the WINNOWER

COMPUTER SCIENCES

Recent trends in agile processes and software engineering research - XP 2014 conference report

DANIEL GRAZIOTIN [1]

1. Free University of Bozen-Bolzano

ABSTRACT

This report summarizes the presentations and discussions on the research activities presented at XP 2014, the 15th International Conference on Agile Processes in Software Engineering and Extreme Programming, which was held May 26-30, 2014 in Rome, Italy. XP conferences are major supporters of the agile vision of software developers, the related multidisciplinary research, and bridging industrial practitioners with academia. XP 2014 continued this trend, hosting research papers divided in the topics of agile development, agile challenges and contracting, lessons learned and agile maturity, how to evolve software engineering teaching, methods and metrics, testing and beyond, and lean development.

CORRESPONDENCE:
daniel.graziotin@unibz.it

DATE RECEIVED:
June 11, 2015

DOI:
10.15200/winn.141901.13372

ARCHIVED:
December 19, 2014

KEYWORDS:
research, author experience, agile software development, conference report, extreme programming

CITATION:
Daniel Graziotin, Recent trends in agile processes and software engineering research - XP 2014 conference report, *The Winnower* 2:e141901.13372 , 2014 , DOI: 10.15200/winn.141901.13372

INTRODUCTION

The 15th International Conference on Agile Processes in Software Engineering and Extreme Programming (XP 2014) was held May 26-30, 2013 in Rome, Italy. The conference is a forum dedicated to the exchange of research, experience, and ideas related to agile software development among academics and practitioners. This year's conference has been rich in terms of content, with six sessions running in parallel throughout the whole week. The tracks hosted several practitioner workshops, academic workshops, tutorials, demo sessions, keynotes, open space sessions, lightning talks, an executive and managers tracks, a PhD symposium, and two co-located scientific workshops.

XP 2014 was attended by 191 registered participants, with a high diversification in terms of country of provenance, as shown by Figure 1. A high participation rate of practitioners has traditionally been part of the conference. XP 2014 was no exception to this, as two thirds of the participants were from small and medium enterprises, corporations, and consultancy agencies. The present author was part of the organizing conference chairs and could attend the main conference, especially the research track. As it is an appreciated common practice to write experience reports of the attended conferences, this report provides a summary of XP 2014 research.

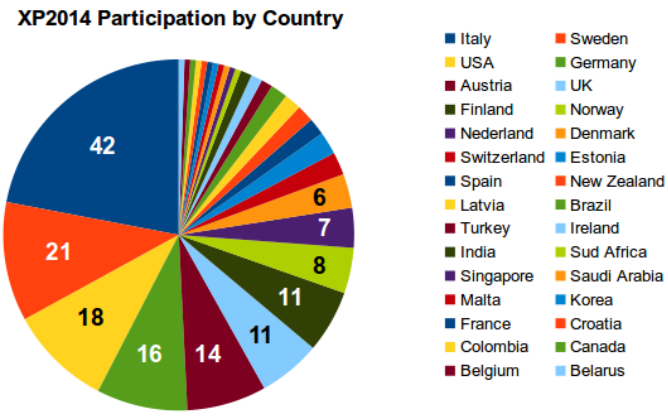

Figure 1: XP 2014 participation by country. Courtesy of xp2014 website.

This report summarizes the presentations and (where possible) the discussions that happened at the conference regarding the research-related tracks. The conference addressed both software engineering, information systems, and human-computer interaction topics, divided into the areas of agile development, agile challenges and contracting, lessons learned and agile maturity, how to evolve software engineering research, methods and metrics, testing and beyond, and lean development.

The remainder of this report is divided into three sections. In the first part, the 15 accepted regular research papers are presented, organized in the conference topics. The second part summarizes the 8 presented short papers. The final part concludes the report and provides a brief experience viewpoint of the present author.

REGULAR PAPERS

The XP 2014 presentations of submitted papers were divided into seven main topics, a short paper track, and experience reports. In this section, the presentations of the regular papers and the discussions are summarized.

AGILE DEVELOPMENT

The agile development section was about research on software development practices, agile development, and software engineering. It comprised of three studies. The article by Plonka, Sharp, Gregory, & Taylor (2014) focused on the integration of User Experience (UX) in the Dynamic systems development method (DSDM). The authors reported a case study conducted by the Agile Research Network, which explored what challenges faces a company when integrating UX in a DSDM project. The company was *LShift*, a hi-tech development company with a wide range portfolio of software products. The data gathered through cycles of interviews and direct observations supported the following findings on integrating UX design during Feasibility and Foundations, and Engineering. The company faced two main challenges during Feasibility and Foundations: the communication between developers and UX designers, and the level of precision in upfront design. Despite the fact that a designer was regularly involved during standup meetings, as encouraged by the literature, different designers worked on the project, and communication issues still arose. The level of precision in upfront design was a major issue. Whereas the literature and common sense do suggest that “just enough” design is enough, it was difficult for the participants to quantify the *just*. Overall, the study suggested adapting two DSDM roles, namely the project manager and the business analyst, to help the integration of UX design. The first needs to possess both technical and UX background. The second needs to bridge the customers, the developers, and the designers.

