Visualizations for Software Development Process Management

Timo LEHTONEN [a], Timo AHO [a], Kati KUUSINEN [b] and Tommi MIKKONEN [b]
[a] *Solita PLC, Åkerlundinkatu 11, FI-33000 Tampere, Finland*
[b] *Department of Pervasive Computing, Tampere University of Technology, Korkeakoulunkatu 1, FI-33720 Tampere, Finland*

Abstract. Software development projects have increasingly been adopting new practices, such as continuous delivery and deployment to enable rapid delivery of new features to end users. Tools that are commonly utilized with these practices generate a vast amount of data concerning various development events. Analysis of the data provides a lightweight data driven view on the software process. We present an efficient way of visualizing software process data to provide a good overall view on the features and potential problems of the process. We use the visualization in a case project that has become more agile by applying continuous integration and delivery together with development and infrastructure automation. We compare data visualizations with information gathered from the development team and describe how the evolution can be understood through our visualizations. The case project is a good example of how a change from a traditional long cycle development to a rapid cycle DevOps culture can actually be made in a few years. However, the results show that the team has to focus on the process improvement continuously in order to maintain continuous delivery all the time. As the main contribution, we present a lightweight way to software process visualization. Moreover, we discuss how such a heuristic can be used to track the characteristics of the target process.

Keywords. software visualization, continuous delivery, DevOps,

1. Introduction

Implementing a modern development tool chain calls for several technical enablers, such as continuous integration [1] and smooth deployment [2]. In general, aim at the reduction of the time it takes from completed implementation to deployment has resulted in DevOps [3] where developers and operators work as a team to deliver value to end users with an intensive feedback loop.

In such a setting, where tools are constantly playing a major role in the fashion the development advances, developers' actions are reflected as recurring patterns like version history commit, deployments and issue management events. These patterns form traces to information systems. This software engineering data can be processed and mined in the databases.

In this paper, we investigate such traces in the light of the evolution of the development process. The paper is built on mined data from the issue management system of an industrial project executed by Solita PLC, a Finnish software development and

consulting company that specializes in web software and business intelligence. We have analyzed thousands of issue management system tasks in detail with a visual approach to describe the actual changes in the development process. The project in question is a public sector web-based data intensive software project that uses a set of typical software development tools that have automatically generated data for analysis during the years.

In this study, we apply information visualization to demonstrate the evolution of the software development process during a five year long period. During the time frame both development tools and practices have evolved. In the beginning, the process could be described with manual long-lasting implementation periods and the Scrum culture approach. After five years, the approach has transformed to rapid cycles and automated mechanisms for infrastructure with monitoring and quality assurance as an integral part of daily development work. As a tool for analysis, we use visualizations of the data stored in the issue management system. Furthermore, we collected the opinions regarding the visualizations from the developers and the project manager of the case project. Our exact research question can be formulated as:

RQ: How to demonstrate a software development process by using automatically generated data?

The rest of this paper is structured as follows. In Section 2 we present the relevant related work. We continue in Section 3 by going through the case study regarding the transition between the different development models. Section 4 analyzes the result and in Section 5 we discuss the results in more detail and finally, Section 6 draws some concluding remarks.

2. Background

During the recent years, numerous software companies have invested considerable effort in building and automating their development tool chain often referred to as ”Climbing the Stairway to Heaven” [4]. The evolution of organizations for adopting continuous integration, continuous delivery and even continuous deployment [5] is often a step by step procedure [4]. Continuous integration is a requirement for continuous delivery, which in turn is a requirement for continuous deployment [6]. These strategies can then be applied in transformation towards DevOps [9] and reliable, predictable release engineering [10].

This extensive infrastructure, needed to maximize development and deployment speed as well as feedback collection mechanisms, commonly includes a version control system, a build server, a test server, automated production installations, and number of other tools to support development and management. These components form a *deployment pipeline* [5], which uses an automated set of tools from code to delivery. Feature-driven development [7] is one approach for designing and delivering valuable changes to a software. The development team often manages the features to be implemented in an issue management system, for instance Jira[1]. The issue management data can then be mined, for instance for software process improvement (SPI) [8] purposes.

