Monitoring Bottlenecks in Agile and Lean Software Development Projects – A Method and Its Industrial Use

Miroslaw Staron[1] and Wilhelm Meding[2]

[1] Department of Computer Science and Engineering
University of Gothenburg
miroslaw.staron@ituniv.se
[2] Ericsson SW Research
Ericsson AB
wilhelm.meding@ericsson.com

Abstract. In the face of growing competition software projects have to deliver software products faster and with better quality – thus leaving little room for unnecessary activities or non-optimal capacity. To achieve the desired high speed of the projects and the optimal capacity,bottlenecks existing in the projects have to be monitored and effectively removed. The objective of this research is to show experiences from a mature software development organization working according to Lean and Agile software development principles. By conducting a formal case study at Ericsson we were able to elicit and automate measures required to monitor bottlenecks in software development workflow, evaluated in one of the projects. The project developed software for one of the telecom products and consisted of over 80 developers. The results of the case study include a measurement system with a number of measures/indicators which can indicate existence of bottlenecks in the flow of work in the project and a number of good practices helping other organizations to start monitoring bottlenecks in an effective way – in particular what to focus on when designing such a measurement system.

1 Introduction

Global software supply chain and competition contributes to adopting the principles of Lean in many software development organizations[1-3]. The organizations adopt the principles with hopes on achieving high development speed, reduction of unnecessary work (waste) and better alignment with customer needs (generate customer value). Such organizations often realize that the flows of software development are unlike flows of work in manufacturing and that the practice shows existence of bottlenecks unlike the ones in manufacturing. Complexity of large software development projects and the fact that each software development project is different from the previous ones cause the methods proven manufacturing to fail in Lean software development. In particular it is harder to identify, monitor and, above all, remove bottlenecks which hinder the efficiency of the software development projects as software is developed and not manufactured. In this paper we explore methods used at one of mature adopters of Lean software development– Ericsson AB– and observe

D. Caivano et al. (Eds.): PROFES 2011, LNCS 6759, pp. 3–16, 2011.

how they interpret and apply concepts related to bottlenecks. By conducting a formal case study we were able to observe and study the methods of monitoring bottlenecks at one of the units which develops large and complex telecommunication network products. The goal of the research project underlying our study was to identify and monitor bottlenecks that hinder efficient software development according to Lean principles. In particular the project addressed the following research question:

How can we identify and monitor bottlenecks in software development projects in order to prevent inefficiency?

Despite this problem being hard to tackle in general, the organization managed to simplify it by applying certain core concepts of Value Stream Mapping and Lean production systems: throughput and queue. Throughputwas defined as the number of units that are processes by a given phase or activity per unit of time, and queue as the number of units remaining to be developed/processed by a given phase or activity. In order to address this research question a set of measurement tools was combined into a dedicated measurement system which monitored the throughput and queue in a number of phases in the studied software project.

In short, our results show that (i) monitoring of bottlenecks can be effectively done using compound, yet abstract workflow models instead of commonly used process models, (ii) measures used to monitor the flow do not need to be very complex, and (iii) by changing the perception of well-known phenomena (e.g. using throughput instead of productivity) the development of the measurement system is straightforward and is not prone to spun conflicts. .

The remaining of the paper is structured as follows. Section 2 presents an overview of the work that has been done in this area to date. Section 3 presents the design of our research study. Section 4 presents the results of the study - the method and its realization at Ericsson. Finally section 5 presents the conclusions from our research.

2 Related Work

Petersen and Wohlin [4] used measurements of flow – cumulative flow diagrams – to identify bottlenecks. Their approach was based on identifying patterns in the flow diagrams and therefore point to the potential bottlenecks. The difference to our work is that we did not intend to measure flow since we recognized the fact that elements in the flow are not of equal size (something assumed by Petersen and Wohlin). We were also able to use measures which are automatically collected (as opposed to Petersen and Wohlin) which makes the method more efficient. Traces of measures related to flow, similar to Petersen and Wohlin, can also be found in [5].