Kaisti, Mujunen, Mäkilä, Rantala, & Lehtonen (2014) presented a paper regarding the suitability of agile principles in the embedded system development. The authors noted that in the context of embedded systems, the research has flourished in terms of experience reports but lacked rigorous studies. Those reports have shown that no fundamental issues exist on why embedded development

could not benefit from agile. The authors presented the characteristics of embedded systems development and mapped the principles of agile software development to embedded systems development, together with the related challenges. For example, agile welcomes changing requirements in any development stage. However, the cost of the change in embedded systems is high, especially in late development. Finally, the authors issued twelve revised proposals of agile principles in the context of embedded systems. Examples are: the highest priority is to satisfy the customer through early and continuous demonstrations, which leads to a valuable system, and balance between simplicity and generality is essential, where simplicity is maximizing the amount of work not done in a short term and generality is minimizing the total amount of work to be done in a long term.

Doyle, Williams, Cohn, & Rubin (2014) presented an analysis of 2229 questionnaires performed on the tool Comparative Agility (Williams, Rubin, & Cohn, 2010), which is for a system for comparing an organization's implementation of agility. The study provided a view of the most popular agile adoption outcomes, the agile practices, and the influence of agile principles on each other. Regarding the outcomes of agile adoption, "more productive" and "delivered functionality to users more quickly" had the most frequent and positive responses. The two most negative responses were related to the "customer usability" and "customer functionality". The survey tool statements were mapped to the twelve agile principles. Overall, most agreement was shown by respondents to the statements related to individuals' motivation and business and technical co-operation. Least agreement was shown to the statements related to simplicity. The authors agreed that this was because the teams are not completely freed of bureaucratic or other non-value tasks when using agile practices. The most popular agile practices were related to individual motivation and support (e.g., "estimates are created collaboratively by the people who will do the work") and to the working software as primary measure of progress (e.g., "all work is done in iterations of no more than 30 days"). The least popular agile practice was the continuous attention to technical excellence and good design enhances agility. The results showed that the least implemented agile-related features are pair programming and manual testing at the end of each iterations. One reason could be the large presence of Web-based software companies in the sample.

AGILE CHALLENGES AND CONTRACTING

The Agile Challenges and Contracting section collected studies about the challenges and issues in adopting and adapting agile in companies. Three studies were accepted for this theme. Sekitoleko et al. (2014) focused on the technical dependency challenges in large-scale agile projects and cross-functional teams. Cross-functional teams were defined as those self-organized teams possessing all core competences for developing a feature. Technical dependencies were defined as the relationships and interactions between artifacts and teams during the development. The authors conducted a qualitative case study at Ericsson AB to identify those challenges and they likelihood of a challenge to occur. They interviewed nine software developers selected from among 300 software engineers. The qualitative data was analyzed with the thematic analysis approach (Guest, MacQueen, & Namey, 2012). The analysis showed five main challenges associated with technical dependencies between teams in large-scale agile development. The challenges, provided in order of likelihood (most frequent is first) were planning, task prioritization, knowledge sharing, code quality, and integration. The planning challenge was mostly related to the uncertainty born by non-technical managers, who split the tasks among too many teams. The task prioritization challenge mostly arose because of bad planning. For example, interviewers complained about implementing components not assigned to their backlog or implementing a component earlier because of an unexpected dependency from another team. Knowledge sharing challenge mostly occurred during meetings. The participants often did not have the opportunity to express their opinions during meetings because of the too many persons involved. Also, some team members refused to share knowledge with each other because of laziness, protectiveness, and lack of communicativeness. The code quality challenge was related to the too many people performing code changes in the same chunks of code, which resulted in too many conflicts in other teams. These code changes made it difficult to maintain a stable version of the code. The function

testers shared the opinion that such changes caused troubles to testing, as well, because of the need to rewrite and re-adapt the tests for too many times. The integration challenge was related to the creation of large branches of code, which could stay independent and unaware of the main branch for several months. At the time of integration, this caused issues. The recommendation of the authors for mitigating such challenges was to start from the knowledge sharing.

Heeager (2014) asked herself how could agile and documentation-driven methods be meshed in practice. The author argued that documentation-driven, traditional methods have the advantage to carry documentation. Therefore, they are better tailored for meeting quality standards and welcome predictability and repeatability. Heeager proposed to mesh (i.e., to become entangled) the two concepts, by implementing both agile and documentation-driven elements in a software practice. For example, the meshed area of management style mediates the two apparently incompatible elements of self-managing teams in agile and the control by management in documentation-driven methods. In total, nine practice areas were identified, namely management style, customer relations, people issues, documentation, requirements, development strategy, communication and knowledge sharing, testing, and culture. The author conducted two complimentary interpretative case studies. One case was a small, agile-driven company trying to mesh by adopting a safety-critical quality standard for medical device. The second case was a large pharmaceutical organization trying to mesh by implementing Scrum into their FDA compliant practice. In total, the author performed 36 interviews, observations, and document studies. The results of the analysis showed that (1) meshing the practice area of customer-relations is difficult. An agile customer relation can hinder the implementation of the documentation-driven customer strategy because of the high trust between the parties. On the other hand, implementing agile customer relations can be difficult in a documentation-driven project if the managers do not prioritize the relation. (2) The practice area of documentation is the hardest to mesh, as the amount of documentation required by documentation-driven methods is not supported by the agile methods. (3) The practice area of requirements is also difficult to mesh, mostly because of the documentation needed. (4) Agile and documentation-driven methods related to development strategy can be meshed by using short iterations within long-term milestones. (5) Communication and knowledge sharing can be meshed if the documentation-driven codification strategy becomes prevalent. (6) The practice area of testing is difficult to be meshed as test-driven development was unappreciated by both developers and managers.

