Supporting Scope Tracking and Visualization for Very Large-Scale Requirements Engineering-Utilizing FSC+, Decision Patterns, and Atomic Decision Visualizations

Krzysztof Wnuk, Tony Gorschek, *Member, IEEE*, David Callele, *Member, IEEE*, Even-André Karlsson, Eskil Åhlin, and Björn Regnell

Abstract—Deciding the optimal project scope that fulfills the needs of the most important stakeholders is challenging due to a plethora of aspects that may impact decisions. Large companies that operate in rapidly changing environments experience frequently changing customer needs which force decision makers to continuously adjust the scope of their projects. Change intensity is further fueled by fierce market competition and hard time-to-market deadlines. Staying in control of the changes in thousands of features becomes a major issue as information overload hinders decision makers from rapidly extracting relevant information. This paper presents a visual technique, called Feature Survival Charts+ (FSC+), designed to give a quick and effective overview of the requirements scoping process for Very Large-Scale Requirements Engineering (VLSRE). FSC+ a[illegible]plied at a large company with thousands of features in the database and supported the transition from plan-driven to a more dynamic and change-tolerant release scope management process. FSC+ provides multiple views, filtering, zooming, state-change intensity [illegible]s, and support for variable time spans. Moreover, this paper introduces five decision archetypes deduced from the dataset and sub[illegible]ently analyzed and the atomic decision visualization that shows the frequency of various decisions in the process. The capabilities and usefulness of FSC+, decision patterns (state changes that features undergo) and atomic decision visualizations are evaluated through interviews with practitioners who found utility in all techniques and indicated that their inherent flexibility was necessary to meet the varying needs of the stakeholders.

Index Terms—Requirements/specifications, initiation and scope definition, management

1 Introduction

The probability of product development success is directly correlated with the quality of the scoping exercise [1]; overscoping, scope-creep and requirements-scrap are known obstacles that negatively impact software projects [2], [3], [4], [5]. The context in which scoping is central is principally large, multinational software companies operating in a Market-Driven Requirements Engineering (MDRE) environment [6]. These companies seek to manage the size and complexity of their products and handle a large continuous inflow of requirements [7] that can lead to overloaded requirements management [8]. These large companies often manage thousands of features where each feature may contain between 20 and 50 detailed requirements. The result is often a requirements database with over 10,000 requirements, defined in the literature as Very Large-Scale Requirements Engineering (VLSRE) [9]. In VLSRE, manual management of feature changes or their interdependencies is practically infeasible and requires scalable techniques that can assist requirements analysts.

Large companies operating in MDRE experience frequent scope changes that generate data at a high rate. Studies have shown that up to 80 percent of initially considered requirements are removed from the scope of a non-agile project during its development [10]. Further, it is not unusual to have several significant inclusions of new requirements late in projects [1], [3], [4], [5], [11]—demonstrating that change is inevitable, even when attempting to freeze the project scope [2]. The volatility of scope changes increases for projects inspired by agile methodologies that promote flexibility and embracing change.

Requirements analysts are only able to grasp these changes on a small scale and quickly become overloaded with information when trying to provide an efficient scope process overview. Similarly, decision makers need to be supported by requirements analysts are unable to "rapidly

- *K. Wnuk is with the Software Engineering Research Lab (SERL), Department of Software Engineering, Blekinge Institute of Technology, SE-371 79, Karlskrona, Sweden and with the MAPCI-Mobile and Pervasive Computing Institute Lund University, Sweden. E-mail: krzysztof.wnuk@bth.se.*
- *T. Gorschek is with the Software Engineering Research Lab (SERL), Department of Software Engineering, Blekinge Institute of TechnologySE-371 79, Karlskrona, Sweden. E-mail: tony.gorschek@bth.se.*
- *D. Callele is with the Department of Computer Science, University of Saskatchewan, Saskatoon, Canada. E-mail: callele@cs.usask.ca.*
- *E. A. Karlsson is with the Add a Lot, Sweden. E-mail: even-andre.karlsson@addalot.se.*
- *E. Åhlin is with Sony Mobile Communications, Lund, Sweden. E-mail: Eskil.Ahlin@sonymobile.com.*
- *B. Regnell is with the Department of Computer Science, Lund University, Lund, Sweden. E-mail: bjorn.regnell@cs.lth.se.*

Manuscript received 27 Oct. 2013; revised 12 Dec. 2014; accepted 31 May 2015. Date of publication 0 . 0000; date of current version 0 . 0000. Recommended for acceptance by J. Grundy. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TSE.2015.2445347

extract relevant information from the flood of data" [12]. The goal of this work is to support decision makers in VLSRE with a visualization technique that reduces the cognitive effort [12] needed to provide scoping overview and to support effective decisions by exploring decision-making patterns. The visualization presented here is an example of visual analytics, empowered by direct interaction with the data, enabling insights and observations for improved decision making. This work further explores the potential of visual requirements analytics in supporting requirements management [13] and constitutes an example of quantitative visualization, a topic underrepresented in requirements engineering visualization literature [12], [13].

Scoping in VLSRE is significantly more challenging as the knowledge required to make decisions is typically across many people. Understanding the impact of changes or decisions requires extensive investigation; manual analysis of scope changes (for hundreds of features) by the case company required, on average, one full week of analyst effort for data extraction and analysis. However, the frequency of scope changes made this manual approach impractical. Providing automated analysis and scalable visualizations reduces, via the tools presented here, reduced this effort to minutes making continuous scope monitoring practical and scalable to the industrial context of thousands of features. In this paper, we adhere to the definition of scalability in software engineering as the "property of increasing the scope of software engineering methods, processes and management according to the problem size" [14].

To address industry needs, Wnuk et al. introduced the Feature Survival Chart (FSC) visualization technique that offers decision support for planners and managers in relation to changes in and additions to ongoing projects [10]. FSC makes it possible to get an up-to-date day-by-day overview of scope evolution. FSC can also be used as a postmortem analysis/reflection tool to visualize what happened to the scope of the development activity during the project. The scope evolution visualized through FSCs can be used to adjust the company's process toward greater scope setting decision flexibility and to enable clearer identification of scope responsibility [2], [5]. The case company used FSC to facilitate evolution from batch-centered scope iterations, where changes and additions were seen as a problem, to a continuous scope and release planning flow where changes and additions were an integrated part of the development effort.

This paper reports on FSC+ (an improved version of FSC) visualization technique and its large-scale industrial validation at a partner company faced with large release projects containing several hundreds of features and with change intensive scope evolution of release projects. A release project is, in this context, a project with hundreds of features that are used in a number of products. The features are implemented on the *platform* which is the common code base shared by all products. The company has thousands of features in the database and is an example of VLSRE. The company was transitioning from plan-driven to a more dynamic and change-tolerant release scope management process and FSC+ was used as one of the techniques to support managers in this effort. The goals set by the company for the requirements management process transition were to: (1) remove the fixed-scope definition period for a project and accept continuous scope evolution, (2) increase the frequency of scope evolution to match the target market, (3) allow the requirements inflow stream to be continuous, and (4) increase their ability to perform efficient requirements management. The company emphasized the need for continuous monitoring of scope evolution and history for both project and holistic views, and demanded the continuous ability to have an overview of the current state of project scope at the product level and at the portfolio level [15], [16].

FSC+ substantially extends FSC with the addition of filtering, zooming, state-change intensity views and support for variable time spans. FSC, introduced in our previous work, focused on project scope visualization and was demonstrated on three example projects with hundreds of features [10], [17]. FSC+continues to support the project view but also provides product, portfolio and company level scope visualizations. The project level focus of FSC was the main limitation of the previous work as it hindered scalability and blocked greater than project scope visualizations and analysis; practitioners were unable to determine whether the observed behavior and characteristics were project specific, or greater. The set of scope metrics introduced in our previous work [10] is not reused in this work but new analytics are presented that measure the lead-time in each state and identify decision patterns to enable atomic decision visualizations. Five new decision archetypes, that extend set of reasons for scope exclusion [10], are presented and these decision patterns, decision archetypes and atomic decision visualizations are evaluated in industry on a large dataset with over 8,000 features. To summarize, the concepts that are shared between this work and previous studies are: visualizing features in two-dimensions with time on the *X*-axis and features on the *Y*-axis and sorting features according to their time in the process. Both concepts are significantly extended in this work.

This paper includes details on how FSC+ was developed and its industrial validation. Professionals working in the case company participated in the iterative improvement of FSC+ including identifying the need to analyze the observed decision patterns which resulted in the identification of 2,248 decision patterns (state changes that features undergo). The significant number of identified decision patterns triggered the exploration of an efficient visual representation of the nature of decision patterns. The decision pattern were later grouped into five *decision archetypes*, defined as a set of decision patterns with similar lead-time and outcome characteristics. FSC+, decision patterns and atomic decision visualizations are intertwined, and the scoping history analysis and FSC+ visualization tool developed in Java provide the necessary support for using FSC+ in industry.

The scope of FSC+ evaluation included using FSC+ for scope visualizations, atomic decision visualizations and generating decision patterns. We focused on the visual scalability of FSC+ in terms of applied visual metaphor and other scalability enablers rather than an in-depth analysis of the tool's quality attributes. As a result, we did not focus on studying tool features such as the user interface or evaluating whether the data parser implemented in the current Java implementation can handle other data formats. Finally,

we did not study whether Java is the most appropriate tool to implement FSC+.

The paper is structured as follows. Section 2 presents background and related work, Section 3 presents the case study company description and Section 4 presents the research methodology. FSC+ is presented in Sections 5 and 6 presents and discusses the identified decision patterns. We discuss the results in Section 7 and conclude the paper in Section 8.

2 Background and Related Work

2.1 Release Planning

Release planning is about planning the software content (e.g. features) to be delivered and when that delivery will occur [18]. Release planning extends requirements prioritization by mapping the value attributes of software [19] (often represented by priorities) into a temporal plan over a number of software releases. The release planning process, unlike project scope negotiation, provides a plan over several releases of software and thus supports software product management by making software available to users of an evolving software product in planned stages [20], [21]. The requirements selection process is central for product and business success of software companies [8], [22] and is at the heart of release planning practice [23]. The release planning process is based on a product roadmap document that identifies the strategies for the product [6], [24].

Release planning can be conducted on strategic, operational or dynamic re-planning [18] levels. The operational level is of high interest for incremental development as many software systems deliver in increments with continuous re-planning [25] triggered by increasing knowledge of the system. The increased number of changes on the operational level, combined with a higher degree of uncertainty and higher release frequency [26], creates a need for feedback and monitoring for strategic release planning [27], [28]. FSC+ contributes to meeting this need for strategic release planning support.

Four challenges for software companies operating in MDRE contexts are highly relevant for this paper. The first challenge is how to select features for the next release, maximizing value added [21], [23], while taking the capabilities and capacity of the company and the scope and pace of changes mentioned above into consideration. The second challenge for release projects is how to minimize wasted resources and time spent on features that are completely or partly excluded from the release due to change [29]. Waste is defined here as any cost or effort incurred as a result of the investigation, analysis, design, or implementation of a feature that does not result in delivering value to the customer or target market [30]. Previous work shows that up to 80 percent of total release project budgets are consumed by waste stemming from changes [10]. FSC+ helps to spot potential waste and to visually quantify the amount of waste in the organization.

The third challenge is uncertainty as the lack of up-front accurate information about feature cost, value and interdependencies is commonplace [21], [31], [32]. As a result, release planning operates on early estimates and incomplete information [18], [21], [31], [33], [34] and much information becomes available later in the project when features are analyzed and e.g. pre-studies are performed. These issues increase the volatility of some features [11] which, together with fluctuating customer priorities [10] and shifting of ill-defined goals, result in more changes to release plans and greater uncertainty [31], [32], [33], [34], [35], [36], [37]. The methods presented in this paper help to analyze the volatility of release plans under frequent changes.

The fourth challenge is the high volume of information (number of features) that should be considered when performing release planning (especially for Market-Driven Requirements Engineering contexts [8]) and its questionable quality [21]. As a result, large companies may have thousands of potential features to choose from when creating release plans [9] and can only implement a fraction of them—there is simply not enough time and resources to implement everything [22] and manual analysis, specification and selection of features to go into a release is time consuming and error prone [21], [25], [38].

These challenges greatly contribute to a situation where changes within a release are common and the selection of features for a release is often suboptimal and generates waste [8], [10], [11], [32]. Current release planning methods primarily focus on the factors influencing the release planning decisions between releases, with only limited support for factoring in what may happen to a feature within a release [10], [23], [37]. The changes are caused by, among others, overscoping [2] and subsequent scope reductions [10], [11] as a result of e.g. overzealous planners trying to compensate for future and unknown scope changes, incomplete information up-front [28], and scope creep [3], [11]. The selection of features for a release is often suboptimal and decisions need to be revisited when more information is available.

Some waste occurs due to external factors that are unpredictable (e.g. changes in technology, new competitors entering the market etc. [8]), and it is not possible to eliminate all change and waste. However, substantial waste reduction can be achieved through improving the initial and subsequent re-planning selection of features for a release [21], e.g. by providing upfront robustness analysis [39]. Previous work observed that the proportion of features originally planned for a release that are dismissed before final release can be as high as 80 percent with approximately 30 percent cancelled in the final quarter of the delivery timeline [10]. Therefore, it is important to provide means to actively monitor volatile features within an ongoing release project, minimizing unnecessary resource consumption by dismissing features as soon as possible if they are likely to be dismissed anyway.

Several methods for supporting release planning were proposed. Table 1 presents an overview of the methods based on a literature study by Svahnberg et al. [23]. The analysis is extended by additional studies and two facets relevant to this study: the type of the validation context and the size of the validation context. Release planning approaches that support dynamic re-planning appear to be underrepresented in the literature. Most studies present evaluations on a relatively small number of requirements and there is a need to conduct and report more

TABLE 1
Release Planning Approaches

Approach	Level of support	Validation context	Size of the validation dataset
A cost-value approach for prioritizing requirements [22]	strategic	Two industrial cases	14 requirements and 11 requirements
Supporting road-mapping of quality requirements [43]	strategic	One industrial case	Unknown
Decision Support for Product Release Planning based on Robustness Analysis [44]	strategic	a survey involving product managers and system engineers	Unknown
Retrospective analysis of release planning [45]	dynamic re-planning level	two case studies	20 randomly selected requirements
Analyzing requirements configuration trade-offs [46]	strategic	an experiment on 63 students	Unknown
Integral Linear Programming [47]	strategic	an industrial case study	between 9 and 99 requirements
Optimization and what-if-analysis [37]	strategic	an industrial case study	between 9 and 99 requirements
Risk-Driven Method for XP Release Planning [27]	strategic and operational	a case study	probably less than 100
Quality improvement paradigm [48]	strategic	a case study	13 features
Explain Dialogue [49]	strategic	an experiment to compare ad-hoc with systematic planning	unknown
Bi-Objective Release Planning for Evolving Systems [50]	strategic	a case study	33 features in 2 releases
Fuzzy Structural Dependency Constraints [35]	strategic level	an example	25 requirements
Measuring Dependency Constraint Satisfaction using Dissimilarity of Fuzzy Graphs [51]	strategic	a hypothetical example	10 requirements
Fuzzy Effort Constraints [33]	strategic	a hypothetical example	30 requirements
Release Planning Simulator [39]	strategic	simulation	8 features
Quantitative Win-Win [52]	strategic	an example from a Web portal project	30 requirements
EVOLVE [42]	dynamic re-planning level	an example	20 requirements
EVOLVE+ [31], [53]	dynamic re-planning level	an example	20 requirements
EVOLVE* [34]	dynamic re-planning level	an example	30 requirements
Planning game with uncertaintity [36]	strategic	a case study	9 stories
Feature partitioning for distributed agile release planning [41]	strategic	simulations	unknown
Adaptive release planning in Agile Projects [40]	strategic	an example project	20 requirements
Constraint Programming [54]	strategic	an example from [25]	15 features

studies based on large and very-large datasets. Further, a need emerges to devise and evaluate methods that can operate on both strategic and operational levels [27]. Finally, only a minority of papers consider uncertainty [31], [32], [33], [34], [35], [36], [37] or focused on incremental release planning [26], [36], [40], [41], [42].

2.2 Requirements Prioritization

The requirements prioritization process is challenging for it must consider conflicting stakeholder priorities [22], dependencies between requirements [55], different requirement abstraction levels rendering comparison difficult [6] and limited scalability [56]. Techniques such as cost-value analysis [22] or dividing requirements into three priority categories [57] can be employed for requirements prioritization. Decisions based on assumed requirements priorities are often changed or altered due to a number of factors, e.g. changing stakeholder preferences or unplanned shortage of resources [10]. Given that requirements are (typically) highly interrelated [55], selecting or removing a requirement often necessitates including or removing dependent requirements and leads to release scope change [37]. Prioritization is also complicated by the influx of new requirements (often asynchronous to development efforts) [7], [8] combined with market pressures from competitors' products. These factors could cause requirements management overload and overscoping [2], [8], [58]. A scalable monitoring system that provides real-time feedback on changes in the priorities for ongoing projects is not only recommended [59] but desired.

2.3 Requirements Decision Making

Both release planning and requirements prioritzation are integral parts of requirements decision making, the focal point of requirements engineering [24]. Decision making is inevitable when managing requirements [60] as most systems evolve around the initial set of requirements and follow changing customer needs. Decisions in the requirements engineering processes can range from the organizational down to the project level [24], [28] creating a need for a supporting method that can visualize the consequences of these decisions for the entire project. Requirements decision making shares several challenges with release planning and prioritization, e.g. incompleteness of available information [28], changing customer needs, finding the right balance between the commercial requirements over internal quality requirements [22], conflicting priorities between stakeholders [22], dependencies between requirements [55], and the identification of business goals when performing release planning [28]. Moreover, decisions in requirements engineering are often semi-structured or unstructured [28] and decisions often need to be changed or altered [10]. The above mentioned challenges constitute a need for further research on providing decision support tools [24]. FSC+, decision archetypes and atomic decision visualizations analyze and visualize the available past experiences as a support for future decision making improvements.

