

Information needs and presentation in agile software development

Henri Bomström^{a,*}, Markus Kelanti^a, Elina Annanperä^a, Kari Liukkunen^a, Terhi Kilamo^b,
Outi Sievi-Korte^b, Kari Systä^b

^a M3S Research Unit, University of Oulu, Pentti Kaiteran katu 1, Linnanmaa, Oulu, 90014, Finland

^b Faculty of Information Technology and Communication Sciences, Tampere University, Kalevantie 4, Tampere, 33014, Finland

ARTICLE INFO

Keywords:

Software engineering
Agile software development
DevOps
Information needs
Visualization

ABSTRACT

Context: Agile software companies applying the DevOps approach require collaboration and information sharing between practitioners in various roles to produce value. Adopting new development practices affects how practitioners collaborate, requiring companies to form a closer connection between business strategy and software development. However, the types of information management, sales, and development needed to plan, evaluate features, and reconcile their expectations with each other need to be clarified.

Objective: To support practitioners in collaborating and realizing changes to their practices, we investigated what information is needed and how it should be represented to support different stakeholders in their tasks. Compared to earlier research, we adopted a holistic approach – by including practitioners throughout the development process – to better understand the information needs from a broader viewpoint.

Method: We conducted six workshops and 12 semi-structured interviews at three Finnish small and medium-sized enterprises from different software domains. Thematic analysis was used to identify information-related issues and information and visualization needs for daily tasks. Three themes were constructed as the result of our analysis.

Results: Visual information representation catalyzes stakeholder discussion, and supporting information exchange between stakeholder groups is vital for efficient collaboration in software product development. Additionally, user-centric data collection practices are needed to understand how software products are used and to support practitioners' daily information needs. We also found that a passive way of representing information, such as a dashboard that would disturb practitioners only when attention is needed, was preferred for daily information needs.

Conclusion: The software engineering community should consider reviewing the information needs of practitioners from a more holistic view to better understand how tooling support can benefit information exchange between stakeholder groups when making product development decisions and how those tools should be built to accommodate different stakeholder views.

1. Introduction

Software companies are redefining how they conduct business by accelerating their development processes [1] and shifting toward practices that enable rapid and continuous software delivery. Transitioning from traditional agile software development toward continuous practices, such as DevOps [2], affects how software teams collaborate [3]. Such transformations can enhance communication and collaboration by breaking down the barriers between organizational departments [4], which requires a closer connection between business strategy and software development [5,6]. Recent research has shown that software pipelines can be considered to extend beyond individual departments

to client organizations [7], requiring further inquiries into the expanding perspectives of software development. However, the type of information executives need for the planning and evaluation of features and for determining how sales and marketing should reconcile their expectations with development are unclear [5].

Information representation and visualization techniques provide value for agile software development teams by facilitating knowledge sharing and awareness regarding the development process [8], as coordination between stakeholders affects both productivity and software failure [9]. Burn-down charts and task boards are commonly used to maintain awareness and track progress in agile software development [8]. Although contemporary software development approaches

* Corresponding author.

E-mail addresses: henri.bomstrom@oulu.fi (H. Bomström), markus.kelanti@oulu.fi (M. Kelanti), elina.annanpera@oulu.fi (E. Annanperä), kari.liukkunen@oulu.fi (K. Liukkunen), terhi.kilamo@tuni.fi (T. Kilamo), outi.sievi-korte@tuni.fi (O. Sievi-Korte), kari.systa@tuni.fi (K. Systä).

<https://doi.org/10.1016/j.infof.2023.107265>

Received 11 November 2022; Received in revised form 17 May 2023; Accepted 23 May 2023

Available online 27 May 2023

0950-5849/© 2023 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

concern various tasks and stakeholders, software visualization research predominantly targets developers with source code information [10] and primarily concerns software structure, behavior, and evolution [10, 11]. An example of software evolution is how the system changes in terms of bugs discovered and how new requirements modify the system [11]. Despite modern software processes producing increasing amounts of data, communicating their meaning to relevant stakeholders in software companies – including those actively participating in developing or procuring software – remains challenging. To support the software engineering community, we adopted a more holistic approach, targeting practitioners throughout the development process, including practitioners not directly related to producing software.

This article investigates what information is needed to support different software practitioners in their tasks and how that information should be represented. We conducted our study in three Finnish small and medium-sized enterprises (SMEs) in the software domain, focusing on the following research question: *What kind of information is needed to support different practitioners in software development?* We organized three workshops (one per participating company) to explore the company use cases, interviewed 12 practitioners from the three companies, and organized three additional workshops to validate our results based on the companies' feedback. Moreover, we asked practitioners about their roles and tasks and their information and visualization needs regarding their daily tasks.

Based on the workshop and interview results, we found that visual information representation catalyzes discussion between stakeholder groups. However, support for facilitating the information exchange between stakeholder groups in software product development is needed. An interesting observation is that practitioners preferred dashboards with a minimal representation of information to address their daily information needs. Furthermore, we found that a mechanism is needed to understand software product usage in customer organizations. These results have implications for the types of views practitioners need to support their tasks, what information those views should convey, how the information should be represented, and what this means for data collection.

The remainder of this article is structured as follows. First, Section 2 introduces the background and related work on the information and visualization needs for agile software development. Section 3 introduces our research question and describes the methodology we utilized to answer it. Section 4 presents the results from the conducted workshops and semi-structured interviews. Section 5 discusses the findings we consider most relevant to the software engineering community and the detailed results with regard to related works. In addition, Section 5 presents the implications of our findings, along with the limitations and validity threats of this study and directions for future research. Finally, Section 6 presents our conclusions.

2. Background and related work

Our approach presumes that software development information and information representation needs exist beyond the practitioners directly involved with the software or source code. For this reason, we directed our focus to tailoring information and visualizations to different needs. Despite approaching these topics from a software engineering viewpoint, we explored these issues through the lens of agile software development, as the participating companies in this study employed agile for product management and actual development.

Regarding terminology, stakeholders are entities such as employees, financiers, customers, and communities [12]. We use the term “stakeholder” when discussing entities more broadly, including those outside individual companies, and the term “practitioner” when explicitly discussing company employees in general, including managers. Furthermore, we consider graphical visualizations as just one way, albeit the most common one, of representing information. Information is an abstract concept that can be difficult to define precisely. In

the context of this article, we approach information from a software visualization viewpoint. From that perspective, information can be considered data [13] or data sources that are utilized by tools [14, 15], making it a target of representation. Information representation in software visualization refers to how information is presented to different stakeholders and consists of aspects such as form, type of view, and techniques [14]. A concrete example of information and its representation is a failed test suite, which provides information that the test failed and that a bug likely needs to be addressed, represented by a bright red circle. The source of this information, or data, would be a software testing tool or framework. Lastly, documents and software components are examples of information artifacts.

2.1. Roles in agile software development

The Agile manifesto promotes self-organizing teams and adjusting development practices periodically over time [16]. The definitions of roles in agile development have been based on what kind of development is done [17,18], such as feature-driven development [19], eXtreme programming [20], or Scrum [21], or through scaling frameworks for larger settings [22], such as the Scaled agile framework (SAFe) [23] and Large-Scale Scrum (LeSS) [24], which have more well-defined practices and role descriptions. For reference, Scrum – one of the most popular development methods [25] – features some pre-described roles, such as product owner, scrum master, and team member. Smaller organizations tend to mix responsibilities with fewer roles overall compared to larger organizations with a wider pool of different roles [17]; however, regardless of the selected approach, agile teams tailor their roles to suit their own needs [26]. The distinctions between team and leadership roles can often be blurred [27], and practitioners may play more informal, self-organized roles along with their organizational roles [28].

The concept of roles is still quite abstract within traditional software engineering, where roles are applied in a vague manner to solve specific tasks or aspects of engineering [29,30]. In this paper, when we refer to software development, we consider agile development practices to be the standard, unless otherwise specified. In addition, due to the informality of roles in agile development, we focus on practitioners' tasks rather than on static descriptions of roles.

2.2. Information needs in agile software development

At its core, software development is a knowledge-intensive process that requires collaboration to produce software artifacts [31]. The amount of information available and used by software developers in their daily activities has been steadily increasing during the last two decades, along with the research interest in mapping developer information needs [32]. Despite the increased research interest, the information needs of various practitioners participating in the development of software are not yet fully understood.

Agile teams have a strong focus on face-to-face communication [33] and a strong reliance on tacit knowledge [34], which are exchanged through conversation. Studies have shown that developers communicate both know-what and know-how information face-to-face, often in sessions, such as code reviews [35,36]. Information related to team problems is central to software project success, and these problems can be avoided with proper feedback [37]; however, practitioners spend extra time communicating with each other due to the missing dependencies of information artifacts [38] and being aware of coworkers' tasks [39,40]. Despite allowing for a more direct exchange of information, face-to-face collaboration may delay answers to questions related to needed topics, such as software behavior, design questions, and bug reproduction steps, as they require communication with others [39].

