SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2009; **00**:1–7 *Prepared using* speauth.cls *[Version: 2002/09/23 v2.2]*

Measuring the Flow in Lean Software Development

K. Petersen*,†, C. Wohlin‡

Blekinge Institute of Technology, Box 520, SE-372 25 Ronneby, Sweden

SUMMARY

Responsiveness to customer needs is an important goal in agile and lean software development. One major aspect is to have a continuous and smooth flow that quickly delivers value to the customer. In this paper we apply cumulative flow diagrams to visualize the flow of lean software development. The main contribution is the definition of novel measures connected to the diagrams to achieve the following goals: (1) increase throughput and reduce lead-time to achieve high responsiveness to customers' needs, and (2) to provide a tracking system that shows the progress/status of software product development. An evaluation of the measures in an industrial case study showed that practitioners found them useful and identify improvements based on the measurements, which were in line with lean and agile principles. Furthermore, the practitioners found the measures useful in seeing the progress of development for complex products where many tasks are executed in parallel. The measures are now an integral part of the improvement work at the studied company.

KEY WORDS: *Agile Software Development; Lean Software Development; Development Flow; Goal-Question-Metric*

1. Introduction

Agile software development aims at being highly focused and responsive to the needs of the customer [1, 2]. To achieve this practices like on-site customer and frequent releases to customers can be found in all agile practices. Agile practices can be further enhanced by adopting practices from lean manufacturing. Lean manufacturing focuses on (1) the removal of waste in the manufacturing process; and (2) analyzing the flow of material through the manufacturing process (cf. [3, 4, 5]). Both aid the responsiveness to customer needs that agile

*Correspondence to: Kai Petersen, Blekinge Institute of Technology and Ericsson AB, Sweden
†E-mail: kai.petersen@bth.se; kai.petersen@ericsson.se
‡E-mail: claes.wohlin@bth.se

Received xx August 2009
Revised

seeks to achieve. Firstly, removing waste (everything that does not contribute to customer value) frees resources that can be focused an value-adding activities. Secondly, analyzing the flow of development aids in evaluating progress and identifying bottlenecks.

For the removal of waste and the improvement of the flow it is important to understand the current status in terms of waste and flow, which is supported by visualizing and measuring it. Improving the flow means shorter lead-times and thus timely deliver of value to the customer. In manufacturing cumulative flow diagrams have been proposed to visualize the flow of material through the process [6]. Such a flow diagram shows the cumulative number of material in each phase of the manufacturing process. To the best of our knowledge, only one source has proposed to analyze the flow of software development through flow-diagrams by counting customer-valued functions [7]. However, the usefulness of cumulative flow diagrams has not yet been empirically evaluated.

The novel contributions of this paper are: (1) The definition of measures to assess the flow of lean software development with the goals of increasing throughput and creating the transparency of the current status of product development; (2) The evaluation of the visualization combined with the proposed measures in an industrial context.

The case studied was Ericsson AB in Sweden. The units of analysis were nine sub-systems developed at the case company. The structure of the case study design was strongly inspired by the guidelines provided in Yin [8] and Runeson and Höst [9]. The data collection started recently and therefore cannot be used to see long-term trends in the results of the measurements. Though, the case study was already able to illustrate how the measures could influence decision making, and how the measures could be used to drive the improvement of the development flow.

The following measures were defined:

- A measure to detect bottlenecks.
- A measure to detect how continuous the requirements flow through the development process.
- A cost-model separating investment, work done, and waste.

The evaluation of the model showed that practitioners found the model easy to use. They agreed on its usefulness in influencing their management decisions (e.g. when prioritizing requirements or assigning people to tasks). Furthermore, different managers integrated the measures quickly in their work-practice (e.g. using the measurement results in their status meetings). However, the quantification of the software process improvements that can be achieved with the model can only be evaluated when collecting the data over a longer period of time.

The remainder of the paper is structured as follows: Section 2 presents related work. Thereafter, Section 3 provides a brief overview of cumulative flow diagrams for visualizing the flow as well as the measures identifying bottlenecks, continuity of the requirements flow, and cost distribution. Section 4 illustrates the research method used, including a description of the case, a description of how the measures were evaluated, and an analysis of validity threats. The results are presented in Section 5, including the application of the measures on the industrial data, and the evaluation of usefulness of the measures. The practical and research implications are discussed in Section 6. Section 7 concludes the paper.

2. Related Work

The related work covers three parts. First, studies evaluating lean software development empirically are presented. Secondly, measures are presented which are used in manufacturing to support and improve lean production. These are used to compare the measures defined within this case study with measures used in manufacturing, discussing the difference due to the software engineering context (see Section 6.3). The third part presents literature proposing lean measures in a software engineering context.

2.1. Lean in Software Engineering

The software engineering community became largely aware of lean principles and practices in software engineering through Poppendieck and Poppendieck [5] who illustrated how many of the lean principles and practices can be used in a software engineering context. Lean software development shares principles with agile regarding people management and leadership, the focus on quality and technical excellence, and the focus on frequent and fast delivery of value to the customer. What distinguishes lean from agile is the focus on the end to end perspective of the whole value flow through development, i.e. from very early concepts and ideas to delivery of software features. To support the end to end focus lean proposed a number of tools to analyze and improve the value flow, such as value stream mapping [10], inventory management [7], and pull systems such as Kanban [11]. Value stream mapping visualizes the development life-cycle showing processing times and waiting times. Inventory management aims at limiting the work in process as partially done work does not provide value. Methods to support inventory management are theory of constraints [7] and queuing theory [12]. Push systems allocate time for requirements far ahead and often overload the development process with work. In contrast pull systems allow software teams to take on new tasks whenever they have free capacity. If possible, high priority tasks should be taken first.

Middleton [13] conducted two industrial case studies on lean implementation in software engineering, and the research method used was action research. The company allocated resources of developers working in two different teams, one with experienced developers (case A) and one with less experienced developers (case B). The practitioners responded that initially the work was frustrating as errors became visible almost immediately and were to be fixed right away. In the long run though the number of errors dropped dramatically. However, after using the lean method the teams were not able to sustain it due to organizational hierarchy, traditional promotion patterns, and the fear of forcing errors into the open.

Another case study by Middleton et al. [14] studied a company practicing lean in their daily work for two years. They found that the company had many steps in the process not adding value. A survey among people in the company showed that the majority supported lean ideas and thought they can be applied to software engineering. Only a minority (10 %) was not convinced of the benefits of lean software development. Statistics collected at the company showed a 25 % gain in productivity, schedule slippage was reduced to 4 weeks from previously months or years, and time for defect fixing was reduced by 65 % - 80 %. The customer response on the product released using lean development was overwhelmingly positive.

Perera and Fernando [15] compared an agile process with a hybrid process of agile and lean in an experiment involving ten student projects. One half of the projects was used as a control group applying agile processes. A detailed description of how the processes differed and which practices were actually used was not provided. The outcome was that the hybrid approach produced more lines of code and thus was more productive. Regarding quality, early in development more defects were discovered with the hybrid process, but the opposite trend was found in later phases, which confirms the findings in [13].

Parnell-Klabo [16] followed the introduction of lean and documented lessons learned from the introduction. The major obstacles were to obtain open office space to locate teams together, gain executive support, and training and informing people to reduce resistance of change. After successfully changing with the help of training workshops and the use of pilot projects positive results were obtained. The lead-time for delivery was decreased by 40% - 50%. Besides having training workshops and pilots and sitting together in open office-landscapes, having good measures to quantify the benefits of improvements are key.

Overall, the results show that lean principles may be beneficial in a software engineering context. Thus, further evaluation of lean principles is needed to understand how they affect the performance of the software process. Though, the studies did not provide details on which lean principles and tools were used, and how they were implemented. None of the studies focused on evaluating specific methods or principles of the lean tool-box. However, in order to understand how specific principles and methods from lean manufacturing are beneficial in software engineering, they have to be tailored and evaluated. This case study makes a contribution by implementing cumulative flow diagrams in industry, defining measures to analyze the development flow, and evaluating the cumulative flow diagrams and defined measures.

2.2. Lean Performance Measures in Manufacturing

A number of lean process measures have been proposed for manufacturing. Maskell and Baggaley [17] summarize performance measures for lean manufacturing. As this paper is related to measuring the flow of development we focus the related work on measurement of throughput:

- *Day-by-the-Hour (DbtH)*: Manufacturing should deliver at the rate the customers demand products. This rate is referred to as takt-time. The measure is calculated as

$$DbtH = \frac{\#quantity}{\#hours} \tag{1}$$

  where the quantity is the number of items produced in a day, and the hours the number of hours worked to produce the units. The rate should be equal to the takt-rate of demand.
- *Capacity utilization (CU)*: The work in process (WIP) is compared to the capacity (C) of the process. CU is calculated as:

$$CU = \frac{WIP}{C} \tag{2}$$

  If $CU > 1$ then the work-load is too high, and if $CU < 1$ then the work-load is too low. A value of 1 is ideal.

