2014 Second IEEE Working Conference on Software Visualization

Information Visualization for Agile Software Development Teams

Julia Paredes
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada
Email: jparedes@ucalgary.ca

Craig Anslow
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada
Email: craig.anslow@ucalgary.ca

Frank Maurer
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada
Email: frank.maurer@ucalgary.ca

***Abstract*—Understanding information about software artifacts is key to successful Agile software development projects; however, sharing information about artifacts is difficult to achieve amongst team members. There are many information visualization techniques used to help address the difficulties of knowledge sharing, but it is not clear what is the most effective technique. This paper presents the results of a systematic mapping study of existing literature on information visualization techniques used by Agile software development teams. The results of the systematic mapping show that Agile teams use visualization techniques for designing, developing, communicating, and tracking progress. Our findings show that visualization techniques help Agile teams increase knowledge sharing and raise awareness about software artifacts amongst team members.**

***Keywords*—Agile software development, information visualization, software visualization, information radiator, big visible charts, knowledge sharing, systematic mapping**

I. Introduction

Information visualization is the use of visual representations of abstract data to amplify cognition [5]. Effective visualizations help observers to quickly identify relevant information, and help people to understand the relationships of what they are seeing [17].

Software visualization is considered part of information visualization [7]. Software visualization is used to make the structure, behaviour and evolution of software, such as code organization, software state, and bugs, more understandable [7]. Software visualization is highly relevant for developers who seek to improve software quality and has been used by developers to raise awareness about software artifacts [62].

Agile software development is an iterative alternative approach to traditional sequential management such as waterfall methods [6]. Agile methodologies are concerned with iteratively delivering high quality software based on feedback gathered from customer collaboration [6]. In Agile teams, the changing nature of Agile projects demands that they are aware of changing requirements, the user stories that they need to complete and the work of other teammates [6][40]. Since projects are progressively implemented in small iterations, Agile teams need to have all the information required to effectively respond to changes. It is a challenging task to make everyone in the team aware of the big picture of the project and the goals that need to be met in a particular timeframe.

For this reason, Agile teams have used different visualizations techniques to understand the scope of the project and keep track of the work that is being done. Agile teams devote substantial time to build informative workspaces [34]. Informative workspaces are workspaces that encourage communication and aid knowledge sharing by broadcasting information to team members [16]. Informative workspaces display information about the work in progress, work completed and the goals for the future [6]. These workspaces need to convey all the relevant information for the team. However, there isn't a single way of building and sharing information in Agile teams. The difficulties in knowledge sharing vary depending on different aspects of the project such as team size, distribution of the teams, and the nature of the project. Picking the right visualization technique to share information is important to deliver relevant information to team members.

The goal of this paper is to explore and summarize information visualization techniques used during the design and development steps of the software cycle, as well as to highlight the importance of building informative workspaces for effective knowledge sharing and communication between Agile teams. This paper presents the results of a systematic mapping of existing literature on information visualization techniques used by Agile teams during the development cycle.

II. Related Work

Jimenez et al. [8] conducted a literature review on the challenges of distributed Agile software development and identified group awareness as one of the major challenges faced by Agile distributed teams. They synthesize different visualization techniques used during the development cycle to communicate to managers which developers are involved on certain tasks, as well as to allow developers to know when code changes and who is responsible for the changes. There is no mention on how distributed teams can benefit on having informative workspaces.

Ahmad et al. [1] conducted a literature review and discuss how Kanban methodologies are used to visualize bottlenecks in the workflow and work in progress. They found that visualizing work and limiting work in progress helps in task management and improves communication amongst team members.

Manzoor and Shahzad [12] conducted a systematic review and an industry survey on information visualization for Agile organizations. The challenges of large organizations that use

978-0-7695-5305-4/14 $31.00 © 2014 IEEE
DOI 10.1109/VISSOFT.2014.32

Agile or Lean practices were identified and information visualization techniques that help overcome these challenges were found. Using a systematic review, information visualization techniques in Agile were classified into six different categories based on purpose: progress tracking and work state, business modelling, data modelling, requirement analysis and traceability, software configuration management, and coordination among teams and project stakeholders. An industry survey, which involved 50 Agile software developers, was conducted to identify the visualizations used in industry. Manzoor and Shahzad [12] proposed visualization techniques to solve existing challenges, but the techniques were not tested in industry and there was not a clear understanding on how Agile teams incorporate these techniques in their workspace.

Storey et al. [62] conducted a survey on software visualization tools to support awareness of developers in software teams. They found that repositories, bug tracking systems, and the code are used to create visualizations to explore artifacts in software development. They claim the main issue with current software visualization tools is that there aren't empirical studies to identify required features to raise awareness, and there is little research on how to evaluate software visualization tools. Through a systematic mapping on software evolution visualizations, Novais et. al [11] reached similar conclusions. They found that software visualization tools are rarely evaluated with a goal in mind and in a real setting.

To understand the importance and the perspective of researchers on software visualization, Koschke [10] conducted a survey, which involved 82 researchers, on software visualization in software maintenance, reverse engineering, and reengineering. The value of visualizations during development and maintenance is that visualizations suggest areas of exploration when developers may not be aware of issues. This is especially important for code understanding. To visualize artifacts such as code, dependencies, complexity and classes, researchers frequently make use of graphs and automatically generated UML diagrams.

III. Methodology

Systematic maps allow reviewing and classifying existing research about a field of interest [15]. We are unaware of any systematic mappings related to information visualization for Agile software development. The study in this paper was conducted following the guidelines outlined by Petersen et al. [15] for conducting systematic mappings in software engineering. This section contains details on the process we followed for each step of the systematic mapping.

A. Research Questions

The goal of this systematic mapping is to explore how information visualization is used for software development in Agile teams based on existing literature. The research questions that guided this systematic mapping are:

Q1: What information visualization techniques are used in Agile software development?

Q2: What information visualization techniques raise awareness of artifacts in Agile software development teams?

Q3: What information visualization techniques raise awareness of artifacts in software development teams?