2.4 Requirements Scoping

Defining project scope that fits into the project schedule is prominent in the project and product management literature [1], [61]. The term "requirements scoping" was introduced later as defining the common part of a software product line [62] and requirements scoping is considered a core function in software release planning for software product lines [63], [64]. The software product line literature focuses on the identification aspect of scoping [63], [64]. However, scoping is a continuous activity, where overscoping [2] and scope reductions are frequent phenomena [10].

Several studies in the project and product management literature focused on the phenomenon of *scope-creep*, defined as uncontrolled scope expansion beyond initial commitments [1], [3], [4], [5], [11]. Scope-creep can, for example, be caused by sales staff agreeing to deliver unrealistically large features without checking the scheduling implications first [5] and by stakeholders unable to agree on project goals [3]. The phenomenon can have serious negative consequences for a project, including project failure [1]. To mitigate the negative effects of scope-creep, Carter et al. suggested combining evolutionary prototyping and risk-mitigation strategies [4] .

Another related phenomenon is *requirements-scrap* defined as a situation when the scope of a project both increases and decreases [11]. Kulk and Verhoef listed shrinking budgets or running out of time as the main reasons for scope reductions [11]. In our previous work, we identified six main reasons for defining excessive scope and confirmed that the phenomenon can have serious negative effects, including many changes after the project scope is set, quality issues, wasted effort and failure to meet customer expectations [2]. We also reported that changing stakeholder requirements and lack of resources are the main reasons for scope reductions [10]. The techniques presented in this paper support requirements scoping as a continuous activity and offer assistance in optimizing the decision making processes to avoid overscoping and scope-creep.

2.5 Visualization in Requirements Engineering

Visualizing information improves comprehension, particularly with multi-dimensional data sets [65] such as the requirements engineering decision making tasks [66]. Despite the communication intensive nature of requirements engineering and software development, only a small number of studies focus on visualization in requirements engineering [66]. Visualizations appear to be principally used in later phases of the development lifecycle [66], e.g. to improve mailing list communication during the development phases of a project [67], and to visualize software evolution [68].

In requirements engineering, visualizations seem to help with: (1) visualizing the structure and relationships between requirements, e.g. using graph-based visualization [69] for exploring traces between difference requirements, (2) supporting requirements elicitation, e.g. by creating a 'Rich Picture' of the system to be developed to explore the complexity before any subsequent analysis [70] or (3) modeling, e.g. the i* [71] that provides visual modeling of requirements specified in a formal language to enable improved validation. However, only a small number of studies focused on providing visual assistance for decision makers in requirements engineering, indicating that the benefits of information visualization techniques [66] are yet to be fully explored.

2.6 Scalable Support for Requirements Decision Making

Providing scalable support for requirements management and decision making is central for efficient requirements management in very-large scale projects. Regnell et al. proposed a requirements engineering classification scheme based on the number of requirements and dependencies between them, claiming that most research papers seek to validate a proposed requirements engineering method or tool in small and medium scale contexts [9]. Berenbach et al. reported that a common misconception about requirements engineering is the expectation that processes that work for a small number of requirements will scale [72]. Despite the evidence that scalable requirements engineering tools and methods should be implemented before the company and the number of requirements begin to grow rapidly [72], [73] the body of published work addressing large or very-large requirements engineering contexts remains small, e.g. see Table 1.

Among the related work that reports a technique or solution suitable for large-scale software engineering contexts, Garg presented an information management model for large automotive projects [74]. Among related work that reports empirical evidence from large-scale contexts, Konrad and Gall reported lessons learned from large-scale requirements engineering projects—mentioning scope change and scope-creep as one of the main challenges [75] and Ebert presented a technique for pragmatically dealing with non-functional requirements in large systems [76]. Boehm identified

increasingly rapid change as one of the future challenges for software engineering processes [77] while Northrop et al. explored challenges in Ultra-Large-Scale systems (ULSS) [78]. Across this body of related work, little research was reported for supporting large-scale requirements decision making by using visualization techniques.

2.7 Summary

FSC+ supports large-scale decision making via decision archetypes, atomic decision analysis and visualizations. The technique supports large-scale dynamic re-planning and scalable retrospective analysis that provides a basis for experience based criteria [79] for guiding decision makers. FSC+ is designed to help identify process overloading [8], [58], and to find an appropriate balance between market-pull and technology-push [8]. Visual representations used in this paper are designed to visualize scope evolution and advance comprehension [65] of large-scale decision making which can help to better control lost opportunity costs [29], requirements-scrap [11], scope-creep [3], [5] and overscoping [2]. The real-time feedback for ongoing project management provides a scalable decision support tool for release planning on both strategic and operational levels [27].

3 Case Study Description

The case study was conducted at a large company that delivers embedded software systems for the global market utilizing a product line approach [63]. The company has approximately 4,000 employees located in two main offices. The first office is responsible for the hardware technology, manufacturing processes, technological innovation and management. The second office is responsible for software development and management. This study was conducted in the software development and management office. The company ships approximately 40 million devices per year, as of 2013. The operational environment for the company is highly competitive with several strong competitors and high market pressure. As a result, time-to-market becomes vital and strongly enables the success of the proposed technological innovations. Delays by months have severe adverse effects on sales.

The company has recently transitioned from a phase-based process to an agile-inspired methodology including ideas and principles from eXtreme programming (XP) [57] and Scrum [80] and today uses an iterative and incremental development model. This change in the process created a need for embracing change [80] and improving decision support capabilities [24] via designing scalable visualizations that enable visual analytics of scope evolution.

The company has adopted the following agile-inspired practices: (1) one continuous scope and release planning flow, (2) cross-functional development teams, (3) gradual and iterative detailing of requirements, (4) integrated requirements engineering, and (5) user stories. The company emphasized user stories for capturing user intentions and user story acceptance criteria to ensure that user stories are correctly implemented. Requirement details are now iteratively refined and all requirements related work is

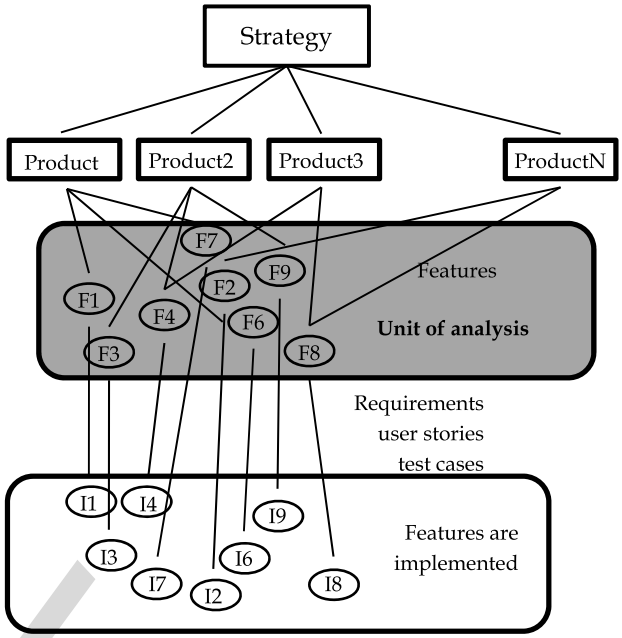

Fig. 1. The company's context.

delivered by cross-functional development teams containing developers, business analysts, requirements engineers and software architects.

The company's context is depicted in Fig. 1. The corporate strategy describes the long term (two to three year) goals and strategies, similar to the roadmaps described by Regnell and Brinkkember [8] or concrete product strategies as described by Khurum and Gorschek [81]. Based on the goals and strategies, the company releases about 30 products each year, with individual product definitions containing up to 100 new features. A *feature* is, in this context, defined as a group of requirements (usually 20 to 50) that constitute a decision making entity for scope management and value for the customers [10] and is often user-visible [82]. As a result a typical product may contain up to 5,000 requirements and there are several products in parallel development. Some features introduce completely new functionality to the common code base (the platform) while other features are internally driven quality improvements of the existing features. Finally, some feature are developed as *enablers* for upcoming functionality or technology and these features can represent a significant portion of the final development cost.

Some products are only small adaptations of previous products while others contain many new features. Features are elaborated as user stories and as requirements and then implemented using test-driven development. Since the company operates in MDRE where direct customer contact is difficult, customer requirements arrive indirectly from the marketing department, competitive analysis and other indirect sources. Members of these departments take active roles in the requirements refinement and decision making processes.

Feature management is performed using the state-machine shown in Fig. 2. Each newly created feature begins in an administrative state called *New Feature* (NF) then enters the process at state *M0*. *M0* validates that this newly created feature has a sponsor, sufficient business

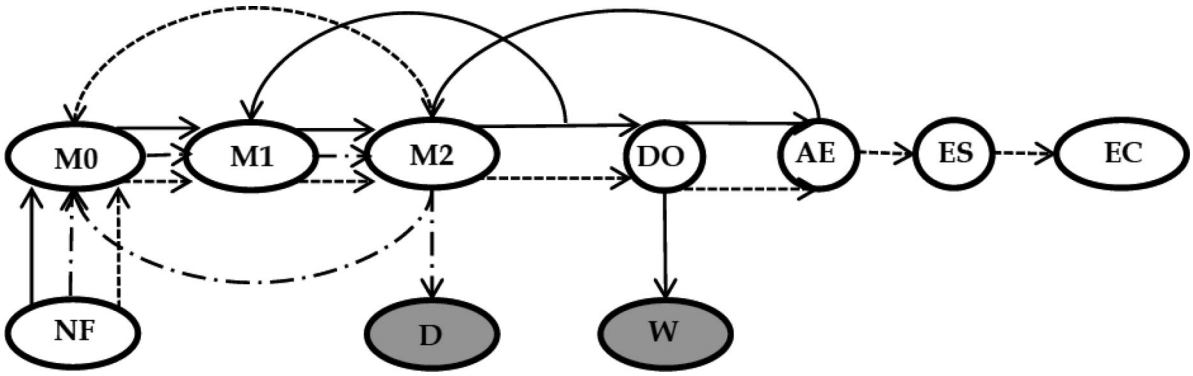

Fig. 2. An example history of changes for three features. The first feature, marked with dashed lines, was implemented. The second feature marked with solid lines was withdrawn. The third feature marked with dashed-dotted lines was discarded. The states are marked as follows: NF - New Feature, M0 - At M0 Forum, M1 - At M1 forum, M2- at M2 forum, DO - Definition Ongoing, AE - Awaiting Execution, ES - Execution Started, EC - Execution Completed, W - Withdrawn, D- Discarded. The states marked gray are the terminal states used in the analysis in Section 6.

justification and is in line with the associated product strategy. Upon validation, the feature is promoted to the state *M1*. Sets of features in *M1* are prioritized, across all product lines, by scope owners using a one-dimensional prioritization method based on business value. The prioritized list is later used by the development teams to guide implementation and integration scheduling. A feature may be returned to the *M0* or *M1* state from any state if further definition or refinement is required.

After prioritization, features are promoted to *M2*. Development resources are consulted and implementation schedules are defined within a continuous delivery pipeline tool that controls resource and delivery scheduling. It is not unusual for feature priorities to be modified at this stage due to changes in delivery date and resource constraints.

Features then pass through several development-related states, including: *Definition Started (DS)*, *Definition Ongoing (DO)*, *Awaiting Execution (AE)*, *Execution Started (ES)* and *Execution Completed (EC)*. A feature can exit this sequence at any time, transitioning to one of the following states: *Withdrawn (W)*, *Discarded (D)* or *Already Supported (AS)*. A withdrawn feature is usually an obsolete feature that is merged with another feature or already implemented. The discarded features are features that will not be implemented for whatever reason. In the three examples depicted in Fig. 2, we note that two features were sent back to the previous state. This behavior is not prohibited by the process but it negatively impacts overall development efficiency. While the visualization depicted in Fig. 2 can support the analysis of a small number of features, it is not suitable for the simultaneous analysis of hundreds or thousands of features.

Records of all state changes, including their time-stamp and reason, are stored in a corporate database. This development history, combined with feature attributes such as product line or stakeholder, was used as the data source for the FSC+ visualizations presented in this paper.

4 Research Methodology

In this section, we present the research methodology, the research questions, and the data collection methods employed in the use and validation of FSC+. The main goal of this research is to support decision makers with the visualization empowered by some visual scalability principles [83] that enables analysis capabilities that can be utilized to better understand the magnitude and frequency of scope management decisions. Current methods have proven inefficient given the size, complexity and the frequency of changes that need to be analyzed, decided upon and executed in a timely manner, see Section 2. FSC+ was designed to support industry with an effective method for supervising, optimizing, and maintaining scope changes and in response to the recent challenge of creating an understanding how visual analytics can best answer the requirements analyst's needs [13] .

4.1 Research Questions

The research questions investigated in this study are outlined in Table 2. The first research question (RQ1) is focused on different aspects related to the usability and scalability of FSC+. RQ1 evaluates the scalability of the visual notation employed for FSC+ and associated visual scalability components (pan, zoom and stacked bar charts). RQ1 also investigates FSC+'s ability to provide analytic support for decision making in VLSRE. RQ1 can be abstracted to the following hypothesis (not in a statistical sense): how can we apply visual analytics [12], supported by visual scalability principles [83] to the task of unlocking information hidden in large datasets generated by decision making processes in VLSRE.

Research question two (RQ2) focuses on how decision patterns facilitate continuous scope management, while research questions RQ2a and RQ2b focus on the usefulness of decision patterns as an analysis and visualization method. RQ2 can be abstracted to the following hypothesis (not in a statistical sense): can similar behaviors be identified among features and how can the identified patterns enable/facilitate decision-making-process assessment and improvement. Finally, does the relationship between lead-time and outcome in the identified patterns provide insight into decision agility.

4.2 Research Design and Operation

This study was influenced by the case study [84] and technical action research (TAR) [85] research strategies. At the meta-level, this study is a case study, however at the operation

TABLE 2
Research Questions

Research question	Aim
RQ1: How useful is FSC+ as a visualization technique for supporting continuous scope management?	Process changes at the case company, see Section 3, combined with challenges in decision making and release planning, see Section 2.1 increase the pressure for scope evolution monitoring and visualizations. Therefore, the goal for RQ1 is to evaluate whether FSC+ can answer industry demands in that regard. RQ1 focuses on zoom, pan and the visual metaphor as a mean to achieve visual scalability.
RQ1a: What should be visualized on the *Y*-axis of FSC+?	FSC+ supports various line thicknesses making the Y-axis scalable according to a given criteria. The goal for RQ1a is to investigate which attributes should be used to scale the *Y*-axis.
RQ1b: How should the temporal aspect of the *X*-axis of FSC + be visualized?	FSC+ provides variable time intervals and time-spans on the *X*-axis. The goal of RQ1c is to investigate the optimal time-spans for FSC+ based on practitioners' needs and opinions.
RQ1c: How should the information in FSC+ be sorted?	Various sorting methods for FSC+ can create various visual patterns. The goal of RQ1c is to investigate which sorting mechanisms are desired by practitioners.
RQ1d: How can filtering be used to improve the visualization capabilities of FSC+?	FSC+ supports filtering based on any available feature attribute. The goal of RQ1d is to investigate mechanisms for filtering the information to be visualized by FSC+.
RQ2: How do requirements decision patterns facilitate continuous scope management?	According to the process description, see Fig. 2, there is an optimal way to promote a feature through the process. Practically, features could be sent back for clarifications or skip some states if they are re-entering the process. The goal of RQ2 is to investigate the number of decision patterns and how their analysis can facilitate scope management monitoring and could provide valuable feedback for minimizing waste and supporting operational or dynamic re-planning [18].
RQ2a: How does the analysis through decision archetypes enable an understanding of the requirements scoping process?	The organization has a need to learn, to improve decisions, and not repeat mistakes. We investigated whether decision archetypes could be used to facilitate learning and improvement.
RQ2b: How does visualization of decision patterns act as a facilitator for process improvement and learning?	An efficient visual representation of the nature of decision patterns can facilitate learning and improvement and assist managers in making process improvement decisions. The goal of RQ2b is to investigate possible ways of visualizing decision patterns to visually support decision making process improvement.

level, we applied TAR practices. TAR has a clear change agenda, which we interleave with the case study observational focus on evaluation and assessment of the change. This combination enabled obtaining practical knowledge in its natural context [86], as a valuable source of information [87] and introduced necessary flexibility when obtaining empirical evidence [88]. Moreover, case study is argued to be suitable for software engineering research [84] and recommended for requirements engineering research [89].

As improvement and scalability were central to this work, we conducted the study influenced by TAR which enables us to design, implement, validate and evaluate FSC + within the engineering cycle [85]. During this cycle, the researchers interact with both the environment under research and the subjects in that environment who used FSC+ and reflected on it [84], [87], [88]. Our goals were not only to understand the scoping process at the case company, but also to make improvement proposals that resulted from applying FSC+ and the accompanying decision patterns and archetypes analysis techniques. In this way, we could both improve the state-of-practice and our understanding of the studied problem [89].

The details of the TAR applications and associated engineering cycle steps are outlined below. The FSC+ evaluation was performed in response to continuous feedback enabling gradual refinement, as depicted in Fig. 3.

