The Social Developer: Now, Then, and Tomorrow

Terhi Kilamo, Marko Leppänen, and Tommi Mikkonen
Tampere University of Technology
Tampere, Finland
{terhi.kilamo, marko.leppanen, tommi.mikkonen}@tut.fi

ABSTRACT

The practice of software engineering needs both individual commitment as well as social interaction. It has long been widely recognized that communication problems are a major factor in the delay and failure of software projects. However, the patterns of communication that can be associated with the different development paradigms have gained less attention. In this paper, we present some views to the evolution of social dimensions in the light of software engineering methodologies and associated tools. To study this, we have surveyed a number of software developers working in industry to reflect our views into the state-of-practice in software development companies and shed light to the impact of distributed and agile development has had on developer communication. Towards the end of the paper, we provide some ideas for future research and draw some final conclusions.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*Programming teams*; D.m [**Miscellaneous**]: Software psychology

Keywords

Software engineering, development methodologies, tool support, socio-technical congruence, social developer

1. INTRODUCTION

The history is full of examples of engineering efforts that manifest the importance of cooperation. The pyramids, the Colosseum, the Moai Statues of Easter Island, the Great Wall of China, the Inca City of Machu Picchu, and the Hoover Dam are all landmarks of engineering as well as human collaboration and coordination, although the latter part is seldom addressed afterwards. However without careful coordination and management, these achievements would, as many other greatest moments of mankind, have been impossible. It has even been hypothesized that the language itself has been developed because of hominin reliance on tools and their coordinated manufacture [18]. So, the 'C's; namely communication, coordination, cooperation and collaboration, build on each other, adding more complexity and mutual trust, ultimately ending up into effective collaboration and integration.

SSE'15, September 1, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3818-9/15/09...$15.00
http://dx.doi.org/10.1145/2804381.2804388

Similarly to the above examples, the practice of software engineering needs both individual commitment as well as social interaction among the participants. When comparing how good a job software engineers do as individuals to their work as a team, the answer is obvious: It has long been widely recognized that communication problems are a major factor in the delay and failure of software projects [4]. In fact, while the software development community is already struggling with communication issues, the emerging practice of global software engineering is raising even more challenges. Software work is undertaken at geographically separated locations across national boundaries in a coordinated fashion, involving both real time (synchronous) and asynchronous interaction [25]. This emphasizes the need for timely, precise and uniform forms of communication across the planet.

As software development takes place at the global scale, the necessary communication should take place at such a scale as well. On this note, the practice of pull-based development is moving from its origin, open source software, to distributed development in general [9]. A recent study shows that the pull-based approach with support for independent development through developer specific branching is used not only to integrate code, but to do code review and, to a large extent, discuss new features [8]. Furthermore, the so-called socio-technical congruence [10], first pointed out by Conway [3], has become an important field for recent research activities (e.g.[30]). There the relation between humans, organizations, and technical artifacts is investigated.

We believe that the new ways of working in software development, culminating in Continuous Integration, Continuous Delivery, and DevOps, will have a profound effect on the fashion social cooperation and collaboration takes place in software development. In this paper, we present some views to the evolution of social dimensions and their evolution in software development in the light of the evolution software engineering methodologies and associated tools. We focus on the interaction between the stakeholders of software development. In particular, we are interested in how closely developers communicate with the customer representatives and with other developers, and how this work is partitioned between face-to-face time, documentation, and tools such as version control systems. Moreover, we also envision tools that will be needed in the future to support development activities as well as interactions between stakeholders taking part in the process. To study this, we have surveyed a number of developers working in industry to reflect our views on the state-of-practice in software development.

The rest of this paper is structured as follows. In Section 2, we discuss the different software development approaches, and in Section 3, we address the evolution of tools that support the development. In Section 4, we provide an insight into our observations regarding the social dimension in the

development. In Section 5, we provide results from our survey regarding the fashion social interaction takes place in software development. In Section 6, we give an extended discussion regarding our views and findings as well as provide some directions for future work. Finally, in Section 7, we draw some conclusions.

2. EVOLUTION OF DEVELOPMENT PRACTICES

The first formal definition for the term software engineering was introduced in the 1968 NATO software engineering conference – the very same event where the term "software crisis" was coined by some attendees [19]. Besides introducing the idea of software engineering, the attendees of the conference agreed that the design concepts essential to maintainable systems are modularity, specification, and generality [17].

The above key concepts enabled development models, where modularity was used as a basis for project executing and task allocation. In particular, Royce's Waterfall model [24], intended to be a straw man for the sake of argument, is often interpreted as meaning a method for sequentially executing specification, design coding, and testing phases, accompanied with definitions how these phases are executed to create software that satisfies user requirements. These phases are used to decompose the system into modules, with various characteristics [21]. These modules then form the basis for task division – each module can be allocated to a particular developer. Once the tasks have been allocated, the developer can implement the modules allocated to her in relative isolation, independently from other developers. This process is illustrated in Figure 1.