Zijdemans & Stettina (2014) presented a preliminary framework to help understanding and choosing contracting practices in agile software projects. The authors reviewed the existing contract types in agile and traditional project management, namely fixed-price, target price (risk of overrun shared between customer and supplier), time & material (hourly rate), and cost-plus (customer pays suppliers costs and a fee for the profit) contracts. In the agile literature, the authors could find agile fixed-price contracts (fixed-price ceiling and variable scope for the software system, with several exit points), and the agile multi-phase variable-model framework. The authors conducted a scientific workshop at a conference with 8 participants, and semi-structured interviews with 5 participants. The concrete practices in use by the participants were the following. Fixed-price was mentioned several times, and the most important challenges involved were related to the customer desire for certainty and upfront specification. Payment per sprint was discovered, where in the contract a certain amount of sprints is stated and additional sprints can be purchased afterwards. Participants had mixed feelings about payment per sprint. The benefits lie in the fast payment (usually after the acceptance of a sprint), risk-reduction, and added flexibility. A disadvantage could be in the setup of too short-term vision. This might tempt developers to put the quality of the system at risk by creating a technical debt. In the agile collaboration agreement contracts, which were suggested by a participant to this experiment, changes in the scope of projects are allowed. Joint ventures are created and the software supplier gets a share of the profit. Under these agreements, the customer is supposed to watch the progress very frequently. The participants often mentioned the early termination. Early termination is an agreement that in the case when the customer wishes to terminate the project earlier, the customer has to pay a percentage of the remaining budget of the supplier. The participants did not see the early termination agreement very often. In order to handle time and money aspect of agile contracting, a participant recommended

the so-called "two-phase contract", in order to cope with the initial high uncertainty of software projects. During the first phase a fixed-price is recommended, because it is a relatively short phase with easier estimation. For the second phase, any contract type can be arranged.

LESSONS LEARNED AND AGILE MATURITY

XP 2014 provided a section dedicated to the lessons learned in agile software development research and the determination of maturity in agile companies. Only two studies were accepted in this topic. Fontana, Sheila, & Malucelli (2014) discussed the maturity in agile software development. Maturity was seen under two main approaches. On the one hand, traditional software process improvement methods have been combined with agile methods and prescriptive process definition, like the Capability Maturity Model Integration (CMMI). On the other hand, agilists defined maturity focusing on agile practices and agile values. The authors reported that the second approach could not find much support from research, as very few studies exist. The authors conducted a survey in an agile conference in Brazil. The participants expressed their opinions on how a roadmap would be for an agile team to get mature. The practices employed by different authors to assess the agility were translated to the empirical domain. Subsequently, the practices were grouped as issues. For example, "software requirements" was called "focus on agile requirements". The authors recruited 87 participants. The results of the survey suggest that maturing agile is not about following a predefined path to maturity. A model would not be useful because the organizations are too different in terms of context. Therefore, there is the need for a guide to maturity instead of a model. The authors suggested focusing on agile values, involving the customer, agile planning, and agile requirements, as these are the essential practices that may provide maturity. The intermediate practices were agile testing and agile coding. Those optional were the metrics, defining processes, and controlling processes.

Liskin, Pham, Kiesling, & Schneider (2014) argued that the quality of user stories impacts communication and coordination in a project. They focused on the regularity of user stories because the granularity can heavily impact the quality of user stories. Although granularity has many facets, the researchers focused on granularity only in the sense of scope sizes. They defined the expected implementation duration of a user story as a concept to quantify its scope. The expected implementation duration (EID) is the estimated time in days that the developer or pair would need to implement a user story. EID ignores those tasks that are not related to use a story but are still done in between its implementation. The authors conducted a two-staged study using online questionnaires. They recruited 53 participants. The results showed that EID is a measurable concept. More than half of the participants stated that more than 30% of user stories take four days or more. However, issues arise when dealing with short user stories because they are perceived as more tangible and predictable.

HOW TO EVOLVE SOFTWARE ENGINEERING TEACHING

A single study was accepted for the theme dedicated to software engineering in education. Wang, Lunesu, Rikkila, Matta, & Abrahamsson (2014) presented the experience and the lessons learned about self-organized learning in Software Factory (Fagerholm, Oza, & Munch, 2013). They argued that a serious challenge faced by software engineering education is that software engineering is difficult to be taught exclusively in a classroom, as software engineering is a competence and not just a body of knowledge. The researchers presented the Software Factory as a shared educational platform for universities where students are engaged in a real-world software project. The Software Factory relies on self-organization as its primary way of organizing the work. This means that no formal lectures are held. Self-organized learning is different from self-teaching. Self-organized learning is peer teaching, where the students teach themselves. The paper reported a multiple case study, where each case was a running course of the Software Factory in the universities of Bolzano and Cagliari. The interviews showed that the Software Factory is an environment for developing real software, whereas with traditional software engineering courses there is too much theory and the application of the skills never happens. The students determined the learning goals, and the learning outcomes were achieved through pursuing self decided learning goals. Peer teaching brought up a relaxing learning experience.

It contributed to the growth of both experienced and inexperienced students. The students with no experience learned to make something, while the expert students consolidated their ideas and learned to mentor. The diversity was one of the key factors associated with the self-organized learning experience. The diversity in terms of competences was evident among the students and fostered collaboration for finding more efficient solutions that were difficult for each single person to find out. Diversity in terms of the disciplines was another factor that contributed to a positive learning. The students came from computer science, design, management, and linguistics. This diversity enlarged the pool of competences that the factories could offer. Additionally, personal attitude and motivation matter in order to make self-organized learning happen. Both sides provided a minimal infrastructure, and the students appreciated this minimalism. From the authors' experience, it appears that there is not the need to have a big budget for Software Factory to happen. Lastly, the majority of the students reported to have never missed traditional, frontal lectures.