The first two research questions focus on information visualization techniques in Agile software development teams. The third question explores information or software visualization tools that have proved to help raise awareness of artifacts in software teams. Exploration of these tools would allow us to discuss if Agile teams are missing important tools that could help them to satisfy their information needs about a project.

B. Conduct Search

After defining the research questions, a search for publications relevant to the proposed research questions was conducted. The sources selected for the search were IEEE, ACM Digital Library, Elsevier, Google Scholar, and Microsoft Academic Search. In addition, a manual search was conducted on leading Agile, software engineering, software visualization, information visualizations, and HCI conferences. The papers included in this systematic mapping come from a variety of Agile, information visualization, and software engineering publications such as: Agile, XP, CHI, ITS, VISSOFT, SoftVis, InfoVis, and ICSE.

The query for the systematic mapping was created by combining keywords from Agile methodologies and information visualization techniques and tools. Some of the keywords for visualizations were chosen after reading several blogs, such as Xprogramming[1] and SoftVis[2], by Agile developers making use of that terminology to refer to visualizations in Agile. Table I presents the keywords used to build the search queries.

TABLE I: Agile and Information Visualization Keywords

Category	Keywords
InfoVis	Information Visualization, Software Visualization, Information Radiator, Big Visible Chart, Burn down chart, Sketch, Prototype, Visualization Technique, Visualization Tool, Graph, Table, Matrix, Tree
Agile	Agile, Lean Software Development, Scrum, Extreme Programming, Continuous Integration, Agile Project, Agile Development

C. Screening papers

A total of 162 papers were collected from the selected sources based on the search terms. The next step was to discard papers that were irrelevant to the research questions. After the initial screening of the papers, an inclusion and exclusion criteria was set. To include a publication in the analysis, the study must have been in English, available online, and must have been relevant to the questions identified. For most papers, the title was sufficient to know if the paper related somehow to the research questions. However, if the title was unclear, a view of the abstract and the conclusion determined if the paper should be included. Papers were excluded if they didn't meet the inclusion criteria or if they were tutorials or demos.

[1]http://xprogramming.com
[2]http://softvis.wordpress.com/

Papers were also excluded if the word 'Agile' didn't refer to the Agile software methodology. After applying the inclusion and exclusion criteria, a total of 50 papers were reviewed as part of the data extraction step. The references section contains the list of 50 papers in the systematic mapping.

D. Keywording and Classification

As suggested by Petersen et al. [15], the abstracts of the chosen papers were read in order to find keywords about the contributions, topic and the context of the paper. Both title and abstract served the purpose of finding keywords to get a high level understanding of the topic. The papers were initially classified by type of paper (e.g. technique, method, experience report, empirical study, application, tool), topic of the paper (by analyzing the keywords of the paper) and problem domain (the problem the paper tries to address). Using the topic of the paper and the problem domain, it was found that information visualization helped software developers in four different categories: design, development, communication, and progress tracking.

IV. Findings

This section provides information about the findings of the systematic mapping based on the categories (design, development, communication and progress tracking) identified during the keywording and classifications steps. Table II summarizes the findings based on the categories.

A. Design

In terms of common design visualizations used by Agile teams, the most prolific techniques are low fidelity sketches (informal drawings that don't follow a convention) and input diagrams (formal drawings that follow a convention) [31, 66]. Developers make use of sketches and input diagrams to understand design decisions, to design interfaces, and to communicate information to others [31, 66]. Understanding is important when developers try to make sense of a code snippet. It can be beneficial when training a new developer or when a developer needs to work with a colleague's code [31]. Rough sketches (see Figure 1) are used in that situation to frame and try to understand a problem or some obscure architecture. Since sketches are also used to develop the UI and to brainstorm about the right architecture for the project, developers often collaborate to develop those sketches, and they save these visualizations (i.e by taking pictures or photocopies and uploading them to a wiki or saving into a hard drive) because it serves as documentation [31, 66].

Sketches and diagrams have a life cycle that goes well beyond their creation [31, 66]. By understanding the purpose of these visualizations one can comprehend the importance of conserving and sharing these artifacts. It is clear that these visualizations are primary sources for UI design and that they also serve as a supportive role in the design and development cycle. Another important aspect of these visualizations is that developers would choose different degrees of formalities to visualize the data depending on the context.

According to Cherubiny et al. [31] and Walny et al. [66] formal diagrams, such as UML diagrams, are only used in more formal settings, to explain and share information with

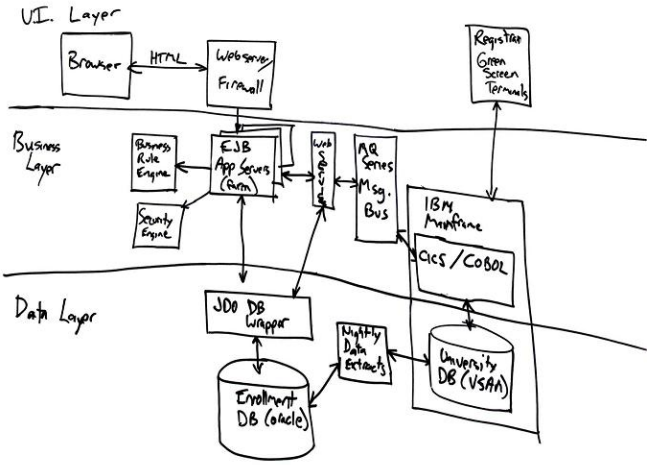

Fig. 1: An Informal Technology Diagram [2].

customers and other departments who might be concerned with the project. Petre [53] conducted an empirical study, which involved 50 software developers, and found that 70% of developers don't use UML diagrams, and if they do use UML diagrams it is usually in the very early stages when exploration of problems and design decisions are being discussed.

Informal visualizations are often used to explore and review the code, explain architectures, and refactoring. Although most developers don't use formal diagrams when drawing, they come out with their own comprehensive visualizations to understand their data [31, 66]. In this manner, these rough sketches are in a way more relevant than other formal forms. The major problem is that sketches and diagrams loose much of their context once you take them out of a meeting or conversation [53, 66].