4.2.1 Problem Investigation

Previous work about Feature Survival Charts [10], [17], [29], challenges in visual requirements analytics [13] and related research, see Section 2, have shaped and influenced the scope and goals of the current research. Several brainstorming sessions with practitioners were held to discuss the goals of the research and adjust it to meet the needs of the practitioners. The strong need for scope visualization was expressed during the meetings, with a special emphasis on the scalability of the visualizations. Practitioners agreed during the meetings that the visualizations should support both a high-level overview and detailed views for

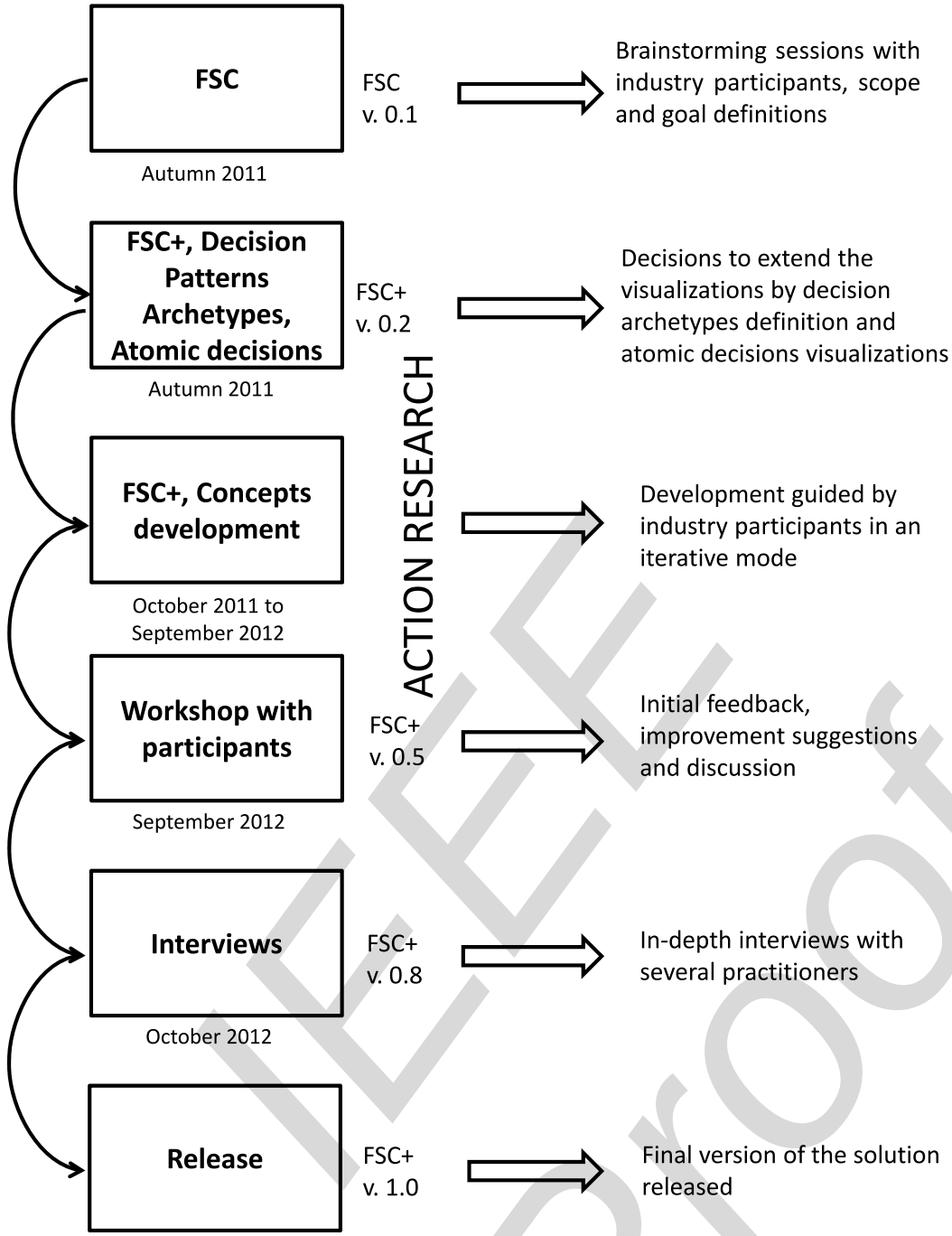

Fig. 3. Research process.

requirements scoping and decision making. The discussions led to the implementation of FSC+ version 0.1.

4.2.2 Treatment Design

During the development of a scalable visualization solution, a new need emerged concerning the analysis of decision patterns. This need emerged as a part of early (pre-development) scope visualizations where many changes and additions to the scope were discovered. When each state of the process was visualized with a different color, FSC+ uncovered interesting patterns of reappearing colors that identify features sent back in the process. To avoid cluttering FSC+ with too many details, it was decided to provide a separated analysis of decision patterns and their visualization and focus on simplifying FSC+ with the same green color for all in the process states.[1] Further, as the company works in a homogeneous domain, with an established customer base, and the company is well versed in the technology used, similarities in decisions were identified. This observation led to the decision to identify the decision archetypes. A decision pattern is seen as a unique path through the states that a feature undertakes. Further analysis also identified a need to derive decision archetypes for the most common decision making patterns. Finally, practitioners stressed the need to analyze and visualize decision patterns to enable the identification of common patterns.

1. An example visualization with unique colors is available at https://serg.cs.lth.se/fileadmin/serg/AllColors.bmp

TABLE 3
The Roles of the Respondents

Code	Role(s) within company	Years within role
A	Project Manager (10 years), Requirements Engineer (1 year), Product Manager and Planner (4 years)	25 years in industry
B	Software Manager (6 years), Requirements Expert (5 years), Requirements Manager (6 years)	23 years in industry
C	Requirements Coordinator (3 years)	9 years in industry
D	Key Customer Requirements Manager (2 years) Application Planner (3 years) Senior Manager Scope Planning (2.5 years)	15 years in industry
E	Software Product Manager (3 years) Project Manager (2 years) Requirements Engineering (3 years)	10 years in industry
F	Requirements Engineer (2 years), Product Manager (4 years)	15 years in industry
G	Senior Staff Engineer (12 years)	20 years in industry
H	Product Development (3 years) Requirements and Scope Manager (4 years)	10 years in industry
I	Requirements Manager (2 years), Product Manager (3 years), Senior Managers (3 years)	16 years in industry
J	Software Project Manager (7.5 years) Requirements Manager (2.5 years)	12 years in industry
K	(Senior) Project Manager (15 years)	30 years in industry

Decision patterns and archetypes also enabled the discussion about problem classes that can be derived from the particular problems at the company [85]. The discussions led to the implementation of FSC+ version 0.2.

The concepts discussed in the first phase of the study were then iteratively refined and developed by industry and academia over a period of almost one year [90]. Tool support for scope visualization and decision patterns analysis was developed in parallel. Researchers were granted access to a database containing the entire history of feature scope changes (for 8,728 features tracked over a three year period). The data extraction was conducted with the support of experts to assure correctness.

FSC+ was supported by a tool written in Java[2] that included functionality for parsing and calculating lead-times between various scope states. The tool was developed under six months of part time effort intertwined with frequent discussions between the first author who developed the tool and the industry partners. The development methodology used during this time was agile-inspired with short two-weeks development cycles separated by discussions and feedback from industry. The solution contains four main components: 1) the data parser, 2) the main component that contains the calculation methods, 3) the visualization component and 4) the analysis and reporting component that provides automated reporting in Excel format. The most challenging component turned out to be the parser since the company did not allowed direct Focal Point database access due to security concerns. As a work around, we decided to export the history of the state changes for the scope attribute into text files (and also other attributes as the parser can also parse other attributes). These text files are parsed and the state changes and dates are recorded, together with the lead-time in each state and the first entry to a given state. The parser detects the feature states using regular expressions, which means that any other state transition model could be supported without changes, see Fig. 2 for the examples of the feature state changes. The only requirement to detect these states is to provide the textual files compatible with Focal Point's textual export format.

The main component calculates lead-times in each state, records first entry to each state and provides configurable sorting and filtering capabilities. Depending on the provided parameters, the data set is reduced and passed over to the visualization component that visualizes the features. The visualization component provides pan and zoom capabilities and shows various details depending on the configuration. To avoid cluttering the figures, these details are only presented in Fig. 6. The tool also produces Excel reports that summarize the lead-time for each feature in each state and other statistics, depending on the configuration. This component is also responsible for the decision archetypes derivation.

4.2.3 Design Validation

In September 2012, FSC+ version 0.5 was presented to 15 practitioners for focus group validation. The input from the workshop participants was used to further refine FSC+ and for designing the interview instrument used in the next phase of the study.

4.2.4 Treatment Implementation and Evaluation

When deciding which practitioners to interview, see details in Table 3, we used a combination of convenience sampling and variation sampling [91]. We managed to interview over 50 percent (11 out of 20) practitioners that used FSC+ to support decision making. One practitioner at the case company acted as a gate keeper ensuring that the selected participants were representative and suitable data points for the study. We focused on inviting senior roles (with nine or more years of industrial experience) from project management, requirements analysis and engineering, software product management and senior management. Semi-structured interviews [88] were conducted using a set of predetermined questions whose order was changed at the interviewer's discretion (see Appendix A). A total of 11 respondents were interviewed in 60 minute sessions and the roles of the participants are outlined in Table 3. Each interview contained significant discussion and included the results from the dataset analysis and our interpretations of the data and our findings. The

2. The architecture diagram of the tool is available at https://serg.cs.lth.se/fileadmin/serg/ClassDiagram.png

interviews were recorded and analysed and the results from each interview were sent back to the respondents for validation. Finally, version 1.0 of FSC+ was released based on the feedback from these interviews.

4.3 Validity Evaluation

We discuss threats to validity according to the four perspectives on validity proposed by Yin [86] and some of the guidelines provided by Runeson and Höst [84].

Construct validity is concerned with establishing appropriate methods and measures for the studied phenomena or concepts. By using the scope change history stored in the database, we minimized the subjectivity of analyzed data. Further, the process of data extraction was supervised by one practitioner from the case company who ensured that all irrelevant data and administrative scope changes that could have skewed the results were removed. Furthermore, we used interviews as a second source of evidence about the studied phenomenon to increase our understanding of the data and the quality of our analysis. The way interview questions were constructed may also pose threats to construct validity. In particular, there exists a risk that questions are stated in a way that requires generalization of participants' experience. As generalization of experience is difficult and often biased (e.g. Siegmund et al. listed nine categories of programming experience and most of them are subjective [92]), we ensured that the participants provided their opinions about FSC+ based on their experience with using it and therefore we believe that we increased the reliability of the received answers. Still, the reader should keep in mind that the results of this study should be interpreted within the studied context; discussion of transferability to other contexts is provided in Section 7. Finally, all features available in the requirements database at the time of the study were analyzed, increasing the construct validity of the findings, especially with respect to the decision archetypes and atomic decision visualizations.

Internal validity is concerned with uncontrolled confounding factors that may affect the studied causal relationships. The main purpose of this study is not to determine whether certain events led to other events and therefore threats to internal validity have limited implications in this work. Still, the phenomena discovered during FSC+ application and decision archetypes identification and atomic decision visualizations discussed with practitioners during the interviews elicit feedback and discuss possible causes, e.g. for the high number of decision patterns or state transitions such as those visualized in Figs. 7 and 8.

Reliability is concerned with the degree of repeatability of the study. We have reported the procedures used in the study to avoid biases and enable seamless replications. The analysis of feature scope evolution was automated and thus complete replicability was ensured. The decisions about which administrative states were removed can be provided upon request. The decision archetypes were automatically derived from the dataset using unique pattern identification which also increases reliability. The results from the interviews were recorded and transcribed to enable further analysis. The analysis procedures for the interview data were recorded to ensure that the data can be reanalyzed in the same way, if required. Finally, the respondents acted as independent reviewers of the analysis results for the scope changes database.

External validity is concerned with the ability to generalize the study's findings. One of the five misunderstandings about case studies that Flyvbjerg addresses is the inability to generalize from single case studies [93]. We are aware that our results are based on a single case study with a single unit of analysis [84]. However, as pointed out by Flyvbjerg, case studies should focus on analytical generalization rather than statistical generalization by comparing the characteristics of a case to a possible target. Therefore, to support transferability, we have included case-specific characterizations of the context and the system domain, with due consideration of confidentiality. We have utilized case based reasoning and generalization by analogy to strengthen external validity and discuss scalability to practice [90].

The fact that all 8,728 features in-process at the case company during the study period were analyzed, and not a subset, greatly increases the external validity of our findings. Still, this study is based on data from a specific case, and the statistically significant results presented in Section 5 should be interpreted with caution. Moreover, the main goal of the study was to propose supporting technology for large-scale requirements scoping in relation to the context of the case rather than to make generally applicable statements. We believe that the two main threats to external validity of this work are: the feature management process used by the studied company, explained in Section 3 and in Fig. 2 and the data format used by the company. The current tool provides sufficient flexibility to visualize alternative processes and to parse other data formats. Despite these mitigating factors, the transferability of the results needs to be assessed by comparing our results to other cases in future work.

The discussion provided in Section 7 is based on the study results and the interpretation resulting from discussions between the authors of the paper and is not based on additional interviews. Thus, the suggestions provided in this section should be interpreted as hypotheses and further explored in future work. However, we would like to stress that all authors of this paper have extensive experience in working with industry as managers and consultants or as researchers working in close collaboration.

5 Feature Survival Charts+

This section gives an overview of FSC+ and illustrates its use with real examples selected from the case company. Later in this section we address each research question. FSC + is demonstrated in Figs. 4[3] and 6.[4] The visualizations give an overview of two significant development efforts at the case company (out of the 15 and 8,728 features analyzed in this paper). Fig. 4 visualizes application features that, to a large extent, represent market-pull [8] while Fig. 6 represents core component features that, to a large extent, represent technology-push. Figs. 4 and 6 visualize over 2,5 years of the requirements management and decision making

3. The full size color picture can be found at http://serg.cs.lth.se/uploads/media/Figure4.png

4. The full size color picture can be found at http://serg.cs.lth.se/uploads/media/Figure6.png

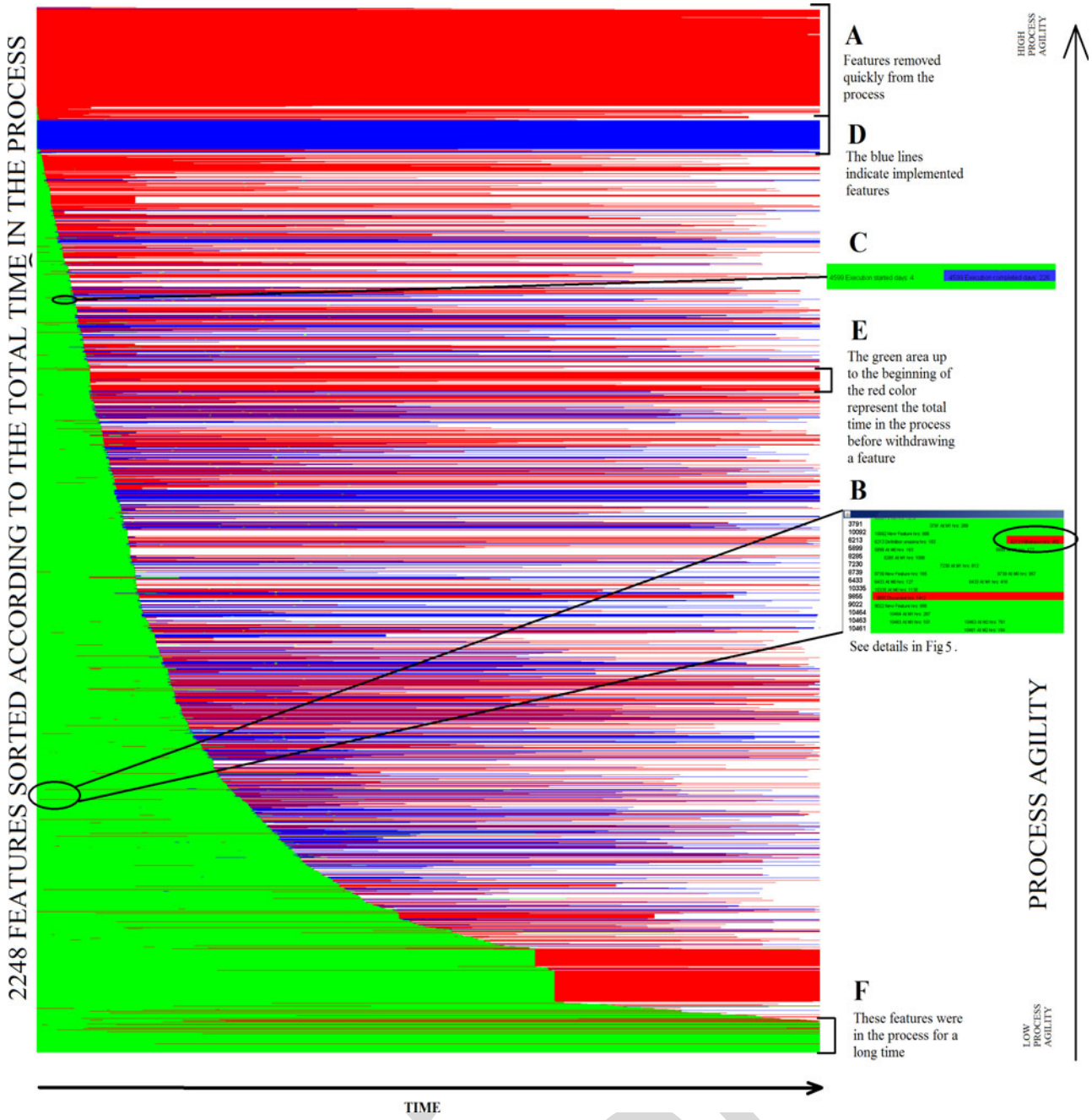

Fig. 4. FSC+ for the main application features stream of development at the case company. Active development is in green, completed development is in blue, canceled development is in red, and absence of further data is in white, as both features that reached their final states and those that did not are visualized on the same graph. Both axes have unit scale - one unit on the *X*-axis represents one unit of time, and one unit on the *Y*-axis represents one feature. A green line transitioning to red means that a feature was removed from the scope. A green line transitioning to blue means that a feature was delivered. The red area at the top of Fig. 4, see mark A, represents features quickly removed from the process.

process. The filtering capabilities, described in detail in Section 5.3, allow creation of multiple views; for example, visualizing only implemented features or features that are "stuck", i.e., not progressing.