Höst et al. [6] used discrete event simulation to explore bottlenecks in requirements engineering. Their approach was based on building a model of requirements engineering phase and using simulations to identify where bottlenecks appear. We used similar concepts as Höst et al. – in particular the concept of capacity – since our goal was also to use automated tools and measures.

Another example of using quantitative methods for identifying bottlenecks in a specific case is the work of Jasmine and Vasantha [7] who used graphs in order to identify reuse bottlenecks. Their approach is different from ours w.r.t. the use of mathematical tools behind the measures – matrices and vectors.

Theory of constraints [8, 9] was used in our work in order to initiate the discussions around the concepts of throughput and queue. In the future we intend to use this theory after we establish the criteria and thresholds for the measures of queue and throughput (planned for the future work) and in this way monitor bottlenecks in broad software product development (compared to the focus of one product only we intend to focus on the organization developing a number of products).

Ericsson's realization of the Lean principles combined with Agile development was not the only one recognized in literature. Perera and Fernando [10] presented another approach. In their work, Perera and Fernando, presented the difference between the traditional and Lean-Agile way of working. Based on our observations, the measures and their trends at Ericsson were similar to those observed by Perera and Fernando.

3 Case Study Design

In this section we shortly describe the design of our case study: industrial context, research questions, and units of analysis.

3.1 Context

The context of the case study was one of the software development organizations of Ericsson AB. The organization and the project within Ericsson, which we worked closely with, developed large products for the mobile telephony network. The size of the organization was several hundred engineers and the size of the projects can be up to 200 engineers. Projects were more and more often executed according to the principles of Agile software development and Lean production system referred to as *Streamlinedevelopment* (SD) within Ericsson [3]. In short, the principles of Streamline development postulated that software development was organized as a series of activities centered around the product and executed mainly in cross-functional teams responsible for the design, implementation and partially testing of their part of the product (e.g. a feature) [11]. This was found to be rather typical process design that addressed market pull in Lean development [12].

A noteworthy fact observed in our study was that in SD the releases of new software versions to customers were frequent and that there was always a release-ready version of the system: referred to as *Latest System Version* [3]. It is the defects existing (and known) in that version of the system that were of interest for the project and product management. Ideally the number of known defects in that version of the system should be 0, however, during the development (i.e. between releases) this number might vary as there might be defects which are discovered during the process of integrating new features (parts of code) into the latest system version. In practice there are almost always defects being fixed as integration and testing is continuous. However, at release times the main branch was frozen with the purpose of removing all defects – this removal was the responsibility of the *release project*, which was a project aimed at packaging the product and preparing it for the market availability.

In order to achieve the speed in releasing the software [2] and be able to quickly respond to changes (new features and requirements) the software development

organization needed to constantly operate at an optimal capacity[1]. This means that there should be no bottlenecks that hinder adding new features to software late in the process (i.e. agile response to change) and that such late changes do not postpone the releases of the software.

The capacity of this type of development was limited at any time by a given set of constraints which result in bottlenecks. Examples of constraints could be an amount of test equipment available, resources available for development of features, integration slots for the main branch, amount of resources available for defect removal after the internal testing processes.

The constraints were always present in the organization, but at certain points of time they were triggered and thus result in bottlenecks. For example if too many defects were discovered in the internal testing the resources available for defect removal might not be sufficient and therefore defect handling would become a bottleneck at a particular time.

The requirement from the studied organization for this research project was to establish a measurement system which would provide an overview of the phases in software development and monitor flow of work in the development process. Measured drops in the flow of work should warn the stakeholders about potential bottlenecks.

3.2 Research Questions and Units of Analysis

The research question presented in this section wasestablished together with Ericsson after a number of discussions with the organization in question. The main research question was:

> *How can we effectively identify and monitor bottlenecks in a Streamline software development project in order to assure efficiency of the process?*

The scope of the research project was deliberately limited to identifying and monitoring of bottlenecks excluding the aspects of how to handle the bottlenecks (i.e. how to remove constraints).