The information needs of developers have garnered increased research interest over the years compared to other roles. Studies have shown the importance of product architecture and design, requirement

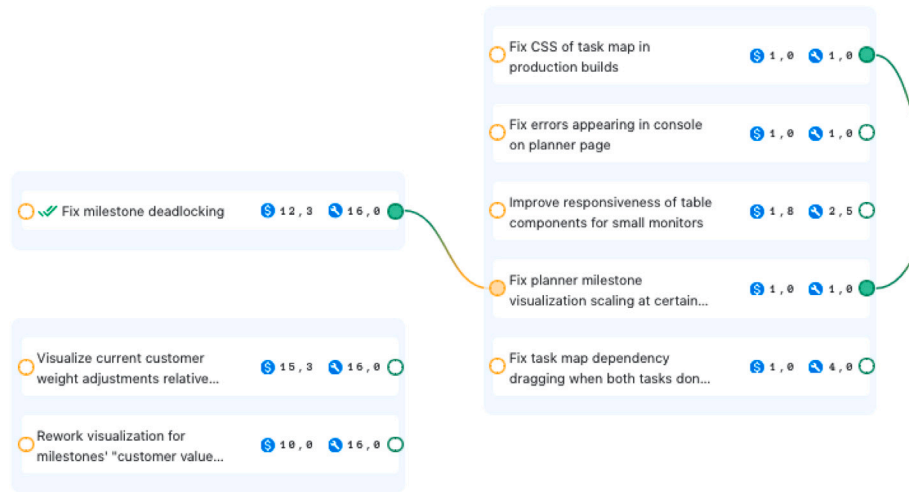


Fig. 1. An example screenshot of how the open-source tool Roadmapper represents information. The figure depicts how information concerning dependencies and synergies between features can be represented to consider them for development. The rectangles with textual descriptions represent features. Each feature has been assigned a value and complexity score, depicted by the icons in that order. Grouped features indicate synergy advantages gained from completing the features simultaneously. The lines represent technical dependencies, indicating that one feature must be completed before another.

collection [41], reachability within software execution [42], and the software team's confidence in delivering software to clients [43]. Other studies have shown that developers frequently require information related to their coworkers, tasks, and software artifacts; [39] features, products, and components have been shown to be the most important artifacts [44]. As an illustrative example of supporting information needs, Fig. 1 depicts how information to support product development decision making can be represented. The example includes information such as features, their value, implementation complexity, synergy advantages for implementing features together, and technical dependencies between features.

Regarding specific information needs, context is key, as the needs may vary greatly depending on domain, task, project, and other factors. Discovering practitioners' information needs is not straightforward because needs may change with performed activities [44,45]. One example is bug tracking, for which practitioners need different types of information depending on a bug's life cycle [45]. Tools, such as PRIME [46], have been proposed to map practitioner information needs, but few studies have proposed solutions encompassing practitioners' information needs regarding software development. Studies related to information needs often have a narrow scope and focus on informational content (the specific information practitioners need regarding a topic or activity). However, research on identifying and operationalizing the different drivers of knowledge sharing for software development – factors that affect information exchange – is still in its infancy [47].

2.3. Tailored software visualization

Software visualization refers to the mapping of software artifacts to graphical representations [48] and facilitates the effective use and understanding of computer software [49]. Agile teams use visualizations for design, development, communication, and progress tracking [8], and in the context of software engineering, they use visualization aids to analyze and understand various structures through visual means [50]. Contemporary software visualizations primarily support tasks that involve the design, implementation, and maintenance of software systems [11] and mainly target software developers with source code information [10,11]. Research on these types of visualizations mainly focuses on the structure, behavior, and evolution of software [10,11,51].

Visualization frameworks (e.g. [14,15]) cover several aspects, such as tasks, why the visualization is needed, who will use it, what data

are needed, and how information should be presented. Therefore, a key issue in software visualization lies within design and validation. Visualizations may often lack a clear focus on specific engineering tasks [52], even though software visualizations generally target specific tasks in their validation [53]. Moreover, software visualizations rarely include strong validation [54], especially with real users. Previous studies have examined whether visualizations target the generic ideas of practitioners rather than their specific roles that exist in the real world [11,55]. A deeper understanding of practitioner usage scenarios could lead to improved practices [54]; however, more research is needed to understand why specific visualizations are effective for certain practitioners and tasks [56].

Recently, there has been increasing interest in tailoring information and document representation to suit practitioner-specific needs. Software requirement specification related studies [57–59] have discussed the problem of varying practitioner information needs and have highlighted the need to further understand the needs for tailored information representation. For example, developers and managers require different information accompanied by tailored representation to suit their needs [44]. Tailored information representation has been shown to improve software managers' decision making regarding technology appropriateness [60], for example; however, contemporary visualizations are often created from specific, pre-determined perspectives. Recent studies have suggested solutions, such as unified views [61] or domain specific languages [62,63], that show promise in allowing practitioners to create visualizations that match their specific needs.

In summary, the related research to date has tended to focus on source code-related information and visualizations, catering largely to software developers attempting to understand software artifacts. Software development is a knowledge-intensive process that requires collaboration between practitioners to produce software artifacts. To collaborate, practitioners, including those not directly working with source code, require need-specific information about the software development process, though from different perspectives. Few previous works discuss both the information and visualization needs of practitioners in relation to software development. What information do practitioners need, and how should it be presented to them? To answer these questions, we aimed to help bridge the gap between stakeholder-specific information needs and visualization in software engineering.

Table 1
Overview of the characteristics of the companies that participated in this study.

Company	Domain	Personnel
C1	Analytics, information management	50
C2	Robotic software testing	120
C3	Development, service design, consulting, and software products	470

3. Research methodology

The aim of this study was to obtain an in-depth understanding of the types of information and visualization needs different software development practitioners have and how the needs can be effectively met. To this end, the following research question was chosen: *What kind of information is needed to support different practitioners during software development?* To answer our research question, we chose a qualitative research approach combining workshops, semi-structured interviews [64], and a thematic analysis [65]. We conducted six workshops and 12 semi-structured interviews in 2019 and 2020.

3.1. Planning and company selection

This study was conducted as an industry–academy collaboration in the context of the ITEA3 VISDOM¹ research consortium. The research consortium provided an opportunity for a long-term and in-depth collaboration with relevant companies, which facilitated the collection of high-quality data. The three companies that participated in this study were partners of VISDOM, and each developed commercial software products and offered consulting services. The companies varied in different aspects, such as domain and size, and their characteristics are listed in Table 1.

Despite differences in size and domain, the participating companies shared the well-aligned goal of improving their visualization capabilities under the research consortium. The selection of companies represents typical SMEs in the Finnish software development industry. This heterogeneous selection of practitioners, including those not directly associated with software development tasks, provided good participants for this study.

3.2. Data collection

The research process (Fig. 2) included three main phases: (1) industry partner use case exploration and definition workshops; (2) primary data collection and analysis with 12 semi-structured interviews (four per company) and a validation of the results of the three workshops (one per company), during which the initial results were summarized and presented; and (3) thematic analysis and synthesis of the data to obtain further insights.

The data collection process began with a use case workshop with each company (Fig. 2, Phase 1). The first research phase aimed to explore and refine the concepts of what each company wanted to achieve within the project and what their needs were concerning information presentation and visualization. The companies all had their own use cases within the project for modern DevOps development-related visualizations, as the research consortium’s goal was to implement new visualizations and dashboards related to these use cases. The workshops were recorded, and the participating researchers took notes during the events. At least three researchers were present at each workshop; one researcher acted as a facilitator for the discussion, while the others focused on taking notes. The workshops included round-table discussions on the company’s use case, and all participants were encouraged to provide feedback from their own perspectives. Each workshop also

included a drawing session, where mockups of the proposed dashboards were drawn to elicit comments and ideas (see Fig. 3 for an example). The results of the drawing sessions were photographed as additional data points, and the workshops were recorded but not transcribed. Researchers then summarized the workshop sessions as preliminary analyses of the company use cases. After discussing the preliminary results and feedback with the companies, a summary use case document was provided to each company. The use case documents outlined how each company took part in the larger scheme of the multi-national project, their current problems, and their needs.

In Phase 2, we interviewed four practitioners from each company, comprising 12 interviewees. The second phase aimed to provide the participating companies with a more objective starting point for their tasks within the project by collecting and preliminarily analyzing data on the challenges and needs of each company. The interviews were carried out between November 2019 and February 2020. Our sampling strategy for informants was a combination of convenience sampling and maximum variance sampling [66] within the companies participating in the study. The researchers did not influence subject selection with the exception of requesting participants with varying roles; thus, the interviewees were selected by the companies based on their relevance to their use case and availability. The interviewees represented a wide range of roles, such as software developers, software architects, product owners, consultants, and executives. Each semi-structured interview included open-ended questions and followed a commonly agreed upon interview protocol with the following structure:

- Background information: name, position, work experience, and role;
- Description of the use case from the interviewees’ perspectives;
- Interviewees’ roles in the use case;
- Interviewees’ typical tasks in the use case;
- Information that the interviewees needed to perform the previously mentioned tasks;
- How interviewees visualize the previously mentioned information;
- Whether the interviewees could identify other information needs that would benefit them in their work; and
- Whether the interviewees had anything else they would like to discuss or any other remarks they would like to make.