FSC+ charts are generated using a Java implementation that supports dynamic filters, multiple *Y*-axis presentation sort orders and control of the meaning and length of the *X*-axis based on user preferences. The implementation also supports pan and zoom operations and data introspection for individual features. Control over the color scheme and temporal representations (absolute and relative) are also supported.

The company introduced an iterative development model with continuous creation and execution of features, see Section 3. The main reason behind which change was the need to increase the speed of innovation by continuously analyzing, prioritizing, developing and integrating features to the main code base. The company experienced an increasing number of incoming features combined by harder market competition and more frequent product releases. To remain competitive, the company had to be much quicker in analyzing, prioritizing and deciding whether or not to implement a feature. In the presence of uncertainty, the iterative approach where the critical

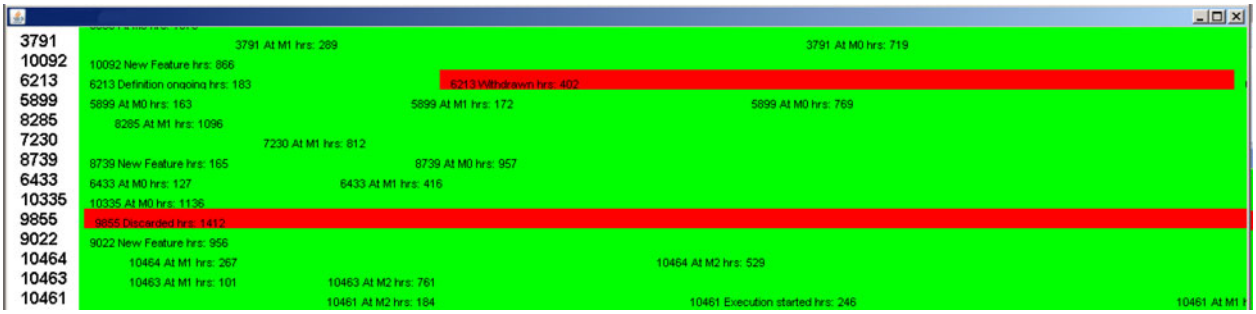

Fig. 5. An example of a zoomed in view from Fig. 4. We see an area of 14 features and their early life in the process. One feature (no. 9,855) was in the process for a very short time (a small green area to the left) and was early discarded.

scope for a certain software milestone is defined earlier and changes are incorporated later became dominating the previous planning method. This pressure for speed and performance, introduced the need for an improved scope visualization and analysis method.

Therefore, FSC+ charts are an evolution of the previously reported visualizations [10] with time now represented on the *X*-axis as a relative value: all features begin at the origin of the *X*-axis regardless of their actual start date. The life-cycle of a single feature can be followed by looking at the same *Y*-axis position in the chart [10]. Ordering along the *Y*-axis is based on the time spent in the process before being removed from the scope. Features that remain in scope for a long time are at the bottom of the chart, see mark F in Fig. 4. Features are presented in a sorted order on the *Y*-axis which puts features removed early in the release schedule (e.g. *Withdrawn* or *Discarded*) at the top of the chart, see mark A in Fig. 4. The various scope changes are visualized using different colors. Features still in scope are in green, completed features are in blue, and withdrawn or discarded features are in red. For simplification, the following states are colored green: M0, M1, M2, DO, AE, ES, EC. However, the FSC+ enables each of these states to be separately color-coded and in this way to uncover potential decision patterns, see Section 6.1. Similarly, both *discarded* and *withdrawn* states are colored red in Figs. 4 and 6 but they could easily be separated by using other colors.

There are 2,248 features visualized in Fig. 4, see also scenario 1 in Section 5.4. Active development is in green, completed development is in blue, canceled development is in red, and absence of further data is in white, as both features that reached their final states (scenario 2 in Section 5.4) and those that did not (scenario 4 in Section 5.4) are visualized on the same graph. However, FSC+ enables filtering out features that reached their final states. Features at the top of a FSC+ chart have spent less time in the development process, see mark A in Fig. 4, than features at the bottom, see mark F in Fig. 4. Given that time on the *X*-axis is relative, the birth date of a feature is not used in Figs. 4, 5 and 6. Both axes have "unit scale"—one unit on the *X*-axis represents one unit of time, and one unit on the *Y*-axis represents one feature. A green line transitioning to red means that a feature was removed from the scope (state *discarded or withdrawn*), see mark B in the magnified area in Fig. 4. A green line transitioning to blue means that a feature was delivered (state *execution completed*), see mark C in the magnified area in Fig. 4.

The red area at the top of Fig. 4, see mark A, represents features quickly removed from the process (the waste in release planning and analysis was minimal in this case [29]), see also scenario 3 in Section 5.4. These features were quickly analyzed by the company and the decision was made to not pursue further analysis or investigation. The magnified area in Fig. 6, seen as one of the "steps" in the visualization represents features that were discarded after approximately the same time in the process. This may suggest that all of these features were discarded based on one decision. The total length of the green line, until it turns blue, represents the total time in analysis and implementation for a feature, see mark E in Fig. 4. The blue areas at the top of Figs. 4 and 6, see mark D, represent features quickly implemented, see also scenario 2 in Section 5.4. For both application features (Fig. 4) and core component features (Fig. 6) significantly more features were quickly removed (see mark A in Fig. 4) than implemented (see mark D in Fig. 4). This imbalance may be an indication of two potential issues in the requirements management process: (1) the market research delivers suboptimal feature candidates or (2) these quickly removed features represent significant development effort and potentially long lead-times and several changes to the current code base that are not worth pursuing.

As we look further down in Figs. 4 and 6, we observe many blue lines which represent transition to the *Execution completed* state. As the blue lines are dispersed across the charts, we interpret this as an indication of a high variation in the development time. At the bottom of the chart, see mark F in Fig. 4, we see a green area which represents the features that were in the process for the longest time without decision, see scenario 4 in Section 5.4. The decision making process agility increases from the bottom to the top of the chart as the quickest decisions (to implement or to discard a feature) are visualized at the top.

The visualization offered in Fig. 4 facilitates identification of two behaviors: (1) the corporate response time for scoping decisions by tracking the boundary between areas in green and areas in any other color and (2) the behavior of features that remain in the process for extended periods, see mark F in Fig. 4 and scenario 4 in Section 5.4. The sooner a feature transitions to blue (analyzed and implemented, see marks C and D in Fig. 4) or red (removed from scope, see marks A and B in Fig. 4), the faster the response time and

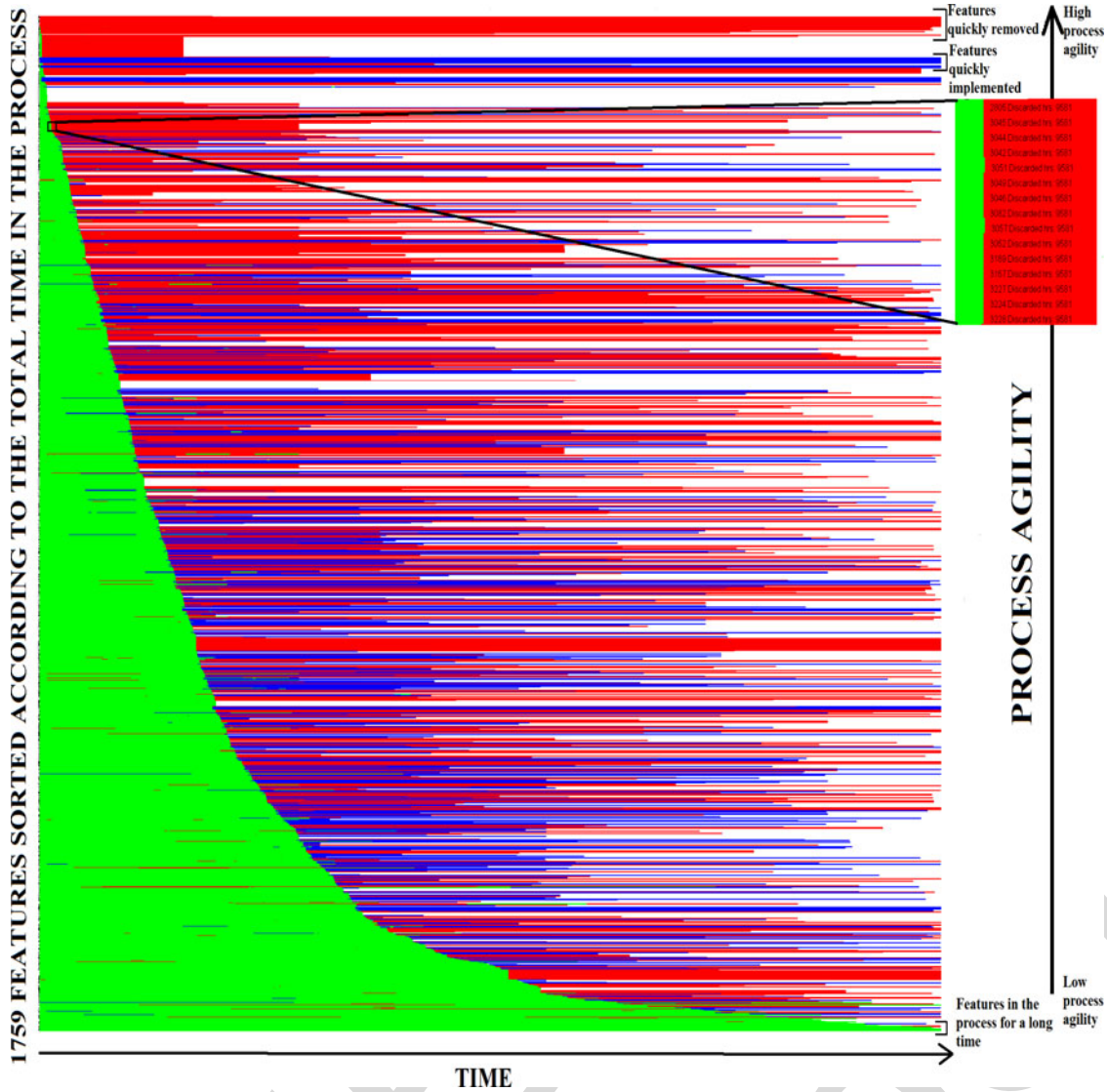

Fig. 6. FSC+ for the core software components stream of development at the case company. The total length of the green line, until it turns blue, represents the total time in analysis and implementation for a feature. The total length of the green line, until it turns red, represent the total time in analysis and/or implementation until a feature was withdrawn or discarded.

less opportunity for wasted effort [29], particularly in rapidly changing MDRE contexts. Decisions are quickly made for features at the top of the charts and the time needed for decisions increases when looking further down in the charts. The FSC+ overview in Fig. 4 shows that a relatively small number of features was quickly implemented, see mark D in Fig. 4. Practitioners can easily identify that there are fewer implemented features in the bottom 20 percent of Fig. 4 as the density of the blue lines is lower.

This easy to make observation (see mark D in Fig. 4) triggered further analysis of the feature decision archetypes, summarized in Section 6. The analysis confirmed our visual perception as only 437 features in the entire dataset were implemented in not more than three months and with three decision cycles or less. We also observe that the dominant color in the top 50 percent of Figs. 4 and 6 is red. This observation is confirmed in the feature archetypes analysis in Section 6 as 1,104 features were withdrawn in less than (or equal to) three decision cycles or less and three months in the process or less. At the same time, we can identify a significant number of green lines turning white or remaining green which represents features that are still in the process (they may be under implementation or awaiting a withdrawal decision). This observation is also confirmed by the analysis in Section 6 since 5,863 features were categorized as *evolving*. FSC+ gives a quick and efficient overview of the above mentioned phenomena and therefore provides an important link between operational release planning statistics and strategic goals for the efficiency of decision making.

A magnified view of mark B in Fig. 4, using the zoom feature, is shown in Fig. 5.[5] We see an area of 14 features and their early life in the process. One feature (no. 9,855) was in the process for a very short time (a small green area) and was discarded quite early. To avoid cluttering the view, this figure shows only the time and state names for features with more than 100 hours in process (the value is

5. The full size color picture can be found at http://serg.cs.lth.se/uploads/media/Figure5.png

configurable). The displayed information can vary depending on the magnification level.

A visual inspection of the main application features stream (Fig. 4) and the core software components stream (Fig. 6) over the same time shows a green region with a similar shape in both charts. However, the red areas seem to be larger in Fig. 4 (application stream). Further analysis revealed that 831 features from the core components stream and 1,351 features from the application stream were rejected. Similarly, there seem to be more blue lines among application features (Fig. 4) and core components features (Fig. 6). Thus, there seems to be an imbalance between the market-pull type of features and the technology-push type of features.

One possible interpretation of the imbalance may be that the case company is more active in eliciting requirements from customers (utilizing the market-pull strategy) rather than internally developing technologies [8]. This behavior is acceptable, but only to a certain degree, since implementing application features requires implementing core software components first (e.g. one core software component feature might enable several application features). Another possible explanation could be that the application features are less complex in their nature and therefore can be processed more efficiently by the process (later analysis of the time needed for implementation or withdrawal confirms this assertion). FSC+ can be used to initiate and facilitate discussions about whether or not this behavior is desired. FSC+ unveils these differences with a simple visual inspection, thus helping to create and continuously adjust [94] a competitive blend of technology-push and market-pull, recognized as one of the success factors for products and services [95].

There is also a blue region on both visualizations identifying features (see mark D in Fig. 4 and the top of Fig. 6) that were sent to the development team then quickly completed. This behavior is highly desired by the case company but it can be seen that only a small number of features exhibited this behavior. One possible explanation could be that the majority of the features are large and complex and therefore require substantial effort and long time to be implemented. Breaking down these features into smaller sub-features that could be more quickly delivered could be one of the remedies, also advocated by agile methodologies [80]. However, in our case significant time elapsed in many cases before the features were either implemented or withdrawn. FSC+ supports a rapid assessment of the proportion of features that demonstrate this behavioral pattern.

There were 428 core software component features (Fig. 4) and 635 application features (Fig. 6) that reached the execution completed state, see also scenario 2 in Section 5.4. The average period from entry into the process to a completed implementation was 132 days for core component features and 90 days for application features, a statistically significant difference (p-value < 0.001 for two-sided Kolmogorov-Smirnov test [96]). This difference is logical as core software component features often have hardware dependencies and constraints that may increase their implementation time. Managers can use this knowledge about the significant execution time differences when planning product content and balancing technology-push related features with market-pull related features [8].

A total of 3,818 features were withdrawn or discarded from the two feature streams (Figs. 4 and 6) and their average time in the process was 62 days for the application features and 102 days for core component features, see also scenario 3 in Section 5.4. The difference is statistically significant (p-value < 0.001 for two-sided Kolmogorov-Smirnov test). Managers should be aware that the analysis and negotiation for technology-push related features can take up to twice as much time as for the application related features. This result is important from the waste reduction perspective as one canceled or delayed technology-push feature could potentially cause twice as much waste as one market-pull feature. The average implementation time, including software definition, was 57 days. Requirements analysis, definition and decision making took, on average, 17 days longer for withdrawn features than for implemented features. This can have several reasons but also provides clear evidence supporting the introduction of waste reduction methods for process optimization [29].

Next, we look at the time spent in requirements definition, analysis and decision making for features that were implemented or discarded. FSC+ enables filtering of withdrawn features, implemented features and sorting according to lead-time in a given state. Implemented features spent an average of 21 days in requirements definition and decision making while withdrawn or discarded features spent 44 days in the same states. This difference is statistically significant (p-value < 0.001 for two-sided Kolmogorov-Smirnov test). This result is especially interesting from the process efficiency and waste reduction perspective. If we assume that a negative decision about a feature is neither financially risky nor process consequential (e.g. no further actions upon a rejected feature need to be made) the decisions could have been made faster and reduce waste. However, our study shows the opposite behavior which confirms the results presented by Kabbedijk et al. [97], who also reported that more time is needed to reject a change request than to accept it. One possible interpretation could be that since the company operates in an MDRE context it has to follow the customers' needs and requests and therefore may struggle to say no to many features, trying to implement as many as it can. Managers using FSC+ receive an opportunity to get an immediate overview of features and their state which could enable them to notice decision making lag.

5.1 FSC+ Supporting Scope Management (RQ1)

FSC+ visualizations were presented to industry practitioners in a workshop setting, see Fig. 3. All workshop participants expressed positive opinions about FSC+. According to the participants, FSC+ charts facilitate a quick review of scoping history. However, explaining FSC+, and how the visualizations can be interpreted, took more time than expected. In particular, explaining the sort method chosen for the visualization and the accompanying reasoning required extensive explanations for the participants and several ways of sorting the charts were discussed.

The color scheme was readily understood and participants considered color an effective presentation of, for example, how many features were withdrawn or

implemented. Participants considered the ability to visualize all features in database useful, but felt that visualizing subsets, e.g. one product or one development stream, was considerably more useful.

During the interviews, see Fig. 3, process managers and those who work with process development graded FSC+ as *very useful*. Respondents A and I gave FSC+ nine out of 10 points, mentioning that FSC+ was (*extremely useful*) while three respondents (E, H and K) graded FSC+ as *very useful*. Respondent G mentioned that FSC+ could be useful for process managers and process owners. Respondent A stated:

"I want to use these charts 1-5 times a week." This was confirmed by Respondent I.

However, respondents who worked with limited sets of features on a daily basis (usually not more than 100) graded FSC+ as only *partly useful*. Respondent B mentioned that

"I remember what is happening with my features so this solution is not particularly useful for me."

