Assisting product line thinking and information sharing using a real-time visual tracking model for agile epics

Master of Science in Technology Thesis
University of Turku
Department of Computing
Software engineering
2023
Mio Mattila

UNIVERSITY OF TURKU
Department of Computing

Mio Mattila: Assisting product line thinking and information sharing using a real-time visual tracking model for agile epics

Master of Science in Technology Thesis, 92 p., 3 app. p.
Software engineering
May 2023

The aim of this thesis is to understand and reduce repeated development of software artefacts, mainly in terms of the software components that are produced in the organization of Natural Resources Institute Finland. The thesis consists of a literature review around code reuse and the software product line method of building software. In the case study of the thesis, a visual model is added to the Jira environment of the organization of Natural Resources Institute Finland DIGI unit. The project workers associated with each project are interviewed both before and after the visual model was added.
Through the results of the interviews, the study identifies improvements to understandability and presentability of projects. Suggesting that the addition of a graphical model for epics on a high abstraction level helps project workers increase information sharing within an organization, but more research is needed to understand and enable the technical impact of the model.

Keywords: Visualization, information, software product line, software reuse, component-based development, software development

Contents

List of Figures

1 Introduction

Since I started studying computer science over seven years ago, I've been wondering how do programmers avoid building the same systems in different places. Back in 2015 I had this thought that there must be ready-made software solutions in the world for managing large organizations and their data, and that the market share of these solutions would be fairly big.

This way there would not be too many providers of the same solution, lets say in the field of hospitals and patient care for example. My thought also was that the building blocks for implementing management systems would be easily available.

I've assumed that the role of a organizations IT department is equal amounts of maintaining the systems and building new software, that the IT department could easily address the needs of the whole organization.

But now, as the dark reality of the world has slowly been setting into me, I've discovered that no, there are no real easy building blocks, and no, there are no widely used solutions worldwide. Everyone's just implementing the same things over and over and over again.

So for my thesis I wanted to do a case study that was grounded in practicality and fit my current employer, possibly adding some new value to my employer's software development methods as well as improving upon the literature that has already been written. The idea was largely centered on how I could reduce the amount of repetitive software development worldwide.

I thought that I cannot find a simple solution to this problem but I can build upon a solution from the bottom up, and this thesis is a part of that bottom solution, me finding my bearings on how repetitive software development could be reduced and how the software development process works on a very basic level.

At Luonnonvarakeskus, the Natural Resources Institute Finland also called Luke, our project team kept running into the challenge of needing new common components to function, while also acknowledging that those common components could be used for our own project and for other projects at Luke, components such as contact information register and login manager. Our team also knew there was a high likelihood of similar components being implemented before in previous and adjacent projects, and us simply not knowing about their existence.

I wanted to see if this problem could be addressed on software production level, if our tools could somehow facilitate the development of software components in a way that made their development more visible and easier to be included in the design process, and since I had worked as a Jira-admin in Luke for half a year, I decided that Jira as a platform would be best suited to answer this challenge.

Jira is a issue tracking system, at Luke it also functions as a requirements management system, as the requirements are documented as jira issues, often at epics level. Because of this it was the tool where software development work was most often listed, referenced and viewed by developers of our project.

To see how we could make software production more complete and reduce repeated work, I decided to do a interview-based case study at Luke. To see if my project team, and other developers at Luke could benefit from a visual Jira add-on, I first implement the add-on, and after a trial period, interview project members to get an idea whether or not the add-on helped with the software development process.

The add-on would help people understand the state and structure of projects better, and also make it easier to communicate with other users using the visual graph-based views. My hypothesis being that if visual clarity and linking of issues between projects was added to Jira, information sharing and identifying repeated software components would get significantly easier.

The research question to direct this thesis formed to be: How can a high-level real-time visual tracking model assist in bringing product line thinking and increasing information sharing within an organization? To answer the research question, the original research question was broken down into three subcategories which would be answered trough the case study interviews. The subquestions were:

- In what ways can a visual Jira architecture model of high level abstraction help software engineering?
- Could a Jira architecture model of high abstraction level help in information sharing for both a project and an organization?
- Could a repeatable Jira architecture model of high abstraction help with developing repeated software components?

The focus of the study was in implementing a visual model for agile requirements artifacts in order for it's users to better detect repetitive components, and to communicate better through being able to see projects state in a clear, up-to-date visual manner, ultimately paving way for the organization of Luke DIGI unit to better avoid building the same components twice and being able to move towards product line thinking. To measure the effect of the study, two rounds of interviews were done before and after the implemented visual model, the interviews are analyzed through an adapted version of thematic analysis.

There are three main chapters in this thesis, the first two are chapters 2 and 3 which go through the current research environment around software reuse and software product line through the means of a literature review. In chapter 4 the research setting, implementation of the visual model, interviews and the results of the study are observed. Lastly, the findings of the thesis are discussed and concluded.

2 Conventional code reuse and software product line

Software reuse is the process of creating new software systems from existing software artifacts instead of always building new software systems from scratch[1, p. 133]. The existing software artifacts can be anything like source code fragments added with simply just copying and pasting, but they can also be things like design structures, module-level implementation structures, specifications, documentation, transformations, and so on[2].

There is a great diversity in software technologies and and artifacts that involve reuse, but there is also commonality among all of the different reuse techniques[1, p. 133]. This commonality can take the form of software component libraries, object-oriented knowledge bases, application generators and code skeletons[3], they all involve abstraction, specialization of use, selection and some level of integration.

Developing a viable reusability system is an investment that does not have an early payoff, meaning that populating a library of reusable artifacts blocks the development of a working reusability system[3]. Most reuse strategies fail in their efforts to have any real technical or economic impact[4]. Making high-quality modern software is hard[5], and making it systematically while also enabling reuse is even harder[6].

This chapter takes a brief look on the different kinds of approaches towards software reuse in the field of software development, weighted towards introducing the software product line engineering method.

2.1 Fortuitous, small-grain reuse

Reuse, as a software strategy for reducing development costs and increasing software quality is not a new idea. Many reuse agendas have focused on reuse of relatively small parts of code, called small-grained reuse. Libraries of algorithms, modules, objects, or components, and developers are urged to use the library instead of writing their own code from scratch. [7, p. 6–7]

Unfortunately, it takes time to locate these small pieces of code, add them to the library, and integrate them into a system, this process often takes more time than making a new system from scratch, and the possibly existing documentation for these pieces of reusability might make the integration easier but not how it can be generalized or adapted to the new system. [7, p. 6–7]

Opportunistic reuse, for example cutting and pasting code from old programs into new ones[5], creates technical debt[8] and the traceability of copy-pasted code may introduce problems tracing back to it's source[9]. Things like the increased productivity from code reuse and cost reduction, have downsides of snowballing dependencies and questionable maintenance[9].

Benefits of small-grained reuse depend on the software engineer's predisposition to use the contents of the library, the applicability of the library's contents for what the engineer needs, and the success of the integration of the library's units into the rest of the system that the engineer is developing. If reuse occurs under these conditions, it is deemed as fortuitous reuse and the payoff is usually nonexistent. [7, p. 6–7]

Maximising the reuse of tested source code can bring improvements in cost, time and quality, while also reducing the need to develop new code. Reusing code is only the low-level beginning of reuse. Practising reuse systematically requires additional effort. [10, p. 11]

Single-System Development with Reuse

There's also single-system development with reuse. If you are developing a new system that is very similar in structure to the one you have built previously, you borrow what you can from your previous offer, modify it as necessary, add whatever it takes, and ship the product, which then assumes its own maintenance cycle separate from the original system, you have just performed an action called "clone your own", which is single system reuse. By doing this, you have taken a economic advantage of your previous work by reusing a part of a another system. But now a new problem has emerged, you have two entirely different systems, not two systems built from the same base. This is also ad hoc reuse. [7, p. 7]

2.2 Component and service based development

In 1999 Gartner Group predicted that by 2003 up to 70% of all new software-intensive solutions will be constructed with "building blocks" such as prebuilt components and templates. [11]

Literature offers many definitions for what the word component means in software development[12]. Perhaps the most used one is Szyperski's definition[13]; *A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.* [12, p. 195]

In component-based development, a component is a independently deliverable piece of functionality providing access to its services through interfaces. It is more than just a modular programming approach, an object or class in an object-oriented system, or a package in a system model. The basic approach of CBD is to build systems from these well-defined, independently produced pieces.[11]

The purpose of component-based development, is to develop systems as assemblies of parts (components), the process of making these parts as reusable entities as well as the maintenance[11][12, p. 4–5] and upgrading of these systems by customizing and replacing such parts. [13]

This is an ideal software development scenario where development is done by building applications by assembling high-level components[10, p. 67], also known as services, from both in-house and from the market of components[7, p. 7]. If a desired component is not available, it can be built from lower-level components or built from scratch by code[10, p. 67].

Szyperski et al.[12] describe component-based development as a middle path between fully self-built custom software and buying a ready solution[12, p. 6], Illustrated in figure 2.1.

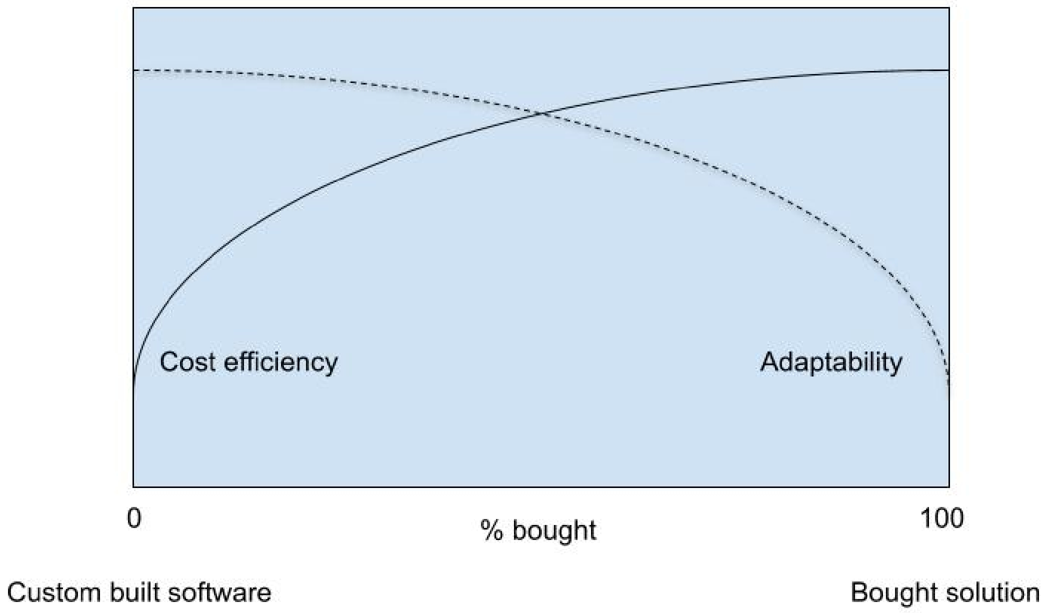


Figure 2.1: Custom software and outsourced software compared, drawn by ideas of Cost efficiency and adaptability by Sryperski et al. [12]

Component-based systems provide advantages when compared to buying ready-made software solutions, they provide more flexibility, as they are extensible and modifiable, they also allow the acquiring party more organizational flexibility, as the organization does not need to fit their practice around the software solution that was bought. [12, p. 5–6]

And while custom software does offer high adaptability and can perfectly fit the needs of a organization, production from scratch is a very expensive undertaking[10, p. 97]. Reaching the optimal solutions is hard in anything but the local areas of expertise, and the risk of failure is high. Interoperability requirements and delivery time into the desired market window can also introduce a burden. [12, p. 5–6]

The ability to both buy and develop components could offer guaranteed quality, adaptability[12, p. 4] and faster time to market when comparing component-based software to both custom built software and a ready made standard solutions[12, p. 4–504].

The advantages of component based design include more effective management of complexity, reduced time to market, increased productivity, a greater degree of consistency, and a wider range of usability. [13]

But there are also disadvantages and risks in using Component-based design which can be a danger to it's success[13]. The factors which can discourage the development of reusable components is the complexity difference in designing and developing components as they require more effort in designing requirements, testing as well as integration[14] [15]. Even though development time of systems using components is reduced in coding, debugging, unit testing, and code inspections, design and testing are modified or increased[14] [15].

In practice, those working in the field of software have noted that the development and design of reusable components takes significantly more effort compared to components that are designed for a singular use [16] [13] [15] while the cost can be gained back from the reuse of the reusable component[16]. Engineers at siemens had made an observation that the overhead cost of developing reusable components was recovered after the fifth reuse[16].

Difference between component-based development and product line engineering

Northdrop et al. [7, p. 7] states that in a product line the generic form of a component is evolved and maintained in the core asset base as opposed to component development where variation is usually accomplished by writing code, and variants are most likely maintained separately. Northdrop et al. also state that component-based development lacks the technical and organizational management aspects that are important to the success of a software product.[7, p. 7]

It is clear that Component-based development is very close to software product line development. Components, plans for reuse and systems planning reuse of the components in CBD are all still very close to the core idea of SPL, CBD could be seen as a precursor to SPL.

2.3 Software product line engineering

Initially this research struggled to find reference points to literature that would appropriately describe the environment which the author was trying to find solutions for, but as a saving grace the term software product line was found. The term "software product line" is used almost synonymously with the term "software product family". In Europe the term software product family is used more often and in North America the term software product line is used more frequently[17, p. 4-18].

So what is a software product line? The way goods are produced has changed throughout human history, formerly goods were handcrafted for individual customers, and later on, product lines started to appear which enabled the production of large amounts of products for large audiences much more cheaply than what the previous method, crafting by had, allowed. Although the drawback to the product assembly line being that it reduced possibilities for diversification. [17, p. 4-18]

Roughly, both individually crafted and mass produced produced products can be identified in the field of software nut just in traditional fields of production. The different kinds of software products can roughly be identified as individual software and standard software. Each type of these products has it's drawbacks, individual software can be costly, and standard software products lack sufficient diversification. [17, p. 4-18]

So how do we move on from individual software to mass produced software where there can also be diversification? We design a common platform on which to build more individualized products. Much like in the car industry where engineers started to build sets of platforms for cars.

A car chassis for example, is a part of the car platform[18]. For example, entirely different cars could be built for different customer segments[17, p. 4-18]. Large-scale production where the products are tailored to individual customers' needs is called mass customization[17, p. 4-18].

By systematically combining a common platform and mass customization, the benefits of both individual and standard software can be achieved. The practice of systematically combining mass customization and the use of common platforms for the development of software-intensive systems is called product line engineering. An example in the physical world would be a car platform: A set of bases is built and variations can be designed on top of the platforms. [17, p. 4–18]

2.3.1 Variability in software product lines

Documenting and managing variability is one of the key properties that makes software product line engineering unique. The explicit definition and management of variability differentiates SPLE from single-system development and software reuse. [17, p. 55–88]

A variant is a representation of a variability object within a domain artefact[17, p. 55–88]. Meaning for example, that if the domain artefact is a car, the variation point for that car could be ”type of wheels” and the different variants could be square wheels and round wheels. Than playing along with this example, the resulting applications from this product line could be square-wheeled car and round-wheeled car. All the parts of the car could be the same exept for the wheels.

Variability enables the derivation of different distinguishable applications in the product line. The importance of being able to communicate the variability and available variability within a software product line to customers entails the distinction between internal and external variability. [17, p. 55–88]

Variability can be represented through various models, plans and textual documents, variability modeling is a central technique required to put software product line engineering into practice. The variability of a software product line is represented and specified in a separate model consisting of the variation points, variants and their relationships. [17, p. 55–88]

A variability model must be established so that developers and other project workers have a consistent way of representing variability, and have a guideline and a shared understanding on the way variability is represented in all of it’s forms in different artefacts. [17, p. 89–113]

It’s worth also noting that the tools for modeling variability play a key role in the variability model representation and understandability. There are additional challenges in documenting variability, as feature models for variability contain variability[17, p. 89–113]. Documenting variability in textual requirements presents additional challenges for readability[17, p. 89–113].

The variability model helps developers in the domain engineering, so that variable requirements artifacts are consistent, and in application requirements engineering the variability model is used to create consistent sets of requirement artifacts. [17, p. 89–113]

In their book, Pohl et al. [17] list the intricacies of variants as well as their documentation in fine detail, this section only represents a very fine scratch on the surface of the subject: product line variability.

2.3.2 The basics of SPL

This sections aims to define and introduce more specific traits of software product line through the benefits and costs of the paradigm.

The benefits

According to Reinhartz-Berger et al. [4], software product line engineering discipline addresses both technical and economic benefits and achieves systematic reuse of software across the product line through the following means:

1. capturing common features and factoring the variations across the domain or domains of a product line
2. developing core assets used in constructing the systems of the product line
3. promulgating and enforcing a prescribed way for building software product line assets and systems
4. evolving both core assets and products in the product line to sustain their applicability

Additionally, Northdrop et al. [7, p. 8–9] list the organizational benefits of having a software product line approach.

- large-scale productivity gains

- decreased time to market
- increased product quality
- decreased product risk
- increased market agility
- increased customer satisfaction
- more efficient use of human resources
- ability to effect mass customization
- ability to maintain market presence
- ability to sustain unprecedented growth

These benefits lead to the organizations using SLP to having competitive advantage, the benefits are derived from the reuse of core assets in a strategic and prescribed way. [7, p. 8–9]

According to Northdrop et al. [7, p. 8–9] once the care asset base for the product line is established, there are direct savings each time a product gets built in requirements, architecture, components, modeling, and analysis:

Requirements There are common product line requirements, which can be reused.

Architecture Architecture documents are a big commitment requiring the time and effort of experienced software developers setting the quality details of a system. If the architecture is wrong, the system cannot be saved. Software product line allows for a singular architecture used for each product of the same product line and only needs to be instantiated. Considerable time and risk are spared.

Components Up to 100% of the components in the core asset base can be used used in each product.

Modeling and analysis Different analysis outputs have been established for existing core assets. With each product, the confidence of estimations gets better and the bugs associated with distributed systems and their networks and such have been corrected.

Testing Generic test plans, test processes, test cases, test data and communication paths have already been built, only new test cases have to be made for variations in the product line.

Planning The production plan has already been made and budgets and schedules from previous plans provide a good basis for product work plans.

Process Configuration management tools and processes, management processes, and the overall software development process are in place, have been in use, are robust, reliable, and responsive to the organization's needs.

People Fewer people are required to build products and people are more easily transferred across the entire product line.

The assets base includes those artefacts that are the most costly to develop from scratch in software development, things like requirements, domain models, software architecture documents, performance models, test cases and components[7, p. 7]. In a product line, all the core assets are designed to be reused and are optimized for use in more than a single system. The reuse with software product lines is comprehensive, planned and profitable[7, p. 7].