The interview template’s outline, including the use case description from the previous phase, was sent before the interviews so that the interviewees could familiarize themselves with the material. The interviews were organized within company premises when possible, lasted one hour on average, and were recorded with informed consent. Two researchers were present for each interview. One researcher acted as the main interviewer and asked follow-up questions based on the interviewees’ answers, and the second researcher took notes and asked clarifying questions when needed. The recordings were then transcribed using professional transcription services. The transcripts were then sent to the interviewees in case they wanted something redacted or corrected. Both the transcripts and interview notes were used as data sources for this study; however, one of the interviews was not successfully recorded due to a human error in operating the recording equipment, and only the notes from the interview were included in the analysis.

One researcher performed a preliminary analysis based on the interview transcripts and created a summary of the results for each company. The summaries, documented in the form of PowerPoint² slides, were presented to the companies during the second workshop with the intention of reporting and discussing preliminary findings, validating them with participants, and collecting feedback. The second workshops featured the same participants, when available, as the first ones and were recorded with informed consent but not transcribed.

¹ <https://itea3.org/project/visdom.html>.

² <https://www.microsoft.com/en-us/microsoft-365/powerpoint>.

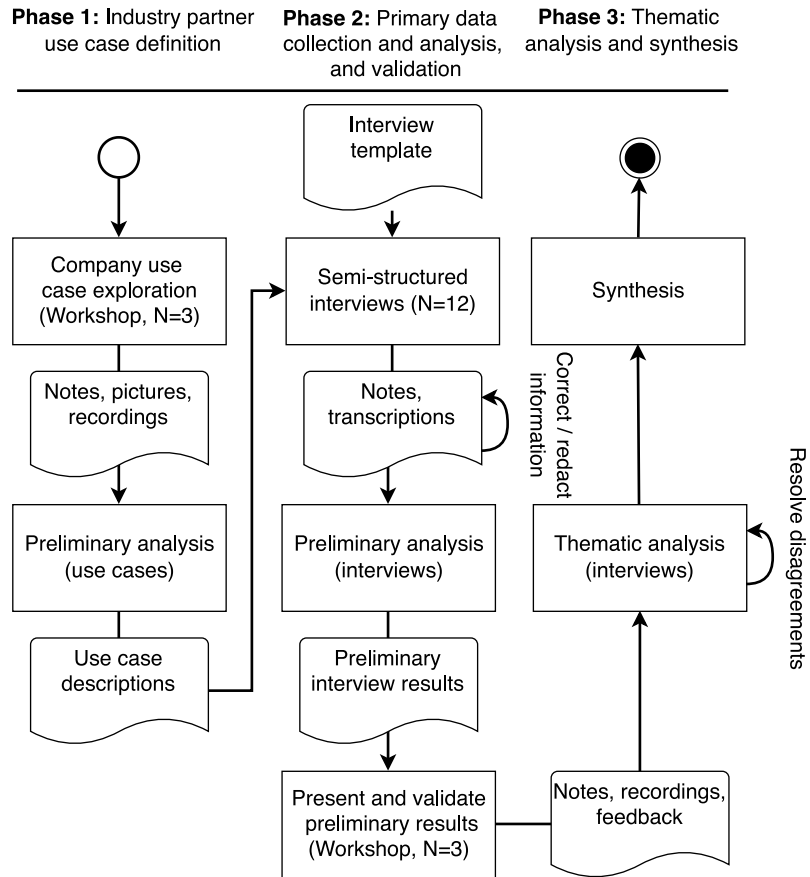


Fig. 2. The research process consisted of three phases: (1) use case exploration workshops, (2) semi-structured interviews, and (3) result-validating workshops and synthesis of the results.

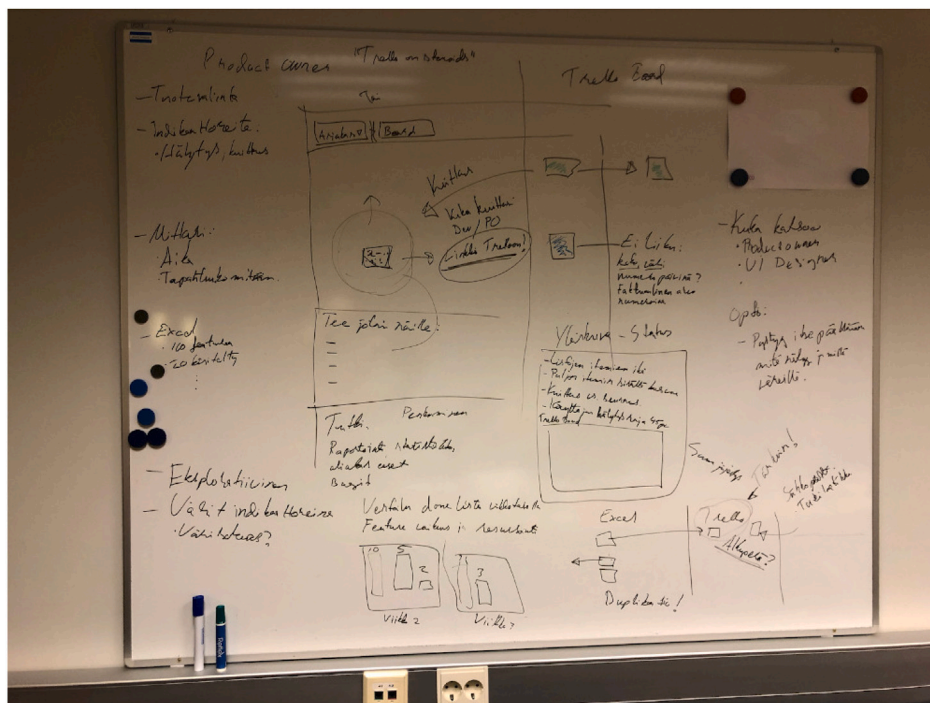


Fig. 3. A typical workshop drawing session result drawn by researchers to facilitate discussions with the participants.

3.3. Data analysis

A thematic analysis – searching across a dataset to find repeated patterns of meaning [65] – was used to analyze the collected data in Phase 3. The third research phase aimed to analyze the collected data further and distil the findings for both practitioners and academic audiences. Our approach to the analysis was two-fold. A theoretical approach guided by an analytic interest in the research area was used to categorize segments of text based on our research aims. This was followed by an inductive step of data-driven coding to theorize about the patterns within each category and to determine their significance and broader implications.

Three researchers took part in the theoretical analysis. The transcripts were coded using NVivo³ software into predefined categories: (1) self-described roles, experience, tasks, and related stakeholders; (2) information needs; and (3) visualization needs. This phase aimed to ensure a consensus on the relevant coded sections for each interview, even though this was not a thematic analysis per se. Each of the three researchers was assigned two-thirds of the collected data, meaning that at least two researchers coded each transcript. Afterwards, coding conflicts were resolved via a round-table discussion. NVivo's interrater statistics (percentage agreement and Cohen's Kappa [67] value) were used to discuss the most conflicting codes.

After that, one researcher conducted a (reflexive) thematic analysis [65,68] of the data from the previous group coding. As for philosophical assumptions, we based our analysis on an interpretive world view [69]. The analysis consisted of six nominal steps [68], which were iterated several times to reflect on the created codes and themes. Step 1, data familiarization, was already well-covered in the previous analysis, as the researchers had read the material several times. Step 2, data coding, began with semantic coding at the surface level, for example, concretely and explicitly requested information or visualizations. The codes were later iterated during the analysis and refined to uncover the latent meaning of each code. Latent coding represents what the interviewees meant instead of what they described word for word. For example, when specific alerts were described verbally, the descriptions were analyzed into themes dealing with attention (Fig. 4, T2). The central concept was interpreted to be about directing personal attention to where and when it mattered the most, cutting down on aspects that require attention.

Eight initial themes were generated in Step 3 (generating initial themes): *management, roadmapping, development, user experience, business, connections, monitoring, and clients*. Steps 4 (developing and reviewing themes) and 5 (refining, defining, and naming themes) represented an iterative process of writing and refining in our analysis. In Step 4, the fit of the initial codes to the data was assessed, and it was decided that further analysis was required to capture the meaning behind the codes instead of reporting only surface-level findings. In Step 5, the themes were re-examined and condensed to tell a compelling story. The essence and implications of each code were refined with the aim of capturing the one central point for each theme. The analysis researcher presented the themes to the other researchers for discussion and refinement. In the end, three main themes remained: *product roadmapping (T1), directing attention (T2), and product usage (T3)*. The resulting thematic map is presented in Section 4, Fig. 4. Step 6, writing, was conducted throughout the process with the aim of telling a compelling story, which helped refine the themes as a reflective process.

³ <https://www.qsrinternational.com/nvivo-qualitative-data-analysis-software/home>.

Table 2

Overview of the self-described roles by the twelve participants of this study. Note that UI refers to user interface.