The unit of analysis was a single release of the product developed in the studied project. We decided to follow the project from the point of time when the previous release was ready (i.e. the release project took over) until the same point of time for the next release. The time between releases was a number of weeks, although due to confidentiality reasons we cannot provide the exact duration.

Our analyses consisted of both analyses of contemporary measurement systems used at Ericsson, documents (in particular documents related to the ways of working and requirements for quality of products), and analyses of interviews with stakeholders of the contemporary measurement systems as well as key persons from the project management team. The subjects of the interviews were: team leader of one of the development teams in the project, integration leader for the project, deputy project manager responsible for handling of defects during development, system test leader and network integration test leader. All of the subjects were senior software engineers with a number of years of experience in their fields. Three of them were also involved in our previous research studies.

[1] In this paper we can define the capacity as a total number of software features developed (i.e. analyzed, designed, implemented, tested and validated) in one release.

During the discussions within the studied organization we noticed that despite our initial goal to measure the flow and capacity in the organization, we should focus on a smaller and easier problem – monitoring throughput and queue. The simplification was dictated by the fact that the goal was to find bottlenecks, which can be found by observing falling/dropping throughput in phases after the bottleneck and growing queue before the phase where the bottleneck is located over time.

Our research process consisted of three parts:

1. Development of models of workflow in the software development projects organized according to the principles of Streamline development. The models simplified the view of the stakeholders about their ways of working, which in general could be different than the prescribed process view.
2. Development of measurement systems for measuring throughputs and queues for each phase in the workflow model.
3. Instantiation of the measurement systems for one project with the goal to evaluate the method.
4. As one could observe, we put effort on the actual ways of working and the perception of those by our stakeholders, rather than analyses of process descriptions.

The first two steps were done through interviews. For the development of models we asked the stakeholders to characterize their way of working in this phase. The interviews were also aimed to find measures and indicators for monitoring throughput and queue. In order to elicit these measures we asked the following two main questions:

- If you were supposed to notify the responsible stakeholder for the phase after yours about potential problems with the overflow of work – what kind of indicator would you like to use? – to elicit measures of throughput.
- If you were supposed to notify the responsible stakeholder for the phase before yours about potential "shortage" of work in your phase – what kind of indicator would you like to use? – to elicit measures of queue.

These questions were broken down into a number of aspects for discussion during the interviews.

4 Monitoring Throughput and Queue

In this section we present the results of the case study – description of how bottlenecks can be monitored using two main concepts: *throughput* and *queue*. We also present the results of the evaluation of our method in the project at Ericsson.

4.1 Definitions

In order to proceed towards the realization of measurement systems we needed to establish the basic vocabulary used in the project. It was interesting to observe that even the definition of the main concept – bottleneck – was different for different stakeholders in the project. The definitions shown below are taken from a number of sources and discussed with the organization.

- ***Bottleneck:*** A bottleneck is a phenomenon where the performance or capacity of an entire system [auth: software project] is limited by a single or limited number of components or resources.
- ***Flow:*** A flow is the motion characteristic of elements or functionality under development in a software project, see also [5].
- ***Throughput:*** Throughput is defined as the number of elements processed during a time period. For example number of test cases executed per week.
- ***Queue:*** Queue is defined as the number of elements to be processed (to flow) through a specific phase or a software project. For example number of test cases to be executed until the end of a given week.
- ***Capacity:*** Capacity is a total number of software features developed (i.e. analyzed, designed, implemented, tested and validated) in one release.

The general definitions of the concepts above are used to describe elements of software development process used at the studied unit of Ericsson.

4.2 Realization at Ericsson

As described in section 3.2, in order to identify and monitor bottlenecks we needed to establish a measurement system (see [13-15]). However, in order to establish this measurement system we needed to describe the process of software development in a model which allows monitoring of flows – e.g. similar to Value Stream Mapping[16] models using in Lean Development in other domains than software development. An example of such a model is presented later in the paper.