Major problems identified with Agile development and design practices are usability issues [42, 60]. Agile teams focus on delivering working software [6]; however, this might not be of the best quality in terms of usability if teams don't integrate usability practices into the project [38, 60]. To overcome this issue, several approaches have been proposed. Most solutions try to minimize the cost and time involved in creating and evaluating medium and high fidelity prototypes for usability testing by incorporating usability from the very beginning of the project. The key idea is to start designing the user interface in a really short sprint before development starts and involve developers in design practices together with the user experience team [65]. In this manner, the user experience team can get quick feedback from customers, developers and stakeholders, and it really allows the development team to be clear on what's important to the customer.

Ferreira et al. [36] interviewed Agile teams who used user-centered design (UCD) practices with Agile methodologies and found that Agile facilitates usability testing on working software, and makes possible to catch and fix usability issues in later iterations. In another study, Ferreira et al. [37] identify that mutual awareness of responsibilities, deadlines, and how work is done affects other parts of the software which is key to integrate usability practices.

Visualizations are used to determine problems in the UI [42]. ActiveStory [42] includes the development of low fidelity prototypes with UI testing practices. This tool allows conducting remote usability testing based on low-fidelity prototypes by gathering information about the time spent on a page, the clicks, and the feedback returned by the end users. This remote testing makes usability more achievable because it is less obtrusive to the Agile development team. Feedback from usability tests is presented to developers as visualizations that summarize the data collected [42].

To understand customer needs, a common practice is to create visualizations of personas [47]. Personas are descriptions of fictional characters who are created to represent different types of users [13]. The purpose of creating personas is to highlight relevant information such as attitudes, context, and main goals of the persona, to focus developers on what is important to the customer when making design decisions [13]. Moss [47] argues that displaying personas pictures, names, and relevant information as a big visible chart helps the team make design decisions and prioritize usability issues when they arise.

The development cycle of an Agile team starts with design, which is often referred as iteration 0 [65]. This is a time when Agile teams are concerned with the system environment, software architecture, user interface, and everything that is relevant to efficiently start with development on subsequent iterations. Ungar [65] suggests that designing software is a collaborative activity; however, current research is not clear on what kind of environment (in terms of tool support and setting) is used by Agile teams.

For this reason, some researchers argued that software development, in general, would benefit from embracing a design studio environment where communication and feedback can be fostered [39, 51, 65]. A design studio would allow developers to be more engaged in usability practices.

Geyer and Reiterer [39] explore multi-surface environments (MSE), which incorporate a variety of devices such as large displays, tablets and laptops into a single interactive space, to allow teams to share, criticize, save, and iterate over different designs. MSE are typically used for collaboration, so in design practices it helps UI designers to communicate and share information to ultimately get a user interface that will satisfy the customer. Nonetheless, expected changes in requirements would also mean changes in the UI requirements, so it is not clear how Agile teams would integrate those practices over subsequent iterations.

B. *Development*

Most visualizations used during development are created to increase awareness, help teams coordinate their activities, and make feedback immediate and clear so that developers can react accordingly to changes. Visualizing aspects of the software that can be improved is also beneficial during development and for the benefit of the customer. This includes being able to visualize problems in the source code, as well as visualizing areas that can be improved by refactoring.

A development practice in Agile teams is continuous integration (CI). Since Agile teams work in small sprints, it is necessary for them to have working software everyday, and

Fig. 2: Traffic light visualization for CI [4].

CI helps to make this possible. Automated tests are setup to make sure integration is less problematic. Agile developers commit their code usually at least daily to make sure they have the latest version. Integration can be difficult and sometimes a commit could break a build. CI tools generally notify developers when something goes wrong with either the build or the tests, but feedback is not readily available and this can be an issue for software developers [28].

Physical artifacts serve the purpose of giving quick feedback to developers on integration (see Figure 2). Lava lamps and traffic lights [28, 43, 58] are often used to notify the developers of the status of the build. Both solutions use colours to represent the status. For instance, in the case of traffic lights, red is used to indicate an error, amber notifies that some tests didn't pass, and a green light means everything is fine. These physical artifacts are a way to get automated feedback about the project when integration occurs; however, teams need to make sure that they don't get feedback that would create a bad environment in the office (i.e name the person who broke the build) [43].

Although, physical artifacts have proved to be reliable and efficient in quickly delivering the status of the software, some researchers found that CI tools gather relevant information that is often hidden to interested parties like developers or even managers [28]. SQA-Mashup is used as a dashboard to visualize all the relevant information collected by different CI tools. Treemaps, timelines, and pie charts are used to see information about the test coverage, classes that haven't been well tested, number of commits, and the date bugs were introduced and fixed. The benefit of this dashboard is that managers and developers can access a summarized version of the project's data without having to go through log files to gather and understand the information [28].

Some benefits of code coverage visualization tools include improvement of code robustness, ability to see the status of automated test cases, and detection of errors [22]. Lawrance et al. [44] conducted an empirical study with 30 professional software developers to analyze the effect that visualization tools for code coverage had on developers. They found that test coverage visualizations harmed testing practices because they didn't show the big picture of the tests and this gave developers

the idea that the code was throughly tested. A positive aspect of visualizations for test coverage is that it provided developers with a standard acceptable number of tests for other classes in the project.

Other aspects like code comprehension, readability, and stability are major sources of concern for developers, especially when changing requirements sometimes leaves the team with an architecture or model that is less than ideal for the current situation. Different techniques have been used to identify code snippets that need special attention from testers, developers or quality assurance people. Heatmaps are used together with the code to identify classes that have been modified recently [61]. This allows developers to increase the coordination of their activities or even detect unstable code snippets throughout the development cycle. Companies such as Ericsson AB and Saab AB were used to test how their Agile projects could benefit from being able to visualize unstable code and it was found that the tool helps identifying risky components that can compromise software quality [61]. Although not necessarily targeted for Agile teams, it was found that researchers have made use of visualizations to easily detect code smells, code dependencies, and problems in the structure of the code [32]. It is possible to see when there are parts of the code that are very risky and error prone [32]. This kind of visualization allows developers to understand why a particular module has been changed over a period of time.