Company	Informant	Role
C1	P1	Chief operating officer
	P2	Product owner, UI designer, consultant
	P3	Testing manager
	P4	Principal consultant, product development leader, team leader
C2	P5	Delivery manager
	P6	Product development leader, product owner
	P7	Senior developer
	P8	Software architect
C3	P9	Product conceptualizer, product owner
	P10	Product development manager
	P11	Customer experience manager
	P12	Software developer, software architect

4. Results and analysis

Twelve practitioners from three companies in different domains (Table 1) were interviewed for this study. Table 2 presents a summary of the interviewees' self-described roles. The described roles included one executive as chief operating officer, eight managers (three of them product owners), two software architects, and one senior developer. Six of the interviewees (P1, P2, P4, P8, P9, P10) described having multiple primary tasks or responsibilities or described their own role broadly or non-specifically, an example being:

I'm a kind of mixed worker, but I'm responsible for product development, production environment, or overall, these environments running in the cloud. (P10)

Interviewees in mixed roles mentioned issues with processing information caused by multiple roles. However, these interviewees were a minority in the sample. Performing tasks in multiple domains, such as development and business, could lead to issues with information, such as:

You could easily drift to the other role at that time if you see information that does not belong to this role. (P2)

The tasks were described in different levels of detail, some being relatively abstract. The tasks were divided into five categories based on their similarities: management tasks, software business tasks, software development tasks, product roadmapping tasks, and client-related tasks. Next, examples are given of each category.

Management tasks were performed by nine participants. These tasks encompassed decision-making tasks and responsibilities for aspects of development. There were also concrete tasks, such as release and sprint planning, backlog management, and workload estimations:

In practice, I make the decisions on what deployments we do and when we do them with what content. (P5)

Business tasks were performed by three participants. These tasks were related to software business decision making, resourcing projects and other ventures, sales, marketing tasks, sub-contractor management, and numerically tracking development. Business-related tasks were also carried out in relation to development and roadmapping:

Company strategy related things how, our software product should be developed so that it specifically supports our business ideas. (P8)

Development tasks were conducted by six participants. These tasks encompassed topics that were related to producing concrete software artifacts, including development, testing, design, and documentation tasks. An example is as follows:

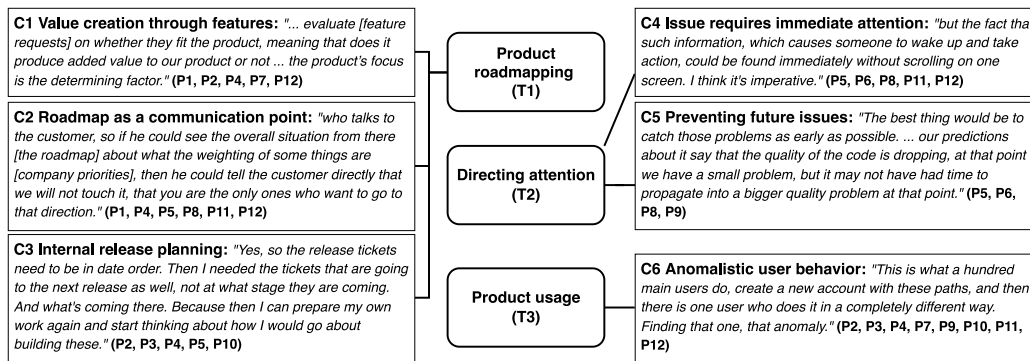


Fig. 4. A thematic map regarding information and visualization needs. Themes (T1, T2, T3) are presented as the rectangles with rounded corners, while individual codes are represented in the outermost rectangles. Each code is accompanied by a quotation that exemplifies the code, followed by parentheses that describe which participants (Table 2) were connected to each code. The lines represent connections between themes and individual codes.

I'm at the moment handling most of the UI work, so whatever, our main product, which is [redacted], so what you see in your browser is, most of it, is done by me. (P7)

Roadmapping tasks – long-term planning of upcoming product development content, stages, and steps based on the strategic goals of the company – were done by nine participants. These tasks were related to managing feature requests, client feedback, and development ideas to provide material for long-term planning. An example is:

Which details should be thought of. Is that possible at all, for XYZ reasons. And, from that, a kind of roadmap is created that is based on the product and what kind of stage, what kind of pieces. (P6)

Requests and feedback were collected from multiple sources, such as emails, phone calls, discussions, internal ideas, and direct requests. Then, the requests were formulated into actionable development items, generalized to serve the product instead of individual clients, queried for missing information, and translated to information repositories, such as Trello.⁴ The features were then estimated based on their workload or desirability from a business angle and prioritized, both internally and with clients, to select the next items for development. An example is as follows:

We must, of course, evaluate the client, the problem, and then how much time does it take. These go into what order they are done and whether they are even fixed. (P10)

Client tasks were performed by six participants. These tasks included product promotions; visits with clients and events; and meeting, discussing, helping, training, and supporting clients in various situations. An example is as follows:

My job is to organize that the client is satisfied and the job gets done. I also do presence support, meaning that I promote our tool to clients. (P5)

4.1. Information and visualization needs

The results of the thematic analysis on information and visualization needs (Fig. 4) were condensed into three main themes: *product roadmapping* (T1), *directing attention* (T2), and *product usage* (T3).

4.1.1. Product roadmapping (T1)

Product roadmapping (T1) – which was comprised of the three codes *value creation through features* (C1), *roadmap as a communication point* (C2), and *internal release planning* (C3) – relates to both short- and long-term planning of software products, representing a point of convergence for different stakeholder groups' information needs. The main takeaway from this theme is that support is needed to facilitate information exchange between stakeholder groups. This support must be implemented so that information representation supports the tasks of each stakeholder, calling for tailored views. We found two levels of communicating information between stakeholder groups in software product roadmapping: a high-level view (C1, C2), represented as a software product roadmap that communicates how the product is being developed in terms of features, and a low-level view (C3), depicted as a release plan with milestones and tickets that communicates what is happening in terms of practical development.

Value creation through features (C1) revolves around selecting product features to create optimal value with software product development. The issue with selecting features in software product roadmapping manifested as unsystematic management of customer feedback and requests, a somewhat ad-hoc process of collecting and evaluating feature candidates and customer feedback while attempting to communicate between different departments and stakeholders (sales, development, and customers). Several viewpoints are needed to evaluate information – which is often incomplete or only based on estimations – so that product development yields optimal value and retains coherency with company strategy. However, to do so, the requests and feedback must be collected and processed to feature candidates, requiring similar practices regardless of the stakeholder group. The main takeaway of this code is that a shared view was needed to make the information visible and to evaluate the value of features, which supports different stakeholders discussing the value of features to decide how the product should be developed further.

Evaluating and prioritizing feature requests were usually done by comparing how much value would be created with said feature versus the effort needed to develop it—or if a specific customer would leave if the request as ignored. The synergy between requests was considered a critical factor in deciding which features were valuable inclusions to the software roadmap; if several different sources requested a similar feature, a request aligned with something the company wanted to do regardless, or a request resurfaced periodically, a generic version could be delivered to several customers with a potentially high-value output. An important observation here is that value could be estimated by discussing the synergy of features without accurate numerical estimations, allowing even ambiguous data to be used as a basis for decision making as long as the information is visible and understandable. However, technical compatibility alone did not necessitate including said features automatically, as each feature was judged on how it fit the company's internal priorities and the product's scope:

⁴ <https://trello.com/>.

If we could see in one place how much something has been requested in the longer term and what we expect the business value and workload to be [...] which things relate to each other when we start implementing them [...] (P12)

Providing sufficient – and preferably structured – information to those evaluating the requests was considered to save time. Practitioners would not have to contact customers for additional information, which could displease customers who had already explained the details to someone else. Regarding the information content of feature requests, the pieces of information needed were when the request was received, who received it, where the request originated, what communication medium was used, what exactly was requested (detailed description or word for word what the customer said), why the feature was requested, the underlying reason or problem (used to generalize the request), and whether any actions (such as further contact) were promised by the company:

[feature requests] can come up here and there, and we should catch them. That's one challenge. (P2)

The **roadmap as a communication point (C2)** revolves around product development-related discussions between sales, development, and customers. The main point of this code is that a high-level view is needed to facilitate discussion between stakeholder groups concerning feature-selection commitments (after feature value has been collaboratively identified in *C1*). Once the participants can see the current software product roadmap, they can understand the viewpoint of others more easily, aiding discussion. The second important function of this view is to provide an up-to-date reference for different stakeholder groups, represented in a way that supports their tasks. Once a roadmap is formulated, feature selection and prioritization commitments must be communicated to synchronize stakeholder groups: which features could be promised to customers from a sales perspective, which should be delivered from a development perspective, and how the promised features are progressing from a customer perspective. A concrete example of using this information is sales events, where indicating features that are already being considered makes them easier to promise. Likewise, development-related stakeholders would know which features have been promised for delivery, making it possible to consider upcoming changes while implementing features that make changes easier. Finally, communication with customers was considered an integral part of the overall process. The interviewees envisioned several customer-facing visualizations to provide more structured information for all parties involved, reducing the need for explicit communication between stakeholder groups:

