MASTER'S THESIS 2021

Architectural Technical Debt - can it be prevented with a software project generator?

Sara Ahrari, Tselmeg Baasan

ISSN 1650-2884
LU-CS-EX: 2023-79

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2023-79

Architectural Technical Debt - can it be prevented with a software project generator?

Arkitekturell teknisk skuld, kan det undvikas med en mjukvaruprojektgenerator?

Sara Ahrari, Tselmeg Baasan

Architectural Technical Debt - can it be prevented with a software project generator?

Sara Ahrari
`sa8623ah-s@student.lu.se`

Tselmeg Baasan
`ts2530ba-s@student.lu.se`

July 12, 2021

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisors: Martin Höst, `martin.host@cs.lth.se`
Niclas Lövdahl, `niclas.lovdahl@jayway.com`

Examiner: Per Runeson, `per.runeson@cs.lth.se`

Abstract

Architectural Technical Debt (ATD) is a challenge that many of today's companies aim to overcome. ATD concerns the overall structure and architecture of a software-intensive system and can be described as the unresolved shortcuts developers take under circumstances of e.g. time pressure and uncertain requirements. Planning and constructing a well-thought-out architecture is time consuming and resource-intensive and if not done carefully, it can cause ATD. This thesis aims to investigate if it is possible to prevent the rise of ATD with a software project generator and how to maintain it. We have carried out a study following the design science research paradigm at a case company, Jayway by Devoteam. We conducted both literature and empirical studies to better understand the current situation at the case company and based on that proposed a feasible solution. According to our findings, a project generator tool can mitigate the causes of ATD at the beginning of a project. However, ATD can be introduced later on due to various factors, for instance negligence of best practices. Moreover, we concluded that the most suitable way for maintaining such a tool is adopting Inner Source.

Keywords: Architectural Technical Debt, Technical Debt, Project Generator, Command Line Interface, Inner Source

Acknowledgements

We would like to express our gratitude to our supervisors Martin Höst and Niclas Lövdahl and our examiner Per Runeson for their valuable guidance and ideas throughout this thesis. We would also like to thank the employees at Jayway for volunteering and donating their time to participate in our research and providing us with valuable information, we could not have done it without you.

Contents

Chapter 1

Introduction

In today's highly competitive world of digitalization, keeping the customers satisfied is a great competitive advantage for the software companies. Not only does the project process need to be agile in order for a company to stay reactive to the customers' changing requirements, but the product itself has to support quick alterations as well. It has to be flexible, maintainable and evolvable. One way to achieve this is by constructing a well designed architecture for the system [13].

While designing the architecture, decisions about the system's structure, frameworks, technologies, languages, development process and platform need to be made. Planning and constructing a well-thought-out architecture is time consuming and resource-intensive. Moreover, companies often strive to minimize the lead time to satisfy the customers, meaning minimizing the time between the identification of a customer's needs and the delivery of a solution. This gives rise to a dreadful dilemma. Either the quality of the architecture is prioritized and the lead time is increased, or the quality of the architecture is compromised and the lead time is cut down. It is a constant balancing between long-term solutions and short-term solutions, and it presents itself in most, if not all, software systems in the industry today. This phenomenon is called Architectural Technical Debt (ATD) which is a sub category of the more general term Technical Debt (TD).

The term Technical Debt was coined in 1992 by Ward Cunningham, with the definition '*Shipping first-time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. Objects make the cost of this transaction tolerable. The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt*' [5]. ATD might give rise to detrimental consequences if not dealt with in time. It is therefore of great importance to recognize and manage the ATD before it is too late. A common misconception is that ATD is only present in mature systems. It has in fact been shown that it can be introduced right from the start. Therefore, it is a good idea to identify ATD and take actions against it instantly [15].

The ATD subject has attracted academic attention during the past years, and several studies have been conducted. There are a number of strategies for managing ATD. However, there are to our knowledge very few studies about strategies for preventing ATD.

This thesis is conducted according to the design science research paradigm, as a case study at a large scale software consultancy company. The employees at the case company stated that the majority of their projects struggle with ATD. They also informed us that they experience various challenges while setting up new software projects. This inspired us to investigate if there is a relationship between ATD and the challenges faced in the project setup.

Furthermore, the case company asked us to study if a software project generator can be used to mitigate these challenges. A software project generator is a tool that automatically produces a standard template, including boilerplate, which can be used as a scaffold or a foundation for further development. The standard template is implemented according to the current projects and the employees' needs at the case company, meaning that the content, i.e technologies and tools, are commonly used and proven to work from projects at the case company. If the challenges appear to be related to cause ATD, and the software project generator shows to be effective in tackling the challenges, does it mean that the software project generator can be used to mitigate ATD? Finally, a software project generator is required to be maintained in order to remain useful. How is it best maintained? This thesis aims to answer these questions.

1.1 Outline

In this chapter we present the thesis, its aim and purpose, and the case company. In Chapter 2 we discuss some background for this thesis, including descriptions of important concepts and related work. Thereafter, in Chapter 3 the design science research method is described in detail and the thesis process is presented step by step. The results are then presented in Chapter 4, which are further discussed in Chapter 5. The discussion also assesses the thesis's rigor, relevance, novelty and introduce potential areas for future work. Lastly, the results and discussion are concluded and summarized in Chapter 6.

1.2 Case Company

The case company of this thesis is Jayway by Devoteam (Jayway) which is an international software consulting company with approximately 240 developers and designers. It was established in Malmö in 2000, and has managed over 1200 different projects over the years. Today it has clients from big companies in a large range of domains, including everything from construction of railways to telecom operators. Jayway is slightly different from typical consulting companies and brands itself as a hybrid between a product company and a consulting company. It takes responsibility for entire projects from clients and assembles in-house teams of employees which develop and maintain the custom designed products for both long run and short run projects. The company has offices in five locations, including Malmö, Copenhagen, Halmstad, Stockholm and Silicon Valley. This thesis was carried out at the Malmö office.

1.3 Research Questions

The purpose of this thesis is to firstly uncover the challenges experienced by the employees at Jayway while initializing new software projects. Thereafter, we aim to ascertain if these challenges introduce ATD to the projects. Most importantly, we seek to investigate if a software project generator is a feasible solution to mitigate the identified challenges, and determine if it can be used to prevent ATD from being introduced at the beginning of a project. Furthermore, the world of software is in constant movement, therefore we aim to study various maintenance strategies for the generator to retain its relevance and usefulness. Finally we choose one strategy that is best suited for the solution and Jayway.

The following research questions are be explored and discussed in this thesis:

RQ1: What challenges do employees at Jayway experience when setting up new software projects from scratch?

RQ2: Do any of these challenges introduce ATD to the projects?

RQ3: How can a software project generator be used to mitigate these challenges that cause ATD?

RQ4: How can such a project generator be maintained in large scale consulting companies such as Jayway?

1.4 Contribution

Both authors have collaborated in all activities; research, implementation and documentation of all parts of this thesis. However, for the literature study, while Tselmeg Baasan focused more on ATD, Sara Ahrari focused more on the maintenance (Inner Source) part.

Furthermore, this master thesis is a case study where we help the case company to further understand their projects and their similarities, and potentially aid them to find a way to improve their software development processes. Our findings regarding maintenance is valuable to the case company and might help them in determining which strategy to pursue for maintenance of future in-house projects. Additionally, the contributions from this thesis might raise awareness about ATD and inspire the case company to take actions. Apart from this we aim to provide the case company with new knowledge, which can be used to further investigate this area in for instance upcoming master theses. We also hope to contribute to the general research on how to develop common project generators for software consulting businesses. Lastly, as mentioned previously, little study has been conducted on methods for preventing ATD, therefore we hope to add scientific content to this area as well.

Chapter 2
Background

This chapter presents the underlying theory behind concepts referred to in this thesis. Section 2.1 provides an overview of ATD. It describes the causes, symptoms and consequences of ATD in software companies. Since an internal software product is developed for employees at Jayway, it is of great importance that it is maintained consistently to remain up to date and relevant. A well established methodology that has proven to be successful when maintaining in-house projects at software organizations is Inner Source, which is introduced in Section 2.2.

2.1 Architectural Technical Debt

Software systems are complex, and technical debt (TD) can be incurred in all its different components and activities, in fact there are 13 different types of TD [1]. Architectural technical debt (ATD) is, as the name suggests, incurred in the architecture of the system, and is estimated to be the most expensive and serious technical debt [7]. Besides ATD, requirements TD is also experienced as tedious work and hard to resolve [2], and hence has a negative impact on daily work. Other examples of TD are source code TD, automation TD and infrastructure TD. All roles involved in the development and management of the software system are affected negatively by ATD [2]. Despite this, management often seems to be reluctant to spare resources to resolve the issues and repay the debt [14].

Deliberately introducing a shortcut or a workaround does not necessarily have to be a bad decision. Quick short-term solutions satisfy the customers, and hence have a large business value. Although, letting it accumulate into a cluster of short-term solutions can have detrimental effects on the development. Constantly postponing refactoring may in worst case lead to a 'crystallized' architecture, meaning that it is impossible to make changes to the system, leaving the system unmaintainable and non-evolvable. Another negative effect of having a stiff architecture is difficulties in carrying out parallel work. Other costly consequences of ATD is reduced development velocity and carrying cost which refers to the cost of the extra resources required. In a poorly structured architecture with unpredictable behaviour debugging is more time consuming than usual. Furthermore, getting a grasp of the system might take several hours, slowing down the development and making it difficult for developers to onboard or introduce new features [27].

There are numerous causes to ATD, business factors such as time pressure and unclear requirements, architectural factors such as unsuitable decisions and insufficient knowledge and documentation, technology factors such as reuse of legacy e.g open source and evolution and human factors such as varying levels of expertise and ambition. The term *ATD item* refers to parts of a system or an entire system in some technical debt. A cause gives rise to an ATD item, which in turn results in one or more consequences [27].

ATD items can be difficult to identify, especially if they are 'dormant', meaning that they are present in the system but do not yet cause any consequences or display any symptoms. These dormant items can however still be harmful to the system if no one dares to touch them, which might lead to the debt increasing. If a 'I don't want to touch it' mindset is present among the developers, it might be a symptom that ATD exists in the system. Another indicator of a system in debt is performance issues such as scalability difficulties. The most apparent symptom is however recurrent customer issues, i.e recurrent patches to the same object. By noticing and paying attention to the symptoms, a development team can trace them back to one or more ATD items [27].

2.2 Inner Source

In today's software industry Open Source software plays a crucial role. The use of Open Source products has become more mainstream than ever and help organizations to accelerate development and reduce time to market. Over the years the Open Source's ability to deliver high-quality and highly successful software has been recognized in software development. Organizations are eager to know how they can leverage from not only the Open Source products, but also from the principles and practices of open source software development to support and improve their development processes [3].

Inner Source is described as adopting Open Source development principles and practices within the organization's environment. The term "Inner Source" was first introduced by Tim O'Reilly in 2000. At that time O'Reilly worked at a company called CollabNet. One of the company's visions was to bring Open Source's collaborative development style into the software industry [4]. An Inner Source program is the organization's effort to run and maintain Inner Source projects. Organizations may have one or more Inner Source projects and the goal of each project is to develop and maintain Inner Source software. Similar to Open Source, Inner Source projects do not have a fixed duration that is set prior to the project's start and lack a defined end date [3]. There are two main features that characterise Inner Source. The first one is the openness of the development process of the Inner Source projects for the entire developer community within the organization. The second feature implies that the resulting Inner Source products are only available within the organization's boundaries [12].

An open process and an open product are the two main characteristics of an Open Source project. The open process welcomes new contributors and the success of an Open Source project highly depends on the number of developers it attracts. The higher this number is the greater is the odds of success. Inner Source adopts Open Source's open process within the organization but limits the product to the organization's boundaries with no Open Source license. However, the organization may decide over time to Open Source their Inner Source products, a phenomena known as opensourcing [12].

Adoption of Inner Source in organizations

Inner Source is adopted differently by different organizations. Each Inner Source implementation must be tailored to the specific context of an organization, thus there is no recipe on how to adopt Inner Source [12]. The foundation of most Inner Source projects in organizations lays in movements started by individuals, project teams or departments. The person initiating the Inner Source project can acquire the role of a benevolent dictator which is common in Open Source communities and as contributors become experts in the project, they can together with the benevolent dictator form a core team [22].

