing approach they want to use and only provide some testing tools to get started. I made the choice not to integrate a specific testing phase in the experimental workflow, but instead the workbench will monitor the code coverage percentage of tests.

Enabling and encouraging researchers to write tests has two advantages: First, they test their software for faults, which helps improve the code quality and improve the accuracy of their results and second: To perform unit tests, a common form of testing, they have to find or create units to actually test. These are units of functionality and by creating these units, they might be creating reusable pieces of code that could be suitable for use by other researchers.

4.4.2 Documentation

Another issue identified during the user interviews and in literature is the lack of documentation in scientific research software. Some argue that documentation is needed to guarantee software quality [18]. The workbench should increase and improve the amount and quality of written documentation, by adding tools to easier write and generate documentation and by encouraging researchers to actually write the documentation.

Software documentation is an artifact with the purpose of communicating information about the software to which it belongs [51]. It comes in different forms, such as a manual for the use of the software, technical documentation for the design of the software or source code comments. In a literature review by Zhi et al. [51] several papers with regards to documentation of software are reviewed. In total, more than twenty-five papers introduce techniques for improving the quality of documentation, thirteen papers introduce tools and seven papers introduce models for documentation.

Overall, the survey of Zhi et al. finds that the most important attributes of high-quality documentation are completeness, consistency and accessibility. After those, other important factors with regards to documentation are the format and the up-to-dateness.

In order to help researchers to write documentation, a documentation generation system could be used [28]. A system specifically designed for Python is Sphinx [18].

[18] http://www.sphinx-doc.org/en/stable/

It does require some specific syntax since it uses reStructuredText, but the syntax is straightforward. Documentation generation systems work by adding documentation to the source code, for example the type of function and what it returns. Sphinx can automatically extract that documentation and format it. Sphinx allows to define a custom document structure, so it can contain all the documentation at once, including a README, installation instructions and code examples.

If the documentation is to be simple and if not thousands of lines of source code are written, it could be considered to just use Markdown to write documentation [28], in wiki style, à la GitHub.

Documentation should be used to document the purpose and design of the software, instead of the internal mechanics of it [48]. Also, Wilson et al. recommend to embed the documentation inside the source code, so that researchers are more likely to change the documentation when the code changes [48].

With regards to the experimental workflow, I believe it to be best that researchers continuously write documentation.

4.4.3 Reusability

One of the goals of the workbench is to increase reusability of code. In order to increase the reusability of code written by researchers, several approaches exist [23]. An important one is to create and maintain a central library of software that can be reused. Another one is to use code reviews, but that is quite difficult in this setting where researchers often work individually. Code reviews do seem to have signifcant benefits, as knowledge and good practices are spread throughout the department [48]. Wilson et al. recommend to use code reviews before they are merged.

Researchers should avoid copying and pasting code and instead modularize their code [48]. Copying and pasting code has the risk of increasing bugs and errors, because the same issue may have to be fixed multiple times throughout the project.

In order to facilitate modularization of code and thus reuse, the experimental workflow will consist of phases that try to separate the steps involved with MOOC research as much as possible, with the hopes that researchers separate their code naturally and implicitly create a workflow with different modules.

4.4.4 Version control

The use of a version control system has been widely accepted to be essential for good reproducibility. Up until now, what it exactly means to use version control properly has not been discussed. Visser et al. describe their best practices for version control. Specifically, they recommend to 'commit specifically and regularly' [47]. They also advise to link the commits in the issue tracker. It is also recommended to integrate code into the main branch regularly. These best practices are mostly only applicable to teams, however. The workbench could monitor the use of version control by researchers and offer suggestions regarding its use.

4.4.5 Infrastructure

The infrastructure of an experiment consists of running tests, building the code and generating the reports, which can save researchers a lot of time and improve produc-

tivity and enable collaboration [28]. For small projects, researchers often do not see the value of investing time and effort into learning and setting up an infrastructure to perform these tasks [28]. Some of these aspects are already discussed separately, such as testing and version control.

Continuous integration (CI) is an important aspect of infrastructure. This is only possible once the code is on a hosted version control repository and only useful if some tests are written. The CI system can then automatically run these tests after every commit. Setting up a CI can be quite time-consuming, although for example Travis CI in combination with GitHub can very quickly be set-up. The workbench should provide a CI system, which helps to continuously remind the researchers to test and to ensure the code is still working.

4.4.6 Metrics

All the problems and solutions discussed so far rely upon the researchers to use them and adhere to them. That is easier said then done in the real world. Only if researchers make the decision to adhere to these principles and improve their habits can reproducibility be improved [28]. To improve the development process of scientific software, the use of metrics should be considered for the workbench, as these are often used to improve the software development process [15]. These metrics could be used throughout the experiment and help researchers to make the right decisions with regards to reproducibility during the development process.

Researchers are rewarded for their experiments and results, not for the reproducibility of their work [28], although that is slowly changing[19]. To make an everlasting impact on the reproducibility of experiments, I believe it is important to give insight into the reproducibility of an experiment. These metrics could be used to set a standard of quality that is expected of an experiment with regards to reproducibility. However, drawing conclusions from metrics should be done cautiously, as they never tell the whole story, so they should be considered one part of the solution.

For most of the aspects discussed so far, a metric could be devised. For example, to measure effective use of version control, the workbench could measure the commit frequency. For testing, many metrics can be devised, such as number of tests or code coverage. For documentation, the coverage of source code could be measured.

Plösch et al. developed an automated quality defect detection program for documentation [9]. This program scans for many defects in documentation. It applies some of the following documentation quality rules: naming conventions, deeply nested sections, duplication detection, empty section detection, incomplete documents, long sentence detection, ultra large document detection, spelling errors, storage structure and more. The researchers test their program and conclude that for 75 percent of the documentation rules the system is suitable. A similar system could be used in the workbench to improve the quality of the documentation of MOOC experiments and tools.

Wingkvist et al. also developed a metric to try and measure the quality of software documentation [49]. Specifically, they use two metrics to determine the quality of documentation, namely documentation duplication and test coverage. The coverage

[19] http://cacm.acm.org/magazines/2016/10/207757-incentivizing-reproducibility/fulltext

of tests is measured by asking subjects to find information for a specific task. Such a test is successful if the subject finds the information in time. Researchers measure the amount of documentation needed to be read before the useful information can be found. Wingkvist et al. apply these metrics to two real-world tests and conclude that the metrics are useful, as they identified quality issues.

4.5 Key workflow properties

Based on all these systems, workflows and approaches, I am now able to define the key properties of the experimental workflow for researchers.

Lightweight

The important property of the workflow is that it is lightweight. While it is possible to simply build a complete scientific workflow system and let the users come, other systems have shown that that is not always the case. Based on the user interviews, researchers are currently writing code that is not very suitable for reuse by others, so in order to achieve that the workflow system actually contains useful modules, that has to be changed first.

The workflow system integrated in the MOOC workbench should be very simple. It should enable researchers to select the modules they want to use during their experiment and then install them in their version control repository. Then, the researchers are free to use the modules in a way that they see fit. This gives them a lot of freedom to construct their own scientific workflow. Once many modules are present, researchers without a computer science background can also easily run experiments.

The end-goal of the workbench is to be able to support experiments that only consist of modules created by other researchers, without ever having to write any code, but that won't happen anytime soon. This initial set-up gives researchers the freedom to add own logic and add functions that are missing. It is then the workbench's job to help them create new reusable pieces of code from their experiments.

Change behavior

Although all the problems listed can be solved technically, there are also social problems at play, namely ingrained habits and the pressure from the incentive models of modern research [28]. An important property is thus that the workbench try and address these social problems.

Promote literate programming

Another key property is that a good experimental workflow uses literate programming tools, such as Jupyter or knitr, which greatly increase the amount of documentation and understanding of code. The workbench should promote literate programming during those phases where it makes sense, for example in the data analysis phase.

Be flexible

Based on the user interviews, the proposed experimental workflow should be flexible and accommodate different ways of working. A researcher should be able to add, remove and re-order phases from the workflow, as they see fit. The workbench should only provide a default, suggested workflow.

Provide time for reflection and optimization

In order to increase reusability, it is important for researchers to take a moment and reflect upon the code of the previous phase and look for parts that might be useful for other researchers. Also, this phase could be used to take some time and optimize the code. Wilson et al. in their best practices for scientific computing advise to only optimize the software after it works correctly [48].

4.6 Design proposal

I believe that the MOOC workbench should support the following experimental workflow steps. By separating these phases of MOOC research, code is better separated and can more easily be modularized. The researcher is free to order the steps and remove or add extra steps of the same type as they see fit. The importance of these steps or phases is that code is separated and that it is clear what the purpose is of each step within the experiment.

1. Data gathering

The first phase is the data gathering phase. During this phase, the MOOC workbench will make the least assumptions, as this phase often differs from researcher to researcher. In this step, the researcher writes code to gather the data, for example by writing some MOOC specific code, and filter and clean the data, as often times some unnecessary or erroneous values will be present in the data.

2. Data schema

During the data schema phase, the researcher creates a data schema from their gathered data. The MOOC workbench should provide additional tools and libraries to make this step as easy as possible. I believe such a data schema can improve reproducibility significantly. Oftentimes, data is not present, with a data schema other researchers can verify if their data is compatible with the experiment at hand.

3. Feature extraction

During this step, a researcher writes code to extract data features from the dataset.

4. Data analysis

During the data analysis phase, researchers create for example models from the extracted data features. Ideally, the workbench recommends and integrates a literate programming tool.

5. Publication phase

In the publication phase, the researcher has the option to publish the experiment in the workbench, so that other researchers can access the experiment in the workbench.

4.6.1 During each step

Many of the discussed software engineering practices are implicitly part of this experimental workflow. These kinds of tasks should be done continuously during each phase. It is then not useful to formally add these to the experimental workflow by reserving a specific phase for them.

4.6.2 Post-step tasks

After each phase has been completed by the researcher, the workbench compiles a report of the work done in that step and prompts the researcher to review the source code and find code that could be extracted and changed into a module. Also, the workbench should provide an overview of the metrics of the past step and offer suggestions for improvement, so that researchers know how to improve their work.

4.6.3 Example experiment

To show how the workbench can help with reproducibility for a MOOC experiment, this section presents a hypothetical example experiment that is created and managed with the workbench. As an example, say that I want to create a MOOC experiment to design an intervention before a student drops out, with my hypothesis that students are likely to drop-out when their answers have negative language in them. For this purpose, I create an experiment in the workbench named Drop-out detection. The workbench initializes the experiment. I then choose my experimental steps: In this case all the steps, and I proceed to set-up my experiment locally.

1. Data gathering

Once I have set-up my experiment locally, which means I have set-up an environment of tools that support reproducibility, as well as my source files, I can get to work. I start with gathering data. For this, I need to create and gather a dataset. I write a script to gather data in some MOOC in Coursera and test it. Luckily, with the static code analysis tool from the workbench, I can easily see that my code contained an incorrect condition and fix it. I also find a package written by my colleague that already did some other Drop-out experiment in Coursera and uses the relevant APIs, so I install that package in my experiment and use some of the functions in that package to easily gather the data I need. Afterwards, I let that code run for one MOOC period and I have a CSV file for my data.

With the data, it turns out some values and rows are erroneous, so with the *make_dataset* script I load the input data file, clean it and calculate some extra values that I need for the next step.

2. Data schema

Now that I have the data file that I can use, I create a data schema. Using the file provided in the boilerplate code, with a single command I have a representation of the used data schema and I can commit this to my repository.

3. Feature extraction

Next, I can move on to the feature extraction step. Using some machine learning magic, I extract from the dataset the features that I want to use and find important. Using the provided boilerplate code, I see that testing is very easy and I write a number of tests to test the feature extraction functions. It turns out I also want to use the NLTK toolkit for Python, so in the dependency tab of my experiment I add this as a package, together with the version I use in my experiment.

At the same time I am doing my experiment, it turns out my colleague also wants to do another experiment on Drop-out detection. Luckily, since I have written documentation in my source code, the workbench has published my documentation already on GitHub, so the colleagues can see if my code is relevant and useful. It turns out it is, so once I am done with my code for feature extraction, I create a package of it with just a few clicks for the other researcher to use.

4. Data analysis

Next, I move on to the analysis step. In this step, I model the extracted features and try to see if my hypothesis holds. In this step, I find a relevant package recommended by the workbench that I can use. When I complete this step, I see on the score card of my experiment in the workbench that a lot of functions and classes are not covered by my documentation. So I return to the source code and review the functions and classes and add the relevant documentation, such as input parameters, return values and what the purpose is of the functions. I then review the new score card and the workbench now shows all green, so I can continue.

5. Publication phase

I then complete my experiment by visualizing the results and by publishing it in the workbench. Publishing it in the workbench means that a unique URL and page is generated for my experiment, that is accessible for all researchers. The page shows all the experiment steps used in the experiment, together with the dependencies and the data schema. It also contains a link to the zipped source code on GitHub, so that others can easily download my code.

Reproducibility

Suppose that another researcher wants to reproduce my experiment. This is much simpler than if the workbench was not used. Since I have published my experiment in the workbench, another researcher can easily download and access the correct version of my source code, review the documentation, see which packages I have used and see my data schema. Because of the boilerplate code, my source code organization is predictable and another researcher does not have to spend long hours looking which

file does what, since the file organization is well-documented and chances are they use the same boilerplate code as me.

Furthermore, even though I cannot send the other researcher my data directly, because of data protection concerns, they can use the data schema to verify if their data is valid for my experiment. Using the available tooling, researchers can easily verify if their data adheres to my data schema and can know how to change their own data set. Also, since I have defined all the dependencies of my experiment, they have no trouble running my experiment. With a single command, the researcher reproducing my work also reproduces the environment in which I did this work, with the same dependencies present and with a single command the researcher verifies if everything is set-up correctly, by running all the tests present in the experiment, so they can get to work quickly.

Chapter 5

Implementation

This chapter describes the implementation phase of the MOOC workbench, how the requirements were translated into workbench features and the workbench system architecture. Also, this chapter answers the research question: 'How can the designed workflow be implemented with modern day web engineering practices?'.

5.1 Web engineering

Modern web engineering approaches are often agile, meaning they work in short sprints, value working code over documentation and strive to add value to the product. However, since this project is done individually, properly applying an agile method is difficult, since they are mainly meant for teams. To overcome this, the implementation uses a modified/lightweight version of Scrum, with only the parts that make sense in a single-person project, namely maintaining a project backlog, working in sprints and planning and reviewing for the next sprints.

5.1.1 Design goals

The workbench is meant to augment the experiments of researchers with useful tools and boilerplate code. One of the design goals of the workbench is that the technology used to accomplish this, should be open technology already in use today, to prevent reinventing the wheel with a subpar solution and to prevent lock-in into the workbench. The latter is especially important: Researchers should not be tied to the workbench. They should be able to move effortlessly to a workbench-less environment if they wish and still be able to use all services that were provided by the workbench, albeit manually.

Another important consideration is longevity. The used technologies in the workbench should be supported for at least a number of years to make sure that it does not require a lot of maintenance and rewrites.

Furthermore, the key properties from section 4.5 should be taken into consideration, namely: Lightweight, change behavior, promote literate programming, be flexible and provide time for reflection and optimization, while at the same time accomplishing the requirements from section 3.3.

5.2 Architecture

To show the architecture of the workbench, [40] advises for complex systems to divide the architectural descriptions into the following views: Context view, functional view, information view, concurrency view, development view, deployment view and operational view.

5.2.1 Context View

The context view is useful for understanding the relationships, dependencies and interactions of the workbench and the environment in which it operates [40].

External entities

External entities are important to understand the context in which the workbench operates. These can be systems, organizations or persons with which the workbench operates [40]. I have identified the following external entities:

Researchers The main users of the workbench are the researchers. Based on the user interviews with the researchers, they have a varying level of technical knowledge. Researchers interact with the workbench to perform their data science experiments, create and find packages from experiments, find and share package resources and publish their experiment.

GitHub Another important external entity that the workbench interacts with is GitHub[20]. GitHub is a service for hosted git repositories. The workbench interacts with GitHub to create new repositories for researchers and to push, commit and view files. To interact with GitHub, the workbench makes use of the GitHub REST API v3[21].

Travis CI Travis CI[22] is the continuous integration system with which the workbench interacts. This CI system is used for the experiments of researchers, to automatically start builds when a researchers commits code and run the tests. The workbench can retrieve the build status and the last build log. For interaction, the Travis CI API is used.

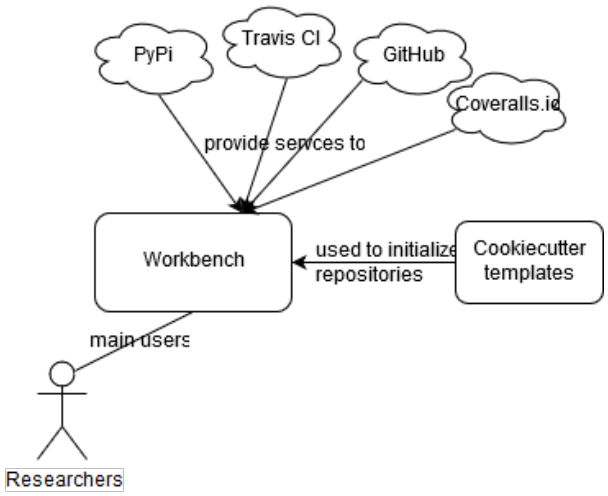
Coveralls.io Coveralls[23] is used to give insight into the code coverage measurements of experiments. The workbench interacts with Coveralls via HTTP post and get requests and retrieves the code coverage percentages. Coveralls is dependent upon Travis CI, as Travis performs the actual code coverage measurements, since that is where the tests are run, and submits the report to Coveralls.

PyPi The workbench interacts with PyPi[24] to retrieve the latest version of external packages. PyPi is responsible for providing version updates to the workbench.

Cookiecutter template creators The workbench is dependent upon Cookiecutter templates that are hosted externally in GitHub and are used for initialization of experiments. Anyone can create these templates, but only workbench administrators can add

[20] https://github.com
[21] https://developer.github.com/v3/
[22] https://travis-ci.org
[23] https://coveralls.io
[24] https://pypi.python.org/pypi

Figure 5.1: Informal context view diagram showing the external entities of the workbench. Researchers are the main users of the workbench, Cookiecutter templates are used for experiment initialization and PyPi, Travis CI, GitHub and Coveralls are external services used by the workbench.



them to the workbench. If the workbench cannot access these templates, for example if they are made private or GitHub is down, researchers will be unable to create new experiments with those templates.

The workbench does not have any external interfaces. See figure 5.1 for a diagram of the context view.

5.2.2 Functional View

The architecture of the workbench has several constraints due to the used framework. For the workbench, I chose to use the Django framework. Django is a structured Python web framework with a built-in Object Relational Mapping system that allows for creating a database through Python. Another advantage is that I am already familiar with Django and Python. Django has a strict system for dividing code. A Django website is divided into a number of apps. Each app provides a single set of cohesive functionality that should be usable on its own. Specifically, a Django app consists of a description of the database models, the view functions, long-running tasks using the Python asynchronous distributed task queue Celery[25], util and helper functions, forms, urls and mixins, if applicable.

Next, the functionality of the most important apps will be discussed. The other Django apps can be found in appendix D.2.1. To read more about the used technologies in the workbench, see appendix D.1.

Cookiecutter manager

The Cookiecutter manager manages the Cookiecutter templates and functionality to provide the researchers with boilerplate code. Using this app, it is possible to define a

[25] http://www.celeryproject.org/

Cookiecutter template. The library Cookiecutter[26] is used to create project templates and initialize them with pre-defined variables.

The templates can either be used for an experiment or a package. The workbench contains by default four Cookiecutter templates: one for a Python experiment [27] (forked), one for an R experiment [28] (new), one for a Python package [29] (forked) and one for an R package [30] (forked). These templates are hosted in GitHub repositories and some of them are based on existing templates.

For example, the Python data science template used in the workbench is forked from the Driven Data data science template[31]. This template was specifically designed for data science. The steps earlier presented in this thesis, with regards to data gathering, data filtering and more, matched quite well with the Driven Data data science template. To prevent confusion, the naming was synchronized between the steps in the workbench and the steps in the data science template by renaming the steps in the workbench.

The forked version of the Python data science template was expanded with the schema step, the step in which researchers create their data schema. Other changes included small changes to the Makefile with the addition of documentation and configuration changes in the documentation generation, namely to include the autodocument and coverage extensions.

This set-up of using Cookiecutter templates allows for easy extension in the future with other templates. Suppose that the department becomes unhappy with the current data science templates. They can then very easily set-up a new GitHub repository, push a new template and add this into the workbench with very minimal effort. To add a template to the workbench, the workbench needs to know the location of the documentation folder and the experiment steps have to be mapped to the locations in the new template, both very easy and trivial tasks possible in the admin section of the workbench. They then supply the GitHub URL of the new Cookiecutter template and are done. Then, researchers can select this new template during the creation of their experiment.

Dataschema manager

The Dataschema manager is responsible for the JSON table schemas used in experiments. The functionality of this app is to manage the data schemas created by researchers.