Software product lines give economies of scope, which means that you take advantage of of the many products in the organizations portfolio, that are very similar to each other, but not by accident. It's because the you it that way. Deliberate, strategic decisions are made and you are systematic in effecting those decisions. [7, p. 5–6]

After a product line has been implemented, creating software is more about integration and less about traditional coding. For each software product line, there is a predefined guide or plan that tells the exact path to building a product. [7, p. 5]

The costs

Even though the strategic reuse of core assets that defines product line practice represents an opportunity for benefits across the board, decision to launch a product line should not be made lightly, any organization should have a clear business goal in mind and the benefits of those goals should align with the achievement of those goals. When establishing a software product line there's going to be a upfront cost of establishment, as well as costs related to the maintenance of the core asset base. [7, p. 11–13].

Bringing a software product line to the market requires a blend of skilful engineering, technical management and organizational management [7, p. 3]. Requirements become much more tasking in terms of negotiation and analysis, requirements negotiations have to account for both common requirements and variation points. [7, p. 11–13]

Defining the product architecture also becomes harder, the architecture must support the variation of the product line, requiring additional expertise, components must also be designed with extensibility in mind so that they can support variability as well as generality. [7, p. 11–13]

Planning the business case, marketing analysis and estimates for time and cost must also be generalized, test artefacts must be made more robust so that they can support more than one product, and they must be extensible so that they accommodate variation among the products. [7, p. 11–13]

And lastly, personnel must be trained beyond general software engineering and corporate procedures, to ensure that they understand software product line practices and can use the core assets and procedures associated with the product line. The training of new personnel also becomes more tasking as they need to be introduced to product line development so training materials must be created that address the product line. As the organization matures, the shift towards more specific domain expertise and technology forecasting, the shift towards this kind of expertise change must be managed. [7, p. 11–13]

It takes a certain level of maturity from the developing organization to use software product line successfully. There are technological practice changes as well as managemental changes to be made in order for a organization to use SPL. The successful use of SPL tests the prowess of an organization in terms of organizational and technological leadership through a multitude of improvements.[7, p. 12-13] The developing organization also need to have process discipline, as organizations that fail to define and adopt processes will have to address those deficiencies early in their path to SPL adoption[7, p. 174]. Software product can be iteratively improved on during it's adoption and use[7, p. 174].

However, for each of the aforementioned costs and drawbacks, the value of the investment is usually far greater than the investment cost, most of the costs are up-front costs, but once the approach is established, the organization's productivity accelerates rapidly, and the benefits far outweigh the costs, developer and customer confidence both rising with each new instantiation. [7, p. 10–13]

2.3.3 Visual assistance

For the longest time, the data produced by computers has been visualized through graphical methods[19]. There are many advantages to using visuality to represent data[20][21].

in terms of readability, Well-designed charts are more effective in creating interest and appealing to the viewers attention than huge text-based tables and visual relationships are easier to grasp and remember. [21, p. 1–51][20, p. 6–14]

Graphical displays also represent many details in a small space, thus providing a comprehensive picture of the problem environment, while also encouraging the eye of the beholder to compare different sets of data simultaneously. [21, p. 1–51]

Software product lines are complex environments where the interactions with the product line as a whole are complex, and in large numbers[22]. This presents new unique challenges and makes maintenance and evolution tasks challenging[22], demanding robust tool support for the software engineers to carry out the task of feature implementation and interaction[23].

Unfortunately, there have been no robust visualization tools to support the work of engineers to identify, analyse and manage features interactions which is indispensable for performing SPL evolution and maintenance tasks[23].

Current research around software product line engineering seems to rely on diagram tools like the graphic API provided by Java SDK and Eclipse Modeling Framework used with the Graphical Editing Framework. Visualization could be used a lot more for SPL activities, when one looks at the publications made in the field of software product line engineering, the visual techniques for testing have only been used by 2 papers up until the year 2017. [24]

The most common artifacts that were visualized in research were feature models. And the most common visualization technique was both trees and graphs. [24]

Complex interactions and layout possibilities are not used much in research, however, using colour coding to distinguish artifacts from each other was a common trend. [24]

The visualization of software is a subject which is gaining attention in the software engineering community[23]. At the end of their paper, Illecas et al. [23]

mention that there are performance issues with larger feature models, and that they aim to further integrate visualization with source code editors by introducing a link between them. This might be for the same reasons that this study aims to include the visual model in the same place as where the developers do their daily work.

2.3.4 SPL and agile development

While SPL and agile methods may seem contradicting, they can be combined in the real world[25].

There exists a method called SPLICE, invented rather recently, a lightweight software product line development process for small and medium size projects[26]. Altohough Vale et al. mentions that there seems to be no literature describing real-life experiences in small and medium size companies using the method, not during the writing of the splice method[26], there appears to be a study [25] by Geir K. Hanssen and Tor E. Fægri, where they observed an organization successfully combine SPL and agile methods.

SPLICE addresses the needs of small development teams who aim to adopt SLP practices with low initial investment and fast return of investment in a volatile environment by combining SPL and agile practices. In SLP there are a set of roles for team members and a scrum team, but instead of a product owner, there's a scope owner. [26]

Instead of sprints starting with a regular planning phase, there's portfolio planning which aims to provide a high-level description of the SLP business domain by setting business goals, marketing strategies, products and major features. [26]

A product map is built along with a feature model, and prioritization is set for the major features. Portfolio planning results in stakeholders being able to perform comprehensive planning for development releases and sprints[26]. However, the paper by Vale et al. does not provide proposals for the business case of the funding structure associated with the agile product line.

According to [26] there are two major phases in SPLICE: Portfolio planning and release development. Portfolio planning occurs first, to set the high-level domain and business plans, Release development focuses more on the sprint planning side of things. The research done by Hanssen et al. and Vale et al. proves that agile and software product line development can be combined with promising results.

Software product lines continue to be the subject of research and research surveys[24], and pose a possibility of a different future for the software development environment. Maybe a future where the constant creation of unique bases for every customer becomes a little less unique in the modern software development environment.

3 Software product line activities

This section uses two main sources to describe the activities of product line engineering, ***A framework for software product line practice, version 5.0*** [7] by Northrop et al. and ***Software product line engineering*** [17] by Pohl et al. The approaches of these two books differ in terms of the terminology used, and slightly in terms of structure, but the outline and meanings behind the terminology are essentially the same.

A notable key difference is that Northrop et al. in ***A framework for software product line practice, version 5.0*** describe the organizational aspects of product line engineering separately in an activity they call management.

In a software product line there are three main activities, core-asset development or domain engineering, product development or application engineering and management[7], illustrated in figure 3.1. The products or applications are built from core-assets, but the core-assets may be built from existing products, or the core assets that they are comprised of. Management oversees the synchronization of these two processes[27].

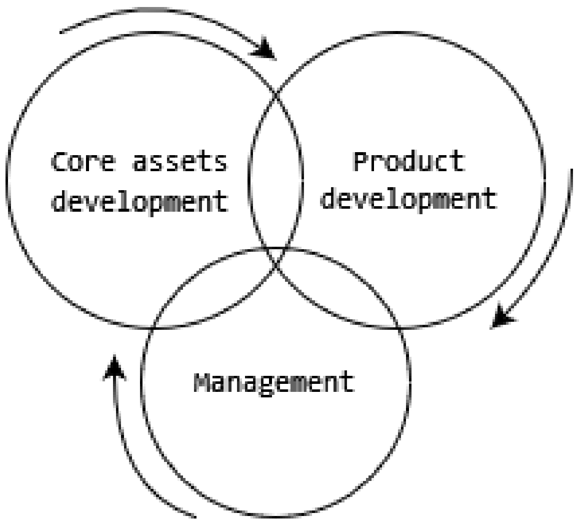


Figure 3.1: The essential activities cycle of software product line according to Northrop, Linda, et al. [7]

3.1 Domain engineering and core assets development

The article *A Framework for Software Product Line Practice, Version 5.0* (2012) [7] lists a different name for the domain engineering process called core assets development. The article also recognizes that the terms core assets development and domain engineering are often used synonymously in literature. To describe the domain engineering process, exclusively in the context of product lines, the term core assets development will be used.

3.1.1 Domain engineering

Domain engineering is a set of activities that aim to develop, maintain, and manage the creation and lifecycle of domains[4]. A domain can be defined by the set of problems that applications can solve in that problem domain[28][10]. Examples of domains include airline reservation systems, software development tools, user interfaces, and financial applications[10].

Software product lines give us a way to design and develop closely related systems in unison. This kind of a application family shares a lot of common features, and thus, it is possible to produce the members of the same family together in order to take full advantage of reusable requirements and other common elements. [29]

To make reuse possible, software engineering is sectioned into two parts: domain engineering to identify, find and implement the common features, and application engineering that handles the production of individual applications[29]. This section focuses on the domain engineering part.

Both SLP literature and component-based development mention domain engineering as a focus. For example the works [30][7][31][17][29] all point out domain engineering as a part of either component development or as a part of software product line component development.

Domain engineering is an activity that should be carried out at the beginning of the software specification phase if reuse is to be supported. Domain engineering can yield an initial taxonomy of the main concepts of entities within an applications domain along with a artefact of some kind, such as a abstract model of the application. Essential properties of the domain are captured and initial candidates of reusable components can already be drafted. [31]

In terms of specified benefits of domain engineering, Bert et al. [32] have noted that domain engineering offers several benefits in the field of digital decision support software(DDSS). These benefits include reusability, adaptability, enhanced communication and collaboration, improved maintainability, reduced development time and costs, as well as increased interoperability. These results are very similar to the benefits of component-based development described earlier, as domain engineering is a part of component-based development[31].

According to Harsu [29] Domain engineering is most often divided into three main phases: Domain analysis, domain design, and domain implementation. Domain engineering process starts with domain analysis. The primary sources of information are the existing applications and experts on the respective domain[31].

3.1.2 Domain analysis

Domain analysis can be seen as a process, the purpose of domain analysis is to use software development information in a manner which leads to structuring and organizing it to enable further reuse. Domain analysis supports the development and evolution of information infrastructure that helps and enables reuse. Pieces of this infrastructure can be domain models, development standards, and libraries or repositories of reusable components. [33]

Domain analysis is not only restricted to product-line related applications, it can also be used in single-systems engineering. The corresponding term for single systems is system analysis. [29]

According to Harsu [29] the domain model, a result of domain analysis can consist of the following parts:

Domain scoping

Domain scoping consists of domain definition and context analysis to find and set the boundaries of a domain.

Commonality analysis

Commonality analysis produces the commonality document, it's purpose is to consider commonalities and variabilities in the domain by studying the requirements and properties of the applications and concepts in the chosen domain.

Domain dictionary

Domain dictionary is like a contextual wikipedia for the domain, it's produced so that communication between stakeholders and project members becomes easier and more precise.

Notations (concept modeling, concept representation)

Notations provide a uniform way of understanding, notations of the domain model are things like object diagrams, state-transition diagrams, entity-relationship diagrams, and data-flow diagrams. A guideline would be to use notations and language that is easy to understand so that learning overhead can be avoided[34].

Requirements engineering (feature modeling)

Requirements engineering is gathering, defining, documenting, verifying[35, p. 24] and managing[35, p. 234] a set of requirements that specify an application[36]. In the domain engineering context, this refers to the reuse and configuration of the requirements among individual applications[34]. These requirements can also be called features[37].

3.1.3 Domain design

Domain design, and domain implementation come after domain analysis, according to Harsu[29].

Domain design sub-process produces the reference architecture for defining the main software structure and it's design patterns[17, p. 218–239]. Domain design provides reference architecture for the software product line from domain realization to application design[17, p. 218–239].

Domain design is performed iteratively along with domain requirements engineering as requirements are revised[17, p. 218–239].

In addition to architecture style, domain design produces a production plan or an architecture to tell how the application can be derived from both the core architecture and from the reusable components[29][17].

A production plan may consist of the following: The organization developing the core assets, core asset development and maintenance, dependencies among the assets and products, core assets configuration, estimations for development time, resources for core-asset development. [27]

Domain design may be a part of analysing the core architecture against its quality requirements in order to reveal possible risks in the architecture[29]. According to [38], the system architecture documents should be evaluated early, since it makes iteration of errors easy and cost effective during the planning phase, instead of during implementation.

It would be preferable to use multiple views for representing the common architecture, since different views can help different project participants understand the architecture better from their respective views[39].

3.1.4 Domain implementation

Domain implementation is essentially implementing the architecture, reusable components and the tools designed in the design phase, it consists of writing documentation and implementing domain-specific languages and generators. [29]

3.1.5 Core assets development

Core assets development process is very similar as the domain engineering process[7, p. 14] described previously, as a whole, it contains almost all the same activities that domain engineering describes. Northdrop et al. [7] describe core assets development through its outputs:

Product line scope

The product line scope describes the products that will be included on the product line or that the product line is capable of including. The simplest scope may just be a list of product names. Typically the descriptions include similarities and differences in the products and operations that they provide in terms of quality attributes.

Core asset base

The core asset base includes all the core assets that form the basis of the products that the product line can produce. Core assets can be things like tools, generic components, processes, personnel, training, test cases, test data, schedule estimates and the like.

Production plan

A production plan is used to describe how the products are produced from the core asset and fills two roles: It includes the building process used for building products and it lays the project details that enable the execution and management of the rest of the process such as the schedule, bill of materials, and metrics.

The production plan also includes the production method for implementing the possible core assets, were they outsourced or built in-house. It includes things like models, processes, and tools to be used in the attached processes across core assets.

3.1.6 Similarities of core asset development and domain engineering

Here we can see that the outputs of core assets development are extremely similar to the outputs of the domain engineering that Harsu [29] describes. Another very similar approach is the division of terms that Pohl et al. [17] uses, they describe domain engineering through these different sub-processes:

Domain requirements engineering

Produces the domain requirements artifacts and variability model for the domain, meaning requirements specification for the realization, design and testing phases.

Domain desing

Reference architecture, reusable software artefact selection.

Domain realization

Design and implementation of reusable software assets such as interface design and reusable components.

Domain Testing

Validates output of other domain engineering sub-processes resulting in domain test artefacts.

It can be concluded that Domain engineering terminology, approaches and descriptions differ from core asset development in the current literature space, but the key ideas and theory is essentially the same, but with different points of focus.

3.2 Product development and application engineering

Product development, sometimes called application engineering[7, p. 14], is the process of design and implementation of individual applications[29].

Product development depends on the outputs of core assets development, product line scope, the core assets, and the production plan.

The builders of the products use core assets such as components to build product applications that meet the requirements of the outputs of the core asset development process described earlier[7][17]. Product builders give feedback about the core assets as they build the applications, so that the core asset base remains high quality and up to date[7].

3.3 Management and organizational management

Pohl et al. [17] describe product line thinking as 2 layers of processes, the domain engineering and application engineering layers. Northdrop et al. [7] add another layer called management.

According to Northdrop et al. management plays a critical role in the success of a product line, an organization must ensure that activities are given resources, coordinated, and supervised. Management must be strongly committed to the software product line effort on technical, project, and organizational level[7, p. 22–23].

Organizational management identifies production constraints and ultimately determines the production strategy. The purpose of organizational management is to create an organizational structure that makes sense for the enterprise and is responsible for appropriate resource allocation, for example in terms of trained personnel[7, p. 22–23].

Northdrop et al. define organizational management as the authority responsible for the utmost success and failure of the product line effort, organizational management determines the ways of funding that enables the evolution of the core assets[7, p. 22–23].

We define organizational management as the authority responsible for the ultimate success or failure of the product line effort. Organizational management determines a funding model that will ensure the evolution of the core assets and then provides the funds accordingly. Organizational management orchestrates the technical activities in and iterations between the essential activities of core asset development and product development. [7, p. 22–23]

In addition to the previous, management iteratively orchestrates essential activities of core assets development and product development. At a sub-management level, technical management makes sure that the core asset development and product development activities follow the processes defined for the product line, collecting data to track it's progress. Technical management decides on the production method and provides the project management elements of the production plan. [7, p. 22–23]

Management makes sure that the operations previously described and the ways of communication of the product line effort are documented in an operational concept. At the organizational level, management mitigates risks that might threaten the success of the product line. [7, p. 22–23]

One of the most important functions that the management process does is that it creates an adoption plan that describes the desired state of the organization and a strategy for achieving that state. The desired state being one where the organization is able to routinely produce products in the product line. [7, p. 22–23]

Technical and organizational management also contribute to the core asset base by making product line management artefacts like schedules and budgets available for reuse. [7, p. 22–23]

Leadership is required in order to keep the organization pointed towards the product line goals to ensure success, especially during the rough early stages of the product line. An individual or a group should be designated to fill the product line management role and act as a product line champion, they should preferably be a visionary leader. Management and leadership are not always synonymous. [7, p. 22–23]

While the book by Pohl et al. [17] does not list management as a core process, they do write the following towards the "Organization" part of their book: "For the successful introduction of the software product line engineering paradigm, organisation aspects are as important as the technical aspects".

It is also worth noting that building a business case and the transition to software product line development are not easy steps for an organization, and requires investments that have to be determined carefully[17, p. 394]. There are multiple strategies for the transition listed in the book Software product line engineering [17] which are outside the scope of this work.

4 Case study

A case study is an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident.[40, p. 13]

In this chapter the author goes through the usual steps of conducting a case study according to R. K. Yin, A case study involves several steps including designing and conducting case study, collecting as well as analysing data and composing a conclusion[40]. In addition to the previous, the methodology of the study will also be included.

4.1 Backround

This section is used to describe the background of this thesis, for the reader to better grasp the context and get more out of the discoveries of this study.

4.1.1 Luonnonvarakeskus as an organization

The Natural Resources Institute Finland (Luke) is a research organisation operating under the Ministry of Agriculture and Forestry of Finland[41]. Some of the Luke's DIGI unit's tasks, are to upkeep most of the in-house software systems and internet sites that are used by the organization of Luke, and to develop, procure and upkeep systems that help researcher's daily work. Although a lot of other software related work is divided into different units and researchers inside Luke.

Luke was established at the start of 2015, it was a product of a merger of 4 different organizations: The Agricultural Research Centre of Finland (MTT), the Finnish Forest Research Institute (Metla), the Finnish Game and Fisheries Research Institute (RKTL), and the statistics services of Tike, the information centre of the Ministry of Agriculture and Forestry[41], illustrated in figure 4.1. The results of the merger can still be seen in the organizational aspects of Luke, as it's organization is very dispersed, diverse and patchy.