Various methods are available for analyzing the software engineering data produced by the tools. For instance, machine learning algorithms could predict forthcoming software engineering events. In general, information visualization is a powerful method not

[1] `https://www.atlassian.com/software/jira`

just for presenting results of statistical analysis but also for exploratory purposes. In particular, good visualization can present large amounts of data in a relatively small space and pinpoint insights of what to analyze further [11]. Visualizations amplify the capabilities of the human brain [12] as they increase processing resources, reduce searches, enable pattern detection, and perceptual inference operations. Moreover, visualizations can expand the working memory used for problem solving [13].

In the literature, there are two major disciplines of visualization [15]. *Scientific visualization* refers to processing of physical data while information visualization refers to processing of abstract data. However, the distinction between scientific visualization and information visualization is not clear [15]. Moreover, software visualization is a term for applying information visualization to the domain of software engineering [14]. Diehl et al. [15] present the goal of software visualization as improving the productivity of the software development process. They define software visualization as the visualization of artifacts related to software and its development process. This covers a wide variety of artifacts from program code and documentation to bug reporting and visualizing the structure and behavior of the software. Software evolves over time through program code changes to extend the functionality of the system or simply to remove bugs [15]. In a narrower meaning, software visualization is often used interchangeably with *program visualization* which means the visualization of the software as an executable program [16]. In this sense, software visualization is related to visualization of computer programs. Moreover, according to Petre et. al [17], software visualization uses visual representations to make software visible.

There are multiple examples of applying information visualization to data concerning software development. Chuah et. al [18] use glyphs for viewing software project management data. They applied a visual approach to highlight interesting patterns and anomalies in the data set. Gall et. al [19] apply information visualization to study the release history of a software system. They conclude that information visualization technologies can be effectively applied to the analysis of software evolution and to uncover valuable information. Ohira et. al [20] collected data for software process improvement from configuration management systems, mailing list managers and issue tracking systems and presented the data visually. They mention that real-time visualizations motivated developers to fix bugs, since they were aware that there were still unresolved issues. As a problem, they report that visualizations can be too complicated to understand. In [21], the authors mine version control system data and examine how developers work together. With the visualizations they are able to find interesting phases during the evolution.

To our knowledge, there is a research gap in applying information visualization to software engineering data. We have already studied the relationship of issue management and other software engineering data in an earlier paper [22]. We developed a mash-up of information from multiple sources including issue management system, version control system data and monitoring platform. We applied visualizations to analyze the detailed development process based on the development data.

3. Case Project

This research is based on mining software process data from an industrial project executed by Solita PLC[2], a Finnish software development and consulting company. The time frame we cover is five years during which both development tools and practices have evolved. The evolution has started with long-lasting implementation periods and manual deployment and moved to rapid cycles and automated mechanisms for infrastructure, monitoring, and quality assurance. The data we study is produced by a public sector web-based data intensive software project that uses an issue management system to manage the process. The teams uses the issue management system very intensively in their daily work. The tool is used in in daily meetings and in communication with the customer.

Over the years, there have been several changes in the infrastructure, practices, and operations related to the system as described in Table 1.

Table 1. Major changes in the case project

2011	Dev / Ops	CI-server (Hudson) was taken to use
2012	Dev	Scrambled production database dump (nightly dump automatically available)
2013	Dev	Build tool evolution (from Ant to Maven)
	Dev	Automatic database migration
2014	Dev	Environment independent build (Jan)
	Ops	Scripted production deployment (Jan)
	Ops	Server configuration automation (Ansible + Vagrant) (Nov)
2015	Dev	Automatic database cloning
	Ops	Application sets to be installed declared in a text file
	Ops	Automatic deployment to customer acceptance testing environment triggered by commits
	Ops	Interfaces for monitoring, smoke-testing, and radiator

The changes in Table 1 are divided into categories ”Dev” and ”Ops”. If the changes is related to development, the category is ”Dev”. For example, scrambled database dump from the production in 2012 was a change that boosted development. Moreover, automatic deployment to customer acceptance testing environment in year 2015 was a change related to operations part of DevOps.

The team described that their development process consists of two parts. First, they use a major project-based development cycle, where development is divided into projects of length of 1-2 months each yielding a release. Second, the team uses a continuous minor development cycle that consists of smaller releases. The goal of minor development is to deliver smaller development items continuously to the production environment in short cycles. The goal is not to fix bugs, but if there are any, they are fixed and deployed with a short cycle.