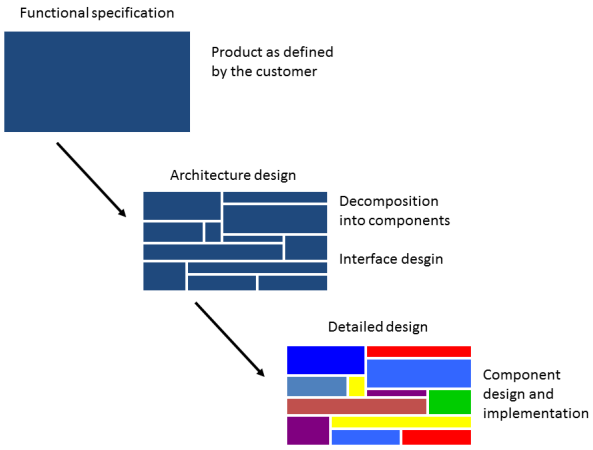

Figure 1: From requirements to architecture to components.

Although the modular structure of the software isolates the developers from each other, the isolation cannot be total. The modules interact with each other via interfaces and this interaction is reflected in the development team as communication. The implementation of interacting modules would be impossible without some degree of cooperation. Thus, the organization of the developers and their communication patterns resemble the structure of the software itself as originally noted by Conway's law [3]. Interestingly, the phenomenon Conway described, actually works the other way around; if the organization of the developers, either intentionally or unintentionally, promotes certain communication between the developers, a similar structure will be visible in the software.

Despite the known problems in the Waterfall approach – user requirements change, it is hard to get designs right on the first try, and the technologies evolve – this approach formed the baseline for software development for well over a decade. This was simply because it was considered as the best option to give structure and rationale in the development process [22].

The Waterfall model is reflected in various activities in the development practices. For example, approaches such as Capability Maturity Model (CMM) [23] can be considered as a derivative of the Waterfall model in its fundamental origins, although its later derivatives can be easily applied in various other settings. Similarly, using modularity for task allocation has led to several proposals for implementing modularity in programming languages, called modules, packages, classes, and so on. Combining modularity with generality has given us inheritance, where modularity is harnessed to support reuse without risking the possibility to use modules as the basis for task allocation. These have led to the perfection of the technical dimension of development, but they have not really assisted in supporting daily communication.

To overcome the challenges of the above mechanistic model, Agile approaches [15] advocate close and frequent communication between the client and the developers as well as among the developers themselves. While often implemented in the form of a development team sharing the same premises, encouraging frequent informal communication, distributed Agile development has also been proposed [29, 20]. In general, Agile software development, culminating in the Agile Manifesto[1], values "individuals and interactions over processes and tools", with the fundamental goal to embrace change that will be inevitably needed as development proceeds [1]. As an extension to Agile, lean software development introduced the concept of waste and proposed delivering value as soon as possible. This calls for several technical enablers, such as continuous integration [7] and smooth deployment [11]. In general, such reduction of time from completed implementation to deployment has resulted in DevOps [12] where developers and operators work as a team to deliver value to end users as quickly as possible.

3. SUPPORTING TOOLS AND OTHER INFRASTRUCTURE

Numerous tools and methods have been proposed to solve issues in different phases of projects, starting from capturing requirements and ending at customer documentation. These tools reflect how developers compose systems, and therefore they are closely related to the practices that are used.

The role of the customer in traditional software development models has been to provide information on what is needed. This is particularly visible in the Waterfall model, but applies also to some extent in the Agile setting. The customer's needs have traditionally been recorded in different systems and formats, be it requirements in a requirements document or a requirements management system, or

[1] http://agilemanifesto.org/

use cases in the product backlog in Agile development. For instance, Figure 2 presents a model where Scrum artifacts are depicted. Once the product backlog contains the requirements, it is the developers that usually create Sprint backlog, and later on implement items allocated in it.

Today, the above model is increasingly being challenged by involving users early on in the development. For instance, Koski et al. [14] report a development effort where a mission critical system – context where document-heavy development style is usually advocated – is built so that at least some of the end users are constantly collocated with the developers.

When considering the developer, the traditional two tools needed are the compiler and the debugger. These help the developers track down what takes place as programs are built and run. As long as the developers were working independently, there was little need for additional infrastructure. However as the number of developers working on the same system – and more recently, even on the same module – grew, facilities for managing the different versions became necessary. The same goes for configuration management systems; it was impossible to manage all configurations that were to be designed in the joint human memory of the development teams.