Other visualizations explored are more concerned with showing developers the context on which they are working or the evolution of contributions made by developers by analyzing project repositories [30, 48]. ProxiScientia [26] outlines the need for communication between developers and helps developers determine coordination needs as they arise. ProxiScientia [26] was developed to raise awareness of coordination needs in real time and has been evaluated in Agile projects. The value of these techniques is that they identify needs for coordination and prompt developers to talk to each other to avoid issues when changing a feature.

Information visualization techniques are also used to facilitate the coding practice of Agile teams. Most Agile teams see software development as a group activity (e.g pair programming in XP). Pair programming allows developers to comment about code and discuss design decisions, but they often have to assume certain roles when they are pair programming because typically they only have access to one input-keyboard and mouse. Hardy et al. claim that the primary tool of the developer an IDE does not support the notion of pair programming very well [41]. Therefore, they created a tool to support collaborative development using a multi-surface environment. This tool allows developers to interact with their laptops in a common workspace (an interactive coffee table), which serves as an entry point for conversations to clarify ideas. This tool uses visualization techniques to display the relevant information about the project: architecture diagrams, code snippets, input diagrams, among others. Parnin [50] also proposes the use of interactive displays as peripheral devices to expand the workspace of developers enabling them to be more productive by allowing developers to focus on artifacts when performing demanding tasks such as refactoring.

SourceVis [19] uses a multi-touch table to support collaborative activities like code review for maintenance. SourceVis supports visualizations to explore the structure and evolution of software, and has subsequently had an empirical evaluation conducted with professional developers. CodeCity [68] was also developed to evaluate the composition and evolution of software. This 3D visualization tool has been validated by empirical studies, which determined that CodeCity increased correctness and completion times over other exploration tools such as Eclipse and Excel spreadsheets. Boccuzzo and Gall [25] make use of multi touch devices as discussion platforms to support collaborative activities such as code reviews and software exploration.

C. Communication

Communication is often fostered directly and indirectly. Direct communication encompasses all the techniques used during group meetings in the Agile development cycle. Indirect communication is the shared knowledge that is transferred and shared amongst the team through big visible charts. Manzoor and Shahzad [12] found that there is a large number of visualization techniques used by teams to improve communication.

Cockburn [6] introduced the term of information radiators to describe graphical charts that teams can use to learn about a project at a glance. The use of big visible charts or information radiators is common and effective in Agile teams for knowledge sharing [43, 58].

Agile teams use their workspace to communicate information (i.e deadlines, work in progress, and goals) about the project. In Agile, workspaces have a huge impact on the outcome of the project [33, 34]. Visualizations are used to maintain the team focused on what's really important for the customer and the team [34, 57, 58]. Visualizations are aids to preserve the big picture of the project when teams are immersed in development activities (see Figure 3). The right visualizations can be of true benefit to Agile developers because they are able to see the most important information with minimum effort [33, 34, 43, 57, 58].

Most software development teams make use of their creativity to display information that is valuable to the team through the use of information radiators. These charts are highly dependent on the particular team and the project that they are working on [40, 57]. A benefit of using information radiators in the workspace is that the team needs to put little effort in creating and understanding these visualizations.

Although, there isn't a standard way to build a particular information radiator, some studies have looked on a set of heuristics to create a valuable visualization of the project in the workspace. The key aspects of information radiators are that they must be visible, valuable to developers or the customers, easy to update, easy to comprehend, and they must display information that is current and specific to an issue to avoid 'visual pollution' [33, 34]. These visualizations are valuable because they encourage communication, coordination and collaboration between team members [57].

FASTDash [24] and Awareness 2.0 [64] are digital visualization tools that help increase awareness of the status of a project and the development process. Dashboards organize and present relevant information for the developer in an easy to read manner. Dashboards are used in Agile to communicate

Fig. 3: Walle-D: An ambient display for Agile developers [14].

information about the project. The benefit of using visualization techniques on dashboards is that they improve the sense of awareness and it prompts developers to act on the conflicts that arise. Awareness 2.0 provides a high level overview using visualizations to detect bottlenecks, deadlines and tasks, which is similar to the information provided by Agile walls. On the other hand, FASTDash is more local in the sense that it allows developers to understand the context on which they are working. FASTDash makes the team more efficient because it encourages communication between team members who are working on similar areas in the project.

To easily share information in the workspace environment, tools like Impromptu [23] have been proposed to make use of ambient displays to show team members contributions relevant to the project such as design ideas, bugs, and information to be reviewed by the group. Impromptu allows developers to add, modify, and remove information that is shared on an ambient display so they can keep up to date with the latest information about the project.

Agile teams have different meetings at different points during the sprint to plan, discuss, and maybe improve the current practices. AgilePlanner [67] and MasePlanner [46] facilitate collaborative iteration planning by making use of interactive surfaces. Users can create, remove, and manipulate story cards using touch gestures on a display. The purpose of these tools is to simulate a collocated environment by providing a common workspace where people can discuss and brainstorm. AgilePlanner extends MasePlanner by allowing the integration of distributed teams in the iteration discussion. Hence, information visualization techniques are used here to improve some of the communication challenges faced by distributed teams. CodeSpace [27] uses a multi surface environment to facilitate collaboration on Scrum, retrospective and miscellaneous meetings for small, collocated teams. CodeSpace allows developers, managers, and stakeholders to transfer important information (on personal devices) to other devices, like large displays, where everyone can see the relevant information.

D. Progress Tracking

Accurate progress tracking allows managers and developers to determine if they are on or off track during an iteration of a project. To keep track of progress Agile teams commonly use Agile walls (see Figure 4) and burn down or burn up charts. A burn up chart shows the functionality implemented over a period time [29]. Burn down charts shows the work remaining to reach a specific goal over a period of time [29]. Burn down charts are considered the most valuable information radiators [52] found in Agile workspaces. Several experience reports claim that these visualizations are highly valuable for the team and are the 'must have' visualizations to understand the progress of the project and detect chances of improving the current situation [40, 58, 59].