- *On-time-delivery (OTD)*: Delivery precision is determined by looking at the number of late deliveries in relation to the total deliveries ordered:

$$OTD = \frac{\#late\ deliveries}{\#deliveries\ ordered} \tag{3}$$

2.3. Lean Performance Measures in Software Engineering

Anderson [7] presents the measures cost efficiency and value efficiency, as well as descriptive statistics to analyze the flow in software development. From a financial (accounting) perspective he emphasizes that a cost perspective is not sufficient as cost accounting assumes that value is always created by investment. However, this ignores that cost could be waste and does not contribute to the value creation for the customer. In contrast to this throughput accounting is more sufficient to measure performance as it explicitly considers value creation.

In cost accounting one calculates the cost efficiency (CE) as the number of units delivered (LOC) divided by the input person hours (PH). Though, this assumes that there is a linear relationship between input and output, meaning that the input is invariable [7]. However, developers are not machines as they are knowledge workers. Therefore, this equation is considered insufficient for software production.

$$CE = \frac{\Delta LOC}{PH} \tag{4}$$

Therefore, the value created over time is more interesting from a lean perspective, referred to as value efficiency (VE). It is calculated as the difference of the value of output V_{output} and value of input V_{input} within the time-window Δt (see Equation 5) [7]. The input V_{input} represents the investment to be made to obtain the unit of input to be transformed in the development process, the value V_{output} represents the value of the transformed input, i.e. final product. Furthermore, V_{input} considers investment in tools and equipment for development.

$$VE = \frac{V_{output} - V_{input}}{\Delta t} \tag{5}$$

In addition to throughput accounting Anderson [7] presents descriptive statistics to evaluate lean performance. Plotting the cumulative number of items in inventory in different phases helps determining whether there is a continuous flow of the inventory. How to implement and use this technique as a way to determine process performance in incremental development is shown in the next section.

3. Visualization and Measures

This section presents our solution for measuring the flow of lean software development based on cumulative flow diagrams. First, cumulative flow diagrams are introduced and thereafter the measures quantifying the diagrams are presented.

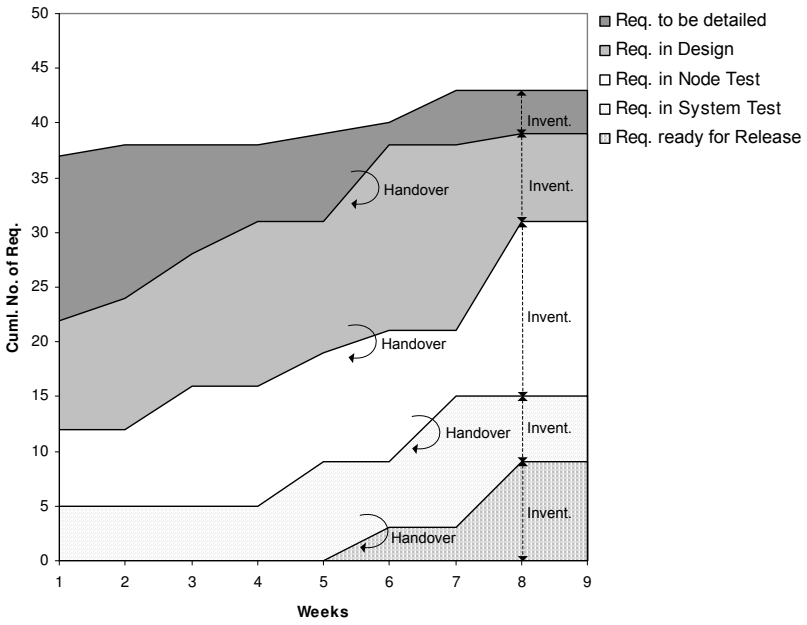


Figure 1. Cumulative Flow Diagram for Software Engineering

3.1. Visualization with Cumulative Flow Diagrams

Cumulative flow diagrams show how many units of production travel through the manufacturing process in a specific time window. In software engineering, it is of interest to know how many requirements are processed in a specific time window for a specific phase of development. A sketch of this is shown in Figure 1 with terms from the studied organization.

The x-axis shows the time-line and the y-axis shows the cumulative number of requirements having completed different phases of development. For example, the line on the top represents the total number of requirements in development. The line below that represents the total number of requirements for which detailed specification is finished and that were handed over to the implementation phase. The area in between those two lines is the number of incoming requirements to be detailed.

Looking at the number of requirements in different phases over time one can also say that the lines represent the hand-overs from one phase to the other. For example, in week six a number of incoming requirements is handed over to implementation, increasing the number of requirements in implementation. If the implementation of the requirement is finished, it is handed over to the node test (in this case in week 7). In the end of the process the requirements are ready for release.

Inventory is defined as the number of requirements in a phase at a specific point in time. Consequently, the difference of the number of requirements (R) in two phases (j and $j+1$) represents the current inventory level at a specific point in time (t), as shown by the vertical arrows in week 8 in Figure 1. That is, the inventory of phase j (I_j) in a specific point in time t is calculated as:

$$I_{j,t} = R_{j,t} - R_{j+1,t} \tag{6}$$

3.2. Measures in Flow Diagrams

The flow diagrams were analyzed with the aim of arriving at useful performance measures to increase throughput. Before presenting the actual measurement solution we show how we constructed the measures with the aid of the GQM approach [18]. The GQM approach follows a top-down strategy. First, goals were defined which should be achieved. Thereafter, questions were formulated that have to be answered to achieve the goals. In the last step, the measures were identified that need to be collected in order to answer the question. The GQM was executed as follows:

1. The company drove the analysis by identifying the goals to improve throughput in order to reduce lead-time and improve the responsiveness to customer needs.
2. Based on the goals the two authors of the paper individually derived questions and measures considering the goal and having the data of the cumulative flow graphs available. Thereafter, the authors discussed their measures and agreed on the questions and measurements to present to the case company. Both authors identified similar measures.
3. The measures as well as the results of their application were presented to an analysis team responsible for implementing lean measurement at the company. In the meeting with the analysis team the measures were discussed and feedback was given on the measures in an open discussion (see Section 4.5.2). The reception among the practitioners regarding the cumulative flow diagrams was also positive.

The goals identified in the first step were:

G1: increase throughput in order to reduce lead-times and improve responsiveness to customers' needs. Throughput is important due to dynamic markets and rapidly changing customer requirements. This is specifically true in the case of the company we studied. In consequence, started development work becomes obsolete if it is not quickly delivered to the customer.

G2: show the current progress/status of software product development. Showing the current status and progress of development in terms of flow allows management to take corrective actions for improving the flow.

Related to these goals the following questions were used to identify the measures:

Q1: Which phase in the development flow is a bottleneck? A bottleneck is a single constraint in the development process. Resolving this constraint can significantly increase the overall throughput. Bottleneck detection allows to continuously improve throughput by applying the

following steps: (1) identify the constraint, (2) identify the cause of the constraint, (3) remove the constraint, (4) go to step (1) [7].

Q2: How even is the workload distributed over time in specific phases? Workload throughout the development life-cycle of the software development process should be continuous. That means, one should avoid situations where requirements are handed over between phases in large batches (see, for example, the hand-over in Figure 1 where a large batch is handed over in week seven from implementation to node test). Large batches should be avoided for two main reasons. First, when having large batches there has to be a time of little activity before the batch is handed over. That means, defects introduced during that time are not detected immediately as they have to wait for the hand-over to the next phase for detection. Defects that are discovered late after their introduction are expensive to fix because investigating them is harder [5, 7]. Secondly, one would like to avoid situations where phases (e.g. requirements, coding, or testing) are overloaded with work at one time, and under-loaded at another time. As discussed in [14, 5, 19] early fault detection and a continuous work-load are an integral part of lean as well as agile software development to assure the reduction of waste, and quick responsiveness to customer needs.

Q3: Where can we save cost and take work-load off the development process? This question connects directly to waste, showing in which part of the development life-cycle waste has been produced. The focus of improvement activities should be on the removal of the most significant type of waste.

We would like to point out that RQ1 and RQ2 are not the same thing from a theoretical perspective. It is possible to have a similar throughput of requirements in two phases, but still one phase can deliver continuously while the other delivers in batches. This is further discussed in Section 6.

3.2.1. Measures to Quantify Bottlenecks

The derivation of metrics is driven by the questions formulated before. For each question (and by looking at the diagrams) we identified the measures.