With increasingly complex configurations and frequent changes in software, constant, time-consuming compilations formed yet another impediment. To this end, build systems provide support for working with the same code base, which is compiled whenever the software is changed [6]. Distributed version control systems further allow development to be done in isolation and the integration of the work of the developers to be tooled. This push towards faster compilations and delivery to end user culminates in DevOps [12], where the developers are working similarly to and in collaboration with the operators, and deliver updates several times a day. This obviously requires a carefully designed interplay of numerous different development tools – and of the people who use them as well.

In addition to the tools that developers themselves use, there is a growing tendency to collect data from end users to support determining which direction the development should go to [5]. With such tools, it is possible to create controlled experiments to evaluate different options with actual end users, in particular in cases where it is next to impossible to reach a satisfactory, truly representative person to consider. The data collection setup resembles scientific experimentation. Moreover, collecting data obviously raises concerns with privacy, but at the same time, end user observation instead of direct communication with end users avoids several biases, such as non-response bias, recall bias, interviewer effects and types of response biases altogether. Thus, the analysis is more in the domain of inferential statistics, and not so much in communication. However, one must still follow the basic rules of scientific experimentation in order to avoid other kinds of subjectivity.

4. SOCIAL DIMENSIONS IN DEVELOPMENT AND DEPLOYMENT

Tools, methods and processes that are used to develop software also define the necessary interactions between the developers and the different stakeholders of the system. In the following, we address the three main eras of software development — Waterfall, Agile, and DevOps – based on the characteristic properties they pose. Obviously, the boundary between any of these approaches is a fine line, and therefore the fashion communication takes place should be considered as a continuum. For instance, executing the waterfall model several times in sequence results in an iterative development approach, which shares some of its characteristics with Agile development. Similarly, an Agile setup that is based on continuous integration and deployment obviously shares some of its properties with DevOps, while the latter also includes properties that are beyond what Agile approaches propose.

Waterfall era: Developers form teams that interact as they design the interaction between modules that are used for allocating tasks; the same paradigm is reflected in many programming languages, where module constructs are designed for such use. In contrast to direct interaction between developers, the interaction with users is vague. Requirements are gathered and documented in a separate document or into an information system, and they are only revised as the last resort. In contrast the deployment of the system takes a long time and requires deep collaboration between the developers and users of the system.

Agile era: Developers form tight-knit teams that share responsibility over the code the team has designed. Consequently, the interaction between developers is frequent, which is sometimes enforced with practices such as the Daily Scrum. As for interaction with the end users, there are different practices, but they share the idea that developers should focus on the actual development. For instance, in Scrum, the Product Owner manages the requirements in the product backlog, and negotiates with the developer team regarding how and in which order to implement them. Regarding deployment, the model is still fundamentally based on installations that require collaboration between developers and users.

DevOps era: Developers cooperate with system operators and end users to define actual needs. Once the needs are understood, the developers can single-handedly implement the changes across the whole system and deploy the outcome. Moreover, when no easy access to end users is available for eliciting requirements, analytic tools can be used to gather real usage data based on which decisions regarding the future of the software system can be made. As tools play a pivotal role in achieving this, they also take care of developer-to-developer communication regarding day-to-day issues, at least partially. However, regular meetings between developers can take place to synchronize tasks and to share common status. Thus, while often advocated as an Agile approach, many of the characteristics of DevOps are not really serving the ideals of the Agile Manifesto mentioned above, but are rather about tools and processes instead of interactions. Moreover, as features are deployed as soon as they are designed and implemented, fewer interactions are required.

To summarize, we claim that the social dimension in software engineering is experiencing a shift from developer-centric interaction to a setup where tools are largely automating this kind of communication. This shift can be seen in the emergence of pull-based communication and in how modern development environments support implicit coordination of work [2]. At the same time, the ability to deliver end-user value more rapidly implies that the developers must also interact with the users more directly. Developers