Figure 1 shows an overview model of process of developing software products in a similar way as describing production systems in Lean manufacturing. The boxes represent "stations" where a number of activities take place and the overview model presents those phases which we worked with during the study (the phases were defined by the stakeholders' responsibility areas). The process was drawn as a sequence, since the dependencies between stations are sequential; however, the whole Streamline development process at Ericsson was parallel. The teams developed new features in parallel; TR handling was done in parallel to development and integration, etc. The sequential dependencies meant that bottlenecks in one of these stations could influence stations after – i.e. that there are sequential causal relationships between these phases.

Each phase contained a number of stations, presented in detail in section 4.3. The model was created together with the stakeholders by asking them to characterize the work which they conduct and iterating the model a number of times in order to find the appropriate abstraction level for the description.

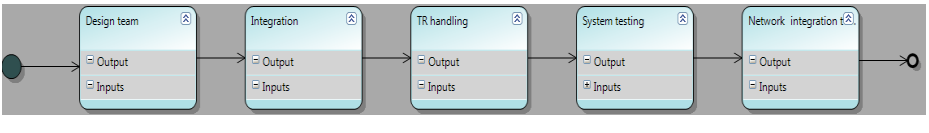

Fig. 1. Overview model of development process and related measurement systems

After describing the process model we needed a number of interviews to validate the model – in particular that it described the de-facto ways-of-working in the studied project. We also needed to perform a number of interviews with the goal to define the main concept of flow for each identified phase and throughput and queue for each phase.

4.3 Results

The results from our case study contain both the model of the workflow in the development connected to measurement systems (Figure 1) and the realization of the measurement systems visualized in a web page updated daily (Figure 6).

The model was created using the notation presented in our previous study [15] and dedicated for modeling measurement systems. The notation was complemented with Value Stream Mapping model of workflow in the development process. The model was intentionally created quite abstract as its purpose was to provide the stakeholders with the overview of the whole development flow. The overview was meant to inform the stakeholders what kind of main activities were executed at each phase. The measures and indicators were linked to the activities captured the notions of throughput and queue for each phase.

The measurement system for monitoring bottlenecks was realized by connecting together a number of measurement systems – one for each phase as it is presented in Figure 1.

The measures for throughput and queue identified during the interviews for the measurement systems are presented in Table 1.

Table 1. Measures for queue and throughput per phase

Phase	Throughput	Queue
Development team	# of function test cases developed per week	# of function test cases planned for the whole feature (but not yet developed)
Integration	# of features integrated per week	# of features planned for integration up to date in the integration plan (but not yet integrated)
Defect handling	# of defect reports entering the state “Closed” per week	# of defect reports that are not in state “Closed”
System test	no indicator was created for this phase[2]	# of test cases planned for execution up to a given week (but not executed yet)
Network integration test	# of test cases executed per week	# of test cases planned for execution up to a given week (but not executed yet)

[2] Due to the process changes in the organization we were not able to define this indicator. Potentially (or theoretically) it could be a similar indicator to the one for network integration test phase.

The measures summarized in the table were collected manually for the development team and automatically for all other phases. The reason for the manual collection was the fact that this information was not available in a form which could be automatically processed.

The rationale behind the measures, as a result of interviews, is presented in the following subsections.

4.3.1 Development Team

The measures defined for the development team were elicited in two interviews with a team leader and a senior system engineer working in the team. In the studied organization, the teams are usually formed by a number of specialists from a number of fields – e.g. designers, programmers, system engineers, function testers, object leaders – and are thus called cross-functional teams. Each team is responsible for the complete development of one feature at a time and is able to integrate the feature into the main code branch after it is tested.