[...] the people who do sales might have a tendency to overfit the product to client requirements [...] whoever speaks with customers, if they could see the whole situation [...] then they could tell the customer directly that we are not going to touch that. (P12)

The **internal release planning (C3)** relates to delivering the selected features through releases. While software product roadmaps often present a long-term plan for a product, a concrete short-term release plan is also needed to communicate development aims, including forecasting whether releases will succeed. A more low-level view was requested in the form of a release plan, including development tickets divided into milestones, to support and understand what is happening in development. An essential information need concerning release planning was how features were divided into releases or development cycles, depending on which suited the company's development practices. This information was used to determine the upcoming features for development, their completion dates, and how the current iteration was progressing, providing an overview of the current and upcoming development cycle. However, tracking the progress of individual development items was challenging, especially regarding time

estimations, which were considered to be educated guesses more than accurate predictions. This made it challenging to forecast the likelihood of releases succeeding. Managers considered forecasts to be critical. Related information, such as release volume and what kind of tickets would be produced per cycle, was used to reflect whether releases were likely to succeed:

Yes, so the release tickets need to be in date order. Then I need the tickets that are going to the next release as well, how complete they are. And what their contents are. Because then I can prepare my own work again and start thinking about how I would go about building these. (P3)

4.1.2. Directing attention (T2)

Directing attention (T2) revolves around proactively directing the attention of practitioners to points of interest. The main takeaway of this theme is that a dashboard focused on causing actions should present information passively, capturing attention only when a reaction is needed and presenting the alert in a way that conveys all the necessary information to resolve the issue. A dashboard with minimal visual content limited to one screen was described as the best representation of this information by presenting as little information as possible until attention was required. When a reaction was expected, the following information was to be included: a description of the issue, the (root) cause (preferably trackable via sub-metrics to software artifacts), a time scale for the reaction, and a recommended action for what could or should be done about the issue. These results were the most relevant to manager-type participants who often had responsibilities over some development areas. The following extract demonstrates the notion of alert-based dashboard views:

But the fact that such information, which causes someone to wake up and take action, could be found immediately without scrolling on one screen. I think it's imperative. (P6)

A key observation for this theme is that half of the interviewees either wanted or described visualizations with traffic lights (green for good, yellow for attention required, red for bad), colored up and down arrows, or other color-based solutions to denote whether something was "good" or "bad". Traffic light-style visualizations, perhaps coincidentally, resolve an issue reported with visualization; they quickly indicate whether tracked values are "good" or "bad" without necessitating contextual knowledge to translate raw numbers into practical meaning. Interviewees also noted that traffic light visualizations could indicate that the individual should start focusing on specific issues.

The theme was split into two codes depending on the issue's time scale: *issues requiring immediate attention (C4)* or *preventing future issues (C5)*. The codes express the main questions to be answered in this theme: *Is there an issue right now* and *Will there be an issue at this rate?* For the first question, alerts were clearly the preferred communication method when information was required immediately. In the second scenario, where attention was required later, traffic light indicators based on trend information was the method of choice. An important distinction was that these lights would not be monitored constantly, and a decision could later be made to investigate the issue further, possibly using other systems with more in-depth information and data.

Issues requiring immediate attention (C4) revolve around reactions with short time frames. Deviations in process-related metrics should be communicated via alerts, presented on an empty screen, for which lower and upper thresholds should be given. When the value exceeds those thresholds, an alert should be generated, requiring a reaction. However, each alert should include contextual information to interpret what the alert is about, what caused it (sub-metrics), and what actions should be taken to resolve the issue. In addition, notifications were requested for managing task-related information flow to prevent information overload. In other words, task-related data should be filtered to understand which items were pending input and attention. The following excerpts emphasize the nature of the code:

Wake-up is the thing in my mind. I would like to know if we have a problem and what I should do about it. (P6)

I don't want any chart stuff, I want to see what's on fire. (P8)

Preventing future issues (C5) relates to providing advanced warnings at a glance on a dashboard. When attention is required soon instead of immediately, the information content and the representation of issues should change but still avoid creating distractions. A single view was proposed, including several traffic light-themed trend indicators communicating whether the current development direction would eventually lead to problems. If the view is all green, no attention is required. Otherwise, a decision can be made whether to react or not. The indicators were requested to communicate which trend was changing, such as software quality, and the underlying reasons for the change, such as sub-trends and metrics to software artifacts. When a decision is made to investigate, the indicators should communicate the issue and direct the investigator to explore or search for further information from other sources. The following extract provides an example of this code:

What would help my job is some kind of traffic light. Have the clients listed, and [...] when my dashboard turns red for some client, I could then know to study it further and drill down to data. (P11)

4.1.3. Product usage (T3)

Product usage (T3) relates to how end users use software products and how abnormalities in software product usage can be detected. The main takeaway from this theme is that practitioners need to understand how a software product is used. According to the interviewees, usage that differs from what was designed can indicate deficiencies in the software product's user interface (UI), allowing or resulting in end users using the product in a way that was not intended. Usability issues and other software product UI design deficiencies should be detected and addressed before negative user feedback is received.

Anomalous user behavior (C6) relates to identifying usage paths within the UI that deviate from the designed way of using the product. According to the interviewees, deficiencies in UI design quality may cause unplanned usage scenarios. While most usage scenarios could be accounted for in the design phase, evaluating each possible way of using the product beforehand was deemed infeasible. In practice, the only way to catch specific usability deficiencies is through negative customer feedback, requiring a solution to catch design issues beforehand. Discovering anomalies in software usage would allow interviewees to address design issues and offer proactive customer support. A suggested realistic representation of this information was paths within the product grouped by user type with differences laid out for comparison. Keeping track of actions taken regarding UI was also deemed helpful for the manual testing of the software, supporting the repeatability of exceptions when reporting them to developers:

I would want us to be able to see, in principle, all the users and their [ways] of using [the software], and then we could find those that use it in a totally different way. [...] These hundreds of main users create a user with this kind of usage path, and then there is that one user that does it completely differently. Finding that one, the anomaly. (P10)

User interface analytics were also described as necessary to understand a product's use. How well a product works and how users use the product are vital to, for example, determining whether feature additions were successfully generalized to a broader audience. Product usage statistics could also be used to evaluate whether the inclusion of a feature was effective in terms of additional users, and unused features are also prime candidates for removal from the product. Examples of needed analytics included login counts, scrolling positions from browser windows, time spent on individual parts of the product, and other behavioral data (e.g., how many orders were placed within the software or incomplete orders left in the shopping cart). However, the interviewees deemed collecting meaningful usability data an ongoing challenge.

Table 3

The main results of this article and their implications categorized by themes.

Theme	Finding	Implication
Product roadmapping (T1)	Tooling support with a shared view is needed to facilitate information exchange between stakeholder groups. The information must be presented in a way that supports their different tasks. Furthermore, the information has two levels for presentation: a software product roadmap for a high-level view to convey what is being discussed and a release plan for a more accurate view depicting what is happening in the field.	Visual information representation catalyzes discussion between stakeholders, allowing even ambiguous information to be used for decision making. The collaboration results must be communicated and visible to all relevant stakeholders, each requiring a different representation of that information based on their tasks.
Directing attention (T2)	A dashboard should present information passively to support daily work. When an alert is generated, it should communicate the necessary information to interpret it, including context, reaction time scale, and required action.	A user-focused way of collecting, processing, and analyzing data is needed. Information should be represented in a way that guides interpretation, instructs on how to address the issue, and provides means for exploring details and context.
Product usage (T3)	Product usage deviating from designed usage can indicate design deficiencies. Analytics should catch product usage deviations systematically to address design flaws before they manifest into negative user feedback.	A mechanism is needed for understanding software product usage in customer organizations. The data collection must be automatic and designed to detect deviations in product usage.

5. Discussion

5.1. RQ: What kind of information is needed to support different practitioners in software development?

Adopting new development practices affects how practitioners collaborate, requiring companies to form a closer connection between business strategy and software development. To this end, we investigated the information needed to support different software practitioners' tasks and how it should be represented. It should be noted that discovering information needs remains challenging, as the needs may change with performed activities [44], which are often tied to how teams or companies develop software. Further research is needed to study how the results apply in general. Our study was conducted in the context of three Finnish SMEs utilizing agile methods to develop software, having either adopted DevOps practices or striving to do so. However, practitioners with different backgrounds were selected for this study to focus on discovering how different stakeholder groups can be supported regarding information needs and representation. When combined, our results (Table 3) show that supporting information exchange between stakeholder groups is vital for product development, as visual information representation catalyzes discussion between stakeholders. Furthermore, we found that information representation requires methods to guide users in their interpretation and allow them to explore details and context while maintaining a simplified appearance. Data collection is essential to support stakeholders, for which more user-centered practices are needed.

T1 highlights the convergence of information needs from different stakeholder groups in software product roadmapping. Roadmapping is a collaborative process integrating different stakeholder groups and their value proposals [70] to explore alternative software product development decisions over time. In accordance with the presented results, studies have demonstrated that connecting customer demands

and the rapid delivery of products has been an increasing trend in continuous software practices [5] and that establishing visibility for feature planning can strengthen the link between development and business decisions [6]. However, incorporating customer feedback into development has proven challenging [71,72], and fragmented customer knowledge is one of the main issues related to software roadmapping activities [73], as shown in this study. To this end, our results show that support is needed to facilitate discussions between stakeholder groups in two related areas.