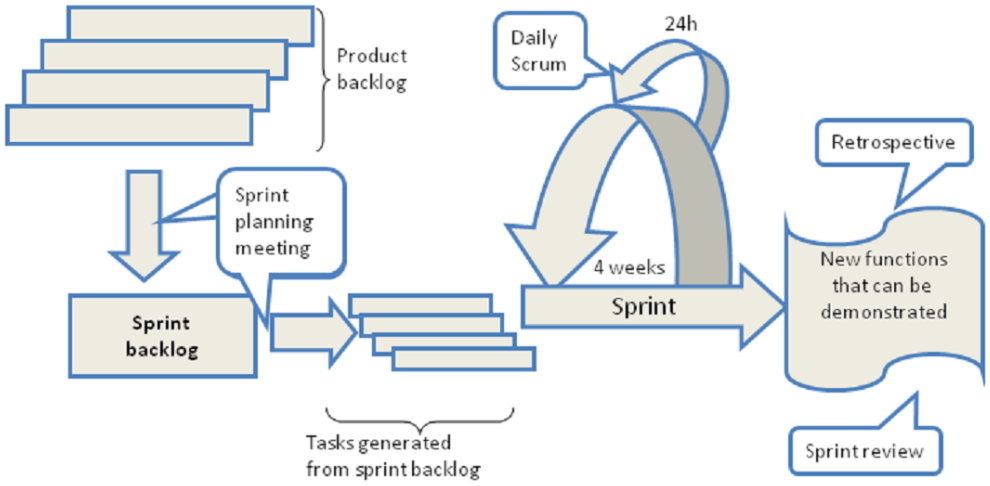

Figure 2: Artifacts and actions in Scrum.

can no longer only rely on mediators to deliver information regarding end-user needs. Figure 3 illustrates the communication levels of modern developers. At present, we lack the means to implement this in a way that is easy, casual, and practical.

5. SURVEY: STATE-OF-THE-PRACTICE IN INDUSTRY

In order to study the state-of-practice development setups in industry, we executed a personal opinion survey extended with some questionnaire-like open ended questions. The questions regarded the social dimensions in software development as well as practices that go along with the social part.

5.1 Study Setup

The role of the cross sectional survey [13] was to collect data on how software developers working in modern software intensive companies communicate with each other as well as with the customer, and what is the role of the development tools. The questions of the survey included background questions, where information was collected regarding the size of the team and the type of the project the developer is working with. In addition, we asked what the preferred communication mechanisms are.

Questions addressing the actual communication in the development team were presented in the following form:

- What feedback mechanisms you have in your project?
- What are the main sources of feedback in technical quality?
- What are the main sources of feedback for the validation of the features?
- What are the main communication ways while deciding the task allocation with the team?

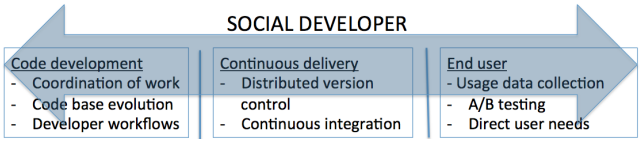

Figure 3: Illustration of the overarching dimensions of communication in modern software development.

with multiple choice questions. In addition to the mechanisms, we asked how well communication methods in the project suited the respondents (scale 1=not at all, 7=ideally), and with an open ended question, how would they improve communication within their projects.

The survey was designed by two researchers and first sent to a test group consisting of researchers working in the same research project with topics related to continuous software engineering and with knowledge of software development. Based on their feedback, the survey questions were improved. The survey was disseminated to respondents working in Finnish software intensive companies by email, by direct messaging and by face to face discussions. The companies were allowed to distribute the survey as they saw fit and forward it to their partners. Among the companies were 20 either large enterprises or SMEs. Additionally several small startups were contacted. The questions were in Finnish. One of the companies requested an English version, which was sent; however the results reported in this paper only cover responses from the Finnish survey, and extending the survey remains a part of future work.

5.2 Results

Background: Of the 25 responses to the Finnish survey, 13 worked in a customer project, 11 in product development, and 1 in a non-specified project type. In the open ended question, 8 respondents were working in a co-located team and 12 in a distributed team. The rest included 1 startup, 3 mixed environments with both co-located and distributed aspects, and 1 Free/Libre/Open source developer (Figure 4). The distributed teams were mostly distributed over different cities in Finland (total of 8 answers). Out of the 4 globally distributed teams, 2 mentioned the US as the other site, and one team had a developer in Costa Rica. One global team was not specified.

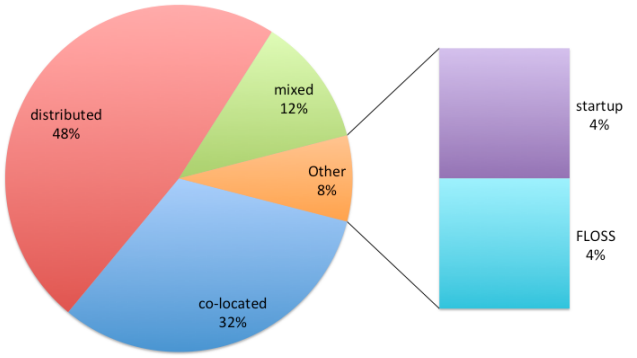

Figure 4: The distribution of the type of the respondents' project work based on their open ended question responses.