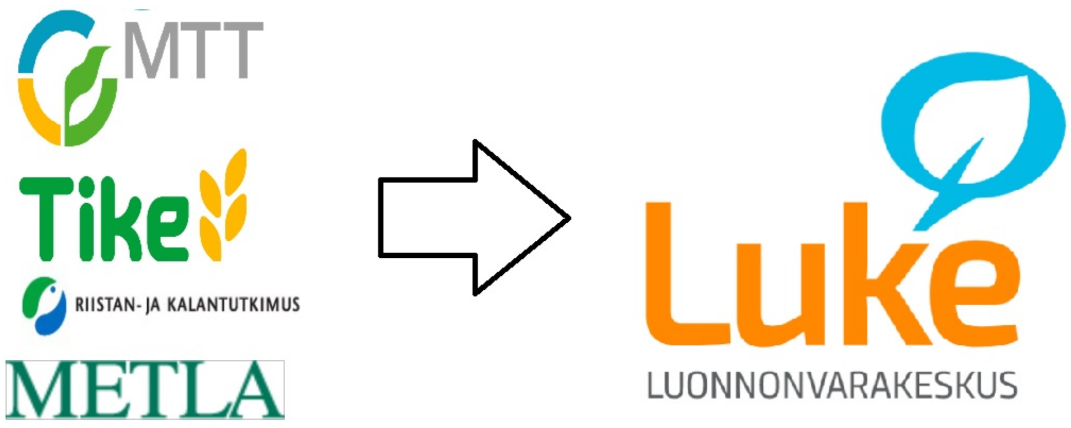


Figure 4.1: Natural Resources Institute Finland being formed from 4 different organizations

During my 2 years of working for Luke, while being hunkered down in a project called Tikal, I slowly gained understanding of how my unit worked, and where I was placed in the monstrosity that is the Natural Resources Institute Finland. During my time working at Luke DIGI unit, the information sharing aspect of things kept coming up: Working methods and practices were not getting shared much within DIGI, use of software languages was not streamlined and policies were unclear on their use, policies regarding data storage were also not clear. A lot of very basic information was retained within tribes of the unit, and since most of the developers of the unit worked alone with little interaction with the rest of the developers, there was constantly repetition of the same information work within each tribe.

One of the biggest dangers of this kind of organization was the possibility of doing the same thing twice: A system that is needed by two different projects might be labeled with different names and technologies without the organization as a whole identifying the system as a common need for projects, if the projects don't communicate to each other about their activities, the margin of error becomes high in this regard. There was also a tendency to have trouble identifying what had been done during previous projects, especially since projects leads were often people with little technical experience. These tendencies were amplified by the fact that Luke came to be from multiple different organizations and their software systems.

So there was a real need for increasing spontaneous interactions between developers as well as increasing the ability identify systems that could be made into commonly used services.

4.1.2 Jira

Jira is a issue tracking system, at Luke it also functions as a requirements management system, as the requirements are documented as jira issues, most often user stories and simple tasks. Most of the work of DIGI unit was documented into Jira as jira issues, issue is a term used for all item entities documented in Jira, such as epics, user stories, tasks and bugs.

Jira is also a really nice collaboration tool also due to it's visual aspect, items can be seen in views called boards, which show the items as squares in different columns based on their readiness in the development cycle. Developers usually work on a single or more items at a time, and update the status on the items as they make progress.

At the start and during 2021-2022 there was a big move at Luke DIGI unit to move requirements documentation from legacy sites to Jira and Confluence, Confluence being a team-based documentation site for storing information in page-based format. A big motivator for the move was the teams at DIGI being able to move towards a more team-based collaboration style, to enable team-based work instead of individual work in future development, as well as to enable agile ways of working.

My team being the pilot for team-based agile work, I was given the task of administrating our Jira environment, which led to the idea of increasing team collaboration through Jira as well as making software systems more comprehensible, which eventually lead to me being able to do this thesis' case study in collaboration with Luke.

In Jira, and at Luke DIGI, a project is usually started by creating a architecture document, but turning that document into an action plan to figure out where to start, is a whole other hurdle. In Jira, the order of actual development work is in a best case scenario started by drafting up the biggest user stories as epics. Epics are collections of user stories[42] that are related to one another in concrete manner and would not really make sense to include separate from each other. When all the user stories are implemented they create somewhat complete pieces of functionality. This enables the lead of a project to think in a manner of: What pieces of functionality does this system get composed of?

A agile epic in Jira usually has a title, which contains the user story of the epic, and a epic name, a field which can be used to describe the epic in a more compact way, I'd call this field the feature field as it can be used to describe the feature that will complete once the epic is completed. An epic in Jira also has a field which describes if the item is dependant on another item, for example if the development of one one item blocks the development of the epic until that item is complete. These dependencies can be used to further structure the development process and make sure that components are developed in the correct order.

All the previously mentioned terminology is unique to the product Jira and does not represent agile development as a whole.

Many of the projects within DIGI unit's Jira organization have only partially been planned using epics, user stories and tasks, and the quality of the execution of this documentation varies a lot by project, as not everyone within the organization has previously worked with agile development. And since many of the project teams consist of a single developer working along with a product owner or a project manager, the ways of using Jira pass along slowly.

Epics are one of the main ways of structuring a project[43] in agile development. The case study of this thesis focuses on epics as a unit of a software systems structure, an epic can be used to describe a component within a larger software system.

4.2 Case description

Initial plan and hypothesis

The initial vision of this study was to make a model that would link Jira items together in a way that anyone could edit, the model would be a 2D map with pieces that could be moved freely - additionally the model would show the state and relations of the components(epics). To describe the 2D map resulting from the contents of a project the word ”view” is used.

The initial hypothesis going into this case study was that if visual clarity and linking of issues between projects was added to the software development environment, information sharing and identifying repeated software components would get significantly easier. There was also an assumption that the visual environment would make designing software and drafting epics for the project more enjoyable and comprehensible.

One of the model's key features would be it's intuitiveness, visual graph with information color coded in would make it's readability very easy when compared to a simple list of text-based requirements.

One of the key difficulties in planning for this models implementation was it's environment, the uncertainty whether or not it was possible to add the model and it's views into Jira or a environment that would easily be used alongside Jira. The hope being that there would at least be a service such as Miro, a online whiteboard tool, that could reference Jira issues from their service and provide a browser based service for hosting the view. There was also the option of coding and add-on to Luke's Jira from scratch, but it was deemed too laborious to implement.

This solution would still suffer from the arduous task of having to switch browser tabs between Jira and Miro when viewing the structure of a project, so the obvious solution was to find ways to implement the model inside Jira through a add-on made by the author or by looking for a solution that had already been made somewhere else.

Following images are from the drafting phase of the case study, the first image is a screen capture from Jira showing the colour coded epics that are listed in a sidebar, to the author's experience, this is the most common way a user of Jira is going to see their own epics on a project. The second picture is the initial vision of the model view that was going to be implemented.

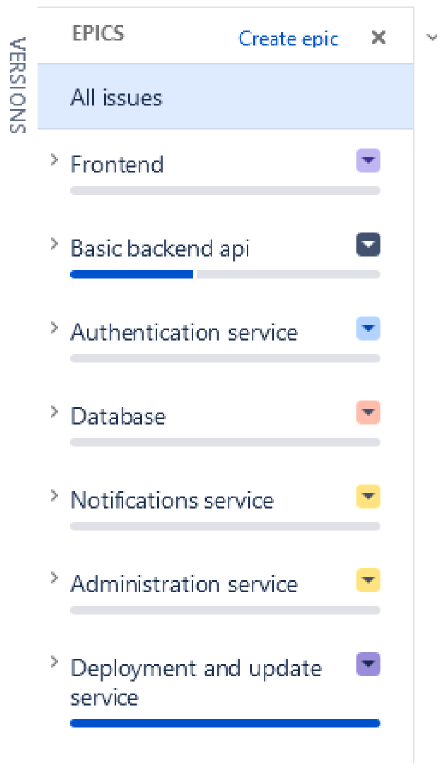


Figure 4.2: The native sidebar in Jira showing epics in a list as well as epic progress

Key features of the planned model

The key features that were going to be implemented in the model of the initial research plan were:

Visuality The model should be compromised of individual square shaped boxes that could be moved independently of each other, the boxes would be epics, and when clicked, the boxes would show all the user stories grouped as similar boxes below the square of the epic that was clicked.

Modifiability The view should be modifiable by anyone with access to the environment in which the model was hosted.

Colour coding The epics of the model view should show some kind of colour coding to indicate how ready the epic would be to being finalized into the production environment.

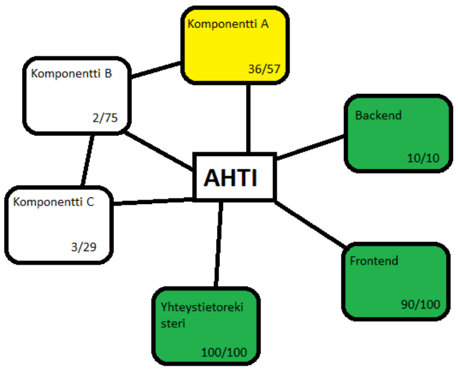


Figure 4.3: Initial conceptual draft for a graph-based view for a system called Ahti showing the epics of the project as movable cells

The readiness of a epic would never be concrete as the epic's user stories would increase, reduce and their contents and scope would change. Despite this having an intuitive percentage-based colour indicator of completeness would help the viewer of the model get a fast and easily comprehensible idea on how well a project is doing.

Non-colour based progress indicator In addition to colour coding based on progress, there should at least be a progress bar or a text-based indicator of progress, the text based progress bar should indicate the amount of completed user stories and tasks inside an epic as either a percentage amount or as a number as in the amount of completed tasks and user stories with the total amount of user stories and tasks to it's side, with a forward slash (/) separator in between the numbers.

Dependency lines The default Jira issue manager allows the user to add dependencies to epics, stories and tasks. [44] The model should present these dependencies as visual lines or some other kind of visual connector element in between the squares of the view that have some kind of a dependency in between each other.

Common items across projects One of the features that makes the model work, would be the ability to use the same epic in many different contexts. The reason for this being that if we can see the same epic and it's progress in multiple project views, that epic can represent the progress of a common component for all the projects that need it and benefit from seeing it's existence and progress!

From the start the plan was to do semi-structured 30 minute interviews with just around 5-10 participants, the role of the interviewees being product owners and software developers, it was also thought that adding system architects to interview would add and interesting viewpoint to the study. During implementation the interviews were extended to 2 separate 30 minute interviews.

The method of interviewing was chosen to be semi-structured interviews. While it was fairly clear on what kind of information was needed for the study, there was still a strong possibility that new ideas and takes could emerge as the interview would go on. Open interviews were considered but deemed too hard to direct in terms of subject and time.

4.3 Methodology

A research methodology is the strategy or approach to finding and solving a specific research problem. This section goes through the decision making process of choosing the research methodology for conducting the research of this thesis and the reasoning of each of the decision points used. The implementation of the interview method is also described.

Research question

The research question formed to be

How can a high-level real-time visual tracking model assist in

bringing product line thinking and increasing information sharing within an organization?

The original research question was broken down into subcategories, The interviews document of the study was aimed at answering these subcategories.

- In what ways can a visual Jira architecture model of high level abstraction help software engineering?
- Could a Jira architecture model of high abstraction level help in information sharing for both a project and an organization?
- Could a repeatable Jira architecture model of high abstraction help with developing repeated software components?

The organization of the third sub question refers to the organization of Luke's DIGI department, the high-level abstraction model refers to the visual model implemented during the case study.

4.3.1 Research outline

After the research problem had been identified, the approach was to create a model to help software development and to interview the users of the model in order to see if the problem could be answered with a possible solution. The aspects and research approach of this thesis was identified and categorized by using a proposed decision-making structure by Wohlin, C. and Aurum, A. [45] The focus of their article is to aid in selecting a research design in empirical software engineering research [45].

This section uses the previously mentioned research decision-making structure by Wohlin, C. and Aurum, A. [45] to describe and categorize the methodological approach in which the research of this thesis was done. Each of the 8 points is further elaborated on to describe the approach that was used. Figure 4.4 illustrates this decision making process.

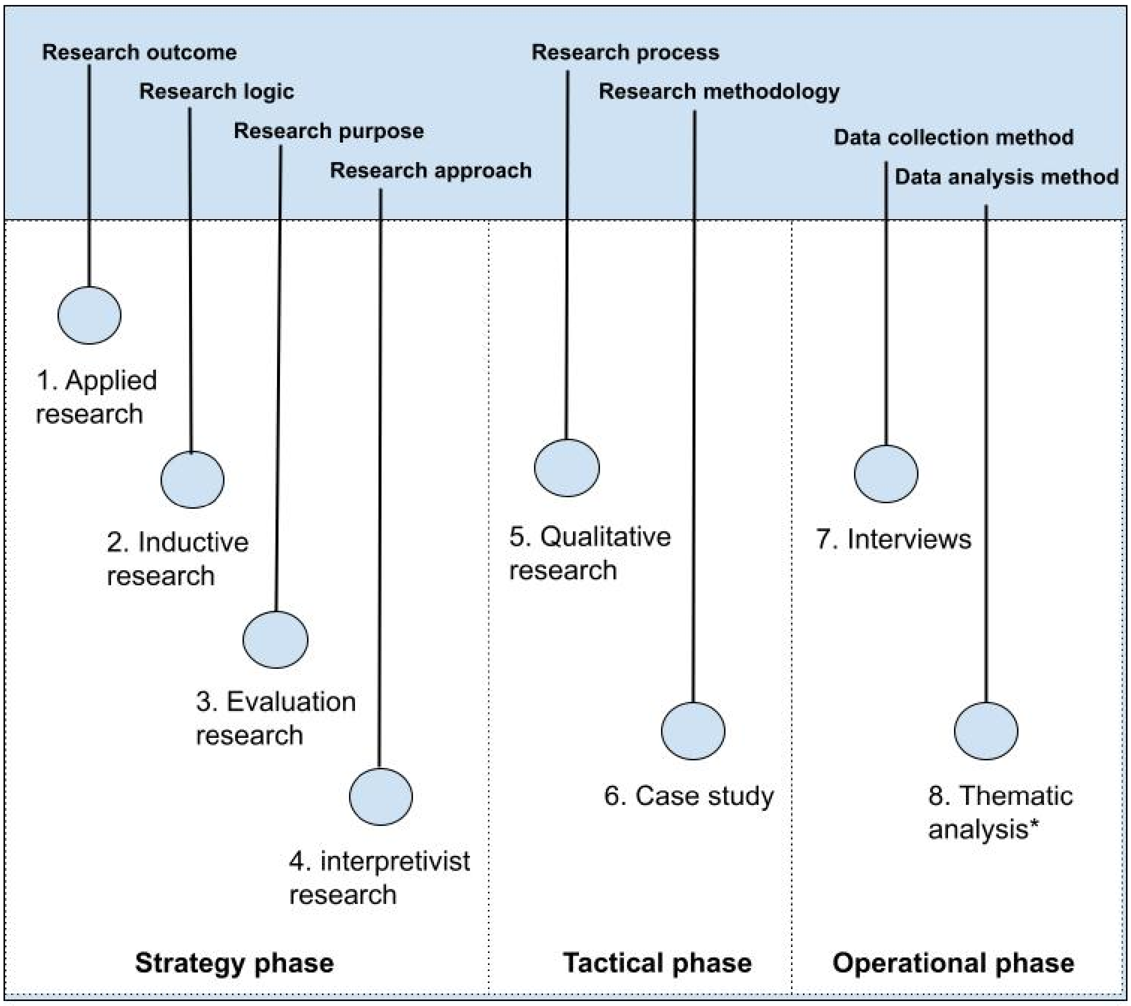


Figure 4.4: Research outline of this thesis drawn with the methodology of Wohlin, C. and Aurum, A. [45]

1. Research outcome: Applied research

The outcome of this thesis' research is applied research, while basic research is about finding a problem and studying it for the general good, applied research is more focused on finding a timely solution in order to improve an existing practice[46][47], and the visual model implemented for the case study is an attempt of finding that solution.

2. **Research logic: Inductive research**

   Inductive research is described as fast science[48]. In inductive logic the truth of the premises provides some[49] or general support[45, p. 1434] for the truth of the conclusion, moving from individual induvidual observations to statements of general patterns[47]. In this thesis, there are many premises which the case study aims to weave into coherent statements and solutions. The specific conclusions or theories made in this research are achieved through themes identified from the interview notes and presented in the discussion chapter.

3. **Research purpose: Evaluation research**

   The research purpose for the research part of this thesis can be considered evaluation research, as it describes the author using tools of research to describe, explore and assess the needs of different groups of users of software in order to improve it's planning and effectiveness[45]. The common trait of all evaluation research is that it aims to be useful and used, either directly, quickly or as an increment to some other, larger body of practical knowledge [50].

4. **Research approach: Interpretivist research**

   Interpretive research assumes that our knowledge of how reality can only be gained through social constructions such as language, shared meanings, documents as well as tools. [51]

   The research approach in this thesis is purely Interpretivist since imperative research aims to understand the deeper structure of the phenomenon much like this thesis. Imperative research assumes that people add their own meanings to things as they interact with the world, and imperative studies aim to collect and make sense out of those meanings, the case study of this thesis aims to do just that. Imperative studies abandon the possibility of a factual truth and seek a relativistic although shared understanding of the phenomena. [52]

5. **Research process: Qualitative research**

   In qualitative research the researchers often try to understand the perspective of their research subject, those individuals who are inside a process often see things in a different way than those outside of it, gaining these insider's perspective allows the researchers to understand the world in a new ways. Qualitative data typically includes direct representations of subjects' thoughts or actions. [53]

   Qualitative research is subjective in its nature and is influenced by the philosophical preferences of the person performing it. [47]

   In this thesis the author collects qualitative data such as interviews and participant observations and that categorizes the thesis case study as qualitative research.

6. **Research methodology: Case study**

   Research methodology is a combination of research methods, processes and frameworks[45]. Case study research is a research inquiry that employs multiple methods of collecting data in order to collect information from multiple sources with the purpose of investigating a phenomenon its natural setting[54].

   In a case study, The boundaries of the phenomenon are not clearly evident at the outset of the research and no experimental control or manipulation is used. [54]

   The research methodology of this thesis is case study, as it's effectiveness is not measured as is in design science[55], even though an artefact is produced. It's also not action research as it's not systematically studying the problem, people, and the organization, even though the research problem does very much exist in real life[56]. This study simply investigates the phenomena in it's natural setting, collecting data from multiple sources.

This research methodology is still very close to being action research. The case study involves collecting data through literature review to understand the phenomenon of repetitive software components and reflecting on the Natural Resources Institute Finland organization as a problem environment(natural setting). A model was created based on what has been learned to further gain understanding, and than interviewing users of the model in the environment of Luke in order to understand it's impact and usability.

7. **Data collection method: Interviews**

   The data collection method is interviews, the interviews are recorded in either audio or video format. More details about the interviews of this case study are described under the case implementation section.

8. **Data analysis method: Adapted thematic analysis**

   The data analysis method of this study is an adapted version of thematic analysis. Qualitative analysis methods such as thematic analysis focus on making sense or understanding a certain phenomenon[57]. During the data analysis, new insights can be identified which may require gathering more data[45].