Hajratwala found that most inefficient Agile teams are ineffective because they can't visualize the actual work accomplished [40]. Proper use of visualizations often increases the team's awareness and their overall performance [40]. The visualizations of the work accomplished are often seen in Agile walls or taskboards. Although, there isn't a particular way to build taskboards, current research suggests that at a minimum teams should be able to visualize the following categories: work in progress, backlog of user stories, and work completed [40]. These categories are represented as columns in the taskboard. Teams need to define what 'done' means for the team to avoid problems [59]. Hence, several columns can be added to visualize the different steps of a task during a sprint (e.g. testing, quality assurance, completed).

The type of taskboard used in Agile highly depends on the scope of the project and the Agile methodology used by the team. Agile teams can use scrum taskboards [6] or Kanban boards which are restricted boards where teams can see the workflow and they can limit the work in progress [54]. Other studies have shown that there's value on splitting the tasks between different components: UI, data layer, business layer, and quality among others [56]. These components are shown as part of the task-boards; the vertically sliced taskboard helps developers visualize the components missing to finish tasks that can be shown to the customer and that thus have business value for them. To visualize the big picture of the project, there are other visualizations techniques like the product release train visualization that helps visualize user stories completed for each iteration in a release, and the bugs fixed in a particular iteration [54].

Distributed Agile teams use electronic commercial solutions, such as KanbanFlow[3] and Trello[4], to keep people posted on the work done. At a minimum, these tools allow developers to create tasks, assign tasks, and change the status of tasks to keep track of progress. Some studies [40, 52, 58] discuss that electronic tools to keep track of the team progress may be ineffective because they are not as flexible and engaging as physical Agile walls. Sharp et al. [58] explain that electronic tools can only be equally effective as physical artifacts if there is a way to provide a social context around these digital tools.

Other visualizations for progress tracking are concerned with the velocity of the team and estimating efforts. Sharp et al.

[3]https://kanbanflow.com/
[4]https://trello.com/

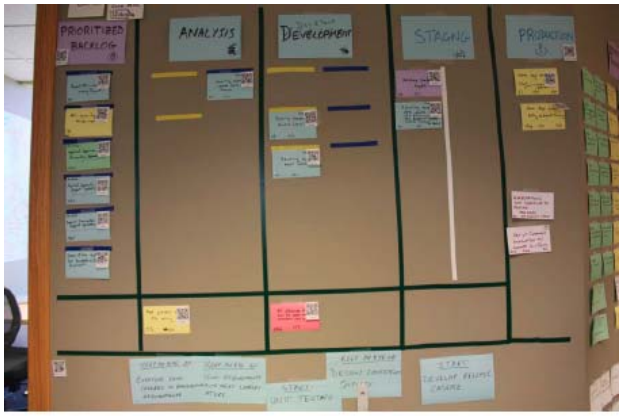

Fig. 4: Physical taskboard [40].

discuss different visualizations used to show the performance of the team for a period of time [58]. This helps developers and managers to see the progress of the team, and it helps to estimate for the future. Agile teams use burn-up or burn-down charts to graphically represent the work achieved and the work left at a certain time in an iteration [29, 54]. These charts show the ideal and actual work done over a period of time, and indicate the performance of a team.

To plan for a sprint and schedule the work properly, Agile teams need to estimate task completion time. Estimating the time to complete a task is challenging, but it is a relevant activity because this determines the user stories that will be targeted in an iteration or release. Raith et al. [55] propose a prototype to estimate task completion in Agile projects that uses the 'Planning Poker' technique. This prototype uses a multi surface environment to allow collaboration to estimate tasks time. Developers can set a time estimate using cards, and a moderator can control the discussion.

V. Discussion

The results of this systematic mapping suggest that visualization techniques are valuable in Agile software development because they lead to understanding, collaboration, and self-organization. Visualization techniques help Agile teams to increase knowledge sharing when designing, developing, communicating, and tracking progress.

A. Design

Sketches and diagrams encourage discussion and brainstorming of ideas between software developers [66]. These visualizations are used as discussion tools to explain and understand artifacts like the architecture, design choices, and the code. Teams use different fidelities for drawings (low, medium, high and computer generated) to represent software artifacts such as architecture and use cases. It was found that most valuable drawings are low fidelity because they are disposable and easily generated. Agile teams include whiteboards in their workspace [6] to make sure developers can share knowledge with other team members through sketches and diagrams. Sketches are widely used when designing an interface because they allow teams to catch errors in design or usability issues with little effort. In this manner, sketches are important to share knowledge about the end users. Although there's real value for developers who use informal sketches, it doesn't seem like formal diagrams that are high fidelity or follow a rigid convention, such as UML diagrams, provide any value that surpasses the effort made to create them [53]. Booch [3] proposes without any empirical evidence the reason for which most developers don't draw diagrams is because the diagrams are usually not useful to advance the work of developers.

B. Development

Teams are interested in knowing the code that changed the state of a program, the behaviour of software, and understanding the implications of the code structure [9]; however, this information is difficult to find. For this reason, visualization techniques have been used to find and present information about software artifacts to team members. Software visualization tools allow developers to locate areas where the code can improve, places to refactor, and areas that have not been well tested. Software visualization can raise awareness of potentially problematic areas and allow developers to discover and explore refactoring opportunities. Tools making use of multi-touch devices have also been proposed to enable collaboration and encourage discussion between team members and thus increase knowledge sharing. Agile teams can benefit from adopting software visualization techniques to acquire their information needs during development. In practice the only visualizations we found that are frequently used for development are physical devices that provide automated feedback when using CI. Other solutions like SQA-Mashup [28] seem to be more comprehensive in the information they can collect.