Regarding the best-suited way to communicate in the project, instant messaging was most popular (92%), closely followed by face to face contact among team members (84%). In addition, emails (64%) and task management systems (60%) rated rather high. In contrast, discussions with end users were mentioned only by 20% of the respondents. The detailed results are presented in Figure 5.

Available feedback mechanisms: The most commonly provided feedback mechanisms were version management systems (84%) and task management systems (76%) as shown in Figure 6. In addition, bug reporting systems and direct feedback from the clients were reported by 64% and 64% of the respondents, respectively. The least common feedback mechanism was testing team (32% of respondents), probably reflecting the fact that in DevOps type of development, there is little need for separate testing function.

Technical quality: The most important feedback system regarding technical quality was bug reporting system (used by 44% of the respondents), closely followed by direct feedback from customers (40%) and peer reviews (40%). The responses are presented in detail in Figure 7.

Feature validation: Regarding feature validation, design meetings were regarded as the most important mechanism (54%), followed by daily meetings (42%), project manager (42%), and – perhaps surprisingly – instant messaging systems (38%). Figure 8 presents all the responses.

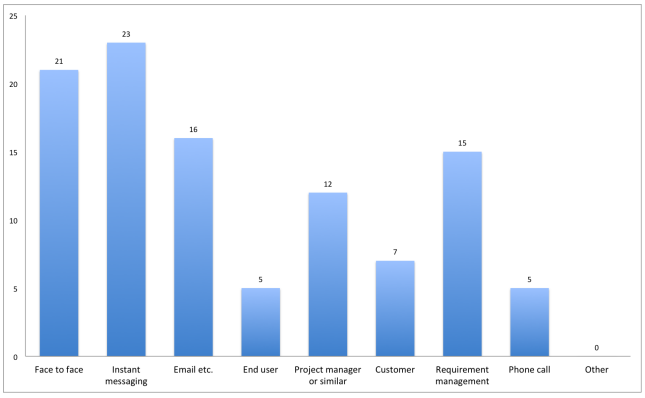

Figure 5: Preferred communication mechanisms.

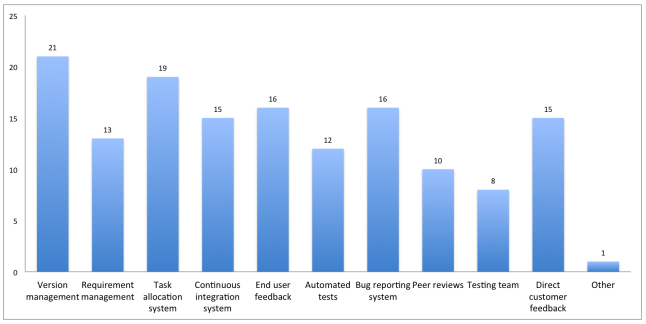

Figure 6: Available communication mechanisms.

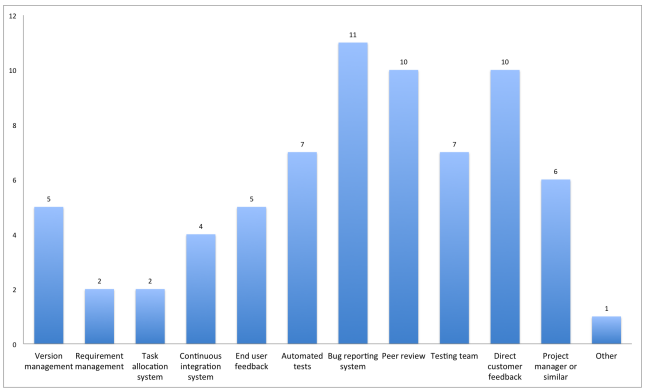

Figure 7: Technical quality.

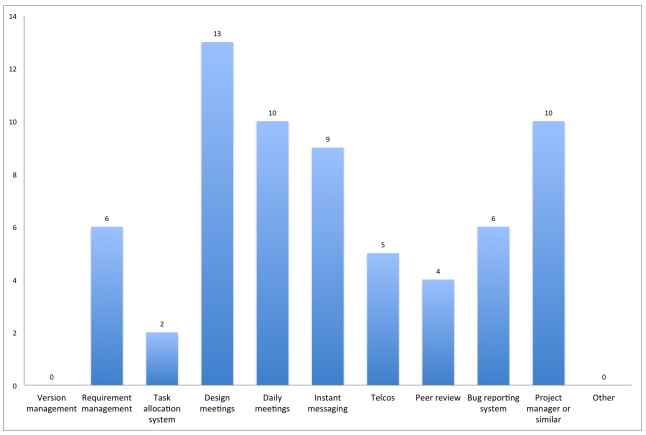

Figure 8: Feature validation.