METHODS AND METRICS

The more traditional topic of methods and metrics collected empirical studies dealing with software and process measurements. Two studies were presented in this part of the conference.

Verdugo, Rodríguez, & Piattini (2014) shared an experience report about the development of the first laboratory in the world to be accredited as meeting ISO/IEC 17025, the standard for the general requirements for laboratories to carry out tests competently, for software for the quality evaluation. In short, they adapted Scrum for the implementation of the lab, and they developed a relational framework and a process for the quality.

Destefanis, Counsell, Concas, & Tonelli (2014) presented a software metrics analysis of eight object-oriented systems. In particular, they were interested in recognizing the use of agile methodologies through the analysis of software metrics, the differences between the metrics distribution of agile projects and traditional projects. For each of the eight projects (five developed using agile methodologies, three developed using plan – driven methodologies) the researchers computed ten metrics on each node of the software graph of each project. The results of the study showed that it is not possible to recognize the use of agile methodologies through software metrics, because the distribution of the metrics was almost equal in all cases. Therefore, the use of auto methodologies and practices does not influence the distribution of metrics in the classes. Finally, the study found that it is not possible to assert that metrics distribution generated from agile methodologies is related to buy the quality of software.

TESTING AND BEYOND

At XP 2014, there were two studies dealing with testing in agile development.Nilsson, Bosch, & Berger (2014) argued that most companies struggle with arranging the highly complex and interconnected testing activities. This struggle is especially strong for companies adopting continuous delivery of software. The researchers conducted a multiple case study involving five software development sites, coming from large software companies. The research question of the study was "How can we visualize end-to-end testing activities in order to support the transformation towards continuous integration?" The data collection was conducted through group interviews, workshops, and email correspondence. Each interviewed group comprised of 5 to 6 people, and lasted about two hours. The results showed several challenges regarding the verification and validation of the systems produced. These challenges were an end-to-end overview of testing companies, a significant duplication of testing efforts, long and slow feedback loops among the stakeholders, quality attributes tested too late in the process, and ad-hoc improvement efforts in terms of tactics. In order to cope with these challenges, the researchers developed a visualization technique called *CIViT* to show the testing activities performed around the product. *CIViT* is concerned with four types of testing, namely the new functionality, legacy functionality, quality attributes, and edge cases. *Edge cases testing* is concerned with testing unlikely or weird situations, which were discovered after significant investigative effort. The motivation of the *CIViT* model was performed with respect to the previously defined challenges. The validation

confirmed that the model serves as a solution to the lack of a holistic end-to-end understanding of the testing activities and their periodicity. The model also enabled the organization's to identify how to change their testing activities and what were the implications of such changes.

Shah, Alvi, Gencel, & Petersen (2014) conducted an experimental study with practitioners in order to compare a hybrid testing process with scripted and exploratory testing. The authors reported that traditional scripted testing is a very common process. However, it has been claimed that the use of rigorous scripted testing is not common. The main reason is that the recommendation of every scenario of a test is a too much time-consuming activity. Exploratory testing, on the other hand, has become popular in the agile communities. In exploratory testing, the tests are not planned and defined in advance. Instead, they are dynamically designed, executed, and modified. The authors formulated the research question of the study as "How does a hybrid testing process affect testing quality as compared to scripted testing and exploratory testing?" They designed the experiments as a one factor with more than two treatments, where the factor was and the three treatments were the three types of testing. The researchers developed a simple application, and several defects were introduced in its release versions. The participants tested the application using the three different approaches. The results of the issue detections indicate that hybrid testing is more effective in defect section then scripted testing but less effective than exploratory testing.

LEAN DEVELOPMENT

Lean was not a central theme at XP 2014. There were only two papers dealing with lean and agile development. Power & Conboy (2014) reported that the first step in creating a lean organization is learning to see and manage waste. However, it is difficult to eliminate waste from knowledge-based work. A smooth flow is difficult to be achieved for many. While waste still exists in software development, the researchers argued that it is more appropriate to focus on impediments to flow instead. One reason is that "waste" is an emotive topic in teams and organizations, which is difficult to be discussed. A focus on flow, however, arguably provides a better catalyst for continuous improvement within knowledge-related work. The researchers presented a framework of nine impediment categories coming from the literature. The nine categories of impediments to flow are (1) extra features, (2) delays, (3) handovers, (4) failure demand, (5) writing progress, (6) context switching, (7) unnecessary motion, (8) extra processes, and (9) unmet human potential. Extra features are those added features, without at valid hypothesis. Handovers happened whenever incomplete work must be handed over. Unnecessary motion refers to the movement of people, work, or knowledge that creates inefficiencies. Unmet human potential is a waste of not using people’s skills and potential.