JsonTableSchema is one of the few libraries that seem to satisfy the requirements of the workbench. *JsonTableSchema* is a standard for describing a CSV table in the JSON format[32]. This is a standard created by Frictionless data, which is supported by Google and the Alfred P. Sloan Foundation. The advantage of using *JsonTableSchema* is that a single standard is used for specifically tabular data, with extensive tooling available for

[26] https://cookiecutter.readthedocs.io/en/latest/
[27] https://github.com/MOOCworkbench/cookiecutter-data-science
[28] https://github.com/MOOCworkbench/cookiecutter-r-datascience
[29] https://github.com/MOOCworkbench/cookiecutter-pipproject
[30] https://github.com/MOOCworkbench/cookiecutter-r-package
[31] https://drivendata.github.io/cookiecutter-data-science/
[32] http://frictionlessdata.io/guides/json-table-schema/

a large number of programming language and even tools, such as Python, JavaScript and pandas. The disadvantage is that the tooling provided for the *JsonTableSchema* is in the early development stages and that the standard is still very new. During this sprint, the tooling still was in the alpha phase of development and does not currently support an R version, but this is planned for the future. Nevertheless, I have found it usable and found the advantages outweigh the current disadvantages, especially since the development of these tools proceeds very quickly.

The Dataschema manager app uses a third-party library called tableschema-py that translates and interprets pushed JSON schemas by researchers. If a researcher pushes a JsonTableSchema in their experiment, this manager parses that file and converts it to a user-friendly front-end view in the workbench. In this view, researchers can easily edit their data schema and make sure that it is correct. Changes made to the data schema are then converted to a new schema and committed and pushed to the GitHub repository of the experiment.

Docs manager

The Docs manager is used for documentation management for experiments and packages. The main feature of the Docs manager is to automatically generate the Sphinx documentation for Python projects and to publish this documentation to GitHub. Normally, publishing documentation to GitHub is a labor-intensive task, which requires the set up of a separate empty branch, build the documentation, move the generated HTML files to this branch and commit the changes. The workbench automates this work and requires zero effort on the user side to publish the documentation. Sphinx, a Python tool to generate documentation, contains a coverage module that automatically checks for functions and classes not documented yet. The workbench can read the results of this coverage check and uses the results in the quality manager, for example suggest the user to write more documentation.

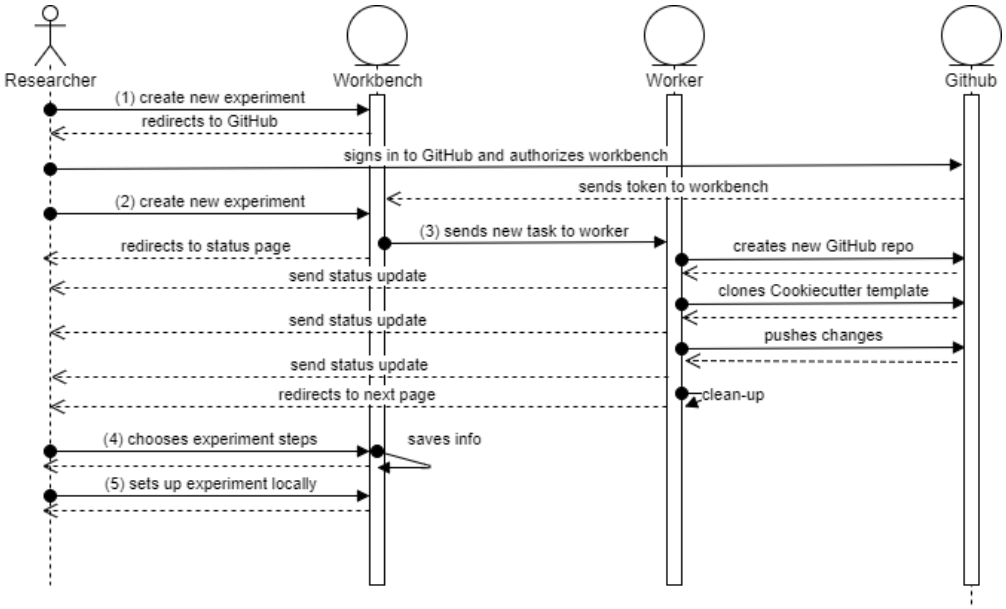
Experiments manager

The experiments manager is the most important Django app of the workbench and contains the views and tasks for experiment management, such as experiment creation.

Experiment creation steps See the state diagram 5.2 for a visual overview. When a researcher creates an experiment (1 in the figure), they first connect with GitHub. Once that process is complete and the workbench is authorized to create experiments, they arrive at the experiment creation page and can enter an experiment title and a project description (2). They can also choose which data science template they want to use for their experiment. By default they can choose between a Python and an R data science template. Once they have chosen the template, the experiment is created by a worker (3). This means a new GitHub repository is initialized with the boilerplate code for their chosen language. During this process status updates are sent to the researcher. After experiment creation, the workbench redirects them to the page where they can select the experiment steps (4) and once complete to the get started page (5), with steps on how to get to work on their experiment.

Experiment detail index On the experiment detail page, researchers can track the progress of their experiment. On this page, they see the experiment steps they have

Figure 5.2: State diagram of creating an experiment



selected. Each step maps to a folder in the experiment. For example, the data step is located in the folder src/data in the experiment. The same holds for the features, modelling and visualization steps. In the file list on the experiment detail page, for each file they see the static code analysis results and the percentage of code covered.

Git manager

The Git manager is responsible for handling git operations within experiments, for example cloning a repository and pulling and pushing changes when many files are changed. It uses the library GitPython to help with common git functions. The GitManager also contains the LanguageHelper classes. These classes enable support for multiple programming languages within experiments, by providing an abstraction for common experiment functions, such as installing a dependency, running static code analysis and generating documentation. The LanguageHelper implements these functions for Python and R. Each experiment has their own LanguageHelper, containing a set of default functions defined by the interface, so that the other apps do not have to worry about language-specific functionality. For more on how multiple programming languages are supported, see section D.2.1.

Marketplace

The Marketplace app in Django helps with package sharing and package creation. The workbench has two types of packages: External and internal packages.

External packages are meant as information resources for the researchers, for example *pandas* or *numpy*. The point of adding such a package to the workbench is that the workbench automatically tracks new versions of the package from PyPi and that researchers can share information about the package in the workbench. Also,

researchers can share Package resources with each other. On the detail page of an external package, they can create a new resource. This is a useful information sharing tool. For example, they can use Package resources to share information on how to use a package, how to use some new functions of a package and more.

Internal packages are meant for code to be shared with other researchers. They can be created as a stand-alone, empty package, where-in a researcher can add whatever they want or they can be created in the context of an experiment step. In the context of an experiment step, when the researcher has completed a step, the workbench gives them the option to create a package. If they choose to do this, the workbench automatically extracts and filters the code from this step and creates a new package from it. This means the user has to do very little work with regards to boilerplate and actual knowledge of creating a package.

The git history of the extracted files of the experiment step are kept and transferred to the package repository, which is useful for the original author, as they can still go back to earlier versions of the code, as well as for others to try and understand the reasoning behind changes. After creating a package, the researcher is expected to perform a number of tasks before they publish the package. Only after publishing the package is it visible for others and can it be installed. The tasks to be performed consist of writing a guide on getting started with the package, ensure that all the dependencies are defined and by making sure the package works or optimize the code. See figure 5.5 for a walkthrough of publishing a package.

For R, publishing is not required. R has the ability to install packages in the right format directly from GitHub, so a special server is not necessary. The workbench generates a special DESCRIPTION file which is required before a package can be installed.

On the package detail page, the researcher has the ability to add a new package version. After they have made some changes, they are expected to create a new version of their package. This entails entering a new version number and a changelog. Once they save their new package version, the workbench for both Python and R automatically updates the required files. Upon creating a new version, the workbench creates a GitHub release and a git tag, so that the new version is not only published within the workbench, but also on their GitHub repository, which prevents workbench lock-in.

Quality manager

The Quality manager is responsible for the dashboard, which shows the status of various aspects in the experiment.

The workbench performs the following measurements. Each measurements implements an interface of three methods: init, measure and save_and_get_results. The second one is to perform the actual measurement and the third one is to save the result to the database.

CI Measurement This measurement checks to see if a Continuous Integration tool is used for the experiment and if researchers pay at least some attention to the build results. This measurement checks if Travis is enabled for the experiment and if so, it retrieves the last build status and saves it. If at least in the last five measurements a build succeeded, the workbench considers this as the researcher paying attention to

CI and it returns a High result. If they have all failed, we make the assumption the user hasn't really paid attention to the builds and it is time to do that, so we return the Medium result to notify the researcher of this. Else, if Travis is disabled, we return Low.

Docs Measurement The Docs measurement checks to see if automatic documentation generation is enabled and checks how many functions and classes are covered. First, the workbench checks if documentation is enabled. If that is the case, a coverage check is performed. This requires the re-generation of the documentation with the coverage check. The SphinxHelper takes care of the underlying details and returns, if successful, the amount of uncovered functions and classes. A High result is returned if both are 0, so if there are no undocumented functions and classes. If there are less then 10, the measurement returns Medium, else Low. The workbench also returns Low if automatic documentation generation is disabled. The Docs Measurement is only applicable to Python, not to R, as that does not have a suitable automatic documentation generation tool in the workbench.

Pylint Measurement The goal of the Pylint measurement is to ensure that researchers write bug-free code that follows coding conventions. The Pylint measurement works by reviewing the results from the latest static code analysis. Only if a scan is successfully completed, this measurement returns a result. If errors are found during static code analysis, a Low result is returned by this measurement. If there are warnings, but no errors, a Medium result is returned. If only other issues exist, less than 30, a High result is returned.

Requirements Measurement The goal of the Requirements measurement is to ensure that researchers have defined all their dependencies for their experiment, which ensures that other researchers can easily run their experiments. To determine whether the researcher has defined all the requirements, the workbench uses the Continuous Integration tool by retrieving and scanning the last log. If the log contains an *ImportError* or a *ModuleNotFoundError*, this most likely means that some package is used that is not defined in the dependency list. If this is the case, this measurement returns a Low result, else High. This measurement works for both Python and R.

Test Measurement The goal of the Test measurement is to ensure that researchers achieve high code coverage for their experiments. For the Test Measurement, the workbench retrieves the code coverage measurement from Coveralls. If the code coverage is lower than 50 percent, this measurement returns a Low result. If it is lower than 80 percent, a Medium result is given, else a High result is returned. If code coverage is disabled in its entirety, a Low result is returned.

VCS Measurement The goal of the Version Control System measurement is to improve the amount of commits of code by a researcher using a version control system. In this case, the workbench uses GitHub webhooks to store commits of the researcher. This measurement retrieves the commits for the researcher. If a researcher commits twice a day for the last three days, this measurement return a High result. If a researcher commits less than twice a day, this measurement returns a Medium result.

5.2.3 Information View

The information view describes "how the system stores, manipulates, manages and distributes information" [40].

Information storage

The information the workbench stores is contained in a relational database, specifically SQLite. SQLite fits the requirements of the workbench and has the advantage of being zero-configuration, severless and is contained in a single file. It should be noted that the Git manager sometimes stores a GitHub token on the file system. This is required so the workbench can perform push and pull operations for a large amount of files and folders. This token is stored in the *.git/* folder in the remote-url variable of the experiment at hand. System actions are written to the log files. Which log files the workbench uses and which log levels are written to these files can be reviewed in the file *MOOCworkbench/settings.py*.

Information management

To manage and manipulate the information present in the workbench, the workbench uses an ORM system. ORM is an Object Relational Mapping, as the name implies it maps objects to relational databases. The workbench uses the default Django ORM. No manual SQL is written to prevent errors and security issues. Models are stored in each Django app in the file *models.py*. When a model is changed, the Django ORM is used to generate SQL and execute the SQL statements to reflect these model changes in the database.

Information quality

To maintain the quality and validity of the information present in the database, transactions are used when multiple objects are dependent upon each other, whenever possible. Also, during experiment creation, if an experiment cannot be initialized, the entire operation is rolled-back. The experiment database object is deleted. By overriding the experiment model delete function, the workbench also removes automatically all the attached models.

5.2.4 Concurrency View

The workbench contains some concurrency. For some tasks, the workbench needs a copy of the experiment git repository present on the local file system. This is required for tasks such as generating documentation and performing static code analysis. It is important that these tasks do not interfere with each other. This concurrency is removed in its entirety wherever possible, by performing tasks sequentially. For example, the tasks that run when a git push via the webhook is received, are performed sequentially, which does have the downside of taking longer. However, that does not completely exclude concurrency, since the researcher is given the option to manually refresh the data or since the edge case that multiple webhooks might be delivered at once for the same repository. Researchers could start that task at the same time as the webhook. To prevent concurrency issues completely, the component responsible for providing the local copy of the git repository provides each task with their own local copy. This means that multiple copies of the same git repository can exist on the file system. To make this possible, a random key is generated that is used as the folder name in which the git repository is cloned. The component requesting the local

copy receives the file path to that git repo, with the random folder name. This prevents interference between tasks.

5.2.5 Development View

The Development view is concerned with the architecture of the software development process, such as testing, the tools used and module and codeline organization [40].

With regards to codeline organization, the Django defaults are used and followed. For how a Django project is built, see the Django docs[33].

For testing, the Mock framework is used where appropriate. Sometimes, access is needed to GitHub and other APIs which require a valid token. It is possible to either locally place the token in an *.env* file in the workbench directory or use the build server to run the tests. Travis CI contains an environment variable GITHUB_TOKEN with a valid GitHub token that can be used for testing interactions with the GitHub and other APIs.

For day to day development, GitHub is used. Large features or refactoring work happens on its own branch. In general, it holds that the master branch always contains a shipping version of the workbench. Other tools that are used during development are Travis CI to provide Continuous integration functionality, since running the entire test suites takes about five minutes, Coveralls for code coverage measurements and requires.io to monitor how up to date the project dependencies are and review if security issues exist in the used dependencies. Code coverage is currently at 81 percent and future commits are expected not to reduce this percentage, as advised as a best practice by [17].

5.2.6 Deployment View

The deployment view is concerned with the environment in which the system will run and what is required to run the system [40].

Runtime platform

The workbench can be deployed directly on a operating system, in which case the workbench is recommended to run on Ubuntu 16.04 LTS with Python 3.5/3.6. It is also possible to use Docker for deployment. The workbench repository contains a docker-compose and Dockerfile to automatically configure the Docker instances. Using Docker is the recommended way to run the workbench. For specific deployment instructions, see the workbench manual[34].

5.2.7 Operational View

The operational view is concerned with how the system will operate, administered and supported while running in production [40].

At least one user is expected to be the administrator of the system. In the administration section of the workbench, reachable via */admin*, the following changes can

[33] https://docs.djangoproject.com/en/1.11/intro/tutorial01/#creating-a-project
[34] https://moocworkbench.github.io/MOOCworkbench/deployment.html

be made: Change the experiment steps (name, description, add and remove steps), add, change and remove Cookiecutter templates, manage default texts of the workbench and modify the text of the quality checks. Please note that the title of the quality checks should not be modified.

Upgrading the workbench is simple and consists of replacing the Docker containers with the new version and migrating the database.

For back-ups, it is necessary to back-up the file *db.sqlite3* regularly. That is the only file that needs to be backed-up, the rest of in the workbench folder can be ignored.

During its use, the workbench logs to several log files in its directory. These can be used to check the status of the workbench and see what is going wrong.

5.3 Functionality

This section outlines the main features of the workbench, why these features were implemented the way they were implemented and the design choices made.

5.3.1 Experiment management

Researchers can create and manage their MOOC experiments in the workbench.

The experiment detail page is where researchers track their progress of their experiments. Work in an experiment always takes place within the context of an experiment step. The experiment detail page has a section for all the steps, with the active step being expanded and showing information for that step.

Adding this extra context is useful for a number of reasons. First, the workbench can show more accurate information. For example, each experiment step is tied to a folder in the boilerplate code. In the experiment step, the workbench shows the files for that step, which are the files the researcher is currently working on. Also, the workbench links directly to the correct documentation written by the researcher in that step, so they can track what they have written so far. The workbench then only scans those files for static code analysis and documentation, which makes it more efficient and accurate. Furthermore, the workbench can recommend specific useful packages from the packages section in the workbench per experiment step.

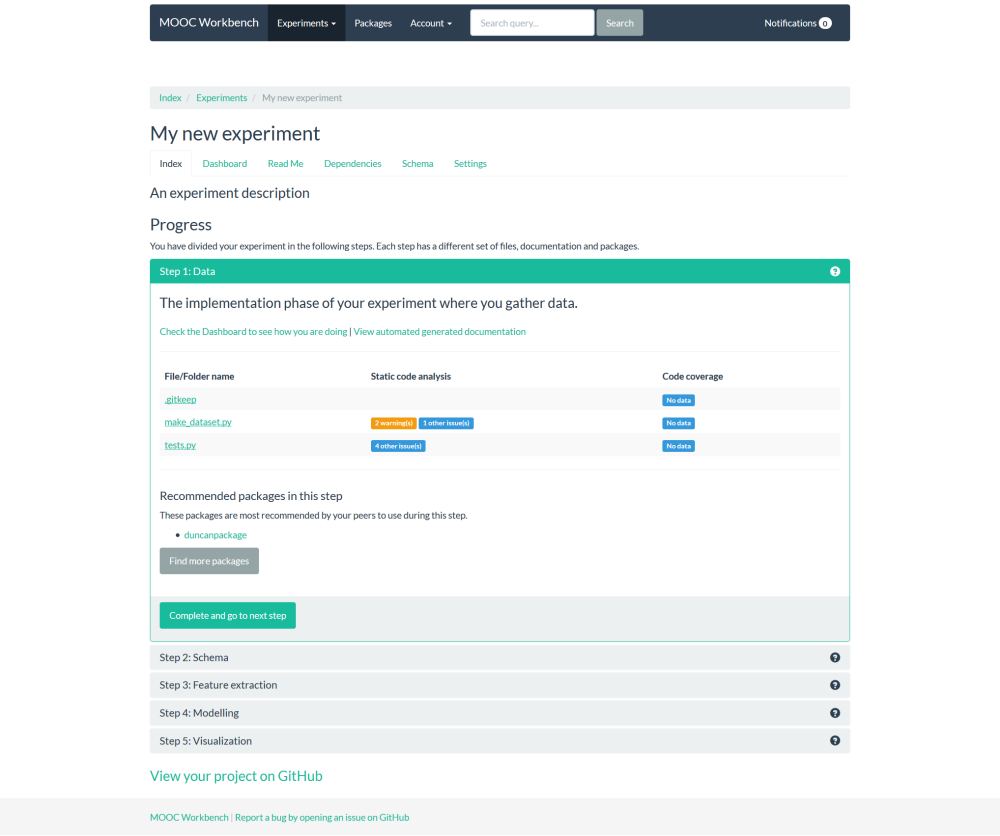
Researchers can click on a file in the experiment detail page and view the contents of that file. If a static code analysis scan was completed, they immediately see the results of that scan embedded in this file.

Once a researcher has completed a step, which means they have written all the code necessary to implement that step of their experiment, they click the button Complete and go to next step. The workbench shows a report score card for the aspects Documentation, Testing, Continuous Integration, Code coverage, Static code analysis and Dependencies. When researchers click Continue, the workbench shows the option to create a package from the code in this experiment step, to share with other researchers.

The experiment detail page consists of several tabs with specific functionality that will be outlined now.

Dashboard On the Dashboard, researchers can review the results of the aspects that the workbench checks for their experiment and take action to improve, if necessary. On the Dashboard, researchers can see the following aspects:

Figure 5.3: Experiment detail page

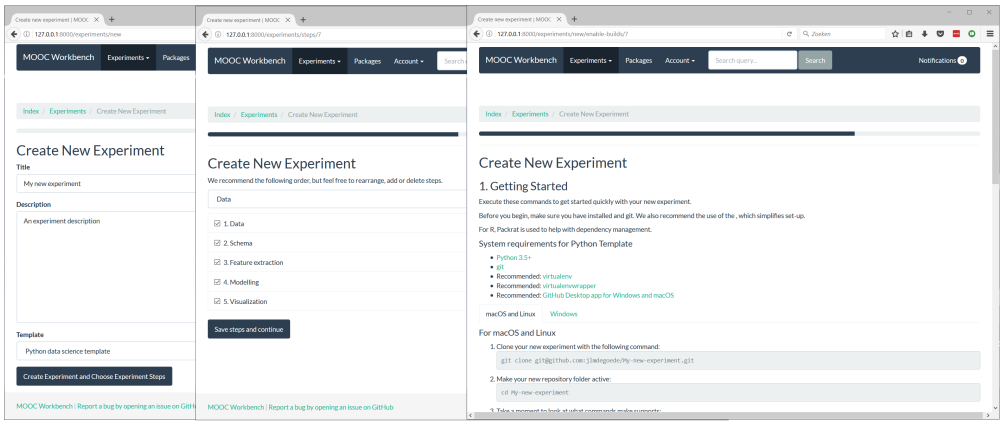


- **Static code analysis** The results from the most recent static code analysis scan of their code.
- **Dependencies** Whether or not all the dependencies are defined within their experiment.
- **Documentation** Track the amount of uncovered functions and classes in their code and the status of automatic documentation generation.
- **Version control** Track the amount they have committed their code.
- **Continuous integration** See the status of the continuous integration tool, for example if builds are passing and the last log of a build.
- **Code coverage** The status of the code coverage tool and an indication of the amount of code covered by tests.