Despite the variety of ways to adopt Inner Source, two main models have been identified, Infrastructure-based model and project-based model. An organization with an infrastructure-based setting facilitates Inner Source projects, by providing the necessary infrastructure such as code repositories, bulletin board software and mailing list servers. Project teams and individual developers can use a distributed development platform such as GitHub to provide transparent code hosting and facilitate contributions to the Inner Source projects throughout the organization. A project-based model is a more strategic approach to adopt Inner Source and focuses on strategic reuse of an Inner Source project. In this setting a dedicated division (project team or product group) often called the 'core team', takes responsibility for development, maintenance and support of an Inner Source project, referred to as a shared asset. One of the core team's key responsibilities is to provide continuous support to customers of the Inner Source product which consist of other teams and projects in an organization. These two models mostly differ in the level of maintenance and support a certain Inner Source project receives. Organizations may adopt both models when implementing Inner Source [12].

In an Inner Source approach, likewise Open Source, there is a community but within the organization that develops software accessible by different teams throughout the entire organization. The community may consist of members from different teams contributing to the Inner Source product. One Inner Source project might not be used by all teams in an organization and there may exist different Inner Source projects with different purposes. An Inner Source project is usually supervised by a trusted developer and committed changes from maintainers need to be reviewed before reaching the main repository. Building and governing of a community around the Inner Source software and encouraging collaboration within the community is critical and similar to Open Source projects, and can increase the success rates. For the project to be useful it should be kept alive, updated and new releases are encouraged to be frequent [3].

Acquiring top-level management support is very important in order to establish a successful Inner Source project. Management can support an Inner Source project by granting resources if they are requested by the community and also advertise and advocate for an Inner Source project throughout the organization. Since Inner Source projects do not follow a strategic planning roadmap justifying work on such projects can be difficult. This can result in managers believing that investing resources in informal projects such as Inner Source is inefficient and wasteful. However if the Inner Source project creates value early and proves to be of great importance to the organization, management can become supportive of the project [23].

Benefits and challenges with Inner Source

One of the main benefits of Inner Source in organizations is increased software reuse and accelerating the development process and thereby reducing time to market. Software development process becomes faster when there is possibility to reuse already-extant high-quality components that are developed and assembled elsewhere. A characteristic of Inner Source is process standardization. Standardized processes in an organization makes it easier to onboard new hires or transfer employees. Recruitment and onboarding are both easier because the development environment and processes are now much more standardized within the organization [3].

One issue with Inner Source is employees' unawareness of ongoing work in Inner Source projects in an organization. Although Inner Source allows openness and facilitates exchange of the information organization-wide, the developers within an Inner Source program are often unaware of relevant work performed by others. This can result in the Inner Source product to drift apart from its purpose. Another challenge is the cost of maintenance of an Inner Source project, both in capital and time. It is usually the community that owns the project and is responsible for maintaining the developed software component. Each portion of code contributed to an Inner Source software component, adds to the code base that needs to be maintained. One problem that can arise is the owner's fear of accepting contributions to the Inner Source projects due to fear of additional maintenance effort [3].

Key Factors for adopting Inner Source

Several key factors that are worth mentioning for adopting Inner Source are described below [22].

Existence of a seed product - In order to initiate an Inner Source project the presence of a seed product that grants significant value to the organization is essential. The seed product must be runnable to attract users that become contributors and form a community around the Inner Source project. The planning, design and initial implementation of the seed product until a runnable version is available, lays in the hands of the initial project founder. There is no need for requirements and features of the seed product to be fully known. The flexibility of the requirements can attract more contributors from a wider community and they can evolve continuously from organization-wide input.

Multiple stakeholders - The Inner Source project must be required by multiple stakeholders. This allows contributions throughout the organization which establish a vibrant Inner Source project. According to Linus's Law "Given enough mindsets, all ideas are obvious". When the input is coming from different parts of the organization the project can benefit from a wide range of expertise and more optimized solutions emerge as a result.

Modularity - Many Open Source projects possess high modularity architecture, and having a modular architecture is generally desired in Inner Source projects to allow parallel development and decrease merge conflicts. Modularity facilitates integration and hence enhances reuse, thus attracting more potential users to the Inner Source project. Modularity enables developers to better understand the project and contribute more efficiently.

Tools - A set of common and compliant development tools to facilitate contribution to the Inner Source community is a key factor for a successful Inner Source project. The variety of tools that are used across an organization can make it difficult for developers to contribute, or may necessitate duplication of the code repository, resulting in further synchronization efforts. As a result, a set of common tools must be available throughout the organization. Tools commonly used by Open Source software projects are (besides compilers) version control systems, issue tracking software, mailing lists and wikis.

Quality Assurance - In Open Source projects Quality Assurance (QA) practices are key tenets. Adopting suitable QA practices are important to ensure that an Inner Source project has achieved a high level of quality. Three approaches to ensure high quality of the product are reviewing the code, writing reliable test cases and rapid release. Peer review is one of the most well-known practices in Open Source development and is considered as a self selected task by other interested developers. Developers in Open Source projects are more likely to provide genuine feedback given their interest in the success of the project they work on, rather than doing a review because they were told to and possibly having to consider relationships with co-workers when pointing out any issues with their contributions.

Coordination and Leadership - Stol at el. [22] discussed the management of an Inner Source project as a flexible approach allowing the community to thrive. In traditional organizations, leadership and coordination are determined based on seniority status. Meritocracy, however is a key principle in Open Source and also adopted into Inner Source. In an Open Source approach typically a "benevolent dictator" is a person who started the project. Developers who make significant and substantial contributions to the project gain deep experience and therefore become "trusted lieutenants", assisting the "benevolent dictator", managing and coordinating the project.

An Empirical Investigation into the Adoption of Inner Source in IT Companies: A Case Study

An Empirical Investigation into the Adoption of Inner Source in IT Companies: A Case Study is a master thesis, which provided us with knowledge for our thesis. The case study was carried out at Chalmers University of Technology by Niloofar Safavi in 2019. The author investigates the challenges and obstacles of adopting Inner Source in a large IT company. The company owns many products and therefore had to decide what products are the best candidates for inner sourcing. The author believes challenges discovered during this case study could help organizations in taking them into consideration if they choose to opt Inner Source. The author also explains the two main models in Inner Source, project-based model and infrastructure-based model and later discusses the most suitable model for the investigated company. She concluded that choosing a model depends on the type of the product in the company. If it is a company critical product, a project-based model with a core team is necessary, and if the product is less critical an infrastructure-based model is a reasonable approach [21].

Chapter 3
Research Method

This thesis follows the design science paradigm, more specifically the visual abstract template provided by Runeson et al [19], see Figure 3.1. The design science research paradigm presents a way of conducting research, meaning how research problems are formulated and assessed. It is suitable for conducting research on areas where human-made constructs are needed to improve the practice, as in engineering sciences and medical sciences. In software engineering examples of human constructs are tools, developers and processes used by project managers as well as the software companies themselves. The scientific contributions from design science research are formulated with technological rules. These are typically phrased as *To achieve < Effect > in < Context > apply < Intervention >*. Furthermore, design science research may have several technological rules on different abstraction levels, sometimes building a hierarchy of rules.

All technological rules need to be validated, meaning that the proposed intervention needs to be tested in the particular context of the rule, with the desired effect as an outcome. The validation can be done empirically, using different methods for data collection, such as interviews and observations. Moreover, a technological rule must be instantiated prior to being validated, meaning that a specific instance of the more general technological rule is tested. Each tested instance strengthens the validity of the rule. Alternatively the researcher may go the other way around and implement the solution in a real-world context first, and from there abstract out a technological rule.

Besides validation, which is performed when a technological rule is formulated, there are activities for defining the technological rule, namely problem conceptualization and solution design. Design science research does not have to include all activities, some research may only focus on conceptualizing a problem while some others may build on previous works to present a solution design or validate a technological rule. Shortly, the problem conceptualization activity helps the researcher understand the problem in detail. In the solution design activity potential solutions are sought after in previous works and literature. Typically, the problem conceptualization and the solution design go hand in hand. In order to gain a conceptual understanding of the problem and from there suggest a feasible solution, the researcher may use empirical research methods to collect information about the problem and the context.

In mature science fields, practitioners may find their own solutions to their problems. However, in relatively new fields such as software engineering, practitioners and researchers may need to cooperate in order to improve a practice or explore solutions. In these industry-academia collaborations, the research is usually performed as a case study.

The visual abstract template is an application of the design science paradigm in the area of software engineering. The template consists of the core ideas of the design science paradigm, as well as guidelines for contribution assessment. According to the template a research should have a theoretical contribution, phrased as a technological rule. It should be validated in a real-world context and the problem conceptualization and the solution design should have both empirical and theoretical support. For assessment the research should discuss relevance, rigor and novelty. The assessment is provided in Section 5.3.

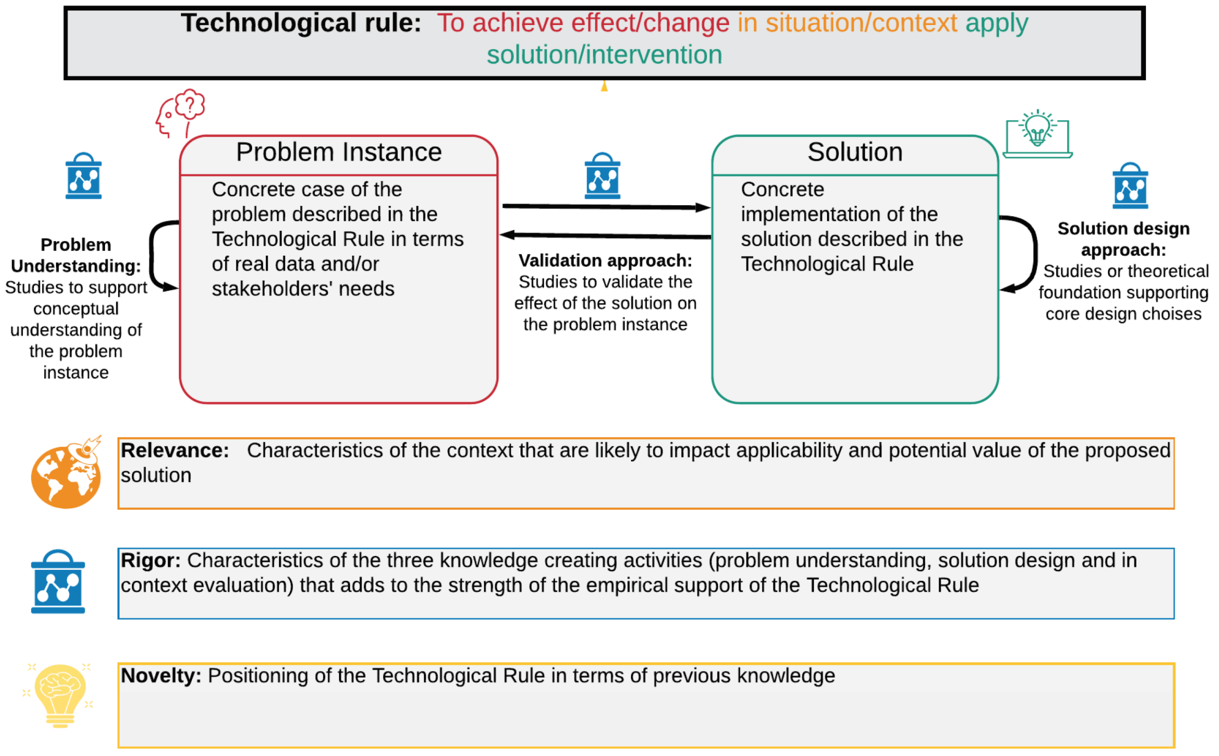

Technological rule: To achieve effect/change in situation/context apply solution/intervention
Problem Instance
Concrete case of the problem described in the Technological Rule in terms of real data and/or stakeholders' needs
Solution
Concrete implementation of the solution described in the Technological Rule
Problem Understanding: Studies to support conceptual understanding of the problem instance
Validation approach: Studies to validate the effect of the solution on the problem instance
Solution design approach: Studies or theoretical foundation supporting core design choises
Relevance: Characteristics of the context that are likely to impact applicability and potential value of the proposed solution
Rigor: Characteristics of the three knowledge creating activities (problem understanding, solution design and in context evaluation) that adds to the strength of the empirical support of the Technological Rule
Novelty: Positioning of the Technological Rule in terms of previous knowledge

Figure 3.1: The figure illustrates the generic visual abstract template including the core constructs in the design science paradigm [19].