Fagerholm & Pagels (2014) examined the theoretical structure of lean and agile values among software developers. The authors argued that while diversity in general can improve the team performance, the diversity and personal values could lead to conflict and lower productivity. While it is easy to understand that integrating culture and value into the software development process is a way to enhance the developer experience, it is difficult to understand values themselves. Therefore, it is important to understand the theoretical structure of values. The researchers employed a quantitative survey approach, investigating the structure of the lean and agile value system among software developers, the relationship between the lean and agile value system and the general human value system, and the relationship between the lean and agile value system and individual personality. The responses of 57 participants were analyzed using agglomerative hierarchical clustering and non-metric multidimensional scaling. Ten clusters emerged from the data analysis. The first cluster represents the view that software developers should focus on technical work and should not deal with work belonging to stakeholders and managers. The second cluster represents valuing flexibility in task execution. The third cluster concerned the values of planning and preparation, which is in conflict with the agile avoidance of long-term planning. The fourth cluster represents the belief of strict adherence to processes, which is preferred to having teams continuously discussing about what to do next. The fifth cluster emphasizes discipline, which mostly refer to sticking to anything, which has been previously agreed. The sixth cluster is about giving value to people. However, the cluster is more about

responsiveness and knowledge of contractual obligations then about the team well being. The seventh cluster is about self-organization and responsiveness. Developers value the freedom to organize, e.g., choosing any tool they wish to use. The eight cluster emphasizes valuing the purpose of the work of people and their roles. The ninth cluster is about a desire for uncertainty reduction. It is related to the desire to base action on evidence rather than unjustified orders. The tenth cluster reflects valuing close, collaborative work. The eleventh cluster is concerned about customer involvement. At a broader view, the cluster deals about involving all the stakeholders, e.g., all the team members should be aware on each other's work. A two dimensional scaling of the responses related to the human values and the lean and agile values clusters showed that there are two different structures. Similarities exist on a continuum regarding the type of decision-making, control, and ambition. For example, valuing individual decision-making and self-enhancements was consistent with the self-focus in universal values. Collective focus was congruent with collaborative decision-making. Finally, the results showed that there might be a weak relationship between the clusters and personality.

SHORT PAPERS

XP 2014 offered a substantial space to short papers, which were published in the conference proceedings. Eight short papers were accepted for presentation and publication.

Barabino, Grechi, Tigano, Corona, & Concas (2014) conducted a survey regarding the use of agile methodologies, tools, and languages for web programming. They recruited 112 participants, who answered 12 questions about their companies, the possible usage of agile methodologies or practices, and the technologies for web application development. The results showed that 69% of the participants use agile for web development. The strongest preference was for Scrum, followed by eXtreme programming. The three most used agile practices were openwork area, daily standup meetings, and user stories. The least employed practice was pair programming. The most frequent answers when dealing with tools were Jira, BugZilla, and none. Given the high number of participants who are not using any tools, the authors argue that there are still many hard-core agilists who prefer to use tangible artifacts rather than electronic tools. The most employed programming languages were Java, JSP, and ASP.net. Seventy percent of the participants were not certified agilists. The authors asked the participants if they use CMS, and 40% of them do not. This is not surprising, because the authors defined web applications as systems making use of Web browsers, including mobile apps. CMS typically fall only under the category of websites. Finally, the most employed database among the respondents was MySQL.

Datta et al. (2014) empirically examined Linus' Law. The law states that "given enough eyeballs, all bugs are shallow". This means that every problem in a software system is easily defined and fixed given a large enough number of people working on the system or observing it. The researchers wanted to see if there is empirical evidence supporting the claim. They studied the Android bug reports data. For each bug, they calculated the resolution time as the number of days between the bug opening and its closure. The researchers retained in the data set only those bugs, which had a resolution time of one year or less, because they assumed that bugs with longer resolution time could not receive enough developer attention. The final data set consisted of 1016 bugs and 73 developers assigned to those bugs. The researchers built the network connecting those developers and the bugs. The analysis showed that a higher *connection* (the degree of developers' connection in bugs' ownership) was related to decreased bug resolution time. A higher *betweenness* (the likelihood of a developer to be in a position to broker the interaction of others) was related to an increased resolution time. Therefore, the more a developer is connected to other developers through the ownership of similar bugs, the more the developer is likely to be more deeply embedded in the development system, which was found to facilitate resolution of the bugs owned by that developer. However, the more involved a developer is in brokering interactions between other developers, the more likely the developer might become more distracted, leading to an higher resolution time. The researchers omit in the limitations of the study that the closure date of a bug does not always or necessarily represent the date when the bug is resolved. Especially in large projects such as Android, there might be long discussions after a fix

is released before the bug can be considered fixed. Additionally, the authors did not mention how they dealt with issue entries that could be closed and opened again. Therefore, the calculation of the bug resolution time might present some issues.

Rooksby, Hunt, & Wang (2014) presented randori coding dojos. The coding dojo is a technique for continuous learning and training coming from martial arts. It was created for supporting the development of skills through repetitive practice of coding and social interaction. The randori is a dojo format, where pairs work in time-boxed rounds. A driver and a navigator compose the pairs, and they work like in pair programming. At the end of each time box, the driver moves to the audience, the navigator becomes the driver, and a third person from the audience becomes the navigator. That is, dynamic pairs perform the learning in front of an audience. The researchers wanted to explain and characterize randori dojos. They obtained a set of twelve recordings made at a randori dojo organized in the UK. The results of the analysis of the recordings showed that the running of a randori dojo is way different than how a randori dojo is described in theory. For example, the majority of the dojo was filled up with discussions between the driver and the navigator, and the pair and the audience. The chosen solutions caused the participants to discuss about pro and contra of those solutions, and many of them will often not familiar with the proposed solutions. The development target where not met at the end of the session. However, the participants evaluated the experience as a positive learning experience. The dojo rarely exemplified good practice, but there were occasions of this and several discussions.