   In thematic analysis verbal data is transcribed word for word into written form in a rather time-consuming manner in order to be able to later analyse the content[58]. There are 2 types of themes that can be associated to the transcribed words, semantic and latent. Semantic includes looking at the meaning of the words, and grouping those words together. Latent themes try to include the meaning of the words in their context and interpret the meaning of these contexts [59] [60].

   Themes are patterns within data, and thematic analysis is about identifying those patterns. Themes organize and describe the collected data in (rich) detail. Thematic analysis at the latent level goes further than just analysing the semantic content of the data and starts to identify various aspects of the research topic including ideas, ideologies, assumptions and conceptualizations that are theorized as shaping or informing the semantic content of the data. [58]

In this case study a much simpler thematic analysis method, dubbed by the author as adapted thematic analysis, will be used. The key difference of this method to the standard thematic analysis is that instead of transcribing all of the interviews, the extensive notes of each interview session will be used to produce themes. The notes are written both during the interviews and after listening to the interview recordings later on.

Main reason for this approach is the aforementioned lack of time, transcribing all of the interviews takes an excessive amount of time from the researcher, and the achievable added quality to this research is not deemed worth the amount of time transcribing the interviews would take. Identifying themes is ideally done by multiple researchers[45], in this study, identifying themes is solely done by the author.

The identified latent themes of each of the interview question will be compiled and summarized under each of the interview questions, most dominant theme being written first.

4.3.2 interview method and implementation

Interview method

The semi-structured interview method mentioned before in the case description was chosen by process of elimination. Doing open interviews would have been too time consuming for the interviewees and the author due to daily schedules being limited, while a structured interview or a survey could have left a lot of details and depth missing, details that would be hard to predict and include in it's structure.

One of the major downsides of the open ended interview method was that open ended interviews should have at least an hour of scheduled time, which the author couldn't afford, they are also quite straining on the person doing the interviews as they require the interviewer to have longer mental and physical transition times in-between the interviews. [61, p. 165–166]

Semi-structured interview is an interview method where the interviewer has a standard wording of the questions to be asked, but after a unclear answer has been given, the interviewer can ask further questions freely to further understand the interviewees view. [62] The questions are open ended and are based on the topics that the researcher wishes to cover, the open-endedness of the questions allows the people involved in the interview to discuss the topics in varying depth.

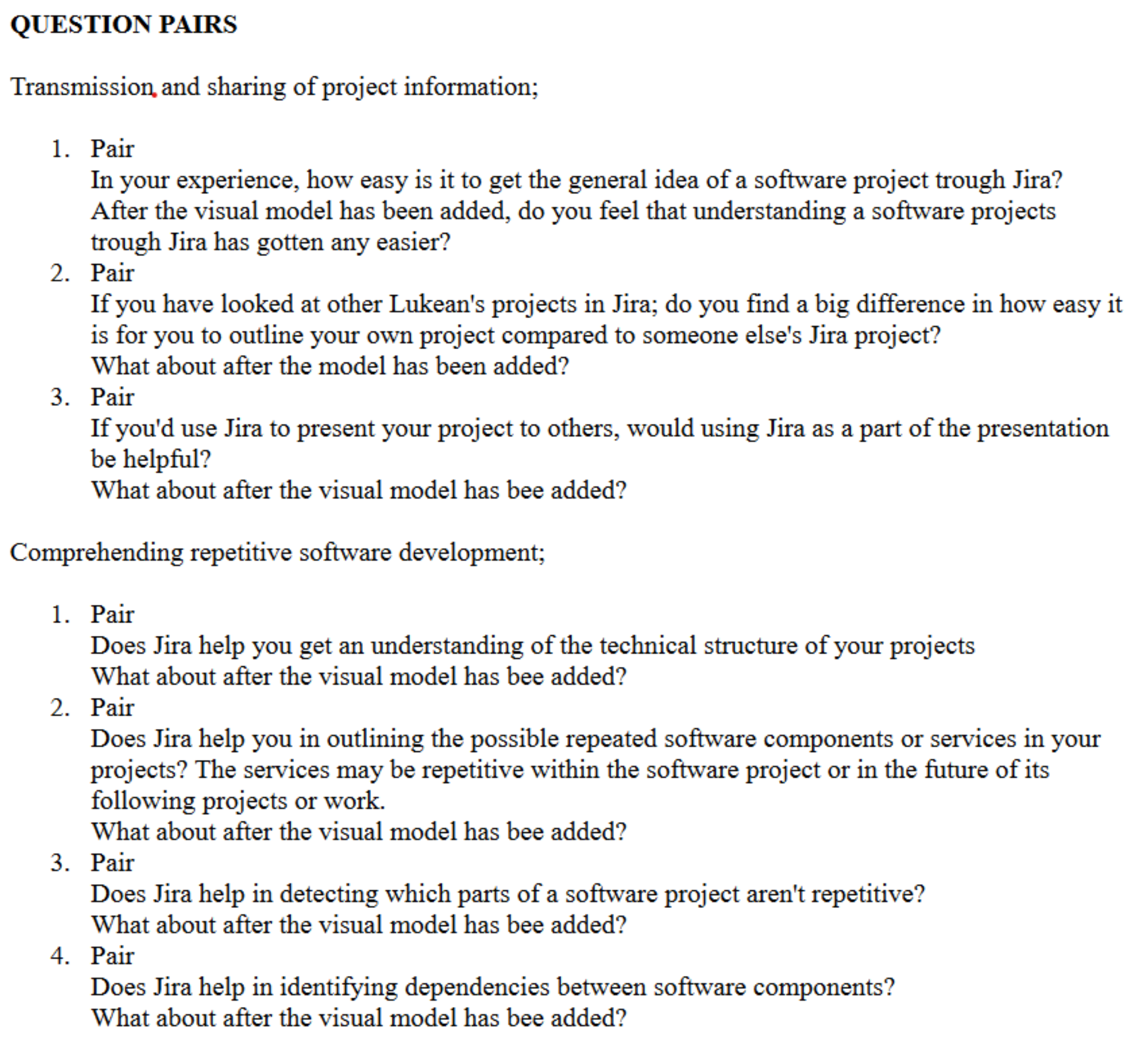
During a semi-structured interview, the interviewer can also ask the interviewee to elaborate on the answers given or to follow a line of thought that that was introduced by the interviewee. [63] This makes it possible to acquire emergent insight into the topic from the interviewee, insight that wasn't planned or understood when the question structure was being drafted.

The flexibility aspect of interviewing was also a major aspect of choosing the semi-structured interview method. The ability to ask key questions in the same, slightly varied way each time and being able to probe for further information was deemed a major upside, albeit the method being more limited than a unstructured interview[61, p. 111].

The weakness of structured interviews is their inflexibility making it really hard to get deeper into the subject to find unpredictable discoveries, also requiring substantial planning beforehand[64]. The same goes for questionnaire surveys as they're less flexible than semi-structured interviews [65].

Interview question structure and goal

The following picture is the structure of the question pairs asked during each of the two interviews. This section will explain the structure and purpose of the interview questions.



QUESTION PAIRS

Transmission and sharing of project information;

1. Pair
   In your experience, how easy is it to get the general idea of a software project trough Jira?
   After the visual model has been added, do you feel that understanding a software projects trough Jira has gotten any easier?
2. Pair
   If you have looked at other Lukean's projects in Jira; do you find a big difference in how easy it is for you to outline your own project compared to someone else's Jira project?
   What about after the model has been added?
3. Pair
   If you'd use Jira to present your project to others, would using Jira as a part of the presentation be helpful?
   What about after the visual model has bee added?

Comprehending repetitive software development;

1. Pair
   Does Jira help you get an understanding of the technical structure of your projects
   What about after the visual model has bee added?
2. Pair
   Does Jira help you in outlining the possible repeated software components or services in your projects? The services may be repetitive within the software project or in the future of its following projects or work.
   What about after the visual model has bee added?
3. Pair
   Does Jira help in detecting which parts of a software project aren't repetitive?
   What about after the visual model has bee added?
4. Pair
   Does Jira help in identifying dependencies between software components?
   What about after the visual model has bee added?

Figure 4.5: The main interview question structure

The interview document that was used had two sections. The first section was for giving the interviewee a simple verbal description of what the study was about and reading a very brief description of what a software product line is and the purpose of the visual model that was being developed.

The first section also had personal background questions, questions related to documentation and communication at Luke and a list of themes that were read aloud to the interviewee. The list of themes was included to give each interviewee a rough idea on the theme of the questions that they were being interviewed on. This first section of the document was only read during the first interview.

The list of themes that were read aloud to each interviewee

- Information sharing
- Comprehensibility of software projects
- Sharing information at Luke
- Detecting repeated software components
- Increasing software product line thinking at Luke
- Assisting development and design of software

General questions related to information sharing at Luke

- On what platforms is the technical documentation of your projects?
- At Luke, how well do you feel information is being shared between projects?

The purpose of the general questions was to get data on the overall situation at Luke, and to give context to the answers of the more specific question pairs later on.

The second section of the interviewing document(figure 4.5) had 7 question pairs, three related to information sharing in projects, and four related to detecting and comprehending of repetitive software development.

The purpose of the first three questions was to gain data on how the project view in Jira is understood in terms of content readability. A lot of the work at Luke is done through Jira but without really understanding what makes a requirement management tool good on a UI design level and how different people can use it in their daily work. There's also a bigger cross-teams dynamic with content readability, which this study tries to shed light in. As for the last four questions, the purpose was to gain further understanding on how the content readability and visibility in Jira affects the understandability of technical details, as well as the detection and structural planning of repetitive software components.

Interview implementation

Two rounds of interviewing were done to set a baseline for the interviewee's answers, interviewees were interviewed during the first round without them having heard or seen the model that was being implemented. The purpose of this was the assumption that if the author implemented the model first and asked the interviewee's opinions afterwards, the interviewee's insights could have been influenced by their relationship with the author and the author's role in the organization.

For example, with the question "In your experience, how easy it is to get the general idea of a software project through Jira?", it was deemed helpful for the interviewee to have a already set a uninfluenced baseline that they could remember and follow set in the second interview. It would also be easier to do the data analysis later on, since the answers of the first interview cycle could be compared to the answers of the second interview cycle.

The interviewees had a period of two weeks in between the interviews to use the visual model that had been added to the software development environment. The second interview had slightly modified questions which were essentially the same questions but were started with the phrase "now, after the model has been added".

Three of the interviews were conducted in person at one of the offices of Luke, the nine others were done through Microsoft Teams. In-person interviews were recorded using the author's phone, and recorded only audio, while online interviews were recorded using the recording functionality of Teams as a recording tool and they included both audio and video footage.

All the interviews were spoken in Finnish. When interviewing online, the interviewer always had their camera on so that non-verbal cues could be given throughout the interview, to better direct the interview and to give feedback to the interviewee. Participants were not required to have their camera on during interviewing simply to make the interviews more comfortable.

The length of one interview session was around 30 minutes, with most interviews of the second cycle lasting under 30 minutes. All interviews were done during the last three weeks of February 2023.

The study had a total of 6 participants for interviewing, selected non randomly, the participants were the author's colleagues working within the same department of Luke. The roles of the interviewees being mostly software developers (3) with the addition of a product owner, a project manager and a systems architect. One of the developers was working as a visiting consultant. The role composition of the interviewees is illustrated in figure 4.6.

It was deemed important to have input from multiple types of professionals of the software development field to increase the likelihood of gaining diverse data points and emergent discoveries from the study.

Time was the main limiting factor of the study, and such the amount of interviewees was limited to only a sampling of 6 people.

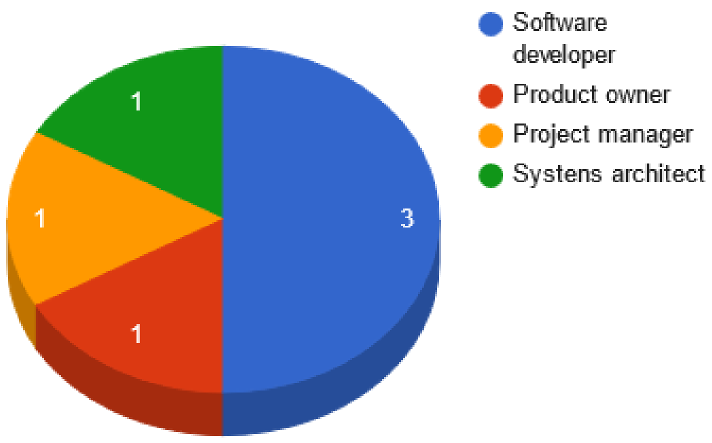


Figure 4.6: The number of participants sectioned by their Job descriptions

Most of the notes of the data collection process were written after the interviews using the recorded audio and video recordings, however, during the interviews, a small amount of notes was written to better direct future interviews, as well as to document discussion that happened before or after recording had been started.

Using follow-up questions, interview challenges

The author's interviewing style was very in line with the standard semi-structured interview method, non-verbal cues were given to give the interviewees reinforcement to keep talking about the points that they were making, questions were asked both to achieve clarity to answers that were deemed not clear enough and a significant amount of questions were asked in order to gain insight into different divergent talking points that the interviewees often drifted off to, follow-up questions were also used to gain clarity on theories and ideas that the interviewees often had.

If a interviewee gave a very punctual answer to a question, follow up questions were not asked.

As for some unconventional interviewing means, sometimes the author would make statements around the topic that the interviewee was making to clarify the interviewees points and to see if the interviewee would agree with the statement, some quotes of such interviewing will be provided in the 'results' section.

Often when interviewing, the participants would drift off from the question that was being asked, when this happened, follow up questions would be asked regarding the topic that the interviewee had drifted off to, and after the follow up conversation was satisfied, the original question would be asked again to clarify the participants answer.

Unfortunately, a few of the answers remained vague due to the author forgetting to ask the original question again due to the interviewee drifting off multiple times while the interviewer would forget whether or not the question was answered within the conversation. This occurrence is illustrated in figure 4.7 below.

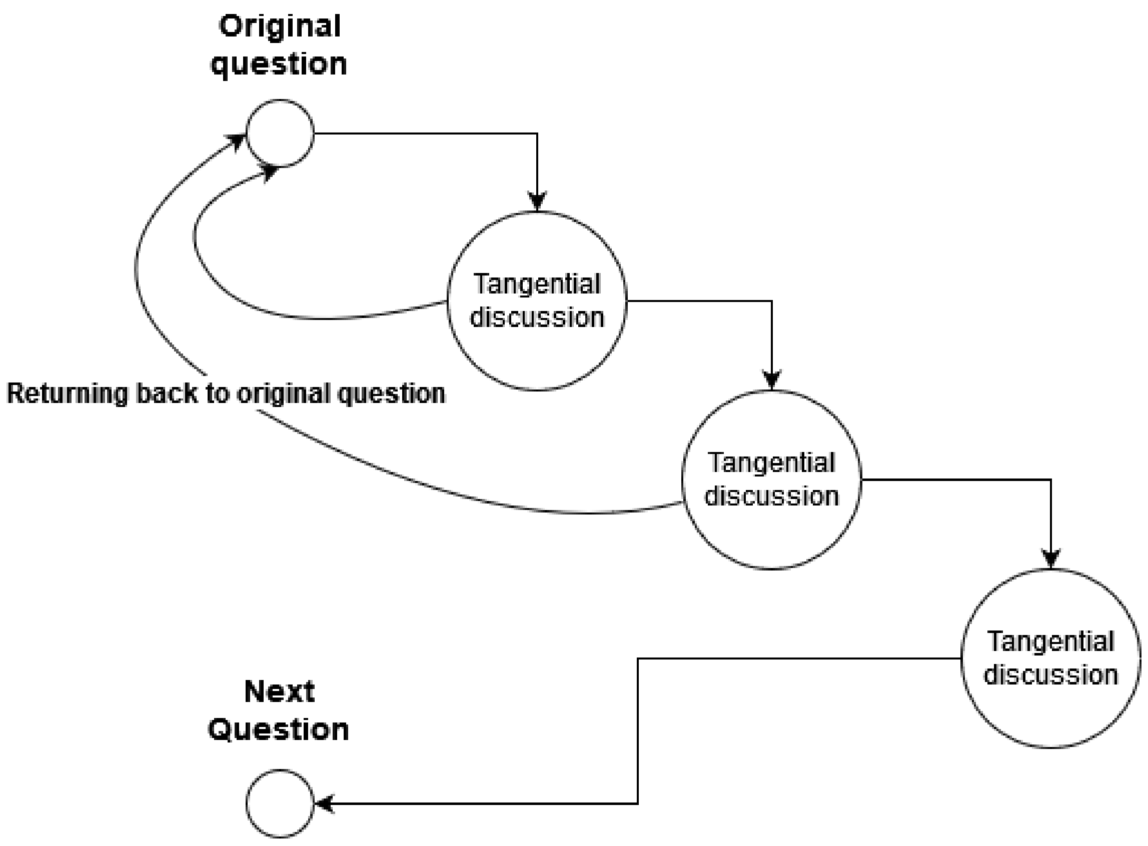


Figure 4.7: The interview challenge of the interview drifting off from the original interview question multiple times and the interviewer eventually moving on to the next question without clarifying the original answer

Another challenge that was found was the interviewees skipping questions on the technical section of the interview, interviewees did not know the question structure so they would answer upcoming questions during the first question of the second section, meaning that in addition to the interviewer being able to ask questions in a varying order, they also had to keep track of which questions had been answered by the interviewee in advance.

During a semi-structured interview, a lot of attention has to be paid to direct the interviewee to give a clear answer to the original interview question.

Other challenges

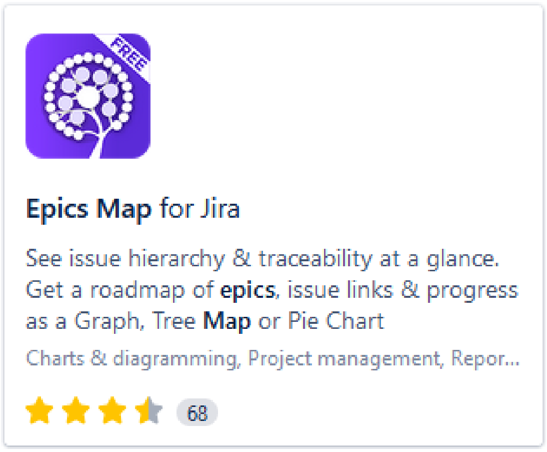
With the last question pair of the interview document, participants would often confuse software development dependencies and item correlation, resulting in slightly more positive answers to the last question pair of the second interview cycle.

There were also unfortunately non-responses to some of the interview questions, even after follow up questions, a non-response at the item level is described as a ”don’t know” -answer[66], which was sometimes repeated even after the same question was asked again from another viewpoint.

The challenges during the interviewing phase only had minor impact on the quality of collected data. The interviews were successful despite this being the author’s first time conducting interviews.