[2] `http://www.solita.fi`

4. Results

We applied an interactive visualization tool [24,25] to a case project where a transformation from older software development methods towards a novel short cycle DevOps-culture has happened during years. We developed the visualization tool further to contain metadata based rules that enable the creation of reference process shapes which can be compared with visual shapes generated from actual software project data. Next, we use the tool to generate various views of the target process. We start by introducing the tool output with an example.

4.1. Sample Issues View

Figure 1 presents issue states and three sample issues extracted from the issue management system data. The initial state for an issue is *Open*.

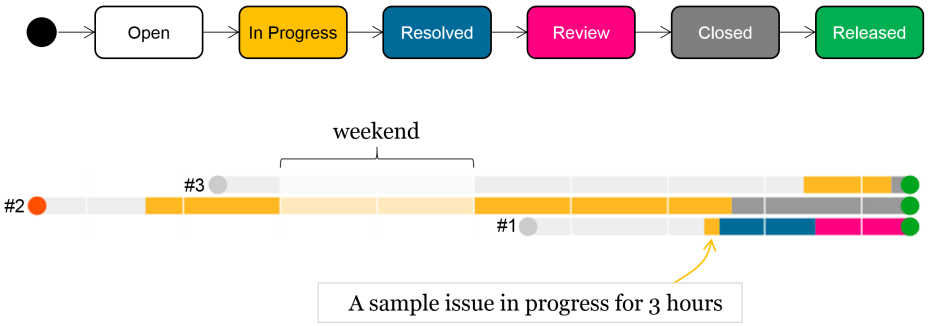

Figure 1. Issue management system states and three sample issues to demonstrate the visualization rules.

In Figure 1, the open state is presented with a dot in the beginning of the issue time line on the left. The color of the dot is gray for issues with default priority and red color indicates higher priority. For instance, issue #2 in Figure 1 has a higher priority.

Issue #2 was created to the issue management system first. The development of the critical issue was started approximately one day after its creation. The developer changes the state to *In Progress*, which is presented with yellow color. Then, apparently nothing happened during the weekend, and finally, issue was done or *Resolved* after six days of development. Then, in a few seconds, the issue was put to *Review* state and then immediately *Closed*. Thus, no blue or pink color is visible for issue #2 . Issue #1 was put to *Resolved* state (blue color) after three hours of development work. Then the issue went to state *Review* (pink color), which means acceptance testing performed by the customer.

The issues are ordered from bottom to top according to the time stamp of *Resolved* state. Thus, issue #1 is on the bottom, because it was resolved first. Then, issue #2 is in the middle because and issue #3last, because it was the latest task that had *In Progress* activity i.e. was resolved last. Next, we construct a reference process shape according to these drawing and ordering rules to Figure 2 and then apply the visualization technique to some tens of released issues presented in Figure 3.

4.2. *Version Release View*

Figure 2 presents the larger major release cycle and minor release cycles below it. The reference process in this case has one major release and two minor releases during a few weeks period.

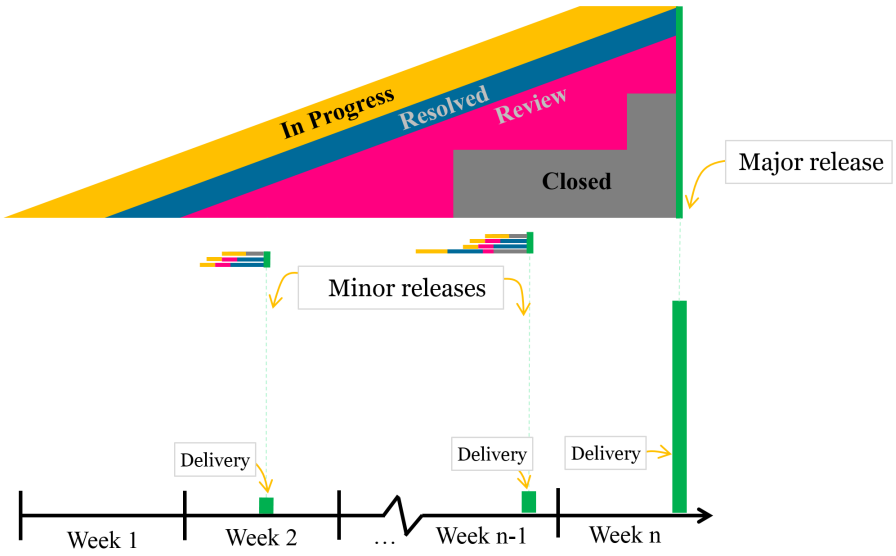

Figure 2. A reference process visualization that reflects issue management system states and the verbal description presented by the team.

