Monitoring Evolution of Code Complexity in Agile/Lean Software Development

A Case Study at Two Companies

Vard Antinyan[1)], Miroslaw Staron[1)], Wilhelm Meding[2)], Per Österström[3)], Henric Bergenwall[3)], Johan Wranker[3)], Jörgen Hansson[4)] Anders Henriksson[4)]

Computer Science and Engineering [2)] Chalmers | [1)] University of Gothenburg
[3)] Ericsson AB, Sweden[4)] AB Volvo, Sweden
SE 412 96 Gothenburg

Abstract. One of the distinguishing characteristics of Agile and Lean software development is that software products "grow" with new functionality with relatively small increments. Continuous customer demands of new features and the companies' abilities to deliver on those demands are the two driving forces behind this kind of software evolution. Despite the numerous benefits there are a number of risks associated with this kind of growth. One of the main risks is the fact that the complexity of the software product grows slowly, but over time reaches scales which makes the product hard to maintain or evolve. The goal of this paper is to present a measurement system for monitoring the growth of complexity and drawing attention when it becomes problematic. The measurement system was developed during a case study at Ericsson and Volvo Group Truck Technology. During the case study we explored the evolution of size, complexity, revisions and number of designers of two large software products from the telecom and automotive domains. The results show that two measures needed to be monitored to keep the complexity development under control - McCabe's complexity and number of revisions.

Keywords: complexity; metrics; risk; Lean and Agile software development; code; potentially problematic; correlation; measurement systems;

1 Introduction

Actively managing software complexity has become an important aspect of continuous software development in large software products. It is generally believed that software products developed in a continuous manner are getting more and more complex over time, and evidence shows that the rising complexity drives to decreasing quality of software [1-3]. The continuous increase of code base and incremental increase of complexity can lead to large, virtually unmaintainable source code if left unmanaged.

A number of methods have been suggested to measure various aspects of software complexity, e.g. [4-10], accompanied with a number of studies indicating how adequately the proposed methods can relate to software quality. One of the well-known complexity measures, McCabe's cyclomatic complexity has been shown to be a good quality indicator although it does not reveal all aspects of complexity [11-14].

Despite the considerable amount of research conducted about the influence of complexity on software quality, little results can be found on how complexity influences on a continuously developed software product and how to effectively monitor small yet continuous increments of complexity in growing products. Therefore a ques-

tion remains how the previously established methods can be as efficiently used for software quality evaluation:

How to monitor complexity changes effectively when delivering feature increments to the main code branch in the product codebase?

The aim of this research is to develop methods and tool support for actively monitoring increments of complexity and drawing the attention of product managers, project leaders, quality responsible and the teams to the potentially problematic trends of growing complexity. In this paper we focus on the level of self-organized software development teams who often deliver code to the main branch for further testing, integration with hardware and ultimate deployment to end customers.

We address this question by conducting a case study at two companies which develop software according to Agile and Lean principles. The studied companies are Ericsson AB in Sweden which develops telecom products and Volvo Group Truck Technology which develops trucks under four brands – Volvo, Renault, Mack and UD Trucks.

Our results show that using a number of complementary measures of complexity and development velocity – McCabe's complexity and number of revisions per week – support teams in decision making, when delivering potentially problematic code to the main branch. By saying potentially problematic we mean that there is a tangible chance that the delivered code is fault prone or difficult to understand and maintain. Monitoring trends in these variables effectively draws attention of the self-organized Agile teams to a handful of functions and files which are potentially problematic. The handful of functions are manually assessed, and before the delivery the team formulates the decision whether they indeed might cause problems. The initial evaluation in two ongoing software development projects shows that using the two measures indeed draws attention to the most problematic functions.

2 Related Work

2.1 Continuous Software Evolution

A set of measures useful in the context of continuous deployment can be found in the work of Fritz [15] in the context of market driven software development organization. The metrics presented by Fritz measure such aspects as continuous integration pace or the pace of delivery of features to the customers. These metrics complement the two indicators presented in this paper with a different perspective important for product management.