First, support is needed to collect and evaluate feature requests and their value. We found that a shared view was needed to promote information visibility when stakeholder groups evaluate the value of features. This finding implies that combining information from several sources and judging it in collaboration with several stakeholders allows even ambiguous data to be used for product development-related decision making. A concrete example is feature synergy used as a proxy to evaluate feature value instead of more accurate numerical information, as cost estimations are often inaccurate. However, the reliability of the input data is a concern, and practices for their systematic collection are needed.

Second, support is needed to facilitate discussion between stakeholder groups in software product roadmapping. We found that visual information representation catalyzes discussion between stakeholders. Furthermore, our results show that these needs manifested in two levels. A software product roadmap was described to provide a high-level view of the current development direction. In this regard, support is necessary to facilitate the discussion between stakeholder groups when making product development decisions. An essential aspect of providing such a view is to use it as a reference for the involved stakeholders; sales events were indicated as a concrete example. Additionally, a lower-level view of development milestones is needed to communicate practical development aims and statuses, including forecasting whether the planned releases will succeed in time. Both views aim to lower the complexity of development decisions by aggregating information and ensuring that each stakeholder understands the current plan similarly. We suggest companies use their product roadmaps as discussion points with key stakeholder groups, such as development, sales, and customers.

The findings concerning T2 related to directing attention – with minimal information – to where and when it matters. This can answer the questions, *Is there an issue right now?* and *Will there be an issue at this rate?* An interesting and counter-intuitive observation is that the interviewees rejected the notion of dashboards filled with information when considering their daily information needs. Instead, a minimal representation of information, focusing on causing reactions, was preferred. Our results for this theme imply that information should be represented in a way that guides how each metric is interpreted, instructs the user on how to address the issue raised, and provides a means for exploring details and contextual information. However, realizing these requirements in practice is a tremendous challenge. Moving towards a solution, we echo Buse and Zimmermann's [44] call for a user-focused way of collecting, processing, and analyzing data. Users' tasks and issues should define what data are collected to create solutions to their problems (instead of collecting data without explicit purpose). These needs imply another wicked problem; users and their tasks must be well-known to generate task-based alerts, and the development process must be well-known and modeled to detect trends that reliably indicate a negative turn.

Product usage (T3) relates to understanding how the deployed software is used by detecting usability issues. One of the primary motivations for detecting these issues was negative user feedback, which was considered an indicator of possible design flaws. According to the interviewees, it is infeasible to plan for each possible way that the software product might be used before releasing it. The findings from this theme imply that a mechanism is needed to understand software product usage in customer organizations. This is especially important

when the product is being sold to several customers, such as in cases of software-as-a-service strategies, where a customer representative may not be present for consultation. Another requirement is that the data collection must be automatic and designed to detect end-user deviations in product usage compared to the designated use of the software. We recommend that further research be undertaken in the following three areas.

First, discussion between stakeholder groups should be supported. Visual information representation catalyzes discussion between stakeholders, making complex topics easier to discuss. However, how a tool should be built to support information exchange between agile software product development stakeholder groups is still being determined. Future research should investigate how tooling support can benefit information exchange between stakeholder groups.

Second, value evaluation is needed in software engineering. Estimating value accurately can sometimes be impossible due to imprecise efforts and market evaluations. Feature synergy has been identified as one of the most important factors in evaluating value without accurate numerical estimations. Future research should investigate how synergy between client requests and feature candidates can be used to estimate value output for software business decisions.

Third, automated anomalous usage detection is needed. Even though contemporary software suites often include comprehensive analytics capabilities, automatically detecting usability issues in deployed software remains challenging. Detecting deviations in end-user paths within UIs was identified as one of the most prevalent needs in terms of product analytics. Future research should investigate how data should be collected and analyzed to detect anomalous usage of software products.

5.2. Limitations and validity

This study's limitations and validity are discussed in terms of Braun and Clarke's thematic analysis [65] guidelines and Creswell and Miller's [74] validity in qualitative research guidelines. The participating companies decided on the number and selection of informants for this study, representing their view of personnel dealing with the issues determined during the first workshop with the companies. The number of interviewees was left relatively low ($n = 12$), but data saturation was reached, and little new information would have been captured by adding more interviews. This view was supported by the study participants when the preliminary findings were presented back to them, and they were given a chance to add to the findings.

The reflexive analysis of collected data means that we attempted to uncover the meaning behind the words and examples told by the interviewees using our understanding of the companies' work processes and of agile software development in general. During the thematic analysis, we first aimed for coding reliability [75] to obtain a shared familiarity with the data, and we discussed the perceived meaning behind the interviews at length. Digging deeper into the meaning of the interviewees' narratives, we then used a reflexive approach to formulate the themes present in this study. We aimed to tackle validity in terms of deep and rich data descriptions [74] that go beyond the facts stated in the interviews and create an understanding of the ways of working in agile software development. During the dissemination of the study, we discussed the research with each other. However, some researchers were only present for part of the interviews. To gain perspective, the research was subjected to internal scrutiny to ensure that the presented findings followed everyone's understanding of the field.

Regarding the aims of our study, the researcher lens was focused on understanding the inner workings of agile software development from the point of view of several internal stakeholders in companies. Our results reflect companies brought together by the same research project and the agile software development practices they employed. In future work, we will determine if these conclusions can be applied to projects using different development methods from a general software engineering viewpoint.

6. Conclusion

This study's main goal was to better understand the information needed to support different practitioners in agile software development, for which we presented the results of six workshops and 12 semi-structured interviews conducted with three Finnish software SMEs. The informants in this study represented a wide range of roles in agile software development, including practitioners not directly related to producing software. A thematic analysis was conducted, from which three main themes were created. Some of the results presented in this article echo well-known findings in software engineering, such as closing the feedback loop with customers [16]. The fact that these findings still emerge implies that existing solutions do not solve these issues or that companies must learn to utilize the proposed solutions in practice.

Our results show that visual information representation catalyzes discussion between stakeholders, indicating that supporting information exchange between stakeholder groups is vital for product development. This can result in numerous information needs from different perspectives. These results contribute to identifying the types of information executives need to plan and evaluate features and how sales and marketing reconcile their expectations with development [5]. Furthermore, we found that dashboards that convey information passively until a reaction is needed were preferred to support the daily information needs of practitioners. Information in these dashboards should guide users in interpreting each metric and allow them to explore details and contexts while maintaining a simplified appearance. Finally, data collection is essential to support stakeholders, and more user-centered practices are needed for this data collection.

Although the current study is based on a relatively small sample of participants, the findings suggest key communication points in software processes, where practitioners with different perspectives require information in different forms to complete their daily tasks. The software engineering community should review the information needs of practitioners from a more holistic view to represent and visualize software processes. This especially relates to how visualizations could be made more relevant to the day-to-day work of software professionals and how communication could be facilitated between stakeholder groups in agile software development. To this end, we have already begun working toward a software roadmapping tool that facilitates communication of different roles (Fig. 4, T2) in software engineering.

CRedit authorship contribution statement

Henri Bomström: Methodology, Formal analysis, Investigation, Data curation, Writing – original draft, Writing – review & editing. **Markus Kelanti:** Conceptualization, Methodology, Formal analysis, Investigation, Data curation, Funding acquisition, Project administration, Writing – review & editing. **Elina Annanperä:** Conceptualization, Methodology, Formal analysis, Investigation, Data curation, Writing – review & editing. **Kari Liukkunen:** Funding acquisition, Investigation, Supervision, Writing – review & editing. **Terhi Kilamo:** Data curation, Investigation, Validation. **Outi Sievi-Korte:** Data curation, Investigation, Validation. **Kari Systä:** Conceptualization, Methodology, Investigation, Funding acquisition, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

Acknowledgments

The authors thank the company representatives for the time and expertise they provided for the interviews and workshops. The work was supported by the ITEA3 project VISDOM and partly by the ITEA3 project Oxilate. The authors thank Business Finland for the funding. Furthermore, we thank the anonymous reviewers for their insightful comments that helped improve this article.