Task allocation: Face-to-face meetings are the most important mechanism for task allocation, advocated by 76% of the respondents. In addition, project manager (60%) and task management systems (44%) play an important role. All the responses are presented in detail in Figure 9.

Improvements: The main improvements that were proposed were still to have the team co-located in order to improve communication between developers. Moreover, tighter integration of the customer in the project was proposed by several respondents. In addition, some of the respondents were critizing informal discussions with product owners and customers in particular over the phone, where it is difficult to record the requirements right.

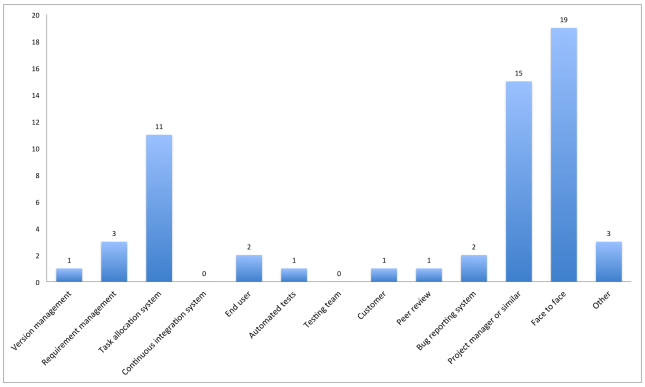

Figure 9: Task allocation.

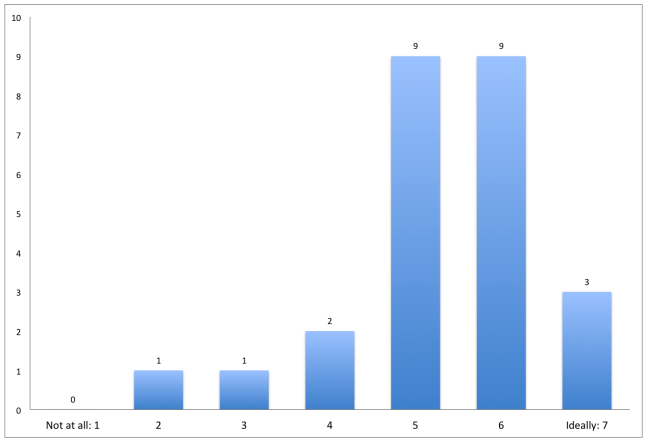

Figure 10: Developer satisfaction with the suitability of the communication channels used.

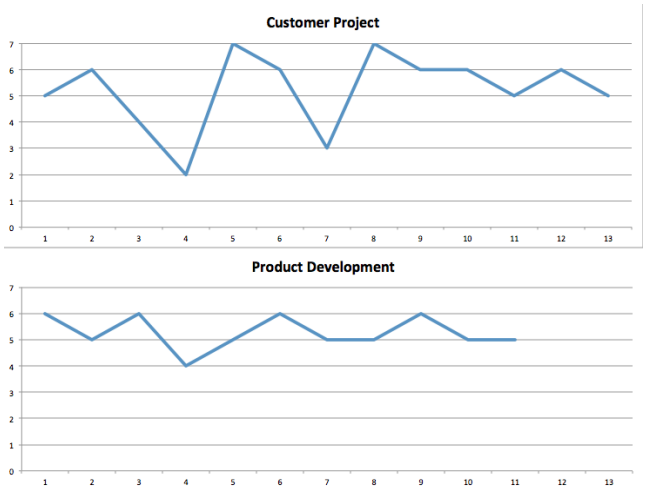

Figure 11: Difference between communication satisfaction of product development and customer projects.

5.3 Summary and Discussion

Based on the data, the need for non-disruptive and asynchronous communication is evident, and also its rationale can be easily explained. Developers simply appreciate fast communication in a tool assisted fashion. In contrast, interruptions that break the development flow are annoying, and for example instant messaging is preferred to phone calls – likely due to the less disruptive nature of messaging, leading in fewer context switches.

Making things visible through tools, boards, war rooms, tickets etc. is highly appreciated. The need for communication tools and open communication instead of meetings and calls came through. While few developers wanted direct interaction with the client, it was also mentioned as something to be improved. Consequently there is room for better feedback loops from the users to the developers, preferably through asynchronous means. While peer reviews are not among the most used feedback channels, they were mentioned as something to do more in order to foster cooperation. Finally, while commonly available, end user feedback appears to be a deprecated way to interact, at least among developers.

Mostly the developers were satisfied with the used communication methods as seen in Figure 10. However, the developers working in product development tended to be more satisfied than those working on customer projects (Figure 11. At the same time, the most satisfied developers were also among those working in customer projects. This may be caused by the customer influence on communication.