4.4 Case implementation

Around the time that the implementation of this model started, the author had been as a Jira administrator for Luke for about one year, the solution for implementing the model was found from the Atlassian marketplace for Jira add-ons, Atlassian being the owner and developer of Jira. Figure 4.8 shows the card link to the add-on used for this study.



Epics Map for Jira

See issue hierarchy & traceability at a glance. Get a roadmap of **epics**, issue links & progress as a Graph, Tree **Map** or Pie Chart

Charts & diagramming, Project management, Repor...

68

Figure 4.8: The add-on used to build the visual model, Epics Map by Herocoders, pictured in the Atlassian marketplace

The add-on worked simply by downloading it, after downloading, projects that were within the author's Jira organization were automatically given a button in the sidebar, which leads to the Epic map view, shown in figure 4.9.

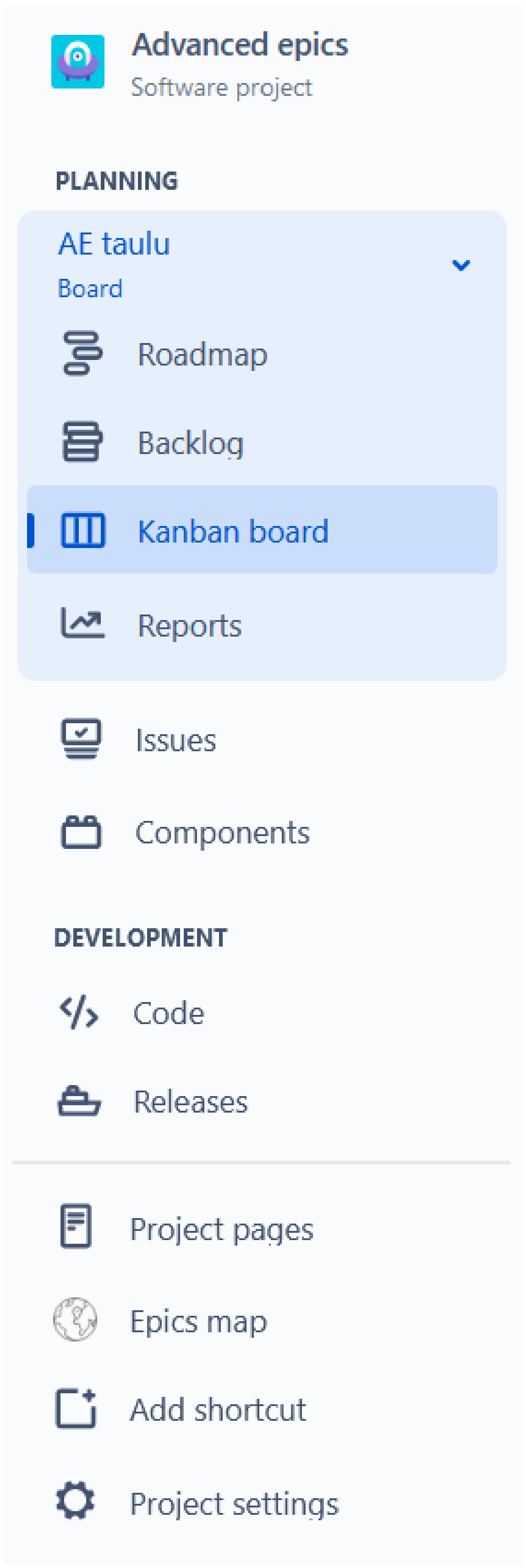
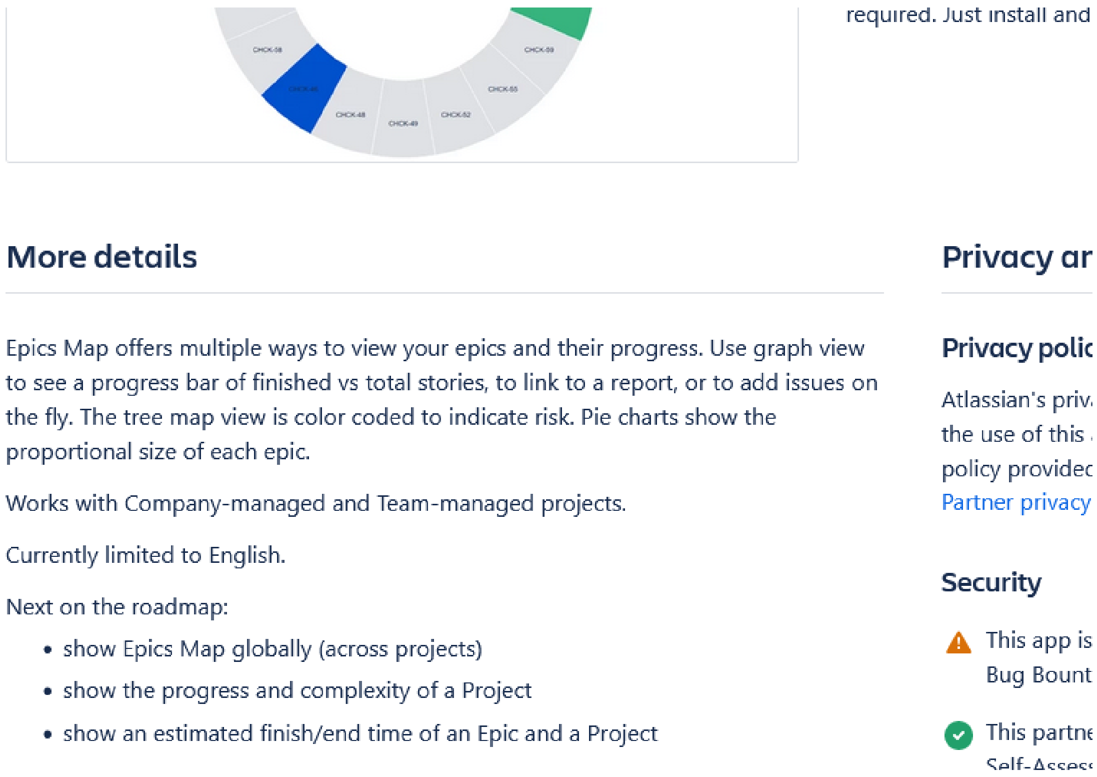


Figure 4.9: The Sidebar of the Jira project view, after downloading the Epics map add-on

The add-on lacked two features that were vital aspects of the desired model's vision. One of these two is being able to globally see the same epic in multiple projects. However the Atlassian marketplace overview for Epics map states that the roadmap of Epics map has global epics as a goal for future development[67] during the time of writing of this thesis (figure 4.10).



required. Just install and

More details

Epics Map offers multiple ways to view your epics and their progress. Use graph view to see a progress bar of finished vs total stories, to link to a report, or to add issues on the fly. The tree map view is color coded to indicate risk. Pie charts show the proportional size of each epic.

Works with Company-managed and Team-managed projects.

Currently limited to English.

Next on the roadmap:

- show Epics Map globally (across projects)
- show the progress and complexity of a Project
- show an estimated finish/end time of an Epic and a Project

Privacy ar

Privacy polic

Atlassian's priv
the use of this
policy providec
Partner privacy

Security

This app is
Bug Bount

This partne
Self-Assess

Figure 4.10: The future roadmap plans of Epics map, Atlassian marketplace

Another key aspect of the original model's vision was that dependencies could be visually seen as lines between the components that were dependant on each other in some way. The model only shows visual lines from an epic to a project, but not from one epic to another, even when a dependency is set between two epics, the current way of showing dependency lines shown in figure 4.11.

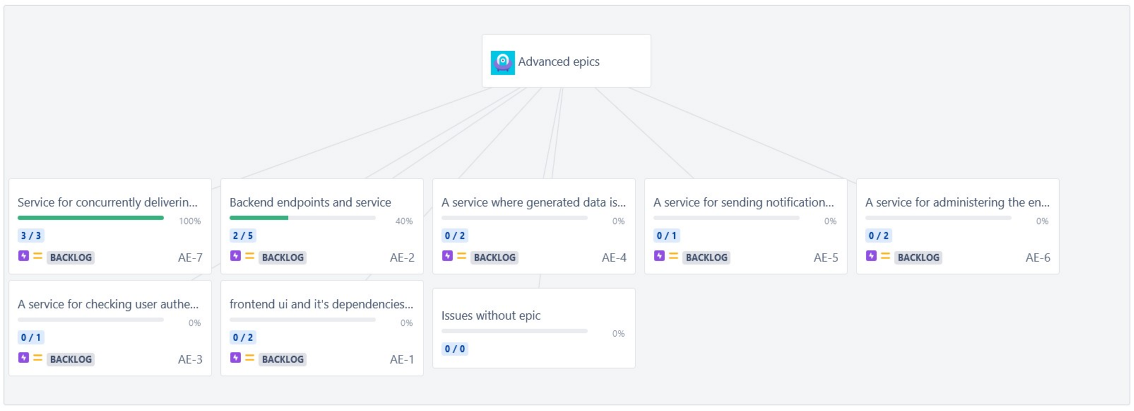

Advanced epics
Service for concurrently deliverin... 100% 3 / 3 BACKLOG AE-7
Backend endpoints and service 40% 2 / 5 BACKLOG AE-2
A service where generated data is... 0% 0 / 2 BACKLOG AE-4
A service for sending notification... 0% 0 / 1 BACKLOG AE-5
A service for administering the en... 0% 0 / 2 BACKLOG AE-6
A service for checking user authe... 0% 0 / 1 BACKLOG AE-3
frontend ui and it's dependencies... 0% 0 / 2 BACKLOG AE-1
Issues without epic 0% 0 / 0

Figure 4.11: Epics map showing dependency lines to the author's test project ”Advanced epics”

In order to convince Luke DIGI unit to pilot this view a test project was made using the Epics map add-on, the test project consists of 6 epics with some of them including users stories, one of the epics in the view has all of it's user stories completed and one having two of it's five user stories completed. The progress bars clearly show how many user stories has been done under each epic.

The author stated early on into building this test project that while epics tend to grow in size during development as additional user stories are added, the viewer can still get an idea on how ready or how well oof a project is into it's planning. For example, if a epic has 90 out of 100 stories done, the epic must be ready or nearly ready simply by 10 of those stories possibly being related to requirements that might never be implemented or that might already be obsolete. Or an epic that has a single or a few user stories under it, might still be in planning.

The model contains a lot of inferable information, that can be understood by professionals from the context of a project.

Workshops

Right after the first interviews were done, the author held workshops for the interviewees that wished to participate, there were three projects that the interviewees had currently under development; Tikal, Näytehallinta and Suomu.

Näytehallinta and Suomu had their development just started and the projects had very few epics under them, and the epics which had been written had very simple titles such as ”integrations”, they weren’t really epics as the were not large user stories, more like categories for task to be placed under. The workshops held for Suomu and Näytehallinta had all of their participants participating for the workshops two for Suomu, one for Näytehallinta. All of the participants were working remotely and screen sharing was used to collaborate through Microsoft Teams.

Together with the participants of Näytehallinta and Suomu we managed to make the projects epic structure include a much more complete structure of the projects themselves, including a number of user stories under those epics. The workshops lasted around 4 hours and were done in two different sessions. Figure 4.12 shows a epic that has been clicked to show the user stories within.

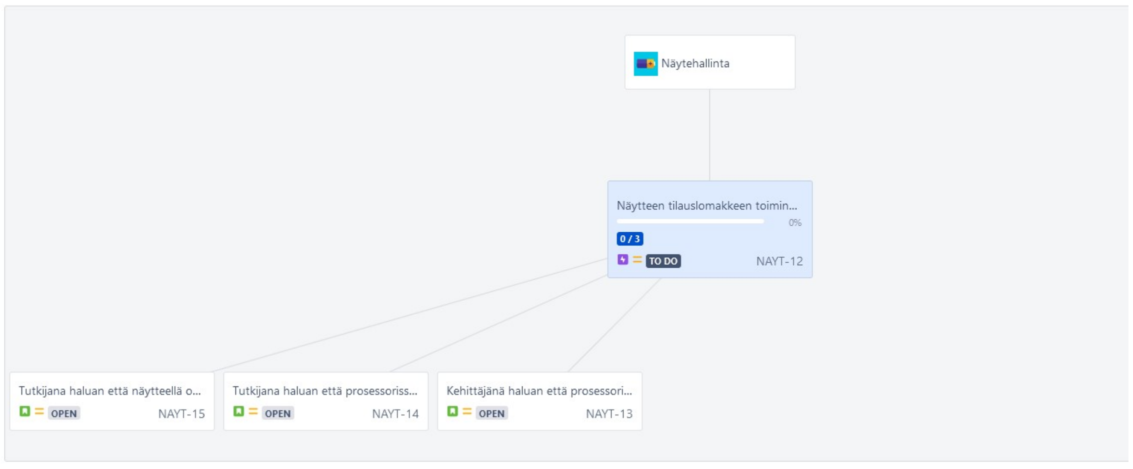

Näytehallinta
Näytteen tilauslomakkeen toimin...
0%
0 / 3
TO DO
NAYT-12
Tutkijana haluan että näytteellä o...
OPEN
NAYT-15
Tutkijana haluan että prosessoriss...
OPEN
NAYT-14
Kehittäjänä haluan että prosessori...
OPEN
NAYT-13

Figure 4.12: The Epics map view showing the user stories under an epic, after the epic has been clicked

The Epics map was added after the interviews were done, so we had the ability to use the view it provided to help us to see the epic structure form as we were adding and editing the epics and user stories of each project. The author was mostly leading the workshops as the project leads of Suomu and Näytehallinta were giving specifics on what kind of user stories were needed for the epics and story items, screen sharing was often used by all participants so everyone could contribute to writing user stories while others provided input. Figures 4.13 and 5.14 show the respective projects views for Suomu and Näytehallinta projects.

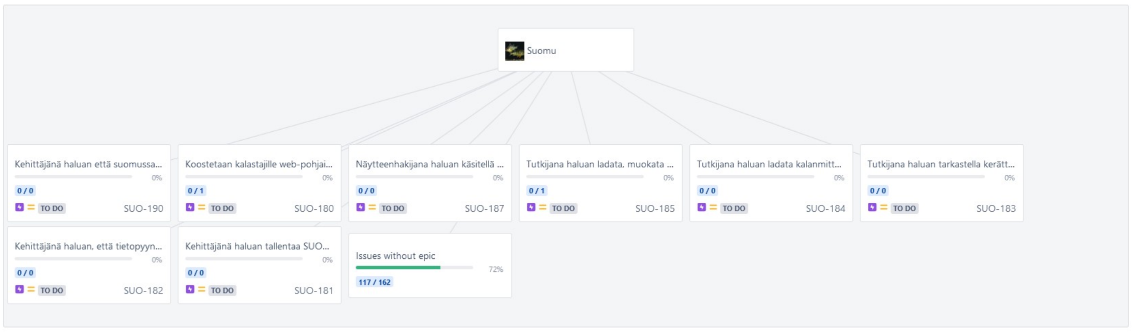

Suomu
Kehittäjänä haluan että suomussa...
0%
0 / 0
TO DO
SUO-190
Koostetaan kalastajille web-pohjai...
0%
0 / 1
TO DO
SUO-180
Näytteenhakijana haluan käsitellä ...
0%
0 / 0
TO DO
SUO-187
Tutkijana haluan ladata, muokata ...
0%
0 / 1
TO DO
SUO-185
Tutkijana haluan ladata kalanmitt...
0%
0 / 0
TO DO
SUO-184
Tutkijana haluan tarkastella kerätt...
0%
0 / 0
TO DO
SUO-183
Kehittäjänä haluan, että tietopyyn...
0%
0 / 0
TO DO
SUO-182
Kehittäjänä haluan tallentaa SUO...
0%
0 / 0
TO DO
SUO-181
Issues without epic
72%
117 / 162

Figure 4.13: View of Suomu project

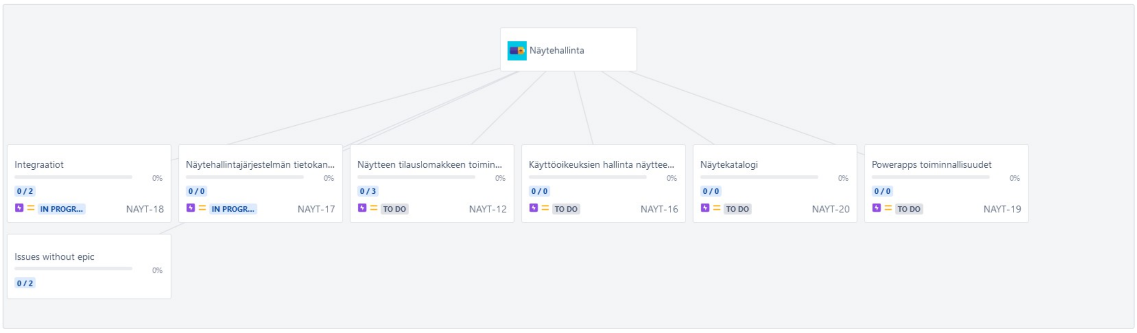


Figure 4.14: View of Näytehallinta project

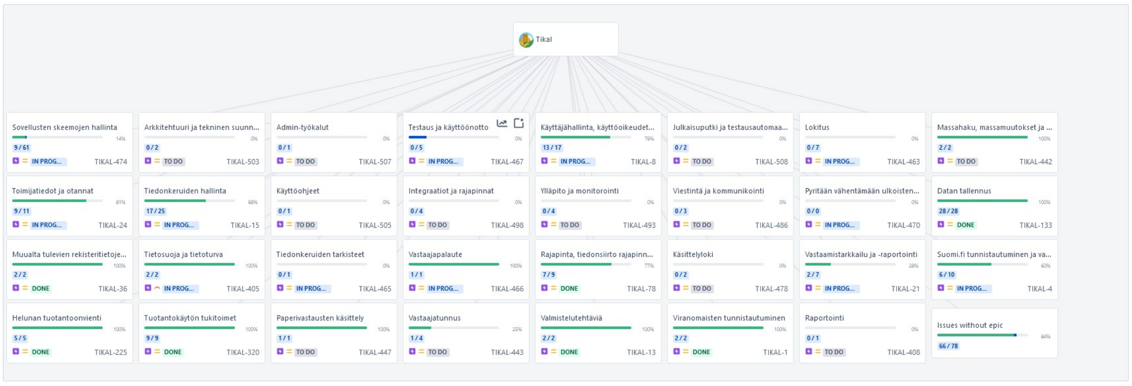
For the third project, Tikal, things were not so simple, Tikal had been in development for 2 years, and the epic structure of Tikal contained epics that were categories of tasks, epics that were properly made large user stories, as well as epics that were named after the services that the project used.

Many of the epics of Tikal had work that had already been done or work that wasn't necessarily part of Tikal itself, but something related to it's branching projects, for example about the already done work, there's an epic named "Helunan tuotantoonvienti" which is an epic about getting a system under Tikal into production. If this would be done again, it would likely be a user story.