The test cases are developed directly before execution (although they are planned in advanced) and in sequence. Therefore the most effective way to monitor the throughput of work in the team is to monitor the number of test cases developed and the most effective way of monitoring the queue was to monitor the changes in the

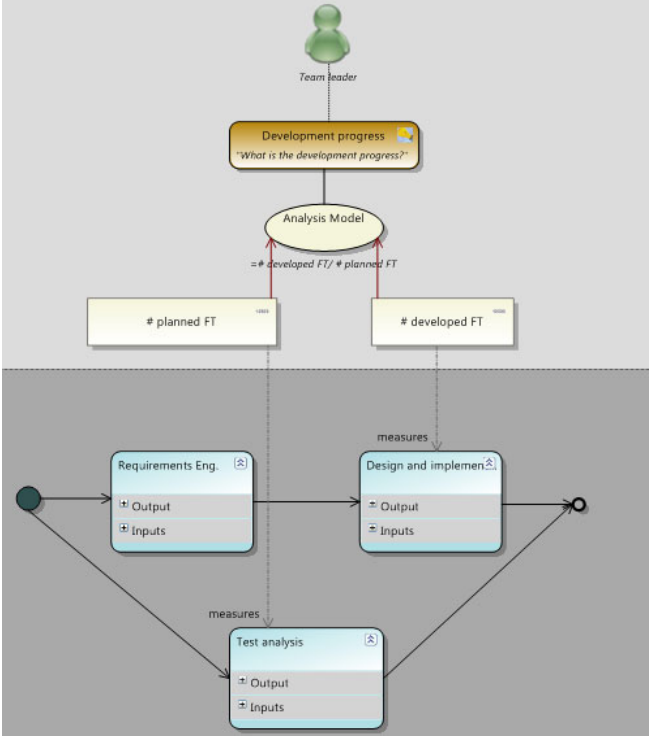

Fig. 2. Stations/activities, measures and indicators in the Development Team

number of test cases planned. The three stations that were identified for this phase are: (i) requirements engineering, (ii) design and implementation, and (iii) test analysis. The model of the flow and the related indicators and measures are presented in Figure 2. The figure shows also how we modeled the work flow in detail and how we linked the activities in the workflow with the measures and indicators for monitoring bottlenecks.

The lower part of the model is the flow model and the upper part is the model of the measurement system according to ISO/IEC 15939 [17] and our own modeling notation [15, 18]. In this flow model the test analysis is drawn in parallel to the requirements engineering and implementation. The reason for this is that there was no sequential dependency between these stations although the implementation and test analysis end simultaneously. The test analysis could influence the queue for this whole phase, i.e. the number of test cases to be developed. If there was a bottleneck in this phase, then the team would not be able to test and develop the test cases as planned.

4.3.2 Integration

When the teams were ready with the development of the feature assigned to them, the feature was integrated into the code at the main branch. This integration was a managed and planned process driven by two main principles: (i) the integration time slots should be used as efficient as possible (meaning that all integration slots should be occupied) and (ii) the integration of features should be spread during the whole project, not gathered in the end (i.e. to avoid big-bang integration).

The integration plan was the basis for the integration and was an electronic document frequently updated. The number of features in the integration plan are planned (but not yet integrated) up to a given week form the queue for this phase. The number of features integrated forms the throughput measure.

The rationale behind the queue was that if there were delays in the integration, it would grow since the features would not be integrated, but remain as planned in the plan. The queue would grow if the throughput drops in case there is a bottleneck in the integration phase since the teams would still deliver new code to be integrated.

If the queue dropped suddenly, it would indicate that there was a bottleneck in the phase before (development team) and therefore the integration process did not operate at the desired/planned capacity.

4.3.3 Defect Handling

During the function testing in the teams or in the integration process, a number of defects might have been reported. Before proceeding with the testing, these defects needed to be removed. The process of assessing the severity of the defect, assignment to the development team and verifying that the defect has been removed needed to finish. There were three main stations that were identified: (i) defect assessment, (ii) defect removal, and (iii) defect verification and closing.