The delivery strategy, which is an extension of the concept of continuous deployment, has been found as one of the three key aspects important for Agile software development organizations in a survey of 109 companies by Chow and Cao [16]. The indicator presented in this paper is a means of supporting organizations in their transition towards achieving efficient delivery processes.

Ericsson's realization of the Lean principles combined with Agile development was not the only one recognized in literature. Perera and Fernando [17] presented another approach. In their work they show 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.

2.2 Related Complexity Studies

Gill and Kemerer [8] propose another kind of cyclomatic complexity metric – cyclomatic complexity density and they show its usefulness as a software quality indicator. Zhang and Zhang [18] developed a method based on lines of code measure, cyclomatic complexity number and Halstead's volume to predict the defects of a software component. Two other studies provided evidence that files having large number of revisions are defect prone and hard to maintain [19], [20].

2.3 Measurement Systems

The concept of an early warning measurement system is not new in engineering. Measurement instruments are one of the cornerstones of engineering. In this paper we only consider computerized measurement systems – i.e. software products used as measurement systems. The reasons for this are: the flexibility of measurement systems, the fact that we work in the software field, and similarity of the problems – e.g. concept of measurement errors, automation, etc. An example of a similar measurement system is presented by Wisell [21] where the concept of using multiple measurement instruments to define a measurement system is also used. Although differing in domains of applications these measurement systems show that concepts which we adopt from the international standards (like [22]) are successfully used in other engineering disciplines. We use the existing methods from the ISO standard to develop the measurement systems for monitoring complexity evolution.

Lowler and Kitchenham [23] present a generic way of modeling measures and building more advanced measures from less complex ones. Their work is linked to the TychoMetric [24] tool. The tool is a very powerful measurement system framework, which has many advanced features not present in our framework (e.g. advanced ways of combining metrics). A similar approach to the TychoMetric's way of using metrics was presented by Garcia et al. [25]. Despite their complexity, both the TychoMetric tool and Garcia's approach can be seen as alternatives in the context of advanced data presentation or advanced statistical analysis over time.

Meyer [26, pp. 99-122] claims that the need for customized measurement systems for teams is one of the most important aspects in the adoption of metrics at the lowest levels in the organization. Meyer's claims were also supported by the requirements that the customization of measurement systems and development of new ones should be simple and efficient in order to avoid unnecessary costs in development projects. In our research we simplify the ways of developing Key Performance Indicators exemplified by a 12-step model of Parmenter [27] in the domain of software development projects.

3 Design of the Case Study

This case study was conducted using action research approach [28-30] where the researchers were part of the company's operations and worked directly with product development units of the companies. The role of Ericsson in the study was the development of the method and its initial evaluation, whereas the role of Volvo Group Truck Technology was to evaluate the method in a new context.

3.1 Ericsson

The organization and the project within Ericsson, which we worked closely with, developed large products for the mobile telephony network. The number of the developers in the projects was up to a few hundreds[1]. Projects were executed according to the principles of Agile software development and Lean production system, referred to as Streamline development (SD) within Ericsson [31]. In this environment, different development teams were responsible for larger parts of the development process compared to traditional processes: design teams (cross-functional teams responsible for complete analysis, design, implementation, and testing of particular features of the product), network verification and integration testing, etc.

The needs of the organization had evolved from metric calculations and presentations (ca. 7 years before the writing of this paper) to using predictions, simulations, early warning systems and handling of vast quantities of data to steer organizations at different levels and providing information from teams to management.

3.2 Volvo Group Truck Technology (GTT)

The organization which we worked with at Volvo Group developed Electronic Control Unit (ECU) software for trucks for such brands like Volvo, Renault, UD Trucks and Mack. The collaborating unit developed software for two ECUs and consisted of over 40 designers, business analysts and testers at different levels. The process was iterative, agile, involving cross functional teams.