Our motivation for using the visual abstract template is to carry out the thesis's work in a more structured and systematic manner. Figure 3.2 provides an overview of the entire thesis and the steps constituting the process.

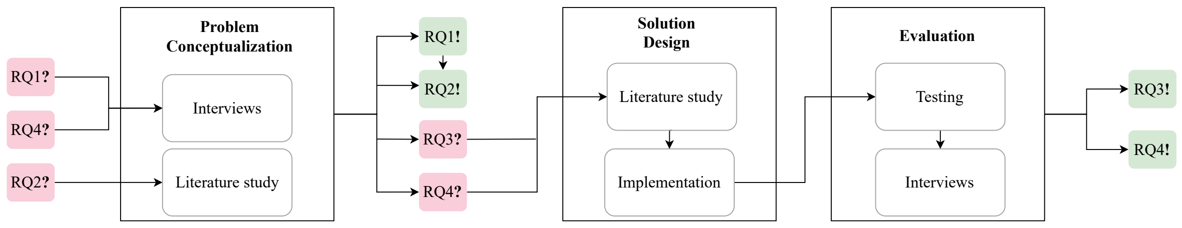


Figure 3.2: This figure illustrates the relationship between the research questions and the thesis process. The red boxes represent the research questions that are yet to be answered and the green boxes are the answered questions. The three large boxes represent the design science activities.

3.1 Problem Conceptualization

The first step in our research was to investigate the problem area and gain a better understanding of the problem. We were initially provided with a brief description of the problem from our supervisors at Jayway. They experienced that the setup of new projects was cumbersome, and wanted to research if it is possible to facilitate the process. In order to confirm their experienced challenges and collect more detailed information we conducted interviews with eleven employees at Jayway.

Data Collection

According to the design science principles the problem conceptualization and the solution design should have both empirical and theoretical support [19]. In this thesis, the theoretical support was collected from a literature study. The empirical support was acquired from conducting semi-structured interviews.

Semi-structured interviews allow the interviewer to have just enough control over the interview process to manage to collect necessary data, but also discover new areas. Whilst there are predefined questions with a strict schedule, there are also open-ended questions which often result in follow-up questions. This type of interview produces richer qualitative data since it allows for exploration and further discussion of the participants' spontaneously raised issues. The flexibility of this method also facilitates the collection of large amounts of in-depth data [20]. Since the aim of the problem conceptualization activity is to explore and deepen the understanding of the problem area, we considered the semi-structured interview to be the most suitable data collection method.

Interviews can be carried out either in groups or one-to-one via various communication channels, email, telephone, text messaging, video calls and face-to-face. In this thesis we conducted one-to-one interviews. One-to-one interviews are often carried out face-to-face. This is advantageous since more information can be obtained when observing the body language and facial expressions of the interviewee in person. Unfortunately these face-to-face interviews have the downside of being costly both in capital and time. Interviews over the phone, email, text or video however are less costly and less time consuming [20]. Due to the pandemic the interviews were mostly carried out over video calls. We considered this communication channel to be the best choice for one-to-one interviews. Firstly we kept social distance, secondly we were able to gain the benefits of seeing the participants, and hence pick up non-verbal cues such as facial expressions and body language. And lastly we saved time and money both for us and the participants.

Moreover, an interview process consists of preparation of the interview, selecting participants, pilot testing, constructing effective questions, implementation and interpretation [25]. For the preparation of the interviews we constructed a consent form giving us permission to record the interviews, which the participants' were asked to fill out prior to the interview (Appendix C).

We performed a criterion sampling strategy for selecting potential participants for the interviews [17]. Our criterion was based on the participant's role. The majority of the participants had a technical role, meaning that they work very closely with the actual source code. Although tech leads, backend, frontend and fullstack developers were our main group of participants, we were interested in hearing other roles' thoughts on certain questions as well, e.g. project managers and CTOs. Therefore we included one project manager, one CTO and one studio lead in our sampling. Table 3.1 presents an overview of the participants' background in more detail.

The interview questions can be categorized into four parts. The questions in the first part were constructed to determined the participants' role and their responsibilities in the company and in the projects. This was mainly for understanding their professional background. Part two consisted of questions that would assist us in understanding the current situation at the case company, more specifically, identifying the challenges faced when setting up new software projects. Part three was relatively technical compared to the other parts. Here, we aimed to learn more about their project's architecture and understand the choices that they had made to this point. We asked them to name the most essential technologies used in their project in order to know what to include in our prototype. This part was omitted in interviews with less technical roles. The last part consisted of questions regarding the potential solution, namely a software project generator. We asked the participants to share their thoughts on using it as well as their thoughts on how it should be maintained. The interview lasted about and hour and the questions can be found in Appendix A. We pilot tested the interview questions prior to the actual interviews. For this matter we turned to our supervisors for assistance, since it is recommended to pilot test the interview on someone who has some knowledge of the research area [20].

Table 3.1: Participants' Demographics

ID	Role	Domain	Project ID
P1	Project Manager	Furniture retailing	PR1
P2	Tech Lead	Furniture retailing	PR1
P3	Tech Lead	Education	PR2
P4	Backend Developer	Construction	PR3
P5	Frontend Developer	Construction	PR3
P6	Fullstack Developer	Telecom	PR4
P7	CTO	–	–
P8	Frontend Developer	E-commerce	PR5
P9	Frontend Developer	E-commerce	PR5
P10	Fullstack Developer	Furniture retailing	PR6
P11	Studio Lead	–	–

Lastly, to analyze the interviews we started by transcribing them, which is the process of documenting the interview recordings in text in order to extract useful information. This was done manually, without utilizing any specific software and by the following steps:

1. We identified the following areas of interests:
    - Web application projects at Jayway
      The data collected in this interest area helped us implement an appropriate prototype for the software project generator. One of the requirements for the software project generator is that the generated project's content is based on common and proven concepts within Jayway. These concepts were identified in this interest area.
    - Current problems with initializing software projects
      This data answered RQ1.
    - Benefits and drawbacks of a software project generator
      The employees' opinions on the software project generator are valuable data. It helped us determine whether there is a demand for a software project generator or not. Additionally, it provided us with interesting points when discussing the usefulness of this generator.
    - Maintenance of a software project generator
      The employees' ideas on maintenance of the software project generator were combined with the theory to determine which strategy is optimal.
2. We both listened to the interview recordings individually and wrote down the interviewees' complete answers under the related area of interest.
3. Each area of interest was then analyzed and summarized.
   Both of us read and analyzed the content under each area of interest individually. Later we discussed our conclusions with each other and documented the joint analysis in Section 4.1.

We complemented the data from the interviews with data from the projects' repositories. The projects' package.json file contains information about external packages used in the projects. Our purpose with the studying of the repositories was to identify these packages.

3.2 Solution Design

During the solution design activity we conducted a literature study. We started by studying the current methods for managing ATD to gather and provide some background of how ATD is dealt with at the moment, see Section 3.2.1. Thereafter we considered different techniques and tools for implementing a prototype of the software project generator, see Section 3.2.2. We also studied numerous literature to learn more about software maintenance methodologies and what alternatives there are for maintaining the software project generator. The different alternatives are presented in Section 3.2.3.

3.2.1 Current ATD Management Strategies

An ATD management strategy describes how ATD is managed within an organization or within a development team. There are a number of strategies, and which strategy is best depends on the system and the company. Furthermore, an ATD management strategy consists of different activities.

Identification, measurement, prioritization, repayment and monitoring of ATD are the activities of an ATD management strategy [14]. In order to make the right decisions about where to invest resources, a prioritization model is referred to. However, to be able to compare refactoring with another activity, for instance feature development, and prioritize one of them, certain metrics need to be available. For example, what is the cost of refactoring? This might seem like a fairly simple question, but it has shown to be quite complex and difficult to estimate.

There are different types of strategies, namely active, reactive and passive ATD management strategies [27]. With an active management strategy the ATD in a system needs to be acknowledged in the first place. After realizing that ATD is present, an active plan is formed. One way to actively manage ATD is to allow for extra resources required to proactively construct a well designed architecture in the budget. This approach is rare due to uncertainty in the pay-off as well as lack of time. ATD can also be managed by systematically allocating time to deal with the repayment. This is the most common approach. Usually 20% - 30% of the development time is spent on refactoring ATD [27]. At rare occasions an entire day per sprint is allocated to repay the debt. Sometimes the presence of ATD might be noticed and acknowledged but still not prioritized. Such reactive strategies delay refactoring for as long as possible, until it is the only way out. Developers are encouraged to keep building on the debt instead of resolving it, and spend just enough resources to work around the problems. In this type of strategy, short-term is the solution. If or when the problems become unavoidable a major refactoring is necessary. While refactoring, other activities that give the company competitive advantage are sacrificed and huge amounts of resources are invested. Lastly, the passive strategy, which is practically just accepting the presence of ATD, and adjusting the development process accordingly, e.g decreasing the development pace. This strategy is adopted when the costs of repaying the debt is considered too high and simply not worth it.

3.2.2 Software Project Generator Prototype

We propose a software project generator to lay a solid foundation for the start of software projects, and thus reduce the risk for ATD. In order to implement a realistic prototype of the software project generator we started by summarizing the fundamental requirements for the software project generator provided by Jayway in their original idea.

Req1. The software project generator should generate a standard project template on demand.

Req2. The generated template should include technologies and tools that are common and proven within Jayway.

Req3. The generated template should include simple boilerplate.

Besides these requirements, we acquired three additional requirements from the interviews with the employees at Jayway. These are listed below.

Req4. The software project generator must be easy to use and understand.

Req5. The software project generator must be configurable by the user.

Req6. The software project generator must be able to run on all employees' computers.

These requirements were motivated as follows. For the software project generator to be preferred over setting up a project manually, it must be easy to use and understand. The learning curve cannot be too steep. Furthermore, for the software project generator to be widely used and become the standard way of setting up new projects, it must run smoothly on all employees' computers. Finally, for the software project generator to be useful in all types of projects it must be configurable so that the user can set the outcome of the generator to fit their project.

For the prototype of the software project generator we focused merely on web application projects. Furthermore, we limited the prototype to only cover the client side of the projects. We were required to narrow down the scope significantly to manage to finish in time. When searching for a method for the implementation of our prototype we used the requirements listed above as a checklist, see Table 3.2 and 3.3.

GitHub Repository Template

An initial idea was to provide a standard software project template via a GitHub repository. GitHub has a relatively new functionality for creating repository templates [10]. With this functionality a repository may be set as a template, which future repositories can use to get the same files and structure. The repository can be filled with files that contain simple boilerplate and a package management file that lists the technologies and tools included. This would satisfy Req2 and Req3. Moreover, the user can simply use GitHub's GUI to generate a repository based on the contents of the repository template, making it easy to use and understand, satisfying Req4. Furthermore, as long as the user has a GitHub account, they can get access to, i.e run, the generator whenever and wherever they want, which meets Req1 and Req6. The only requirement that fails is Req5. It is in theory possible to satisfy this requirement by creating numerous repositories, each with different configuration, that the user can choose from. However, we concluded that this solution is highly impractical and non-optimal.

Table 3.2: GitHub Repository Template

Requirement	Satisfied?
1	✓
2	✓
3	✓
4	✓
5	-
6	✓

CLI or GUI?

Another method for implementation of the software project generator is a script. A script may ask for user input and configure the template to be generated according to the input, hence meeting Req5. Furthermore, a script satisfies Req1, Req2 and Req3. It may meet Req4 if we make efforts to make it more user friendly. Since the script will interact with the user it must have an interface, either a graphical (GUI) one or one that runs in the command line (CLI). Both alternatives have their benefits and drawbacks.