Q1: Which phase in the development flow is a bottleneck? A bottleneck would exist if requirements come into one phase (phase j) in a higher rate than they can be handed over to the next phase (phase $j+1$). In Figure 1 an example of a bottleneck can be found. That is, the rate in which requirements are handed over from implementation to node test seems to be higher than from node test to system test. This indicates that the node test phase is a bottleneck. To quantify the bottleneck we propose linear regression to measure the rate of requirements flow for each phase. Linear regression models with two variables are used to predict values for two correlated variables. In our case the variables are *Weeks* and *Cumulative Number of Requirements*. The linear function represents the best fit to the observed data set. To determine the linear function the least square method is commonly used [20]. Furthermore, when the linear regression model is created, it can be observed that there is a difference between the actual observations and the regression line, referred to as the estimation error ϵ. This leads to the following formula:

$$y = f(x) = \beta_0 + \beta_1 * x + \epsilon \tag{7}$$

When conducting the actual prediction of the parameters, β_0 and β_1 are estimated as those represent the linear regression function. For the analysis of the bottlenecks the predicted variable β_1 is important, representing the slope of the linear functions. The measure for the bottleneck is thus defined as follows: If the slope of phase j (slope is referred to as β_j) is higher than the slope of the subsequent phases (β_{j+p}) then phase j is a bottleneck in the process. Though, it is important to note that the cause of the bottleneck is not necessarily to be found in phase j. To show the results of the measurement to management we propose to draw bar-plots of the slope which are well suited to illustrate the severity of the difference between phases.

Example: Let's assume a situation where the current inventory level of requirement in the specification phase S is 10 ($I_S = 10$). The regression of the incoming requirements of the last four weeks was $\beta_{inc,S} = 3$, and for the outgoing requirements $\beta_{out,S} = 1$. This leads to the following functions over time ($t = weeks$) for incoming and outgoing requirements: $f_{inc,S}(t) = 3t+10$ and $f_{out,S}(t) = 1t+10$. If in the following four weeks the rates do not change then the inventory level (and with that work in process) will almost double, as $f_{inc,S}(4) = 22$ and $f_{out,S}(4) = 14$. This leads to a new inventory level of $I_S = 18$. Depending on the available capacity this might lead to an overload situation indicating that actions have to be taken to avoid this rate of increasing inventory.

3.2.2. Measures to Quantify Workload Distribution Over Time

Q2: How even is the workload distributed over time in specific phases? Examples for an uneven work-flow can also be found in Figure 1. Continuous flow means that the number of requirements handed over at different points in time varies. For example, high numbers of requirements were handed over from incoming requirements to implementation in weeks two to four and week five, while there is few hand-overs in the remaining weeks. In comparison the variance of the flow of incoming requirements is lower. In order to quantify how continuous the flow of development is we propose to use the estimation error ϵ. The estimation error represents the variance around the prediction line of the linear regression, i.e. the difference between the observed and predicted values. The estimation error ϵ_i is calculated as follows:

$$\epsilon_i = y_i - \hat{y}_i \tag{8}$$

For the analysis the mean estimation error is of interest. In the case of our measurements i in the equation would be the week number. The estimation error should also be plotted as bar-plots when shown to management as this illustrates the severity of differences well.

Example: Let's assume that the variance in hand-overs from the specification phase to the development teams was very high in the last few month based on the average of estimation errors ϵ. Hence, hand-overs come in batches (e.g. in one week 15 requirements are handed over, while in the previous weeks only 3 requirements were handed over). High variances make the development harder to predict. In a perfectly even flow all observed data points would be on the prediction line, which would only be the case in a completely predictable environment.

Table I.
Costs

Phase (j)	Investment (I)	Work Done (WD)	Waste (W)	Cost (C)
$j = 1$	$r_{1,I}$	$r_{1,WD}$	$r_{1,W}$	$r_{1,C}$
$j = 2$	$r_{2,I}$	$r_{2,WD}$	$r_{2,W}$	$r_{2,C}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$j = n$	$r_{n,I}$	$r_{n,WD}$	$r_{n,W}$	$r_{n,C}$
all	$\sum_{j=1}^{n} r_{j,I}$	$\sum_{j=1}^{n} r_{j,WD}$	$\sum_{j=1}^{n} r_{jW}$	$\sum_{j=1}^{n} r_{j,C}$

3.2.3. Measures of Cost Related to Waste

Q3: Where can we save cost and take work-load off the development process? In order to save costs the costs need to be broken down into different types. We propose to break them down into investment (I), work done (WD), and waste (W) at a specific point in time (i) and for a specific phase (j), which leads to the following equation:

$$C_{i,j} = I_{i,j} + WD_{i,j} + W_{i,j} \tag{9}$$

The components of the cost model are described as follows:

- *Investment (I):* Investment is ongoing work that will be delivered to the next phase in the future (i.e. considered for upcoming increments to be released to the market). As long as it is potentially delivered it will be treated as investment in a specific phase j.
- *Work done (WD):* Work done is completed work in a phase, i.e. what has been handed over from phase j to phase $j + 1$.
- *Waste (W):* Waste is requirements that are discarded in phase j. That means work has been done on them, but they are not further used in the future; i.e. they will never make it into a release to customers.

An overview of a cost table for one specific point in time is shown in Table I. For each phase j at time i and type of cost (I, WD, and C) the number of requirements is shown. At the bottom of the table the sums of the types of cost is calculated across phases ($j = 1, ...n$).

When analyzing a time-frame (e.g. month) the average waste for the time-frame should be calculated. For example, if the points in time $(i, \ldots, m)$ are weeks in a given interval ($i = 1$ being the first week of the interval, and m being the last week) we calculate the average across phases as:

$$C_{avg,all} = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} r_{i,j,I}}{m} + \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} r_{i,j,WD}}{m} + \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} r_{i,j,W}}{m} \tag{10}$$

For an individual phase j we calculate:

$$C_{avg,j} = \frac{\sum_{i=1}^{m} r_{i,j,I}}{m} + \frac{\sum_{i=1}^{m} r_{i,j,WD}}{m} + \frac{\sum_{i=1}^{m} r_{i,j,W}}{m} \tag{11}$$

This summary allows to observe the average distribution of I, WD, and W within the selected time-frame for the analysis. In order to present the data to management, we propose to also calculate the percentages to show the distribution between types of costs. Besides being a detector for waste, this information can be used as an indicator for progress. If the investment is always much higher than the work-done in consecutive phases that means that the investment is not transferred into a work-product for the investigated phase(s).

Example: Let's assume a cost distribution for the implementation phase of $C_{imp} = 9 + 100 + 200$. The distribution shows that few new requirements are available as input for implementation phase ($I = 9$), and that implementation has completed most of the work ($WD = 100$). Consequently there is a risk that the implementation phase does not have enough input to work with in the future. Furthermore, the cost distribution shows that the majority of the requirements was discarded ($W = 200$). For the requirements and implementation phase this means that work done (requirements specification, review, coding and unit testing) on the discarded requirements does not end up in the end product. Too much of the cost is put on requirements that are never delivered and it shows that actions have to be taken in the future to minimize these types of costs.

4. Research Method

Candidate research methods for our research questions were design science [21], action research [22], and case study [23, 8]. Design science proposes that artifacts (in this case measures) should be created and evaluated. One way to evaluate the artifacts is case study. Hence, design science and case study complement each other. An alternative to case study is action research, also suitable to answer our research questions. However, this would require the researcher to work with the teams and managers using our proposed solution continuously which was not feasible. In consequence, the research method used is an embedded case study allowing to observe and understand the usefulness of the proposed solution. Embedded means that within the case (Ericsson) different embedded units (the process flow for different sub-systems being developed) were analyzed. The design of the case study is strongly inspired by the guidelines provided in Yin [8] and Runeson and Höst [9]. We used Yin's guidelines in identifying the case and the context, as well as the units of analysis. Yin provides an overview of potential validity threats relevant for this case study and stresses the importance of triangulation, i.e. to consult different data sources, such as documentation, quantitative data and qualitative data from workshops, interviews, or observations.

4.1. Research Context

Ericsson AB is a leading and global company offering solutions in the area of telecommunication and multimedia. Such solutions include products for telecommunication operators, multimedia solutions and network solutions. The company is ISO 9001:2000 certified. The market in which the company operates can be characterized as highly dynamic with high innovation in products and solutions. The development model is market-driven, meaning that the requirements are collected from a large base of potential end-customers without knowing exactly who the

customer will be. Furthermore, the market demands highly customized solutions, specifically due to differences in services between countries. The following agile practices are used: continuous integration, internal and external releases, time-boxing with sprints, face-to-face interaction (stand-up meetings, co-located teams), requirements prioritization with metaphors and detailed requirements (digital product backlog), as well as refactoring and system improvements. Version control is handled with ClearCase [24] and TTCN3 [25] is used for test automation. Documentation and faults are tracked in company proprietary tools.

4.2. Case Description

The case being studied was Ericsson in Sweden and India. On a high level all systems were developed following the same incremental process model illustrated in Figure 2. The numbers in the figure are mapped to the following practices used in the process.