The company used measures to control the progress of its projects, to monitor quality of the products and to collect data semi-automatically, i.e. automatically gathering of data from tools with the manual analysis of the data. The metrics collected at the studied unit fall into the categories of contract management, quality monitoring and control, predictions and project planning. The intention of the unit was to build a measurement system to provide stakeholders (like project leaders, product and line managers or the team) with the information about the current and predicted status of their products.

3.3 Process

According to the principles of action research we adjusted the process of our research with the operations of the company. We worked closely with project teams with dedicated designers, architects and managers being part of the research team. We conducted the study according to the following pre-defined process:

- Obtaining access to the source code of the products and their different releases
- Calculate complexity of all functions in the code
- Identify functions which changed complexity through 4 main releases
- Identify functions which changed complexity in 5 service releases between the two main releases
- Identify drivers for complexity changes in a subset of these functions
- Add new measures to the study:
  - Complexity per file
  - # revisions – to explore files which were changed often
  - # designers – to explore files which were changed by many designers in parallel

[1] The exact size of the unit cannot be provided due to confidentiality reasons.

  - # Number of lines of code (size) – to explore large files and functions
- Correlate measures to explore their dependencies
- Develop a measurement system (according to ISO 15939) to monitor the potentially problematic files.
- Monitor and evaluate the product during two releases

The above process was used during the development of the method at Ericsson and replicated at Volvo Group Truck Technology.

3.4 Units of Analysis

During our study we analyzed two different products – software for a telecom product at Ericsson and software for one electronic control unit from Volvo GTT from the automotive domain.

Ericsson: The product was a large telecommunication product composed by over one million lines of code with several tens of thousands C/C++ functions. Most of the source code was developed using C. The product had a few releases per year with a number of service releases in-between them. All versions of the source code of the product including the main and service releases were stored in version control system, IBM/Rational ClearCase. The product was a mature telecommunication product with a stable customer base. The product has been in development for a number of years.

The measures specified in the previous section were collected from different baseline revisions of the source code in ClearCase. In order to increase the internal validity of data collection and the quality of data we communicated closely with a reference group during bi-weekly meetings over a period of 8 months. The reference group consisted of 2 senior designers, one operational architect, one research engineer from the company, one manager and one metric team leader. The discussions considered the suitability of measures, measurement methods and functions (according to ISO/IEC 15939), validity of results and effectiveness of our measurement system.

Volvo GTT: The product was an embedded software system serving as one of the main computer nodes for a product line of trucks. It consisted of a few hundred thousand lines of code and several thousand C functions. The version control system is ClearCase. The software product had tight releases every 6-8 weeks. The analyses that were conducted were replications of the case study at Ericsson under the same conditions and using the same tools. The results were communicated with designers of the software product after the data was analyzed.

At both companies we developed measurement systems for monitoring the files and functions that can be risk driving when merging new code into the main branch. We defined the risk of merging a newly developed or a maintained function to main code base as a chance that the merged code would introduce new faults or would be noticeably more difficult to understand and maintain.

3.5 Measures in the Study

Table 1 presents the measures which we used in our study and their definitions:

Table 1. Metrics and their definitions

Name of measure	Abbreviation	Definition
Number of non-commented lines of code	NCLOC	The lines of non-blank, non-comment source code in a function

McCabe's cyclomatic complexity of a function	M	The number of linearly independent paths in the control flow graph of a function, measured by calculating the number of 'if', 'while', 'for', 'switch', 'break', '&&', '||' tokens
McCabe's cyclomatic complexity of a file	File M	The sum of all functions' M in a file
McCabe's cyclomatic complexity delta of a function	ΔM	The increase or decrease of M of a function during a specified time interval. We register the file name, class name (if available) and function name in order to identify the same function and calculate its complexity change in different releases.
McCabe's cyclomatic complexity delta of a file	File ΔM	The increase or decrease of File M during a specified time interval
Number of revisions of a file	NR	The number of check-ins of files in a specified ClearCase branch and its all sub-branches in a specified time interval
Number of designers of a file	ND	The number of developers that do check-in of a file on a specified ClearCase branch and all of its sub-branches during a specified time interval
Complexity of the most complex function in a file	Max M f	The complexity number M of the most complex function in a file