Dependencies On the Dependencies tab, researchers can manage the dependencies of their experiment. Here they can define other external and internal packages that they used in their experiment and change existing dependencies. Upon saving the changes, the workbench starts a task to write the changes to GitHub.

Schema On the Schema tab, researchers can manage the data schema for their experiment. The data schema describes the schema of the data used in the experiment, so

Figure 5.4: Creating an experiment in the workbench (shortened)



for each column in the data several properties are defined, such as the data type, data format, a description of the data and more. Researchers can create their schema via two ways: They can either define it manually here in the workbench or they can use the boilerplate code to generate a data schema from their data. If they commit that data schema, the workbench parses the file and shows it in the front-end in an editable format. If a researcher adds manually all the fields, the workbench automatically converts the schema and commits it to their GitHub repository.

5.3.2 Boilerplate code

The boilerplate code serves multiple purposes and sits at the core of the workbench. The boilerplate code:

- sets a standard for writing and reading data with a default directory and read/write methods;
- defines an entry-point for the code;
- creates a test scaffolding;
- creates documentation scaffolding;
- defines a way for defining dependencies;
- creates config files for CI and code coverage systems.

By default, four Cookiecutter templates are present. One for Python data science, one for R data science, one for a Python package and one for an R package.

Python data science template The main template is the Python data science template. For this template, the workbench builds upon existing work. The Driven data data science template was forked and modified for this purpose.

The following boilerplate code was already present in the template and are considered advantages: **src/data** The *src/data* folder contains a file *make_dataset.py* which

uses a library named *Click*[35]. *Click* is useful because it is possible to easily add command-line arguments to Python functions. For example, for the *make_dataset* function in this file two command-line arguments have to be set: *input_filepath* and *output_filepath*. This means you can call this script as follows: *python3 make_dataschema.py input_filepath=/path/to/input/data output_filepath=/output/file* instead of hardcoding the used files in the code directly.

Makefile The Cookiecutter data science template contains a Makefile to automate common tasks, such as setting up an experiment environment, which entails creating a virtual environment and installing the dependencies, running the tests and more.

Default sane folders The boilerplate code contains default sane folders, such as a *src/* folder, but also folders for output such as *data/* and *models/*.

The following additions were made to the existing data science template: **Test scaffolding** To improve reproducibility of experiments, researchers need to test. As such, the workbench adds test scaffolding to the data science template. By default, five tests are provided that call the existing functions and assert whether they return True. Using these tests, researchers can see how easy it is to write a simple test. I also mended the Makefile to include a test_run command so that researchers can easily run all the tests.

CI and Coveralls configuration The new data science template provides zero-configuration Travis and Coveralls set-up. This means each repository created by the workbench is ready to be used by Travis. For Travis, the build configuration automatically runs the tests and submits them to Coveralls for code coverage results.

Documentation improvements While the original template already contained a *docs/* folder and a Sphinx configuration, the workbench improves this configuration through autodoc, which takes Python docstrings that researchers have written in the source files and automatically places them in neatly created HTML files. The workbench then publishes those HTML files automatically on GitHub. One issue arose with *Click*, as *Click* uses decorators in Python, which Sphinx does not handle properly. Luckily, the library *sphinx-click* solves this problem.

Data schema The workbench also adds a data schema and data schema tooling. Researchers can run a Python script in the same manner like the *make_dataset* file, providing the file path to their dataset, after which the *make_schema* file automatically creates the JSON table schema file. There is one limitation with this tooling. It is in the early stages. During development this tooling still was in the alpha phase. One of the issues that arose, was that submitting large files for creating a data schema takes an extremely long time. Clear instructions are added that instruct researchers to slice their data sets to maximum of 100 lines.

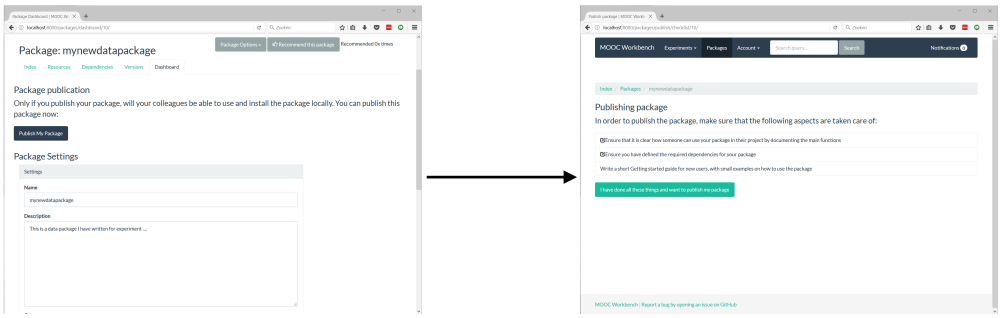
Packages

The Packages section in the workbench allows researchers to both create packages from their own code as well as add external packages.

Creating a new package For creating a package, I want researchers to consider at each experiment step sharing code with others, so it should be integrated into the workflow and feel natural to sharing code.

[35] http://click.pocoo.org/5/

Figure 5.5: Walkthrough of publishing a package



To accomplish this, researchers are invited to share their code each time they complete an experiment step. Once this process is complete, researchers are redirected to the Internal package dashboard. Here they can change aspects of their package, such as a description and the title, and enable some workbench services for their packages, such as automatic documentation generation and continuous integration.

Package detail page Just like the experiment detail page, a detail page exists for a package. This page consists of the following tabs and elements.

Index The index of the package detail page shows the description of the package, some package information and the README file of the GitHub repository, which offers a starting point for this package.

Resources The resources tab is used for viewing Package resources. A Package resource is a piece of information about a package. For example, it can be a guide on getting started with the package, a URL to a useful blog post about the package and more. Package resources can be added by anyone and are written using Markdown.

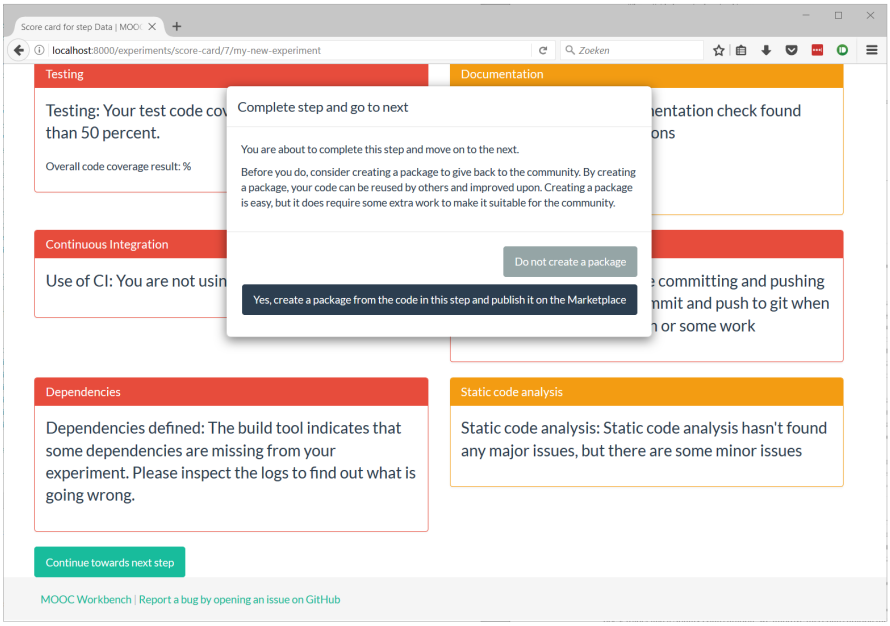
Versions On the versions tab, researchers can view the version history of this package and the changelog for each version. For the owner of the internal package, they can publish a new version on this page, by entering a new version number and the changelog.

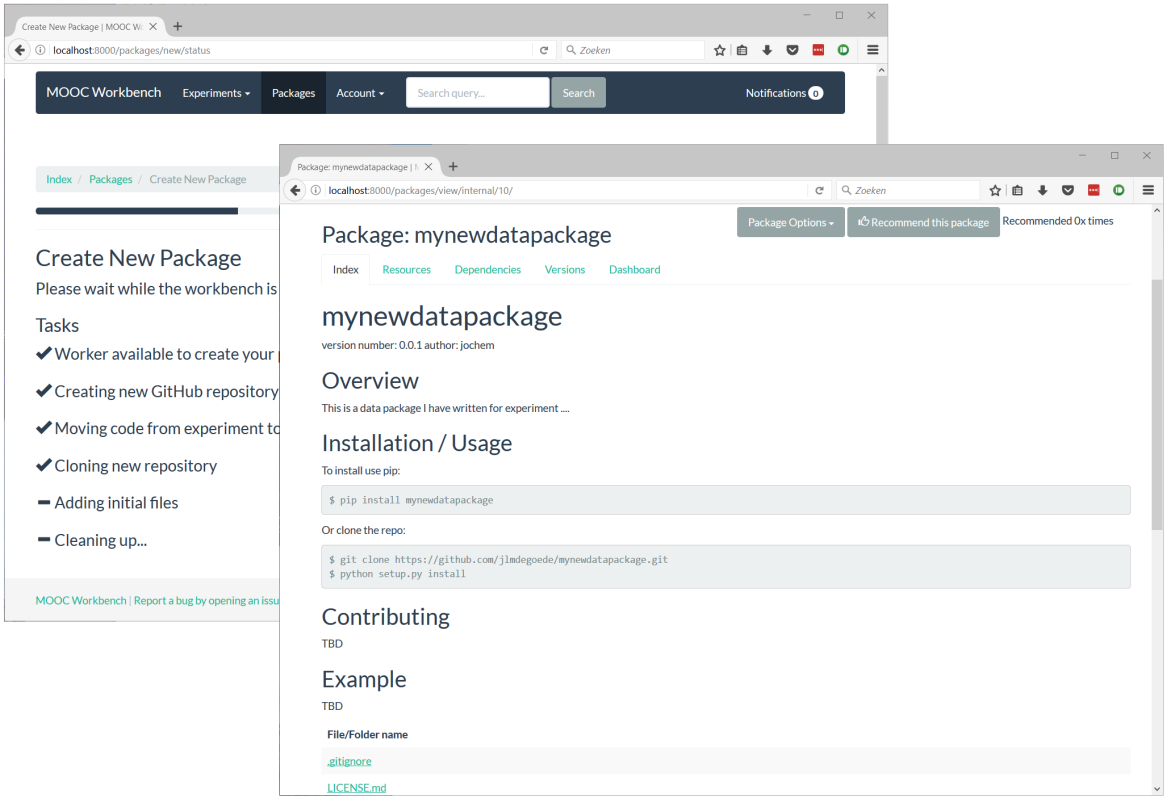
Dependencies On the dependencies tab, researchers can add the packages that their package depends on.

Dashboard On the dashboard tab, researchers can manage their package settings and publish their package.

Publishing a package Before a package is visible on the workbench and it can be installed by others, researchers have to publish their experiment package. They can do this on the Dashboard by clicking Publish My Package. See figure 5.5 for the steps in publishing a package. The workbench then offers a final checklist. Before publishing a package, researchers are encouraged to write a getting started guide as a Package resource, to make sure that the package works and to define all the dependencies of their package. Once they click Publish, other researchers can visit the Package detail page and install the package from the PyPi server, if it is a Python package.

Figure 5.6: Walkthrough of creating a new package





Finding and installing packages

Researchers can find and install packages on the workbench in the Packages section. On the index page, they see new internal and external packages recently published. If they want to install a package in their own experiment, researchers visit a package detail page and click Package Options / Install package. Researchers can choose from a drop-down which experiment they want to install the package to. Once selected, the workbench adds this package as a dependency to their project and starts a task to update the GitHub repository with the changes. They then pull the git changes and update their experiment environment so that the package is installed.

Suggesting packages

On the experiment detail page, for each step researchers can view recommended packages. These recommendations are provided by researchers that have recommended that package for that category. They can recommend a package on the package detail page by clicking the Recommend button.

5.4 Requirements validation

This section outlines how the requirements were translated into functionality of the workbench and why these functionalities meet the requirements set. Each subsection is one of the requirements defined in Chapter 3.

5.4.1 should give insight into how reproducible each experiment is

- Experiment divided into steps, the steps of research this thesis earlier defined.
- Researchers can use the experiment page to track the progress with regards to these steps.
- After completing a step, they are presented with a score card of how they did during that step.

5.4.2 should give useful actions on how to improve reproducibility of an experiment

- During their work, researchers can visit the Dashboard tab and see how they are doing with regards to testing, committing and writing documentation.
- The dashboard gives actions and messages on how to improve these aspects.

5.4.3 should be a MOOC tool marketplace

- The workbench contains a Package section where researchers can add packages.
- They can add external packages that they've commonly use in their experiments.
- The workbench tracks PyPi versions of a package.
- When the researchers complete a step and move on to the next, before they move on they are invited to contribute that code in the form of an internal package.

5.4.4 should facilitate dependency management

- The workbench facilities dependency management through a web interface where researchers can enter which packages they have used and which version.

5.4.5 should facilitate version control

- The workbench facilitates version control through a graph on the Dashboard page and through the action messages delivered by the workbench.

5.4.6 should facilitate writing documentation throughout the experiment

- The workbench automatically enables document generation for Python projects.
- The workbench tracks documentation coverage and notifies the researcher through action messages or the report score card if they have undocumented classes or functions.

5.4.7 should make it easier to test and verify source code

- The workbench provides boilerplate code for testing.

5.4.8 should enable researchers to easily share and publish experiments

- The workbench contains the packages functionality and allows for publication of an experiment in the workbench.

5.4.9 should support the different phases of MOOC research

- An experiment is divided into a number of steps, defined in an earlier chapter.

5.4.10 could facilitate the separation of the code from the data

- The workbench uses *JSONTableSchema* for researchers to easily define their data schema used in the experiments.

5.5 Conclusion

This chapter has shown how the requirement from the previous chapter, together with the key properties and experiment workflow, were translated into a functional workbench. This chapter has given an architectural description using the various views suitable for such a description and described the main functionalities of the workbench. See appendix D.4 for a discussion on how the agile development process took place with the iterations and evolutions involved.

The research question to be answered in this chapter is: 'How can the designed workflow be implemented with modern day web engineering practices?'. This chapter answers this question by showing these modern day web engineering practices: A version control system was used, together with iterative programming, a continuous

integration tool for continuous insight into code quality, a modern web framework with reusing a lot of existing tools, together with a set-up that separates long-running tasks from the web server in their own worker, making the workbench scalable and user-friendly. The result of this chapter is the implemented workbench.

Chapter 6

Evaluation

This chapter will evaluate the design and implementation of the MOOC workbench. This chapter answers the research question 'Having implemented the design, does the implemented solution achieve its requirements defined by RQ1?'.

6.1 User testing

To validate if the workbench meets the requirements set in the earlier chapters, a user test is performed. The three PhD researchers working on large-scale learning analytics from the user interviews in Chapter 3 together with one visiting researcher from Estonia, use the system to perform a real-world data science task.

6.1.1 Goals

The goal of this user test is to determine the extent to which the goals and requirements from the earlier chapter are met. I consider the goals and requirements of the workbench to be met when researchers can accomplish their own research goals using the workbench, while using the functionalities outlined in the previous section, without too much interference or change in their workflow, and when the artefacts they created show that they have used these services. For example, are they writing tests, defining data schemas and defining dependencies. I also use feedback given by the researchers to find out what they like and do not like about the system.

6.1.2 Experimental design

The participants in the user test performed a data science task from Kaggle.com. Kaggle is a well-known, large and popular site where data science challenges are posted. It features a wide variety of real-world data science problems, each with a data set and an explanation of possible research goals. Since the goal of the evaluation is to simulate real-world data science tasks in this user test, Kaggle is very suitable.

Beforehand, two different data science tasks were selected. I have selected two data science tasks so that the tasks can be evenly divided among the group. During the first session, two researchers work on one dataset and two on the other. They are asked to create a package with their code. During the second session, the two data sets are

swapped so that the code sharing feature of the workbench can be tested. This allows for evaluating the effectiveness with and without sharing code.

These two data science tasks were saved and compressed in a ZIP file, containing the dataset and the assignment in txt format. Specifically, the following two data science tasks were chosen:

- Task A: Crowdedness at the Campus Gym[36] with the goal to predict how crowded the gym will be in the future and figure out the important features that determine crowdedness.
- Task B: IMDB 5000 Movie Data set[37] with the goal to predict the greatness of a movie without relying on critics and to find if a correlation exists between the number of human faces in the movie poster with the movie rating

I have selected these data sets on their perceived difficulty. The sessions are relatively short so I needed a data set that can be understood easily and one that is not too complicated, as well as selecting datasets and goals that are similar, for example two classification problems, as to improve the chance that code can be shared among researchers. See figure 6.1 for an overview of the experimental design of this user test.

Extracting the data from the session was done using two ways: At the end of each session, a debriefing group interview was taken and I gathered specific logs and data from the workbench. Each session was scheduled for two hours from 13:00 until 15:00 hours. The next two sections describe in detail how the two sessions progressed and what happened during them, such as how the sessions were set-up exactly, which activities were performed and the goals for each session.

First session

For the first session, the four researchers were in the same room together with the writer of this thesis.

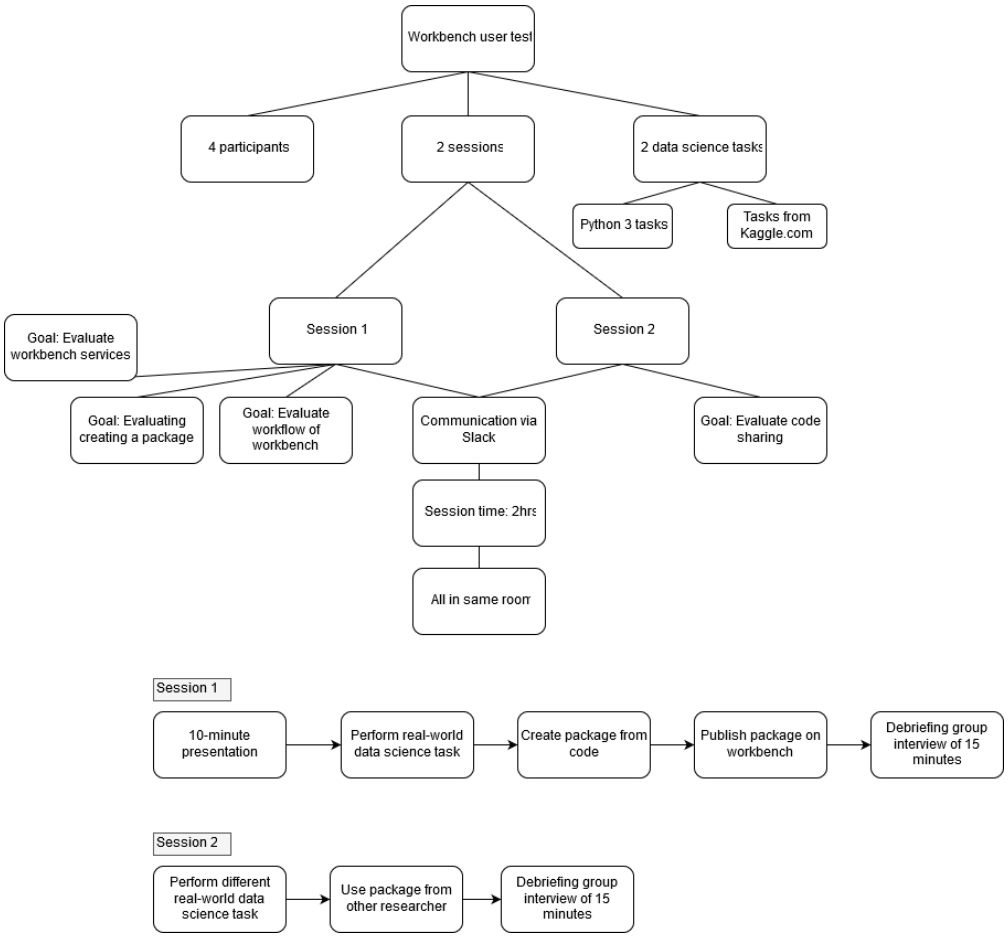
Initially, a short presentation was given, which can be reviewed in appendix B. This presentation consists of 11 slides and took less than 10 minutes. The presentation explained a number of things: It explains what the workbench is, so that the researchers know what to expect when they use the workbench.

It also explains the goals that of the evaluation session today, namely them performing a real-world data science task using the workbench with the purpose of evaluating the workbench and testing the code sharing feature of the workbench. For code sharing, the researchers are asked to create at least one package from one of their data science task experiment steps. For example, create a package from the feature extraction part of their experiment, or the modelling part of their experiment, and publish this in the workbench.

Finally, the presentation contains a workbench introduction, introducing the experiment steps, the detail page and the dashboard, which are the most important aspects of the workbench that also might be the most overwhelming. The introduction was intentionally kept short without introducing all the features, because the workbench

[36] https://www.kaggle.com/nsrose7224/crowdedness-at-the-campus-gym
[37] https://www.kaggle.com/deepmatrix/imdb-5000-movie-dataset

Figure 6.1: An overview of the experimental design of this user test.



already has a tour of its features and I wanted to know how usable the workbench is to new users. Explaining all the features in detail influences these results.