Based on this process the measurement for the throughput was quite straightforward – it was the number of defects that were closed per week. The queue was then the number of defects that were still open and remained to be closed.

If there was a bottleneck in this phase then the queue would raise and it would eventually stop the whole development chain as no features would be ready for the delivery because of insufficient quality.

If there was a bottleneck in the previous phase then the number of open defect would drop significantly meaning that the test cases were not executed or that no new features were ready for integration.

4.3.4 System Test

System test (ST test) was the first phase where the source code integrated into the main branch was tested as a whole. The phase consisted of two main activities – test planning and ST testing (i.e. ST test execution), the measures for throughput and queue were therefore

- the number of test cases planned to be executed (and not yet executed) up to date – measure of queue.
- the number of test cases executed – the measure of throughput. Please note that this is an indicator proposed based on a similar indicator in the forthcoming phase. We were not able to verify this indicator since the internal testing processes were different.

If there was a bottleneck in this phase then the queue would raise while the throughput would drop. A growing queue also indicated an existence of a bottleneck in the ST test phase.

4.3.5 Network Integration Test

Network integration test phase (NIV testing) was the phase when the software products were tested in their target environment and at the target platforms. The function and system test cases were used to test the software in new configurations and therefore it mainly consisted of the test execution itself (defect handling was captured in a separate phase as described before). Test planning was then only limited to planning when different test cases should be executed and, according to the stakeholder, should not be distinguished from the test execution phases as both planning, re-planning and execution were conducted continuously and in parallel.

The two measures were:

- the number of test cases planned to be executed (and not yet executed) up to date – measure of queue.
- the number of test cases executed – the measure of throughput.

If there was a bottleneck in this phase then the throughput would drop as the number of test cases executed would decrease. An example of a constraint causing a bottleneck could be the access to the test equipment or the need to reconfigure the equipment more than planned.

4.4 Industrial Validation

The measurement system for bottlenecks was used in one project so far and resulted in identifying an efficient way of monitoring the flow of work in the project as presented in Figure 3[3]. The measurement system contributed to the identification of problems

[3] Please note that the numbers in the figure are only an example and do not show the real data from the organization due to the confidentiality agreements with the company.

with the last phase of the flow, which were forecasted based on the indicators from the previous phases. The throughput of the test process for the network integration test phase was shown to be insufficient to handle the growing number of features to be delivered. The release of the product was postponed as a result of this insufficient throughput. The future projects took another approach to testing in order to increase the throughput of that phase.