References

- [1] J. Järvinen, T. Huomo, T. Mikkonen, P. Tyrväinen, From agile software development to mercury business, in: International Conference of Software Business, Springer International Publishing, Springer, Cham, 2014, pp. 58–71, http://dx.doi.org/10.1007/978-3-319-08738-2_5.
- [2] C. Ebert, G. Gallardo, J. Hernantes, N. Serrano, DevOps, IEEE Softw. 33 (3) (2016) 94–100, <http://dx.doi.org/10.1109/MS.2016.68>.
- [3] A. Hemon, B. Lyonnet, F. Rowe, B. Fitzgerald, From agile to DevOps: Smart skills and collaborations, Inf. Syst. Front. 22 (4) (2020) 927–945, <http://dx.doi.org/10.1007/s10796-019-09905-1>.
- [4] L. Riungu-Kalliosaari, S. Mäkinen, L.E. Lwakatare, J. Tiihonen, T. Männistö, DevOps adoption benefits and challenges in practice: A case study, in: Product-Focused Software Process Improvement, Springer International Publishing, Cham, 2016, pp. 590–597, http://dx.doi.org/10.1007/978-3-319-49094-6_44.
- [5] B. Fitzgerald, K.-J. Stol, Continuous software engineering: A roadmap and agenda, J. Syst. Softw. 123 (2017) 176–189, <http://dx.doi.org/10.1016/j.jss.2015.06.063>.
- [6] L. Lehtola, M. Kauppinen, J. Vähäniitty, M. Komssi, Linking business and requirements engineering: is solution planning a missing activity in software product companies? Requir. Eng. 14 (2) (2009) 113–128, <http://dx.doi.org/10.1007/s00766-009-0078-8>.
- [7] E. Klotins, T. Gorschek, K. Sundelin, E. Falk, Towards cost-benefit evaluation for continuous software engineering activities, Empir. Softw. Eng. 27 (157) (2022) <http://dx.doi.org/10.1007/s10664-022-10191-w>.
- [8] J. Paredes, C. Anslow, F. Maurer, Information visualization for agile software development, in: 2014 Second IEEE Working Conference on Software Visualization, 2014, pp. 157–166, <http://dx.doi.org/10.1109/VISSOFT.2014.32>.
- [9] M. Cataldo, J.D. Herbsleb, Coordination breakdowns and their impact on development productivity and software failures, IEEE Trans. Softw. Eng. 39 (3) (2013) 343–360, <http://dx.doi.org/10.1109/TSE.2012.32>.
- [10] A.-L. Mattila, P. Ihantola, T. Kilamo, A. Luoto, M. Nurminen, H. Väättäjä, Software visualization today: Systematic literature review, in: Proceedings of the 20th International Academic Mindtrek Conference, AcademicMindtrek '16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 262–271, <http://dx.doi.org/10.1145/2994310.2994327>.
- [11] N. Chotisarn, L. Merino, X. Zheng, S. Lonapalawong, T. Zhang, M. Xu, W. Chen, A systematic literature review of modern software visualization, J. Vis. 23 (4) (2020) 539–558, <http://dx.doi.org/10.1007/s12650-020-00647-w>.
- [12] R.E. Freeman, The politics of stakeholder theory: Some future directions, Bus. Ethics Q. 4 (4) (1994) 409–421, <http://dx.doi.org/10.2307/3857340>.
- [13] J.B. Anderson, R. Johnnsson, Understanding Information Transmission, John Wiley & Sons, 2006.
- [14] M.-A.D. Storey, D. Čubranić, D.M. German, On the use of visualization to support awareness of human activities in software development: A survey and a framework, in: Proceedings of the 2005 ACM Symposium on Software Visualization, SoftVis '05, Association for Computing Machinery, New York, NY, USA, 2005, pp. 193–202, <http://dx.doi.org/10.1145/1056018.1056045>.
- [15] J.I. Maletic, A. Marcus, M.L. Collard, A task oriented view of software visualization, in: Proceedings First International Workshop on Visualizing Software for Understanding and Analysis, 2002, pp. 32–40, <http://dx.doi.org/10.1109/VISSOF.2002.1019792>.
- [16] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, et al., Manifesto for agile software development, 2001, URL <http://agilemanifesto.org/>.
- [17] M. Yilmaz, R.V. O'Connor, P. Clarke, Software process improvement and capability determination, Commun. Comput. Inf. Sci. 290 CCIS (2012) 198–209, http://dx.doi.org/10.1007/978-3-642-30439-2_18.
- [18] M. Yilmaz, R.V. O'Connor, P. Clarke, Software development roles: A multi-project empirical investigation, ACM SIGSOFT Softw. Eng. Notes 40 (1) (2015) 1–5, <http://dx.doi.org/10.1145/2693208.2693239>.
- [19] S.R. Palmer, M. Felsing, A Practical Guide to Feature-Driven Development, first ed., Pearson Education, 2001.
- [20] K. Beck, Extreme Programming Explained: Embrace Change, Addison-Wesley Professional, 2000.
- [21] K. Schwaber, M. Beedle, Agile Software Development with Scrum, first ed., Prentice Hall PTR, USA, 2001.
- [22] M. Alqudah, R. Razali, A review of scaling agile methods in large software development, Int. J. Adv. Sci. Eng. Inf. Technol. 6 (6) (2016) 828–837, <http://dx.doi.org/10.18517/ijaseit.6.6.1374>.