5.4 Limitations

Internal validity: There is an unavoidable selection bias in the selection of the companies due to the research project context and the local IT sector. This is mitigated by the wide range of companies contacted. There is also an evident self-selection bias as the companies distributed the survey internally as they saw fit and thus may have chosen certain types of development teams and products as their focus. Additionally, the respondents may have chosen to answer based on their own interests in DevOps and modern tool and communication chains.The wording of the questions can further mean different things to different people. Some are also more likely to use the extremities of the answer scale than others.

External validity: The main threat to the external validity is the companies being only Finnish and mainly operating in the Tampere region in Finland. This may lead to the results be less generalizable. This threat is mitigated by the amount of geographically distributed teams, some of which are also globally distributed.

6. DISCUSSION

The practice of software engineering seems to be driven with the necessity to do software that is just about as complex as we can implement with the present tools, methods and practices. Thus, the evolution in software engineering practices has had a profound effect on how developers interact with customers, end users, and each other. The mechanistic model visible in the Waterfall model has been replaced with approaches that build on interaction and rapid implementation. Moreover, the recent trends are leading towards interactions that manifest in information systems that help in the development and deployment of the resulting software. The faster deployment in turn supports interactions

by providing data regarding the actual use of the system as well as user feedback.

The need for interaction between developers and with other stakeholders of software projects has already changed the characteristics of a successful software engineer. Today, a representative software engineer is almost an opposite of the traditional caricature nerd – excellent people skills are needed to succeed in customer-related operations and in in-house development, where the developers work closely together. This has placed totally new requirements for the education, as each developer must be able to solve problems individually but also be a team player. This also indicates the need for asynchronous communication despite the increased focus on team cohesion.This was also affirmed by our study, as the developers seemed to value instant messaging over meetings.

In addition to the increasing amount of interaction, tool chains that allow faster delivery of end-user value are gaining importance. Based on interactions with various stakeholders, the developers are introducing new features that the customers or end users need, with tools that allow hundreds or even thousands of changes per day. Then, understanding what is truly valuable either is evaluated with further interactions, or by using additional tools, to the extent that at times the development is simply a series of changes that need no direction at all [5]. Moreover, while it is possible to operate in a mode where close interactions between developers and other stakeholders result in new features, comprehensive updates and refactoring may require a change in the mode of operation in the development organization [26].

As individual developers are able to carry out changes in the system single-handedly and in isolation, communication among developers as well as between the developers and customers requires using information systems. For the former, it is common to use wikis, version control systems and instant messaging systems to communicate progress in the development. In contrast, when communicating with the customer, requirements management systems such as Jira[2] are used. According to our study, while developers seem to appreciate face to face interaction within the team, asynchronous communication means are generally valued.

Since none of the communication systems used is intended to be used as a communication system among different parties, we expect that the increasing social interactions between developers and other stakeholders will result in new type of information systems, that resemble both traditional requirements management systems as well as social media. This trend is already visible in the developer's use of social media [28, 27]. Today, these new properties could probably best be studied by using mashups that connect data from various sources (e.g. [31, 16]) instead of focusing on each aspect in isolation. There is an intricate interplay of social dimension from writing code in isolation to communication of end user needs that requires further research focus. Obviously there are additional new, unforeseen directions for future research – new socio-technical implications, coordination patterns, the interplay of development in isolation and in collaboration and stakeholder connections.

Understanding how socio-technical congruence will manifest itself in the new ways of working will further provide excellent research topics, which will also help understanding how software of the future should be created and maintained. Moreover, there is a lot of room for tool infrastructure that helps in maintaining the congruence as well as supports the different types of interactions.

[2]https://www.atlassian.com/software/jira

7. CONCLUSIONS

The liberation of the development team to operate on their own terms, rather than being forced to a joint process, has advanced in parallel with the introduction of tools that truly help the developer in creating the software. This evolution is leading to shared responsibility over the code base and, to increasing extent, also over satisfying end-user expectations. Moreover, the characteristics that a successful developer must display are rapidly evolving as a part of this change as well.

In general, the transition from allocating responsibility in accordance with software architecture to a model where developers bear a shared responsibility will have consequences that can only be hypothesized at this time. The burning open questions include what will be the socio-technological congruence of future software systems, how should this be taken into account in their design and development infrastructure, and what consequences this will have on the actual development. However, it is apparent that a modern software developer does not shy away from asynchronous communication and development tools as communication media.

8. ACKNOWLEDGMENTS

The authors wish to thank Digile's Need4Speed program[3] funded by the Finnish Funding Agency for Innovation Tekes[4] for its support for this research.

9. REFERENCES

[1] K. Beck. *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2000.

[2] K. Blincoe and D. Damian. Implicit coordination: A case study of the rails oss project. In *Open Source Systems: Adoption and Impact*, pages 35–44. Springer, 2015.