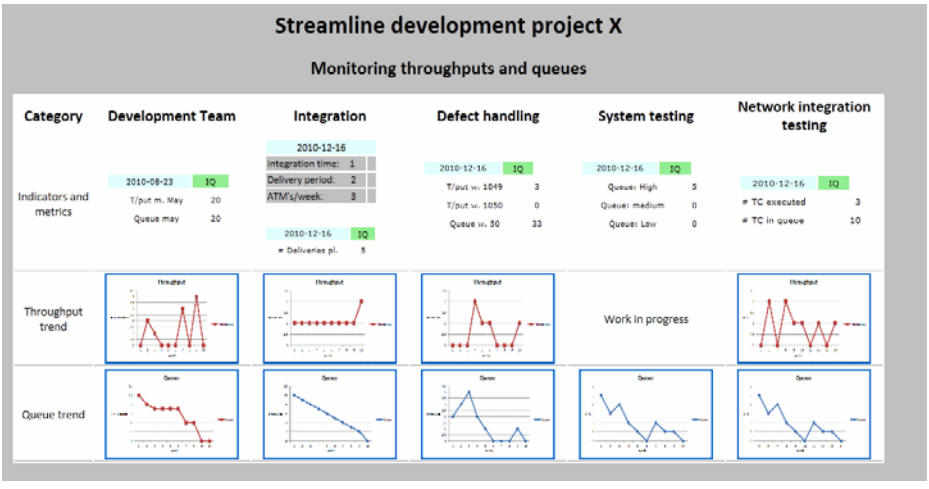

Fig. 3. Realization of the measurement system for monitoring bottlenecks

In Figure 3 we present how we realized the measurement system in practice. Behind the web page shown in the figure we used 5 measurement systems collecting the data from a number of systems, which together form an important tool for monitoring bottlenecks. Although we cannot show the real data, we illustrated dependencies between measures which we observed in the project – the most important one being the dependency between throughput and queues. We can also observe that the phase with the most stable flow of work is the integration phase, which is caused by the fact that the integrated code is usually divided into parts of equal size and that the builds usually take a constant period of time (as the merge, compile, etc. time does not significantly depend on the size of the integrated functionality).

In our further work we intend to investigate whether there are indeed bottlenecks in the flow of work and whether these are measured accurately by the measurement system. As one could observe from Figure 3, at this point of time there seem to be no bottlenecks in the flow.

4.5 Good Practices

While developing and deploying the measurement system we observed a number of good practices which helped the organization to be effective in monitoring bottlenecks.

1. **Monitor throughput, not productivity.** Although monitoring of productivity is tempting in a context like this (i.e. identifying bottlenecks), it was much harder to

achieve consensus on how the productivity should be measures and how it should be compared. As productivity in general is perceived related to humans, the consensus might even be impossible for larger organizations (e.g. productivity of designers and testers cannot be easily compared). However, since throughput is related to the number of items processed per time unit, the focus is on the items processed – hence consensus can be achieved more easily.

2. **Monitor capacity, not speed.** For a similar reason as above one should avoid measuring speed. Since speed is usually related to humans, the consensus is harder to obtain as no one wished to be "the slowest one". Capacity, on the other hand is a property of the organization and increasing speed is only one of the possible solutions (another one is to increase the number of person-hours in the project) and therefore it shifts the focus from individuals to collective responsibility of the project.
3. **Focus on measuring flow, not constraints.** When asked about potential bottlenecks in the organization, the stakeholders in our project could sometimes point to constraints which limited the flow. An example of such a constraint could be knowledge of particular technology. Although commonly referred to a bottleneck such a phenomenon is constraint that limits the capacity of the organization (or speed of individuals who has to learn instead of performing). In addition to that it is often not possible to measure such a constraint. However, if a throughput drops in one of the monitored phases, one could easily identify this constraint and decide to remove it – it is also more important to remove the constraint than to measure it.
4. **Do not forget about velocity of items flowing through the system.** When focusing on monitoring the flow the organization might forget that flow is not the only aspect important to monitor. The fact that the project or organization operates at the optimal capacity does not mean that the items are develop fast enough – it might still take a long time between the idea of the feature and its delivery to the customer. Therefore the organization also should measure whether the velocity of the flow is sufficient, since it is often important to be the first one to market with new features, not only the one with reasonable price.

The observed good practices seem to be rather generic and straightforward to use at other companies than Ericsson. They have shown themselves to be important when establishing and deploying the measurement systems.

4.4 Validity Discussion

In order to evaluate the validity of our study we have used the framework presented by Wohlin et al. [19].

The main threat to the external validity of our study is the fact that we have studied only one software development project. Although this threat is real, we still believe that the concepts of throughput and queue could be defined operatively for other projects.

The main threat to the internal validity is the fact that our study used the process description which was elicited in the study itself. This bears the risk that if we had another process description, then the results of our study would be different. However, since we explicitly asked about the measures during the interviews, the risk for it is minimized.

The main threat to the construct validity of our study is the fact that we have based the identification of our measures on interviews and not on statistical data analysis. In particular we have not used Principal Component Analysis to find related measures or time-series analysis. Since the period of the study was short and the measures were new we were not able to construct a valid statistical support.

5 Conclusions

In this paper we presented a new method for identifying and monitoring of bottlenecks in the flow of work in Lean software development. We conducted a case study at one of the software development units of Ericsson in order to develop the method together with the stakeholders in our project.