3.6 Focus Group

During this study we had the opportunity to work with a reference group at Ericsson and a designer at Volvo GTT. The aim of the reference group was to support the research team with expertise in the product domain and to validate the intermediate findings as prescribed by the principles of Action research. The group interacted with researchers on a bi-weekly meeting basis for over 8 months. At Ericsson the reference group consisted of:

- One product manager with over 10 years of experience and over 5 years of experience with Agile/Lean software development
- One measurement program/team leader with over 10 years of experience with software development and over 5 years of experience with Agile/Lean at Ericsson
- Two designers with over 6 years of experience in telecom product development.
- One operational architect with over 6 years of experience
- One research engineer with over 20 years of experience in telecom product development

At Volvo GTT we worked with one designer who had the knowledge about the product and over 10 years of experience with software development at the company.

4 Results and analysis

4.1 Evolution of the Studied Measures Over Time

We measured M for 4 main and 5 service releases at Ericsson and for 4 releases for the product at Volvo GTT. The results showed there are many new complex functions introduced as part of service releases. We observed that a large number of functions change the argument list during development. Many functions had long list of arguments which meant that the designers need to add or remove an argument or change the argument name to resolve a specific task. Thus the majority of the functions that

has been included as "new" in the statistics were actually old functions, which have changed argument's list. The designers agreed that these functions may introduce risks but with considerably less exposure than if these functions were indeed newly developed. Hence we disregarded the argument's list of functions in our measurement. Figure 1 shows the complexity evolution of functions in 5 service releases of the telecom product. Each line on the figure represents a C/C++ function.

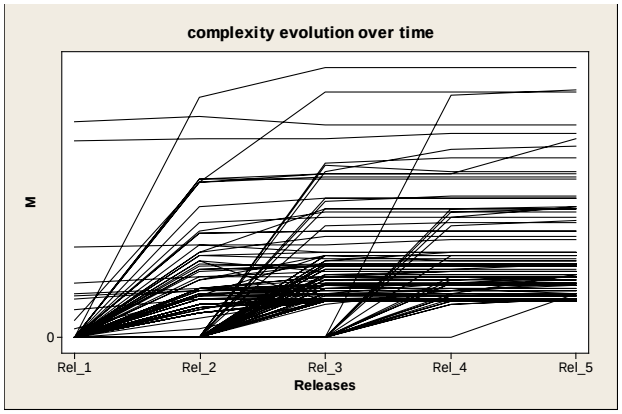

Figure 1. Evolution of complexity for functions with large complexity delta for one release and subsequent service releases in Telecom product

Measuring the evolution of McCabe's complexity M through releases in this manner resulted in:

- Observation that it is the newly developed functions which drive complexity increase between two major releases, as shows in Table 2.
- Observation that the majority of functions that are created complex keep the complexity at the same level over many releases – e.g. see Figure 1.

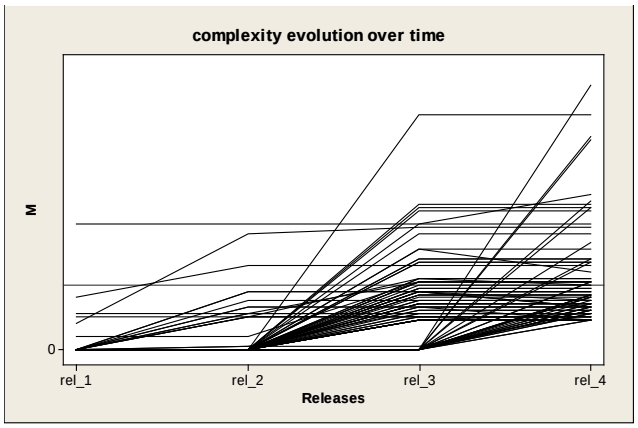

Figure 2. Evolution of complexity for functions with large complexity delta for four releases in product ECU of trucks

Figure 2 shows the complexity development of ECU of trucks for 4 releases.