- [23] D. Leffingwell, *Scaling Software Agility: Best Practices for Large Enterprises*, Addison-Wesley Professional, 2007.
- [24] C. Larman, B. Vodde, *Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum*, first ed., Addison-Wesley Professional, 2010.
- [25] T. Dybå, T. Dingsøy, Empirical studies of agile software development: A systematic review, *Inf. Softw. Technol.* 50 (9–10) (2008) 833–859, <http://dx.doi.org/10.1016/j.infsof.2008.01.006>.
- [26] T. Gustavsson, Assigned roles for inter-team coordination in large-scale agile development: A literature review, in: *Proceedings of the XP2017 Scientific Workshops, XP '17*, Association for Computing Machinery, New York, NY, USA, 2017, pp. 1–5, <http://dx.doi.org/10.1145/3120459.3120475>.
- [27] R. Hoda, L.K. Murugesan, Multi-level agile project management challenges: A self-organizing team perspective, *J. Syst. Softw.* 117 (2016) 245–257, <http://dx.doi.org/10.1016/j.jss.2016.02.049>.
- [28] R. Hoda, J. Noble, S. Marshall, Self-organizing roles on agile software development teams, *IEEE Trans. Softw. Eng.* 39 (3) (2013) 422–444, <http://dx.doi.org/10.1109/TSE.2012.30>.
- [29] H. Zhu, M. Zhou, P. Seguin, Supporting software development with roles, *IEEE Trans. Syst. Man Cybern. - A* 36 (6) (2006) 1110–1123, <http://dx.doi.org/10.1109/TSMCA.2006.883170>.
- [30] G. Beranek, W. Zuser, T. Grechenig, Functional group roles in software engineering teams, in: *Proceedings of the 2005 Workshop on Human and Social Factors of Software Engineering, HSSE '05*, Association for Computing Machinery, New York, NY, USA, 2005, pp. 1–7, <http://dx.doi.org/10.1145/1083106.1083108>.
- [31] Y. Ye, Supporting software development as knowledge-intensive and collaborative activity, in: *Proceedings of the 2006 International Workshop on Workshop on Interdisciplinary Software Engineering Research, WISER '06*, Association for Computing Machinery, New York, NY, USA, 2006, pp. 15–22, <http://dx.doi.org/10.1145/1137661.1137666>.
- [32] A. Bouraffa, W. Maalej, Two decades of empirical research on developers' information needs: A preliminary analysis, in: *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, ICSEW '20*, Association for Computing Machinery, New York, NY, USA, 2020, pp. 71–77, <http://dx.doi.org/10.1145/3387940.3391485>.
- [33] A. Hess, P. Diebold, N. Seyff, Understanding information needs of agile teams to improve requirements communication, *J. Ind. Inf. Integr.* 14 (2019) 3–15, <http://dx.doi.org/10.1016/j.jii.2018.04.002>.
- [34] A. Cockburn, *Agile software development joins the "would-be" crowd*, *Cutter IT J.* 15 (1) (2002) 6–12.
- [35] L. Pascarella, D. Spadini, F. Palomba, M. Bruntink, A. Bacchelli, Information needs in contemporary code review, *Proc. ACM Hum.-Comput. Interact.* 2 (CSCW) (2018) 1–27, <http://dx.doi.org/10.1145/3274404>.
- [36] A. Sutherland, G. Venolia, Can peer code reviews be exploited for later information needs? in: *2009 31st International Conference on Software Engineering - Companion Volume*, 2009, pp. 259–262, <http://dx.doi.org/10.1109/ICSE-COMPANION.2009.5070996>.
- [37] F. Kortum, J. Klünder, O. Karras, W. Brunotte, K. Schneider, Which information help agile teams the most? An experience report on the problems and needs, in: *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2020, pp. 306–313, <http://dx.doi.org/10.1109/SEAA51224.2020.00058>.
- [38] S.C. Müller, T. Fritz, Stakeholders' information needs for artifacts and their dependencies in a real world context, in: *2013 IEEE International Conference on Software Maintenance*, 2013, pp. 290–299, <http://dx.doi.org/10.1109/ICSM.2013.40>.
- [39] A.J. Ko, R. DeLine, G. Venolia, Information needs in collocated software development teams, in: *29th International Conference on Software Engineering (ICSE'07)*, 2007, pp. 344–353, <http://dx.doi.org/10.1109/ICSE.2007.45>.
- [40] I. Omoronyia, J. Ferguson, M. Roper, M. Wood, A review of awareness in distributed collaborative software engineering, *Softw. - Pract. Exp.* 40 (12) (2010) 1107–1133, <http://dx.doi.org/10.1002/spe.1005>.
- [41] J. Josyula, S. Panamgipalli, M. Usman, R. Britto, N. Bin Ali, Software practitioners' information needs and sources: A survey study, in: *2018 9th International Workshop on Empirical Software Engineering in Practice (IWESEP)*, 2018, pp. 1–6, <http://dx.doi.org/10.1109/IWESEP.2018.00009>.
- [42] T.D. LaToza, B.A. Myers, Developers ask reachability questions, in: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE '10*, Association for Computing Machinery, New York, NY, USA, 2010, pp. 185–194, <http://dx.doi.org/10.1145/1806799.1806829>.
- [43] A. Ahmad, O. Leffler, K. Sandahl, Software professionals' information needs in continuous integration and delivery, in: *Proceedings of the 36th Annual ACM Symposium on Applied Computing, SAC '21*, Association for Computing Machinery, New York, NY, USA, 2021, pp. 1513–1520, <http://dx.doi.org/10.1145/3412841.3442026>.
- [44] R.P.L. Buse, T. Zimmermann, Information needs for software development analytics, in: *2012 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 987–996, <http://dx.doi.org/10.1109/ICSE.2012.6227122>.
- [45] S. Breu, R. Premraj, J. Sillito, T. Zimmermann, Information needs in bug reports: Improving cooperation between developers and users, in: *Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work, CSCW '10*, Association for Computing Machinery, New York, NY, USA, 2010, pp. 301–310, <http://dx.doi.org/10.1145/1718918.1718973>.
- [46] H. Holz, F. Maurer, Knowledge management support for distributed agile software processes, in: *Advances in Learning Software Organizations*, Springer, Berlin, Heidelberg, 2003, pp. 60–80, http://dx.doi.org/10.1007/978-3-540-40052-3_7.
- [47] S. Ghobadi, What drives knowledge sharing in software development teams: A literature review and classification framework, *Inf. Manage.* 52 (1) (2015) 82–97, <http://dx.doi.org/10.1016/j.im.2014.10.008>.
- [48] G.-C. Roman, K.C. Cox, Program visualization: The art of mapping programs to pictures, in: *Proceedings of the 14th International Conference on Software Engineering, ICSE '92*, Association for Computing Machinery, New York, NY, USA, 1992, pp. 412–420, <http://dx.doi.org/10.1145/143062.143157>.
- [49] B.A. Price, R.M. Baecker, I.S. Small, A principled taxonomy of software visualization, *J. Vis. Lang. Comput.* 4 (3) (1993) 211–266, <http://dx.doi.org/10.1006/jvlc.1993.1015>.
- [50] O.C. Gotel, F.T. Marchese, S.J. Morris, The potential for synergy between information visualization and software engineering visualization, in: *2008 12th International Conference Information Visualisation*, 2008, pp. 547–552, <http://dx.doi.org/10.1109/TV.2008.56>.
- [51] S. Diehl, *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*, Springer Berlin, Heidelberg, 2007, <http://dx.doi.org/10.1007/978-3-540-46505-8>.
- [52] R.L. Novais, A. Torres, T.S. Mendes, M. Mendonça, N. Zazworka, Software evolution visualization: A systematic mapping study, *Inf. Softw. Technol.* 55 (11) (2013) 1860–1883, <http://dx.doi.org/10.1016/j.infsof.2013.05.008>.
- [53] A. Seriai, O. Benomar, B. Cerat, H. Sahraoui, Validation of software visualization tools: A systematic mapping study, in: *2014 Second IEEE Working Conference on Software Visualization*, 2014, pp. 60–69, <http://dx.doi.org/10.1109/VISSOFT.2014.19>.
- [54] L. Merino, M. Ghafari, C. Anslow, O. Nierstrasz, A systematic literature review of software visualization evaluation, *J. Syst. Softw.* 144 (2018) 165–180, <http://dx.doi.org/10.1016/j.jss.2018.06.027>.
- [55] L. Merino, M. Ghafari, O. Nierstrasz, Towards actionable visualization for software developers, *J. Softw.: Evol. Process* 30 (2) (2018) e1923, <http://dx.doi.org/10.1002/smr.1923>.
- [56] R. Koschke, Software visualization in software maintenance, reverse engineering, and re-engineering: A research survey, *J. Softw. Maint. Evol.* 15 (2) (2003) 87–109, <http://dx.doi.org/10.1002/smr.270>.
- [57] A. Gross, J. Doerr, What you need is what you get!: The vision of view-based requirements specifications, in: *2012 20th IEEE International Requirements Engineering Conference (RE)*, 2012, pp. 171–180, <http://dx.doi.org/10.1109/RE.2012.6345801>.
- [58] A. Gross, J. Doerr, What do software architects expect from requirements specifications? results of initial explorative studies, in: *2012 First IEEE International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks)*, 2012, pp. 41–45, <http://dx.doi.org/10.1109/TwinPeaks.2012.6344560>.
- [59] A. Hess, J. Doerr, N. Seyff, How to make use of empirical knowledge about testers' information needs, in: *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, 2017, pp. 327–330, <http://dx.doi.org/10.1109/REW.2017.63>.
- [60] A. Jedlitschka, N. Juristo, D. Rombach, Reporting experiments to satisfy professionals' information needs, *Empir. Softw. Eng.* 19 (6) (2014) 1921–1955, <http://dx.doi.org/10.1007/s10664-013-9268-6>.
- [61] T. Panas, T. Epperly, D. Quinlan, A. Saebjornsen, R. Vuduc, Communicating software architecture using a unified single-view visualization, in: *12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007)*, 2007, pp. 217–228, <http://dx.doi.org/10.1109/ICECCS.2007.20>.
- [62] A. Kaya, S. Dutre, J. Stegen, S.P. Jha, J. Denil, Stakeholder specific visualisation from heterogeneous modeling tools, in: *CEUR Workshop Proceedings, Vol. 2245*, 2018, pp. 213–222.
- [63] I. Logre, A.-M. Déry-Pinna, MDE in support of visualization systems design: A multi-staged approach tailored for multiple roles, *Proc. ACM Hum.-Comput. Interact.* 2 (EICS) (2018) <http://dx.doi.org/10.1145/3229096>.
- [64] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, *Empir. Softw. Eng.* 14 (2) (2009) 131–164, <http://dx.doi.org/10.1007/s10664-008-9102-8>.
- [65] V. Braun, V. Clarke, Using thematic analysis in psychology, *Qual. Res. Psychol.* 3 (2) (2006) 77–101, <http://dx.doi.org/10.1191/1478088706qp063oa>.
- [66] I. Etikan, S.A. Musa, R.S. Alkassim, Comparison of convenience sampling and purposive sampling, *Am. J. Theor. Appl. Stat.* 5 (1) (2016) 1–4, <http://dx.doi.org/10.11648/j.ajtas.20160501.11>.
- [67] J. Cohen, Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit, *Psychol. Bull.* 70 (4) (1968) 213–220, <http://dx.doi.org/10.1037/h0026256>.

- [68] V. Clarke, V. Braun, Thematic analysis: a practical guide, *Themat. Anal.* (2021) 1–100.
- [69] W.J. Orlikowski, J.J. Baroudi, Studying information technology in organizations: Research approaches and assumptions, *Inf. Syst. Res.* 2 (1) (1991) 1–28.
- [70] P. Rodríguez, C. Urquhart, E. Mendes, A theory of value for value-based feature selection in software engineering, *IEEE Trans. Softw. Eng.* 48 (2) (2022) 466–484, <http://dx.doi.org/10.1109/TSE.2020.2989666>.
- [71] T. Sauvola, L.E. Lwakatara, T. Karvonen, P. Kuvaja, H.H. Olsson, J. Bosch, M. Oivo, Towards customer-centric software development: A multiple-case study, in: 2015 41st Euromicro Conference on Software Engineering and Advanced Applications, 2015, pp. 9–17, <http://dx.doi.org/10.1109/SEAA.2015.63>.
- [72] P. Rodríguez, A. Haghhighatkhah, L.E. Lwakatara, S. Teppola, T. Suomalainen, J. Eskeli, T. Karvonen, P. Kuvaja, J.M. Verner, M. Oivo, Continuous deployment of software intensive products and services: A systematic mapping study, *J. Syst. Softw.* 123 (2017) 263–291, <http://dx.doi.org/10.1016/j.jss.2015.12.015>.
- [73] M. Komssi, M. Kauppinen, H. Töyhönen, L. Lehtola, A.M. Davis, Roadmapping problems in practice: value creation from the perspective of the customers, *Requir. Eng.* 20 (1) (2015) 45–69, <http://dx.doi.org/10.1007/s00766-013-0186-3>.
- [74] J.W. Creswell, D.L. Miller, Determining validity in qualitative inquiry, *Theory Into Pract.* 39 (3) (2000) 124–130, http://dx.doi.org/10.1207/s15430421tip3903_2.
- [75] V. Braun, V. Clarke, One size fits all? What counts as quality practice in (reflexive) thematic analysis? *Qual. Res. Psychol.* 18 (3) (2021) 328–352, <http://dx.doi.org/10.1080/14780887.2020.1769238>.