Similar statements were made by respondents C, D, F and J. The usefulness of FSC+ appears to be correlated with the size of the dataset, reaching its full potential for very large datasets. Nevertheless, the respondents who worked with limited sets of features on daily basis (up to 300) expressed an interest in analyzing and visualizing the subset of features for which they were responsible.

All respondents considered the pan and zoom capabilities as very positive. Respondents A and H suggested that the panning capability facilitated direct comparisons between adjacent charts, facilitating alignment. Respondent B, who worked with features on a daily basis, did note that filters could be used instead of zooming.

5.1.1 Tasks that FSC+ Could Support

Interview respondents identified several further tasks that FSC+ could potentially support. Communicating the current scope and scope evolution were the most frequently mentioned tasks (mentioned as a challenge in [73] and associated with balancing market-pull with technology push [9]). Seven respondents (A, C, D, G, E, I, K) mentioned that FSC+ could provide an overview of the "scope history" and the "health of the scope" (mentioned as a challenge in [66]), which could be used as support for high-level decision making. Respondent B mentioned that FSC+ allows users to see the process from outside without learning or analyzing all the details. Respondent H suggested that FSC + could help to visualize congestion [7] and thus help to explore process bottlenecks [58] and overloads [7]. Respondents A, E, I and K mentioned that FSC+ can help analyze features that are "stuck in the process" (potential causes of significant waste) mentioning that FSC+ is much better than the current database filters since it also visualizes how long the features remained in the process without a final decision. Respondent H suggested that FSC+ could visualize how various sub-processes are synchronized, e.g. requirements definition, decision making and development processes. Another respondent (J) suggested that FSC+ could show the wasted implementation effort for features that were sent to implementation and then canceled. The use of lead-time and waste reduction metrics [98] obtained from FSC+ was considered useful from a process efficiency point of view. Nine respondents pointed out that the ability to compare FSC+ visualizations that were generated for differing criteria was *very useful* for improved process understanding, and could provide valuable feedback for process improvement activities.

5.2 *X*-Axis and *Y*-Axis Representations: RQ1a, RQ1b

The majority of the interview respondents (B, E, F, G, I, J, K) suggested that the *Y*-axis should represent development effort, and the greater the effort required for a certain feature, the thicker the line. As a result, the green color areas for withdrawn features would more accurately represent the effort spent on them. At the same time, the green areas for implemented features would represent the estimated effort spent on them. However, it has to be recognized here that some discrepancies between the reported lead-time and the actual lead-time could be present. Thus FSC+ can support continuous cost management at the product level for both implemented and withdrawn features giving a basis for improved opportunity loss optimization and waste reduction [29]. Respondents C and H suggested that priority should be on the *Y*-axis, increasing the thickness on the *Y*-axis for prioritized features. However, three respondents (E, F and J) said that priority is changing frequently, and therefore it would be less useful than effort. Finally, respondent D pointed out that the current 'one-size for all features' representation is sufficient since it provides a good overview in relation to the number of features.

Regarding the *X*-axis, three respondents (B, E, J) suggested analyzing historical data one year in the past. Respondent F suggested that the unit on the *X*-axis should be one week and the maximum time should be half a year. However, two respondents (A and I) working with process management and improvement wanted to see up to two years on the *X*-axis. Respondent G suggested starting the FSC+ chart from the *M2* state focusing on discussions with development. Respondent E wanted FSC+ to start from the *definition ongoing* state (see Fig. 2), since that is the moment where the implementation effort and waste can be calculated. The FSC+ implementation was modified to allow starting the chart from any point on the *X*-axis, making it possible for multiple tasks to be performed and to satisfy the needs mentioned by the respondents in this section. As a result, FSC+ supports both high-level decision making process improvement and retrospective analysis of past decisions [45].

5.3 Sorting and Filtering FSC+: RQ1c and RQ1d

Respondents A, E, H, J, K suggested that FSC+ should be filtered per release project, supporting scoping at a product level. At the same time, respondents A and I suggested that looking at the entire development stream (just like in Figs. 4 and 6) is interesting from the management and overview perspective and this could support scoping at a product line level [15], [63] or even at a portfolio level. Respondent B wanted to filter out only legacy features (features reused or adapted from previous projects) and thus facilitate discussions about possible integration into the common code base of the product line [15]. Six respondents (F, G, H, I, J, K) suggested the following filtering techniques: (1) executed

features, (2) withdrawn features and (3) features in the process. Among them, three respondents wanted to filter out features that were stuck in the process for a long time which could help to identify if they were obsolete [99] or if they were potential causes of waste. Finally, one respondent suggested filtering by the stakeholder that issued the features thus enabling rapid identification of biases towards some stakeholders [100] and how the corresponding features were handled in the decision making process.

Two respondents suggested sorting the charts by lead-time in a specific state. One respondent focused on sorting by lead-time in *M2* state (E), one (B) focused on the *M1* state. Sorting by *M2* state lead-time could give an overview of how long it took to reach an agreement with the development department and this could support (the handshaking with unclear implementation proposals [101]). Sorting by *M1* state could give an overview of how long the prioritization phase took, facilitating the identification of possible bottlenecks in the requirements prioritization activities [8], [56] and impediments to the requirements management processes [58]. Further, three respondents stressed that the current way of sorting, by the time spent in the process (see Figs. 4 and 6), is sufficient and provides a good overview of the agility of the analysis and decision processes. Finally, respondents A and I suggested sorting by priority and placing the most important features at the top of FSC+. This view facilitates comparing the lead-time for the most important features (placed at the top) with the least important (placed at the bottom) and could provide valuable insights regarding whether the development actually focuses on the most important features (from a prioritization perspective [22]). FSC+ was extended to support all of the needs mentioned in this section.

5.4 FSC+ Usage Scenarios

This section provides abstracted usage scenarios derived from the case analysis. The presented scenarios can be utilized at the project, product, portfolio and company levels since and each of the scenarios is time and process invariant.

5.4.1 *Scenario 1: Visualizing the Decision Making Process at the Company Level*

In this scenario, the company's executives can visualize the decision making process for all available features. FSC+ provides a general overview of decision agility—the balance between the number of implemented, withdrawn and "still in process" features. FSC+ offers visual indications of the variability of the development time and the withdrawal time and aggregates quickly removed and implemented features at the top for direct visual comparison. Associated decision patterns and archetypes provide general feedback about process adherence and potential process bottlenecks.

Descriptive statistics such as the average time to implement or withdraw a feature and the most frequently associated decision patterns are also provided. These can be directly mapped to time-to-market demands and translated into necessary improvement activities. Implemented and withdrawn features are readily identifiable providing direct feedback on the balance between the aggressiveness of feature triage versus the tendency to accept as many features as possible. The global decision patterns and archetypes analysis provides direct feedback about the most frequent decision patterns and the dominant decision archetypes. The atomic decisions visualization provides an overview of potential process bottlenecks and helps to identify congestion states.

5.4.2 *Scenario 2: Visualizing Implemented Features at the Project, Product, Portfolio and Company Levels*

FSC+ offers visual representations of the development time volatility for implemented features. Analysts can visually compare the number of quickly developed features to the number of features that required extensive development effort. The derived descriptive statistics can be combined with further filtering per project, product or portfolio, e.g. to compare different development structures within the company. Enriched by complexity measures, FSC+ can provide an efficient visual overview of the ability to implement complex features in a timely manner and their implementation time volatility. This information can be used by feature definition teams as they consider changes to the abstraction level of a feature or to the refinement and design processes. The fast execution and slow execution decision archetypes provide direct input for further analysis and reasoning about development processes. The atomic decision visualizations support identification of possible congestion states and can help to identify potential development resources shortages in critical process states.

5.4.3 *Scenario 3: Visualizing Withdrawn Features at the Project, Product, Portfolio and Company Levels*

By presenting only withdrawn features, FSC+ offers visual representations of the feature analysis and triage time volatility. Analysts can visually compare the balance between quickly withdrawn features and the features that stayed in the process for a long time. Features that stayed in the process for a long time are good candidates for waste reduction as they correspond to significant requirements analysis or even development effort [10]. Taking action to reduce these long leadtimes before feature are withdrawn is a process improvement that not only frees resources but also reduces the scoping complexity. Visualized withdrawn features provides further analytics for supporting decisions about the optimal time that a feature should stay under consideration and be granted investment or analysis effort. For some rapidly changing markets, this time could be as little as weeks, since prolonged indecision may render these features either obsolete or unprofitable. The fast rejection and slow rejection archetypes analysis provides direct reasoning for these decisions and the atomic decision visualizations helps to identify potential process bottlenecks that hinder efficient feature triage and process states where features are "stored" before their late removal.

5.4.4 *Scenario 4: Visualizing Features "Stuck in the Process" at the Project, Product, Portfolio and Company Levels*

FSC+ offers visual analytics for feature leadtime by presenting only those feature that have not reached their final state

TABLE 4
Feature Decision Patterns

Frequency	Type	Decision pattern
294	ToDismiss	M0 − > D − > M0 − > D − > M1
244	ToDismiss	M0 − > M2 − > M0 − > D
231	ToDevelop	M0 − > M1 − > M2 − > DO − > AE − > ES − > EC
201	InProcess	M0 − > M1 − > M0 − > NF
194	InProcess	M0 − > M1 − > M2
151	ToDismiss	M0 − > W − > M0 − > W
132	InProcess	M0 − > M1 − > M2 − > DO
109	InProcess	M0 − > M2
103	InProcess	M0 − > M1 − > M2 − > DO − > AE − > ES
102	ToDevelop	M0 − > M2 − > AE − > EC − > M0 − > EC − > M0 − > EC
98	InProcess	M0 − > M1 − > M0
93	ToDevelop	M0 − > EC − > M1 − > EC − > M0 − > EC − > M0 − > EC − > M0 − > EC
88	ToDismiss	M0 − > D − > M1 − > D
80	ToDismiss	M0 − > M2 − > D − > W − > D − > W
79	ToDevelop	M0 − > M1 − > M2 − > EC
75	InProcess	M0 − > M1
73	ToDevelop	M0 − > M2 − > EC − > M0 − > EC − > M0 − > EC
72	InProcess	M0
66	InProcess	M0 − > NF
64	ToDismiss	M0 − > M1 − > W

The feature state names are available in the caption of Fig. 2.

for a long time relative to the average. Analysts can quickly see how many of the features are "stuck" in the process for a long time and they can further estimate how many features still have an acceptable probability of reaching their final state within the allotted time. The feasibility of successful implementation for features "stuck in the process" should be strictly evaluated within the time-to-market constraints and taking action to drop these features with low probability of success helps to free resources for other tasks and simplifies the scoping process. Scenario 4 can also be directly compared to Scenarios 2 and 3 to identify whether the lead-time characteristics for implemented or withdrawn features are the same or different from those "stuck in the process". Further filtering based on additional feature characteristics may reveal other factors that could be used to support decisions as to whether a feature should be withdrawn or implemented.

6 Feature Decision Pattern Analysis and Visualization: RQ2

The overall motivation for identifying decision patterns was to identify similarities in decisions taken regarding features, enabling learning and possible predictability. If we could identify a "typical" behavior given a feature then decision-associated learning could be enabled, leading to process improvement efforts and potential scope management improvement. During FSC+ development, several views were created with distinctive colors for each of the visualized states. These visualizations uncovered an interesting phenomenon: several features were sent back in the process, sometimes more than once. As the frequency of this phenomenon turned out to be high in the studied data, a decision was made to simplify FSC+ by using only three colors and form a new analysis and visualizations that only cover this aspect. This helps to avoid cluttering FSC+ visualizations but the possibility of this confirmation remains in the implementation and works sufficiently well in the zoom view.

In this section, we identify, analyze and visualize feature decision patterns. Decision patterns are state changes that features undergo, example patterns are shown in Fig. 2. A total of 2,248 decision patterns were identified in the dataset and the decision patterns were analyzed in collaboration with industry participants from the case company. Table 4 depicts the top 20 decision patterns. The patterns were sorted according to their frequency in the dataset and each of the top 10 patterns represent decisions taken for over 100 features. The patterns were categorized into *To Dismiss (ToDismiss)*, *To Develop (ToDevevelop)* and *In Process (InProcess)*. *ToDevelop* decision patterns represent features that were implemented, *ToDismiss* represent features that were withdrawn or discarded and *InProcess* represent features still in the process.

Overall, there did not seem to be a dominant pattern in the dataset as the most frequent pattern was applied to only 294 out of 8,728 features. The two most frequent patterns were identified as *ToDismiss* patterns and had four or five state changes. In both patterns, features were sent back in the process to the initial state *M0*. Some example features that follow these patterns include: feature requests that were discarded quickly during feature triage, features that originate from smaller customers and do not fit to the current product roadmaps (the second most frequent pattern with 244 features) and some low priority features that are "stuck" placed low on the priority list and continuously down-prioritized by more important features (the most frequent pattern with 294 features). The third most frequent decision pattern was a *ToDevelop* pattern. Some examples of features that followed this patterns are well defined features that fit to the product strategies and are implemented without unexpected complexity issues or additional efforts. The third most frequent decision pattern (with 231 features), represent features that

went straight through the process, according to the official process description, and were not sent back.

Nine out of the 20 most frequent decision patterns were *InProcess* patterns. As these features had not yet reached their final states, we could assume that their final decision patterns would change. Five patterns were *ToDevelop* patterns and six patterns were *ToDismiss* patterns. Looking further at the frequency of the patterns, there seemed to be no clear dominance of one type of decision pattern over another. Therefore, we further analyzed the identified decision patterns to identify decision archetypes. A decision archetype is identified as a set of decision patterns that have similar lead-time and outcome characteristics. Decision archetypes form "common" decision patterns in relation to features in release projects.

According to feedback during the workshop (FSC+ version 0.5, see Fig. 3), the large number of decision patterns was highly surprising for the workshop participants. Several of them insisted on booking individual follow-up interviews to further investigate this phenomenon. In essence, the participants were interested in investigating the reasons for the large number of decision patterns. Respondents indicated that some variation of decision patterns could be a result of the process' flexibility; in theory the process allows for both forward and backward transitions between any two states for a feature in a release project.

At the same time, process guidelines attempt to enforce the most optimal chain of state changes that lead to feature analysis and implementation. Although our respondents expected many decision patterns with a substantial number of backward transitions (representing sending a feature back to investigation or decision making), the actual number of these transitions was much higher than expected. This negative experience triggered a hypothesis that the decision making process could have a hidden "bottleneck" [58] and created interest among our respondents to visualize decision patterns, see Section 6.3. We suspect that some participants of the process may have greater aversion to decision making and thus accumulate undecided features that are later trapped in the long investigation delays. This may have a potential negative impact on the company's market response and time-to-market efficiency as any delay in decision making delays timely delivery of software products.

6.1 Decision Archetypes: RQ2a

We identified five decision archetypes based on the dataset:

- *Fast execution*—is the most desirable decision archetype from the process efficiency and waste reduction perspective. Fast executions represent features that were quickly analyzed and executed with few or no backward transitions or decision cycles. A decision cycle happens when a feature passes the same transition between any two states more than once. In Figs. 4 and 6 these features should form blue areas towards the top of the charts. However, the fast execution archetype was not common at the case company. The third most frequent decision pattern (with 231 features, see Table 4) was characterized by not sending features back in the process. Further analysis confirmed the visual observations provided by FSC+ since only 190 features out of 8,728 went through the process in less than three months and with zero or one decision cycle. 437 features were implemented in not more than three months and with up to three decision cycles. An example of a fast execution could be a bug fix that, e.g. on average takes up to 20 hours to implement [102] or a well-defined low-complex feature that does not require additional clarifications and little development effort.
- *Fast rejection*—is the second most desirable decision archetype from the requirements and product definition efficiency perspective. We define a fast rejection feature as a feature that was rejected in less than one month from its inception and with not more than one decision cycle, thus causing minimal waste. In Figs. 4 and 6 these features form large red areas towards the top of the charts, see mark A in Fig. 4. Only 135 features were categorized as fast rejection based on the above criteria. Increasing the time to reject to three months (more than the average lead-time for all withdrawn features) gave 249 fast rejections. Therefore, it appears that the number of decision cycles is influencing the behavior more than the lead-time, since Figs. 4 and 6 visualize many early withdrawn features. Setting the thresholds to up to three decision cycles and three months lead-time identified 1,104 fast rejection features. An example here could be a feature that was removed during requirements triage [103] as it did not fit into the products strategies or was neither important nor critical. Q1
- *Slow execution*—is a still successful but less desirable decision archetype from the process efficiency perspective. Slow executions are features that were executed in more than three months. In Figs. 4 and 6 these features form larger blue areas towards the bottom of the charts. The intensity of the blue lines increases towards the bottom of the charts. 2,176 features were categorized as this archetype. Further, for 188 features there were four or more decision cycles before these features reached implementation. This suggests that these features had to be re-evaluated, perhaps sent back from implementation to the requirements analysis phase or their scope was significantly reduced due to unexpected budget constraints. The case company would like to avoid these scenarios as they impede development and decision making efficiencies. Some examples of slow execution features include *enabler* features that provide technical framework for other features that deliver value to the customer and therefore represent extensive complexity and implementation effort. Another example here may be a vaguely defined feature that needs to be extensively discussed and redefined before implementation.
- *Slow rejection*—is the decision archetype where a feature stays in the process for an extended period, circulates over the state machine and is finally rejected. In Figs. 4 and 6 these features form larger red areas towards the bottom of the charts. Interestingly, the intensity of red lines decreases while looking from

the top to the bottom of the charts. This suggests that the company is more eager to keep features under analysis than to cancel them. Therefore, there are many green lines that turn white in the charts—2,293 features took more than three months to reject or withdraw. Further, 617 features cycled three or more times throughout the state machine before being withdrawn or executed. This archetype is the main contributor to wasted effort during the requirements process and therefore slow rejection features should be carefully analyzed and actions should be taken to convert them into fast executions or fast rejections. Some examples of slow rejection features include features that are significantly challenging in terms of complexity and required technology and therefore delayed due to unexpected difficulties. These delays lower their priorities and cause their rejection. Other examples include features developed by with low quality and therefore rejected during the customer validation phase.