Because of the ordering rules described in section 4.1, the reference process forms a triangular shape. The release date of the major release forms a sharp edge to the right. The released issues are drawn to the bottom of the diagram as a green bar which indicates the amount of issues delivered. In the reference process diagram there are three versions delivered – one major version and two minor versions. States *In Progress* and *Resolved* are in the reference visualization drawn with equal length but in practice, their length varies from seconds to weeks. Because the customer reviews many tasks at once, the *Review* states form a shape of a stairway. The number of tasks in one release varies significantly. The team stated in discussions that a total number some tens or hundred tasks would be suitable for their needs.

The visualization in Figure 3 is an actual version released in 2014. The visualization can be compared to the triangular reference process shape in Figure 2.

We can now point out some spots in the visualization that follow the reference visualization characteristics. The version contained over 50 issues that were released in the end of October in one major release annotated in the figure with label `#major`. In spot `#first`, the implementation of the first task belonging to the released version was finished and the state then changed to *Resolved* (blue color). Approximately three weeks later (the weekends are highlighted with translucent white color) the task then went to review state (pink color) in the middle of October. The issue was released in the major version (`#major`) among tens of other features. After the weekend, fix versions `#fix1`,

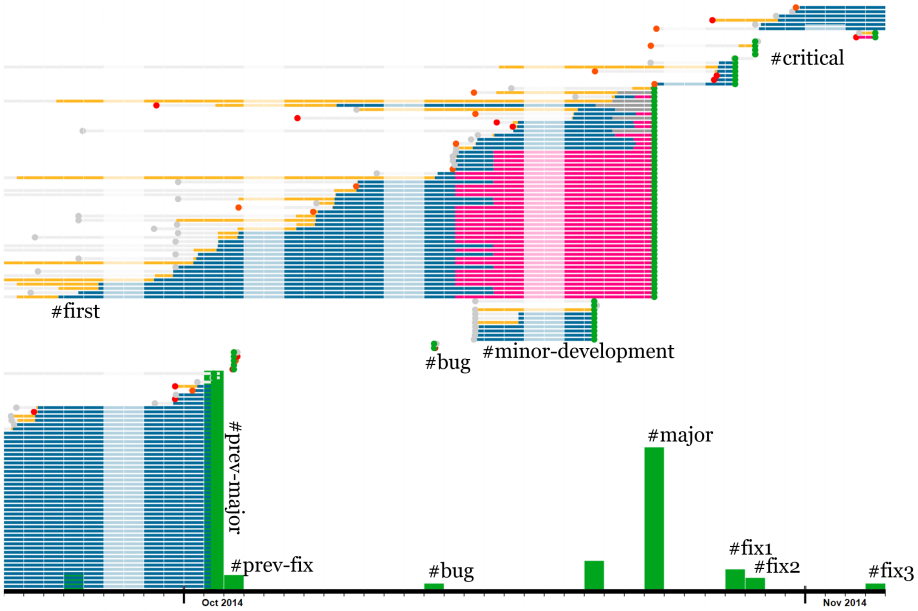

Figure 3. Sample major version released that was released in the end of October 2014. Two fix versions and a smaller minor development release are shown.

`#fix2` and `#fix3` were released. Some of the tasks in `#fix1` was a critical bug pointed out with a red spot (`#critical`). In the mean time, there was also a parallel minor version released in spot `#minor` on the bottom of the visualization. The minor version contained approximately ten tasks. Moreover, in spot `#bug` there was a critical bug fix deployed with a lead time of some hours.

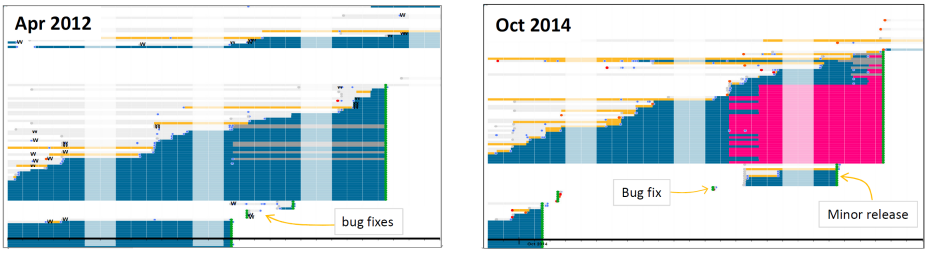

Figure 4. The evolution of the process – in the past there was no minor parallel development, but nowadays the team uses rapid cycle minor development.