There's nothing wrong with having epics that had already been done, but user stories and categories that shouldn't be listed as epics should not exist, as they clutter the epic view and make the structure of the project harder to read, even without the Epics map, but that is another conversation about technical debt.

The workshop for tikal was held with both the author and the product owner of Tikal project, and the project was deemed too large to go through in the relatively short workshop that we had. The product owner of Tikal unfortunately only had little time to organize the project structure in terms of epics, as software development related tasks were taking most of their time.

As a result of the aforementioned, the Epics map view of Tikal ended up containing over 30 epics, making it rather hard to read, and the epics listed were not categorized clearly, making them overlap. Tikal Epics map shown in figure 4.15.



Tikal

Sovellusten skeemojen hallinta 14% 9 / 61 IN PROG... TIKAL-474
Arkkitehtuuri ja tekninen suunn... 0% 0 / 2 TO DO TIKAL-503
Admin-työkalut 0% 0 / 1 TO DO TIKAL-507
Testaus ja käyttöönotto 0% 0 / 5 IN PROG... TIKAL-467
Käyttäjähallinta, käyttöoikeudet... 76% 13 / 17 IN PROG... TIKAL-8
Julkaisuputki ja testausautomaa... 0% 0 / 2 TO DO TIKAL-508
Lokitus 0% 0 / 7 IN PROG... TIKAL-463
Massahaku, massamuutokset ja ... 100% 2 / 2 TO DO TIKAL-442

Toimijatiedot ja otannat 81% 9 / 11 IN PROG... TIKAL-24
Tiedonkeruiden hallinta 68% 17 / 25 IN PROG... TIKAL-15
Käyttöohjeet 0% 0 / 1 TO DO TIKAL-505
Integraatiot ja rajapinnat 0% 0 / 4 TO DO TIKAL-498
Ylläpito ja monitorointi 0% 0 / 4 TO DO TIKAL-493
Viestintä ja kommunikointi 0% 0 / 3 TO DO TIKAL-486
Pyritään vähentämään ulkoisten... 0% 0 / 0 IN PROG... TIKAL-470
Datan tallennus 100% 28 / 28 DONE TIKAL-133

Muualta tulevien rekisteritietoje... 100% 2 / 2 DONE TIKAL-36
Tietosuoja ja tietoturva 100% 2 / 2 IN PROG... TIKAL-405
Tiedonkeruiden tarkisteet 0% 0 / 1 IN PROG... TIKAL-465
Vastaajapalaute 100% 1 / 1 IN PROG... TIKAL-466
Rajapinta, tiedonsiirto rajapinn... 77% 7 / 9 DONE TIKAL-78
Käsittelyloki 0% 0 / 2 TO DO TIKAL-478
Vastaamistarkkailu ja -raportointi 28% 2 / 7 IN PROG... TIKAL-21
Suomi.fi tunnistautuminen ja va... 60% 6 / 10 IN PROG... TIKAL-4

Helunan tuotantoonvienti 100% 5 / 5 DONE TIKAL-225
Tuotantokäytön tukitoimet 100% 9 / 9 DONE TIKAL-320
Paperivastausten käsittely 100% 1 / 1 TO DO TIKAL-447
Vastaajatunnus 25% 1 / 4 TO DO TIKAL-443
Valmistelutehtäviä 100% 2 / 2 DONE TIKAL-13
Viranomaisten tunnistautuminen 100% 2 / 2 DONE TIKAL-1
Raportointi 0% 0 / 1 TO DO TIKAL-408
Issues without epic 84% 66 / 78

Figure 4.15: View of Tikal project

One issue that was noticed during the implementation phase of the case study, was that the epics in the Epics map model weren't very readable in terms of their text content. The epics that were written with their user story outline in the title, were very easily cut off after a few words, it would have made more sense to use the ”epic name” field instead in the card headers of the Epics map model to make the cards more readable. Problem shown in figure 4.16. The epic edit screen shown in figure 4.17 to show the relation of the epic name and the user story of the epic.

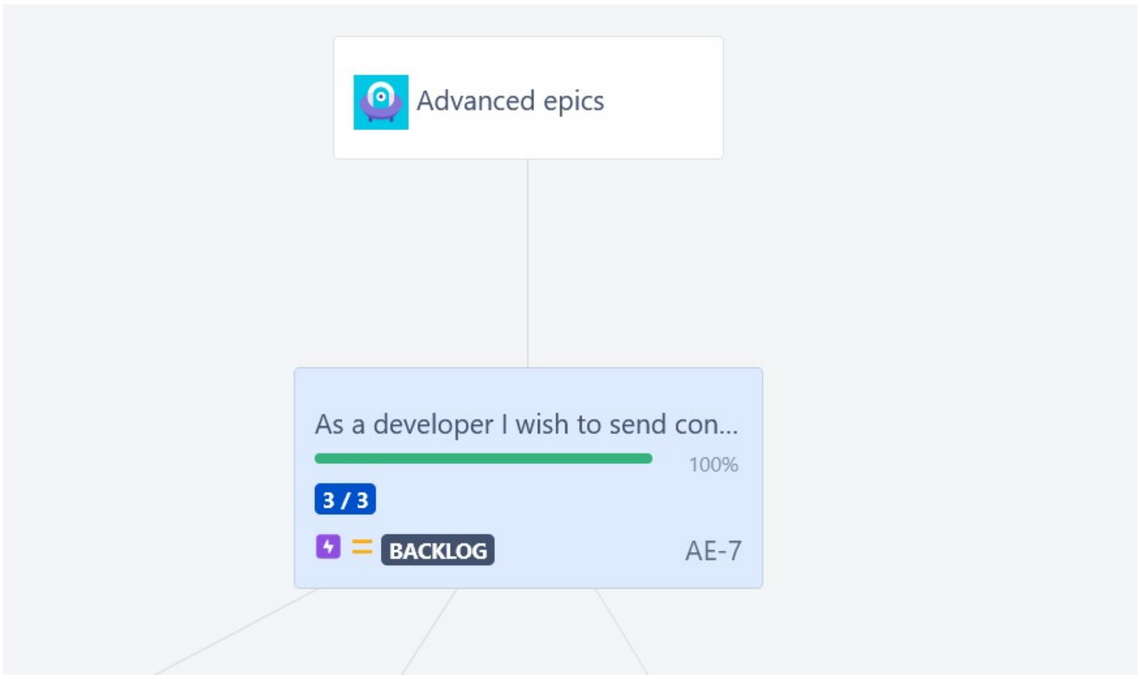
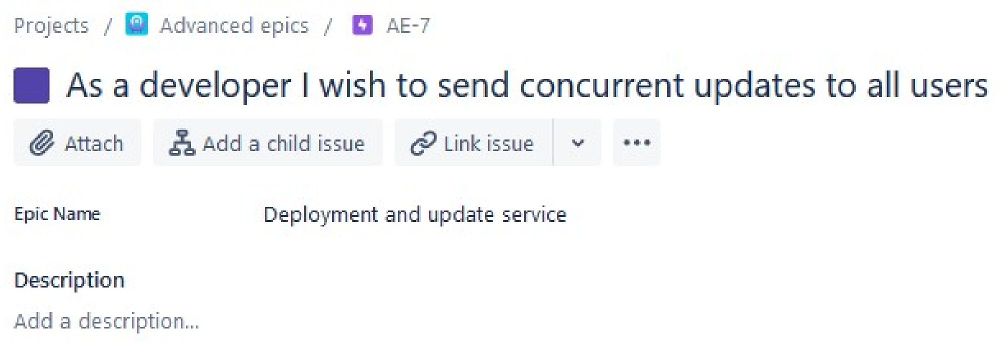


Figure 4.16: An epic card showing it's title being cut off

After the author e-mailed Herocoders, the developers of Epics map, Herocoders did answer that they did want to improve the epics map by including the epic names in the cards instead of the epic title, but the change had not been made by the end of may 2023, when this paragraph was being written.


Projects / Advanced epics / AE-7

As a developer I wish to send concurrent updates to all users

Epic Name Deployment and update service

Description

Add a description...

Figure 4.17: A issue view of an epic's title being shown first, and the much shorter epic name being shown after in a modified Jira issue view

4.5 Interview results

This section presents the results of each interview question listed by the most common themes being listed first. The results of each interview question were identified through notes written both during and after the interviews. The results of some of the interview questions are also backed up by quotes of the interviewees.

The interviewees are categorised as interviewee 1, 2, and 3 for software developers and x, y and z for project manager, product owner and systems architect in that order.

4.5.1 Background question answers

Before the question pairs of the first round of interviews were asked, some basic background questions were asked, questions about each interviewees role in the organization, education as well as their time in the field of software and Luke. Additionally, there were two questions related to the information sharing of Luke and DIGI.

The questions were:

On what platforms is the technical documentation of your projects?

At Luke, how well do you feel information is being shared in relation to other projects?

The first question was mostly answered with Confluence and Jira, Confluence having the most answers, 5/6 and Jira coming second with 4/6. A notable detail was that three of the participants felt that their documentation was in Github, both as code and as documentation under each of the project folders. Other notable services that were mentioned were Miro, Slack and Teams, as well as a in house tool called Tiimeri and a older legacy documentation site called Redmine.

The information sharing question got a significant amount of answers saying that information was simply not getting shared. Although conflicting opinions did arise. Four out of the six interviewees felt that information was not being shared, of the remaining two answers, one deemed that there wasn't really a problem with information sharing, and the another did not wish to answer as they had no personal experience.

Three out of the six interviewees felt that knowledge was shared within ”tribes” and teams of Luke, but not outside of those tribes due to the tribes working in their own ”silos”.

One of the interviewees answered that "The repeated mechanisms for sharing information are not as good as in the previous projects that I've been in", referring to their prior employers. Also mentioning that information is shared well within some groups, but also there being groups that do not receive enough information.

Another interviewee was asked if information was being shared to them from outside of their own project, they answered with: "No, not in my opinion, there's a lot to be improved, on the documentation level, everything can be found if you know what you're looking for, but myself, I'd wish for, what we're lacking for example is all kinds of, like, what's going on? What thas been done? Like, review type of 'what's been done and what is being prepared', although it does take a lot of time. We'd do nothing but that." Finishing their answer with "I'd wish for more transparency, in all of this work", referring to the work of DIGI.

One of the developers had this to say about personal projects and repeated software development: "Projects are pretty personalized and siloed, so not a lot of information gets shared between them. A little bit of a bad habit that we do the same things again in different projects."

Two of the interviewees also felt that the employees own initiative mattered a lot in terms of how much information was being shared, and that they had a lot of freedom in this regard.

"in my own view, from my own perspective, work-wise I am isolated, for one reason or another, whether I wanted or not, in my own world. I know very little of what is going on." -Interviewee x

Additionally, in their answer to the second information sharing question, interviewee z referred to the previous question of what platforms are used to store the technical documentation, they mentioned that the architecture based work is hard to bring into the daily work of the developer due to it being hard to integrate into the Jira environment:

"We don't really have a streamlined tool, jira is good for like, like the tracking of a software based-project. But Jira does not have the capability of what the Archi brings for example. Of course there are those clunky add-ons, they don't quite serve the, the -of course we could get them as pictures, and links." -z
"So what you do doesn't quite get transmitted into other work." -Interviewer
"Yeah, unfortunately it gets quite fragmented." -z

4.5.2 Interview round 1

The first round of interviews went smoothly, I was able to interview all participants of the case study. One of the six interviews was conducted in person, while the rest were online through Microsoft Teams.

Transmission and sharing of project information

1. In your experience, how easy is it to get the general idea of a software project through Jira?

Interviewees 1, 2, 3, y, and z answered this question with a no, saying that Jira does not help in getting the general outline of a project. Most of these answers included a saying that Jira was more for documenting work than for giving an idea of what the work was about.

"As a developer, mostly when looking at the board, the kind of a overview is left unnoticed as most of the things visible there are more like detail related stuff." -Interviewee 2

Interviewee x answered: "Poorly, adequately, it depends a lot on how well the leads of a project update their documentation". They felt that they did not know enough about Jira to give a good answer to the question.

While participants 1, y and z did make remarks that Jira could be used to gain some perspective on a project all of them mentioned, in one way or another, that the focus of jira was not in giving a overview on a project.

"The general idea of a project does not transmit through Jira, since Jira isn't currently a service through which we aim to transmit the idea" "It comes from Miro, for exampl.e" -Interviewee y

"The overview of a project comes from elsewhere" "It comes from all around, a sum of many places." -Interviewee 3

Three of the interviewees, 2, x and y also mentioned that Jira projects often contained old documentation in regards to epics and user stories which would likely never be implemented.

"And often jira is the kind of service where there is lots of things some of which is also old, which of course clutters up the overview." -Interviewee y

"Backlog usually contains all kinds of things it’s hard to tell what is, what is the barrel of wishes and what is coming." -Interviewee 2

In a rather long conversation with the interviewer, interviewee z noted that their work with architecture and solution models had trouble being shown in projects that were using Jira to guide development work, they felt that adding these models to Jira had a lot of trouble, it was possible but required a lot of work. Interviewee z hoped for solutions of implementing ways to make architecture work more usable for the Jira environment.

”You’re at the core of it, how could we get the, when we have the project that has the project architecture or the architecture report done, and we transition it to Jira, and then the report would get reflected against the solution” -z ”Yeah, yeah, so it(the architecture solution model) gets made but it doesn’t quite find the active tracking-” -interviewer
”(interrupting) Or, or, the progression doesn’t get implemented, it gets kind of left on nothing.” -z

"Backlog usually contains all kinds of things it’s hard to tell what is, what is the barrel of wishes and what is coming." -Interviewee 2

2. If you have looked at other Lukean's projects in Jira; do you find a big difference in how easy it is for you to outline your own project compared to someone else's Jira project?

Interviewees 2, 3, and x answered by saying that they had not looked at Jira projects that were not their own. While y and 1 answered that projects that were not their own were understandable in a "relative" or "basic" way. While adding that their own project was one of the hardest to understand as they were "One of the most complicated project packages that we(DIGI) have in Jira".

"(I) Can perceive the basic Jira but I'm not able to perceive the product that is being built." -y

When prompted about their answer, with "So in other words, Jira simply has nothing to present the general outline of a project?" Interviewee 1 answered with a "Yes", continuing that "In some way one could think about using the epic level structure of where the components would go, would it even be, in the project level? Well, it can be though about."

Interviewee z answered by saying that the ease of understanding fully depends on how well the project is prepared into Jira, there's a lot of variability in the completeness of the content that gets brought into a Jira project.

"Do, you feel that Jira that the ease of understanding a project depends more on the level of preparation of a project in Jira, than what kind of project we're viewing?" -Interviewer "Yes, there should be more time spent on the planning of projects." -z

"There should be more time and concentration, projects tend to take off with too many flying colours." -z

Interviewee z also noted in their answer that if the epic and user story level is not well thought out during the start of a non-sprint based project, old unspecified epics tend to be left to the bog of Jira, cluttering it.

3. If you'd use Jira to present your project to others, would using Jira as a part of the presentation be helpful?

five of the six answers to this question mentioned that Jira could be useful for presenting a project in one way or another, most of them mentioning the Jira items and work that was currently being done.

"Presentation for the short future time-interval, maybe, but there might be better tools." -Interviewee 2

Interviewee y answered simply by saying that in it's current state, Jira would not be helpful for presenting their project, unless add-ons were added such as the Jira Portfolio add-on.

Comprehending repetitive software development

1. Does Jira help you get an understanding of the technical structure of your projects?

Four of the interviewees, 1, 2, x and y, answered that Jira would not help in understanding the technical structure of their software projects.

In their answer, interviewee y said that Jira would give an idea on the technical structure, if the technical structure was included in some of the Jira items but thus far it has not been done.

Interviewee z couldn't say as they had no experience in working with Jira and 3 said that "well yes" and "it might not be the primary help of Jira but it does contain information and helps with understanding".

2. Does Jira help you in detecting the possible repeated software components or services in your projects? The services may be repetitive within the software project or in the future of its following projects or work.

Four interviewees answered the question by saying that Jira does not actively help with detecting repeated software components, although it could help with their detection through things like text search and being able to associate tasks in their context which makes detecting repeated tasks easier.

"Possibly, but not in a main way" -3

"It's better to use it than not to use it" -x

"it could help but in practice, it does not" -2

Interviewees z and y answered with a yes. z Also mentioned about simply noticing that tasks might be repeated under the same epic or story leading to possibly noticing repeated items, adding that they're not sure if this is a good practice or a good way of detecting repeated software components. While y took a look at the big picture:

"Yeah it helps yes. That is what Jira is good at, it helps in outlining and classifying those tasks and also through classifying, detecting these kind of cases." -y

3. Does Jira help in detecting which parts of a software project aren't

repeated?

The answers mostly reflected the answers to the question 2 of this topic. 1, x and z answered by reflecting on their answers to the question before, saying that Jira does not directly help with detecting software components that are not repeated.

> ”Same thing that there you maybe can look at some, invent some kind of search words of some tickets(Jira items) and if you only find one it’s probably not a shared component or is not related to a shared component” ”but it doesn’t really, actively” -1

Interviewee 3 couldn’t say and z as well as y answered by saying that jira would help with detecting non-repeated parts of a project.

Additionally y noted that Jira labeling could be used in identifying custom-made parts that would not get repeated, and also by seeing the ticket history of a project.

> ”You can also observe it from upcoming tickets as well as tickets that have already been done, you just got to find the information.” -y

4. Does Jira help in identifying dependencies between software components?

Interviewees 1, 2, y and z answered this question with a "yes", most of them mentioning that Jira does not actively help in the identification of dependencies.

> "I don’t have experience with defining dependencies in places other than the descriptions(of items)... but maybe with links, in that way yes, if there are issues linked from one Jira project to another, from that you can see some kinds of dependencies." -2

> ”Yeah it helps. Referring to my last answer” -z ”So how was it, when specifying the tickets you can kind of tell what is related to what?” -interviewer ”Yeah, like, what it’s related to and what body it belongs to and what is dependent on it, even the order of implementation.” -z

Interviewee x answered with a "I cannot really say, that’s the coder’s expertise", and 3 with "It does not really in it’s main function." "Maybe it happens somewhere else, and Jira is more of its manifestation(the dependencies)".

4.5.3 Interview round 2

During the second round of interviews, new perspectives started to emerge, I was able to interview all selected participants for the case study. two of the six interviewees were interviewed in person, while the rest were interviewed through Microsoft Teams.

Questions were not asked word for word as there wasn’t really a coherent spoken questions written within the interview document, usually the interviewer simply added "now after the model has been developed" to the start of the phrase, and asked the question of the first round roughly in the same way as it is written in the interview document.

These interview questions have been formatted with the number 2 before each interview question, to make the list more readable in terms of understanding that each question is in fact a question of the second round, after the visual model has been added.