A command line interface (CLI) is solely text-based. Its main function is to receive short text commands from the user and interpret them with the help of a command line interpreter or a so-called shell. When necessary, the CLI will prompt the user for a certain input, which makes it interactive. CLIs are often used to interact with the Operating System (OS), although graphical user interfaces (GUI) are more common nowadays. An example of an OS CLI command is 'ls', which displays the directory system to the user. This, and other commands are more often called from a GUI provided by the OS. Besides operating systems, applications can also use CLIs for their interaction with the user. The syntax for Linux, MacOS (GNU) and Windows OS CLI commands are all similar, it often looks like this *command arg1 arg2 arg3 … argx*, sometimes preceded by a prompt [24, 16]. Moreover, CLIs are often used in applications that generate coding environments or boilerplate. For example, a skeleton for React applications can be set up with a CLI called *create-react-app* and Vue applications are initiated with Vue's built-in CLI [8, 28]. CLIs often require less resources than GUIs, e.g graphic performance, memory etc. making CLIs faster than GUIs. This also allow CLIs to run on almost any machine since the performance requirements are lower. Additionally, the simple interface allows the user to perform repetitive tasks quickly and smoothly. There are numerous advantages with using the CLI instead of the GUI [26], however there is a reason why GUIs are preferred by most users. GUIs are often more user-friendly and easier to understand for the typical computer user. Graphical illustrations are together with the computer mouse very powerful in making tasks on the computer easier.

As mentioned, a CLI requires less resources in terms of performance and graphics compared to a GUI. This allows our prototype to run faster and on almost all computers, satisfying Req6. Regarding Req4, a GUI seems like the better choice. However, implementing a GUI would require too much time and effort on features outside of the project's scope. A CLI does not require the same amount of resources, and can be implemented in a way that makes it more user-friendly. Moreover, seeing that similar services, e.g Vue-cli, are provided via a CLI, we supposed that it would feel more natural and intuitive for developers to understand, adopt and use our prototype if it too was a CLI. In other words, if the prototype resembles similar services, the users may find the prototype easier to use and understand. Lastly, since this generator's target users are developers with high knowledge in computers, a CLI should not be too difficult to use if implemented correctly.

Table 3.3: CLI vs GUI

Requirement	Satisfied?	Requirement	Satisfied?
1	✓	1	✓
2	✓	2	✓
3	✓	3	✓
4	✓	4	✓
5	✓	5	✓
6	✓	6	-

3.2.3 Choice of Maintenance Strategy

Maintenance of an internal tool or software is essential. There are several strategies for maintaining an internal asset, Open Source and Inner Source. Open Source is a popular option where the asset is public and developed by a community outside the borders of the organization. Inner Source is the proprietary alternative to Open Source, where the asset is private and the community is within the organization, as presented in Section 2.2. These strategies have their benefits and challenges. In an Open Source project, the asset can grow successfully without extra resources, however it requires a large number of contributors. The first step towards openness might be hard. Therefore, we believe that the Open Source strategy might be easier to adopt for the project generator if Jayway is already familiar with hosting Open Source projects or contributing to other Open Source projects. Adopting Inner Source has its challenges as well. However, the upside is that it keeps the asset proprietary, which is suitable for assets with sensitive content. As the software project generator will include common practices at Jayway and become the standard method for creating new projects, this information might be secret. Moreover, since the software project generator is highly tailored to Jayway, it might not even be beneficial with contributions from outside. With this in mind it seems like Inner Source would be the better option, however there are concerns. For example, keeping an Inner Source project alive requires resources and a general interest among the employees. If the project generator does not attract enough interest it will eventually become too outdated to be useful, since no one is contributing. Additionally, if the management does not see the value of the project generator, maintenance will not be financed, which makes it difficult for employees to maintain the project.

Which strategy to pursue depends on the organization in question and the asset. In order to determine the best suitable maintenance strategy, we conducted literature studies about different software maintenance methodologies. Thereafter, we studied the current structures at Jayway. We asked employees about their thoughts on our idea of a software project generator and discussed potential ways to maintain it. Thereby, the final suggested strategy was based on literature studies, the answers from the interviews and from our own observations at Jayway.

3.3 Evaluation

An important activity in the design science research paradigm is evaluation. It refers to the validation of the technological rules presented in the research and hence to assess whether the solution design addresses and solves the problems identified during the problem conceptualization. In this thesis, we present three technological rules, see the results in Section 4. The first one concerns the effectiveness of using a software project generator to reduce ATD. The second rule suggests a method for the implementation of the software project generator, and lastly, the third rule proposes a solution for maintaining the software project generator. Technological rule two was validated with a practical testing of the prototype, and technological rule one and three were validated with interview questions.

Prototype Testing Session

We selected seven individuals with a software developer role among the employees at Jayway for our testing process, see Table 3.4 for more details about the participants. Since our prototype concerns the client side in web application projects, the participants we selected were either working as web developers or have had work experience in this area. The level of expertise differed between the participants; two seniors, two mid-seniors and three juniors, some of whom we also interviewed during the problem conceptualization phase. Every test session lasted for approximately half an hour. At the beginning of each session, we presented a short description of the thesis's context to the tester and explained the different steps of the testing process. We then asked the tester to follow a scenario and think aloud when performing the actual tasks. The scenario was to set up the frontend of a web application with the React framework using our prototype. After setting up the project we required the participant to explore the generated project and lastly start the application.

Table 3.4: Participants' Demographics

ID	Role	Level of Expertise
P1	Tech Lead	Senior
P2	Backend Developer	Mid-Senior
P3	Frontend Developer	Junior
P4	Frontend Developer	Junior
P5	Fullstack Developer	Senior
P6	Fullstack Developer	Mid-Senior
P7	Fullstack Developer	Junior

Interviews

Since the number of participants for the testing process was small it was more convenient to gather qualitative data. Therefore, in each testing session, we conducted a semi-structured interview with each participant after all the testing tasks were performed. The qualitative data from the interviews provided a deeper understanding which outweighed a quantitative data collection method with a large number of participants. The interview process was similar to the interviews during the problem conceptualization phase (Section 3.1), meaning one-to-one interviews over video calls which were recorded. We also asked the participants to share their screen while testing the prototype in order to better observe their behavior when using it.

The interview lasted between 15 to 20 minutes. It started with a number of questions that regarded the prototype directly, such as ease of use. The data from these questions were used to assess the chosen method for the software project generator with regard to its requirements, see Section 3.2.2. Later the participants' were asked to share their thoughts on using a software project generator to mitigate the challenges related to ATD. Lastly, we asked their opinions about the maintenance of the generator. The questions are presented in Appendix B.

The interviews in this phase were much shorter in comparison to the problem conceptualization activity's interviews and we used verbatim transcription following steps below:

- We wrote the the complete answers to every question.
- If a portion of a participant's answer to a particular question suited another question's answer more, we moved it to the latter question instead.

To summarize this sections, we carried out the three activities of the design science research paradigm. Firstly, the problem conceptualization activity, where we conducted interviews and literature studies to answer RQ1 and RQ2. Thereafter, we moved on to the solution design, where we studied current methods for managing ATD, implementing software generators and maintaining software in order to formulate our technological rules. The technological rules were later validated in the evaluation activity. After these activities we were able to answer RQ3 and RQ4. The results of each activity are available in the next chapter.

Chapter 4
Results

The results from the design science activities are described below. All three activities contributed to answer the research questions in this thesis. To present the results as clearly as possible, the results for each activity are discussed individually under respective sections.

4.1 Problem Conceptualization

The problem conceptualization activity consisted of interviews and studying of project repositories, as described in Section 3.1. The analysis of the interviews resulted in a number of interest areas. These areas are:

1. Web application projects at Jayway
2. Current problems with initializing software projects
3. Benefits and drawbacks of a software project generator
4. Maintenance of a software project generator

4.1.1 Web application projects at Jayway

The results have shown that many projects at Jayway have moved towards cloud solutions. They are hosted on various large cloud services such as Azure, AWS and GCP, commonly in Kubernetes clusters and Docker containers. Furthermore, microservices seems to be the architecture of choice, over the more traditional monolithic architecture. It also appears that many projects inherit an architecture or parts from a system from their clients to either extend the system or rebuild and modernize it. Lastly, many projects seem to be built to be maintained by the customers, therefore certain aspects of compatibility with the clients current system, knowledge and competence of the clients need to be considered.

Node.js and React.js have shown to be the most common tech stack at Jayway, although other technologies such as Java Spring boot and Vue are used occasionally. In most cases Express.js seems to be the preferred framework for http handling. In our sampling, all projects used React.js for the client side, except one project which used both React.js and Vue.js. States appeared to be handled with Redux.js. Furthermore, we found that styled-components.js is utilized in many of the projects, as well as Husky.js. Other libraries that were mentioned several times are Eslint.js and Prettier.js. See Table 4.1 for more details.

Table 4.1: Common technologies in web application projects at Jayway

Technology	Usage
Node.js	Javascipt runtime environment
React.js	Library for frontend implementation
Express.js	Framework for http handling
Redux.js	Library for managing application state
Husky.js	A library that can be used to define Git hooks. Git hooks may be used to run a script before performing a git action, such as pull, push or commit. A common use is making sure that the local code is formatted according to the configurations prior to a commit. This type of hook is called a pre-commit hook.
Eslint.js	Eslint is a linting tool used for catching errors and forcing rules for different actions. There are several templates for the eslint configuration that people prefer to use instead of defining their own rules, for example airbnb, standard and google template.
Prettier.js	Prettier is a package used for code formatting. It makes sure that the appearance of the code follows the configuration.
StyledComponents	Library for React that allows component-level styling, i.e mixing CSS and JS

4.1.2 Current problems with initializing software projects

In the interviews the participants were asked to describe their daily work and tell us about their experience with setting up new software projects. They were also asked to inform us about the challenges they experienced while initializing these projects. The answers were all very similar. The challenges that we found from our analysis of the interview in the problem conceptualization phase are; understanding requirements, choosing technologies, time pressure, code legacy, including documentation and code analysis tools and finally, the use of Proof Of Concepts (POCs) in the final product.

1. Understanding requirements

All participants, regardless of their role and years in business, stated that they find setting up a new project challenging and time consuming. Unsurprisingly, different participants with varying degrees of experience and different roles found different things challenging. However, the majority of the participants argued that the difficulties mostly lie in understanding and interpreting the customers' requirements rather than understanding the technology. One of the participants (P8) said *"learning how to write for example typescript takes a few hours of googling, the real challenge is to talk to the customers and understand their ecosystem and produce software that fits into it"* . If the requirements are vague, it is difficult as a developer to make the right decisions regarding the architecture. For example, how scalable is the system required to be, does is have to support extensions, where is the software supposed to be run? There are numerous decisions that rely on the requirements.

2. Choosing technologies
Many of our participants found it difficult to make certain decisions about the architecture and the content of a project due to ambiguous requirements. The participants expressed that they find it easier to communicate with a technical customer due to speaking *"the same language"* as the developers. However, a technical customer often has stricter and more specific technical requirements. In situations like these, the challenge is to build software that is compatible with the customer's current system. The software has to be part of an infrastructure, therefore certain technologies must be used.

In other cases where the customer is non-technical, the development team has greater responsibility and authority in deciding what technologies to utilize. Deciding what framework or language to use is vital for the entire development. Grounded decisions about the software's construction need to be taken with respect to the requirements as well as the team's ability and competence. Additionally, numerous developers highlighted the importance of investigating the technology thoroughly prior to introducing it to the software. However, a non-technical customer has its drawback as well. They might lack the ability and the competence to correctly convey their requirements. One of the interview participants (P4) had experienced this in their project, and gave us the following example *"a customer might ask for a service similar to Facebook, when they actually just want to have a messaging service"*. This example exposes the potential dangers of communicating with a non-technical customer.

Finally we asked the interviewees what they grounded their selection of technologies on and all of them answered that they always try to consider what the team's abilities are. If all members of a team are more familiar with coding in React, it would not make sense to use Vue for instance. They also emphasized that the selections must satisfy the customers and their needs.

Choosing technologies is a dangerous activity from an ATD perspective. Introducing inappropriate technologies can become costly in the future. Therefore the technologies should be chosen with care and consideration.

3. Time pressure
All participants stated that they do experience time pressure, and that it from time to time causes the team to make rushed decisions. One of the participants (P8) phrased it like this *"you don't have all the time in the world to prevent something that might or might not happen in the future"*. Shortage of time is very common in software projects, and is often caused by the customers. It is also the largest source to ATD.

4. Code legacy
A number of participants stated that they struggle with adopting and understanding source code left behind by former employees. Especially, one interviewee reported that his team avoid mainte-

nance of some parts of their system. The person responsible for producing and maintaining those parts has left the project and the remaining team members lack the competence to alter the code as they wish. This challenge is more of a symptom of ATD than a cause. An "I don't want to touch it" mentality is usually present in systems with ATD. Another participant advised every team to keep at least one person from the original team, as this may help avoid situations like the one above.