- *Prioritized requirements stack (1) and anatomy plan (2):* The company continuously collects requirements from the market and prioritizes them based on their importance (value to the customer) and requirements dependencies (anatomy plan). Requirements highest in the priority list are selected and packaged to be implemented by the development teams. Another criterion for packaging the requirements is that they fit well together. The anatomy plan also results in a number of baselines (called last system versions, LSV) and determines which requirement packages should be included in different baselines.
- *Small projects time line (3):* The requirements packages are handed over to the development projects implementing and unit testing the increments of the product. The projects last approximately three months and the order in which they are executed was determined in the previous two steps.
- *Last system version (4):* As soon as the increment is integrated into the sub-system a new baseline is created (LSV). Only one baseline exists at one point in time. The last version of the system is tested in predefined testing cycles and it is defined which projects should be finished in which cycle. When the LSV phase has verified that the sub-system works and passed the LSV test with the overall system the sub-system is ready for release.
- *Potential release (5):* Not every potential release has to be shipped to the customer. Though, the release should have sufficient quality to be possible to release to customers.

Overall the process can be seen as a continuously running factory that collects, implements, tests, and releases requirements as parts of increments. When a release is delivered the factory continuous to work on the last system version by adding new increments to it. The sub-systems used for illustration in the paper have been around for more than five years and have been parts of several major releases to the market.

4.3. Units of Analysis

In total the flow of nine sub-systems developed at the case company were analyzed with the flow diagrams, each sub-system representing a unit of analysis. Two of the sub-systems are

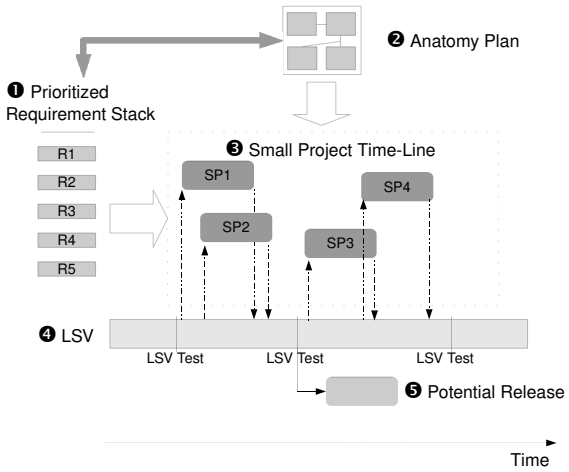


Figure 2. Incremental Process Model

presented in this paper representing different patterns. The difference in flows helps (1) to identify patterns that are worthwhile to capture in the measures; and (2) to derive more general measures that can be applied to a variety of flows. In addition to the two individual sub-systems we conducted a combined analysis of all nine sub-systems studied at the case company. The sub-systems have been part of the system for more than five years and had six releases after 2005. The sub-systems vary in complexity and the number of people involved in their development.

1. *Unit A*: The sub-system was developed in Java and C++. The size of the system was approximately 400,000 LOC, not counting third party libraries.
2. *Unit B*: The sub-system was developed in Java and C++, the total number of LOC without third party libraries was 300,000.

The nine sub-systems together had more than 5,000,000 LOC. The development sites at which the sub-systems were developed count more than 500 employees directly involved in development (including requirements engineering, design, and development) as well as administration and configuration management.

4.4. Research Questions

We present the research questions to be answered in this study, as well as a motivation why each question is of relevance to the software engineering community.

- *Research question 1 (RQ1): Which measures aid in (1) increasing throughput to reduce lead-times, and (2) as a means for tracking the progress of development?* The aim is

to arrive at measures that are generally applicable and suitable to increase throughput and track the progress of development. As argued before, throughput and lead-times are highly important as customer needs change frequently, and hence have to be addressed in a timely manner. Thus, being able to respond quickly to customer needs is a competitive advantage. In addition knowing the progress and current status of complex software product development should help to take corrective actions when needed, and identify the need for improvements.

- *Research question 2 (RQ2): How useful are the visualization and the derived measures from an industrial perspective?* There is a need to evaluate the visualization (cumulative flow diagrams) and the measures in industry to provide evidence for their usefulness. For this purpose the following sub-questions are addressed in the evaluation part of this study:
  - *Research question 2.1: How do the visualization/measures affect decision making?*
  - *Research question 2.2: What improvement actions in the development processes do practitioners identify based on the visualization and measures?*
  - *Research question 2.3: How should the visualization and the measures be improved for practical use?*

4.5. Data Collection

4.5.1. Collection of Data for Visualization and Measurements

The current status (i.e. the phase in which the requirements resides) was maintained in a database. Whenever the decision was taken to move a requirement from one phase to the next this was documented by a time-stamp (date). With this information the inventory (number of requirements in a specific phase at any point in time) could be calculated.

After the initial data entry documenting the current status of requirements the main author and the practitioners reviewed the data for accuracy and made updates to the data when needed. From thereon the practitioners updated the data continuously keeping it up-to-date. To assure that the data was updated on a regular basis, we selected practitioners requiring the status information of requirements in their daily work and who were interested in the visualization results (cumulative flow diagrams) and measurements. The following five inventories were measured in this study:

- *Number of incoming requirements:* In this phase the high level requirements from the prioritization activity (see practice 1 in Figure 2) have to be detailed to be suitable as input for the design and implementation phase.
- *Number of requirements in design:* Requirements in this phase need to be designed and coded. Unit testing takes place as well. This inventory represents the requirements to be worked on by the development teams, corresponding to practice 3 in Figure 2. After the requirements are implemented they are handed over to the LSV test for system testing.
- *Number of requirements in LSV test (node and system):* These inventories measure the number of requirements that have to be tested in the LSV. The LSV test is done in two steps, namely node LSV (testing the isolated sub-system) and system LSV (testing the

integration of sub-systems) measured as two separate inventories. When the LSV test has been passed the requirements are handed over to the release project. This inventory corresponds to practice 4 in the process shown in Figure 2.

- *Number of requirements available for release:* This inventory represents the number of requirements that are ready to be released to the customer. It is important to mention that the requirements can be potentially released to the customer, but they do not have to (see Practice 5 in Figure 2). After the requirements have been released they are no longer in the inventories that represent ongoing work.

Only ongoing work was considered in the analysis. As soon as the requirements were released they were removed from the diagrams. The reason is that otherwise the number of requirements would grow continuously, making the diagrams more or less unreadable and harder to compare over time.

4.5.2. Collection of Qualitative Data on Usefulness of Visualization and Measures

The result of the evaluation answered research question 2, including the two sub-questions to be answered (research question 2.1 and 2.2).

Research question 2.1: In order to answer research question 2.1 a workshop was conducted by an external facilitator (a consulting company represented by three consultants running the workshop), where the researcher acted as a participant and observer in the workshop. In addition to identifying the effect of the measures on the decision making in the company, the workshop was also used to reflect on possible improvements and measures complementary to the ones identified in this study. During the workshop the participants first noted down the roles that are affected by the measures. Those were clustered on a pin-board and the roles were discussed to make sure that everyone had the same understanding of the responsibilities attached to the identified roles. Thereafter, each participant noted down several effects that the measures have on the decision makers. The effects were discussed openly in the workshop. The researcher took notes during the workshop. In addition to that the external facilitator provided protocols of the workshop session.

Research question 2.2 and 2.3: These research questions were answered by participating in regular analysis meetings run by the company once or twice a month. The purpose of these meetings was to reflect on the usefulness of the measures as well as on the actual results obtained when measuring the flow of the company. During the meetings it was discussed (1) how the measurement results can be interpreted and improved, and (2) what improvement actions can be taken based on the measurement results. The researcher took an active part in these discussions and documented the discussions during the meetings.

The roles participating in the workshop and the meetings were the same. The participants of the meeting and workshop all had management roles. An overview of the roles, and the number of participants filling out each roles are shown in Table II. The criteria for selecting the participants of the workshop and meetings were (1) good knowledge of the processes of the company, and (2) coverage of different departments and thus disciplines (i.e. requirements, testing, and development).

Table II.
Roles

Role	Description	No. of Persons
Process Improvement Driver	Initiate and Monitor SPI activities	2
Project Manager	Motivate teams, control projects, reporting	3
Program/Portfolio Manager	Prioritize implementation of requirements, request staffing for program development	2
Line Manager	Allocation of staff, planning of competence development	4

4.6. Data Analysis

The data analysis was done in three steps. In the first step the yellow notes were clustered on a high level into groups for statements related to the effect of the measures on decision making (RQ2.1), improvement actions identified based on the measures (RQ2.2), and improvements to measurements (RQ2.3). The clustering in the first step was done in the workshop. Within these groups further clusters were created. For example, clusters in the group for RQ2.1 were requirements prioritization, resources and capacity, and short-term as well as long-term improvements in decisions. The clusters were also compared to the notes of the facilitator running the workshop (see data collection approach for research question 2.1 in Section 4.5.2) to make sure that similar clusters were identified. For the meetings (see data collection approach for research question 2.2 and 2.3) the researcher and a colleague documented the outcome of the meetings, which also allowed to compare notes. In the second step the notes taken during the workshop and meetings were linked to the clusters. Based on these links each cluster was narratively described by the researcher. In the third and final step the analysis was reviewed by a colleague at the company who also participated in the workshop and meetings.