Figure 4 presents an example of a change in the process. The sample release visualization from Oct 2014 on the right shows that there is a minor release in parallel to the major release. There was also a critical bug fix in the middle of October in 2014. When we compare this visualization to the other visualization from Apr 2012 on the left, we can point out three differences. Firstly, the earlier release visualization is missing a minor release. However, there are bug fixes visible. Secondly, in the earlier visualization, work estimates were made per issue, which is shown with character `'W'` in the visualization.

Finally, there is no *Review* state (pink color) in the earlier release. The team took *Review* state to use in September 2012. According to these simple visual observations, it is known that the process has evolved during years. The team has found out a way to fulfill the urgent needs of the customers with a short feedback cycle. Moreover, the focus of the team has changed from work estimating to value adding activities.

4.3. Version Release Series View

Figure 5 presents the reference process shapes of a series of versions. The development work of the previous version continues immediately as development work in the next version. Continuous delivery is visible at the bottom of the diagram.

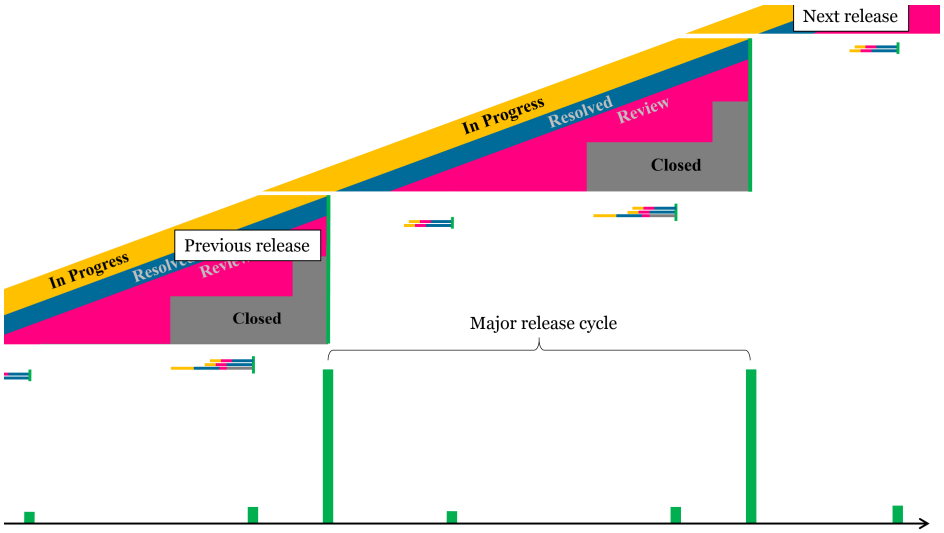

Figure 5. A series of reference process visualizations with certain major release cycle and continuous delivery of parallel minor versions.

Figure 6 presents a wider perspective to the version presented in Figure 3. It is possible to observe that there are multiple simultaneous development tasks going on in the mean time in the upper part of the diagram. Furthermore, the number of delivered features per day are presented in the bottom of the diagram. In spot `#empty` in February 2015, there is a break in the continuous delivery of features apparently because of the giant version release on the top of the visualization annotated with label `#major`.

4.4. Evolution View

To get a total view of the process evolution, we combined an infographic presenting over 5 000 issues from years 2011 to 2015 into Figure 7. There is a reference process in the bottom left corner of the infographic which gives a hint of what the layout of the actual process shapes should be. In this case, the reference process consists of 150 issues released per version with major release cycle of two months, which accompanies the verbal description in section 3. The combined bar chart presenting the throughput on the top of the infographic presents the number of issues released per year half. For instance, dur-