- *Evolving*—is a feature that was sent back in the process. 5,863 features (67 percent) were categorized as evolving, 1,950 features (22 percent) were not evolving and for 915 features it was uncertain whether they would be evolving or not as they had not reached one of the final states. The green areas at the bottom of Figs. 4 and 6 contain many evolving features. Since the company operates in a rapidly changing MDRE context, some of the evolving features that are in the process for a long time could be candidates for obsolete features [99] and are potentially significant waste. Therefore, the company needs to control the number of evolving features and adjust the maximum allowed time for their evolution to adopt to rapidly changing market situations and changing customer needs. While evolving more features provides more flexibility and delivers more potentially beneficial features, the situation hinders predictable and controllable product management efforts. It could also indicate potential weakness in the feature definition process and inaccuracies that need to be corrected. Some examples of evolving features include features that initially were highly relevant and important but down-prioritized after some delays by more important features and remained low on the priority list. The changes for these often partly-implemented features to be completed are low due to continuous inflow of more important features.

6.2 Interview Results Regarding Decision Patterns Analysis: RQ2

All respondents considered analyzing decision patterns useful. Five respondents (B, E, G, H, K) suggested that analyzing decision patterns helped to see how many backward and forward transitions were made (see Table 4), and thus assist in bottleneck identification [58] by looking at which states send the most features back in the process and why they were sent back. Respondent B stated: *"decision patterns analysis helps us to see how many ping-pong between the states we had."*

Five respondents (A, F, G, H and I) suggested that the decision pattern analysis could help in understanding to what degree the process was followed. This information is critical for planning process development and improvement and for checking to what degree the official process corresponds to the real process used, often a pre-requisite for any improvement activity [104]. Respondent G suggested that decision pattern analysis could help to understand how the process was implemented/interpreted by different parts of the company while respondent F stated that *"this analysis allows us to understand how the process was used in reality and if it was according to the official process description."*

All interview respondents were negatively surprised by the number of decision patterns identified. Respondent A suggested that the high number of decision patterns (and many state changes) could be a result of using the process to discuss features by changing the states back and forth rather than making a state change after these discussions. Respondents I, J and K pointed out that the high number of patterns was related to intensive administrative changes in the features database. Further it may have been a result of incorrect process understanding among some practitioners and thus incorrect usage of the tool in relation to the process description. Finally, respondent H stated that:

"this analysis is very interesting and should be presented to the top management."

6.3 Visualizing Decision Patterns: RQ2b

The significant number of decision patterns identified, see Table 4, triggered the exploration of an efficient visual representation of the nature of decision patterns. In particular, we focused on understanding the magnitude of atomic decision in decision patterns that could unveil states in the process that could overload both inbound and outbound transitions and help to identify process bottlenecks [58]. In response, we utilized flow graphs with visually weighted edges to represent the number of atomic transitions. Figs. 7 and 8 depict atomic decision visualizations with arrows representing atomic transitions. The length of the arrows does not carry a meaning. The width of the arrows represent the frequency of a certain transition in the entire dataset. The transitions are divided into *forward transitions* (Fig. 7) and *backward transitions* (Fig. 8). Forward transitions are transitions that lead to implementation while backwards transitions lead to rework. Transitions from the two terminal states (*withdrawn* and *discarded*) and from *execution completed* are considered to be backward transitions.

The most frequent forward transition among all analyzed transitions is the transition between the states *M0* and *M1*, see Fig. 7. This behavior was expected according to the process description, see Section 3, as the purpose of the *M0* state is to evaluate the business and technological feasibility of the proposed features. After the evaluation at the *M0* stage, features are further evaluated during prioritization at the *M1* stage. The second most frequent transition was between *M1* and *M2*. Fig. 7 reveals that there were over 2,000 fewer outbound transitions (e.g. see Fig. 7) from the *definition ongoing* state than inbound transitions into this state. This may suggest that the *definition ongoing* state is a potential bottleneck in the process.

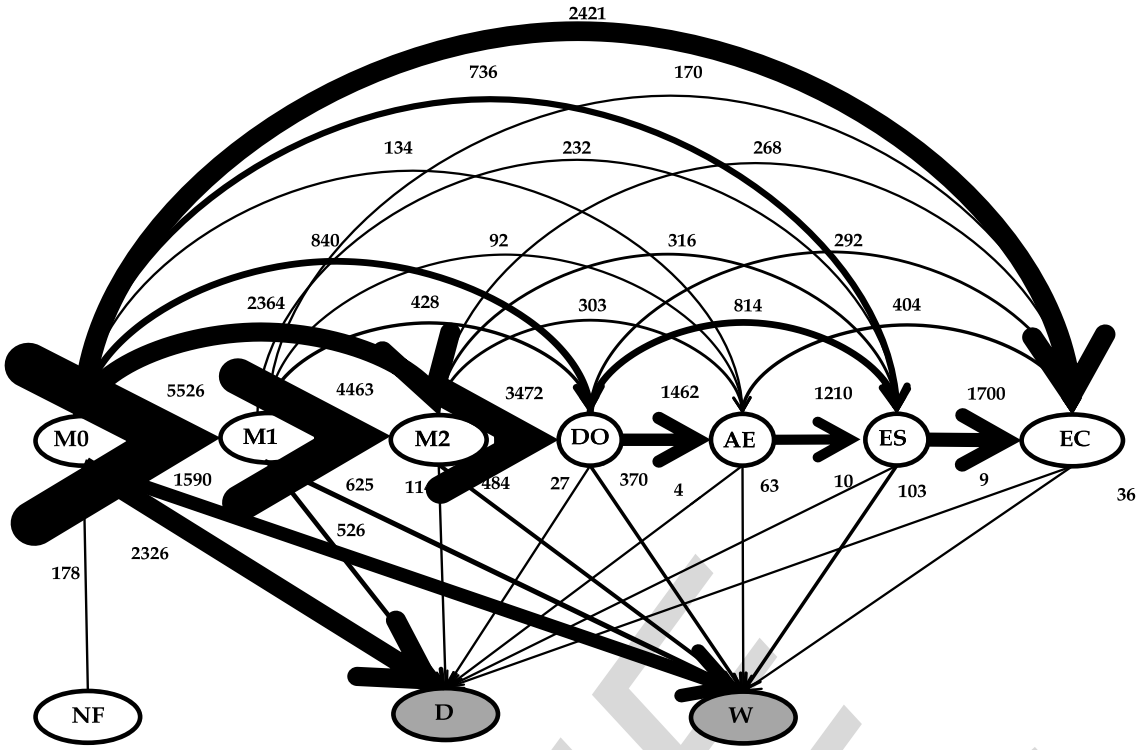

Fig. 7. Forward transitions visualized.

Interestingly, the fourth most frequent transition in the dataset is the transition between *M0* and *execution completed*, see the top of Fig. 7. This transition was a way of "bypassing" the entire process and sending the features from the start to the end. Administrative changes in the data were one of the reasons for these transitions together with the fact that several features cycled back and forth in the process. Finally, when we compare forward transition visualization (see Fig. 7) with backward transition visualization (see Fig. 8) we can see a clear dominance of forward transitions. This means that the overall process is tuned to push features forward in the process or cancel

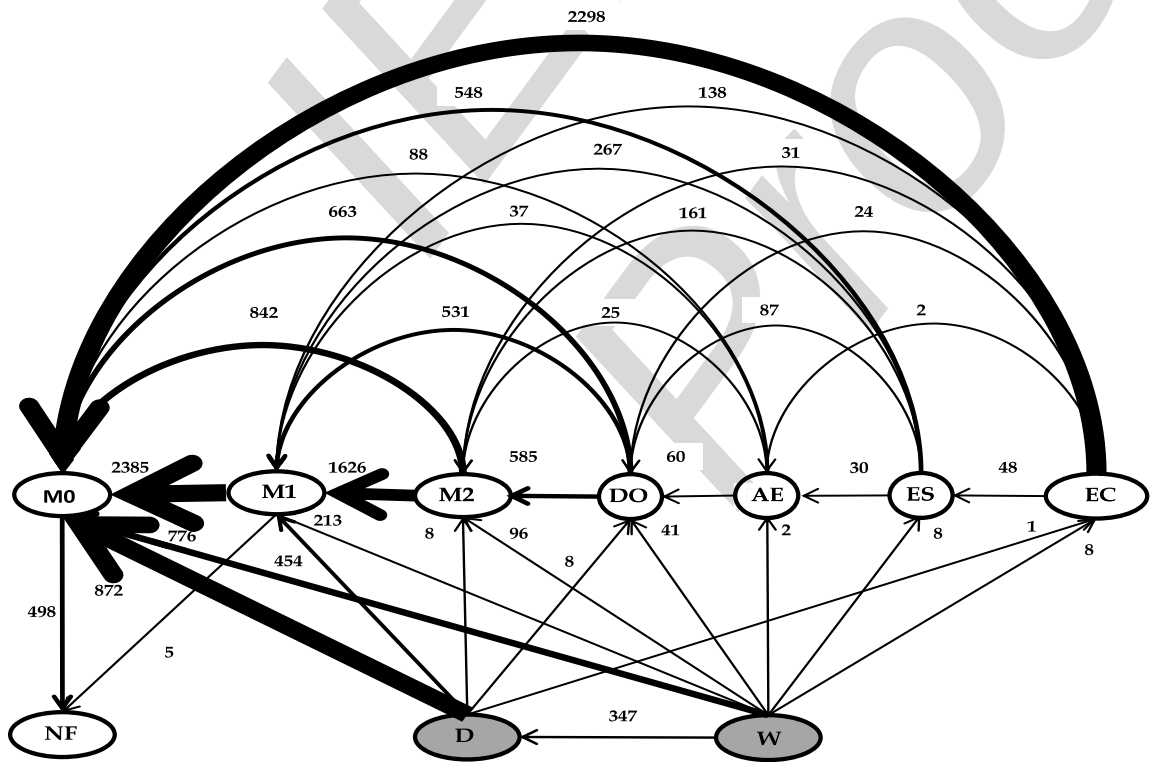

Fig. 8. Backward transitions visualized.

them rather than sending them back. This is a desired behavior that needs to be carefully designed so that the flow of implemented or rejected features balances the incoming flow of features.

6.3.1 Results from Interviews

Respondents C, D, F and G considered the atomic decision visualizations as *partly useful* and the remaining respondents (A, B, E, H, I, J, and K) considered them *very useful*. The reason why some respondents considered the visualization only partly useful was because they would prefer the atomic decision visualizations filtered by the development streams, just like in Figs. 4 and 6, as every organization could have a slightly different way of using the process. However, respondents B, E, H and J agreed that the visualization of the entire dataset shows general process adherence at the case company.

Respondents working with process improvement and management (A and I) considered the visualizations as *very useful* and pointed out that they can be effective support in discussions with high-level management. The most interesting transition to these respondents was the transition *M0 − > EC (Execution completed)*. Respondent I mentioned that these transitions are unusual according to the official process description and thus it is very valuable to investigate the reasons for this process behavior. Respondent K suggested that the transition *M0 − > EC* might be used to clean up features that were stuck in the process for a long time (*execution completed* state was used instead of *withdrawn* or *discarded*). Respondent A suggested that the visualizations (Figs. 7 and 8) clearly show that the process was used as a way to communicate and resolve questions regarding features instead of email or in-person negotiations.

All respondents confirmed our assumptions that the *definition ongoing* state could be a potential process bottleneck, visible in Fig. 7. Our respondents gave the lack of resources as the main reason for the significant difference between the number of inbound and outbound transitions at the *definition ongoing* state. Limited resources often resulted in sending a feature back to re-prioritization (585 transitions directly from *definition ongoing* to *M2* state, see Fig. 8) or back to the business feasibility analysis stage (842 transitions from *M2* to *M0*). At the same time, we should notice that over 800 transitions were made directly to the *execution started* state (from *definition ongoing* state) which is definitely a positive behavior from the process efficiency perspective as these features were not waiting for execution. Overall, Fig. 7 suggests that the filtering at the *M0*, *M1* and *M2* stages seems to be rather efficient, as designed in the process, and therefore the development resources are allocated for analysis and implementation of rather thoroughly investigated and economically sound feature concepts.

Respondents B, H, I, K mentioned that Figs. 7 and 8 clearly show that some people did not use the process exactly according to its formal specification. Finally, one respondent (H) pointed out that the visualizations show that the democratic model was insufficient at the company and suggested that a dictatorship model would have been much better in this case. In the democratic model, decision about features are made via a consensus followed by discussions with interested and impacted stakeholders. In a dictatorship model, decisions are made by one person based on her personal opinion and not necessarily in consultation with others. The respondent's conclusion is especially interesting in relation to the Ultra Large-Scale Systems (ULSS) literature [78] which suggests that for ultra-large systems overall control is not possible and thus these systems need to exist as separately managed systems. This would suggest the democratic model may be more suitable for ULSS while, in our case, it seems to be that approximately 10,000 features is not yet large enough for efficient decentralized management.

7 Discussion

This section discusses the results of the study in the context of what the collaborating industry participants found useful with FSC+, decision archetypes and atomic decision visualization. We focus on discussing which findings are pertinent to the studied context and which can be transferred to other contexts or further developed to support decision making for other companies in other domains.

There are several general findings about FSC+ resulting from its application at the company and subsequent interviews, see Section 5. The central outcome, that scope change visualization is highly desired by industry and supportive of process analysis and improvement planning, suggests that FSC+ should be valuable for other companies in other domains. Requirements management processes for large companies are often overloaded [8], by different types of requirements [6] and not designed or prepared to scale up with the complexity explosion demands [72]. The high change frequency significantly contributes to the process overload, especially for agile-inspired processes. Previous work introduced requirements state models to manage large volumes of requirements in MDRE [8]. Our work helps to visualize both the current states and the states history and therefore further support efficient requirements management. We believe that FSC+ can be particularly supportive in managing requirements for MDRE and also for large bespoke project where changes play a significant role.

The study results suggest that the full potential of the FSC+, decision patterns and atomic decision visualization, is reached for large contexts with many frequent decisions while for smaller projects the visualizations may bring limited benefits. This limited usefulness for smaller projects is associated with two aspects: 1) for a limited number of features, requirements analysts or product managers usually can recall the scope changes from their memory or notes, 2) the visual metaphor employed (stacked and sorted horizontal bar charts [83]) facilitates pattern discovery for large datasets (in our case the decision agility and the balance between fast, slow and evolving decisions). In our case, FSC + reaches its full potential for pattern discovery over 1,000 features. With smaller datasets the patterns are also visible, however they may not be significant or sharp. An example can be seen in Fig. 5 where a magnified view of mark B in Fig. 4 is depicted. Approximately 10 features are depicted and the decisions are represented as thick lines. In the zoomed out view, we see these decisions as very thin lines that form a curve when following the state change

coordinates. This observation also provides positive evidence towards the scalability of FSC+.

Based on the study results, we believe that the ability of FSC+ to provide an overview of the "scope history" and the "health of the scope" is agnostic to the used development process, the scoping process and the application domain. It is important to bear in mind that FSC+ does not define what is a "healthy scope" but rather provides analytical support to analyze, assess and define actions to reach and maintain a healthy balance of accepted and rejected features. At the same time, it appears logical that visualizing congestion, process overloads [7] or bottlenecks [58] is more straightforward with the help of the decision pattern visualizations than FSC+. FSC+ can, however, provide indications for process congestion while decision pattern visualizations provide more details regarding which state of the process is a bottleneck.

The concept of representing features as horizontal lines and scope changes as changes in line color seems to be intuitive, easy to grasp by the practitioners, and facilitates quick overview of the scoping history. Based on the study results it can be assumed that pan and zoom are important for scalability of any method or technique that visualizes project dynamics or decision making. Using pan and zoom, recognized in other domains as increasing visual scalability [83], for this case provides improved effectiveness and interactivity. In our case, the zoom factors are greater than 5:1 [83] since the full zoom out offers the possibility to discover hidden patterns in decision making [13].

It appears logical that the *X*-axis should represent time while the *Y*-axis should be configurable, representing effort, complexity or other attributes. In this way, assuming that the lead-time data is reliable, the set of sorted features will form a green area that visually approximates the effort spent on features before they are implemented or withdrawn. This is particularly important for withdrawn features that were not implemented, regardless of the used methodology or process. While the meaning of the *X*-axis should not be changed, flexibility in the amount of time visualized appears to be very important, both for agile-inspired and traditional processes since both the smallest (last sprint) and the largest (entire project) elements can be illustrated. This flexibility also supports creating several visualizations to compare different time spans, different project phases or requirements engineering versus development processes efficiencies.

The study findings suggest that filtering (considered as a way to improve visual scalability [83]) is one of the main features desired by industry. In addition to the above mentioned flexibility in *X*-axis filtering, filtering based on the feature characteristics or its final state is also highly appreciated. By looking at three separate visualizations for: (1) executed features, (2) withdrawn features and (3) features in the process, a company can identify potential behavior discrepancies. Visualizing only features stuck in the process for a long time also helps to identify process bottlenecks [58] that can be expected regardless of the used process or operational domain.

The results regarding feature decision patterns analysis and visualization, see Section 6, lead to the hypothesis that analyzing decision patterns by similarity identification provides valuable process improvement suggestions, enabling learning and possible predictability. These benefits seem to apply also for other processes and contexts as long as they have a defined decision making process that systematically records actions.