Transmission and sharing of project information

2.1. After the visual model has been added, in your experience, how easy is it to get the general idea of a software project through Jira?

Interviewees y x and z all agreed that the visual model helps in getting the general outline of a project. While interviewees 1 and 2 agreed that the model could possibly be useful in the future, if they had used it more.

”Yeah it would be easier to understand, at least I like the visual box-based view it like shows the pieces better in Jira than what the list views would” -y

When interviewee y was prompted about why the box based view helped them more than the list-based view that base Jira has they answered that:

”I always read a list in order, a list is an order which has a beginning and an end, and this kind of visual order, which has boxes above and below each other, doesn’t have a beginning and an end. You can read it downwards, from the left to right, from left upper corner to right corner below.” and ”when there isn’t a beginning and an end the whole becomes more balanced(to read).”

”How easily? A lot better than without it(the view), so fairly well.” -x

”Yeah in my opinion this is like... enlightening view from which one can see the general outline.” -z

”Just by hunch I’d say that the visual structuring could help with getting a general picture” -2

”Even there(In the workshop), we brainstormed a lot of those(items).” -1 ”Yeah, yeah, could we say that when we had the item workshops, it was noticed that it did help understanding?” -interviewee ”Yeah, yes” -1

Interviewee 3 felt that they couldn’t really answer the question, as the model for their project was too cluttered, it had not been used as intended, they said.

”Well, then, yes maybe it can give visual help, but in this case where tickets haven’t been made using the model it might not help is much as it maximally could.”

In a rather lengthy conversation with interviewee z, when prompted about the list-based view as opposed to the visual box-based view they answered with the following: "But if you’re doing things through the model, it directs you to really, like really, to write the stories, like it just becomes more thorough."

Another detail that was added in z’s answer, was that since visual space was physically limited in the model, it passively directed the user to keep the view compact and readable.

2.2. If you have looked at other Lukean’s projects in Jira; do you find a big difference in how easy it is for you to outline your own project compared to someone else’s Jira project?

Interviewees 1, 2, 3, x and z answered the question by saying that they had not inspected the projects of other Lukeans, 1 and x saying that they’d assume that the model would at least make reading the outline of other projects easier.

Interviewee y felt that there wasn’t a difference in comprehensibility between projects that were their own versus projects that weren’t, the visual model made them all equal in terms of baseline readability, but not in content.

”The comprehension process is the same in projects regardless of whose project it is, the main difference is how each of the projects has built their epics.” -y

2.3. If you’d use Jira to present your project to others, would using Jira

as a part of the presentation be helpful?

Interviewees 1, 2, x, y and z felt that the Jira model would help with presenting the project to others through Jira.

> ”Yeah, one can, one can present it through the model.” -1 ”If one presents it through Jira, than it is the software side.” -1

> "Yes, yes, without a doubt yes, and especially this kind of compact package of what kind of stories are associated with it and this kind of structure, what kind of things are associated under each epic." -z

> ”This makes these things much more accessible than just to the development team.” -y

> ”Yeah, it would be yes! If I were to think about showing this to a customer then yeah this would be much easier to show in a view like this to them.” -y

When prompted about the progress elements of the model view, y added that: ”Yeah that is true, here you can see better where each thing is going, which is usually visualized with things like roadmaps or some other tools.” Also adding that it would be nice if the model could somehow include information regarding the order of items to be implemented, the list is much harder to show and explain to the customer.

Developer 3 answered that they did not think the model view would help with presenting their own Jira project, as the view was cluttered from epics that were yet to be sorted.

> ”Not necessarily with the project but it could help with the readability of Jira and with tracking Jira tasks" "Not with Tikal at least” -3

Comprehending repetitive software development

2.1. Does Jira help you get an understanding of the technical structure of your projects?

Interviewees 1, x and z answered with a yes, but twith the answers of 1 and x there was noticable hesitancy

"If the epics are associated with technical solutions, I don't quite remember from the top of my own head" "The way I remember, they did" -1 "So they help?" -interviewer "Yeah." -1

"Yes, yeah" -z

"Well, I think" -x "So, a hesitant yes?" -interviewer "Well yeah, a hesitant yes." -x

Interviewee y answered with a yes, saying that "in our project it helps, yes" referring to the fact that in their projects, some of the epics had been categorized from the start by epic name

"Maybe that's what leads to the fact that, For example, in our project the epics have been named in such a way that there's it's own epic for integrations and APIs. So that we can see that 'oh there's that technical structure' but it does not apply to everything." -y

"In our project you can understand the tehnical structure from this, but it's not possible for all projects since those epics are those big user stories." -y

Interviewees 2 and 3 answered with a no, saying that their projects contained a lot of clutter, meaning that the model did not just contain technical information, making the readability of technical elements harder.

”Not in this case at least since these(items) haven’t been grouped into logical groupings” -2

”It does not necessarily, because the technical model itself isn’t that well pictured in the model, there’s all kinds of tickets in the view.” -3

2.2. Does Jira help you in detecting the possible repeated software components or services in your projects? The services may be repetitive within the software project or in the future of its following projects or work.

Interviewees 1, 2 x and z answered by saying that they could not make a proper statement as there was too little time to use the model, interviewee z also added that the model add-on itself could still use some work to actively help the detection of dependencies.

”At first hand, I cannot say.” -2

”Maybe dependencies yes but I’m not quite sure about the repetition, I cannot yet say.” -1

”We’re not quite at that stage yet, that this kind of repetitiveness could be modelled in this.” -z

Interviewee z mentioned that while the model does not help in detecting detecting repeated software components, it does help indirectly, meaning that when a repetitive software component has been done once, the model view on a epic level can be used as a template for the next component, indicating which user stories worked(were completed) and which ones did not.

Interviewee x couldn't say about the model helping in identifying repetitive components across projects, but when prompted about their own project, which the model view had been added to, they answered by saying that the view does help in identifying repetitive components within their project.

”Well maybe the more, like, general understanding of repetition would require a wider and similar acknowledgement of sister projects and other projects” -x

”Yeah and because this model does not support bringing epics from other projects so it might be impossible to understand through this” -interviewer ”Yes I'd maybe say no, but also because our organization culture does not support this kind of cross project practice” -x ”It would require the use of this model across projects.” -x

”but would you say that this helps with identifying repetitive components within this project?” -interviewer ”Yes it helps, as a tool, it helps for sure” -x

x was additionally prompted about as to why the visuality helps them, they answered by saying that the picture might be easier to read as it's easier to concentrate on compared to a list.

Interviewee 3 and y answered positively to the question. 3 answered with the following: ”In principle it does provide a view which enables you to see more tickets at once and through that you can maybe spot similar tickets from the same category, so in that way it can help.”

y had a similar way of approaching the problem, they answered by saying that the model helps by enabling the user to see epics that have already been done, and compare them to upcoming items.

> ”It helps because if an epic has been moved to done, or if epics are small for example, when they’re done and get moved to done, than in this model you can get the epics that have been moved to done to be seen.” -y

> ”So you can look at ongoing, done and future epics and see if there are repetitive items” -y

y also mentioned that they often reflect on the old epics list against the epics of a new project, so they would likely be doing the same using the visual model.

2.3. Does Jira help in detecting which parts of a software project aren’t repeated?

Interviewee 1, y and x answered with a yes, 1 saying that while they answered no to the previous question, the visuality helps with seeing which items aren’t associated with other items.

> ”Yes in my opinion it does, in that perspective it does.” -1

> ”Visually, I can see that it(the epic) isn’t associated to other items.” -1

> ”Yes, it helps with that too.” -x

Interviewee y noted that the visuality helps with identifying if something is going to be repeated or has been repeated, and also that the model helps indirectly in detecting non repeated components.

> ”Yes it does help, we can see from all the epics that are associated with a project, which ones are done, which ones are under development and which are up and coming we can, for example, see through the model that one similar epic doesn’t exist.” -y

> ”Would you say that it helps indirectly? In detecting non repeated components?” -interviewer ”Yeah, you kind of, need interpretation, that we already wish to identify if something has been repeated or might get repeated.” -y

Interviewees 2 and 3 couldn’t say, interviewee 3 mentioned that if the epics of their project had been done with the model from the very start, they might be able to say better.

Interviewee z referred to the previous answer of theirs, by saying that they did not think the model was quite there yet. They added that the model might help in practice with repetitive components.

> ”The model isn’t necessarily quite there yet, it’s gonna be seen here in due time but I’d see that in terms of repetitiveness it might help to see a representation of how a project has gone and how it has developed”

2.4. Does Jira help in identifying dependencies between software components?

Interviewees 1, x, y and z answered positively with 1 and y saying that Jira with the added visual model outright help with identifying dependencies, while x and z believed that Jira indirectly helps with identifying dependencies.

> ”In a way, no, because this does not show issues that have been linked to each other.” -y

> ”The only way it helps is through the visuality, if a person knows that they are looking for dependencies, on a high abstraction level at least, than through that they can find dependencies using this” -y ”So in this way it does help with identifying dependencies.” -y

> ”It does help in that too.” -x

> ”Yeah it helps more in a indirect kind of way, in presenting the kind of things that can be dependent on each other” -x

> ”yes, yeah, not actively” -z ”Would you say that it also does help through it being easier to perceive the possible dependencies?” -interviewee ”yeah, yes.” -z

Interviewees 2 and 3 answered by saying that it was too early for them to give a concrete answer on the topic.

4.5.4 Other disscussion during interviews

In a lengthy conversation after the first interview rounds questions, with interviewee z, it was noted that the problem with Jira is that it does bring out some structure of the work, so developers often don’t go out of their way to find and read architecture documents, so those documents should be where the actual work takes place. Another problem that was mentioned was that there were lots of tools that did the same thing as the tools in Jira, but the cost of switching an environment was high for the individual, as a certain threshold of stress was required for the software worker to change the tool that they tend to use.

Another interviewee, interviewee 2 noted after the first interview round’s questions were done that in their opinion Jira was just pointless nonsense, and that the times of flip chart were much better.

They felt that using small pieces of paper were a much better way of representing requirements, when prompted as to why, the interviewee mentioned that it's just better to passively read a board, it's always in the same place and the movement of requirements is very visible as they're moved by individuals.

One can see cluttered areas of the board's columns very easily at a first glance, and everything can be colour coded. With the obvious downside being required to be in the same space as the board. The addition of bugs is also very visible.

After the interview, interviewee 2 also noted that the physical closeness enables the user to be able to tell what parts of a project are related. They also hoped to see colour coding on when epics where wholly done, with all stories done inside an epic, saying that colours are great since they enable the user to see information without having to specifically read anything.

Interestingly interviewee 2 also mentioned that the visual model was one of the best ways of possibly modelling dependencies, as they'd be way more visible than just in the item descriptions after opening a specific item. They also say that dependencies can be lost in the structure due to them being so hidden, while in a model, they cannot be lost as easily.

Overall they agreed with the idea that the big downside of Jira was it's activeness. A developer is required to look at Jira in order to take a look at the current status of work, while it was deemed much nicer to be able to passively get that information, without it requiring any kind of initiative.

Before the interview with interviewee 3, they noted that the epic titles in the model view get cut off, so it's considerably harder to read the actual text of the different epics in the model. 3 also noted that since their project was cobbled together way before the model view was added, the model view was much more cluttered than in the other projects that had their base made specifically with the model in use during workshops. The views would be much more clear if the project was started while using the visual model.

During their interview, one of the interviewees mentioned that the move to Jira has been made in a appropriate manner, but the benefits to be achieved from making the move, are yet to be achieved.

There was also a note about adding one layer to the model, if one were to inspect repetitive components across projects, they'd need a view from which they could see all the places that a single epic would be linked to. Jira already has a implementation that can provide this view through a feature called a template.

It was noted multiple times by interviewee x that visuality helps them with concentration, as there's a richer information environment in using the visual model.

4.5.5 To summarize

To summarize, a lot of the benefits of the visualization of requirements artifacts has to do with the implementation of visuality, and how the work environment of software workers is set up, the conversations suggest that the comprehensibility and closeness of visual features to everyday work can lead to benefits and a better understanding of requirements, but can also become too demanding to a developer so the implementation should be compact. It seems that Luke can still gleam more benefit from taking advantage of Jira and it's benefits to software development.

4.6 Case study results

During the interviews, the answers to the interview questions changed from the first round of interviews to the second. The most clear change was in the category of "transmission and sharing of project information" Where answers to the questions 1 and 3 changed.

The question 1 regarding the overall comprehensibility of software projects changed from all the interviewees saying that Jira does not really help with understanding the general idea of a software project, to half of the interviewees saying that Jira, with the addition of the visual model, helps in understanding the general idea of a software project. The rest saying that if they had more time with the model, they'd likely be able to give defined answers to the question, two of them adding that their vision was that the model would help them with understandability, expressing positive attitude towards the visual model.

The question 3 to that same category regarding the presentability of projects through Jira, changed from five out of six of the answers being partly positive, to five of the answers agreeing that the model would help with presenting their project.

There were also changes in the answers to the technical section of comprehending repetitive software development.

The most considerable change in this category was the change to answers regarding question 1 of whether or not the visual model in Jira helped the interviewees with understanding the technical structure of their projects. During the first interview round, most of the interviewees(4/6) answered with a no, and in the second round, with most of the answers (4/6) being positive towards Jira with the visual model added.

Unfortunately, due to the short window of time between the interview rounds, and the lack of initiative towards actively using the model, the answers to technical questions 2 and 3 regarding the detection of repeated software components or services, remained inconclusive, most of the interviewees gave non answers due to simply not using the model. The same unfortunate lack of time of use was observed in the second interview round for the question 2 of the information sharing topic, which lead to the interviewees not being able to answer the question again. The model also lacked the feature of being able to use the same requirement artefact across many projects as well as issues with name tags of the visual items. Having these features would have added value to the visual model.

It can be said with certainty that this research failed in it's goal of answering whether or not the visual model helps with detecting repeated software components and increasing software product line thinking, but strongly suggests that the visual model helps with information sharing.

5 Discussion

This study takes a look into reusability in software development, and a brief look into the visual assistance of models for software product line development. The case study takes a look into the minds of developers and project workers around different software project and asks them, how does a visual model help them in their daily work, and does the model help them identify structures associated with commonality of requirements and components in projects.

The focus of the questions is simply around the concept of project members identifying individual components from the requirements structure. The main goal of the research being to uncover if the visual model helps the identification of reusable components, and as a consequence, to bring the organization of Luke slightly closer to software product line thinking. The research also tries to map out how visualization helps the daily work and inter-project communication of project workers in ways that would be not inherently identifiable.

This research was definitely headed in the right direction, the model that had been implemented, the interview questions, question structure, and the subjects that were chosen were aligned with the current research environment. The reception for the model that was implemented into views was positive, the workshop sessions while using the view to plan requirements were very engaging and productive for the participants, and during the interviews the interviewees attitudes towards the model were positive.

There were also very interesting findings related to offshoots of conversation related to the topic: Interviewee z noting that the architecture work isn't very present in projects during the first interview round, and during the second interview round, being delighted that his work could be brought visible on a requirements level, while additionally being able to track project progress in real time. There was a related conversation with interviewee 2 regarding the availability of information and the mental threshold for having to check requirement status, as opposed to having it readily in the space where work is done, best case of this being a physical flat board at the office with requirements as post-it notes. The digital implementation of similar concepts should remain clear and compact for readability and accessibility.

Another observation was that members of a certain project couldn't read the visual model view in their own project due to the project being too cluttered and the way of writing requirements into the project had changed through it's lifecycle. The way of writing requirements should be streamlined in projects to increase readability, which could lead to better information sharing both in a singular project, and across many projects.

The model could have great benefits for the DIGI unit of Luke, as it would help personnel understand the purpose of epics as well as user stories better and streamline the way items are written into Jira, ultimately leading to benefits of increased information sharing, like lessened repeated work, as well as opening the door for the possibility of developing common components in the future.

The case study research did not achieve all of its goals, specifically with finding out whether or not the visual model helped with finding repeated features or components. Future research could be conducted with very similar methods, but with improved research setting. The case study mainly had too little time for users to use the model, and the Jira add-on used for the model could have more features.

The study answers the main research question **How can a high-level real-time visual tracking model assist in bringing product line thinking and increasing information sharing within an organization?** by stating that the results of the case study strongly suggest that the model does increase information sharing within an organization by improving visual clarity of project requirement items such as epics. This answer can be retrieved from the answers the sub questions.

The most dominant themes to the sub questions below were as follows:

In what ways can a visual Jira architecture model of high level abstraction help software engineering?

The study strongly suggests that a visual architecture model of high level abstraction can help software engineering in terms of information sharing and in terms of presentability and understandability of projects, which can lead to improvements in information sharing across different projects and inside the projects themselves.

This answer is based in the answers to the interview questions 1 and 3 in the category of information sharing having a very drastic change in answers, and them being mostly positive towards the addition of the model.

Could a Jira architecture model of high abstraction level help in information sharing for both a project and an organization?

The study suggests that a Jira architecture model can lead to improvements in information sharing across different projects and inside the projects themselves as the understandability and presentability of projects becomes easier.

This answer is based in the same answers as described in the sub question above. The answers to the interview questions 1 and 3 in the category of information sharing changed drastically in favour of the visual model.

Could a repeatable Jira architecture model of high abstraction help with developing repeated software components?

The study cannot answer this question, the answers to the technical section of the study remained inconclusive, mostly due to the short time window of the study. The add-on that was used for the case study, also lacked key features that could have added technical value to the model, such as global epics and better epic name representation.

To conclude, the subcategories and the goals of the main research question could not all be answered, but further research can gain a lot from the results of this case study. The results of the study strongly suggesting that the model can increase information sharing within an organization are based on the answers of six people related to the author. Because of both the author's influence, the small sample size of interviewees and the short time window of the study, it's hard to generalize about the results of the study. However, in the DIGI unit of Luke, the visual model was received well and there are benefits to information sharing within certain projects.

6 Conclusion

In this study, a visual model for the tracking and presenting of agile epics is added to the Jira view of the Natural Resources Institute Finland(Luke) DIGI unit.

The interview results from the answers of the participant developers and other project workers from the Luke DIGI unit strongly suggests that a visual architecture model of high level abstraction can help software engineering in terms of information sharing and in terms of presentability and understandability of projects, which can lead to improvements in information sharing across different projects and inside the projects themselves.

The study could not answer how the model helps with technical software engineering practices mostly due to the limited time window of the study. But provides insight towards future research in how to assist project workers with graphical tools in their daily work and shift an organization towards product line thinking.

The visual model seems to help with comprehensibility of projects in terms of overall idea and technical structure, but it's hard to say whether or not it helps repetitive software development due to the model lacking some features, and the study being hampered by a short time window for the interviewees' use of the model.