4.7. Threats to Validity

Threats to validity are important to consider during the design of the study to increase the validity of the findings. Threats to validity have been reported for case studies in Yin [8] and in a software engineering context in Wohlin et al. [26]. Four types are distinguished, namely construct validity, internal validity, external validity, and reliability. Internal validity is concerned with establishing a causal relationship between variables. External validity is concerned with to what degree the findings of a study can be generalized (e.g. across different contexts). Reliability is concerned with the replication of the study (i.e. if the results would be the same when repeating the study). Internal validity is not relevant for this study as we are not seeking to establish the casual relationship between variables in a statistical manner.

Construct validity: Construct validity is concerned with obtaining the right measures for the concept being studied. There is a risk that the researcher influences the outcome of the study with his presence in industry (reactive bias) [26]. This risk was reduced as the researcher is not perceived as being external as he is also employed by the company. Thus, the participants of the workshops and meetings did not perceive the researcher as an external observer and hence their behavior was not influenced by his presence. Another threat is the risk that the practitioners misinterpret the measurements and visualizations (incorrect data). In order to minimize the threat of incorrect data a tutorial was given on how to read and interpret the visualization and measurements. Furthermore, tutorial slides were provided to all users of our proposed solution. The practitioners were also given the possibility to ask clarification questions which aided in achieving a common understanding of the solution.

External validity/ generalizability: The main threat to external validity is that the study has been conducted in a single company. Consequently, the results were obtained in the unique environment in which the case is embedded, namely the context [27, 28, 8]. The threat to external validity was reduced by describing the context carefully (see Section 4.1) and by that make the degree of generalizability explicit [19, 27]. Hence, the results of this study are likely to be generalizable to companies working with large-scale development in an incremental and iterative manner, and developing software for a dynamic mass market. In order to employ our solution in other contexts the requirements inventories might differ depending on which other phases of development can be found in another company. In summary, it is always very hard to generalize a case study. However, the findings and experiences gained are expected to provide valuable inputs to others interested in applying a similar approach, although the specific context has always to be taken into account.

Reliability: When collecting qualitative data there is always a risk that the interpretation is affected by the background of the researcher. The threat is reduced because the notes of the researcher were compared to the notes of the external facilitator for the workshop, and with notes from a colleague for the meetings (see Section 4.5.2). Additionally, the analysis was reviewed by a colleague at the company who also participated in the workshop and meetings. Possible misunderstandings are further reduced due to that the researcher has good knowledge of the processes at the company and thus understands the terminology (cf. [29]). The comparison of the notes showed that there were no conflicting statements in the notes.

5. Results

The quantitative results (visualization and measurements) are illustrated first. Thereafter, the qualitative evaluation of the usefulness of visualization and our proposed measures is presented.

5.1. Application of Visualization and Measures

We applied the measures on data available from the company. The following data is provided for this: (1) the original graphs including the regression lines to visualize the relation between original data and metrics collected; (2) bar-plots of the slope to illustrate the bottlenecks; (3)

bar-plots of the estimation error to illustrate how even the flow is; (4) a table summarizing costs as I, WD, and W.

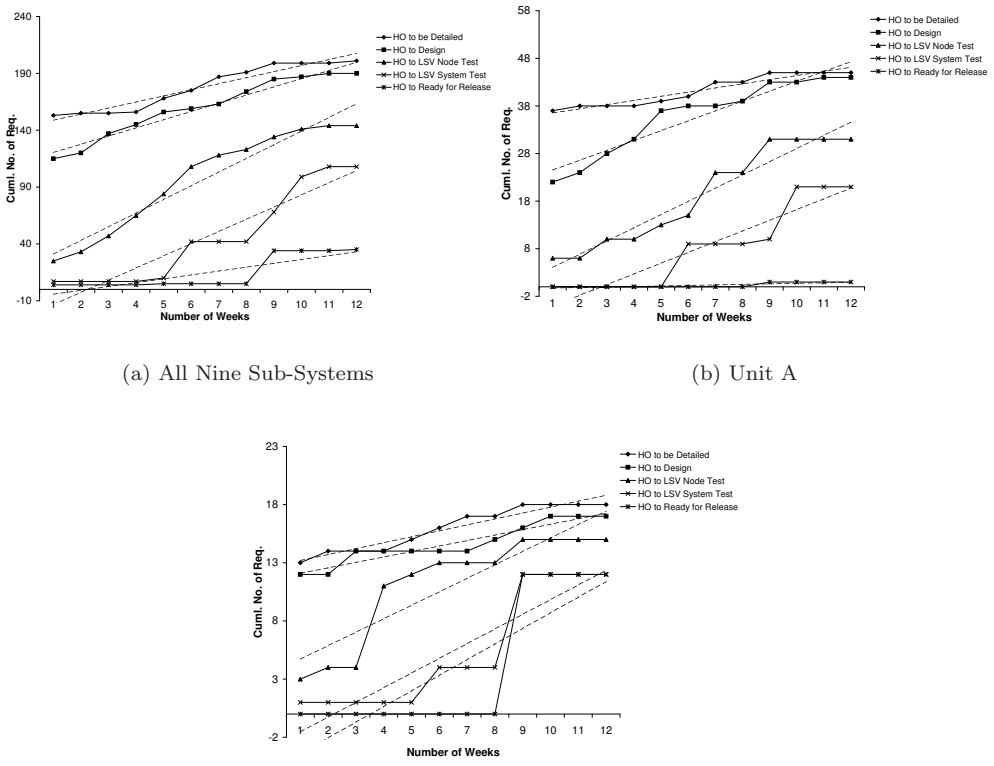
5.1.1. Bottlenecks and Even Flow

Figure 3 shows the cumulative flow diagrams for all nine sub-systems combined and the considered units of analysis, including the regression lines. As discussed before the regression lines are a measure of the rate in which requirements are handed over between the phases. The purpose of the figures is to show how the visualization of the original data (cumulative flow diagrams) and the metrics collected are connected. Drawing the regression lines in combination with the original data has advantages.

Measurement makes visualization more objective: The sole visualization of the cumulative flow diagram does not always reveal the real trend as it is not easy to recognize which of the phases that has a higher rate of incoming requirement. A high variance around the regression line makes this judgment even harder. Therefore, it is important to calculate the regression. An example is shown in Figure 3(b), where the curves for requirements to be detailed (HO to be Detailed) and requirements to be implemented (HO to design) have a fairly similar trend. However, the best fit of the data (regression) makes explicit that the slope of requirements to be detailed is higher than the slope of requirements to be implemented, which (according to our measures) would be an indicator for a bottleneck. Thus, the measurement can be considered more objective in comparison to the sole visualization.

Identifying bottlenecks: All figures show examples of bottlenecks, the main bottleneck (for the designated time-frame) was the node testing phase (i.e. slope HO to LSV Node test > slope HO to LSV System test). The figures also show opposite trends to bottlenecks, i.e. the rate in which requirements come into a phase was lower than the requirements coming out of the phase (e.g. slope HO to Design < slope HO to LSV Node Test 3(c)). Such a trend can be used as a signal for recognizing that there is not enough investment (or buffer) in this phase to make sure that the forthcoming phase has input to work with in the future. Thus, there is potentially free (and unused) capacity within the process. In order to show the significance of bottlenecks to management, we proposed to draw bar-plots. The bar plots illustrating the slope (value of β) for the different hand-overs and systems are shown on the left side of Figure 4. The hight of the bars shows the significance of the bottlenecks. For example, for all systems (Figure 4(a)) as well as unit A (Figure 4(c)) and B (Figure 4(c)) the rate in which requirements are handed over to system test was almost four times higher than the rate the requirements are handed over to the release, which indicates that the work in process before system test will grow further. In consequence an overload situation might occur for system test. In comparison, the bottleneck in the LSV node test was less significant in Figures 4(a) and 4(c). It is also important to mention that a high slope is desired, which implies high throughput.