We believe that the presented five decision archetypes can be expected in other contexts as well. It appears logical that every project or product development effort has: 1) a number of features that are relatively easy to implement and important (fast executions), e.g. *bug fixes* that take on average up to 20 hours to fix [102], 2) a number of core features that are relatively complex but important and therefore pushed into the implementation (slow executions), 3) a number of highly irrelevant or overly expensive features that can be easily discarded, e.g. during requirements triage [103], 4) a number of relevant but either overly expensive or complex features that may be implemented if more important features are done or left for future releases (evolving). These long leadtime features, if not left to evolve, are often discarded late and become late rejections [10]. For fast executions, this leadtime may not always be fixed but a clear point distinguishing fast decisions from other decisions should be visible relative to the process characteristics, products' expected lifetime or the frequency of the innovation stream. The same logic applies for slow executions as we believe there is a tipping point between fast and slow executions, e.g. correlated with the task complexity [105] or with risky choice behavior [106].

The number of decision cycles depends on the selected methodology and we expect that companies that use agile-inspired methodologies should have many more decision cycles for both fast and slow executions. Agile requirements engineering is characterized by emerging rather than defined requirements that are based on incomplete customer and technology knowledge and thus highly volatile. Moreover, requirements engineering in agile projects is continuous since high-level descriptions are discussed and redefined in each iteration. This, in turn, leads to frequent priority changes as new details and/or challenges are discovered. Also, being responsive to unanticipated changes results in constant re-planning including adding and dropping features [107]. Adding a new high-important feature usually moves other features down in the priority list which, in turn, results in one fast decision (executed or withdrawn later) and one slow decision (executed or withdrawn later). The features that are constantly pushed down the priority list are often significantly delayed and become *evolving*. FSC+ offers the ability to investigate pattern clustering or specific patterns, e.g. those that include backward transitions, helping to uncover process misunderstandings as suggested by our interviewees, see Section 6.

Many managers delay deciding about difficult cases and keep them under consideration for a longer time [10], [97] than cases that can be easily executed, see also mark D in Fig. 4. Therefore, analyzing fast and slow rejections can provide valuable process insights and can help to quantify the seriousness of the delays. Finally, we believe that any company has a number of *evolving* features that are neither implemented nor rejected and identifying and analyzing these features is beneficial as it may lead to obsolete feature identification [99] or process bottleneck identification [58]. However, determining whether the percentage of evolving

features is equally high for other companies remains to be investigated in future work.

Using flow graphs with visually weighted edges appears to promote ready comprehension of the magnitude of an atomic decision in decision patterns and reveal overloaded states that cause bottlenecks. This simple, yet powerful, technique could be used for other contexts and processes as long as there is sufficient data to generate flow graphs. Moreover, tool vendors that consider adding these visualizations should consider introducing meaning to the arrows' length, e.g. the average transition time between two atomic states. The frequency of the decisions embodied in the arrows' size enables straightforward identification of the most frequent atomic transitions.

We believe that the three phenomena identified with the help of atomic decision visualization, namely: 1) bypassing some states, e.g. due to process complexity [8], 2) congestion represented by significant differences between inflow and outflow transitions from a state, and 3) the imbalance between the total number of forward or backward transitions could be expected in other contexts or companies, whether following agile or "traditional" processes. There could be an exception for companies that need to follow certified processes, e.g. companies working in the safety domain, building safety-critical systems. Identification of overloaded states and bottlenecks is the first step toward process improvement and can provide significant cost reduction and improved process efficiency. In our case, the *definition ongoing* state is a clear process bottleneck that delays feature progress. The imbalance between forward and backward transitions is an early sign of overloaded states or congestion and proactive actions can be taken to mitigate their efforts. Finally, based on analyzing atomic decision visualizations the process improvement team may decide to introduce collective decision making for some stages and single person decision making for other stages. In this way, unnecessary discussions could be reduced and overall decision making efficiency should increase.

Based on the study results, we believe that the presented techniques are scalable beyond the dataset analyzed in this paper. FSC+ adaptation is possible for other contexts that record feature states and use a state-machine inspired process for managing feature progress in analysis and development. Managing requirements by recording their states is common and particularly important for MDRE [8] and facilitates requirements change management [108]. The current data parser can be extended or changed with moderate effort to support other data sources. The visualizations are implemented in Java and therefore platform independent. The requirements states names, and filtering and sorting rules are configurable. Therefore, we believe that the solution is suitable for a broad spectrum of software-intensive companies.

The visual notation used for both FSC+ and atomic decision visualizations seems to be scalable for much larger datasets, e.g. having 20,000 features. Moreover, the ability of FSC+ to represent multiple views and layers of complexity depending on the context provides a form of visual notation scaling [14] without over-complicating the technical solution. Still, further empirical studies are required to prove or disprove assumptions inherent in this work.

8 Conclusions

Problems and challenges associated with scoping are threats to successful software project and products [1], [2], [3], [4]. Therefore, companies operating in rapidly-changing business contexts and utilizing processes that embrace frequent scope changes should not only master time-efficient and revenue-optimal scope identification methods [25], [63], [64], but also more carefully monitor the scoping processes on project, product and portfolio levels [15], [16]. The scoping challenge is not only quite unpredictable in rapidly changing context, e.g. MDRE [8], but also quite risky as the consequences of errors or issues can be serious and can even lead to project failures [1], [2], [3], [4].

In this paper, we present Feature Survival Charts+ for visualizing scope evolution which we developed in close collaboration with industry and successfully applied in a context with thousands of features. FSC+ provides useful overviews of the scope evolution history and the "health" of the development efforts currently in scope. We report the solution to be scalable and capable of both supporting the general view of the decision making process and the detailed view of desired products, time-periods or process parts. We successfully analyzed decision patterns based on the scoping history of 8,728 features and propose a simple, yet powerful technique for visualizing atomic decisions. Finally, we define five decision archetypes based on the analyzed dataset: fast execution, slow execution, fast rejection, slow rejection and evolving.

In general, FSC+ was positively received, but the practitioners indicated that considerable flexibility was necessary to meet the widely varying needs of the stakeholders, particularly in the area of filtering the datasets (research question **RQ1**). FSC+ confirmed its potentially valuable role in supporting congestion visualization [7], exploring process bottlenecks [58], process overloads [8] and waste reduction analysis [98]. The flexibility in the temporal scale visualized on *X*-axis (research question **RQ1a**) and the ability to express effort to scale the *Y*-axis (research question **RQ1b**) were highly appreciated by our respondents.

Flexible filtering (research question **RQ1d**) was highly appreciated by our respondents as it facilitates detailed analysis of, e.g. only executed features, features 'stuck in the process' (which could be candidates for obsolete features [99] or significant waste [29]) and withdrawn features. Furthermore, thanks to flexible filtering and proven scalability, FSC+ can support scoping at the product line level [15], [63], portfolio level and project level [3]. Finally, sorting (research question **RQ1c**) by lead-time in a specific state shows promise for process stage optimization, e.g. handshaking with implementation proposals [101] or prioritization [22].

The analysis of the number and nature of decision patterns (research question **RQ2**) brought valuable insights into which process areas require process adjustments or improvements and how many features are potential waste. Decision pattern analysis also helped to quantify the degree of process adherence. The definition and analysis of decision archetypes (research question **RQ2a**) confirmed its utility in facilitating learning and decision making process improvement and demonstrated its potential to uncover

waste contributing features. Atomic decision patterns throughout the dataset were summarized using flow graphs with visually weighted edges representing frequency of occurrence within the dataset (research question **RQ2b**). The industry practitioners found that this visualization allowed them to quickly identify unexpected behavior in the development process and spot potential process bottlenecks [58] .

The reviewing practitioners found utility in all aspects of the present work; senior management found the work slightly more useful than those charged with day-to-day execution. The new visualization techniques greatly facilitate comprehension of process operations compared to direct inspection of the raw data and practitioners felt that the visualizations could be used to guide management decisions such as process optimization.

8.1 Future Work

Future work focuses mainly on replicating the study at other companies, preferably in other domains. As the results in the paper are based on a large dataset from one company, further research effort is needed to explore the transferability of the proposed technique to other companies and contexts. Conducting further case studies would also increase the reliability of the presented results, especially for interview results. More empirical data would also help to identify potential improvements or changes in the data collection methods used.

Future work should also focus on further improvements to the FSC+ technique and further improvements to the performance of the visualization tool. We plan to investigate the possible entrenchments of FSC+ with aggregated level views, e.g. using treemaps. Using treemaps appears to be promising for creating aggregated views of the decision patterns within decision archetypes as they are to some degree hierarchical.

Further empirical studies to confirm or deny the industrial applicability of the technique are planned. Finally, further investigations of feature decision archetypes will attempt to determine whether the identified archetypes are general, rather than specific to the case company, and whether there may be any omissions from the set of archetypes.

Acknowledgments

This work was party funded by Vinnova in the ITEA2 project 12018 SCALARE and by a research grant for the IKNOWDM project (reference number 20150033) from the Knowledge Foundation in Sweden.

Appendix A: Interview Questions

Usefulness of Visualizing the Scope and FSC+:

1) How useful according to You is FSC+ technique for scope visualization?
2) Which tasks can FSC+ support?
3) How often would you use scope visualization in your daily work?

Show an FSC for Their Context and Discuss the Possible Extensions in *Y*-Axis

1) What do you think about the moving map solution?
2) What do you think about the zooming in solution?
3) How would you like to be represented in the *Y*-axis? (priority, criticality, size, impact etc.)

Filtering and Sorting

1) What type of information should be filtered by FSC+?
2) How would you like the chart to be sorted?
3) What is the most optimal time-spam to visualize?

Decision Patterns

1) Do you think that analyzing decision patterns would be useful for your work?
2) Do you think that visualizing decision patterns would be beneficial in your work?

Analyzing Decision Pattern

1) How would you interpret the number of unique patterns in the dataset?
2) How would you interpret the number of cycles in the patterns?
3) Why do so many paths terminate at illogical states?

References

[1] C. Iacovou and A. Dexter, "Turning around runaway information technology projects," *IEEE Eng. Manage. Rev.*, vol. 32, no. 4, pp. 97–112, Dec. 2004.

[2] E. Bjarnason, K. Wnuk, and B. Regnell, "Are you biting off more than you can chew? A case study on causes and effects of overscoping in large-scale software engineering," *Inf. Softw. Technol.*, vol. 54, no. 10, pp. 1107–1124, 2012.

[3] T. DeMarco and T. Lister, "Risk management during requirements," *IEEE Softw.*, vol. 20, no. 5, pp. 99–101, Sep./Oct. 2003.

[4] R. A. Carter, A. Antn, A. Dagnino, and L. Williams, "Evolving beyond requirements creep: A risk-based evolutionary prototyping model," in *Proc. 5th IEEE Int. Symp. Requirements Eng.*, 2001, pp. 94–101.

[5] T. Hall, S. Beecham, and A. Rainer, "Requirements problems in twelve software companies: An empirical analysis," *IEEE Softw.*, vol. 149, no. 5, pp. 153–160, Sep./Oct. 2002.

[6] T. Gorschek and C. Wohlin, "Requirements abstraction model," *Requirements Eng. J.*, vol. 11, pp. 79–101, 2006.

[7] L. Karlsson, S. Dahlstedt, J. Natt Och Dag, B. Regnell, and A. Persson, "Challenges in market-driven requirements engineering-an industrial interview study," in *Proc. 8th Int. Workshop Requirements Eng.: Found. Softw. Quality*, 2002. Q2

[8] B. Regnell and S. Brinkkemper, "Market–driven requirements engineering for software products," in *Eng. and Managing Software Requirements*, A. Aurum and C. Wohlin, Eds. New York, NY, USA: Springer, 2005, pp. 287–308.

[9] B. Regnell, R. B. Svensson, and K. Wnuk, "Can we beat the complexity of very large-scale requirements engineering?" in *Proc. 14th Int. Conf. Requirements Eng.: Found. Softw. Quality*, 2008, vol. 5025, pp. 123–128.

[10] K. Wnuk, B. Regnell, and L. Karlsson, "What happened to our features? Visualization and understanding of scope change dynamics in a large-scale industrial setting," in *Proc. 17th IEEE Int. Requirements Eng. Conf.*, 2009, pp. 89–98.

[11] G. Kulk and C. Verhoef, "Quantifying requirements volatility effects," *Sci. Comput. Program.*, vol. 72, no. 3, pp. 136–175, Aug. 2008.

[12] D. Keim, G. Andrienko, J.-D. Fekete, C. Grg, J. Kohlhammer, and G. Melanon, "Visual analytics: Definition, process, and challenges," in *Information Visualization*, A. Kerren, J. Stasko, J.-D. Fekete, and C. North, Eds. Heidelberg, Germany, Springer, 2008, vol. 4950, pp. 154–175.

[13] S. Reddivari, S. Rad, T. Bhowmik, N. Cain, and N. Niu, "Visual requirements analytics: A framework and case study," *Requirements Eng.*, vol. 19, no. 3, pp. 257–279, 2014.

[14] M. Laitinen, M. E. Fayad, and R. P. Ward, "Thinking objectively: The problem with scalability," *Commun. ACM*, vol. 43, no. 9, pp. 105–107, Sep. 2000.

[15] K. Pohl, G. Bockle, and F. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. New York, NY, USA: Springer-Verlag, 2005.

[16] R. Cooper, *Winning at New Products: Creating Value Through Innovation*. New York, NY, USA: Basic Books, 2011.

[17] K. Wnuk, B. Regnell, and L. Karlsson, "Visualization of feature survival in platform-based embedded systems development for improved understanding of scope dynamics," in *Proc. 3rd Int. Workshop Requirements Eng. Vis.*, 2008, pp. 41–50.

[18] A. Al-Emran, D. Pfahl, and G. Ruhe, "Decision support for product release planning based on robustness analysis," in *Proc. 18th IEEE Int. Requirements Eng. Conf.*, 2010, pp. 157–166.

[19] M. Khurum, T. Gorschek, and M. Wilson, "The software value map: An exhaustive collection of value aspects for the development of software intensive products," *J. Softw.: Evolution Process*, vol. 25, no. 7, pp. 711–741, 2013.

[20] A. van der Hoek, R. Hall, D. Heimbigner, and A. Wolf, "Software release management," in *Proc. 6th Eur. Softw. Eng. Conf.*, 1997, pp. 159–175.

[21] G. Ruhe, *Product Release Planning: Methods, Tools and Applications*. New York, NY, USA: Auerbach, 2009.

[22] J. Karlsson and K. Ryan, "A cost-value approach for prioritizing requirements," *IEEE Softw.*, vol. 14, no. 5, pp. 67–74, Sep. 1997.

[23] M. Svahnberg, T. Gorschek, R. Feldt, R. Torkar, S. Saleem, and M. Shafique, "A systematic review on strategic release planning models," *Inf. Softw. Technol.*, vol. 52, no. 3, pp. 237–248, Mar. 2010.

[24] A. Aurum and C. Wohlin, "The fundamental nature of requirements engineering activities as a decision-making process," *Inf. Softw. Technol.*, vol. 45, no. 14, pp. 945–954, 2003.

[25] G. Ruhe and M. Saliu, "The art and science of software release planning," *IEEE Softw.*, vol. 22, no. 6, pp. 47–53, Nov./Dec. 2005.

[26] V. Heikkila, K. Rautiainen, and S. Jansen, "A revelatory case study on scaling agile release planning," Lille, France, 2010, pp. 289–296.

[27] M. Li, M. Huang, F. Shu, and J. Li, "A risk-driven method for extreme programming release planning," in *Proc. 28th Int. Conf. Softw. Eng.*, New York, NY, USA, 2006, pp. 423–430.

[28] A. Ngo-The and G. Ruhe, "Decision support in requirements engineering," in *Eng. and Managing Softw. Requirements*. New York, NY, USA: Springer, 2005, pp. 267–286.

[29] K. Wnuk, D. Callele, E.-A. Karlsson, and B. Regnell, "Controlling lost opportunity costs in agile development: The basic lost opportunity estimation model for requirements scoping," in *Proc. 3rd Int. Conf. Softw. Bus.*, 2012, vol. 114, pp. 255–260.

[30] K. Petersen and C. Wohlin, "Software process improvement through the lean measurement (spi-leam) method," *J. Syst. Softw.*, vol. 83, no. 7, pp. 1275–1287, Jul. 2010.

[31] A. Ngo-The and G. Ruhe, "A systematic approach for solving the wicked problem of software release planning," *Softw. Comput.*, vol. 12, no. 1, pp. 95–108, Aug. 2007.

[32] P. Carlshamre, "Release planning in market-driven software product development: Provoking an understanding," *Requirements Eng.*, vol. 7, pp. 139–151, 2002.

[33] A. Ngo-The, G. Ruhe, and W. Shen, "Release planning under fuzzy effort constraints," in *Proc. 3rd IEEE Int. Conf. Cognitive Informatics*, 2004, pp. 168–175.

[34] G. Ruhe and A. Ngo, "Hybrid intelligence in software release planning," *Int. J. Hybrid Intell. Syst.*, vol. 1, no. 1-2, pp. 99–110, Apr. 2004.

[35] A. Ngo-The and M. Saliu, "Fuzzy structural dependency constraints in software release planning," in *Proc. 14th IEEE Int. Conf. Fuzzy Syst.*, 2005, pp. 442–447.

[36] K. Logue and K. McDaid, "Agile release planning: Dealing with uncertainty in development time and business value," in *Proc. 15th Annu. IEEE Int. Conf. Workshop Eng. Comput. Based Syst.*, Piscataway, NJ, USA, 2008, pp. 437–42.

[37] M. van den Akker, S. Brinkkemper, G. Diepen, and J. Versendaal, "Software product release planning through optimization and what-if analysis," *Inf. Softw. Technol.*, vol. 50, no. 1-2, pp. 101–111, Jan. 2008.

[38] Amandeep, G. Ruhe, and M. Stanford, "Intelligent support for software release planning," in *Proc. Product Focused Softw. Process Improvement*, 2004, vol. 3009, pp. 248–262.