After this short presentation, the researchers were sent their data science task via Slack Appendix with a README fileC. Two researchers received task A and two received task B. They then created an account on the workbench and started using it like they would in the real world, namely creating a new experiment, using the boilerplate code to work on their experiment and use the features of the workbench.

Because of the small setting in which this test took place, all the researchers were asked to use Python for their experiments, even though the system also supports R. This ensures that for the second session code can be shared among researchers.

Second session

The goal of the second session is to test the code sharing feature of the workbench, using it within their experiment and accessing package documentation. The two data science tasks are swapped between researchers and they are asked to complete this new task. The main difference between this session and the previous one is that this

Figure 6.2: A timeline overview for the first session.

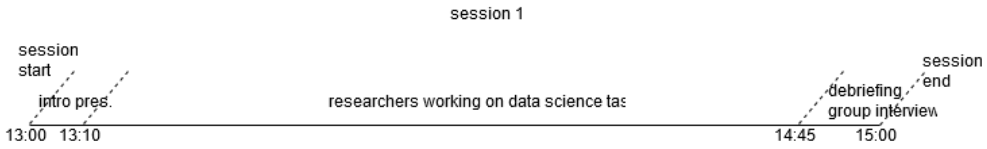
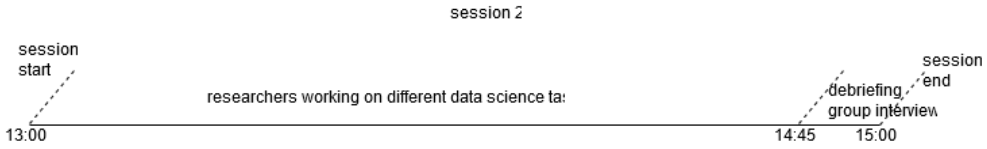


Figure 6.3: A timeline overview for the second session



time they are asked to use at least one package from another researcher in their own experiment. This way the researchers experience using code from someone else within the workbench and can evaluate that part effectively. Also, this set-up with two tasks ensures that researchers can compare whether code sharing helped them achieve their data science task.

For the second session, the set-up is the same as the first session, except now no presentation is given.

During the experiment

During the experiment, the researchers were asked not to talk to each other about the experiment. Instead, a dedicated communications channel was set up on Slack[38] on which the researchers could ask questions regarding the workbench. I have made this decision so that an accurate log exists of the reaction of the researchers when they first use the workbench. This way it is immediately clear what they did not understand and with which aspects they struggled. It eliminates the need to make notes during the sessions, notes that could be incomplete or inaccurate.

In the results section of this chapter, these chat logs are analyzed, to show what questions they had and which aspects of the workbench they discussed and how. The chat logs and their timestamps will be compared with the timestamped workbench logs, so that I can see which actions lead to which questions.

Evaluation

The workbench is evaluated using semi-structured interviews with the researchers, using the log files and using the artifacts created by the researchers.

Interview After all the researchers of the user test have finished their data science experiment and published at least one package, a short group interview is performed in the room with the researchers. The purpose of this interview is to determine how disruptive they have found the workbench to their normal workflow, to understand if

[38] https://moocworkbenchusertest.slack.com

and how they use the services provided by the workbench and to understand whether they find the code sharing feature of the workbench an improvement over how they normally share code.

This interview consists of the following questions:

- What were your experiences getting started with the workbench?
- During your work, did you review the actions given on the Dashboard in the workbench, and if so, what did you think of these actions? (if unclear: Actions regarding CI integration, VCS use, documentation and static code analysis)
- Did you make some changes in your experiments based on these actions? If so, what did you change?
- session 1 only: What were your experiences when creating a package from your own experiment?
- session 2 only: What were your experiences when using a package in your experiment from another researcher? (Ask if not mentioned: Were all the dependencies present, was it clear how to use and install the package?)
- Did you use the Packages section in the workbench for finding resources or other external packages?
- What were your experiences when using the Packages section in the workbench?
- Did you use any of the provided services of the workbench, such as defining dependencies and a data schema, and if so, how did you use them?
- What were your experiences with these services provided by the workbench?
- What were your experiences with publishing an experiment in the workbench?
- Compared to how you normally work, how disruptive do you find the workbench to your workflow?
- What aspect or aspects of the workbench did you find most disruptive to your workflow?
- Do you have any other comments or feedback on the workbench?

The same questions will be asked in both sessions, apart from the indicated questions.

Logs and database During the experiment, user logs are captured within the system, outlining the system state and the user actions. Afterwards, the database used in this experiment is copied and analyzed locally, so that I can determine to what extent the features in the workbench were used. The logs are used to determine the following:

- How much time did the researchers spend on each experiment step?
- Did the time spent on each step vary between researchers (with the same dataset)?
- Did the time spent on each step change when researchers used a package?
- How long did it take for researchers after joining to git commit for the first time?
- How much time did the researchers spend on the dashboard?
- How often did they check the Dashboard with their experiment status?

- When were they most likely to visit the dashboard?
- How long did it take from package creation to researchers publishing it?
- How long did it take from installing a package to committing code using that package in their experiment?
- How long did it take for researchers after arriving at the schema step to committing a data schema?
- Did they use the workbench to define dependencies?
- Did they use the workbench to check and/or modify their data schema?
- Did they complete all the steps of their experiment, as they defined for their experiment?

Artefacts I also take a look at the final artefacts produced by the researchers, in the form of their GitHub repository and code, of both their experiments and their produced packages, to see how they used the new boilerplate code and if, and to what extent, they adopted new practices introduced by the workbench. Some questions that will be answered here are for example: Did the researchers use the boilerplate code as intended? and: How much changes did researchers make in their code and comments between creating and publishing a package?

These three methods combined should give a complete picture to be able to determine if the requirements of the system were met or not.

6.1.3 Results

The first user test took place with four participants. For this test, version 1.0 rc1 [39] was used of the workbench, in conjunction with version 1.0 of the boilerplate code template[40] for the data science part and the version 1.0 of the boilerplate code template for a Python pip package [41]. For an overview of how the user test was executed, see figure 6.4.

First session

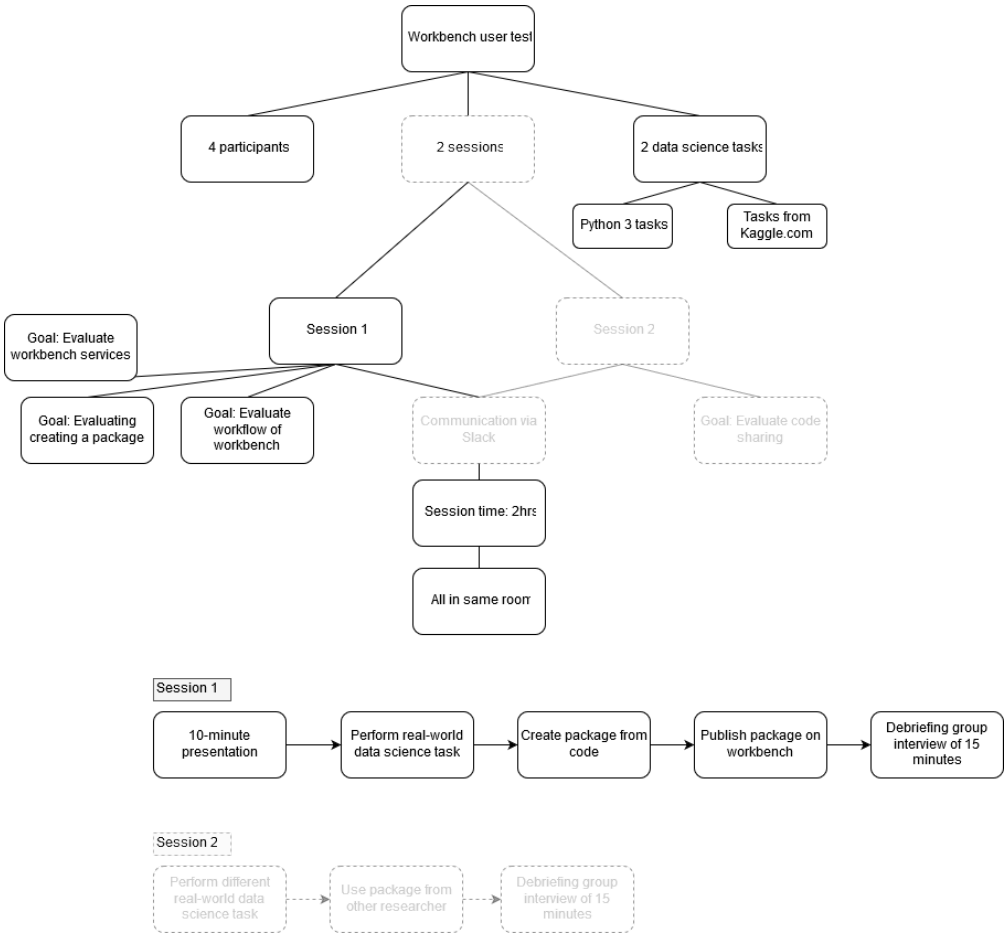
The first session started at 13:00 hours with all the participants in the same room.

Even though the participants were asked to ask questions using the Slack communications channel, this did not happen. The Slack channel is empty. This is likely due to the nature of the set-up: They are all together in one room and this made it much simpler to ask questions directly rather then type them. What might have also played a role are the number of problems experienced while working.

All participants were not able to use the workbench to its full potential and had trouble getting started. Two were unable to complete the data science task. Each researcher opted to use all the experiment steps available in their experiment. This means their research consists of the following experiment steps: Data, Schema, Feature, Modeling, Visualization. Researcher A ignored the steps and created a Jupyter

[39] https://github.com/MOOCworkbench/MOOCworkbench/releases/tag/1.0-rc1
[40] https://github.com/MOOCworkbench/cookiecutter-data-science/releases/tag/1.0
[41] https://github.com/MOOCworkbench/cookiecutter-pipproject/releases/tag/1.0

Figure 6.4: The execution of the user test.



notebook performing the data task completely at once. Afterwards they placed the Python file in the correct experiment step folder of the boilerplate code. This user did create a package. Researcher B performed the data science task, but did not complete the task. They did go through all the experiment steps and created a package from the Feature extraction step. Researcher C was unable to use the boilerplate code, due to a technical issue, and did not start the data science task or commit any code. As a result, they skipped the Data and Schema experiment step and started working on the Feature step. Researcher D made it until the Schema step and did commit a data schema. They tried to start the data science task, but never committed any code, due to being unfamiliar with Python.

As a result of what happened in the first session, I decided to cancel the second session with code sharing, as they would be unable to do that and no extra information or feedback would have been gathered. As a result, the code sharing part of the workbench was not evaluated.

Getting started with the workbench was difficult, as it requires some preliminary software that was not present on the computers of the participants. Python 3 and git were not present on most systems, for example. Once they installed and set-up

this software, which took for researchers B and D 10 minutes and for C and A 30 minutes, there was a lack of understanding of the workbench. The presentation did not mention this aspect, as the workbench itself contains the necessary commands to set-up a suitable environment. Unfortunately, the researchers had a false start, as due to a bug they were not automatically redirected to the GitHub sign-in page [42]. Once they signed up, they were able to create an experiment.

Most issues during the evaluation arose due to system differences and software set-up issues. There were some issues getting Python packages like *virtualenv* en *dotenv* working on the laptops of researchers, which impeded their ability set-up the experiment environment. Eventually, three out of four got it working. These issues were resolved within 20-30 minutes.

The main issue that arose during working on the experiment was that the workbench introduced too many new concepts for the researchers and they did not know where to begin. The new concepts for them are the boilerplate code, specifically its structure, as well as virtual environments, continuous integration, static code analysis, defining dependencies and defining a data schema.

The first session ended at 15:00 hours.

Debriefing interview

The debriefing interview took place at the end of the first session. It was performed with all the four researchers present as a group interview. The total interview time was 15 minutes. This interview was recorded.

Some of the questions were skipped due to the developments and difficulties of the user test.

What were your experiences getting started with the workbench? The researchers had issues getting started with their data science task using the workbench. Researcher A notes:

> *Yeah, getting started was difficult. It took a long time.*

They found the steps to get started on their experiment difficult and found the experiment steps vague. Researcher B says:

> *I think more specific instructions would be helpful. (...) I have no idea what the database schema is used for. So for some steps, like, briefly state the purpose behind this step, which would be helpful I think.*

There were some technical issues, either in the workbench or on their own laptop. Researcher D:

> *I literally lost track because I didn't have anything necessary available on my laptop. So, for me, it just failed.*

During your work, did you review the actions given on the Dashboard in the workbench, and if so, what did you think of these actions? (if unclear: Actions regarding CI integration, VCS use, documentation and static code analysis) Two researchers did not check the dashboard and had no feedback on the actions given

[42] https://github.com/MOOCworkbench/MOOCworkbench/issues/79

by the dashboard. Researcher A did review the dashboard and they found that the dashboard provided too much information. Researcher C also checked the dashboard:

> *I noticed a lot of messages, but I did not got the point of these.*

One researcher suggested that the dashboard should be visible at all times and that they did not find it at the beginning of their experiment.

Did you make some changes in your experiments based on these actions? If so, what did you change? The researchers did not make any changes based on the actions given in the dashboard.

What were your experiences when creating a package from your own experiment? Researcher B encountered a technical issue during the creation of a package for the first time [43]. The second time it did work correctly.

Did you use any of the provided services of the workbench, such as defining dependencies and a data schema, and if so, how did you use them? The researchers used the provided services by the workbench only very limited, most likely because they were focused on getting their experiment to work, so no answer was provided here. Researcher B did like the way a data schema was created:

> *That part is good I think. It is clear.*

What were your experiences with these services provided by the workbench? As the researchers used the provided service only very limited, they were unable to provide feedback with regard to their experiences for the services.

What were your experiences with publishing an experiment in the workbench? The researchers did not have enough time to publish an experiment.

Compared to how you normally work, how disruptive do you find the workbench to your workflow? The researchers found the steps overall to fit to their experiment workflow in general, but two participants found that during each experiment step, they did not really know what to do and found insufficient explanation of what was expected of them during each step. Researcher C notes:

> *I do not think there is enough explanation to each step.*

Do you have any other comments or feedback on the workbench? Researcher A notes that:

> *It was a lot of... we had to do a lot of stuff I did not understand, just because we have to, and that is the best thing I can do to describe it.*

Also:

> *...maybe I haven't done a project big enough, to need something like this, but I don't understand why you can't just use GitHub for all this. Like, what does this offer what GitHub doesn't?.*

Researcher A also did not like that the workbench asked for access to their private GitHub repositories.

Another point of feedback was that one researcher had encountered a bug with creating a data schema, where it multiplied the fields[44] and they asked for the option

[43] https://github.com/MOOCworkbench/MOOCworkbench/issues/81

[44] https://github.com/MOOCworkbench/MOOCworkbench/issues/80

to re-order the fields. The introduction tour of the main features of the workbench that researchers receive when they first sign-up, was skipped by at least one researcher. Researcher A found that it contained too many details and clicked through without reading all the information provided:

> *That [the initial tour, auth.] was so, that was really like, detailed and I don't know, there's no context, I was just pressing Next.. next.. next to get to the interface.*

Results from Logs & database

The following are the results from the logs, the database and the created artefacts of the researchers.

How much time did the researchers spend on each experiment step? After the first session, it was clear that the experiment steps for their data science task might be too broad to be applicable to this test. For example, they did not have to gather data. Some did create a data schema and then they went to the Feature or the Modelling step and did all the work at once. They did not do the final Visualisation step to visualise their data.

Time it took for researchers after joining to git commit for the first time Only researcher C did not commit any code or a data schema. Researcher A committed code after 58 minutes. Researcher B committed their code in a little over 21 minutes. Researcher C never made a git commit. Researcher D took 1 hour and 15 minutes before a git commit was made.

Amount of time the researchers spent on the dashboard Researcher B never visited the dashboard. Researcher A, C and D spent between 1 minute and 1 minute and 45 seconds on the dashboard. On average, researchers spent 67 seconds on the dashboard.

Amount of times the researchers checked the Dashboard with their experiment status They did not check the dashboard often. Researcher A did this three times, researcher C and D one time and the researcher B never.

When they were most likely to visit the dashboard Researchers visited the dashboard either after just creating an experiment, looking around. No other pattern emerges.

Amount of time it took from package creation to researchers publishing it While the researchers did not publish a package, two researchers were able to create a package. After completing their step, they were able to create the package and split their code. They did not publish their packages.

Defined dependencies The researchers did not require any extra dependencies beyond the ones already present in the boilerplate code, so they did not use the workbench to define any dependencies.

Checked and/or modified their data schema Two researchers used the workbench to add their data schema and add this to their experiment.

Completed all the steps of their experiment, as they defined for their experiment set-up The researchers did not complete all the steps of their experiment as defined.

6.1.4 Improvements

The workbench itself could benefit from a number of improvements, based on feedback provided during the workbench:

Show what is optional

The researchers had problems getting started. These problems can be avoided, as they are optional. The problematic steps are setting up a virtual environment, running the initial tests and testing if the environment is set-up correctly. In order to ease the set-up process, I have added some text is to show that these steps are optional[45].

Unclear what to do during each step

Researchers provided feedback that often they did not know what to do during each step and that the explanation in each step is not enough. Currently, in each experiment step a one sentence explanation is provided of what the purpose is of this step. Clearly, this sentence is not enough, as it does not describe accurately what it is the researcher is expected to do. The text for the steps Schema and Modelling could be improved. This text can easily be changed in the future, as it is an editable text in the administration section of the workbench and is meant to be optimized and changed by the workbench administrators. While not all descriptions were as clear as they could be, some were. Why researchers then did not know what to do in most of the steps, is unknown. Possible explanations can be that they were too overwhelmed and did not get the grander picture or that they overlooked the text. To improve the situation overall, a link in the step to the user documentation of this step is added[46] so that the researchers can take a step back and review the documentation.

Lightweight template

The boilerplate templates from Cookiecutter can take some time getting used to. For example, the one used in the user test contains eighteen folders total. They do, however, offer several advantages. First, when everyone has the same experiment set-up, this leads to more consistency and means that other researchers can easily understand and use experiments created with this template. Also, the boilerplate code contains test scaffolding to get started with testing, a README file, a dependencies file and documentation. For more on the Cookiecutter templates and all their advantages, please review section 5.3.2. For researchers, it was difficult to use the provided boilerplate code, most likely because they did not have time to review all the code and what the purpose was of each part. This is understandable and might be solved through more education. These templates are specifically designed by Driven Data for data science, but it is a lot to learn. Sometimes, researchers want to perform a shorter experiment without a heavy template. For this purpose, a lightweight template could be added with the bare minimum to get started: just the step folders and some configuration files, as opposed to the large amount of folders present in the current templates. This reduces the number of packages that have to be installed and lessens the burden on the

[45] https://github.com/MOOCworkbench/MOOCworkbench/issues/82
[46] https://github.com/MOOCworkbench/MOOCworkbench/issues/83

researcher, but it also means that researchers have more freedom in their experiment set-up, which means less consistency between experiments and less reproducibility of the experiment overall.

Dashboard not checked

During the experiment, the dashboard was not checked often. One researcher suggested to always show the dashboard. Always showing the dashboard does not seem like a good idea, as then researchers no longer see their experiment progress. Furthermore, on the experiment progress page, researchers can already view some important information coming from the dashboard, namely the static code analysis results and the code coverage percentage. In a future version, more work could be performed to design a page that combines both more information from the dashboard together with the experiment progress page. In this version, this is not feasible.

Dashboard provides too much information

One point of feedback is that the dashboard provides too much information. This point of feedback cannot be addressed in this thesis, because information should not be randomly removed and this feedback was only provided by one researcher, as such this point is currently left untouched.

High first git commit time

The first participant was able to commit code in about 20 minutes. This seems like a high amount time for new users to get started, considering the small data science task. It should be noted that this value can be explained by the fact that the first step, Data, was not needed for the participants, as they already had a ready to use dataset. This meant they immediately turned to step two, creating a data schema, which took some time figuring out. Decreasing this time should be made a priority. A high first git commit time means researchers need a lot of time to get started. That might mean the workbench runs more risk that researchers completely give up on it. To further decrease this time, a lightweight data template can help.

Too many concepts

Another issue that emerged during the test is that there are a large amount of concepts that the researchers have to know about, such as continuous integration, static code analysis, how to define dependencies, the libraries used within the boilerplate code such as *Click* and the data schema tooling. During the relative short test session this can become a large impediment. More documentation is now provided that explains why these concepts are useful and how to use these concepts and tools can be used.

At the beginning of the evaluation, a short presentation was given to introduce why the workbench exists and what its main functionalities are. Some researchers did not get the point of the workbench after the presentation and its use. It is difficult to explain all the functionalities and the uses of the workbench in such a short session. The evaluation would have benefited from a longer presentation, but this wasn't possible due to time constraints (with the 10-minute initial presentation, we barely finished

on time). Also, the workbench itself could introduce these concepts better, as some researchers skipped the introduction tutorial since it contained too many details. This issue can best be fixed through separate user tutorials and sessions.