Rejab, Nobl, & Allan (2014) dealt with locating expertise in agile software development projects. In particular, the authors argued that it is important to know who knows what in the context of software projects. They wanted to answer the question "how do agile team members depend on each other in locating expertise in their teams?" The researchers interviewed 16 agile practitioners from companies based in New Zealand and Australia. The category "locating expertise" emerged from the data analysis, describing how team members identify the relevant expertise in teams. Four ways to identify expertise were found. Frequent and effective communication can help to determine when a team member has desirable expertise. Working closely together provides opportunities for team members to identify and confirm people expertise. Self-identified expertise needs to be declared by agile team members, but the declaration must be followed by concrete impact. The last identified way to find expertise in agile teams is using an expertise directory. That is, some companies develop a skills database, in order to let people find competent team members.

Concas, Monni, Orrù, & Tonelli (2014) conducted a study on 29 releases of five software systems written in Java. In particular, the study aimed to understand if refactoring practices were related to clusters in Java software. They built networks of the examined software systems, by observing the relationships between classes. For every release of the software, the researchers tried to understand if classes are more connected with each other after undergoing a refactoring operation on not. They compared the number of clusters formed by refactored classes with the average number of clusters formed by random selection of them. The results showed that randomly selected classes are poorly coupled. Therefore, if a class needs refactoring, there is a high chance that a connected class needs refactoring, as well.

Power (2014) sees teams and organizations as complex adaptive systems. He wrote that self-organization in complex adaptive systems evolves through a set of simple rules. Power discussed how social contract theory, or how people in a social system have got rights and benefits determined by the system itself, applies to agile teams and organizations. The author argued that human systems dynamics provides a model for understanding self-organization in human existence.

Kettunen (2014) argued that agile software development teams are always striving for high performance. His paper proposed a performance analysis approach for agile software organizations. In particular, the researcher aimed to provide means to distinguish high performing software teams and measures to establish performance in organizational transformations. Although there is no universally recipe for creating and improving performance in software teams, such means stem from the processes, tools, organization, and the people in the team. Also, several negative factors limiting

software developer's performance can be turned into positive ones. In the case of software organizations, the agile transformations are the adoption of agile software development methods and principles. However, there are no models for realizing such transformations, especially for large-scale software enterprises. Kettunen proposed three dimensions regarding team-based transformation management. The first one has the company as scope and emphasis on goals. The related transformation traits are the needs and goal attainment strategies. The measures are the rating of drivers and the goals, while the instrument is the *Agility Profiler* (Kettunen, 2012). The second dimension has its scope in R&D and software development, and emphasis on the means. The transformation traits are in team diversity portfolio coordination and alignment. The measures are the team performance positioning, or key capabilities, while the instrument is the *Orientation Frame* (Kettunen, 2013). The third dimension has a scope on teams with emphasis on enablers. The transformation traits are the team performance self-management. The measures are in performance profile gauges, while the instrument is the *Capability Analyzer and Monitor* (Kettunen, 2014a). The researcher reported a case example revisited retrospectively to provide partial validation of the framework. The presented framework is holistic and only guides organizations to see their team performance.

Peters, Knieke, Krämer, & Schulze (2014) reported that there is a new trend in the automotive industry software development towards model-based development. The vehicle functions for engine console units are modules using modules of software written in ASCET, MATLAB/Simulink, Statement, and similar software. The ISO 26262 standard for road vehicles and functional safety specifies that each module has to be tested separately. However, the requirements are often documented only at the function level, while there is the need to test at the module level. The authors suggested applying a test driven approach to model-based development. The researchers provided a case study from the automotive field. The case study showed that the domain models provide valuable information for further integration of modules.

THE PRESENT AUTHOR'S EXPERIENCE

The present author reports a positive experience both as attendant of the conference and as a member of the organizing chairs. The presentations belonging to all the tracks were interesting and insightful. In particular, the present author appreciated several presentations from the executive and managers track, which offered interesting agile-related insights of important companies like Spotify and Scuderia Ferrari. However, the present author is of the opinion that, as a conference deeply related to agile, XP 2014 lacked studies dealing with the human aspects of agile software development. This is surprising, as agile software development has one of its pillar values as *individuals and interactions over processes and tools* (Williams & Cockburn, 2003). One would expect the research related to agile to adhere to the same stance. Instead, the conference was often more a general software engineering venue than one related to agile software per se. However, the present author appreciated those few studies dealing with human aspects of software development, e.g., (Fagerholm & Pagels, 2014; Rejab et al., 2014).

It has been reliving to note the high participation of practitioners, because software engineering research has the urge to become relevant in practical applications. The practitioners serving as research "commenteers" characterized this year's conference. After each research paper presentation, a 3-minute slot was reserved for a comment by a practitioner. That is, practitioners from the industry conducted a short post-publication review of the research papers. The purpose was to stimulate the exchange of viewpoints between research and practice. The practitioners were able to provide several points for improving the studies in ways that could benefit more the industry. Many future research points were suggested. Both the practitioners and the researchers appeared to be satisfied with this approach.