Evaluating even flow: The highest variances can be found in the HO to LSV system test for all systems, as shown in the graphs on the right side of Figure 4. That is, there was a high deviation between the regression line and the observed data. As discussed before, this indicates that a higher number of requirements was delivered at once (e.g. for all systems quite a high number was delivered between week 8 and 11, while there was much less activity in week 1-7, see Figure 3). Figures 4(b), 4(d) and 4(f) show the significance of the variances. It is



(a) All Nine Sub-Systems

(b) Unit A

(c) Unit B

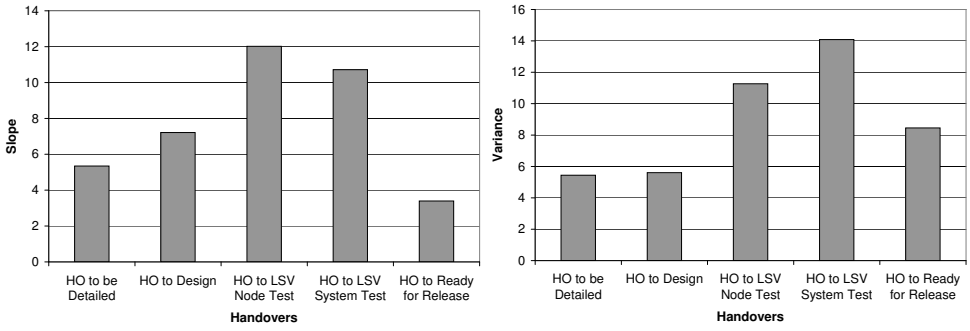
Figure 3. Regression Analysis

clear that the highest variances can be found in the implementation and testing phases, while the requirements phase is characterized by lower variances. This is true for the situation in all three figures (4(b), 4(d) and 4(f))

Lower variances can be found in the phases where the requirements are defined (i.e. HO to be detailed, and HO to Design).

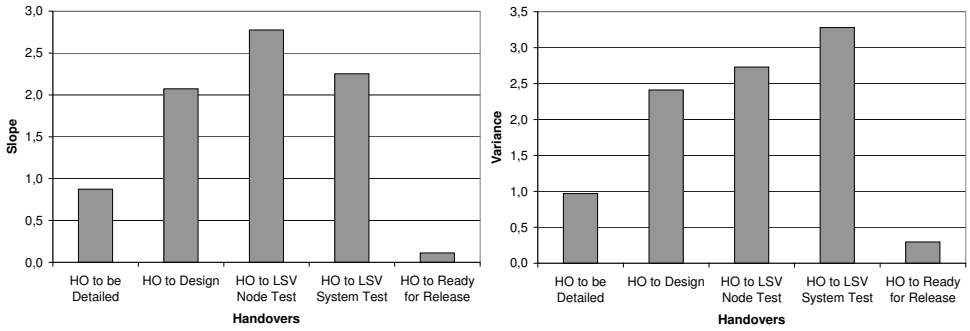
5.1.2. Distribution of Costs

The distribution of costs for all nine sub-systems is shown in Table III. The costs were calculated for each phase ($C_{avg,j}$) and across phases ($C_{avg,all}$). The value of waste (W) was 0 in all phases as no requirements were discarded for the analyzed data sets. The data in the table



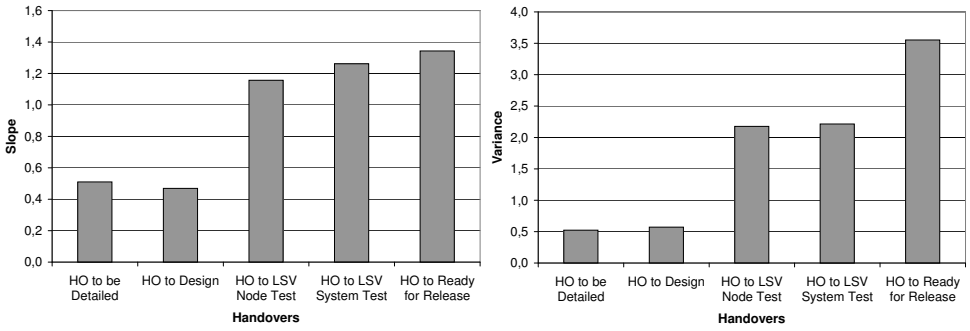
(a) All Nine Sub-Systems Slope

(b) All Nine Sub-Systems Variance



(c) Unit A Slope

(d) Unit A Variance



(e) Unit B Slope

(f) Unit B Variance

Figure 4. Slope and Variance

Table III. Costs All

Phase	I	I (%)	WD	WD (%)	W
$C_{avg,Inc.Req.}$	18.00	10.15	160.00	89.85	0.00
$C_{avg,Design}$	62.92	39.30	97.17	60.70	0.00
$C_{avg,LSVNode}$	51.58	53.09	45.58	46.91	0.00
$C_{avg,LSVSys}$	31.17	68.37	14.42	31.63	0.00
$C_{avg,ReadyRelease}$	14.42	100.00	0.00	0.00	0.00
$C_{avg,all}$	35.63	35,69	63.45	64.04	0.00

shows that in the early phases (incoming requirements till requirements to be defined), there was little investment (I) in comparison to work done (WD). That means that investments were transferred into work-done, which is an indicator of progress. However, if there is only little investment in a phase it is important to create new investments (e.g. increasing the number of requirements to be detailed by focusing more on requirements elicitation). Otherwise, the company might end up in a situation where their developers and testers are not utilized due to a lack of requirements (although highly unlikely). From a progress perspective, this analysis looks positive for the early phases. The later phases indicate that the investment was higher than the work done. That means, from a progress perspective most investments in the phase LSV node and LSV System still have to be transferred to work done.

In the long run we expect requirements to be discarded (e.g. due to changes in the needs of customer, or that requirements are hold up in a specific phase). If waste becomes a significant part of the cost distribution, then the reasons for this have to be identified. For example, only requirements with a certain priority should be transferred to a future phase to assure their timely implementation.

5.2. Industry Evaluation of Visualization and Measures

Two main aims were pursued with the evaluation. Firstly, we aimed at identifying the relevant roles in the organization that could make use of the measures in their decision making. This first aim helps understanding whether the measures are of use, and to whom they are most useful (Research Question 2.1). Secondly, we aimed at identifying process improvement actions based on the measures (Research Question 2.2). The second aim shows that the measures trigger practitioners in identifying potential improvement areas/actions, which is an indicator for that the measures serve as an initiator for software process improvement activities.

5.2.1. The Effect of Measurements on Decision Making (RQ2.1)

During the workshops with the company representatives, it was discussed how the measures can affect decision making in the company. In the following the different types of decisions supported by the measures are discussed.

Requirements prioritization: The measure of continuous work-flow may show a trend where too many requirements are handed over at once. Thus, the measure helps in assuring that requirements are not handed over in big bulks, but instead in smaller chunks and more continuously. Furthermore, if the measure signals that the development is overloaded, no more requirements should be handed over to avoid an overload situation. As the decisions are related to requirements prioritization the measures help improving from a short-term perspective.

Staff allocation: In case of bottlenecks managers can use the information from the bottleneck analysis to allocate staff when bottlenecks occur. As discussed during the workshop, this has to be done with care, as the reason for a bottleneck might be something else than staffing. However, if staffing is the problem the figures support managers in arguing for the need of staff. Staffing can also be seen as an ad-hoc solution with a short-term perspective.

Transparency for teams and project managers: The measurement system allows seeing which requirements are coming into phases, which are currently worked on, and which have been completed. In consequence, the current status of development for each team is always visible. For instance, the testing team can see all requirements they are supposed to be working on in the upcoming weeks. This helps them in planning their future work as they are aware of future requirements. Furthermore, seeing what has been completed has a motivating effect on the teams.

Software process improvement: Software process improvement needs to look at the data from a long-term perspective (i.e. quarterly is not long enough, instead one should use data for 6 or 12 months). If there are significant bottlenecks or uneven requirements flows the causes for this have to be investigated. Thus, from a process improvement perspective focusing on long-term improvement the data should be used as an indicator of where to look for problem causes.

The result of the workshop was that the practitioners judged the measurements as useful for different roles in the organization, which supports their practical relevance. Furthermore, the measures can be used to take actions to improve in the short and the long term.

5.2.2. Process Improvement Actions Based on Measurements (RQ2.2)

In the analysis team meetings, process improvement actions were identified from a software process improvement perspective baed on the measures. The analysis was done on quarterly data as the data for half a year is not available yet. However, the analysis meetings already show that the information provided by the visualization and measurements aids the practitioners in identifying concrete areas for improvement. The most important improvement areas identified so far are (1) avoid to be too much driven by deadlines and market-windows; and (2) integrate more often to improve quality.