General technical issues

During the set-up, quite a few technical issues emerged, mostly present on the laptops of of the participants. For one participant, some of the packages would not install due to some unexplained reason. There was not enough time to troubleshoot these issues. In order to prevent these kinds of issues for a future evaluation, I recommend to let the participants ahead of time create and set up an experiment environment. Optionally, a pre-experiment session could be set-up so that researchers have enough time to set up an experiment environment with the right tools, but this does require a substantive commitment from the participants that might not be available. This was not done here because it was expected that the workbench and the provided boilerplate code was well-tested, since it is based on an existing data science template, was easy to use and that the workbench provided adequate instructions on how to use the boilerplate code. As such, such a session was deemed unnecessary initially. I also assumed that the participants were all familiar with and used Python 3 for their experiments, which turned out not to be the case.

Small task

While the data task was suitable for this evaluation, as at least some were able to complete it, the problem with this task is that the workbench is not designed for these small experiments and does not work well for them. It works better for larger experiments. For a small task, you might already be done after one git commit, which means the workbench was never able to provide you with feedback on the dashboard. In the real-world, researchers will work on experiments multiple days, during which they can check the dashboard and do have the information available to improve their experiment and workflow.

Gather more information about participants

This evaluation made some assumptions about the participants. They were expected to be somewhat technical and familiar with one programming language and programming version. During the test, it became clear that this was not to be the case for all participants. That immediately resulted in problems for two of the participants, as the evaluation hinged upon everyone using one programming language with the next day sharing code with each other. That could not be done when one researchers used on language and another used another language, because this is not very practical and this workflow is currently unsupported by the workbench.

6.2 Conclusion

During the evaluation, useful feedback was provided that can be used to improve the workbench. While the user test itself was not very successful, as many researchers

had issues getting started and not all researchers completed the data science task, the feedback gathered can be used to improve the workbench.

The goal of this evaluation was as follows: I consider the goals and requirements of the workbench to be met when researchers can...

- **accomplish their own research goals using the workbench**: Some researchers were able to complete their data science tasks using the experiment. Not all though, partly due to reasons unrelated to the workbench, namely client-side software issues and unfamiliarity with the used programming language.
- **while using the functionalities outlined in the previous section**: The adoption of the workbench functionality was quite low and should be improved.
- **without too much interference or change in their workflow**: Researchers found the experiment steps overall to fit to their workflow.
- **and when the artefacts they created show that they have used these services. For example, are they writing tests, defining data schemas and defining dependencies**: The artefacts show that they did not use these services. No tests were written, some data schemas were defined and no dependencies were defined.
- **I also use feedback given by the researchers to find out what they like and do not like about the system**: Useful feedback was provided by the researchers to improve the workbench.

The larger issue at play here is that the workbench introduces many new concepts that have to be adopted by the researchers. This is a difficult task and is part user education and part the workbench task. The workbench makes many of these concepts optional, for example, so that at the start they do not have to use these tools which makes adoption easier, as it can be done step by step.

The main feedback that can be drawn from this user test is that there either should be a 'workbench light' or much more documentation and information. On both aspects, work is done to improve this, by providing a lightweight template and extended end-user documentation. Also, for the future, I recommended to educate researchers on why these aspects are important so that they see the point of the tools offered by the workbench, as it was clear that the point did not come across.

As is clear from the experimental design and the actual performed sessions as well as the conclusion, the evaluation did not go as planned. Overall, with regards to the complete process of this thesis and its different phases, a balance had to be struck between the implementation phase and the evaluation phase. At some point the project has to be ended, because there are time limits. On the one hand, you want to have enough implementation time, in order to be able to actually build something that solves the problem at hand and being able to evaluate it. On the order hand, you want to have enough time for a rigorous evaluation. In this case, the balance between the implementation phase and the evaluation phase might have been missing and more weight was placed on the implementation phase.

This resulted in the fact that there was too little time to do more in the evaluation and to prevent some of the issues that were present and might have been avoided with more preparation. For example, to set up a pre-session for a softer introduction to the

workbench, and doing interviews with the participants of this user test to determine their skill and knowledge level. I also might have been able to educate users more with regards to the purpose and use of the workbench.

At the start of this chapter, the research question was asked: 'Having implemented the design, does the implemented solution achieve its requirements defined by RQ1?'. Based on this evaluation, I cannot answer this question. The user-test is designed to evaluate the extent to which the requirements are met. According to this user test, there is room for improvement to meet the requirements of the workbench, because researchers had much difficulty using the system to the fullest extent. With regards to reproducibility, the fact that researchers were able to create a data schema is an improvement over the existing way experiments are created. However, to make a more everlasting impact on the reproducibility of experiments created by the researchers, much more adoption of the workbench features is required.

Chapter 7

Conclusions and Future Work

This chapter gives an overview of the project's conclusions, contributions and future work.

The objective of this research is to improve the ability of researchers to share and reproduce experiments of MOOC research by building a workbench targeted for large-scale learning-analytics researchers.

To accomplish this goal, this thesis first looks at relevant literature to determine what MOOCs are, what MOOC research is and where MOOC research is headed. Based on this, it is clear new challenges for MOOC research exist, namely the use of data science in education and improvements in sharing data. Then, the issues with sharing and reproducibility in Computer Science in general are studied. Here, this thesis finds that challenges exist in difficulty in selecting and using tools that enable reproducible research, lack of a workflow on what tasks should be accomplished to enable reproducibility and the manual work required by researchers to enable reproducibility, with a lack of resources or incentives to perform that work. Also, specific to MOOC research is a struggle with regards to sharing data, with large data sets and concerns regarding data protection.

After the literature review, this thesis turns its attention to the Web Information Systems department of the University of Technology Delft, that serves as a use case for this thesis. This is a representative group of people for the larger research community, considering the cutting-edge research performed at this department regarding large-scale learning analytics. To better understand how researchers are currently designing, building, sharing and reproducing experiments, a set of user interviews are designed and performed. Based on the interviews, this thesis concludes that no standard workflow or approach exists for creating an experiment, not many steps are taken to ensure reproducibility of experiments created by researchers, researchers have limited experience with reproducing experiments created by others and code sharing happens via different mediums with issues encountered such as lack of code quality or missing documentation. The result of the user interviews and literature is a set of requirements for a possible workbench.

This thesis then focuses on the conceptual design of an experimental workflow. It studies other workflow systems and 'reproducibility enhancing systems'. Based on these systems, the thesis presents the key properties that the experimental workflow should adhere to, together with the actual experimental workflow. Key properties are

that it should be lightweight, change behavior, promote literate programming, be flexible and provide time for reflection and optimization. In the design proposal, this thesis presents the steps the experimental workflow should consist of.

After the conceptual design, I have implemented a solution that promotes this workflow. This is done in the form of a workbench, where researchers can create and manage their experiments. The workbench checks for several aspects known to improve reproducibility of experiments, such as defining dependencies, performing static code analysis and applying software engineering principles in general to experiments. An important part of the workbench is sharing code. The workbench implements this by asking researchers to share their code whenever they complete a step in the experimental workflow. With a few clicks, researchers can create a package and publish it in the workbench, making it easily installable by other researchers.

In the final chapter, this thesis designs an evaluation to determine the extent to which the implemented workbench fulfills its requirements. The evaluation consists of two session and simulates real-world data science work. Researchers are asked to solve these data science problems by using the workbench and share code. In the second session, researchers perform a different task, but then use shared code from the previous session, so that I can identify if the means of sharing code promoted by the workbench is efficient. Based on the evaluation, I can determine that the workbench does not fulfills its requirements fully. The evaluation did not succeed: Only one session was executed, since researchers had many problems getting started with the workbench. Issues arose especially during the phase of setting up their experiment and researchers did not fully understand the purpose of the workbench. However, much useful feedback was gathered from the evaluation and improvements were made to the workbench.

Based on the evaluation, it is impossible to consider the workbench a success. While some researchers were successful in using the workbench, to share code and create a data schema, more work is needed to educate researchers on the functions and need of the workbench and future work is needed to make the workbench more suitable for researchers that have less technical knowledge.

7.1 Contributions

The main contributions of this thesis are:

- A literature study of MOOCs and MOOC research and sharing and reproducibility in Computer Science and MOOC research, together with challenges identified from literature regarding these aspects.
- A user interview set-up to determine and solicit requirements for sharing and reproducibility in MOOC research.
- A set of requirements where a solution that wants to improve sharing and reproducibility in MOOC research should focus on.
- A key properties and a conceptual design of an experimental workflow for MOOC researchers and that wants to improve sharing and reproducibility in MOOC research.

- An architectural overview of the implemented solution.
- A workbench designed to meet the set of requirements identified earlier and implemented according to modern web engineering practices.
- An evaluation designed to simulate the real-world data science tasks in order to determine the extent to which a solution designed to improve sharing and reproducibility for MOOC research achieves the set of requirements identified earlier.

7.2 Future work

For future work, the evaluation shows that the workbench needs to be simpler. While the concepts introduced are useful for these experiments, as shown by literature, effectively applying them is a difficult task. The workbench succeeds in doing this up to a certain point, namely for the more advanced users that were able to use some of the functions, most of them do not have the technical knowledge required for using these functions. As such, for future work the goal should be to make the workbench as lightweight and user-friendly to use as possible.

Ideally, the workbench is tested using an actual large-scale learning analytics experiment for the evaluation. This way the need for simulating a real-world data science task is eliminated. In that case, it is sure that the workbench is tested in the real-world and furthermore it can be tested continuously that way.

The goal of this thesis is to improve sharing and reproducibility of MOOC science. Initially, the idea of this thesis was to create a workflow system, comparable to systems that are often seen in natural sciences and economics, where it is possible to drag and drop modules to perform some computation. This workbench should be seen as a first step towards that goal. If the workbench achieves improving code sharing and improves the reproducibility of experiments, then a next version of the workbench could start the work towards that goal. When some packages are present in the workbench, that are peer-reviewed and well-tested, a workflow system can naturally be introduced in the workbench that allows for quick experimentation in learning-analytics problems.

Also, for future work the workbench could address literate programming tools by integrating support for them and provide better means for visualizing results. Ideally, this is combined with the workflow system, so that a single environment is provided where researchers can import existing shared packages and write their Jupyter notebooks directly in the browser for prototyping purposes.

Another consideration might be to use metrics with gamification. This was not addressed in this thesis, as it is first more important to find the right metrics that researchers find useful and apply them. Once these metrics have been found, then a gamification system might be devised.

Another addition could be to introduce code reviews. An example workflow for this could be as follows: For each experiment, this experiment is done with another researcher in the same department that knows the goal and status of the experiment. After each experiment step is completed, the other researcher can perform a code review and offer suggestions and improvements to the source code. Code reviews are a

useful tool to maintain a coding standard, plus they have the advantage that researchers work together more and thus can share more knowledge and code.

Finally, the literature survey found some systems that were meant to improve the reproducibility in some way, shape or form, but also failed to get researchers excited and had disappointing usage numbers. This was taken into account in this thesis, but it should be seen as a warning, also for future work. What could help is, as a start, an improved computer science curriculum with more focus on software engineering practices, which might create more awareness and in turn more interest in a solution such as the workbench and possibly in other reproducibility enhancing systems.

7.3 Source code

The workbench that is created in this thesis is available on GitHub[47] and released under the MIT license.

[47] https://github.com/MOOCworkbench/MOOCworkbench

Bibliography

[1] Ryan SJD Baker and Kalina Yacef. The state of educational data mining in 2009: A review and future visions. *JEDM-Journal of Educational Data Mining*, 1(1):3–17, 2009.

[2] Adam Barker and Jano Van Hemert. Scientific workflow: a survey and research directions. *Parallel Processing and Applied Mathematics*, pages 746–753, 2008.

[3] Carl Boettiger. An introduction to docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1):71–79, 2015.

[4] Philippe Bonnet, Stefan Manegold, Matias Bjørling, Wei Cao, Javier Gonzalez, Joel Granados, Nancy Hall, Stratos Idreos, Milena Ivanova, Ryan Johnson, et al. Repeatability and workability evaluation of sigmod 2011. *ACM SIGMOD Record*, 40(2):45–48, 2011.

[5] Aras Bozkurt, Nilgun Ozdamar Keskin, and Inge de Waard. Research trends in massive open online course (mooc) theses and dissertations: Surfing the tsunami wave. *Open Praxis*, 8(3):203–221, 2016.

[6] Guanliang Chen, Dan Davis, Markus Krause, Efthimia Aivaloglou, Claudia Hauff, and Geert-Jan Houben. Can learners be earners? investigating a design to enable mooc learners to apply their skills and earn money in an online market place. *IEEE Transactions on Learning Technologies*, 2016.

[7] Guanliang Chen, Dan Davis, Jun Lin, Claudia Hauff, and Geert-Jan Houben. Beyond the mooc platform: gaining insights about learners from the social web. In *Proceedings of the 8th ACM Conference on Web Science*, pages 15–24. ACM, 2016.

[8] Vasa Curcin and Moustafa Ghanem. Scientific workflow systems-can one size fit all? In *Biomedical Engineering Conference, 2008. CIBEC 2008. Cairo International*, pages 1–9. IEEE, 2008.

[9] Andreas Dautovic, Reinhold Plösch, and Matthias Saft. Automated quality defect detection in software development documents. In *First International Workshop on Model-Driven Software Migration (MDSM 2011)*, page 29, 2011.

[10] Dan Davis, Guanliang Chen, Tim van der Zee, Claudia Hauff, and Geert-Jan Houben. Retrieval practice and study planning in moocs: Exploring classroom-based self-regulated learning strategies at scale. In *European Conference on Technology Enhanced Learning*, pages 57–71. Springer, 2016.

[11] Dan Davis, Ioana Jivet, Rene Kizilcec, Guanliang Chen, Claudia Hauff, and Geert-Jan Houben. Follow the successful crowd: Raising mooc completion rates through social comparison at scale. In *Proceedings of the 7th International Conference on Learning Analytics and Knowledge*.

[12] Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.

[13] H Drachsler and M Kalz. The mooc and learning analytics innovation cycle (molac): a reflective summary of ongoing research and its challenges. *Journal of Computer Assisted Learning*, 2016.

[14] Sarah Eichhorn and Gary W Matkin. Massive open online courses, big data, and education research. *New Directions for Institutional Research*, 2015(167):27–40, 2016.

[15] Norman Fenton and James Bieman. *Software metrics: a rigorous and practical approach*. CRC Press, 2014.

[16] Richard G FitzJohn, Matthew W Pennell, Amy E Zanne, Peter F Stevens, David C Tank, and William K Cornwell. How much of the world is woody? *Journal of Ecology*, 102(5):1266–1272, 2014.

[17] Daniel Greenfeld and Audrey Roy. *Two Scoops of Django: Best Practices for Django 1.8*. Two Scoops Press, 2015.

[18] Dustin Heaton and Jeffrey C Carver. Claims about the use of software engineering practices in science: A systematic literature review. *Information and Software Technology*, 67:207–219, 2015.

[19] Holger Hoefling and Anthony Rossini. Reproducible research for large-scale data analysis. *Implementing Reproducible Research*, pages 1–17, 2014.

[20] Maria Joseph Israel. Effectiveness of integrating moocs in traditional classrooms for undergraduate students. *The International Review of Research in Open and Distributed Learning*, 16(5), 2015.

[21] Upulee Kanewala and James M Bieman. Testing scientific software: A systematic literature review. *Information and software technology*, 56(10):1219–1232, 2014.

[22] Andreas M Kaplan and Michael Haenlein. Higher education and the digital revolution: About moocs, spocs, social media, and the cookie monster. *Business Horizons*, 2016.

[23] Raman Keswani, Salil Joshi, and Aman Jatain. Software reuse in practice. In *Advanced Computing & Communication Technologies (ACCT), 2014 Fourth International Conference on*, pages 159–162. IEEE, 2014.

[24] Ji Liu, Esther Pacitti, Patrick Valduriez, and Marta Mattoso. A survey of data-intensive scientific workflow management. *J. Grid Comput.*, 13(4):457–493, 2015.

[25] Tharindu Rekha Liyanagunawardena, Andrew Alexandar Adams, and Shirley Ann Williams. Moocs: A systematic study of the published literature 2008-2012. *The International Review of Research in Open and Distributed Learning*, 14(3):202–227, 2013.

[26] Fabrizio Marozzo, Domenico Talia, and Paolo Trunfio. Js4cloud: script-based workflow programming for scalable data analysis on cloud platforms. *Concurrency and Computation: Practice and Experience*, 27(17):5214–5237, 2015.

[27] Timothy McPhillips, Tianhong Song, Tyler Kolisnik, Steve Aulenbach, Khalid Belhajjame, Kyle Bocinsky, Yang Cao, Fernando Chirigati, Saumen Dey, Juliana Freire, et al. Yesworkflow: a user-oriented, language-independent tool for recovering workflow information from scripts. *arXiv preprint arXiv:1502.02403*, 2015.

[28] K Jarrod Millman and Fernando Pérez. Developing open-source scientific practice. *Implementing Reproducible Research*, 149, 2014.

[29] Ahmed Mohsen. Moocs integration in the formal education. *International Journal of Infonomics*, 9(3):1210–1216, 2016.

[30] Gina Moraila, Akash Shankaran, Zuoming Shi, and Alex M Warren. Measuring reproducibility in computer systems research. Technical report, 2014.

[31] Andrew Ng and Jennifer Widom. Origins of the modern mooc (xmooc). *Hrsg. Fiona M. Hollands, Devayani Tirthali: MOOCs: Expectations and Reality: Full Report*, pages 34–47, 2014.

[32] Zachary A. Pardos and Kevin Kao. moocrp: An open-source analytics platform. In *Proceedings of the Second (2015) ACM Conference on Learning @ Scale*, L@S '15, pages 103–110, New York, NY, USA, 2015. ACM.

[33] Michael Quinn Patton. *Qualitative evaluation and research methods* . SAGE Publications, inc, 1990.

[34] Roger D Peng. Reproducible research in computational science. *Science*, 334(6060):1226–1227, 2011.

[35] Fernando Pérez and Brian E Granger. Ipython: a system for interactive scientific computing. *Computing in Science & Engineering*, 9(3), 2007.

[36] R. Prieto-Diaz. Status report: software reusability. *IEEE Software*, 10(3):61–66, May 1993.

[37] Todd Proebsting and Alex M Warren. Repeatability and benefaction in computer systems research. Technical report, Technical report, Univ. of Arizona TR 14-04, 2015.

[38] Justin Reich. Rebooting mooc research. *Science*, 347(6217):34–35, 2015.

[39] Robert A Rhoads. *MOOCs, high technology, and higher learning*. JHU Press, 2015.

[40] Nick Rozanski and Eóin Woods. *Software systems architecture: working with stakeholders using viewpoints and perspectives*. Addison-Wesley, 2011.

[41] Carolyn B. Seaman. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering*, 25(4):557–572, 1999.

[42] George Siemens. Massive open online courses: Innovation in education. *Open educational resources: Innovation, research and practice*, 5:5–15, 2013.

[43] George Siemens and Ryan SJ d Baker. Learning analytics and educational data mining: towards communication and collaboration. In *Proceedings of the 2nd international conference on learning analytics and knowledge*, pages 252–254. ACM, 2012.

[44] Victoria Stodden, Christophe Hurlin, and Christophe Pérignon. Runmycode. org: a novel dissemination and collaboration platform for executing published computational results. In *E-Science (e-Science), 2012 IEEE 8th International Conference on*, pages 1–8. IEEE, 2012.

[45] Pieter Van Gorp and Steffen Mazanek. Share: a web portal for creating and sharing executable research papers. *Procedia Computer Science*, 4:589–597, 2011.

[46] George Veletsianos and Peter Shepherdson. Who studies moocs? interdisciplinarity in mooc research and its changes over time. *The International Review of Research in Open and Distributed Learning*, 16(3), 2015.

[47] Joost Visser, Sylvan Rigal, Gijs Wijnholds, and Zeeger Lubsen. *Building Software Teams: Ten Best Practices for Effective Software Development*. " O'Reilly Media, Inc.", 2016.

[48] Greg Wilson, DA Aruliah, C Titus Brown, Neil P Chue Hong, Matt Davis, Richard T Guy, Steven HD Haddock, Kathryn D Huff, Ian M Mitchell, Mark D Plumbley, et al. Best practices for scientific computing. *PLoS Biol*, 12(1):e1001745, 2014.

[49] Anna Wingkvist, Morgan Ericsson, Rudiger Lincke, and Welf Lowe. A metrics-based approach to technical documentation quality. In *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the*, pages 476–481. IEEE, 2010.

[50] Yihui Xie. knitr: A general-purpose package for dynamic report generation in r. *R package version*, 1(7):1, 2013.

[51] Junji Zhi, Vahid Garousi-Yusifoğlu, Bo Sun, Golara Garousi, Shawn Shahnewaz, and Guenther Ruhe. Cost, benefits and quality of software development documentation: A systematic mapping. *Journal of Systems and Software*, 99:175–198, 2015.

Appendix A

User interview questions & summarized answers

background information

What kind of research do you do? Often researchers are involved with LAK and EDM domains. Some are also working in Learning at scale, Behavior analytics and sometimes also some data mining, data science and web science.

What are some of the topics you research? Topics range from designing interventions of students to detecting how students are cheating in MOOCs. Also, research is being done to try and measure the attention of students via the webcam. With regards to behavior, research is being done on how students personality affects their MOOC performance, as well as trying to help students get immediate benefits by teaching them tasks and finding freelance tasks for them.

how researchers are currently creating experiments (which methods, tools, approaches)

How do you get started on an experiment? Each new experiment is based off of previous work, according to one researcher. Each new finding, each study and each experiment leads to more questions. Usually, researchers design an experiment in such a way as to address what is not known yet. A MOOC experiment usually consists of two parts, namely an experimental part or data-gathering phase, and an analysis part, but this can depend upon the type of experiment being performed.