The results of our study show that in order to identify bottlenecks one can monitor throughputs and queues at a number of phases in software development. We also identified one bottleneck in the last phase of the development project, which provided the basic evidence that our method can effectively identify and monitor bottlenecks. Therefore we can conclude that monitoring of bottlenecks in software development projects can be done when we can use two crucial concepts - throughput and queue.

Our future work is planned to apply this method for other software development programs and to monitor bottlenecks there. We also plan to generalize this method and find "thresholds" for queues and throughputs that could be reused (something that requires collecting a significant amount of statistical data over longer period of time).

Acknowledgements

We would like to thank SoftwareArchitectureQualityCenter at Ericsson and IT University of Gothenburg for their support in the study. We would also like to thank all managers and stakeholders at EricssonAB for their contributions, insight and support for the project.

References

[1] Staron, M., et al.: A method for forecasting defect backlog in large streamline software development projects and its industrial evaluation. Information and Software Technology 52, 1069–1079 (2010)
[2] Mehta, M., et al.: Providing value to customers in software development through lean principles. Software Process: Improvement and Practice 13, 101–109 (2008)
[3] Tomaszewski, P., et al.: From Traditional to Streamline Development - Opportunities and Challenges. Software Process Improvement and Practice 2007, 1–20 (2007)
[4] Petersen, K., Wohlin, C.: Measuring the flow in lean software development. Software: Practice and Experience, n/a–n/a(2010)
[5] Oppenheim, B.W.: Lean product development flow. Syst. Eng. 7, 2 (2004)
[6] Höst, M., et al.: Exploring bottlenecks in market-driven requirements management processes with discrete event simulation. Journal of Systems and Software 59, 323–332 (2001)

[7] Jasmine, K.S., Vasantha, R.: Identification Of Software Performance Bottleneck Components In Reuse based Software Products With The Application Of Acquaintanceship Graphs. In: International Conference on Software Engineering Advances, ICSEA 2007, p. 34 (2007)
[8] Dettmer, H.W.: Goldratt's theory of constraints: a systems approach to continuous improvement. ASQC Quality Press, Milwaukee (1997)
[9] Dettmer, H.W.: The logical thinking process: a systems approach to complex problem solving. ASQ Quality Press, Milwaukee (2007)
[10] Perera, G.I.U.S., Fernando, M.S.D.: Enhanced agile software development - hybrid paradigm with LEAN practice. In: International Conference on Industrial and Information Systems (ICIIS), pp. 239–244 (2007)
[11] Akg, A.E., et al.: Antecedents and consequences of team potency in software development projects. Inf. Manage. 44, 646–656 (2007)
[12] Liker, J.K.: The Toyota way: 14 management principles from the world's greatest manufacturer. McGraw-Hill, New York (2004)
[13] Staron, M., et al.: A Framework for Developing Measurement Systems and Its Industrial Evaluation. Information and Software Technology 51, 721–737 (2008)
[14] Staron, M., Meding, W.: Predicting Weekly Defect Inflow in Large Software Projects based on Project Planning and Test Status. Information and Software Technology, p. (available online) (2007)
[15] Staron, M., Meding, W.: Using Models to Develop Measurement Systems: A Method and Its Industrial Use. Presented at the Software Process and Product Measurement, Amsterdam, NL (2009)
[16] Dolcemascolo, D.: Improving the extended value stream: lean for the entire supply chain. Productivity Press, New York (2006)
[17] International Standard Organization and International Electrotechnical Commission, ISO/IEC 15939 Software engineering – Software measurement process, International Standard Organization/International Electrotechnical Commission, Geneva (2007)
[18] Meding, W., Staron, M.: The Role of Design and Implementation Models in Establishing Mature Measurement Programs. Presented at the Nordic Workshop on Model Driven Engineering, Tampere, Finland (2009)
[19] Wohlin, C., et al.: Experimentation in Software Engineering: An Introduction. Kluwer Academic Publisher, Boston (2000)