Observation and improvement for (1): The development is done continuously, but with a deadline (market-window) in mind. This is a way of thinking that leads to that work is started late before the deadline (e.g. the high amount of requirements handed over in week 8 to LSV system test in Figure 3(c), and the related high value in variance seen in Figure 4(f)). In consequence, there were fewer deliveries of requirements from one phase to the other until shortly before the deadline, where a big bulk of requirements was delivered at once. As an improvement we found that people should not focus on deadlines and market-windows too much when planning the flow of development. Instead, they should focus more on the

continuous production with optional releases. In order to achieve this, an option is to follow the Kanban approach. In Kanban, the work (e.g. maximum number of requirements with similar complexity) that can be in process at a specific point in time is limited [11]. In consequence, big bulk deliveries of requirements are not possible. Thus, the Kanban approach enforces to continuously work and deliver requirements between phases, reducing the variance in the flow.

Observation and improvement for (2): The practitioners observed that the system was integrated too late, and not often enough. An indication for this observation can be seen in the slope analysis of in Figures 4(a) and 4(c). Hence, the planning of testing cycles needs to enforce short periods between integration. The main purpose of this is to allow for regular feedback on the builds helping the company to further improve the quality of the products. In order to technically enable more frequent integration the company is pursuing a higher degree of testing automation.

5.2.3. Improvements to the Measures (RQ3.3)

The visualization and measures need to be further improved in order to increase their usefulness and accuracy. Together with the practitioners providing a critical reflection on the visualization and measures we identified the following improvements:

Treatment of quality requirements: Quality requirements do not flow through the development process in the same fashion as functional requirements. Functional requirements are part of an increment, and are completed with the delivery of the increment. Some quality requirements (e.g. performance) are always relevant for each new release of the product and thus always stay in the inventories. That means, one has to be aware that those requirements are not stuck in the inventory due to a bottleneck. Therefore, we recommend to remove requirements from the analysis that are always valid for the system.

Requirements granularity and value: If requirements vary largely in complexity it is not a valid approach to only count the requirements. One has to take the weight of each requirement into consideration. The weight should be based on the size of the requirements. We propose to estimate the size of the requirements in intervals for small, medium, and large requirements. What small, medium, and large means differs between organizations, meaning that each organization has to determine their own intervals. When counting the requirements, a small requirement is counted once, a medium requirement twice, and a large requirement thrice. Another distinguishing attribute of requirements is their value as suggested by the concept of minimum marketable requirements [30]. High value requirements should flow through the process more quickly and with higher throughput than low value requirements. Consequently, the solution would benefit from also weighting requirements based on their value.

Optimization of measures: Optimization of measures is always a risk when measurements are used to evaluate an organization. For example, it is possible to improve the rate of requirements hand-overs by delivering detailed requirements with lesser quality to implementation. We believe that this will be visible in the measures as design will not be able to work with the requirements and thus a bottleneck in design becomes visible. However, it is still beneficial to complement the evaluation of flow with other quality related measures. Possible candidates for quality measures are fault-slip through [31], or number of faults reported in testing and by the customer.

Visualization of critical requirements: The visualization does not show which requirements are stuck in the development process. In order to requirements into account that are stuck in the process one could define thresholds for how long a requirement should stay in specific phases. Requirements that are approaching the threshold should be alerted to the person responsible for them. In that way, one could pro-actively avoid that requirements do not flow continuously through the development.

Time frames: The time-frame used at the company for evaluating the flow is each quarter. As discussed with the practitioners, it is also important to have a more long-term perspective when evaluating the data (i.e. half-yearly and yearly). The data in the short-term (quarterly) is more useful for the teams, program/line-managers, and requirements/portfolio managers, who use the data from a short-term perspective to distribute resources or prioritize requirements (see answers to RQ2.1). However, the practitioners responsible for long-term improvements require measurements over a longer period of time to reduce the effect of confounding factors. Examples for confounding factors are Midsummer in Sweden, upcoming deadline of an important release, or critical fault report from customer.

The next section discusses practical implications and research implications of the visualization and measurements presented. Furthermore, the measures are compared to those presented in the related work section, which leads to a discussion of the difference in measuring lean manufacturing and lean software development.

6. Discussion

The application and evaluation of the visualization and the defined measures lead to some implications for both research and practice. In addition, we discuss how the measures relate to lean measurements used in manufacturing.

6.1. Practical Implications and Improvements to the Measures

The implications describe the relevance of the visualization and measures to industry and what evidence in the results (Section 5) supports the relevance.

Perceived usefulness of visualization and measures: The evaluation showed that the visualization and measures are useful from an industrial point of view. The measures were introduced in February 2009 and are mandatory to use in the evaluations of product development of the systems considered in this case study since July 2009 for the studied system developed in Sweden and India. The reason for this is that the measures were able to give a good overview of progress. This kind of transparency is especially important in complex product development where many tasks go on in parallel [32]. The rapid application of the solution and the commitment of the company to use the solution in their daily work is strong evidence of its perceived usefulness. The evaluation of the measures with regard to usefulness was very much based on the perception and feedback by the practitioners. From past experience we found that when transferring a new practice to industry the feedback and commitment from the practitioners is invaluable, the reason being that the practitioners can influence and improve the approach (as has been documented in the improvement suggestions made by the

practitioners) [33, 34]. Furthermore, a buy-in by the management is important for a long-term success, which can be achieved through feedback and building trust by incorporating that feedback.

Visualization and measures drive practitioners towards lean and agile practices: The proposed process improvement actions are an indication that the measures drive the organization towards the use of lean and agile practices. The evidence for the claimed driving force of the measures is the identification of lean and agile improvements that the company would like to adopt (see Section 5.2.2). The first improvement proposed is directly related to lean principles (Kanban) [11], stressing the importance to change from a push to a pull mechanism, and by that limiting the work in process [5]. The second improvement stresses the importance of frequent integration and early feedback to improve quality, which is an important principle in all agile development paradigms [2, 19]. Currently actions are in implementation at the company to support the pull approach from the requirements phase to development, and to enable earlier testing (e.g. changes in architecture to allow for continuous upgrades and earlier testing of these upgrades). The actual improvements will not be visible in the visualization and measurements immediately as a large-scale organization with more than 500 people directly and indirectly involved in the development of the investigated systems has been studied. Lean is a new way of thinking for many people in the organization who have to familiarize with the concepts, requiring training and getting a commitment to this new way of working throughout the organization. As pointed out by [35, 13] the introduction of lean software development is a continuous process and the attempt of a big-bang introduction often leads to failure.

Measures help in short-term and long-term decision making: The practitioners perceive the visualization and measurements as beneficial in supporting them in the short and the long term. Evidence is the identification of decisions by the practitioners that were related to long as well as short term decisions. (see Section 5.2.1). Three decisions were related to short term, i.e. requirements prioritization, staff allocation, and project management and planning decisions. Furthermore, undesired effects seen in the measures help in making decisions targeted on long-term process improvements. A prerequisite to take the right long-term decisions is to identify the causes for the undesired effects.

Understand complexity: The large system studied is highly complex with many different tasks going on in parallel, which makes transparency and coordination challenging [19]. Hence, measurement and visualization focusing on the end to end flow increases the transparency for the practitioners. In particular our solution was able to show the behavior of the development flow of a highly complex system in terms of bottlenecks, continuous flow, and distribution of costs.

Overall, the discussion shows that the results indicate that the measures can be useful in supporting software process improvement decisions.

6.2. Research Implications

The research implications provide examples of how other researchers can benefit from the results in this study.

Complement agile with lean tools: The characteristic that distinguishes lean and agile development is the focus on the flow of the overall development life-cycle. Lean provides

analysis and improvement tools focusing on the overall development life-cycle while agile focuses on solutions and prescribes sets of practices to achieve agility. Given the results of the case study we believe that the agile research community should focus on making lean analysis tools for the development flow an integral part of agile practices. We believe this to be particularly important in large-scale software development where many teams develop software in parallel, making it harder to achieve transparency of what is going on in a dynamic and agile environment.

Use solution to evaluate process improvements: The measures provided are potentially useful for researchers who would like to evaluate and test improvements increasing efficiency of development. For example, simulation models often focus on lead-time (cf. [36, 37]). Complementary to that the simulations could benefit from making use of the measures identified in this study to provide a solution allowing to analyze throughput with regard to bottlenecks, variance in flow, and distribution of cost.

6.3. Comparison with State of the Art

In the related work section, we presented measures that are applied in measuring lean manufacturing. However, in the context of software engineering the measures have drawbacks.

Throughput measures: Throughput measures calculate the output produced per time-unit. Three throughput measures have been identified in the related work, namely day-by-the-hour [17], cost efficiency [7], and value efficiency [7]. In the context of software engineering, this measure would determine the number of requirements completed per hour. However, this measure is very simplistic and does not take into account the variance in which requirements are completed. Therefore, we use regression in order to calculate the rate and also calculate the variance to evaluate how continuous the flow of development is.

Capacity utilization: In manufacturing it is predictable how many units a machine can produce, and thus the desired value of capacity utilization is clear (i.e. $CU = 1$) [17]. However, software developers are knowledge workers and their behavior is not as predictable. There is a high variance between developers in terms of productivity [38]. Furthermore, when thinking about concepts and creative solutions no output is produced in this time. This makes the measure unsuitable in the software engineering context.