C. Communication

The most effective visualizations for communication in Agile teams highly depends on the project, the environment setup, and the team itself. The results suggest that most teams need to adapt their taskboards and other common information radiators, so that the team can make sense of the information that is being conveyed. What works for one team, may not work for other teams. This implies that teams need to know how to easily adapt information radiators to their information needs.

Knowledge sharing is not the only benefit of information visualization in Agile workspaces. Information radiators are built around a social context, and they engage team members to see, update, and discuss the visualizations. Ko et al. [9] imply that collocated workers are also information sources. Hence, there is true value that co-located teams can enjoy when using the workspace to display information, but this has implications for distributed Agile teams. The findings of the systematic mapping show that digital versions of common artifacts (i.e taskboards) are not as effective because they are not permanently available, and they are easy to ignore and more difficult to consult [43, 58].

Dashboards are highly valuable for distributed teams because they allow all interested parties to know the big picture of the project even though the whole team is not in the same place. They are helpful to indicate when needs for

TABLE II: Summary of findings from the systematic mapping. Presented by Category, Purpose, Reference of Paper, Domain of Paper, Type of Study, where Participants came from, and number of Participants in the study. Legend: Empirical Study (Emp), Ethnographic study (Eth), Experience Report (Exp), Grounded Theory (Gt), Interviews (I), Literature Review (Lit), None (N), Observational study (Ob), Software Engineering (SE), Students (St), Survey (S), Unknown (Unk), User study (US).

Category	Purpose	Paper	Domain	Type of study	Participants	# Participants
Design	Understanding overall system	31	SE	I & S	SE	69
		35	SE	N	N	N
		36	SE	Gt	SE	Unk
		37	SE	Eth	SE	4 Teams
		38	SE	Gt	SE	10
		39	HCI	N	N	N
		42	SE	N	N	N
		47	SE	Exp	N	N
		51	SE	Exp	N	N
		53	SE	Emp	SE	50
		60	SE	Lit	N	N
		65	SE	Exp	N	N
		66	Vis	I & S	St	8
Development	Overviews	19	Vis	US	St	6
		20	SE	Emp	Project	16
		21	SE	Case Study	Project	3
		22	SE	Exp	N	N
		25	SE	Tool	SE	2
		26	SE	US	SE	6
		28	SE	US	SE	16
		30	SE	N	N	N
		32	SE	US	SE	5
		41	SE	US	SE	6
		44	SE	Emp	SE	30
		45	SE	Case Study	SE	5 Projects
		48	SE	N	N	N
		50	Vis	N	N	N
		61	SE	Ob	SE	3 Companies
		63	SE	Lit	N	N
		68	SE	US	SE	41
Communication	Awareness and status	23	SE	Field Study	SE	13
		24	HCI	Ob	SE	6
		27	SE	US	SE	9
		33	SE	I & S	SE	8 Teams
		34	SE	Ob	SE	8 Teams
		43	SE	Exp	N	N
		46	SE	Tool	N	N
		49	SE	Case Study	SE	3 Projects
		57	SE	Ob	SE	3 Teams
		58	SE	Ob	SE	6 Teams
		62	SE	S	Tools	12
		64	SE	Emp	SE	150
		67	SE	US	SE	9
Progress Tracking	Keep track of progress	29	SE	Theory	N	N
		40	SE	Exp	N	N
		52	SE	Exp	N	N
		54	SE	Exp	N	N
		55	SE	Tool	St	21
		56	SE	Exp	N	N
		59	SE	Ob	SE	23

coordination and collaboration arise. Any kind of information can be displayed in a dashboard; however, the most common ones are digital taskboards, progress tracking information, and individual developers' workload and progress.

D. Progress Tracking

Progress tracking is essential for Agile teams. The most frequently used techniques to visualize work in Agile projects are taskboards and progress tracking charts (i.e. burn down chart or burn up chart) [40, 58, 59]. This is in line with Ko's et al. [9] findings that show developers are interested in maintaining awareness of the work done by their coworkers, dependencies amongst tasks and the information that is relevant to the tasks. Taskboards help teams to meet tight deadlines and deliver high priority user stories. It was found that there are many different variations of taskboards to keep track of projects [40]. As with any information radiator, teams need to adapt the taskboards to their needs so that is easy for team members to keep track of the big picture within an iteration.

E. Implications

The literature suggests there is a wide variety of novel software visualization tools to support practices such as code reviews, code exploration, and software maintenance. Most of these tools have been tested with small groups of developers (see Table II). Researchers have received good feedback because the tools raise awareness of artifacts required to advance software development. What is not clear is how these tools are better than what developers are currently using. This could be the reason for which adoption of software visualization tools is very low in industry [10]. Without clear supporting evidence that visualization tools can help improve development practices, developers won't make an effort to adopt these tools especially if it means they have to learn a new tool, learn how to interact with new devices, obtain expensive hardware, and maybe even change their working practices.

An alternative approach would be to design visualization tools to support software development tasks based on the needs of real software practitioners and teams. Conducting observational studies of real software practitioners to obtain requirements for visualizations tools would be an initial step to gain insight into how they currently use development tools and what information visualization techniques they use if any.

F. Limitations

A systematic mapping methodology was used to explore information visualization techniques in Agile software development. We did an exhaustive search that identified 162 papers which then yielded 50 papers as part of the data extraction step. Limitations of this study include possible bias during the keywording and classification steps. A common limitation due to the nature of these types of systematic mapping studies is that there is no guarantee that all papers were found by the search queries. These papers were predominantly academic research papers which meant we might have missed commercial visualization tools used by Agile practitioners. We did not perform any interviews or surveys with Agile practitioners as we see those steps as out of scope for this paper.

VI. Conclusion

Sharing information about artifacts is a challenging task amongst team members in Agile projects. Many information visualization techniques have been used to overcome the challenges that software development teams face when trying to share knowledge. A systematic mapping of the research literature was conducted to explore and summarize the use of information visualization techniques for designing, developing, communicating, and tracking progress. The main contribution of this paper is the systematic mapping that provides a variety of visualization techniques that help to increase knowledge sharing and awareness about artifacts. Based on the literature we found sketches and informal diagrams, CI feedback visualizations and test coverage tools, and information radiators and agile walls are the most frequently used visualization techniques in Agile software development.