[3] M. E. Conway. How do committees invent. *Datamation*, 14(4):28–31, 1968.

[4] B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268–1287, 1988.

[5] D. G. Feitelson, E. Frachtenberg, and K. L. Beck. Development and deployment at Facebook. *IEEE Internet Computing*, 17(4):8–17, 2013.

[6] M. Fowler. Continuous Integration, May 2006.

[7] M. Fowler and M. Foemmel. Continuous integration. *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*, 2006.

[8] G. Gousios, Z. Andy, and S. Margaret-Anne. Work practices and challenges in pull-based development: The integrator's perspectivel. In *Proceedings of the 37th International Conference on Software Engineering*, pages 358–368. ACM, 2015.

[9] G. Gousios, M. Pinzger, and A. v. Deursen. An exploratory study of the pull-based software

[3]http://www.n4s.fi/

[4]http://www.tekes.fi/en/tekes/

development model. In *Proceedings of the 36th International Conference on Software Engineering*, pages 345–355. ACM, 2014.

[10] J. D. Herbsleb. Global software engineering: The future of socio-technical coordination. In *2007 Future of Software Engineering*, pages 188–198. IEEE Computer Society, 2007.

[11] J. Humble and D. Farley. *Continuous delivery: Reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[12] J. Humble and J. Molesky. Why enterprises must adopt devops to enable continuous delivery. *Cutter IT Journal*, 24(8):6, 2011.

[13] B. A. Kitchenham and S. L. Pfleeger. Personal opinion surveys. In *Guide to Advanced Empirical Software Engineering*, pages 63–92. Springer, 2008.

[14] A. Koski and T. Mikkonen. Rolling out a mission critical system in an agilish way: Reflections on building a large-scale dependable information system for public sector. In *RCoSE'15*, 2015.

[15] R. C. Martin. *Agile software development: principles, patterns, and practices*. Prentice Hall PTR, 2003.

[16] A.-L. Mattila, T. Lehtonen, K. Systä, H. Terho, and T. Mikkonen. Mashing up software management, development, and usage data. In *RCoSE'15*, 2015.

[17] M. D. McIlroy, J. Buxton, P. Naur, and B. Randell. Mass-produced software components. In *Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany*, pages 88–98. sn, 1968.

[18] T. Morgan, N. Uomini, L. Rendell, L. Chouinard-Thuly, S. Street, H. Lewis, C. Cross, C. Evans, R. Kearney, I. de la Torre, et al. Experimental evidence for the co-evolution of hominin tool-making teaching and language. *Nature communications*, 6, 2015.

[19] P. Naur and B. Randell. Software engineering: Report of a conference sponsored by the nato science committee, garmisch, germany, 7-11 oct. 1968, brussels, scientific affairs division, nato. 1969.

[20] M. Paasivaara, S. Durasiewicz, and C. Lassenius. Using scrum in distributed agile development: A multiple case study. In *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on*, pages 195–204. IEEE, 2009.

[21] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[22] D. L. Parnas and P. C. Clements. A rational design process: How and why to fake it. *Software Engineering, IEEE Transactions on*, (2):251–257, 1986.

[23] M. Paulk. *Capability maturity model for software*. Wiley Online Library, 1993.

[24] W. W. Royce. Managing the development of large software systems. In *proceedings of IEEE WESCON*, volume 26. Los Angeles, 1970.

[25] S. Sahay. Global software alliances: the challenge of'standardization'. *Scandinavian Journal of Information Systems*, 15(1):11, 2003.

[26] J. Savolainen, N. Niu, T. Mikkonen, and T. Fogdal. Long-term product line sustainability with planned staged investments. *Software, IEEE*, 30(6):63–69, 2013.

[27] M. Squire. Should we move to stack overflow? measuring the utility of social media for developer support. In *Proceedings of the 37th International Conference on Software Engineering*, pages 219–228. ACM, 2015.

[28] M.-A. Storey, L. Singer, B. Cleary, F. Figueira Filho, and A. Zagalsky. The (r) evolution of social media in software engineering. In *Proceedings of the on Future of Software Engineering*, pages 100–116. ACM, 2014.

[29] K. Sureshchandra and J. Shrinivasavadhani. Adopting agile in distributed development. In *Global Software Engineering, 2008. ICGSE 2008. IEEE International Conference on*, pages 217–221. IEEE, 2008.

[30] M. Syeed. *On the Socio-Technical Dependencies in Free/Libre/Open Source Software Projects*. Publication 1300, Tampere University of Technology, Tampere, Finland, 2015.

[31] J. Yu, B. Benatallah, F. Casati, and F. Daniel. Understanding mashup development. *Internet Computing, IEEE*, 12(5):44–52, 2008.