Do you base your experiment of existing work, do you start from scratch or does it vary? Overall, researchers base their work often off of existing work, whether it is their own work or other people's work. At the very least, researchers usually find some parts of their earlier experiments that can later be reused. In terms of code, the parts that can be reused are usually translating the raw data logs into a database and the data analysis part.

What programming language(s) do you use during the experiment? Most researchers use a combination of Python and R. In terms of versions, both Python 2 and Python 3 are used, but the current trend is moving towards Python 3. Sometimes, for the implementation phase, JavaScript is used, as well as SQL.

What tools do you use during the experiment to help with the development? A large part of the useful libraries used throughout the experiment happens during the

analysis phase. An important part is data visualization, for that part a lot of popular libraries are used. One of the researchers, even though they usually program in Python, specifically for this purpose uses R.

One of the researchers prefers to use Jupyter notebook. The researcher finds that Jupyter enables them to write some code and explain what the code does, what inputs are expected and what outputs are generated.

Another important tool is a machine learning toolkit, used for clustering and classification, as well as the database that is used, namely MySQL.

What factors were involved in choosing these tools? One researchers chooses libraries based on popularity. Other reasons mentioned by researchers are familiarity by colleagues and that the tool helps to improve readability of the code.

What are your experiences with these tools? In general, R tools are very well documented.

The experiences with Jupyter are generally positive. It allows to write code section by section with clear documentation. A limitation of Jupyter is that code written in the notebook cannot easily be important into other Python files. Instead, researchers often copy from notebooks into their own experiments, because this cannot be done directly.

MySQL was chosen because of familiarity, as well as that researchers like that it can be updated easily.

How often do you reuse code from your own experiments on average? Although oftentimes an experiment is started from scratch, it happens often that some code is carried over between experiments. For the analysis part, one researcher mentions that almost always code is carried over from one experiment to another. This is because the analysis part is almost always the same, as often the same statistical tests are being run, with only small variations.

Other researchers also indicate that carrying over code from one experiment to another happens quite often. The main parts that are carried over are about data translation, translating the raw data logs of a MOOC into a database.

What is your experience with reusing your own code? Researchers encounter difficulties when reusing their own code, such as that they no longer know the way the code has to be used or do not understand it because it was written quite some time ago.

what steps are already being taken to ensure reproducibility of the experiments created

How do you account for reproducibility in your experiments? For one paper, a researcher published a manuscript in advance. outlining what they were going to do in terms of the analysis to be run, as well as the data that was going to be used.

In general, during the development of an experiment, researchers do not seem to consider reproducibility very much. One of the reasons mentioned is, for example, when a deadline is approaching, the focus of the project is the completion of it, not so much making sure the research is reproducible. Oftentimes though, if the code is released to the public, certain actions are taken, such as making the code more readable and adding a README document. Also, actions are being taken to create some common tools, such as extracting daily logs and placing them in the database, making sure the code can be used by other researchers.

One researcher has published a manuscript in advance of an experiment. This manuscript outlines all the analysis that were going to run. Also, work is being done together with other universities to apply an existing experimental design to other courses.

Another researcher uses Jupyter, which helps to improve the reproducibility of an experiment, as this kind of workflow enables them to write extensive documentation for each section of code.

How do you manage the dependencies within an experiment? In R it is possible to define within the script which libraries to install, so that is a form of dependency management directly within the code that is used by researchers.

Most researchers directly install their dependencies system-wide into their machine. They do not use virtual environments. Although they do write down the required dependencies, this usually only happens when an experiment is made public, although also not every time. Also, no versions are recorded of dependencies.

What kind of documentation do you write for your experiment? All of the researchers write source code comments, as well as usually a README file containing what the project does, how to use it, which dependencies are used and what it is designed for.

When do you write your documentation? The source code comments are usually written during the development phase of the experiment, the README is only done afterwards, but not always, only if the code will be released publicly.

Do you use a version control system, e.g. git? The use of version control systems is limited by researchers. One of the researchers does not use it, others use it sparingly.

How do you use the version control system? One researcher only pushes code when they consider their project to be complete. Another uses it a bit more often, whenever they consider their source code to be complete, they commit and push their changes, and one other only uses it as a back-up, for example when the laptop is destroyed or stolen. Branching and tagging is not used.

How do you test your source code? All of the researchers run their code with the data set until the desired result is achieved. No formal testing is being done to verify the correctness of the source code, although it should be noted that this is sometimes not possible, because of the lack of a training set.

After the experiment is finished, how do you make the source code available to others? One of the researchers uses the Open Science Framework. This is a place where researchers can upload their code and data, to make it openly available to others. The advantage of the OSF is that it requires little to no extra work for the researcher and that it is public for all.

Other researchers publish their source code on GitHub. One researchers mentions that they link directly to the GitHub source code in their paper, to publish a version of the code that works.

What kind of actions do you perform to prepare the source code for public release? For using OSF and to make an experiment available to others, almost no actions are required, apart from uploading the code.

For GitHub, in order to prepare the source code for public release, some passwords and/or sensitive information has to be scrubbed. Also, some of the researchers focus on making the code more readable, more efficient and more concise before making it public on GitHub.

After the experiment is finished, how do you make the data set available to others? One of the researchers also uses the Open Science Framework for this goal, as this also allows for the data set to be uploaded.

However, the EdX data set cannot be made public, as this violates the privacy policy of EdX, so usually the data set is not made public.

How could the reproducibility of MOOC experiments be improved? An important part to improving the reproducibility of MOOC research, mentioned by almost all researchers, is data. Many improvements can be made towards writing high-quality functions to translate the raw data into a database. This means specifically that this code is easily accessible by all and well-documented. Another issue with regards to the data, is that the collected data within a MOOC differs from university to university. The code written to analyze these data is very sensitive to changes, which makes reproducing research hard.

Another issue is documentation. One researcher mentions that others often use different strategies or approaches or methods, and feels that the documentation in projects is lacking, at the very least a complete README document should be present.

One researcher mentions that papers of MOOC research almost always do not contain links to the source code and suggests that more experiments should be open source.

Also, even though efforts have been made in prior literature, even mentioned in this thesis, in the form of MOOCdb, here also issues exist. MOOCdb is a conceptual schema and does not contain an actual implementation. Universities are free to implement it as they see fit. The TU Delft, for example, has added some database fields and disregarded others.

Another improvement that can be made is the use of more reusable code. In a lot of experiments, the same code can be used again, for example to extract the scores of learners, or the time spent on videos in a MOOC, but this code is not shared or reused among researchers.

Another point that is part of how reproducibility can be improved, is that researchers do not know of the existence of useful functions, as there is no centralized database containing MOOC experiments and functions.

Also, researchers in general do not seem to focus on making code reusable, because they simply focus on making sure the code runs correctly, making sure the entire program runs fast and correctly, retrieve the results and then move on. There is no phase during the experiment to look back at the experiment and assess the work done, to look at the code and think: this might be useful in the future for me or for others.

how researchers are currently reproducing experiments and the problems experienced with reproducing

What are your experiences with reproducing a MOOC research experiment? Reproducing an experiment from beginning to finish has not happened yet for a lot of researchers.

One of the researchers has reproduced an experiment. Their experience was mixed. First of all, the relevant papers contained no source code, only an approach. The researcher used this approach to write the code needed to execute it. The results were quite different, though, so the researcher decided to contact the original authors to

request the source code. They were reluctant to provide source code, and set the condition that sharing of code should be bi-directional, as well as that they wish to be co-authors of any papers published using that source code. As such, both parties send each other source code of their experiments. The sent source code, in the end, however, could not be used directly, because of differences in the data format, among other things. The researcher notes that the sent source code is quite different from the original paper.

What are the steps involved with reproducing a MOOC experiment? There are no pre-defined steps for reproducing a MOOC experiment.

An example of some of the steps involved with reproducing a MOOC experiments, are the following. First, the procedure between the two researchers would have to be made clear, explaining what they are going to do, understanding their expectation and their experiment in detail. Depending if they require expert knowledge, the researcher would agree to participating in reproducing an experiment. The next step would be to read some documentation. Once it is clear what is going on in the experiment, the experiment will be reproduced step by step.

What could be improved upon the steps involved on reproducing a MOOC experiment? Suggesting ways to improve the reproduction of a MOOC experiment was found difficult by many researchers, as they have little to no experience reproducing experiments.

In general, the reproducibility of MOOC research as a whole should be improved. So, for example, one researcher mentions that a common toolkit could help. This toolkit could process specific data. Another suggestion is the increased use of visualization tools, which allows to better get a sense of what is in the data.

how researchers are currently sharing experiments and code, why they are sharing and the problems experienced with sharing

How often do you reuse code of other researchers? Reusing code from other researchers happens rarely. For one researcher, this has happened once or twice during their MOOC research life. Another researcher has reused code only once from other researchers, the same for the third one. Code is also reused from Stack Overflow, but only to solve a specific problem.

What are your experiences when reusing code from other researchers? The experiences are mixed. For two researchers, the code they received ended up not being used in the end. For another researcher, they made a selection of two libraries for a specific function. In one library, it was difficult to extract the functions needed to be used in their own experiment, as this library also performed other functions and was part of a larger whole. The second library, on the other hand, was made specifically for reuse among researchers.

Another researcher mentions that the biggest issue with sharing code is readability and the lack of documentation. This researcher also used some code from an external university, and ended up in the end not using the code. The researcher was unable to run the code and decided to write it by themselves.

How do you acquire the code or experiment from other researchers? Sharing either happens via GitHub, a personal project page of the author or via email.

What is your opinion of the current sharing process? The current way of sharing is for researchers sufficient.

Appendix B

User test presentation

User test

Workbench
set-up

What is the workbench?

the workbench...

- is a tool to make MOOC experiments more reproducible
- applies software engineering principles to experiments
  - e.g. continuous integration, static code analysis, testing and more

What is the workbench?

the workbench...

- helps you with sharing code
  - Think you've written code others can use? Easy to share/publish

Today

- you are going to evaluate the workbench
- using a real-world data set performing an actual small data science task
- ..and are asked to create a package of your code using the workbench
- two different data science tasks
- schedule:
  - 13:10 start with data science task
  - 14:45 end
  - 14:45 debrief group interview

Tomorrow

- we switch the data science tasks
- again perform a data science task, but this time using a package from someone else

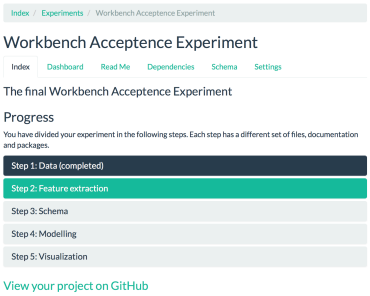
Short workbench intro

- experiment divided into steps, e.g. data gathering, feature selection, modelling
- for each step, the workbench helps you with SE stuff, (documentation, testing, packages)

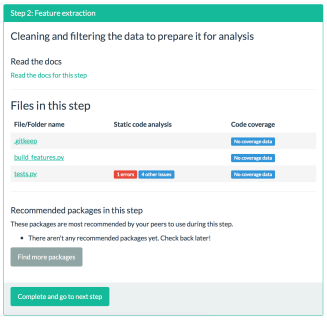
Short workbench intro

- after you have completed a step, you have the option to take the code into that step and move it to a package
- (make sure to do that today at least once, and before the last step (visualization))

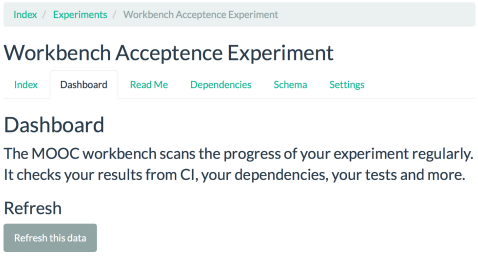
Short workbench intro



Short workbench intro



Short workbench intro



One more thing...

- ...for questions or comments about the workbench, please use Slack
- the chat log helps me with my thesis, as it provides a detailed log of confusion/reactions/etc.
- for other questions, ask me

- for today:
- please evaluate the workbench
- using a real-world data set performing an actual small data science task
- ..and are asked to create a package of your code using the workbench
- two different data science tasks
- schedule:
  - 13:10 start with data science task
  - 14:45 end
  - 14:45 debrief interview

Appendix C

Dataset readmes provided during user test

This readme was provided for the dataset IMDB movies and is an exact copy of the description and goal of the task at Kaggle.com.

C.1 IMDB movie dataset

The readme below was taken at 28th of May from Kaggle.com [48].

C.1.1 Background

How can we tell the greatness of a movie before it is released in cinema?

This question puzzled me for a long time since there is no universal way to claim the goodness of movies. Many people rely on critics to gauge the quality of a film, while others use their instincts. But it takes the time to obtain a reasonable amount of critics review after a movie is released. And human instinct sometimes is unreliable.

C.1.2 Question

Given that thousands of movies were produced each year, is there a better way for us to tell the greatness of movie without relying on critics or our own instincts? Will the number of human faces in movie poster correlate with the movie rating? Method

To answer this question, I scraped 5000+ movies from IMDB website using a Python library called "scrapy".

The scraping process took 2 hours to finish. In the end, I was able to obtain all needed 28 variables for 5043 movies and 4906 posters (998MB), spanning across 100 years in 66 countries. There are 2399 unique director names, and thousands of actors/actresses. Below are the 28 variables:

"movie_title" "color" "num_critic_for_reviews" "movie_facebook_likes" "duration" "director_name" "director_facebook_likes" "actor_3_name" "actor_3_facebook_likes" "actor_2_name" "actor_2_facebook_likes" "actor_1_name" "actor_1_facebook_likes" "gross" "genres" "num_voted_users" "cast_total_facebook_likes" "facenumber_in_poster"

[48] https://www.kaggle.com/deepmatrix/imdb-5000-movie-dataset

"plot_keywords" "movie_imdb_link" "num_user_for_reviews" "language" "country" "content_rating" "budget" "title_year" "imdb_score" "aspect_ratio"

To answer question 2, I applied the human face detection algorithm on all the posters using python library called dlib, and extracted the number of faces in posters.

C.1.3 Blog and Github codes

See here for more details about the scraping steps, the EDA, and the predictions : https://blog.nycdatascience.com/student-works/machine-learning/movie-rating-prediction/

Github page: https://github.com/sundeepblue/movie_rating_prediction

C.1.4 Important notes

This dataset is by no means to be a comprehensive scraping of all attributes relating to movies. It stemmed from one of my project built from scratch and finished in around one week. So please do not be surprised if you find something is off. This dataset is a proof of concept. It can be used for experimental and learning purpose to get hands dirty on web scraping, basic EDA, and learning algorithms in R or Python. For comprehensive movie analysis and accurate movie ratings prediction, 28 attributes from 5000 movies might not be enough. A decent dataset could contain hundreds of attributes from 50K or more movies, and requires tons of feature engineering.

There are around 800 "0"s in the "gross" attribute. This was either caused by (a) no gross number was found in certain movie page, or (b) the response returned by scrapy http request returned nothing in short period of time. So please make your own judgement when analyzing on this attribute.

There are around 908 directors whose "director_facebook_likes" attribute are 0. If somebody did analysis on "directory_facebook_like" attribute, there could be some off, and say, the top10, or top50 directors could be inaccurate. Thanks for pointing this out by user Kryslor. This is interesting, since the code I used to scrape everybody's facebook like were identical. See function parse_facebook_likes_number(). It was hard to directly scrape this data from IMDB website (due to dynamic embedded div frame), so I had to use a hacky way by directly sending request to facebook website (see line 38 of this file). Perhaps for some directors, facebook did not respond with reasonable result within short timespan (< 0.25 second) and returned "None" in Python (translated to 0 in my code). For those 0s, you might want to treat them as "missing value" when using certain machine learning algorithms.

Thanks to user "Quinton", who found a bug in the dataset on 11/23/2016: (November 23, 2016 at 12:08 am) We actually used your IMDB dataset for an Advanced Data Mining class at Rockhurst University in Kansas City, MO. We love the data set and we really appreciate the time it took to create the it. However, we believe we found a small flaw in the data. Not all of the IMDB movie budget numbers are in US dollars, for example, the South Korean movie "The Host" has its budget numbers in S. Korean Won (Korean currency). But there is no data in the dataset that tells you the currency. The existance of foreign currencies skews the budget data for foreign films particularly for currencies with extreme exchange rates when compared to USD. For instance, many could assume the data set shows "The Host" cost $12 billion to make when it truthfully cost only 12 billion Won, but the dataset doesn't make the distinc-

tion. It is not just an issue with Korean movies we found Turkish and Japanese movies with the same issue. Quinton was right. When I parsed the currency, I didn't take the Korean currency into consideration. Therefore please be cautious if you analyze the currency related attributes for non US dollar currencies. The fix is actually quite simple in the corresponding python code.

Please be mindful that, analyzing currency related attributes, such as "gross" or "budget", is actually more complicated than it seems. For a really thorough and accurate analysis (EDA or prediction), we may want to do some feature engineering on those attributes in a systematic way. For example, one US dollar in 1920 is different from that of 2010. So we need to take inflation factors across years into consideration, and normalize all US dollars into one basis (a certain year). So do all other currencies (British pound, Chinese RMB Yuan, etc). If you also consider exchange rate between two different currencies and wanted to convert everything into dollars, things become tricker, because even those rates also varies over time. $1 equals RMB8.4 in 2000 but RMB6.8 in 2015.

C.2 Gym crowdedness dataset

The readme below was taken at 28th of May from Kaggle.com [49].

C.2.1 Background

When is my university campus gym least crowded, so I know when to work out? We measured how many people were in this gym once every 10 minutes over the last year. We want to be able to predict how crowded the gym will be in the future.

C.2.2 Goals

Given a time of day (and maybe some other features, including weather), predict how crowded the gym will be. Figure out which features are actually important, which are redundant, and what features could be added to make the predictions more accurate. Data

The dataset consists of 26,000 people counts (about every 10 minutes) over the last year. In addition, I gathered extra info including weather and semester-specific information that might affect how crowded it is. The label is the number of people, which I'd like to predict given some subset of the features.

Label:
Number of people
Features:

- date (string; datetime of data)
- timestamp (int; number of seconds since beginning of day)
- day_of_week (int; 0 [monday] - 6 [sunday])

[49] https://www.kaggle.com/nsrose7224/crowdedness-at-the-campus-gym

- is_weekend (int; 0 or 1) [boolean, if 1, it's either saturday or sunday, otherwise 0]
- is_holiday (int; 0 or 1) [boolean, if 1 it's a federal holiday, 0 otherwise] temperature (float; degrees fahrenheit)
- is_start_of_semester (int; 0 or 1) [boolean, if 1 it's the beginning of a school semester, 0 otherwise]
- month (int; 1 [jan] - 12 [dec])
- hour (int; 0 - 23)

C.2.3 Acknowledgements

This data was collected with the consent of the university and the gym in question.

Appendix D

Implementation

This appendix contains extra content for chapter 5 Implementation.

D.1 Used technologies

The underlying framework of the workbench is Django[50]. This is a modern Web framework for Python that is very mature. The version used is 1.11, which is a long-term release and has support for critical fixes until 2020. Django contains an object-relation manager, which is quite powerful and it removes the need to write manual SQL with all the security issues possibly involved. Django is 'secure by default', which means it offers CSRF protections by default, SQL injection prevention through parameterised queries used in their ORM and it protects against XSS attacks by automatically escaping variables.

The workbench's features often result into long-running I/O tasks or POST tasks where it is dependent upon another web service. To prevent blocking and a negative user experiences, long-running tasks are run inside a set of Celery workers. Celery is an open source asynchronous task queue[51]. It uses a distributed messaging system and allows for adding an arbitrary number of workers, which are immediately available to perform tasks. Blocking tasks in the workbench are for example experiment creation.

Another used technology in the workbench is a library called Cookiecutter. To solve the boilerplate code problem, Cookiecutter is very useful, as it is specifically designed to solve this problem. Cookiecutter takes a template, some user input with for example the project name, project description and author and automatically fills the template with that information. This template is meant for any project, for example a data science project, which is then immediately ready to be used.

The workbench contains a Dockerfile so that it can easily be deployed using Docker, since it does require some extra technologies. For Celery, the workbench needs Redis and you need to ensure at least one Celery worker is always running, but more depending upon the number of users. For communication between client and server, the workbench uses websockets with django-channels[52]. This allows the workbench to push messages from the server to the client. The workbench uses this to send status

[50] https://www.djangoproject.com
[51] http://www.celeryproject.org/
[52] https://channels.readthedocs.io/en/stable/

updates with regards to experiment creation, for example. This set-up uses Daphne[53], a Django http server, together with a number of workers for handling the web requests.

Furthermore, the workbench contains a PyPi-server using the pypi-server package[54], to be used for sharing code. This pypi-server also needs to be running constantly.