5. Documentation and code analysis tools
Many interview participants stated that documentation and code analysis could be deprioritized from time to time, or perhaps even forgotten. This came as a surprise, as many of the participants expressed the paramount importance of including these practices in their projects. One benefit of documenting the code is that it helps convey the original idea of the code and its architecture to the coworkers. Thereby, it can reduce the amount of misunderstandings, and minimize the number of wrong decisions regarding the architecture. Wrong decisions may lead to ATD, therefore, reducing misconceptions and unsuitable decisions is one way to reduce ATD. Code analysis tools, such as linting and formatting tools may be useful in reducing other kinds of TD, for example code TD, since it concerns the code quality itself.

6. POC for the final product
Proof of Concepts, POCs, are common in the world of software development. Potential customers are often interested in investigating if an idea or a concept is possible to accomplish, which is often proven with a POC from the company's side. A POC is a prototype for the actual product. More precisely it may include a set of main features and an overview of the architecture and the infrastructure. It is supposed to present a solution for a problem quickly, and hopefully get the customer to invest in the actual product's development. Typically POCs are supposed to be discarded as soon as the real development starts. This is due to the fact that POCs often are developed with less resources and care, with intention. However, POCs are apparently kept in the actual products at Jayway. This is a problem, and the employees are aware of it. One of the participants (P11) compared this to a house construction *"You would never build the foundation for a house carelessly. This should apply to software as well. POCs are built to be temporary, therefore, make sure to throw them away!"* . Building a bad foundation for the projects immediately introduces ATD, hence it is a bad practice and should be avoided.

4.1.3 Benefits and drawbacks of a software project generator

One of the goals of having a software project generator tool is to introduce standardization to some degree into projects of similar nature. According to the answers from the interviewees there are different opinions of how valuable such a tool can be for a consulting company such as Jayway. From the interviews we can conclude that there are both benefits and drawbacks for standardization of a project template using a project generator.

One participant stated *"In a perfect world, you would like to have the same boilerplate and use it for every project, but it is difficult to achieve in reality"*. In general it is difficult to standardize things for various reasons. There are two scenarios for deciding the building blocks of a software project in terms of technologies. In the first scenario, the customers are technical themselves, and they have strong opinions on what technologies to include in their projects. The reason behind this is that the final product is often taken over by the customer and integrated and should function correctly together with the rest of the organization's software ecosystem. In this situation, the development teams are limited and obligated to fulfil the customers' requests. The second scenario is that the customers have limited knowledge in IT and leave most decisions to the development team. In this context, the choices of technologies are influenced by the team members' experiences from previous projects and competencies that are available among them. If the first scenario is true, then a standardized project generator from Jayway's side would be needless and unnecessary since the customer's needs may be in great contrast with what the project generator tool has to offer. However, in the latter scenario developers could benefit from a tool that can facilitate the developers' work on deciding what packages to include in the project.

Developers can be categorized into two groups concerning their level of expertise, seniors and juniors. It was easy to see a pattern of who was positive and who was negative towards the project generator tool. The majority of junior developers were positive and could see the benefits of the generator. However, the more senior developers were sceptical about how useful it really is. With years of experience in software development, seniors have been involved in many projects and are quite familiar with the project setup process. They often have their own routines for initiating projects that they are comfortable with. Hence, it would be challenging to convince seniors of using a project generator and they would not appreciate it as much. Juniors, however have very little experience on how new projects are set up and what packages are necessary to include in order to develop a high quality product. A project generator tool would therefore help juniors to get started with a new project faster. The participants who could see the benefits thought that the tool would serve as a template for less experienced developers and encourage them to use best practices and proven concepts. It would also streamline and automate repetitive tasks and speed up different configuration set up that otherwise would take time if performed manually. Setting these two groups aside, for a project generator to be efficient it still needs to accelerate the starting phase of the project and not add extra work for the employees or the purpose of the tool is lost. Considering that the project generator is fast, easy to use and up-to-date, bootstrapping a new project in this way becomes very beneficial especially when testing a new concept by creating a POC. By accelerating the testing and experimenting phase more time can be spent on developing the actual product.

In a consultancy company with various types of projects, every project is tailored to meet specific requirements and consist of technologies that are particular to that project. Requirements may also change continuously. One issue with a generator tool is that it can be difficult to adopt, Hence, such a tool should be implemented in a way that it is configurable and customizable and only includes basic and most common packages that are often used in most projects.

Some participants stated that in a company such as Jayway which embraces innovation and encourages its employees to be updated and learn new technologies, having standardization can be harmful. Some participants argued that standardization would limit the creativity of the employees when it comes to choosing technologies and solving problems. Since decisions are made for them new possibilities are unlikely to be explored. One participant mentioned that *"There always has to be room left for innovation"*.

In the initial phase of every project, it is very difficult to determine exactly what frameworks, packages and libraries to included. So it is important that the tool enables the user to choose exactly what to include. The best approach is to start little and add things iteratively as the need for them becomes relevant. When developers use a project generator tool there is a risk for including resources in the beginning, hoping to utilize them in the future. This might result in the appearance of resources in the project that are never used. Existence of unnecessary and unused resources in a project can cause confusion for developers especially for new members who have not been involved in the project from the start. These unused resources might also introduce unnecessary complication into the project. Moreover, developers may refuse to remove them due to lack of knowledge about these resources and fear of breaking something in the program.

4.1.4 Maintenance of a software project generator

The ways software is developed are changing at an ever-increasing pace. New technologies, frameworks, packages etc., are developed for different purposes and for tackling various problems. A valuable project generator tool should be maintained and updated to keep up with new trends and technologies or it becomes outdated, less accurate, casted aside and never used again. The interviewees had different suggestions on how such a tool can be maintained.

Over the years, Jayway has driven various in-house projects. One possibility is that the generator tool is treated as an in-house project. When team members leave a project or when an entire team completes a project, they sometimes have to wait for a short time until they are assigned to a new project. Interviewees stated that employees often maintain in-house projects during this period. Thus, they suggested that the generator tool can be maintained in this manner as well. There are in-house projects that have resulted in business critical products and they are used by employees every day. These projects are maintained regularly, new features are developed and bugs are detected and fixed frequently. The conclusion here is that in-house projects that live long in the company are highly coveted and regularly used by people in the organization. Resources are allocated for in-house products if they are of great value and have major benefits for the employees.

Although there are examples of successful in-house projects at Jayway, it is worth mentioning as one participant stated *"In a consulting company, the primary goal is to sell consultants, so in-house projects are less prioritized"*. With that said, it would be challenging to allocate a large amount of time and resources to maintain a product, even if it would improve the overall development process in the company.

Since juniors have been recognized as the primary group that would benefit the most from the project generator tool, they are likely to be the candidates for maintaining the product as well. Although the term Inner Source was never brought up by the interviewees, the process of developing and maintaining the in-house projects at Jayway can be associated with Inner Source practices. Many mentioned that the success of the former in-house projects at Jayway lied in the fact that some passionate individuals had taken ownership of the product and were responsible for its maintenance. The product was also used either by the owner itself and/or by others in the company. According to the interviewees there are two main reasons for termination of in-house projects. The first reason is that the owner of the project leaves the company and no other person takes ownership of the project. Another reason is that the product does not bring major value to the company and there is nearly no demand for it from the employees. Thus ownership and multiple stakeholders are critical for in-house projects' survival.

Jayway has had experience with Open Source software development and has driven several Open Source projects. One way of maintaining such a tool for long-term goals and adoption is to make it Open Source, and Jayway can act as a sponsor and contribute with time, resources and help marketing for the product. However, there are some factors to take into consideration. An Open Source software's success highly depends on the number of contributors and users, which the interviewees mentioned as well. If the numbers are high in both groups, it enhances the opportunities for improvements and developing new features in the software and increases the probability of detecting bugs. Thus, resulting in a high quality product.

4.2 Solution Design

In this section we provide the results from the solution design. During the solution design three technological rules were defined as follows:

TR1. In order to mitigate the causes to ATD, use a software project generator to setup standard projects.

TR2. For the implementation of such a software project generator, use a CLI design.

TR3. To maintain such a software project generator at Jayway, maintain it as an Inner Source project.

4.2.1 In order to mitigate the causes to ATD, use a software project generator to setup standard projects

The majority of challenges experienced by employees at Jayway have shown to be related to architectural technical debt. As mentioned in Section 2.1, three factors that may cause ATD in software projects are uncertain requirements, time pressure and human factors. However, it may also be caused by POCs that got stuck, which is described in more detail in the following page. The result from the problem conceptualization shows that all participants struggle more or less with the requirements in their projects. Furthermore, most interview participants reported that they do feel time pressure to some degree and that some decisions are compromised due to this. Additionally, there are people of different skill levels at Jayway with varying experience. Although the ambition level seems generally high, skills vary naturally in all workplaces. Therefore problems may arise as a consequence of human factors. Finally, POCs that get stuck are a major source of debt, and as discovered in the interviews, POCs tend to get stuck at Jayway.

Our investigation shows that sources of debt are highly present at Jayway. Not only are the sources present but also the symptoms. Two symptoms of a system in debt are difficulties in adding new features and avoidance of certain parts of a system. Both of these symptoms have been mentioned recurring times during the interviews. We have hence concluded that projects at Jayway struggle with architectural technical debt. The causes that we have found at Jayway are:

- Uncertain requirements

- Time pressure
- POCs that got stuck
- Human Factors

Uncertain requirements and time pressure
Ambiguous requirements can be clarified through better communication with the customers, however misunderstandings and confusion will probably always exist to some extent. One way to minimize the risk of large conflicts in the future is to build a foundation that supports quick alterations, or a foundation that takes future requirements into consideration. This is a challenging task and it might require large quantities of time and resources.

A software project generator would not help mitigate this cause on its own. Instead, we believe that it might assist in securing and streamlining less complex and repetitive tasks. Such as setting up an environment for the development or choosing frameworks for the base, and thereby leave more room for heavier tasks, for example designing a well-structured architecture that satisfies the current requirements and has potential to support the requirements in the future. We also believe that execution of these less complex tasks can be quality secured by following proven concepts within and outside of the organization.

A project template generator would include technologies that have proven to work for other similar projects, therefore making sure that all projects that use the generator hold the same standard from the beginning. Furthermore, we have found that the majority of projects at Jayway have similar bases, allowing standardization to some extent and the usage of proven concepts. We also believe that the project generator might help streamline the project setup process. Manually including configuration for testing, linting, formatting and documentation is error prone and time consuming. Using a software project generator would both make it less error prone and require less time since it is automated.

POC that got stuck
Another cause of architectural technical debt is POCs that stick along during the development. They are built to be thrown away as soon as the point is proven, however, many projects at Jayway and at other companies seem to have difficulties throwing it away. In most cases the customers are the ones who want to keep it, since they already spent quite a lot of money and time on financing and waiting for it. Occasionally, developers *"fall in love" - P8* with their code since they have put effort on getting it right, making it hard to get rid of or change it. Our idea is to make it easier to discard POCs. By using our project template generator, POCs may be initialized more quickly, requiring less money and time from the customers and requiring less time and effort from the developers, hence making it less intimidating to throw it away.

Human factors
ATD may arise from problems caused by various human factors. These problems can be avoided if the human factors are reduced. Human factors include lack of experience, competence or ambition. As mentioned previously, a project template generator would consist of proven concepts regarding technologies and other practices such as testing, code analysis and documentation. A number of things have to be taken into account when deciding what technologies to include in a project. Not all employees have the skill set to make a good decision regarding the content and structure of a

project. With a predetermined set of technologies, less decisions have to be made, leading to less unsustainable decisions caused by inexperienced employees. We also argue that these junior developers will find our project generator reassuring and give them more confidence in setting up new projects.

Other challenges at Jayway, less related to ATD, may also be mitigated with a software project generator. For example:

Documentation, code analysis and testing
ATD is only one of many categories of technical debt. Oftentimes people refer to source code debt when talking about technical debt. Source code TD is caused by poorly written code. There are a number of code analysis tools including code linters and formatters that help improve the source code. General rules for the appearance of code is often formulated in and followed by formatting tools. Examples of such rules are single/double quotation, tabs/spaces and with/without trailing commas. Code linters are more complex and may be used to catch errors and sometimes heavier formatting mistakes.

By providing these code analysis tools in our generated projects the overall code quality is increased and the code becomes more readable and easier to understand due to everyone following the same coding style. Additionally, pre-commit hooks may be included which allows automatic analysis of the code for errors prior to committing the code to the shared repository. This prevents poorly written code from being spread across the project.