In the future we would like to conduct surveys and interviews with Agile developers to identify the visualization techniques and tools that are commonly used in industry. Surveys and interviews with Agile practitioners would allow us to compare the use of visualization techniques and tools with the results of this systematic mapping.

References

[1] O. Ahmad, J. Markkula, and M. Ovio. Kanban in software development: A systematic literature review. In *Proc. of SEAA*. IEEE, 2013.

[2] S. Ambler. Architecture envisioning: An Agile best practice. http://agilemodeling.com/essays/initialArchitectureModeling.htm.

[3] G. Booch. Why don't developers draw diagrams? In *Proc of. SoftVis*, pages 3–4. ACM, 2010.

[4] R. Bradders. Cruisecontrol.net traffic lights. http://www.bradders.org/trafficlights/.

[5] S. Card, J. Mackinlay, and B. Shneiderman. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers Inc., 1999.

[6] A. Cockburn. *Agile Software Development*. Addison-Wesley, 2002.

[7] S. Diehl. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer Berlin Heidelberg, 2007.

[8] M. Jiménez, M. Piattini, and A. Vizcaíno. Challenges and improvements in distributed software development: A systematic review. *Adv. Soft. Eng.*, 2009.

[9] A. Ko, R. DeLine, and G. Venolia. Information needs in collocated software development teams. In *Proc. of ICSE*. IEEE, 2007.

[10] R. Koschke. Software visualization in software maintenance, reverse engineering, and re-engineering: A research survey. *Journal of Software Maintenance*, 15(2):87–109, March 2003.

[11] R. Lima, A. Torres, T. Souto Mendes, M. Mendona, and N. Zazworka. Software evolution visualization: A systematic mapping study. *Information and Software Technology*, 55(11):1860–1883, 2013.

[12] N. Manzoor and U. Shahzad. Information visualization for Agile development in large-scale organizations. Master's thesis, School of Computing, Blekinge Institute of Technology, September 2012.

[13] L. Nielsen. *Personas*. The Interaction Design Foundation, 2013.

[14] K. Olofsen. The ultimate wallboard. http://blogs.atlassian.com/2010/12/ultimate_wallboard_winner/.

[15] K. Petersen, R. Feldt, S. Mujtaba, and M Mattsson. Systematic mapping studies in software engineering. In *Proc. of EASE*, pages 68–77. British Computer Society, 2008.

[16] J. Shore. *The Art of Agile Development*. O'Reilly Media, Inc., 2007.

[17] C. Ware. *Information Visualization: Perception for Design (3rd Edition)*. Morgan Kaufmann, Burlington, MA, USA, 2012.

SYSTEMATIC MAPPING PAPERS

[19] C. Anslow, S. Marshall, J. Noble, and R. Biddle. SourceVis: collaborative software visualization for co-located environments. In *Proc. of VISSOFT*, pages 1–10. IEEE, 2013.

[20] F. Beck. Analysis of multi-dimensional code couplings. In *Proc. of ICSM*, pages 560–565, 2013.

[21] M. Beck, J. Trumper, and J. Dollner. A visual analysis and design tool for planning software reengineerings. In *Proc. of VISSOFT*, pages 1–8. IEEE, 2011.

[22] S. Berner, R. Weber, and R. Keller. Enhancing software testing by judicious use of code coverage information. In *Proc. of ICSE*, pages 612–620. IEEE, 2007.

[23] J. Biehl, W. Baker, B. Bailey, D. Tan, K. Inkpen, and M. Czerwinski. Impromptu: a new interaction framework for supporting collaboration in multiple display environments and its field evaluation for co-located software development. In *Proc of. CHI*, pages 939–948. ACM, 2008.

[24] J. Biehl, M. Czerwinski, G. Smith, and G. Robertson. Fastdash: A visual dashboard for fostering awareness in software teams. In *Proc. of CHI*, pages 1313–1322. ACM, 2007.

[25] S. Boccuzzo and H. Gall. Multi-touch collaboration for software exploration. In *Proc of. International Conference on Program Comprehension (ICPC)*. IEEE, 2010.

[26] A. Borici, K. Blincoe, A. Schrter, G. Valetto, and D. Damian. Proxiscientia: Toward real-time visualization of task and developer dependencies in collaborating software development teams. In *Proc. of CHASE*, pages 5–11. IEEE, 2012.

[27] A. Bragdon, Rob DeLine, K. Hinckley, and M. Ringel Morris. Code space: Touch + air gesture hybrid interactions for supporting developer meetings. In *Proc. of ITS*, pages 212–221. ACM, 2011.

[28] M. Brandtner, E. Giger, and H. Gall. Supporting continuous integration by mashing-up software quality information. In *Proc. of CSMR-WCRE*, pages 109–118. IEEE, 2014.

[29] A. Cabri and M. Griffiths. Earned value and agile reporting. In *Proc. of AGILE*, pages 6 pp.–22. IEEE, 2006.

[30] A. Caudwell. Gource: Visualizing software version control history. In *Proc. of SPLASH*, pages 73–74. ACM, 2010.

[31] M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko. Let's go to the whiteboard: How and why software developers use drawings. In *Proc. of CHI*, pages 557–566. ACM, 2007.

[32] G. de Figueiredo Carneiro, M. Silva, L. Mara, E. Figueiredo, C. Sant'Anna, A. Garcia, and M. Mendona. Identifying code smells with multiple concern views. In *Proc. of SBES*. IEEE, 2010.

[33] R. de Melo Oliveira and A. Goldman. How to build an informative workspace? an experience using data collection and feedback. In *Proc. of AGILE*, pages 143–146. IEEE, 2011.

[34] R. de Melo Oliveira, A. Goldman, and C. Melo. Designing and managing Agile informative workspaces: Discovering and exploring patterns. In *Proc. of HICSS*, pages 4790–4799. IEEE, 2013.