For the front-end, things are kept as simple as possible. Sometimes jQuery is used to perform AJAX calls to the back-end, but only there where it makes sense to improve the user experience, otherwise generic HTML and CSS is used. In terms of front-end framework, Bootstrap 3 is used. While maybe not the most exciting choice in 2017, it is stable and complete and provides all the necessary functionality. Other JS libraries include intro.js[55] to provide the user with a first-time tour of their experiment, highlight.js to provide code highlighting and chartjs to plot a graph of the number of commits of the user on the dashboard.

To manage these JS dependencies, Bower is used, which unfortunately become end-of-life during this thesis. However, this is not considered to be a major pain point, since all the JS versions and libraries are still widely available.

During development, GitHub was used for version control management, together with pip to manage the Python dependencies. Travis CI provided continuous integration by running the test suite after every commit, while using Coveralls to measure code coverage and using Landscape.io to provide continuous static code analysis. Unfortunately, at the end of the thesis, Landscape halted static code analysis for open-source projects because it could not handle the traffic, so Pylint was run manually.

D.2 Implementation

D.2.1 Functional view

Build manager

The build manager is responsible for managing builds of the experiments. For this task, it uses Travis CI. To communicate with Travis CI, the Python library TravisPy[56] is used, which provides useful abstractions for the API for example to run a build, retrieve the last log of a build and to enable builds for an experiment with limited user intervention. The build manager is limited with its functionality, because Travis already provides most of the required functionality. For example, the workbench does not check in real-time whether builds failed or succeeded, since Travis already sends an email to users if their build has failed. Also, I simply display the Travis CI build badge of the repository for the researcher indicating whether builds fail or pass on the dashboard. To use TravisPy within the workbench, a special class is created that imports TravisPy and calls it appropriately, the TravisCiHelper. It provides functions to enable and disable travis for a certain repository, to trigger a build and more. This way the rest of the build manager does not have to worry about the TravisPy code and, in the future, it is easy to swap out the TravisCiHelper to use or add another build tool.

[53] https://github.com/django/daphne
[54] https://pypi.python.org/pypi/pypiserver
[55] http://introjs.com/
[56] https://travispy.readthedocs.io/en/stable/

Coverage manager

The coverage manager is responsible for providing code coverage results for the experiment of the researchers. For this purpose, it integrates with Coveralls. For Coveralls, a web API exists, but no Python library exists, so I manually perform the web requests to get the desired information from Coveralls. This information includes code coverage results per file and the total amount of coverage. To interact with coveralls, a helper class is used.

Docs manager

The Docs manager is very lightweight. The actual documentation generation happens in a shell script, that only receives the location of the cloned experiment on the server, which is a temporary folder. This is necessary because in order to build the documentation of an experiment, the dependencies of this experiment have to be satisfied. To avoid conflicts with the workbench, the workbench uses a separate environment in which to create a new virtual environment, install the dependencies and then generate the documentation.

Pylint manager

The Pylint manager is not they most accurate name for this Django app, as it also runs rlint for R scripts. This app does not contain any views, but it does contain several models to store the results from static code analysis. For each static code analysis result, the workbench stores the type of message from pylint, the line number and the relevant file, as well as the message. This result is connected to the entire scan object, which has a datetime to log when the scan was performed. This is useful, so that the static code analysis results can be shown in the front-end, when the researcher views a file we can show directly in the file the static code analysis results.

To perform the static code analysis for Python, the scan is only performed within the experiment step folder, so that only the active files, currently under development, are scanned and taken into account. A shell script is used to initialize the virtual environment of the experiment just like with the docs and the correct folder is passed as an argument to scan. The results of the Pylint scan are placed in a *.json file, which is after the scan is done parsed by the workbench and processed to a database object.

For R, a Python package importr is used. That way R scripts can be called directly from within Python. To scan an R file, we first install the package lintr from CRAN and scan the main module file and save the results.

Requirements manager

The requirements manager is a Django app that helps researchers manage their dependencies in both their experiments and their created packages. This app contains a model requirement, that is connected to an experiment. A requirement consists of a version number and a package name.

On the front-end, researchers can access the tab Dependencies and add, edit or delete a dependency. The views for this are managed through this app. When researchers are done editing, a task is started to update the dependencies in the GitHub

repository. For example, for a Python experiment, the Requirements manager generates a new dependency text file and commits this file to the repository, so that the changes are directly reflected in their GitHub repository and no other work is required. Inversely, when a researcher changes their dependency file locally, the workbench automatically parses this file, deletes the existing dependencies and adds the new dependencies, so that the information in the workbench is always up-to-date.

One specific design problem that had to be solved is that both the workbench and the actual dependency file in the repository have to be in sync. It is undesirable if a researcher can make some changes in the workbench, but those changes are never synced to the workbench. At the same time, the workbench also should not 'spam' the repository. You want to aggregate changes and commit them at once. However, if you wait too long, you risk going 'out of sync' and if you do it too fast, you risk that the workbench commits often which clutters the git log.

To solve this problem, the Dependency tab uses an edit mode. Before a researcher can make changes to the dependencies, they have to click the Edit button. Only then do the icons appear to make changes to dependencies, such as editing, deleting and adding. Once they are done, they can click the button Save Changes and Commit to GitHub. While the database changes are already saved, when pressing this button the task is started to update the dependencies in the GitHub repository.

User manager

The user manager contains functionalities for managing user and profile related tasks. This app is also responsible for the index page of the workbench.

The index contains an Activity feed. This Activity feed provides updates of actions within the workbench, with the purpose of improving awareness of researchers for the actions of their colleagues, in the hope that they themselves are more likely to contribute as well. In the Activity feed, actions such as publishing a package, adding a new package version, adding a package resource, publishing an experiment and more are added.

Initially, I looked at libraries providing existing functionality. One such library is django-activity-stream[57]. However, during development no stable version was available that was compatible with the used Django version. The library was integrated and evaluated, but a lack of documentation and customization of the activity messages made it unsuitable.

Instead, a simple solution was added: For the index, I gather the five most recent objects for a number of relevant models, combine this list and sort by the created date. Each object has a method get_activity_message with returns a user-friendly activity feed message.

SphinxHelper

The SphinxHelper is responsible for generating documentation of experiments. Sphinx is language-dependent, meaning it only works for Python. Currently, the workbench has no alternative for R. The SphinxHelper uses the GitHelper to clone the repository and then run the Sphinx command in the docs folder. To find the folder where

[57] https://github.com/justquick/django-activity-stream

the documentation field, it uses the experiment.template field. For each Cookiecutter template, the documentation folder has to be provided. Also, before the workbench can generate any documentation, a virtualenv has to be created and all the package dependencies have to be installed. Once Sphinx is done, the workbench moves the built HTML files into their own folder and into the gh-pages branch, which enables publishing the documentation on GitHub.io and makes them automatically visible and easily accessible.

GitHubHelper

The GitHubHelper provides an abstraction for interacting with GitHub. This helper uses the library PyGithub. The GitHubHelper contains methods for retrieving the GitHub username, creating a new GitHub repository, viewing a file, committing a single file, fetching commits, creating a new GitHub release and adding a new file. To access all these functions, the user initially signs in with GitHub in the workbench. The workbench then retrieves a social token, that is stored in the database. When the user accesses a function requiring GitHub integration, the GitHubHelper is instantiated with this token so they can perform all the actions in the workbench.

GitHelper

The GitHelper is a helper for commonly used git commands. It uses a package GitPython to interact with the Git program through Python. The GitHelper is necessary for more git related tasks locally on the file system. For example, for performing static code analysis the workbench needs an up-to-date version of the repository locally on the file system. The GitHelper can do that: Clone the repository and return the location of the folder of the repository.

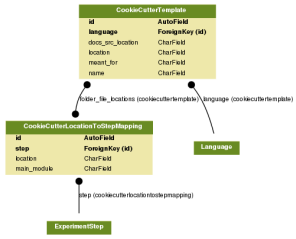
GitHub webhooks

The workbench has to stay up-to-date with the actions performed in the GitHub repository. To achieve this, upon experiment creation the workbench adds a GitHub webhook to the created repository. Using this webhook, every time the researcher pushes to the repository, the workbench receives a POST with the details of the commit. Upon receiving a commit, the workbench starts a number of tasks. First, it starts a task to update the data schema and update the dependencies, in case the user has changed either. After those, it starts a complete quality check, meaning all the quality measurements from the Quality Manager are run to ensure that the dashboard is always up to date.

R & Python support

To enable support for multiple programming languages, the workbench uses the class LanguageHelper. This LanguageHelper contains an interface with common functions for each programming language. An experiment has a property Language, which refers in the current version to either Python or R. An experiment also contains a function to return the relevant LanguageHelper, through a dictionary that contains as key the programming language and as value a non-instantiated LanguageHelper. When this

Figure D.1: UML diagram of Cookiecutter manager



function is called, the correct language is inserted in the dictionary at runtime and the right LanguageHelper is returned.

When some task is run, for example if we want to update the requirements of an experiment, this task can retrieve the correct LanguageHelper and instantiate it. The task then calls the function to update the requirements in the relevant LanguageHelper. The LanguageHelper contains all the specific language-dependent code for updating a dependency, so that the task itself does not have to worry about which programming language is used in the experiment.

D.2.2 Information view

Information structure

For the Cookiecutter manager, see figure D.1. When the administrator wants to add a new Cookiecutter template, the following information is required: A foreign key to a Language object, for example Python or R, the docs_src_location, the folder where the documentation is stored, the location, the GitHub URL to the template, meant_for which can be either an experiment or package and a name for the template.

For the Dataschema manager, see figure D.2. A data schema consists of a name and has a OneToMany relation to DataSchemaField. A DataSchemaField consists of a datatype, a description, name, a primary key and a title. A DataSchemafFeld has a OneToManyRelation to DataSchemaConstraints.

For the Experiments manager, see figure D.3. To store an experiment, the workbench needs the following information: A title, description and a language. The experiment model inherits from BasePackage, as both share a the language property to indicate which programming language the experiment supports. The experiment contains foreign keys to a docs, pylint, schema and travis instance. Furthermore, the experiment model contains the owner and a git_repo foreign key, if the repository has been initialized, and a completed boolean. unique_id and publish_url_zip are used at the end. When a researcher publishes their experiment, a unique id is generated and the GitHub release zip URL is stored, for the read-only experiment page, if they choose to publish their experiment in the workbench. An experiment contains a OneToManyField to ChosenExperimentSteps. At experiment creation time, researchers have to select their experiment steps. For an experiment step, the workbench stores the order of that step, when the step was started, if and when the step was completed and if the step is currently active. The location property is a file path to a folder in the Git repository, where the files of this step reside. The main_module is the main file

Figure D.2: UML diagram of Dataschema manager

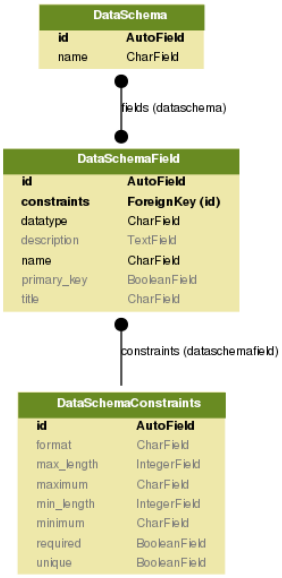
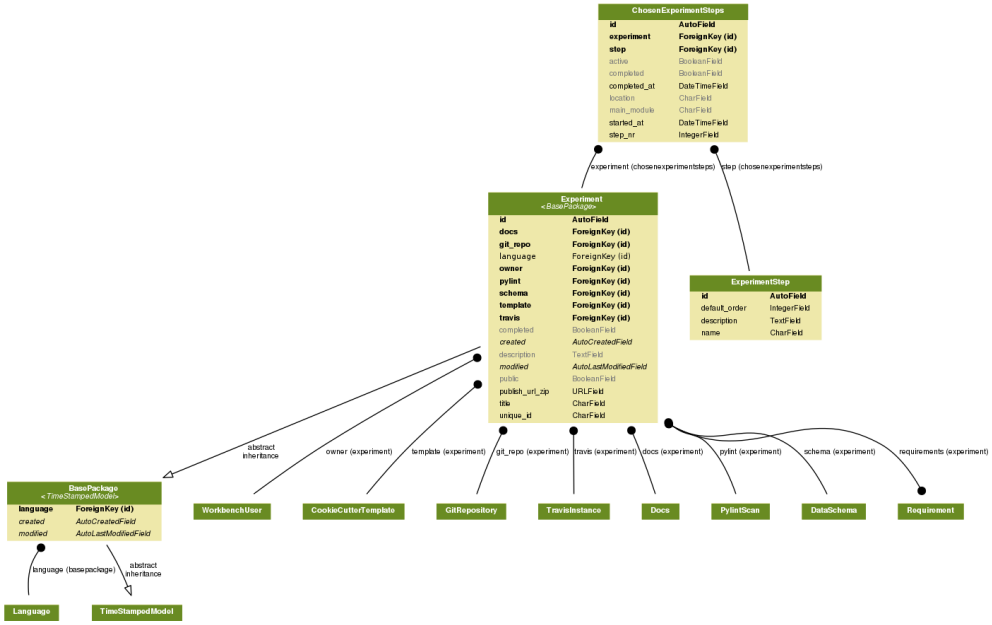


Figure D.3: UML diagram of Experiments manager



of this step, pre-defined in the Cookiecutter template to experiment step mapping. A ChosenExperimentStep has a foreign key to an ExperimentStep, which are the steps that can be chosen and can be changed in the workbench.

For the Git manager, see figure D.4. A GitRepository is connected to an experiment and stores the github_url, so that the workbench knows the location of the GitHub repositories. A GitHub repository has a OneToMany field to a Commit. A Commit

Figure D.4: UML diagram of Git manager

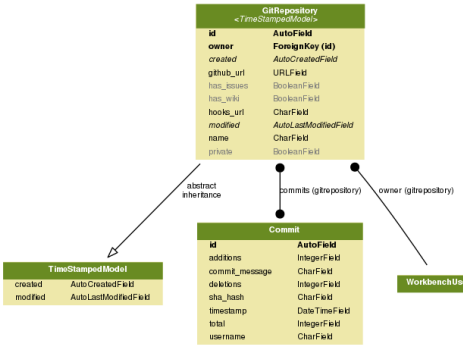
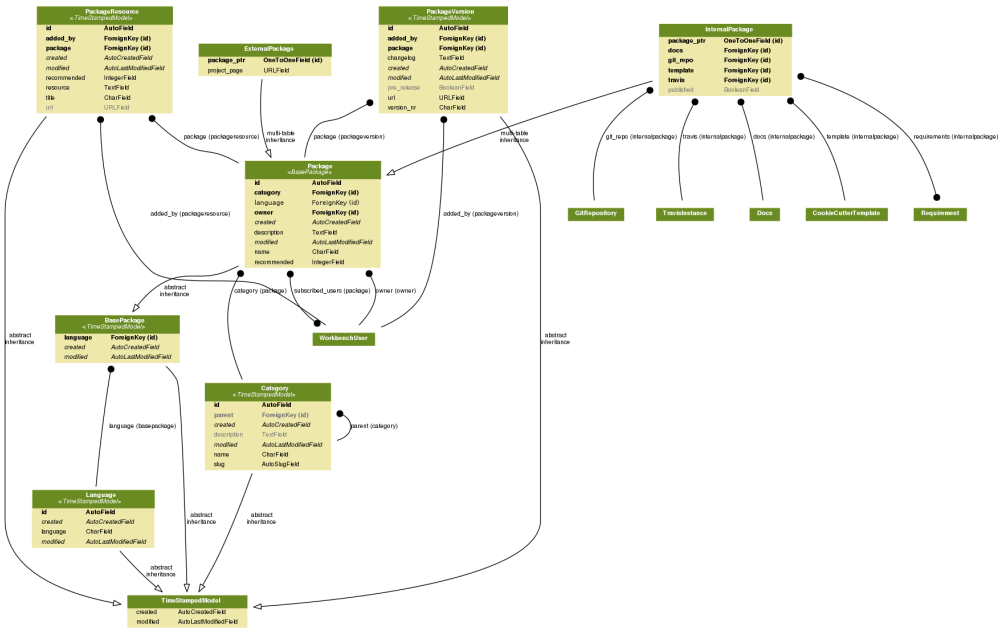


Figure D.5: UML diagram of Marketplace



is created upon receiving a GitHub webhook and stores the number of additions and deletions in that commit, the commit message, the timestamp and the SHA hash.

For the Marketplace app, see figure D.5. The Package model stores the category, language, owner and description, as well as when the package was created, the name of the package and when the package was last modified. Two models inherit from this model: ExternalPackage and InternalPackage. ExternalPackage adds the fields project_page, which is the source of this external package. InteralPackage adds foreign keys to docs, git_repo and travis instances, as well as a foreign key to the template used to initialize this package. A package has many PackageResources and PackageVersions.

The following models are present in the Quality manager. An ExperimentMeasure model defines the measurements that the workbench will check. For example, documentation is a measurement, which checks the docs coverage. An experiment mesaure

Figure D.6: UML diagram of Build manager

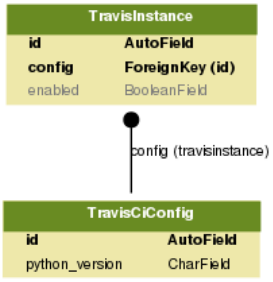


Figure D.7: UML diagram of Coverage manager

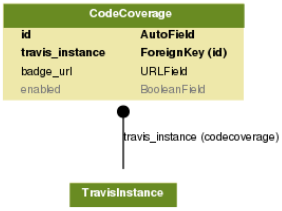


Figure D.8: UML diagram of Docs manager

Figure D.9: UML diagram of Pylint manager

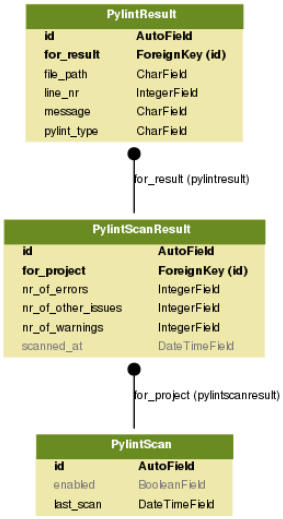


Figure D.10: UML diagram of Quality manager

Figure D.11: UML diagram of Requirements manager

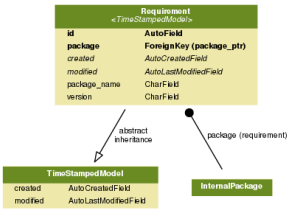
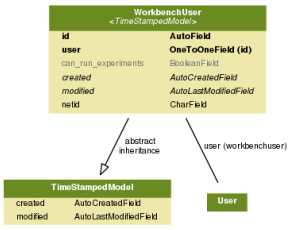


Figure D.12: UML diagram of User manager



consists of a name, a description and three messages, a message for a low, medium and high score. Each scan can have three results: high, medium or low. High means a good quality, low means low quality.

When an actual experiment is scanned, an ExperimentMeasureResult model is created. This model contains a foreign key to the experiment step that was scanned, to the ExperimentMeasure to define which measurement was performed, as well as a result consisting of low, medium or high. An ExperimentMeasureResult also optionally contains a raw value, for example the number of classes and functions left uncovered by documentation. A raw result consists of a key and value in the form of a character field, so the model is free to store whatever type they want.

D.2.3 Deployment view

Third-party software requirements

In case the software is run on bare metal, the workbench needs the following services to be available: Redis, R, Python3, Python3-pip, and git. To make the actual workbench website available, nginx or Apache can be used.

Hardware and software requirements

The workbench needs to be deployed on a system with at least 1 GB of memory, due to the number of services required. In case of Docker deployment, at least 2 GB of memory is recommended, together with at least 3 GB of available disk space.

D.2.4 Operational view

I recommend to set a cron job to empty the github_repositories/ folder every 24 hours. In case a task fails, a GitHub folder might not be cleaned up by the workbench. Everything in github_repositories can be considered as temporary files and are not pertinent to the functioning of the system.

During operation, at some point the system might grow and that might lead to long wait times before tasks are be completed. Administrators can monitor the Celery tasks using Celery Flower, reachable at http://host:5555. From this environment, tasks that are stuck can be killed. If Celery Flower indicates that tasks suffer from long wait times, we recommend to deploy extra Celery workers, possibly on different hosts. We refer to the Celery 4 documentation on how to do this.

D.3 Functionality

D.3.1 Creating a new experiment

Researchers will first interact with the workbench by creating a new experiment. They do this at the start of their experiment work.

Prior to creating an experiment, researchers have to sign in to GitHub using the workbench. They are automatically redirected upon creating a new experiment. The workbench asks for permission for all public and private repositories. That way the workbench can still be used when a researchers makes a repository private.

Walkthrough In the workbench, researchers enter a title for their experiment, as well as a short description, and choose which Cookiecutter template they want to initialize their experiment with. Next, the actual experiment is created. This means a GitHub repository in their account is created, the Cookiecutter template is initialized and committed to their repository. For a graphical overview of experiment creation, see figure D.13.

Researchers can add an existing GitHub repository as experiment in the workbench. To accomplish this, they have to enter the GitHub repository name as the title of their experiment. The workbench recognizes a GitHub repository already exists, skips the Cookiecutter initialization and immediately moves on towards the next step of the experiment creation process.

After experiment creation is complete, the wizard continues and researchers select which steps they expect to use during their experimental process, consisting of the renamed steps outlined in the previous chapter. They are free to add, remove and reorder these steps, but have to select at least one.