Apart from practices mentioned above, testing and documentation are vital parts of the code. Interviewees mentioned that testing and documentation could be prioritized lower than feature development, and sometimes even forgotten altogether. This is worrying. Our project template generator will make sure that tools for testing and documentation are provided from the very beginning.

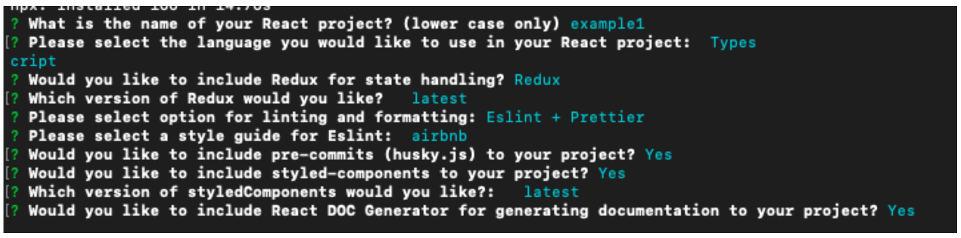
Competence sharing
Several interview participants stated that it takes time to adjust to and get acquainted with a new project. Especially if the tech stack is unfamiliar or has legacy code from former employees. We propose our project template generator as a solution to this problem. If all projects are initiated with the same set of technologies and practices, employees will not feel as new to a project. We believe that it will become easier to rotate project members. We also believe that asking for help from outside the project will be simplified, hence increasing the competence sharing.

4.2.2 For the implementation of such a software project generator, use a CLI design

Based on the analysis in Section 3.2.2, we decided to implement a command line interface (CLI) for the software project generator prototype. It meets all the requirements for the software project generator and the prototype, under the condition that we implement it correctly, make it user-friendly and provide usage documentation.

For the actual implementation of the CLI it is strongly recommended to use a command-line argument parsing library; therefore we decided to use Oclif.js. Oclif.js is an open source framework for CLI development under the MIT Licence, but it was formerly developed by Heroku. When generating the CLI project with Oclif, we were asked to decide whether to write the CLI in Javascript or the static typed version Typescript. We chose Typescript. Furthermore, we had to choose which type of CLI to generate. There were two alternatives, single-command and multi-command. Single-command CLIs contain only one command such as ls, while multi-command CLIs such as git contain multiple commands, e.g git pull, git push, git commit etc. Despite only focusing on generating the client side for a web application project for the prototype, we still wanted the CLI to be reused to generate for example a server or other types of applications. With this in mind we chose to implement a multi-command CLI, where each type of project generator is one command [11].

Moreover, in order to generate a software project template the CLI must interact with the user as well as the shell. The communication with the user is achieved via a number of command line prompts, see Figure 4.1, while the contact with the shell is achieved with child processes. We followed the common programming standard for the communication between parent- and child processes, meaning that a parent process may start one or more child processes in the background, which in turn typically return a zero exit code to their parent upon successful completion, and a non-zero code for failure. Outputs and error messages are sent to stdout and stderr respectively. The idea is to map the user's input to a certain action, and then execute the action's commands in one or more child processes. An example of an action is installing a dependency or creating/deleting a file.



```
? What is the name of your React project? (lower case only) example1
[? Please select the language you would like to use in your React project:  Types
cript
? Would you like to include Redux for state handling? Redux
[? Which version of Redux would you like?   latest
? Please select option for linting and formatting: Eslint + Prettier
? Please select a style guide for Eslint:  airbnb
[? Would you like to include pre-commits (husky.js) to your project? Yes
[? Would you like to include styled-components to your project? Yes
[? Which version of styledComponents would you like?:   latest
[? Would you like to include React DOC Generator for generating documentation to your project? Yes
```

Figure 4.1: Screenshot of the CLI.

Regarding the project to be generated, its content is based on the technologies and tools found in the web application projects at Jayway, see Table 4.1 on page 28. Furthermore, React has its own CLI called create-react-app that is very popular. It includes webpack configuration, boilerplate for different purposes among other things. We used create-react-app as a foundation. The boilerplate was then adjusted to fit the projects at Jayway. For example, more specific code analysis configuration has been added in Prettier and Eslint. Another feature of the CLI is the possibility to add Git hooks with Husky.js. Furthermore, we added Redux for handling states. Here, the user gets the option to use vanilla redux or a tool called redux-toolkit. For the styling of the project, the CLI provides two options. Either the project is styled with classic css or with the help of styled-components.js, which lets the developer create components with the styling included. Lastly, some employees experienced that documentation was forgotten, which is why we have included a tool for auto generating documentation for React projects called React DOC generator. All these features are optional. A framework for testing is automatically provided with Jest.js.

There are a number of things that we did to make the prototype easy to use and understand. First of all, we provided a user manual for the prototype. Secondly, we designed the CLI to remind as much of a real-life conversation as possible, containing questions and status messages. Apparently, CLIs are experienced as more intuitive if it resembles a conversation between people [18]. We also used different colors that convey different messages to the users. For instance, the color green to communicate success and the color red to indicate that an error has occurred. Moreover, symbols were used to clarify things, for instance a spinner for indicating that something is loading. At any time, the user may get help and instructions via the -h or the –help flag.

4.2.3 To maintain such a software project generator at Jayway, maintain it as an Inner Source project

The success of many Open Source projects has led to emergence of Inner Source. We decided that the most suitable strategy to maintain in-house projects even in a consultant company is Inner Source. Inner Source is a well established phenomena to support software development processes in companies. As mentioned in the theory, it can be divided into two models, infrastructure-based and project-based. According to our findings adoption of both models would be a suitable approach at Jayway. In an infrastructure-based model, an Inner Source project receives support in terms of resources such as time, tools, developers etc. from management which employees at Jayway also desired so that the product is updated consistently. One advantage of this is that even if employees lack motivation to maintain the product, the management which is in charge, always can allocate people to the Inner Source project. Even though support from management is essential there still need to be some sort of ownership and community that support the development and maintenance of the asset. This is possible with a project-based approach. We believe that there already exist structures for it at Jayway. There are communities for competence development at Jayway for different topics such as Data Engineering and Web Development. People join a community and share ideas, knowledge and their expertise in that particular field. The Web Development community can mainly be a suitable candidate for maintaining a project generator tool that set up a foundation for web development projects. Furthermore, there are always people who have not yet been assigned to external projects, which can be suitable candidates to maintain the tool. These result in input from people from different teams and different backgrounds, providing a wide range of knowledge to the Inner Source asset.

There are a number of key factors to consider in order to establish a successful Inner Source asset. As for the coordination and leadership, the community that composes the core team should consist of people who are interested in developing the product further. Those with most contributions become the trusted lieutenants since they know the system best. They review other contributors' contributions and protect the seed product from bad commits. Existence of a seed product is crucial for motivating employees to join the Inner Source project development process. Using a common set of tools facilitates contributions from the Inner Source community and therefore is important for a successful Inner Source project. Since multiple people would maintain the asset simultaneously the modularity of the shared resource is an important factor. Last but not least multiple stakeholders affect the quality of the asset, the more people using the product the more likely it is to discover bugs and/or improve the product for better performance. As long as the product is of high demand and there are many employees benefiting from it, it is more likely that it receives support from the management which increases the chances of success.

4.3 Evaluation

As a final activity we conducted test sessions followed by interviews to validate our technological rules. The result from the evaluation phase is presented below.

4.3.1 For the implementation of such a software project generator, use a CLI design

All participants agreed that the project generator tool was easy to use with clear instructions. Participants who have had previous experience with other CLIs, noticed close resemblance between those and our prototype and thus found it familiar. The project generator presents users with options regarding what features/packages they can include into their projects, which was greatly appreciated by the testers. Another positive aspect of the generator according to the testers was that it did not introduce many dependencies and the generated project was quite clean with the necessary initial configurations in place. Since it is difficult to determine all the essential dependencies early in a project's life cycle, it is important to start small and add more if required.

4.3.2 In order to mitigate the causes to ATD, use a software project generator to setup standard projects

Time pressure
The majority of testers experienced the project setup process with our software project generator as fast. Since junior developers have limited experience in setting up new projects, our project generator's speed and ease of use were much more appreciated and well-received. Additionally, they believed that such a tool can facilitate and accelerate the setup process of POCs. Starting new projects from scratch can be cumbersome and time consuming, which can be accelerated with the tool. One tester expressed that currently the company lacks any particular boilerplate and its presence can have a huge impact on accelerating the development process. Some testers, mainly seniors and some experienced juniors expressed that the tool would save them some hours when initializing new projects but not days. However, they mentioned that it could save a lot of time for inexperienced employees.

Mistakes depending on human factors
Including static code analyser tools in projects can impose programming rules that can mitigate or even hinder errors, bugs, stylistic errors and suspicious constructs. These types of mistakes are often caused by humans and can be eliminated by using the project generator since it offers the opportunity to include these tools in the project from the beginning. There are numerous other issues that may arise in a project such as realising the wrong choice of technology and/or detecting a flawed architecture and the project generator is not a solution for these types of human errors. However the questions and options in the project generator can serve as a reminder that certain technologies and tools are essential for projects and must be included, preferably from the start. Lastly, the participants agreed that it is very beneficial to have a foundation to build new projects on, and that having the proper configurations in place can minimize technical as well as communication errors in initial phases.

Code quality
If options such as linting and code formatting are included when setting up a new project, they can increase the code quality by eliminating stylistic errors and preventing bugs to some degree. However, if static code analyser tools are introduced late into a project's life cycle the consequences can be catastrophic. Many errors can emerge in the project which might take several weeks to fix. Initializing a new project with the project generator can configure these tools early into the project, and the problem mentioned above can therefore be avoided. Some participants believed that there are numerous factors that constitute high code quality in a project. One of them is the structure of the project. A project can have a perfect and solid structure at the beginning but suffer later due to negligence of rules to retain its structure and high code quality.

Best practices
One of the best practices according to testers was the option for including different linting features when initializing new projects with the project generator. Strict linting options like airbnb, which is a popular linting configuration in web development, assess developers to find issues in their code before it is too late. Furthermore, the user has the option of choosing Typescript, which is a strict syntactical superset of JavaScript with optional static typing. It prevents unexpected runtime errors and is considered good practice in web application development. Another advantage of the project generator which is considered good practice, is the its ability to create projects with tested and stable versions of different packages/features. Latest versions are not always the best alternatives since they might be erroneous and contain bugs. Some participants believe that there are numerous other factors that constitute best practices in software development. Since the structure of code, which plays a key role in delivering a high-quality product, will change with time, best practices must be followed consistently over the course of a project's life cycle. A good foundation at the start does not guarantee a good product at the end.

Competence sharing
The majority of participants agreed that a project generator tool incorporates some level of standardization into projects with similar tech stack. Currently Jayway lacks standardization. There are no guidelines on how to start a new project either. A project generator tool can be thought of as a competence sharing tool which people can use to share their knowledge and experiences from previous projects with others. For instance if a technology is outdated it can be removed or if there are additional features that future projects can benefit from can be included in the tool. The standardization and shared competence decrease the threshold to move developers between projects and reallocate resources. Developers would be familiar with the structure, since they have seen it before in previous projects. One downside however is that the project setup process becomes a big black box, which means that developers become unaware of how things work under the surface and therefore the knowledge of setting up things from scratch disappears.

4.3.3 To maintain such a software project generator at Jayway, maintain it as an Inner Source project

The participants' answers varied regarding whether they were interested in maintaining the software project generator in the future. All participants were interested in utilizing the project generator to create new projects in the future, as long as it is regularly updated. However, it is challenging to keep the tool at the forefront since software development technologies change rapidly. Everyone agreed upon the importance of a maintenance strategy to prevent the tool from becoming outdated. Juniors were eager to be part of the maintenance team. However, seniors despite the interest, lack time for maintaining in-house projects. The participants believed that there is a lot of potential for expanding the tool and that many technologies that have been proven to be efficient in previous projects can be included. Thus, allocating resources for maintenance is reasonable. The majority mentioned that management support is required in order to keep in-house projects alive, and senior participants believed that taking ownership of the product is an important factor for achieving this goal. If the management allocates time and resources there is a higher chance of success. However, in order to justify resource allocation for the maintenance, the product must be highly demanded by the employees and have significant value for the business. Seniors believed that juniors are proper candidates for maintaining such a tool. It would give them the opportunity to learn a lot about how new projects are initialized. Forming a community from different teams in the company for the maintenance of the tool, provides broad knowledge and insights from different people with varied levels of expertise. Lastly, when the Inner Source idea for maintenance was presented to the participants they agreed that this strategy is a good way to maintain the product and the idea is very close to what employees prefer to use when maintaining in-house projects.