The present author regrets to report how little was the knowledge of open access among the participants. The majority of the participants, which was from the academic world, did not even know what open access is. Therefore, the majority of the participants was not aware of open access journals

and self-archiving practices. The participants showed lack of understanding of the copyright transfer agreements that they signed with the conference proceedings publisher and the rights that they were given when they signed that contract. Some of them lacked a basic understanding of copyright. Several of them admitted to blindly sign the copyright transfer agreement when they publish a paper. This gathered experience is in line with what reported by Graziotin (2014), where the majority of the surveyed computer scientists was not aware of green open access, and was not performing it. Of the 23 regular and short papers presented at the conference, only three have been self-archived and are freely accessible on the Web. One of these papers was self-archived after the tweet about the publication was noticed on Twitter by the present author, and self-archiving was solicited. Therefore, only about the 8% of the presented papers have been spontaneously self-archived by the authors. This is further evidence that self-archiving is not widespread among computer science authors. The absence of self-archived papers was an issue also because of the following reason. In computer science, the conferences are considered formal publication venues with a need for speed in publishing the articles. Unfortunately, the proceedings were only present in printed book format during the conference. The electronic version (PDF) of the articles was missing. The PDF version of the papers has been published more than one month after the conference end. Therefore, the presented articles were practically non-existing for the scientific community, up to more than one month after the conference end. The electronic proceedings are now available from the publisher's website, behind a paywall. The present writer hopes that the authors of future conferences will think about the availability of their own produced knowledge, and begin self-archiving their research articles. Yet, there is still a long way needed in terms of advocacy for open access in computer science.

ACKNOWLEDGEMENTS

The present author is thankful to all XP 2014 participants and organizers. This author is thankful to Elena Borgogno for her tireless help in improving this article.

REFERENCES

Barabino, G., Grechi, D., Tigano, D., Corona, E., & Concas, G. (2014). Agile Methodologies in Web Programming: a Survey. In G. Cantone & M. Marchesi (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (Vol. 179, pp. 234–241). Rome, Italy: Springer Science + Business Media. doi:10.1007/978-3-319-06862-6_16

Concas, G., Monni, C., Orrù, M., & Tonelli, R. (2014). Are Refactoring Practices Related to Clusters in Java Software? In G. Cantone & M. Marchesi (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (Vol. 179, pp. 269–276). Rome, Italy: Springer Science + Business Media. doi:10.1007/978-3-319-06862-6_20

Datta, S., Sarkar, P., Das, S., Sreshtha, S., Lade, P., & Majumder, S. (2014). How Many Eyeballs Does a Bug Need? An Empirical Validation of Linus' Law. In G. Cantone & M. Marchesi (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (Vol. 179, pp. 242–250). Rome, Italy: Springer Science + Business Media. doi:10.1007/978-3-319-06862-6_17

Destefanis, G., Counsell, S., Concas, G., & Tonelli, R. (2014). Software Metrics in Agile Software: an Empirical Study. In G. Cantone & M. Marchesi (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (Vol. 179, pp. 157–170). Rome, Italy: Springer Science + Business Media. doi:10.1007/978-3-319-06862-6_11

Doyle, M., Williams, L., Cohn, M., & Rubin, K. S. (2014). Agile Software Development in Practice. In G. Cantone & M. Marchesi (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (Vol. 179, pp. 32–45). Rome, Italy: Springer Science + Business Media. doi:10.1007/978-3-319-06862-6_3

Fagerholm, F., Oza, N., & Munch, J. (2013). A platform for teaching applied distributed software development: The ongoing journey of the Helsinki software factory. In *2013 3rd International Workshop on Collaborative Teaching of Globally Distributed Software Development (CTGDSD)* (pp. 1–5). IEEE. doi:10.1109/CTGDSD.2013.6635237

Fagerholm, F., & Pagels, M. (2014). Examining the Structure of Lean and Agile Values Among Software Developers. In G. Cantone & M. Marchesi (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (Vol. 179, pp. 218–233). Rome, Italy: Springer Science + Business Media. doi:10.1007/978-3-319-06862-6_15

Fontana, R. M., Sheila, R., & Malucelli, A. (2014). Maturing in Agile: What Is It About? In G. Cantone & M. Marchesi (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (Vol. 179, pp. 94–109). Rome, Italy: Springer Science + Business Media. doi:10.1007/978-3-319-06862-6_7

Graziotin, D. (2014). Green open access in computer science - an exploratory study on author-based self-archiving awareness, practice, and inhibitors. *ScienceOpen Research*, *1*(1), 1–11. doi:10.14293/A2199-1006.01.SOR-COMPSCI.LZQ19.v1

Guest, G., MacQueen, K. M., & Namey, E. E. (2012). Applied Thematic Analysis. In I. SAGE Publications (Ed.), *Applied Thematic analysis* (11th ed., pp. 1–320). Thousand Oaks, California.

Heeager, L. T. (2014). How can Agile and Documentation-driven Methods be Meshed in Practice? In G. Cantone & M. Marchesi (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (Vol. 179, pp. 62–77). Rome, Italy: Springer Science + Business Media. doi:10.1007/978-3-319-06862-6_5

Kaisti, M., Mujunen, T., Mäkilä, T., Rantala, V., & Lehtonen, T. (2014). Agile Principles in the Embedded System Development. In G. Cantone & M. Marchesi (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (Vol. 179, pp. 16–31). Rome, Italy: Springer Science + Business Media. doi:10.1007/978-3-319-06862-6_2

Kettunen, P. (2012). Systematizing Software-Development Agility: Toward an Enterprise Capability Improvement Framework. *Journal of Enterprise Transformation*, *2*(2), 81–104. doi:10.1080/19488289.2012.664610

Kettunen, P. (2013). Orienting High Software Team Performance: Dimensions for Aligned Excellence. In J. Heidrich, M. Oivo, A. Jedlitschka, & M. T. Baldassarre (Eds.), *14th International Conference on Product-Focused Software Process Improvement (PROFES 2013)* (Vol. 7983, pp. 347–350). Springer Berlin Heidelberg. doi:10.1007/978-3-642-39259-7_31