On the final step, researchers can set up Continuous integration and code coverage measurements, by signing into Travis CI and Coveralls.io, and set-up their experiment locally so they can get to work.

Setting up the experiment locally consists of the following steps: cloning the git repository, installing the packages present in the Cookiecutter template, running the default tests. Optionally, researchers set up a virtual environment.

Figure D.13: Walkthrough of creating a new experiment

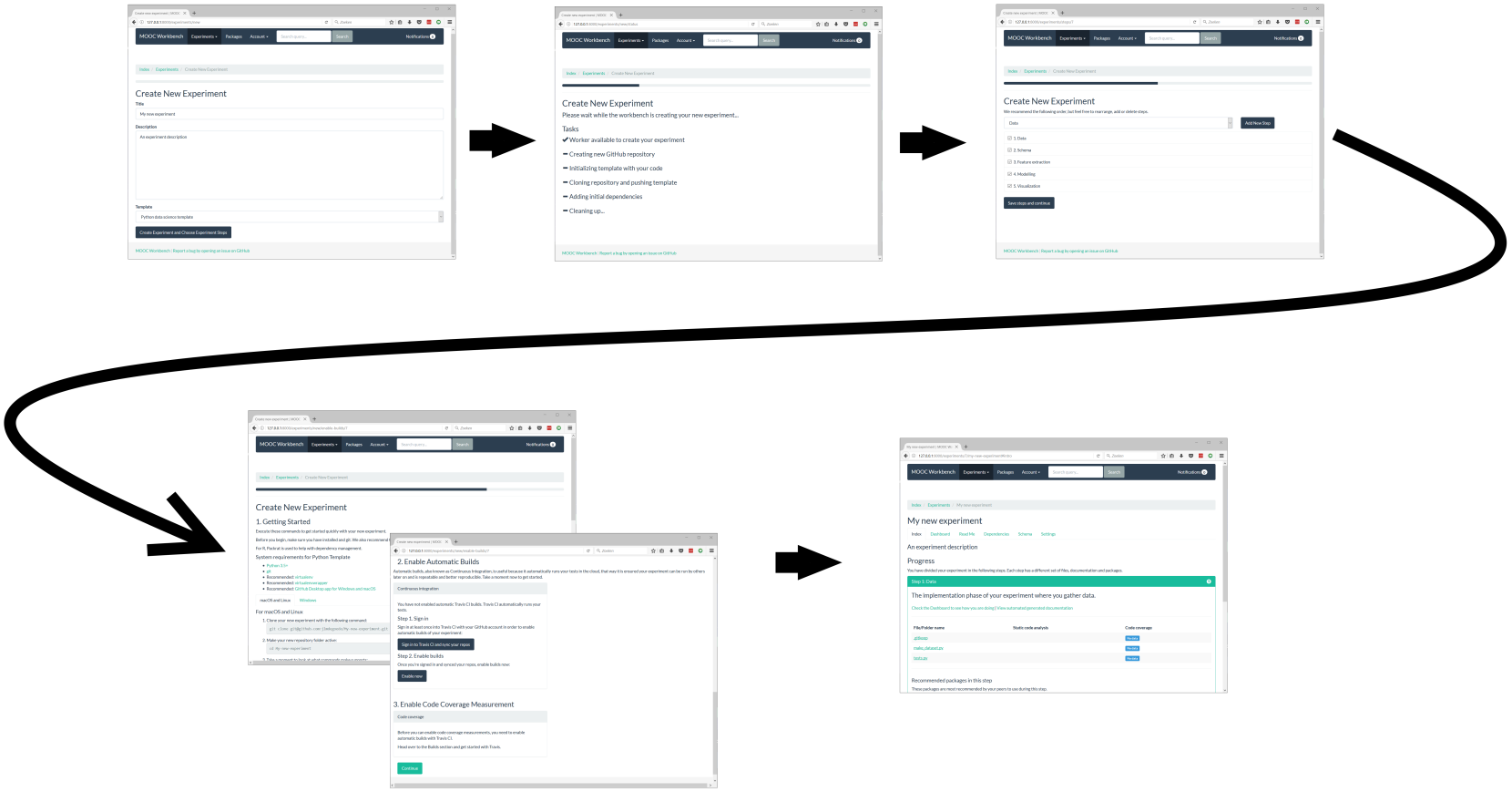
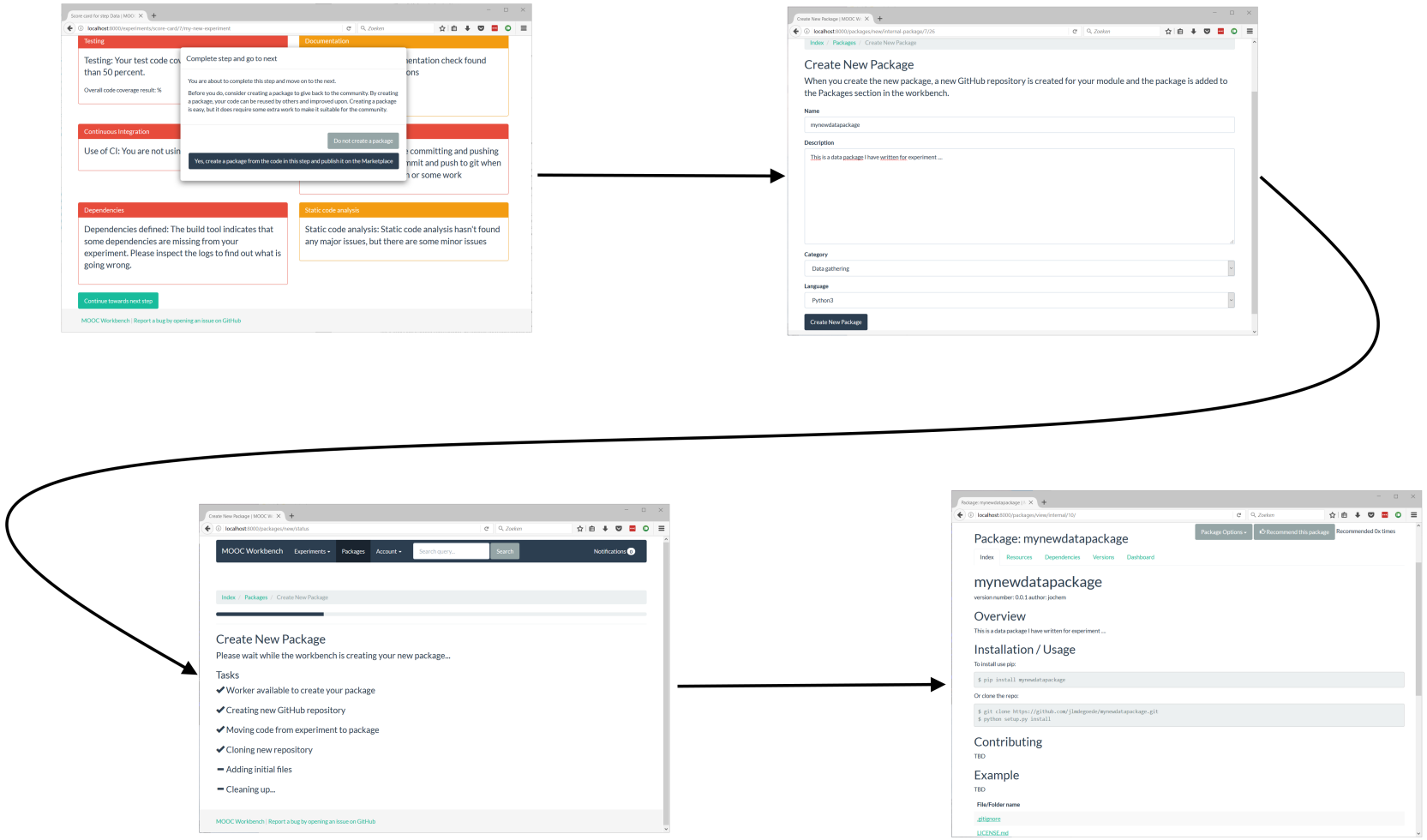


Figure D.14: Walkthrough of creating a new package (1) Upon experiment completion, researcher asked to create package (2) Filling in required package information (3) Creating new package, such as moving code, creating GitHub repo, (4) Package creation complete



D.4 Implementation phases

The implementation development process used an agile approach towards developing. The characteristics of an agile approach are that the work is being done in sprints of 2 to 4 weeks. This development process consisted of the following sprints:

- Initial phase
- Marketplace & Initial setup
- Sharing & Dashboard
- UX & Data Schema
- Packages & R
- Beta-ready

For each development phase, beforehand I determined which requirements would be addressed in that phase. Those were then translated into tasks in the form of GitHub issues to track the progress of the sprint as a whole. At the beginning of each sprint, I first started with creating the necessary database models. Then, the actual work took place implementing the functionality. At the end of the sprint, two or more days were used to verify and validate the features.

In the next few sections, this chapter outlines each development sprint individually, show what work was done and what the sprint focused on, which requirements were addressed in each sprint, how these requirements were translated into functionality of the workbench, why the choices were made and how the chosen way to address these requirements fits with the larger design goals of the workbench.

Initial phase

During the initial phase, development on the workbench was restarted and I took some time to orientate upon the work ahead. In this phase, the foundation was laid for the future work, by selecting a web framework for both the back-end and the front-end. For more regarding this choice, review the section D.1. Also, I set-up the main development infrastructure and selected the development tools that would be used during the development phase. In this phase, I also started work for the following requirements:

should support the different phases of MOOC research Early designs were created for the experiment detail page, showing the experiment steps and the total progress of the steps.

Should be a MOOC tool marketplace, where researchers can add, find, use, run and install common MOOC tools and functions Let researchers create external MOOC packages and start work on the Marketplace index where researchers can view already created external packages.

Marketplace & Initial setup

During this sprint, the work was mainly focused on the set-up experience of researchers, such as creating an experiment, the main experiment page and getting started with the Packages section of the workbench. Functional requirements addressed in this sprint, are:

Should facilitate version control Measure the frequency of version control use: In this sprint, I used the GitHub API to retrieve the last three weeks of commits and calculate the median and mean of these commits. In a later sprint, I determined this did not adequately meet this requirement and it was changed.

Should facilitate writing documentation throughout experiment Track whether or not documentation is present in GitHub with Sphinx: In the initial sprint, the boilerplate code was integrated in the workbench. To accomplish this requirement, I added Sphinx to the boilerplate code and used the extension sphinx-coverage to retrieve a list of undocumented functions and classes. Sphinx is specifically meant for Python and it fits nicely with the design-goal to prevent workbench lock-in, since you can use the default Python way of writing docstrings directly into the source code. Sphinx extracts these and builds them into a complete static HTML website.

However, choosing Sphinx has a down-side, as this means researchers have to use reStructredText. That is a trade-off that had to be made with regards to the mark-up language. Ideally, Markdown is used for documentation, since it is much easier to use and write and I expect researchers to be more familiar with this mark-up language. On the one hand, you have Sphinx with reStructured Text, with the pros of being able to write your docstrings directly into the source code, on the other hand there is Markdown, which would allow researchers to more easily write extra documentation.

I made the decision to use Sphinx with reStructedText, knowingly researchers might struggle getting started with this mark-up language and be off put with writing extra documentation. This choice was made because: (1) encouraging researchers to write docstrings directly into the source means they can most accurately describe their source code (2) the extra documentation of research is often already provided in the form of papers (3) if they want to write extra documentation using markdown, they can still do so using GitHub README / wiki. Also, Sphinx is very popular among Python developers in general and has wide support.

Should facilitate dependency management Allow researcher to define requirements within their experiment: Created the relevant dependency models in the database and allow the researcher to define their own dependencies through the workbench.

Parse requirements.txt file: Defining dependencies is a two-way street and can be done both in the repository and in the workbench. To support defining dependencies in the repositories, the requirements.txt file needs to be parsed in the workbench.

Boilerplate code: Provided default way of defining dependencies. I made the choice to use the requirements.txt file for Python, as this is the default way in Python to specify dependencies and it is a nice fit for this context, as it allows for freezing all the dependencies and their versions and this approach can be extended with own packages.

Should give insight into how reproducible each experiment is Track previous progress: For each quality measurement, such as measuring version control and measuring the amount of undocumented functions, the workbench supports storing raw values, so that measurements can retrieve previous raw measurements and use historic data in the decision.

Should give useful actions on how to improve reproducibility of an experiment One overview page for the entire experiment (Dashboard): In this sprint, work was started with the addition of a dashboard page that showed an overview of all the

measurements the workbench performed. Up and until now that was only shown in the experiment detail page.

Initially, code coverage measurement did not exist, instead the workbench measured the amount of new tests written, as due to the oracle problem researchers would probably never achieve high code coverage and might be discouraged to put efforts into tests. By rewarding them when they wrote new tests, I hoped to at least achieve that they thought about tests and work from there.

Other features added in this sprint consisted of a wiki using an existing package, as well as the start of designing and creating the boilerplate code. During this sprint, this design focused on providing researchers with a requirements file, a starting file, test scaffolding and a default folder structure, which created for each experiment step a different folder with default files, as well as signing in with GitHub through django-allauth[58].

Sharing & Dashboard

The Sharing and Dashboard sprint focused on creating an internal package based on code from the experiments and ways of sharing code.

Should be a MOOC tool marketplace, where researchers can add, find, use, run and install common MOOC tools and functions With this sprint, I started the work on code sharing within the workbench. Before I began designing this functionality, it was clear that it needed to fit with the workflow of the experiment steps. It seemed like the best time to let researchers create a package is when they have just completed an experiment step. That is the point where they have just written the code, so they still know what the purpose of the code is and if it might be suitable to be shared with others. As such, upon completing an experiment step, the workbench prompts the researcher with the option to create a new package from the experiment step they have just completed. If they choose to do so, they are redirected to the package creation page.

When a researcher wants to share their code from an experiment step, the workbench creates a Pip package of their code. Pip stands for "Pip install Packages" and is a package manager for Python. It is was chosen for the workbench because it is the standard package manager for Python. Since Python 3.4, pip is installed by default. Furthermore, using pip is very simple. To install a package, a researcher only has to run the following command: pip install <package-name>. For more about package creation, see the section 5.3.2.

Other work done with regards to packages is to give the researcher the ability to update their package. Whenever they have committed some code, they can create a new package version by providing a version number and a changelog. The workbench handles creating a new version and creates a new GitHub release.

Also, to prevent workbench lock-in, for each new package created, the workbench creates a new GitHub repository for the user with the package code.

Creating and publishing a package are two different steps: Just because a package is created does not mean it is ready for publishing. The workbench expects the researcher to write some documentation of their experiment and to define the required dependencies.

[58] https://github.com/pennersr/django-allauth

An earlier chapter identified code sharing as being a separate step in the total experiment step workflow. This approach, where find and reuse is a part of every step, is preferred, since it fits more naturally into the experiment workflow and it sits closer to when researchers have written their code.

Should give useful actions on how to improve reproducibility of an experiment I also decided to change the code coverage and only use the code coverage measurement, as this is the default way of using code coverage and think it should be possible for researchers to achieve sufficiently high code coverage.

UX & Data Schema

This sprint focused on improving the overall user experience and adds a way for researchers to define their data schema. This sprint also improved the boilerplate code templates.

The initial approach of providing boilerplate code had too many downsides, as was determined during testing. The old approach was a completely custom solution, which interferes with the design goal of preventing to reinvent the wheel. Also, since the template was not public, it interfered with the design goal of preventing workbench lock-in. As such, with the changes learned from the initial version, I looked for improvements in this regard.

These improvements were found in the form of Cookiecutter[59]. This is a Python library that allows for creating boilerplate code using a template. Creating a template is very easy and this library allows for saving templates in a GitHub repository, making them easily accessible to others.

Supporting Cookiecutter means I don't reinvent the wheel on two counts: (1) not much custom code is needed to provide templating functions for the workbench, since it is a Python library that the workbench can import directly and (2) using Cookiecutter allows for using all the other created Cookiecutter templates, of which there are plenty, so that whoever uses the workbench has the possibility to add another or their own templates and are not limited to the ones provided by the workbench.

Should give useful actions on how to improve reproducibility of an experiment In this sprint it became clear that researchers can 'evade' the Dashboard and completely miss the actions given there. To improve this situation, the workbench adds a report score card. When a researcher wants to move on to the next experiment step, the workbench first redirects them towards this report score card, showing how they have done with regards to documentation, static code analysis and more in this step, showing code-colored panels for each step, so that they can see how they have done (red - not good, yellow - room for improvement and green is good).

Should facilitate the creation of a data schema of an experiment To facilitate the creation of a data schema of an experiment, several approaches were considered.

One approach that was initially considered, was very drastic: namely completely deprecating the use of CSV files and using SQL. However, this approach was dismissed because of the impact it would have on researchers, the interference with used libraries such as pandas and the limited advantages provided by such an approach, other than having a perfect data schema in the form of a SQL table description.

[59] https://github.com/audreyr/cookiecutter

A different and more suitable approach came in the form of JsonTableSchema. This is one of the few libraries that seems to satisfy the requirements of the workbench.

Packages & R

This sprint focused on packages in the workbench, namely publishing them and installing them, and support for the R programming language.

For packages, this sprint focuses on publishing a package and making sure that packages can be installed by others. Because the workbench uses Pip packages for Python, which means the workbench can use the library PyPi server to serve these packages, so that they can easily installed by others. Publishing a package is then limited to publishing it on the PyPi server, which is very easy.

Initially, the idea was that the workbench should be a complete tool for information exchange. To accomplish this, the workbench integrated a third-party wiki solution. This package had to be removed because of instability in combination with other used library functions. No suitable replacement was found, partly due to the used version of the web framework, which was still relatively new during the months of development. This Django version was heavily preferred however, due to it being a long-term version.

Furthermore, in this sprint a first time guided tour was added to show the user the main functionalities of the workbench.

Also, in this sprint the workbench changed some default settings and automatically enables all the workbench services whenever possible, so for example automatic documentation generation, static code analysis and more.

During the user interviews, it was clear not all the researchers used Python to create their experiments. Because of this, the workbench has to provide the same services for the R programming language. This posed several challenges in this sprint: (1) all of the functions so far were all created specifically for Python, (2) for all the functions so far R alternatives needed to be found and they needed to be integrated into the workbench. One might argue that it would have been better to initially design the workbench in such a way as to immediately support several programming languages, but I disagree: Because of the initial prototypes with Python, using it, I learned a lot of lessons of what is possible and what works, lessons that can be applied to R.

To achieve this, some code had to be refactored. An interface was created that has all the methods specific for a programming language, such as adding a dependency and generating documentation. These were implemented for both Python and R.

For R, I have made the following choices: For Package management, Packrat[60] is used. Packrat is specifically meant for making research using R more reproducible. It supports easily defining dependencies and supports shipping R experiments with all the dependencies present. It also features a similar system like virtualenv in Python, where dependencies are only installed locally in a project instead of system-wide. The disadvantage of Packrat is that it is a third-party library and not natively supported by R. For dependencies, the workbench is for R and Python feature equivalent.

For documentation, no R alternative is present. I could not find a library suitable for this context. A possible solution had to fulfill the following requirements: (1) writing documentation directly into R source files and (2) extracting this documentation

[60] https://rstudio.github.io/packrat/

and building HTML pages from them and (3) showing which functions and statements are not covered with documentation. One serious contender was roxygen2[61]. However, such a library only works for packages. If the workbench were to use such a solution, this meant that the R boilerplate template had to be redesigned in such a way that every experiment step of the R template became a package. Such a set-up interferes too much with the workflow of researchers, due to the large number of folders and boilerplate code required, because that meant the R template contained five packages by default. That was unacceptable and thus I abandoned this library.

Being unable to find a suitable library for R makes sense, since R is often used for short scripts that are supposed to be self-documenting. Most document tools we have found, support creating a paper directly from an R script, but that is unsuitable for the workbench, since it does not document the actual source code and thus does not help with code sharing.

For testing, the template provides R boilerplate testing code. For Continuous Integration with R, the Cookiecutter template has a configuration for Travis CI that automatically installs all the R dependencies and runs the tests. For code coverage, this is not supported in R due to technical difficulties: Travis did not submit the R code coverage measurements to Coveralls, but this might work in a future version. Due to the use of Cookiecutter, creating and adding an R template to the workbench was very easy. For R packages, the workbench supports the same features as Python, meaning researchers can create packages from an experiment step. The workbench creates all the boilerplate code for an R package. To serve packages, I made the choice to use the R feature of being able to install packages directly from GitHub, since this does not require any extra work on the server-side. For the data schema, JsonTableSchema's R version is not ready yet. In a future version, as a temporary work-around a short Python file might be added that creates the data schema or better yet, tooling is built on the workbench website itself, so that the data schema step becomes language-independent.

Beta-ready

The final sprint was focused on getting all the features of the workbench into such a shape that they were ready to use and test by researchers. As a single developer, you have to balance testing and adding value to the workbench, but since you are alone, testing and finding bugs is difficult since you know how to use the workbench. Thinking outside the box for testing is difficult and a different skill then engineering.

During this sprint, the workbench was tested extensively, also by another person and I have also made sure the relevant logs for the user test for the evaluation phase are present. At the end of the sprint, the test suite of the workbench achieved a code coverage of 81 percent.

[61] https://cran.r-project.org/web/packages/roxygen2/vignettes/roxygen2.html