[39] D. Pfahl, A. Al-Emran, and G. Ruhe, "A system dynamics simulation model for analyzing the stability of software release plans: Research sections," *Softw. Process*, vol. 12, no. 5, pp. 475–490, Sep. 2007.

[40] P. Kaur, "Reinforcement learning based approach for adaptive release planning in an agile environment," in *Proc. Int. Conf. Comput. Intell. Softw. Eng.* Piscataway, NJ, USA, 2010, pp. 4–8.

[41] A. Szoke, "A feature partitioning method for distributed agile release planning," vol. 77, LNBIP, 2011, pp. 27–42.

[42] D. Greer and G. Ruhe, "Software release planning: An evolutionary and iterative approach," *Inf. Softw. Technol.*, vol. 46, no. 4, pp. 243–253, 2004.

[43] B. Regnell, R. Svensson, and T. Olsson, "Supporting roadmapping of quality requirements," *IEEE Softw.*, vol. 25, no. 2, pp. 42–47, Mar./Apr. 2008.

[44] B. Regnell, L. Karlsson, and M. Höst, "An analytical model for requirements selection quality evaluation in product software development," in *Proc. 11th IEEE Int. Conf. Requirements Eng.*, Washington, DC, USA, 2003, pp. 254–263.

[45] L. Karlsson and B. Regnell, "Introducing tool support for retrospective analysis of release planning decisions," in *Proc. 7th Int. Conf. Product-Focused Softw. Process Improvement*, 2006, pp. 19–33.

[46] S. Ziemer and I. C. Calori, "An experiment with a release planning method for web application development," in *Proc. 14th Eur. Conf. Softw. Process Improvement*, 2007, pp. 106–117.

[47] J. van Den Akker, S. Brinkkemper, G. Diepen, and J. Versendaal, "Determination of the next release of a software product: An approach using integer linear programming," in *Proc. 11th Int. Workshop Requirements Eng.: Found. Softw. Quality*, 2005, pp. 247–262.

[48] O. Saliu and G. Ruhe, "Supporting software release planning decisions for evolving systems," in *Proc. 29th Annu. IEEE/NASA Softw. Eng. Workshop*, 2005, pp. 14–26.

[49] G. Du, J. McElroy, and G. Ruhe, "Ad hoc versus systematic planning of software releases – A three-staged experiment," in *Proc. 7th Int. Conf. Product-Focused Softw. Process Improvement*, 2006, pp. 435–440.

[50] M. O. Saliu and G. Ruhe, "Bi-objective release planning for evolving software systems," in *Proc. 6th Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng.*, New York, NY, USA, 2007, pp. 105–114.

[51] A. Ngo-The and M. Saliu, "Measuring dependency constraint satisfaction in software release planning using dissimilarity of fuzzy graphs," in *Proc. 4th IEEE Conf. Cognitive Informatics*, 2005, pp. 301–307.

[52] S. Maurice, G. Ruhe, O. Saliu, and A. Ngo-The, "Decision support for value-based software release planning," in *Value-Based Softw. Eng.*, S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, and P. Grnbacher, Eds. Berlin, Germany, Springer, 2006, pp. 247–261.

[53] G. Ruhe and D. Greer, "Quantitative studies in software release planning under risk and resource constraints," in *Proc. Int. Symp. Empirical Softw. Eng.*, 2003, pp. 262–270.

[54] B. Regnell and K. Kuchcinski, "Exploring software product management decision problems with constraint solving-opportunities for prioritization and release planning," in *Proc. 5th Int. Workshop Softw. Product Manage.*, Aug. 2011, pp. 47–56.

[55] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell, and J. Natt och Dag, "An industrial survey of requirements interdependencies in software product release planning," in *Proc. 5th IEEE Int. Symp. Requirements Eng.*, 2001, pp. 84–91.

[56] L. Lehtola and M. Kauppinen, "Suitability of requirements prioritization methods for market-driven software product development," *Softw. Process: Improvement Practice*, vol. 11, no. 1, pp. 7–19, 2006.

[57] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change*, 2nd ed. Reading, MA, USA: Addison-Wesley, 2004.

[58] M. Höst, B. Regnell, J. Natt och Dag, J. Nedstam, and C. Nyberg, "Exploring bottlenecks in market-driven requirements management," *J. Syst. Softw.*, vol. 59, no. 3, pp. 323–332, 2001.

[59] V. Basili and H. Rombach, "The tame project: Towards Improvement-oriented software environments," *IEEE Trans. Softw. Eng.*, vol. 14, no. 6, pp. 758–773, Jun. 1988.

[60] G. DeGregorio, "Visual tool support for configuring and understanding software product lines," in *Proc. 9th Int. Symp. Int. Council Syst. Eng.*, 1999, pp. 1–7.

[61] P. M. Institute, "Project scope management," in *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*. Newtown Square, PA, USA: Project Management Institute, 2000, ch. 5, Q4 pp. 47–59.

[62] J. Withey, "Investment analysis of software assets for product Q5 lines," Tech. Rep., 1996.

[63] K. Schmid, "A comprehensive product line scoping approach and its validation," in *Proc. 24th Int. Conf. Softw. Eng.*, 2002, pp. 593–603.

[64] J. Savolainen, M. Kauppinen, and T. Mannisto, "Identifying key requirements for a new product line," in *Proc. 14th Asia-Pacific Softw. Eng. Conf.*, Washington, DC, USA, 2007, pp. 478–485.

Q6 [65] E. Tufte, *Envisioning Information*. Graphics Press LLC, 1990.

[66] O. Gotel, F. Marchese, and S. Morris, "On requirements visualization," in *Proc. 2nd Int. Workshop Requirements Eng. Vis.*, Washington, DC, USA, 2007, pp. 11–20.

[67] C. Oezbek, L. Prechelt, and F. Thiel, "The onion has cancer: Some social network analysis visualizations of open source project communication," in *Proc. 3rd Int. Workshop Emerging Trends Free/Libre/Open Source Softw. Res. Develop.*, New York, NY, USA, 2010, pp. 5–10.

[68] N. Hanakawa, "Visualization for software evolution based on logical coupling and module coupling," in *Proc. 14th Asia-Pacific Softw. Eng. Conf.*, Dec. 2007, pp. 214–221.

[69] M. Austin, V. Mayank, and N. Shmunis, "Paladinrm: Graph-based visualization of requirements organized for team-based design," *Syst. Eng.*, vol. 9, no. 2, pp. 129–145, May 2006.

[70] P. Checkland, *Systems Thinking, Systems Practice*. Hoboken, NJ, USA: Wiley, 1981.

[71] E. Yu, "Towards modelling and reasoning support for early-phase requirements engineering," in *Proc. 3rd IEEE Int. Symp. Requirements Eng.*, Jan. 1997, pp. 226–235.

[72] B. Berenbach, D. Paulish, J. Kazmeier, and A. Rudorfer, *Software & Systems Requirements Engineering: In Practice*. Pearson Education Inc., 2009.

[73] K. Wnuk, B. Regnell, and B. Berenbach, "Scaling up requirements engineering: Exploring the challenges of increasing size and complexity in market-driven software development," in *Proc. 17th Int. Working Conf. Requirements Eng.: Found. Softw. Quality*, 2011, vol. 6606, pp. 54–59.

[74] P. Garg, "On supporting large-scale decentralized software engineering processes," in *Proc. 28th IEEE Conf. Decision Control*, Dec. 1989, vol. 2, pp. 1314–1317.

[75] S. Konrad and M. Gall, "Requirements engineering in the development of large-scale systems," in *Proc. 16th Int. Requirements Eng. Conf.*, 2008, pp. 217–222.

[76] C. Ebert, "Dealing with nonfunctional requirements in large software systems," *Ann. Softw. Eng.*, vol. 3, no. 1, pp. 367–395, 2004.

[77] B. Boehm, "Some future trends and implications for systems and software engineering processes," *Syst. Eng.*, vol. 9, no. 1, pp. 1–19, 2006.

[78] L. Northrop, P. Felier, R. Habriel, J. Boodenough, R. Linger, M. Klein, D. Schmidt, K. Sullivan, and K. Wallnau, *Ultra-Large-Scale Systems: The Software Challenge of the Future*. Software Engineering Institute, 2006.

[79] L. Strigini, "limiting the dangers of intuitive decision making," *IEEE Softw.*, vol. 13, no. 1, pp. 101–103, Jan. 1996.

[80] K. Schwaber and M. Beedle, *Agile Software Development with Scrum* (series in agile software development). Englewood Cliffs, NJ, USA: Prentice-Hall, 2002.

[81] M. Khurum and T. Gorschek, "A method for alignment evaluation of product strategies among stakeholders (mass) in software intensive product development," *J. Softw. Maintenance Evol.*, vol. 23, no. 7, pp. 494–516, Nov. 2011.

[82] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (foda) feasibility study," DTIC Document, Softw. Eng. Instit., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-90-TR-021, 1990.

[83] S. G. Eick and A. F. Karr, "Visual scalability," *J. Comput. Graph. Statist.*, vol. 11, no. 1, pp. 22–43, 2002.

[84] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Softw. Eng. J.*, vol. 14, no. 2, pp. 131–164, 2009.

[85] R. Wieringa and A. Moral, "Technical action research as a validation method in information systems design science," in *Proc. 7th Int. Conf. Des. Sci. Res. Information Syst. Adv. Theory Practice*, 2012, vol. 7286, pp. 220–238.

[86] R. Yin, *Case Study Research: Design and Methods*. Newbury Park, CA, USA: Sage, 2003.

[87] S. Easterbrook, J. Singer, M. Storey, and D. Damian, "Selecting empirical methods for software engineering research," in *Guide to Advanced Empirical Software Engineering*. New York, NY, USA: Springer, 2007, pp. 285–311.

[88] C. Robson, *Real World Research*. Oxford, U.K.: Blackwell, 2002.

[89] R. Wieringa and H. Heerkens, "Designing requirements engineering research," in *Proc. 5th Int. Workshop Comparative Eval. Requirements Eng.*, 2007, pp. 36–48.

[90] R. Wieringa. (2014). Empirical research methods for technology validation: Scaling up to practice. *J. Syst. Softw.* [Online]. *95(0)*, pp. 19–31. Available: http://www.sciencedirect.com/science/article/pii/S0164121213002793

[91] M. Patton, *Qualitative Research & Evaluation Methods*. Newbury Park, CA, USA: Sage, 2002.

[92] J. Siegmund, C. Kstner, J. Liebig, S. Apel, and S. Hanenberg, "Measuring and modeling programming experience," *Empirical Softw. Eng.*, vol. 19, no. 5, pp. 1299–1334, 2014.

[93] B. Flyvbjerg, "Five misunderstandings about case-study research," in *Qualitative Research Practice*. Newbury Park, CA, USA: Sage, 2007, pp. 390–404.

[94] C. Freeman and L. Soete, *The Economics of Industrial Innovation*. Pinter, 1997.

[95] C.-Y. Lee, "A simple theory and evidence on the determinants of firm r&d," *Economics Innovation New Technol.*, vol. 12, no. 5, pp. 385–395, 2003.

[96] N. Smirnov, "On the estimation of the discrepancy between empirical curves of distribution for two independent samples," *Bull. Math. Univ. Moscow*, vol. 2, pp. 3–14, 1939.

[97] J. Kabbedijk, K. Wnuk, B. Regnell, and S. Brinkkemper, "What decision characteristics influence decision making in market-driven large-scale software product line development?" in *Proc. 16th Int. Working Conf. Requirements Eng.: Found. Softw. Quality*, 2010, pp. 42–53.

[98] K. Petersen and C. Wohlin, "Measuring the flow in lean software development," *Softw. Practice Exp.*, vol. 41, no. 9, pp. 975–996, Aug. 2011.

[99] K. Wnuk, T. Gorschek, and S. Zahda, "Obsolete software requirements," *Inf. Softw. Technol.*, vol. 55, no. 6, pp. 921–940, Jun. 2013.

[100] C. Taylor, A. Miranskyy, and N. Madhavji, "Request-implementation ratio as an indicator for requirements prioritisation imbalance," in *Proc. 5th Int. Workshop Softw. Product Manage.*, Aug. 2011, pp. 3–6.

[101] S. Fricker, T. Gorschek, C. Byman, and A. Schmidle, "Handshaking with implementation proposals: Negotiating requirements understanding," *IEEE Softw.*, vol. 27, no. 2, pp. 72–80, Mar./Apr. 2010.

[102] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *Proc. 4th Int. Workshop Mining Softw. Repositories*, Washington, DC, USA, 2007, p. 1.

[103] A. M. Davis, "The art of requirements triage," *IEEE Comput.*, vol. 36, no. 3, pp. 42–49, Mar. 2003.

[104] M. Unterkalmsteiner, T. Gorschek, A. K. M. M. Islam, C. K. Cheng, R. B. Permadi, and R. Feldt, "Evaluation and measurement of software process improvement: A systematic literature review," *IEEE Trans. Softw. Eng.*, vol. 38, no. 2, pp. 398–424, Mar./Apr. 2012.

[105] R. M. Hogarth, "Decision time as a function of task complexity," in *Utility, Probability, and Human Decision Making* (series theory and decision library), D. Wendt and C. Vlek, Eds. Netherlands, Springer, 1975, vol. 11, pp. 321–338.

[106] H. B. Zur and S. J. Breznitz. (1981). The effect of time pressure on risky choice behavior. *Acta Psychologica*. [Online]. *47(2)*, pp. 89–104. Available: http://www.sciencedirect.com/science/article/pii/0001691881900019

[107] B. Ramesh, L. Cao, and R. Baskerville, "Agile requirements engineering practices and challenges: An empirical study," *Inf. Syst. J.*, vol. 20, no. 5, pp. 449–480, 2010.

[108] C. Hood, S. Wiedemann, S. Fichtinger, and U. Pautz, "Change management interface," in *Requirements Management: The Interface between Requirements Development and All Other Softw. Engineering Processes*, C. Hood, S. Wiedemann, S. Fichtinger, and U. Pautz, Eds. New York, NY, USA: Springer, 2008.

Q7 **Krzysztof Wnuk** received the MSc Degree from the Gdansk University of Technology, Poland in 2006 and the PhD degree from Lund University, Sweden in 2012. He is an assistant professor at the Software Engineering Research Group (SERL), Blekinge Institute of Technology, Sweden. His research interests include market-driven software development, requirements engineering, software product management, decision making in requirements engineering, large-scale software, system and requirements engineering and management and empirical research methods. He is interested in software business, open innovation, and open source software. He works as an expert consultant in software engineering for the Swedish software industry.

Tony Gorschek is a professor of software engineering at the Blekinge Institute of Technology (SERL-Sweden). He has more than 10 years industrial experience as a CTO, senior executive consultant and engineer, but also as a chief architect and product manager. In addition he has built up six startups in fields ranging from logistics to internet-based services and algorithmic trading. Currently, he manages his own consultancy company, works as a CTO, and serves on several boards in companies developing cutting edge technology and products. His research interests include requirements engineering, technology and product management, process assessment and improvement, quality assurance, and practical innovation. He is a member of the IEEE.

David Callele is the president and CEO of Experience First Design Inc., a game developer and consultancy based in Saskatoon, Saskatchewan. He is a technology commercialization and a startup specialist, leading or participating in nine startups, and has advised and mentored more than 100 projects in the last four years. A member of the board of directors for the Saskatchewan Capital Network, the Saskatchewan angel investor network, he is also an inventor on nine patents, has more than 40 academic publications and is an adjunct professor in the Department of Computer Science, University of Saskatchewan. He is a member of the IEEE.

Even-André Karlsson received the PhD degree in software engineering from NTNU, Norway. He is a very experienced leader of change and improvement initiatives and currently works as a senior consultant at Addalot Consulting in Malm, Sweden. Even has applied, refined, and developed methodology for successful improvements in many R&D and IT organisations active in different fields with an emphasis on technical systems, e.g. telecommunications. His expertise covers most aspects of system development with an emphasis on software intensive systems, e.g. requirements management, software architecture, development processes, development and modelling tools, quality management, measurements, information modeling, program and project management, agile development, and assessment based on improvement models, e.g. CMMI and ASPICE. He is the author of 30+ journal and conference papers.

Eskil Åhlin has worked as a professional in the mobile telecom industry for more than 15 years. Beginning his career in writing the first standards for 3G and accompanying services, and then moving on to managing the software scope of very large projects in the mobile phone business. He is now the head of business development for Sony Mobile in Silicon Valley.

Björn Regnell is a professor in software engineering at the Faculty of Engineering, LTH, Lund University, Sweden. He has contributed to several software engineering research areas including requirements engineering, software quality, software product management, and empirical research methods in software engineering. He was ranked among the top 13 scholars in the world in experimental software engineering in *IEEE Transactions on Software Engineering*, 31(9):733-753 (2005). He is/was a reviewer for several high-impact journals and peer-reviewed conference program committees and is currently a member of the editorial board of the *Requirements Engineering Journal* (Springer) and the steering committee chair of www.refsq.org. He has published more than 80 peer-reviewed articles in journals and conferences. He has edited several special issues in journals and proceedings and is a co-author of several books including the widely cited *Introduction to Experimentation in Software Engineering* (Springer, 2000) and *Case Study Research in Software Engineering-Guidelines and Examples* (Wiley, 2012). He worked part time as a senior researcher at Sony Ericsson, CTO Office, Lund, Sweden from 2005 to 2007, and he works as an expert consultant in software engineering for the Swedish software industry.

Queries to the Author

Q1. There is discrepancy in Ref. citation in the text between the PDf and the source file. We have followed the source file. Please check.
Q2. Please provide page range for Ref. [7].
Q3. Please provide full bibliographic details for Refs. [26] and [41].
Q4. Please check whether Refs. [61] and [82] are ok as set.
Q5. Please provide the department name and location also provide the report number for Ref. [62].
Q6. Please provide publishers location for Refs. [65], [72], [78], and [94].
Q7. Please provide authors photos.