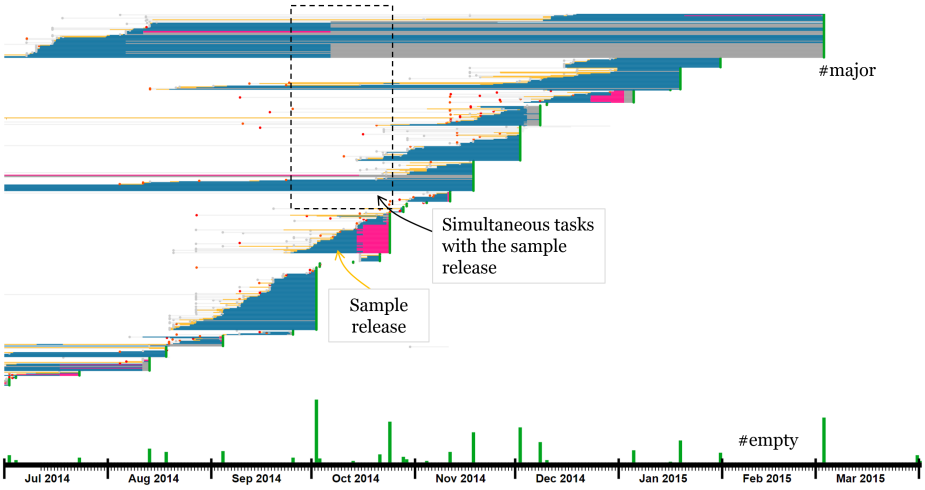

Figure 6. A wider perspective to the version released in figure 3

ing the first year half of 2012, a total number of approximately 500 issues was released. Some of the issues released in the beginning of the year 2012 were developed during the end of year 2011.

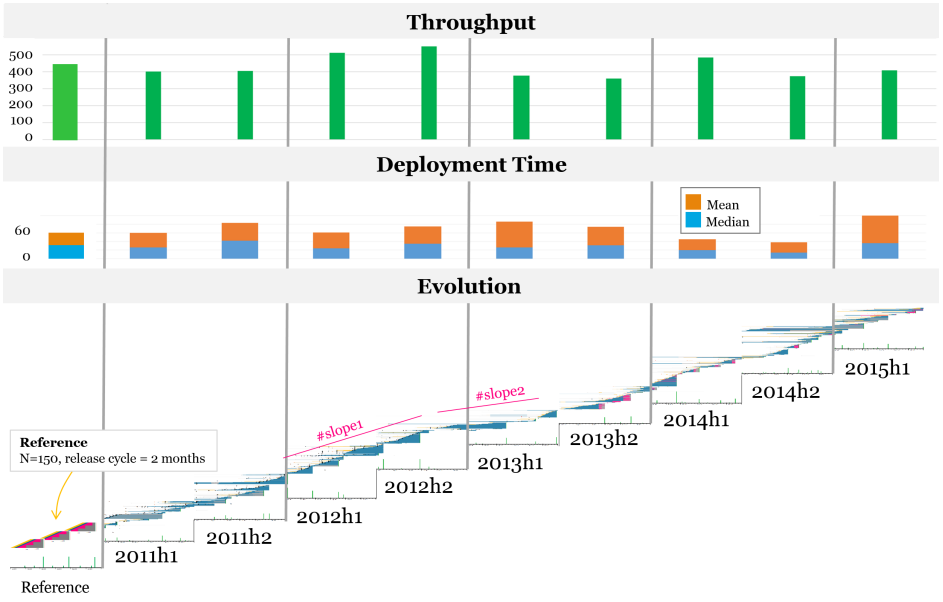

Figure 7. An infographic of the evolution of the software development process during a five year period.

Annotation `#slope1` presents the slope factor reflecting the high throughput during year 2012, which was higher than the slope factor `#slope2` in the beginning of year 2013. The throughput of the team was higher in 2012 than it was in 2013. Naturally, the throughput is affected by, for instance, the size of the tasks – if features are split to smaller

tasks than before, the throughput increases. Moreover, *deployment time* or the lead time from *Resolved* to *Released* is shown in the middle of the infographic. *Deployment time* answers to the following question: when a feature was done, how long did it take until the feature was deployed to the production environment? When we observe the throughput bar graph and the deployment time bar graph visually, the first half of year 2014 seems to have low mean and median deployment time with a high throughput. If we then compare the first half of year 2014 visually to for example the first year half of 2015, the difference is clear. Year first year half of 2015 contains issues with very long tails, of even half year long. This leads to longer deployment times.

5. Discussion

In this section we reflect the results to our research question: *How to demonstrate a software development process by using automatically generated data?*. We start by discussing the results and then reflect them to the opinions collected from the project manager and the developers.