Kettunen, P. (2014a). Directing High-Performing Software Teams: Proposal of a Capability-Based Assessment Instrument Approach. In *6th International Conference on Software Quality. Model-Based Approaches for Advanced Software and Systems Engineering (SWQD 2014)* (pp. 229–243). Vienna, Austria. doi:10.1007/978-3-319-03602-1_15

Kettunen, P. (2014b). Realizing Agile Software Enterprise Transformations by Team Performance Development. In G. Cantone & M. Marchesi (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (Vol. 179, pp. 285–293). Rome, Italy: Springer Science + Business Media. doi:10.1007/978-3-319-06862-6_22

Liskin, O., Pham, R., Kiesling, S., & Schneider, K. (2014). Why We Need a Granularity Concept for User Stories. In G. Cantone & M. Marchesi (Eds.), *Agile Processes in Software Engineering*

and Extreme Programming (Vol. 179, pp. 110–125). Rome, Italy: Springer Science + Business Media. doi:10.1007/978-3-319-06862-6_8

Nilsson, A., Bosch, J., & Berger, C. (2014). Visualizing Testing Activities to Support Continuous Integration: A Multiple Case Study. In G. Cantone & M. Marchesi (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (Vol. 179, pp. 171–186). Rome, Italy: Springer Science + Business Media. doi:10.1007/978-3-319-06862-6_12

Peters, H., Knieke, C., Krämer, M., & Schulze, A. (2014). A Test-driven Approach for Model-based Development of Powertrain Functions. In G. Cantone & M. Marchesi (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (Vol. 179, pp. 294–301). Rome, Italy: Springer Science + Business Media. doi:10.1007/978-3-319-06862-6_23

Plonka, L., Sharp, H., Gregory, P., & Taylor, K. (2014). UX Design in Agile: A DSDM Case Study. In G. Cantone & M. Marchesi (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (Vol. 179, pp. 1–15). Rome, Italy: Springer Science + Business Media. doi:10.1007/978-3-319-06862-6_1

Power, K. (2014). Social Contracts, Simple Rules and Self-organization: A Perspective on Agile Development. In G. Cantone & M. Marchesi (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (Vol. 179, pp. 277–284). Rome, Italy: Springer Science + Business Media. doi:10.1007/978-3-319-06862-6_21

Power, K., & Conboy, K. (2014). Impediments to Flow: Rethinking the Lean Concept of “Waste” in Modern Software Development. In G. Cantone & M. Marchesi (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (Vol. 179, pp. 203–217). Rome, Italy: Springer Science + Business Media. doi:10.1007/978-3-319-06862-6_14

Rejab, M. M., Nobl, J., & Allan, G. (2014). Locating Expertise in Agile Software Development Projects. In G. Cantone & M. Marchesi (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (Vol. 179, pp. 260–268). Rome, Italy: Springer Science + Business Media. doi:10.1007/978-3-319-06862-6_19

Rooksby, J., Hunt, J., & Wang, X. (2014). The Theory and Practice of Randori Coding Dojos. In G. Cantone & M. Marchesi (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (Vol. 179, pp. 251–259). Rome, Italy: Springer Science + Business Media. doi:10.1007/978-3-319-06862-6_18

Sekitoleko, N., Evbota, F., Knauss, E., Sandberg, A., Chaudron, M., & Olsson, H. H. (2014). Technical Dependency Challenges in Large-Scale Agile Software Development. In G. Cantone & M. Marchesi (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (Vol. 179, pp. 46–61). Rome, Italy: Springer Science + Business Media. doi:10.1007/978-3-319-06862-6_4

Shah, S. M. A., Alvi, U. S., Gencel, C., & Petersen, K. (2014). Comparing a Hybrid Testing Process with Scripted and Exploratory Testing: A Pre-Experimental Study with Practitioners. In G. Cantone & M. Marchesi (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (Vol. 179, pp. 187–202). Rome, Italy: Springer Science + Business Media. doi:10.1007/978-3-319-06862-6_13

Verdugo, J., Rodríguez, M., & Piattini, M. (2014). Using Agile Methods to Implement a Laboratory for Software Product Quality Evaluation. In G. Cantone & M. Marchesi (Eds.), *Agile*

Processes in Software Engineering and Extreme Programming (Vol. 179, pp. 143–156). Rome, Italy: Springer Science + Business Media. doi:10.1007/978-3-319-06862-6_10

Wang, X., Lunesu, I., Rikkila, J., Matta, M., & Abrahamsson, P. (2014). Self-organized Learning in Software Factory: Experiences and Lessons Learned. In G. Cantone & M. Marchesi (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (Vol. 179, pp. 126–142). Rome, Italy: Springer Science + Business Media. doi:10.1007/978-3-319-06862-6_9

Williams, L., & Cockburn, A. (2003). Agile Software Development: It's about Feedback and Change. *Computer, IEEE*, (June), 39–43. doi:10.1109/MC.2003.1204373

Williams, L., Rubin, K., & Cohn, M. (2010). Driving Process Improvement via Comparative Agility Assessment. In *2010 Agile Conference* (pp. 3–10). IEEE. doi:10.1109/AGILE.2010.12

Zijdemans, S. H., & Stettina, C. J. (2014). Contracting in Agile Software Projects: State of Art and How to Understand it. In G. Cantone & M. Marchesi (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (Vol. 179, pp. 78–93). Rome, Italy: Springer Science + Business Media. doi:10.1007/978-3-319-06862-6_6