Chapter 5
Discussion

We have developed a software project template generator to solve the identified problems at Jayway; choosing technologies, time pressure, including documentation, code analysis tools and testing, and finally, POCs that get stuck. Furthermore, we have proposed Inner Source for the maintenance of this generator. In this section we discuss potential challenges with these solutions. We also assess the scientific contribution with regard to its rigor, relevance and novelty. Lastly, we briefly discuss future works.

5.1 Software Project Generator

While studying this area we have come to the conclusion that it will require great effort from Jayway to keep the generator relevant. Software is constantly evolving, which makes software project templates short-lived. What is a good template today, might be completely outdated tomorrow. The templates must constantly be updated to keep up with the software evolution. Therefore a software project template generator would be under construction most of the time. In order to know what the template should look like, a thorough research must be conducted, similar to the one in this thesis. Then the findings must be implemented into the generator. Shortly, the maintenance of the generator is time consuming and resource-intensive. Are the benefits of using the generator worth the struggles of maintaining it?

We, as well as the employees at Jayway, can see the advantages of using a project template generator. However, there are challenges of actually applying it to real projects at Jayway. First of all, the project generator must work as intended to be used by the developers. It must be robust and trustworthy. Moreover, it must simplify the project setup significantly and gain a good reputation to become the standard way of setting up new projects. Some advantages that we have mentioned only exist when the majority of, preferably all, projects use the generator. Therefore, the more projects use the generator the more powerful the generator becomes.

Currently, the prototype only consists of basic functionality. In a real application of the project generator solution, the generator would have to consist of more technologies to be useful. There is however a fine line between including just enough technologies and including too much. Determining what and how much to include is a research in itself. Some employees at Jayway wanted to have an opinionated generator with default technologies for certain actions, while others wanted the generator to be fully configurable. Both approaches have their pros and cons. A strict generator would be harder to maintain, but it would yield standardization benefits, such as ease to reallocate employees within the organization and ease to ask for help outside of the project. However, it would probably be harder to convince people to use a strict generator. A configurable generator would be easier to maintain, and more people would be open to use it. However, we might lose some standardization benefits in this approach. It is worth mentioning that depending on the number of possible templates that can be generated by the project generator, it may or may not be possible to standardize the projects. If the number is too high, having a template is pointless since all projects can be initialized with different templates. It will be impossible to standardize the projects, however all new projects will be up to date with the latest trends and technologies. On the other hand, if the number is too low, the projects will be standardized, but they will also become stagnated.

Another challenge is the actual implementation of the project template generator. It must be implemented in such a way that allows for extensions and alterations and it must be built to run on all computers at Jayway according to the requirements described in Section 3.2.2. We experienced difficulties trying to achieve this, however, more seniority and programming experience would probably be advantageous.

5.2 Maintenance

Even though we consider Inner Source to be the best alternative, we can still see numerous challenges with keeping the project alive. After maintenance options suggested by the employees at Jayway and our research on Inner Sourcing in companies, the challenges that may be encountered by Inner Source practitioners can be categorized in three groups: organizational, people and product challenges.

Organizational Challenges
One of the organizational challenges is coordination of people. In a traditional project, it is more likely that the project manager assigns tasks to the team members. However, self-assigning tasks are very common in Open Source as well as Inner Source projects. This is not an efficient approach for maintaining a generator that requires constant change. If developers only self-assign themselves to tasks, the updates to the asset may occur in an uncontrolled and irregular manner. This may harm the project's speed and performance, as well as result in an outdated generator and lead to unavailability of new changes when it is demanded by different teams when creating new projects. Therefore pure voluntary contributions are insufficient for a successful Inner Source project. Another issue is that contributors and even the founder can leave the project and take no further responsibility for its maintenance. This can lead to incomplete Inner Source projects.

People Challenges
People can from time to time lack motivation for maintaining an Inner Source project. It can depend on different factors. One reason, as some of the interviewees mentioned, is daily work pressure. People who already are swamped with tasks in their own projects can not spare time for maintenance of in-house projects. One other reason can be that the employees have yet not discovered the benefits of the Inner Source project and they see no value in maintaining it. Even people who are active contributors and see the value of the project can become demotivated from a small number of contributors and/or lack of interests of the stakeholders. Another challenge is considered as lack of domain knowledge. This refers to people's unawareness of the ongoing Inner Source projects. This might harm the product if they unwillingly join to contribute.

Product Challenges
There are also product related challenges in Inner Source projects. One of the biggest challenges is to create a product with modularity architecture. The more people contributing to the project the higher the chances of finding bugs as well as expanding it. It is only possible if the software is modular which facilitates simultaneous development. Another issue to point out is the risk of poor quality. Even though open source development encourages developers to join and contribute, it is still important to monitor, supervise and review the commits, for insurance through pull requests, in order to retain a high quality of the asset. Having a strong sense of ownership of the product helps to maintain it in a more controlled manner for better outcomes. Furthermore, some people might treat in-hose projects as less important than external client-based projects which can result in putting less effort into best practices activities such as documentation, testing, refactoring and ect. Neglecting these activities deceases the quality of the asset and increases the risk of failure of the internal Inner Source project.

Strategies to mitigate maintenance challenges
One solution to the coordination problem is the existence of clear and defined responsibilities which can be determined by an employee who is assigned by the management to manage the project generator. For a successful project, acquiring top-level management is important which interviewees also agreed. For this to work, management at Jayway must be convinced that investing resources in the project generator is not inefficient and wasteful. This is only possible if the project creates value early and proves to be of great importance. Meaning, the project generator must simplify the project setup significantly and gain a good reputation to become a standard way of setting up new projects.

The employees must also be able to see the advantages of using the generator. Once management is on board, it can support the project by allocating resources, such as paying for changes that are necessary to the asset. This also mitigates the people related challenges since allocating paid time for maintaining the asset encourages and motivates more employees to contribute. Furthermore, management can minimise the unawareness challenge by advocating and advertising for the assets throughout the company.

Moreover, since support from the management is a necessity and they are the ones paying for it, they have supremacy on what changes should be included. Suggestions for changes must however be welcomed from all over the organization, but primary from the asset's stakeholders. The input and involvement of people from different teams with different backgrounds provide a wide range of knowledge to the generator and more optimized solutions may emerge as a result. Lastly, to mitigate the product challenges regarding retaining the quality of the generator, a core team must be assembled, preferably by the management. It must be clear who has authority to make changes to the asset, to protect it from bad and/or unauthorized commits.

5.3 Research Method Assessment

A significant activity in the visual abstract template is assessment of the research with respect to the research's rigor, relevance and novelty. Therefore, this section assesses how trustworthy and useful this thesis's scientific contribution is.

5.3.1 Rigor

Rigor refers to a science's validity. All activities in a design science research paradigm contributes to rigor of the research. Therefore we assess all activities one by one, to eventually summarize them and determine the rigor for this thesis. In the problem conceptualization phase and the evaluation phase, a factor which may increase or decrease the rigor is the use of empirical methods and the analysis of data. In the solution design activity, using previous work and referring to current research may increase the validity. Also, if alternative methods have been considered and discussed, the rigor may increase. When assessing the rigor of a qualitative study, trustworthiness is the main criterion. Credibility, dependability, confirmability and transferability are all terms that define the trustworthiness, known as the rigor of qualitative content. Credibility measures how well the researcher's interpretation of the participants' answers corresponds with their actual answers. Transferability describes how well the gathered data and the drawn conclusions may be transferred to other settings with other participants. Confirmability shows if it is possible for other researchers to confirm the findings of the research, and dependability refers to the consistency of the process over time [6].

Furthermore, there are several different strategies for achieving trustworthiness. For example, prolonged engagement, persistent observation, triangulation and member checking may be used to increase credibility, and thick description and audit trail may be used to increase transferability, dependability and confirmability. Prolonged engagement refers to getting to know the setting and spending enough time with the participants to be able to identify potential misinformation or deviation. Persistent observation means focusing on the most relevant parts and triangulation is a strategy where more than one data collection method, data type or investigator is used. To validate the data collection results, the interpreted data may be checked with the participants to confirm that it is correct, this strategy is called member check. The strategy for achieving transferability is thick description. It means, providing detailed information about the setting and the specific context of a data collection. Lastly, in order for a qualitative research to be dependable and confirmable, the research method must be described thoroughly from start to finish, this is called audit trail [6].

The problem conceptualization activity consisted of eleven one-to-one interviews. The intention of these interviews was to collect rich qualitative data regarding the current situation at Jayway, which makes the sample size of eleven participants appropriate. In order to assess the rigor, the trustworthiness of the interviews and the analysis of them, the following terms are discussed: credibility, transferability, dependability and confirmability.

Since this thesis was conducted at the office among the employees, where we have had the opportunity to get familiar with the setting and the participants, a prolonged engagement strategy has been followed. Furthermore, the interview questions were formulated with the thesis' research questions in mind, which ensures that all collected data is relevant to the problem, meaning that the persistent observation strategy has been used. For the sampling, a criterion sampling technique was used, where employees at Jayway with different roles and experience levels were chosen. This is an example of data triangulation. Moreover, each interview was transcribed by us both, which is a triangulation of investigators. And lastly, parts of the data from these interviews were later confirmed with observations in code repositories, which is a triangulation of methods [9]. Ultimately, all rigor strategies have been used to ensure credibility. Furthermore, the thick description strategy has guided us to make the interviews transferable, meaning that details about the setting and context of the interviews are provided along with information about the participants and the interview questions. Lastly, the research process has been documented step by step under Section 3 where information regarding for instance the interview transcription and analysis is present, which proves that a audit trail strategy has been used. With this motivation we conclude that the qualitative research in this stage has sufficient confirmability and dependability.

As a summary, the analysis based on these four terms suggest that the interviews are trustworthy, and hence may be considered to have relatively high rigor. However, there is always room for improvement. All stages can certainly be further elaborated. The biggest area for improvement in this research is the analysis of the interviews. Firstly, we did not use verbatim transcription when transcribing the interviews, meaning writing down every word of the interviewees. Instead, we used a method known as edited transcription where we listened to the recordings and grouped the information into areas of interest, which we had previously defined based on the interview and research questions. The downside of this method is that valuable information may be missed since the complete dialogue is not written down. Also, we as researchers may be biased and unintentionally ignore data that we did not expect. As a whole, we therefore estimate that the rigor of this activity is medium to high.

The solution design activity in this research was based on both literature and empirical studies. In order to reach high rigor in this activity, the potential solutions to the problem must be compared and the comparison must be grounded in theory. We started by researching current strategies for managing ATD. Thereafter, we compared two methods for implementing the project template generator, CLI and GUI. Lastly, we discussed whether to maintain the project template generator as an Open Source project or as an Inner Source project. All methods were found in literature and previous works. Consequently, we consider the solution design to have high rigor.

We assess the evaluation activity to have the lowest rigor due to various factors. Firstly, the technological rule about architectural technical debt is not validated in an optimal way. Since ATD grows over time and is often invisible until the symptoms start to appear, it requires testing over a longer period of time to validate whether the solution solves the problem or not. Since we only had two weeks for the validation, we decided to ask the participants about the solution and collect their predictions on its success. Furthermore, a comparison between two identical projects, with and without our proposed solution applied, would give a better understanding of the solution's effectiveness. The only way to achieve this would be to mock a testing environment and create one project two times, once with and once without the project generator. This would require time and resources, which we could not obtain for this thesis. Secondly, there were metrics that we could have measured, but we did not, for instance the time consumed to set up a project manually versus the time consumed to set up a project with the generator. This comparison would help us assess if the generator actually speeds up the development.

Unfortunately, ATD is extremely difficult to track. There are no specific metrics that indicates ATD in a system. This fact makes it hard to measure ATD which makes it hard to determine if an ATD reducing method is effective. Also, the sample of participants for the evaluation interview did not consist of any employee with a management role. The maintenance technological rule would be better validated by those who actually make decisions regarding the resource placement and the business model. Other than this, the interview process and analysis was conducted in a similar manner as the data collection in the problem conceptualization activity, and hence has a relatively high rigor. As a conclusion, the evaluation activity as a whole has a number of flaws which brings down the rigor.