Tools play a major role in novel software development work. The data set of traces the tool usage produces creates a great possibility to evaluate the evolution of the development process. In this paper we combined tens of thousands of events related to issue management system tasks to a compact visual format. From the visualizations, we are able to recognize changes in the development process. For instance, the change towards a development process with separate major and minor cycles can be recognized. Moreover, the continuous delivery phenomenon is made visible and can thus be evaluated. We pointed out problems in continuous delivery with the new visual representation which enables pattern recognition and quick inferences of the process. By combining simple statistics concerning throughput and lead times into a single infographic, we are able to evaluate the evolution of the process.

The visualizations produced by the tool presented in this paper forms a basis for software process evaluation. However, data quality problems related to software engineering data collected are many. Keim et al. [14] list the following error sources as threats to visual data analysis: noise, outliers, low precision, missing values, coverage errors and clones. Problems in raw data quality can jeopardize the conducted visualizations. Moreover, Rosli et al. in their mapping study [26] recognize several typical flaws for data quality in software engineering research. In the context of this research, a typical source for inaccurate data is the everyday usage of the issue management system. For instance, a developer can forget to update the issue management system task state when the actual development work starts or ends. This leads to inaccurate time stamps which affect the visualization. However, we assume that the data is adequately accurate for visual analysis. The data describes actual real-world events that were performed by persons participating to the development of the software. In this sense, the data consists of facts and thus the visualizations describe real-world events that actually happened.

The visualizations reveal interesting facts about the software development process. In Figure 3 it is noteworthy that there are three fix releases after the major release. The reason for them is unknown, but one obvious explanation is that targeting to zero bugs is expensive. The team deploys features actively to the production environment with a short feedback cycle and lets the end-users partly report of the bugs. Naturally, critical

and serious bugs have to be avoided. For instance, bugs in a global marketing system related to billing functions may have expensive consequences and thus have to be avoided. However, bugs related to non-critical sections in a standard public sector software are not life critical and thus partly acceptable.

We collected the opinions regarding the visualizations from the development team in an informal manner. The project manager mentioned that the visualizations make it possible to get an overview of the project with a glance. Especially the comparison of different versions or projects is made possible. According to the project manager, such a comparison would not be possible otherwise. As an improvement, the project manager mentioned that textual labels to the versions would make the visualization easier to read.

The developers of the case project mentioned several points that the visualizations present effectively. Firstly, the visualizations reveal low quality versions by presenting the number of fix versions needed after the release. Secondly, tasks with exceptionally long lead times are revealed. According to the developers, exceptionally long lead times are always a signal of a problem in the development process. Thirdly, the visualization reveals the amount of continuous bug fixing needed. The bug fixes are visible beneath the versions released. Finally, the visualizations indicate if smaller versions are released continuously or not. According to a developer, this reveals if value is produced to the customer continuously or not.

Moreover, the visualizations were also criticized by the developers. They mentioned that errors in the visualizations are many. For instance, some work may not be entered to the issue management system and is thus not visible. Furthermore, one of the developers mentioned unknown correlations as a problem. The interpretations made based on the visualizations may not reflect what actually happened in the real world. Use of other analysis methods to explain the events is needed. As a future work, one of the developers was interested of the impact of using visualizations as a method for reflection in the organization.

As future work, the visualizations techniques could be developed further. The tools developed should be applied to more than one project context in order to evaluate generalizability of the results. Moreover, an analytic pipeline which could demonstrate the status of the project in a continuous manner, could produce valuable information to the project stakeholders. The data collection methods described in this paper can be automated. A tool chain covering all steps from the initial data collection to the visualization of the data can be implemented.

6. Conclusions

In this paper we presented a novel visualization approach for illustrating software engineering projects based on issue tracking tool data. It is important to note that this kind of data is usually generated automatically as a side effect of the project when the tools are used. Data is usually readily available and does not need any extra activities to be used as a basis for visualizations.

Visualization is a light-weight way to get a good view on the overall development process. In addition, it can be used to understand the process more deeply by showing what kind of sprint lengths and common deployment times actually exist, for instance. On the other hand, also anomalies like uncommonly long delivery times for some features are easily noticeable.

We used the visualization to analyze a case software project. In this, we demonstrate how evolution towards a more agile process can both be validated and the effects recognized. The general lead time and feedback cycle has significantly reduced and amount of waste in process diminished. The developed interactive visualization tool can be applied in different scenarios and with different levels of abstraction.