On-time-delivery: On-time delivery [17] is tightly connected to deadlines in software development. However, in the case of incremental development one should not focus too much on a specific deadline, but on being able to continuously deliver a product with the highest priority requirements implemented [5].

The analysis of the related work show that there were no comprehensive measures to capture the flow of development. We addressed the research gap by proposing measures to detect bottlenecks, discontinuous flow, and distribution of costs. The measures are connected to an easy to understand visualization of the development flow, which aids in communicating with management.

7. Conclusion

In this study we applied cumulative flow diagrams to visualize the flow of requirements through the software development life-cycle. The main contribution of this study is a set of measures to achieve higher throughput and to track the progress of development from a flow perspective. The measures were evaluated in an industrial case study. In the following we present the research questions and the answers to the questions.

RQ1: Which measures aid in (1) increasing throughput to reduce lead-times, and (2) as a means for tracking the progress of development? Three metrics were identified in the context of this study. The first metric allows to identify bottlenecks by measuring the rate of requirements hand-over between different phases through linear regression. The second metric measures the variance in the hand-overs. If the variance of hand-overs is very high then big batches of requirements are handed over at once, preceded by a time-span of inactivity. The third metric separates requirements into the cost types investment, work done, and waste. The purpose of the measure is to see the distribution of requirements between the different types of cost.

RQ2: How useful are the visualization and the derived measures from an industrial perspective? This research question was evaluated from two angles. First, we evaluated how the measures can affect decision making. The findings are that: (1) requirements prioritization is supported; (2) the measures aid in allocating staff; (3) the measures provide transparency for teams and project managers of what work is to be done in the future and what has been completed; and (4) software process improvement drivers can use the measures as indicators to identify problems and achieve improvements from a long-term perspective. Secondly, we evaluated what improvement actions practitioners identified based on the measurements. The improvement areas are: (1) an increased focus on continuous development by limiting the allowed number of requirements in inventories; and (2) earlier and more frequent integration and system testing of the software system to increase quality. The solution has been successfully transfered to industry and will be continuously used at the company in the future.

In conclusion the case study showed that the visualization and measures are perceived as valuable from an industrial perspective. It should be emphasized that they are especially valuable when developing large scale products with many teams and tasks going on in parallel, as here transparency is particularly important.

Future work should focus on evaluating our solution in different contexts and how it has to be adjusted to fit these contexts. For example, cumulative flow diagrams and the suggested measures could be applied to analyze software maintenance, or software testing processes. In addition future work should focus on the analysis of what type of improvements support a lean software process, the measurements and visualizations proposed in this paper can be an instrument to evaluate such improvements from a lean software engineering perspective. Little is also known about the long-term effect of implementing agile practices such as Kanban, which is also an interesting area for future research.

From an analysis perspective other lean tools are available, such as value stream maps and theory of constraints/ queuing theory. The usefulness of these tools will allow to learn more about the benefits that could be achieved when using lean practices in the software engineering context. As the approaches aim for similar goals (i.e. the identification of waste) the approaches should be compared empirically to understand which of the approaches is the most beneficial.

REFERENCES

1. Larman C. *Agile and iterative development: a manager's guide*. Addison-Wesley: Boston, 2004.
2. Beck K, Andres C. *Extreme Programming explained: embrace change*. 2. ed. edn., Addison-Wesley: Boston, 2005.
3. Cumbo D, Kline E, Bumgardner MS. Benchmarking performance measurement and lean manufacturing in the rough mill. *Forest Products Journal* 2006; **56**(6):25 – 30.
4. Womack JP, Jones DT. *Lean thinking: banish waste and create wealth in your corporation*. Free Press Business: London, 2003.
5. Poppendieck M, Poppendieck T. *Lean Software Development: An Agile Toolkit (The Agile Software Development Series)*. Addison-Wesley Professional, 2003.
6. Reinertsen DG. *Managing the design factory: a product developers toolkit*. Free: New York, 1997.
7. Anderson D. *Agile management for software engineering: applying the theory of constraints for business results*. Prentice Hall, 2003.
8. Yin RK. *Case study research: design and methods*. 3 ed. edn., Sage Publications, 2003.
9. Runeson P, Höst M. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering* 2009; **14**(2):131–164.
10. Mujtaba S, Feldt R, Petersen K. Waste and lead-time reduction in a software product customization process with value-stream maps. *Proceedings of the 10th Australian Conference on Software Engineering (ASWEC 2010)*, 2010.
11. Gross JM, McInnis KR. *Kanban made simple: demystifying and applying Toyota's legendary manufacturing process*. AMACOM: New York, 2003.
12. Höst M, Regnell B, och Dag JN, Nedstam J, Nyberg C. Exploring bottlenecks in market-driven requirements management processes with discrete event simulation. *Journal of Systems and Software* 2001; **59**(3):323–332.
13. Middleton P. Lean software development: Two case studies. *Software Quality Journal* 2001; **9**(4):241–252.
14. Middleton P, Flaxel A, Cookson A. Lean software management case study: Timberline inc. *Proceedings of the 6th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2005)*, 2005; 1–9.
15. Perera GIUS, Fernando M. Enhanced agile software development hybrid paradigm with lean practice. *Proceedings of the International Conference on Industrial and Information Systems (ICIIS 2007)*, 2007; 239–244.
16. Parnell-Klabo E. Introducing lean principles with agile practices at a fortune 500 company. *Proceedings of the AGILE Conference (AGILE 2006)*, 2006; 232–242.
17. Maskell B, Baggaley B. *Practical lean accounting: a proven system for measuring and managing the lean enterprise*. Productivity Press, 2004.
18. Basili VR. Quantitative evaluation of software methodology. *Technical Report*, University of Maryland TR-1519 1985.
19. Petersen K, Wohlin C. A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. *Journal of Systems and Software* 2009; **82**(9):1479–1490, doi:doi:10.1016/j.jss.2009.03.036.
20. Montgomery DC, Runger GC. *Applied Statistics and Probability for Engineers*. Wiley, 2006.
21. Hevner AR, Salvatore MT, Park J, Sudha R. Design science in information systems research. *MIS Quarterly* 2004; **28**(1):75–103.
22. Somekh B. *Action research: a methodology for change and development*. Open University Press: Maidenhead, 2006.
23. Eisenhardt KM. Building theories from case study research. *Academy of Management Review* 1989; **14**(4):532–550.
24. White B. *Software configuration management strategies and Rational ClearCase: a practical introduction*. Addison-Wesley: Harlow, 2000.
25. Willcock C, Deiss T, Tobies S, Keil S, Engler F, Schulz S. *An Introduction to TTCN-3*. John Wiley & Sons Ltd., 2005.
26. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslen A. *Experimentation in Software Engineering: An Introduction (International Series in Software Engineering)*. Springer, 2000.
27. Petersen K, Wohlin C. Context in industrial software engineering research. *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement*, 2010; 401–404.
28. Robson C. *Real world research: a resource for social scientists and practitioner-researchers*. 2. ed. edn., Blackwell: Oxford, 2002.

29. Damm LO. Early and cost-effective software fault detection. PhD Thesis, Blekinge Institute of Technology Doctoral Dissertation Series No. 2007:09 2006.
30. Denne M, Clelund-Huang J. .
31. Damm LO, Lundberg L, Wohlin C. Faults-slip-through - a concept for measuring the efficiency of the test process. *Software Process: Improvement and Practice* 2006; **11**(1):47–59.
32. Petersen K, Wohlin C, Baca D. The waterfall model in large-scale development - state of the art vs. industrial case study. *Proceedings of the 10th International Conference on Product Focused Software Development and Process Improvement*, 2009; in submission.
33. Gorschek T, Wohlin C. Requirements abstraction model. *Requir. Eng.* 2006; **11**(1):79–101.
34. Gorschek T, Garre P, Larsson S, Wohlin C. A model for technology transfer in practice. *IEEE Software* 2006; **23**(6):88–95.
35. Pascale RT. *Managing on the edge: how the smartest companies use conflict to stay ahead*. Simon and Schuster: New York, 1990.
36. Donzelli P, Iazeolla G. A software process simulator for software product and process improvement. *Proceedings of the International Conference on Product Focused Software Process Improvement (PROFES 1999)*, 1999; 525–538.
37. Raffo D. Evaluating the impact of process improvements quantitatively using process modeling. *Proceedings of the 1993 Conference of the Centre for Advanced Studies on Collaborative research (CASCON 1993)*, IBM Press, 1993; 290–313.
38. Kemayel L, Mili A, Ouederni I. Controllable factors for programmer productivity: A statistical study. *Journal of Systems and Software* 1991; **16**(2):151–163.