5.3.2 Relevance

A scientific contribution may be relevant to both other practitioners and/or entire research communities. For a research contribution to be relevant to a research community the problem to be solved needs to be common, and the proposed solution should be generalizable. For a fellow practitioner, a research contribution may be relevant if it contributes to their research in some way [19]. Despite the low abstraction level, we believe that the three technological rules proposed in this study may be applicable to a wider audience. Practitioners in large to medium scale software consulting companies similar to Jayway may find all or some of the technological rules relevant. Also, future master theses at Jayway might use this thesis as a foundation. With this motivation, we claim that the scientific contribution of this thesis has a high relevance to other practitioners. When it comes to the research community, this thesis has a potential to contribute to areas such as Inner Source, Architectural Technical Debt and Standardization of Software Projects. All these areas have been subjects of many studies, however, case studies on large to medium sized software consulting companies are rare to non-existent. Therefore, we hope to contribute with new and relevant findings to these areas.

5.3.3 Novelty

The degree of novelty of a design science study indicates how new the scientific contribution is. Depending on the formulation of technological rules, all studies may contribute with novel research. Technological rules on a lower abstraction level may be more novel, but become too narrow and hence have less relevance for the research community [19].

Since this thesis research is a case study, the scientific contribution is novel due to the low abstraction level. This kind of study has never been conducted at Jayway before and the findings are new. Regarding Inner Source, there are several previous studies. However, none of these are case studies in IT consulting companies. Likewise, research on standardization of software projects in large-scale IT consulting companies is sparse. Furthermore, there are numerous studies on architectural technical debt and its causes, consequences and methods for managing it. However, the method that this study suggests has to our knowledge not been studied before. Therefore, we claim that this research's contribution is novel even on a higher abstraction level.

5.4 Future Work

As mentioned in Section 1.4 on page 9, this thesis may be used as inspiration or foundation for future theses or research at Jayway or other consulting companies. Moreover, we had to compromise on the quality of the evaluation activity due to lack of time and resources. Therefore, we believe that this thesis would benefit from another round of research where all technological rules are validated in a more trustworthy manner. An interesting topic is to study if severe cases of ATD are prevented in the future if the initial building blocks of the system are debt free.

Chapter 6
Conclusion

This thesis aimed to answer four research questions and the qualitative research was conducted in a large scale consultant company. The questions have been discussed throughout the thesis and are concluded in this section with the summarized answers to the questions.

RQ1: What challenges do employees at Jayway experience when setting up new software projects from scratch?

We have found that the challenges faced by Jayway employees when initializing new software projects are understanding requirements, choosing technologies, time pressure, code legacy, using POCs for the setup and including documentation, formatting/linting and testing.

RQ2: Out of these challenges, which ones introduce ATD to the project?

Challenges in understanding requirements and choosing technologies are strongly related to ATD. As well as time pressure and code legacy. New employees can make mistakes during the setup of software projects due to human factors, for instance inexperience. Lastly, POCs are used as foundation in many projects which leads to ATD. All these challenges introduce ATD to the projects right from the start.

RQ3: How can a software project generator be used to mitigate these challenges that cause ATD?

A software project generator would help Jayway create standard projects with all recommended technologies included from scratch, which mitigates the challenge of deciding what technologies to include. This also prevents inappropriate decisions regarding the project's architecture and content, which is a major cause of ATD. The generator would also help speed up the initialization and ease the time pressure, leavening more time on other activities such as devoting more time to understand the requirements better, which can reduce ATD. Additionally, the initialization of software projects will not be as sensitive to errors caused by human factors since the generator takes care of most of it.

RQ4: How can such a project generator be maintained in large scale consulting companies such as Jayway?

The best recommended solution for maintenance after our investigations is inner sourcing the project generator tool. The recommended model for the Inner Source is a hybrid between infrastructure-based model and project-based model. The top-level management support is acquired and plays a key role for the project's success. However the top-level management is not enough, following best practices, enthusiastic contributors, a well structured and a well managed core team are also essential factors for the project's survival.

References

[1] N. S. R. Alves, L. F. Ribeiro, V. Caires, T. S. Mendes, and R. O. Spínola. Towards an ontology of terms on technical debt. In *Proceedings - 2014 6th IEEE International Workshop on Managing Technical Debt, MTD 2014*, pages 1–7, 2014.

[2] Terese Besker, Antonio Martini, and Jan Bosch. Impact of architectural technical debt on daily software development work - a survey of software practitioners. In *43rd Euromicro Conference on Software Engineering and Advanced Applications*, pages 278–287. IEEE Computer Society, 2017.

[3] Maximilian Capraro and Dirk Riehle. Inner source definition, benefits, and challenges. *ACM Computing Surveys*, 49(4):67:1–67:36, 2017.

[4] Danese Cooper and Klaas-Jan Stol. *Adopting InnerSource*. O'Reilly Media, Inc, 2018.

[5] Ward Cunningham. The wycash portfolio management system. *OOPS Messenger*, 4(2):29–30, 1993.

[6] Satu Elo, Maria Kääriäinen, Outi Kanste, Tarja Pölkki, Kati Utriainen, and Helvi Kyngäs. Qualitative content analysis: A focus on trustworthiness. *SAGE Open*, 4(1):1–10, 2014.

[7] Neil Ernst, Stephany Bellomo, Ipek Ozkaya, Robert Nord, and Ian Gorton. Measure it? manage it? ignore it? software practitioners and technical debt. In *ESEC/FSE 2015: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 50–60. ACM, 2015.

[8] Inc Facebook. Create react app: Set up a modern web app by running one command. `https://create-react-app.dev/`, 2021. Accessed: 2021-03-02.

[9] Patricia1 Fusch, Gene E. Fusch, and Lawrence R. Ness. Denzin's paradigm shift: Revisiting triangulation in qualitative research. *Journal of Social Change*, 10(1):19–32, 2018.

[10] GitHub. Creating a template repository. `https://docs.github.com/en/github/creating-cloning-and-archiving-repositories/creating-a-repository-on-github/creating-a-template-repository`. Accessed: 2021-05-25.

[11] Heroku. Oclif. `https://oclif.io/docs/multi`, 2020. Accessed: 2021-05-25.

[12] Martin Höst, Klaas-Jan Stol, and Alma Orucevic-Alagic. Inner source project management. In *Software Project Management in a Changing World*, pages 343–369. Springer, 2014.

[13] Philippe Kruchten. An ontology of architectural design decisions in software-intensive systems. *2nd Groningen Workshop on Software Variability*, 2004.

[14] Zengyang Li, Peng Liang, and Paris Avgeriou. Architecture viewpoints for documenting architectural technical debt. In *Software Quality Assurance: In Large Scale and Complex Software-intensive Systems*, pages 85–132. Elsevier Inc, 2015.

[15] Antonio Martini, Jan Bosch, and Michel Chaudron. Architecture technical debt: Understanding causes and a qualitative model. In *40th Euromicro Conference on Software Engineering and Advanced Applications*, pages 85–92. IEEE Computer Society, 2014.

[16] Microsoft. Command shell overview. `https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-xp/bb490954(v=technet.10)?redirectedfrom=MSDN`, 2021. Accessed: 2021-03-02.

[17] Lawrence A. Palinkas, Sarah M. Horwitz, Carla A. Green, and Jennifer P Wisdom. Purposeful sampling for qualitative data collection and analysis in mixed method implementation research. *Administration and Policy in Mental Health and Mental Health Services Research*, 42(5):533–544, 2013.

[18] Aanand Prasad, Ben Firshman, Carl Tashian, and Eva Parish. Command line interface guidelines. `https://clig.dev/`. Accessed: 2021-03-02.

[19] Per Runeson, Emelie Engström, and Margaret-Anne D. Storey. The design science paradigm as a frame for empirical software engineering. In *Contemporary Empirical Methods in Software Engineering*, pages 127–147. Springer, 2020.

[20] F. Ryan, Michael Coughlan, and Patricia Cronin. Interviewing in qualitative research: The one-to-one interview. *International journal of therapy and rehabilitation*, 16(6):309–314, 2009.

[21] Niloofar Safavi. An empirical investigation into the adoption of inner source in it companies: A case study. Master's thesis, Chalmers University of Technology, 2019.

[22] Klaas-Jan Stol, Paris Avgeriou, Muhammad Ali Babar, Yan Lucas, and Brian Fitzgerald. Key factors for adopting inner source. *ACM Transactions on Software Engineering and Methodology*, 23(2):18:1–18:35, 2014.

[23] Klaas-Jan Stol and Brian Fitzgerald. Inner source-adopting open source development practices in organizations: A tutorial. *IEEE Software*, 32(4):60–67, 2015.

[24] GNU Operating System. Shell commands. `https://www.gnu.org/savannah-checkouts/gnu/bash/manual/bash.html#Shell-Commands`. Accessed: 2021-03-02.

[25] Daniel Turner. Qualitative interview design: A practical guide for novice investigators. *Qualitative Report*, 15(3):754–760, 2010.

[26] tutorialspoint. Difference between gui and cui? `https://www.tutorialspoint.com/difference-between-gui-and-cui`. Accessed: 2021-03-02.

[27] Roberto Verdecchia, Philippe Kruchten, and Patricia Lago. Architectural technical debt: A grounded theory. In *Software Architecture - 14th European Conference, ECSA 2020, L'Aquila, Italy, September 14-18, 2020, Proceedings*, volume 12292, pages 202–219. Springer, 2020.

[28] Evan You. Vue cli: Standard tooling for vue.js development. `https://cli.vuejs.org/`, 2021. Accessed: 2021-03-02.

Appendices

Appendix A
Interview - Problem conceptualization

Part 1

- What is your role and your responsibilities in the project?
- What is the project you are currently working on? Please describe the project briefly.

Part 2

- What are your experiences of starting new projects at Jayway?
- What challenges do you experience when you set up a new project?
- What challenges do you/developers face when choosing technologies such as frameworks, libraries, languages, etc.?

Part 3

- Can you give us an overview of the architecture of your application?
- Can you briefly explain why you chose this particular architecture?
  - How does your web application look like?
  - What frameworks and packages are used for frontend and backend?
- How do you decide what packages, frameworks and libraries to include in your project?
- Is there any architectural design decision you regret today and you wish to replace?
- How common is it that you realize that a choice of package, framework or library was wrong later on in the project?
- What are some challenges that you face in the project now that you think could have been mitigated or avoided with a better project start?

Part 4

- In your opinion, which areas in a project's initial process can/need to be improved?
- Would you find a standardized software project generator useful? Would you use it?
  - In what way, do you think, such a generator would help you in setting up a new project?
- Do you think others would be willing to use it? And if yes why and how often? If no why not?
- How can a software project generator tool be maintained at a company such as Jayway?
- What are some words of wisdom that you would like to share with new projects?

Appendix B
Interview - Evaluation

- What is your overall impression of the CLI?
- Was it easy to use the CLI?
- How much do you think the CLI accelerated the project setup process?
- Do you think the CLI can be used to reduce mistakes at the start of the project due to human factors?
- Do you think the CLI can improve the code quality in general?
- Do you think the CLI can help to follow best practices?
- Do you think that the CLI can help increase collaboration and competence sharing between different projects within Jayway?
- What other, if any, benefits do you see from using the CLI?
- Do you think there are any disadvantages from using the CLI?
- Do you think it is worth allocating resources to maintain the CLI?
- What are your thoughts about Inner Source as a maintenance strategy for the CLI?
- Would you like to maintain the CLI in the future?
  - Why?
  - Why not?
- Would you like to use the CLI to set up new projects in the future?
- How useful do you think the CLI will be for setting up new projects at Jayway?
- Do you have any suggestions for improvement of the CLI?

Appendix C
Consent Form

Consent for participation in a research interview for master thesis

By signing this consent form

- I agree for this interview to be tape-recorded.
- I understand that the audio recording from this interview will be analysed and used only for this master thesis carried out at Jayway.
- I would not be personally identified, and the data from the interview may be used in any conference presentation, report or journal article developed as a result of the research.
- I understand that no other use will be made of the recording without my written permission, and that no one outside the research team will be allowed access to the original recording.
- I understand that the recording will be deleted after the thesis work is complete.

--	----------------------	--------------------------
Name of the participate	Date	Signature
---	----------------------	--------------------------
Name of the Investigator	Date	Signature
--	----------------------	---------------------------
Name of the Investigator	Date	Signature