As future work, we are interested in using this visualization tool for multiple projects in different kinds of environments. This way we could visually recognize differences in the patterns of software processes and ask if they actually exist. It would be very intriguing to find some kind of general fingerprint for a healthy process and see how actual process visualizations differ from it.

Acknowledgments

The work was supported by Tekes DIGILE Need for Speed project. We would also like to thank Solita, the case project, and Finnish Broadcasting Company for support and the possibility to perform this research.

References

[1] M. Fowler, "Continuous integration," Available at http://www.martinfowler.com/articles/continuousIntegration.html, 2006, accessed 27.11.2015.

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

[3] J. Humble and J. Molesky, "Why enterprises must adopt devops to enable continuous delivery," *Cutter IT Journal*, vol. 24, no. 8, p. 6, 2011.

[4] H. H. Olsson, H. Alahyari, and J. Bosch, "Climbing the" stairway to heaven"–a mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software," in *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on*. IEEE, 2012, pp. 392–399.

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

[6] M. Fowler, "Continuous integration," http://martinfowler.com/bliki/ContinuousDelivery.html, retrieved: November 2014.

[7] S. R. Palmer and M. Felsing. *A practical guide to feature-driven development*. Pearson Education, 2001.

[8] W. A. Florac and A. D. Carleton. *Measuring the software process: statistical process control for software process improvement*. Addison-Wesley Professional, 1999.

[9] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional, 2015.

[10] A. Dyck, R. Penners, and H. Lichter, "Towards definitions for release engineering and devops," in *Proceedings of the Third International Workshop on Release Engineering*. IEEE Press, 2015, pp. 3–3.

[11] E. R. Tufte and P. Graves-Morris, *The visual display of quantitative information*. Graphics press Cheshire, CT, 1983, vol. 2, no. 9.

[12] S. K. Card, J. D. Mackinlay, and B. Shneiderman, *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999.

[13] D. A. Norman. *Things that make us smart: Defending human attributes in the age of the machine*. Basic Books, 1993.

[14] D. Keim, F. Mansmann, J. Schneidewind, H. Ziegler *et al.*, "Challenges in visual data analysis," in *Information Visualization, 2006. IV 2006. Tenth International Conference on*. IEEE, 2006, pp. 9–16.

[15] S. Diehl, *Software visualization: visualizing the structure, behaviour, and evolution of software*. Springer Science & Business Media, 2007.

[16] J. Stasko. *Software visualization: Programming as a multimedia experience*. MIT press, 1998.

[17] M. Petre, E. de Quincey, et al. A gentle overview of software visualisation. *PPIG News Letter*, pages 1–10, 2006.

[18] M. C. Chuah and S. G. Eick. Information rich glyphs for software management data. *Computer Graphics and Applications, IEEE*, 18(4):24–29, 1998.

[19] H. Gall, M. Jazayeri, and C. Riva. Visualizing software release histories: The use of color and third dimension. In *Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on*, pages 99–108. IEEE, 1999.

[20] M. Ohira, R. Yokomori, M. Sakai, K.-i. Matsumoto, K. Inoue, and K. Torii. Empirical project monitor: A tool for mining multiple project data. In *International Workshop on Mining Software Repositories (MSR2004)*, pages 42–46, 2004.

[21] P. Weißgerber, M. Pohl, and M. Burch. Visual data mining in software archives to detect how developers work together. In *Mining Software Repositories, 2007. ICSE Workshops MSR'07. Fourth International Workshop on*, pages 9–9. IEEE, 2007.

[22] A.-L. Mattila, T. Lehtonen, H. Terho, T. Mikkonen, and K. Systä, "Mashing up software issue management, development, and usage data," in *Proceedings of the Second International Workshop on Rapid Continuous Software Engineering*. IEEE Press, 2015, pp. 26–29.

[23] J. Pearl, "Heuristics: intelligent search strategies for computer problem solving," 1984.

[24] T. Lehtonen, V.-P. Eloranta, M. Leppanen, and E. Isohanni, "Visualizations as a basis for agile software process improvement," in *Software Engineering Conference (APSEC, 2013 20th Asia-Pacific*, vol. 1. IEEE, 2013, pp. 495–502.

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

[26] M. Rosli, "Can we trust our results? a mapping study on data quality," in *Software Engineering Conference (APSEC), 2013 20th Asia-Pacific*, Dec 2013, pp. 116–123.