In future research, if the model would mainly include cross-project epics, much better results could be gained in medium and large organizations, and even in small ones. The increased time window of at least one month instead of the two weeks for model use would most likely greatly increase result quality. Another key takeaway would be to simply have a larger sample size of both developers as well as other project workers with questions targeted to both groups separately.

Additionally the model could introduce dependency lines between different dependent requirement items, as well as better epic name representation of epic name, instead of the user story of the epic, to increase technical understandability and overall readability, respectively.

Lastly, there could be more initiative to get projects to write their epics in a similar manner across all projects of the organization, before conducting the research.

References

[1] C. W. Krueger, “Software reuse”, *ACM Computing Surveys*, vol. 24, no. 2, pp. 131–183, 1992, ISSN: 0360-0300. DOI: 10.1145/130844.130856. [Online]. Available: https://doi.org/10.1145/130844.130856.

[2] P. Freeman, “Reusable software engineering: Concepts and research directions”, in *ITT Proceedings of the Workshop on Reusability in Programming*, vol. 129, 1983.

[3] T. Biggerstaff and C. Richter, “Reusability framework, assessment, and directions”, *IEEE Software*, vol. 4, no. 2, pp. 41–49, 1987. DOI: 10.1109/MS.1987.230095.

[4] I. Reinhartz-Berger, A. Sturm, C. T., S. C., and B. J., eng. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, ISBN: 9783642366543.

[5] D. C. Schmidt, “Why software reuse has failed and how to make it work for you”, *C++ Report*, vol. 11, no. 1, p. 1999, 1999.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Reading MA: Addison-Wesley, 1994, p. 1.

[7] L. Northrop, P. Clements, F. Bachmann, *et al.*, “A framework for software product line practice, version 5.0”, in Software Engineering Institute, 2012.

[8] R. Capilla, T. Mikkonen, C. Carrillo, F. A. Fontana, I. Pigazzini, and V. Lenarduzzi, “Impact of opportunistic reuse practices to technical debt”, in *2021*

IEEE/ACM International Conference on Technical Debt (TechDebt), IEEE, 2021, pp. 16–25. DOI: 10.1109/TechDebt52882.2021.00011.

[9] N. Mäkitalo, A. Taivalsaari, A. Kiviluoto, T. Mikkonen, and R. Capilla, "On opportunistic software reuse", *Computing*, vol. 102, pp. 2385–2408, 2020.

[10] J. Sametinger, *Software engineering with reusable components.* Springer Science & Business Media, 1997.

[11] A. W. Brown, *Large-scale, component-based development.* Prentice Hall PTR Englewood Cliffs, 2000, vol. 1, pp. 70–73.

[12] C. Szyperski, D. Gruntz, and S. Murer, *Component software: beyond object-oriented programming.* Pearson Education, 2002.

[13] I. Crnkovic, "Component-based software engineering - new challenges in software development", eng, in *Proceedings of the 25th International Conference on Information Technology Interfaces, 2003. ITI 2003*, IEEE, 2003, pp. 9–18, ISBN: 9789539676962.

[14] M. Morisio, C. B. Seaman, A. T. Parra, V. R. Basili, S. E. Kraft, and S. E. Condon, "Investigating and improving a cots-based software development", in *Proceedings of the 22nd International Conference on Software Engineering*, ser. ICSE '00, Limerick, Ireland: Association for Computing Machinery, 2000, pp. 32–41, ISBN: 1581132069. DOI: 10.1145/337180.337186. [Online]. Available: https://doi.org/10.1145/337180.337186.

[15] I. Crnkovic, M. Chaudron, and S. Larsson, "Component-based development process and component lifecycle", in *2006 International Conference on Software Engineering Advances (ICSEA'06)*, IEEE, 2005. DOI: 10.1109/ICSEA.2006.261300.

[16] M. Mrva, "Reuse factors in embedded systems design", *Computer*, vol. 30, no. 8, pp. 93–95, 1997.

[17] K. Pohl, G. Böckle, and F. Van Der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*, eng. Berlin, Heidelberg: Springer Nature, 2005, ISBN: 3540289011.

[18] J. Axelsson, A. Kobetski, Z. Ni, S. Zhang, and E. Johansson, "Moped: A mobile open platform for experimental design of cyber-physical systems", in *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, IEEE, 2014, pp. 423–430. DOI: 10.1109/SEAA.2014.38.

[19] C. Ebert, "Correspondence visualization techniques for analyzing and evaluating software measures", English, *IEEE Transactions on Software Engineering*, vol. 18, no. 11, pp. 1029–1034, Nov. 1992, Copyright - Copyright Institute of Electrical and Electronics Engineers, Inc. (IEEE) Nov 1992; Last updated - 2022-11-11; CODEN - IESEDJ. [Online]. Available: https://www.proquest.com/scholarly-journals/correspondence-visualization-techniques-analyzing/docview/195562660/se-2.

[20] P. C. Wang, *Graphical representation of multivariate data.* Elsevier, 2014.

[21] E. R. Tufte, *The visual display of quantitative information.* Graphics press Cheshire, CT, 2001, vol. 2.

[22] L. Linsbauer, E. R. Lopez-Herrejon, and A. Egyed, "Recovering traceability between features and code in product variants", in *Proceedings of the 17th International Software Product Line Conference*, ser. SPLC '13, Tokyo, Japan: Association for Computing Machinery, 2013, pp. 131–140, ISBN: 9781450319683. DOI: 10.1145/2491627.2491630. [Online]. Available: https://doi.org/10.1145/2491627.2491630.

[23] S. Illescas, R. E. Lopez-Herrejon, and A. Egyed, "Towards visualization of feature interactions in software product lines", in *2016 IEEE Working Conference on Software Visualization (VISSOFT)*, IEEE, 2016, pp. 46–50.

[24] R. E. Lopez-Herrejon, S. Illescas, and A. Egyed, "A systematic mapping study of information visualization for software product line engineering", eng, *Journal of software : evolution and process*, vol. 30, no. 2, e1912–n/a, 2018, ISSN: 2047-7473.

[25] G. K. Hanssen and T. E. Fægri, "Process fusion: An industrial case study on agile software product line engineering", *Journal of Systems and Software*, vol. 81, no. 6, pp. 843–854, 2008, Agile Product Line Engineering, ISSN: 0164-1212. DOI: `https://doi.org/10.1016/j.jss.2007.10.025`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0164121207002518`.

[26] T. Vale, B. Cabral, L. Alvim, *et al.*, "Splice: A lightweight software product line development process for small and medium size projects", in *2014 Eighth Brazilian Symposium on Software Components, Architectures and Reuse*, IEEE, 2014, pp. 42–52.

[27] P. Trinidad, D. Benavides, and A. Ruiz-Cortés, "Improving decision making in software product lines product plan management", in *Proceedings of the V ADIS 2004 Workshop on Decision Support in Software Engineering*, University of seville, vol. 120, ADIS, 4ª, 2004, Málaga, 2004.

[28] W. Tracz, "Domain-specific software architecture (dssa) frequently asked questions (faq)", *ACM SIGSOFT Software Engineering Notes*, vol. 19, no. 2, pp. 52–56, 1994.

[29] M. Harsu, *A survey on domain engineering.* Citeseer, 2002, vol. 12.

[30] K. Czarnecki, "Domain engineering", in *Encyclopedia of Software Engineering.* John Wiley & Sons, Ltd, 2002, ISBN: 9780471028956. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/0471028959.sof095`.

[Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/0471028959.sof095.

[31] L. F. Capretz, M. A. Capretz, and D. Li, "Component-based software development", in *IECON'01. 27th Annual Conference of the IEEE Industrial Electronics Society (Cat. No. 37243)*, vol. 3, IEEE, 2001, pp. 1834–1837.

[32] M. Bertl, T. Klementi, G. Piho, P. Ross, and D. Draheim, "How domain engineering can help to raise decision support system adoption rates in healthcare", in *HEDA 2023: 3rd International Health Data Workshop, July 21st, 2023, Leicester, UK*, CEUR Workshop Proceedings, 2023.

[33] R. Prieto-Dıaz, "Domain analysis: An introduction", *SIGSOFT Softw. Eng. Notes*, vol. 15, no. 2, pp. 47–54, 1990, ISSN: 0163-5948. DOI: 10.1145/382296.382703. [Online]. Available: https://doi.org/10.1145/382296.382703.

[34] W. Lam and J. A. McDermid, "A summary of domain analysis experience by way of heuristics", *ACM SIGSOFT Software Engineering Notes*, vol. 22, no. 3, pp. 54–64, 1997.

[35] I. K. Bray, *An introduction to requirements engineering.* Pearson Education, 2002.

[36] J. Dick, *Requirements engineering*, eng, 4th ed. Cham: Springer, 2017, pp. 9–10, ISBN: 9783319610733.

[37] K. Czarnecki, K. Østerbye, and M. Völter, "Generative programming", in *Object-Oriented Technology ECOOP 2002 Workshop Reader: ECOOP 2002 Workshops and Posters Málaga, Spain, June 10–14, 2002 Proceedings*, Springer, 2002, pp. 15–29.

[38] G. Abowd, L. Bass, P. Clements, R. Kazman, and L. Northrop, "Recommended best industrial practice for software architecture evaluation", eng, Tech. Rep., 1997.

[39] P. B. Kruchten, "The 4+ 1 view model of architecture", *IEEE software*, vol. 12, no. 6, pp. 42–50, 1995.

[40] R. K. Yin, *Case study research : design and methods* (Applied social research methods series ; vol. 5), eng, 2nd ed. Newbury Park, CA: Sage Publications, 1994, ISBN: 0-8039-5662-2.

[41] T. N. R. I. Finland. "Presentation of luke". (2023), [Online]. Available: `https://www.luke.fi/en/about-luke/presentation-of-luke` (visited on 2023).

[42] J. Esquenazi. "The undervalued art of naming epics, releases, features, and user stories". (2022), [Online]. Available: `https://bootcamp.uxdesign.cc/the-undervalued-art-of-naming-epics-releases-features-and-user-stories-343a5e444074` (visited on 2023).

[43] M. Rehkopf. "Stories, epics, and initiatives". (2023), [Online]. Available: `https://www.atlassian.com/agile/project-management/epics-stories-themes` (visited on 2023).

[44] Anna. "Jira issue links and dependencies management", BigPicture. (2023), [Online]. Available: `https://community.atlassian.com/t5/Marketplace-Apps-Integrations/Jira-Issue-Links-and-dependencies-management/ba-p/2050756` (visited on 2023).

[45] C. Wohlin and A. Aurum, "Towards a decision-making structure for selecting a research design in empirical software engineering", *Empirical software engineering : an international journal*, vol. 20, no. 6, pp. 1427–1455, 2014, ISSN: 1382-3256.

[46] J. F. Nunamaker Jr, M. Chen, and T. D. Purdin, "Systems development in information systems research", *Journal of management information systems*, vol. 7, no. 3, pp. 89–106, 1990.

[47] J. Collis, *Business research: a practical guide for undergraduate and postgraduate students*, eng. Basingstoke: Macmillan, 1997, p. 13, ISBN: 0-333-60704-X.

[48] R. W. Clower, "Economics as an inductive science", *Southern Economic Journal*, pp. 805–809, 1994.

[49] J. Hawthorne, "Inductive logic", 2004.

[50] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering", *Empirical software engineering : an international journal*, vol. 14, pp. 131–164, 2009, ISSN: 1382-3256.

[51] H. K. Klein and M. D. Myers, "A set of principles for conducting and evaluating interpretive field studies in information systems", eng, *MIS quarterly*, vol. 23, no. 1, pp. 67–93, 1999, ISSN: 0276-7783.

[52] W. J. Orlikowski and J. J. Baroudi, "Studying information technology in organizations: Research approaches and assumptions", eng, *Information systems research*, vol. 2, no. 1, pp. 1–28, 1991, ISSN: 1047-7047.

[53] D. R. Hannah and B. A. Lautsch, "Counting in qualitative research: Why to conduct it, when to avoid it, and when to closet it", *Journal of Management Inquiry*, vol. 20, no. 1, p. 17, 2011.

[54] I. Benbasat, D. K. Goldstein, and M. Mead, "The case research strategy in studies of information systems", *MIS quarterly*, pp. 369–386, 1987.

[55] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research", *Management Information Systems Quarterly*, vol. 28, no. 1, pp. 76–86, 2004.

[56] R. Davison, M. G. Martinsons, and N. Kock, "Principles of canonical action research", *Information systems journal*, vol. 14, no. 1, p. 82, 2004.

[57] A. Bhattacherjee, *Social science research: Principles, methods, and practices.* Global Text Project, 2012, p. 113.

[58] V. Braun and V. Clarke, "Using thematic analysis in psychology", *Qualitative research in psychology*, vol. 3, no. 2, pp. 77–101, 2006.

[59] D. Valdez, A. C. Pickett, and P. Goodson, "Topic modeling: Latent semantic analysis for the social sciences", *Social Science Quarterly*, vol. 99, no. 5, pp. 1665–1679, 2018.

[60] T. K. Landauer, P. W. Foltz, and D. Laham, "An introduction to latent semantic analysis", *Discourse processes*, vol. 25, no. 2-3, p. 261, 1998.

[61] J. Ritchie, J. Lewis, C. M. Nicholls, R. Ormston, *et al.*, *Qualitative research practice: A guide for social science students and researchers*. Sage Publications, 2013.

[62] T. S. BRUGHA, P. E. BEBBINGTON, and R. J NKINS, "A difference that matters: Comparisons of structured and semi-structured psychiatric diagnostic interviews in the general population", *Psychological Medicine*, vol. 29, no. 5, pp. 1013–1020, 1999. DOI: `10.1017/S0033291799008880`.

[63] N. J. Mathers, N. J. Fox, and A. Hunn, *Using interviews in a research project*. NHS Executive, Trent, 1998, p. 2.

[64] A. Oatey, "The strengths and limitations of interviews as a research technique for studying television viewers", 1999.

[65] B. Byrne, "Qualitative interviewing", *Researching society and culture*, vol. 2, p. 212, 2004.

[66] W. Donsbach and M. W. Traugott, *The SAGE Handbook of Public Opinion Research*, eng. London: SAGE Publications, Limited, 2008, ISBN: 9781412911771.

[67] Herocoders. "Epics map for jira". (2023), [Online]. Available: `https://marketplace.atlassian.com/apps/1220137/epics-map-for-jira` (visited on 2023).

Appendix A The original interview document in Finnish

TAUSTOITUS: (luodaan reilu tutkimusasetelma jotta Jiraa voidaan tarkastella osana kokonaisuutta)

Taustatiedot tutkimuksesta sekä toistuvasta ohjelmistokehityksestä:

Tervetuloa lopputyöni case-tutkimukseen aiheesta Toistuvan ohjelmistokehityksen tukeminen reaaliaikaisen Jira seurantamallin avulla (Assisting product line thinking and information sharing using a real-time Jira tracking model)

Tutkin sitä, miten visuaalinen eepoksien kautta koostettu Jira malli voi tukea informaation välittämistä organisaatiossa, ohjelmistosuunnittelua sekä jokapäiväistä työtä.
Tarkastelemme myös tiedon jakamista Luonnonvarakeskuksessa

Erityispiirteenä tutkimuksessani on software product line ajattelun tuominen Lukelle, tutkimuksessa tuotettujen Jira mallien kautta.
Tuotetulla Jira-mallilla pyritään myös selvittämään, voisiko se auttaa toistuvien ohjelmistokomponenttien tunnistamista projektissa.

Software product line ajattelun keskiössä on ajatus siitä, että tunnistetaan tarvittavia ohjelmistopohjaisia kokonaisuuksia eli tuotteita ja tehdään tältä pohjalta yhteisiä komponentteja kaikkien näiden tuotteiden tarpeisiin. Lopuksi kootaan näistä tuotetuista komponenteista valmiita tuotteita omine custom-osineen.
Eli aluksi tunnistetaan yhteiset komponentit ja sitten yhteisien komponenttien pohjalta tuotetaan valmiita ohjelmistokokonaisuuksia eli tuotteita.
Esimerkiksi Lockheed Martinin Marinetime Systems and Sensors division käyttää software product linea

Tutkimukseni keskittyy seuraaviin teemoihin:
Tiedon jakaminen
Ohjelmistoprojektin hahmottaminen
Tiedon jakaminen Lukella
Toistuvan ohjelmistokehityksen tunnistaminen
Software product line ajattelun lisääminen Lukella
Kehitystyön ja suunnittelutyön tukeminen

Kysymyksiin vastaamatta jättäminen on myös hyvä tapa vastata, eikä ole mitenkään väärin sanoa, ettei vastaajalla ole näkemystä kysyttyyn kysymykseen
Haastattelut nauhoitetaan

Yleiset kysymykset:
Tehtävänimike Lukella ja työnkuvaus
Koulutustaso, viimeisimmän tutkinnon nimike
Kuinka kauan olet ollut lukella työsuhteessa?
Kuinka kauan koet olleesi teknologia-alalla töissä?

Millä alustoilla on projektin(ne?) tekniseen rakenteeseen liittyvä dokumentaatio?
Miten koette tiedon välittyvän Lukella suhteessa muihin ohjelmistoprojekteihin?

KYSYMYSPAREJA

Projekti-informaation välittyminen ja tiedon jakaminen;

1. Pari
Miten helposti koette ohjelmistoteknisen projektin kokonaiskuvan välittyvän Jiran kautta?
Koetteko että mallin toteuttamisen jälkeen, muiden ohjelmistoprojektien kokonaiskuvan hahmottaminen olisi helpompaa Jiran kautta

(Jatkokysymys: Jos ei Jirasta, mistä?)

2. Pari
Jos olette tarkastellut muiden lukelaisten projekteja Jirassa; koetteko suurta eroa siinä, miten helppoa teillä on hahmottaa oma projekti verrattuna jonkun muun Jira projektiin?
Entä mallin toteuttamisen jälkeen?

3. Pari
Jos käyttäisitte jiraa esitelläksenne projektianne, olisiko Jirasta apua projektinne rakenteen esittelyssä?
Entä mallin toteuttamisen jälkeen?

Toistuvan ohjelmistokehityksen hahmottaminen;
1. Pari
Auttaako Jira hahmottamaan teidän ohjelmistoprojektienne teknistä rakennetta?
Entä mallin toteuttamisen jälkeen?

2. Pari
Auttaako Jira hahmottamaan ohjelmistoprojekteissanne mahdollisten toistuvien komponenttien tai palveluiden tunnistamista? Palvelut voivat toistua projektin sisällä tai tulevaisuudessa muissa sitä seuraavissa projekteissa tai työssä.
Entä mallin toteuttamisen jälkeen?

3. Pari
Auttaako Jira tunnistamaan mitkä osat ohjelmistoprojektissa eivät olisi toistettavia?
Entä mallin toteuttamisen jälkeen?

4. Pari
Auttaako Jira tunnistamaan riippuvuuksia ohjelmistokomponenttien välillä?
Entä mallin toteuttamisen jälkeen?