[35] Chuan Duan and J. Cleland-Huang. Visualization and analysis in automated trace retrieval. In *Proc. of REV*, 2006.

[36] J. Ferreira, J. Noble, and R. Biddle. Agile development iterations and UI design. In *Proc of. AGILE*, pages 50–58, 2007.

[37] J. Ferreira, H. Sharp, and H. Robinson. Agile development and user experience design integration as an ongoing achievement in practice. In *Proc. of AGILE*, pages 11–20, 2012.

[38] D. Fox, J. Sillito, and F. Maurer. Agile methods and user-centered design: How these two methodologies are being successfully integrated in industry. In *Proc. of Agile*, pages 63–72. IEEE, 2008.

[39] F. Geyer and H. Reiterer. A cross-device spatial workspace supporting artifact-mediated collaboration in interaction design. In *Extended Abstracts on CHI*, pages 3787–3792. ACM, 2010.

[40] N. Hajratwala. Task board evolution. In *Proc. of AGILE*. IEEE, 2012.

[41] J. Hardy, C. Bull, G. Kotonya, and J. Whittle. Digitally annexing desk space for software development: Nier track. In *Proc. of ICSE*, 2011.

[42] A. Hosseini-Khayat, T. Hellmann, and F. Maurer. Distributed and automated usability testing of low-fidelity prototypes. In *Proc. of AGILE*, pages 59–66. IEEE, 2010.

[43] J. Hunt, T. Hume, and D. Lozdan. On rabbits, space and cards: Moving towards an informative workspace. *Proc. of AGILE*, 0, 2007.

[44] J. Lawrance, S. Clarke, M. Burnett, and G. Rothermel. How well do professional developers test with code coverage visualizations? an empirical study. In *Proc. of VLHCC*, pages 53–60. IEEE, 2005.

[45] Mircea Lungu, Michele Lanza, Tudor Grba, and Romain Robbes. The small project observatory: Visualizing software ecosystems. *Science of Computer Programming*, 2010.

[46] R. Morgan and F. Maurer. Maseplanner: A card-based distributed planning tool for Agile teams. In *Proc. of ICGSE*. IEEE, 2006.

[47] C. Moss. Big visible testing. In *Proc. of AGILE*. IEEE, 2013.

[48] M. Ogawa and Kwan-Liu Ma. codeswarm: A design study in organic software visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):1097–1104, Nov 2009.

[49] Ciaran O'Reilly, David Bustard, and Philip Morrow. The war room command console: Shared visualizations for inclusive team coordination. In *Proc. of SoftVis*, pages 57–65. ACM, 2005.

[50] C. Parnin, C. Görg, and S. Rugaber. Codepad: Interactive spaces for maintaining concentration in programming environments. In *Proc. of SoftVis*, pages 15–24. ACM, 2010.

[51] J. Patton. Hitting the target: Adding interaction design to Agile software development. In *OOPSLA Practitioners Reports*. ACM, 2002.

[52] T. Perry. Drifting toward invisibility: The transition to the electronic task board. In *Proc. of AGILE*, pages 496–500. IEEE, 2008.

[53] M. Petre. UML in practice. In *Proc. of ICSE*, pages 722–731. IEEE, 2013.

[54] R. Polk. Agile and Kanban in coordination. In *Proc. of AGILE*, pages 263–268. IEEE, 2011.

[55] F. Raith, I. Richter, R. Lindermeier, and G. Klinker. Identification of inaccurate effort estimates in Agile software development. In *Proc. of APSEC*, pages 67–72. IEEE, 2013.

[56] I. Ratner and J. Harvey. Vertical slicing: Smaller is better. In *Proc. of AGILE*, pages 240–245. IEEE, 2011.

[57] H. Sharp and H. Robinson. Collaboration and co-ordination in mature extreme programming teams. *Int. J. Hum.-Comput. Stud.*, 66(7), 2008.

[58] H. Sharp, H. Robinson, and M. Petre. The role of physical artefacts in Agile software development: Two complementary perspectives. *Interacting with Computers*, 21(1-2), 2009.

[59] H. Sharp, H. Robinson, J. Segal, and D. Furniss. The role of story cards and the wall in XP teams: A distributed cognition perspective. In *Proc. of AGILE*, pages 65–75. IEEE, 2006.

[60] O. Sohaib and K. Khan. Integrating usability engineering and Agile software development: A literature review. In *Proc. of Computer Design and Applications (ICCDA)*, pages V2–32–V2–38. IEEE, 2010.

[61] M. Staron, J. Hansson, R. Feldt, A. Henriksson, W. Meding, S. Nilsson, and C. Hglund. Measuring and visualizing code stability – a case study at three companies. In *Proc. of IWSM-MENSURA*. IEEE, 2013.

[62] M. Storey, D. Čubranić, and D. German. On the use of visualization to support awareness of human activities in software development: A survey and a framework. In *Proc. of SoftVis*, pages 193–202. ACM, 2005.

[63] A.R. Teyseyre and M.R. Campo. An overview of 3D software visualization. *Visualization and Computer Graphics, IEEE Transactions on*, pages 87–105, 2009.

[64] C. Treude and M. Storey. Awareness 2.0: Staying aware of projects, developers and tasks using dashboards and feeds. In *Proc. of ICSE*, pages 365–374. ACM, 2010.

[65] J. Ungar. The design studio: Interface design for agile teams. In *Proc. of AGILE*, pages 519–524. IEEE, 2008.

[66] J. Walny, J. Haber, M. Dörk, J. Sillito, and S. Carpendale. Follow that sketch: Lifecycles of diagrams and sketches in software development. In *Proc. of VISSOFT*, pages 1–8. IEEE, 2011.

[67] X. Wang and F. Maurer. Tabletop agileplanner: A tabletop-based project planning tool for Agile software development teams. In *Proc. of Tabletop*, pages 121–128. IEEE, 2008.

[68] R. Wettel, M. Lanza, and R. Robbes. Software systems as cities: A controlled experiment. In *Proc. of ICSE*, pages 551–560. ACM, 2011.