The trends presented in Figure 2 are similar to the trends in Figure 1 and the number of functions in the diagram reflects the difference in size of the products.

Table 2 presents the results of complexity change between two service releases. The dashes in the table, under **old M** column indicate that the functions did not exist in the previous measurement point. The table shows that there are few functions that are new and already complex. In this particular measurement interval there are also 5 functions that were removed from the release. These functions are indicated by dashes under **new M** column (not shown in Figure 1). The results were consistent for all service releases for the telecom product, irrespective if there was a new functionality development or correction caused by customer's feedback. As opposed to the telecom product the number of newly introduced complex functions was dependent on whether a new end-to-end feature is implemented for truck. In Figure 2 we can see that for ECU software after the first release the number of functions with increased complexity is 5, whereas from second and third release there are many of them.

Table 2. Top functions of telecom product with highest complexity change between two service releases

file name	function name	old M	new M	Δ M
file 1	function 1	25	-	-25
file 2	function 2	83	-	-83
file 2	function 3	26	-	-26
file 3	function 4	57	90	33
file 4	function 5	27	-	-27
file 5	function 6	22	-	-22
file 5	function 7	-	25	25
file 6	function 8	-	30	30
file 6	function 9	-	51	51
file 7	function 10	-	23	23
file 8	function 11	-	26	26
file 9	function 12	-	26	26
file 10	function 13	-	22	22
file 11	function 14	-	27	27

In both products new complex functions appeared over time regardless the development time period. We investigated the reasons for high complexity of newly introduced functions in each release (both service and main) and unchanged complexity of existing functions. We observed that both companies assure that the most complex functions are maintained by the most skilled engineers to reduce the risks of faultiness. One of these functions was function 4 in Table 2, which between two releases increased the complexity significantly from an already high level. We observed the change of complexity for both long time intervals (between main releases) and for short time intervals (one week). Table 3 shows how the complexity of functions changes over weeks. The initial complexity of functions is provided under column **M** in the table (the real numbers are not provided for confidentiality reasons).We can see the week numbers on the top of the columns, and every column shows the complexity growth of functions in that particular week. Under **ΔM** column we can see the overall delta complexity per function that is the sum of weekly deltas per function.

The fact that the complexity of these functions fluctuates irregularly was interesting for the designers, as the fluctuations indicate active modifications of functions, which might be due to new feature development or represent defect removals with multiple test-modify-test cycles. Functions 4 and 6 are such instances illustrated in Table 3.

Table 3. Visualizing complexity evolution of functions over weeks

file name	function name	M	Δ M	w1306	w1307	w1308	w1309	w1310	w1311	w1312
file 1	function 1	M1	0	0	0	0	0	0	0	0
file 2	function 2	M2	15	0	0	0	0	0	15	0
file 2	function 3	M3	0	0	0	0	0	0	0	0
file 3	function 4	M4	5	4	-9	11	-11	10	0	0
file 4	function 5	M5	3	0	0	0	0	3	0	0
file 5	function 6	M6	13	17	0	11	-11	0	0	-4
file 5	function 7	M7	22	0	0	0	0	0	0	22
file 6	function 8	M8	20	0	0	0	18	2	0	0
file 6	function 9	M9	17	0	0	0	17	0	0	0
file 7	function 10	M10	11	0	0	0	11	0	0	0
file 8	function 11	M11	13	0	0	0	0	13	0	0
file 9	function 12	M12	28	0	28	0	0	0	0	0
file 10	function 13	M13	12	0	0	0	12	0	0	0

4.2 Correlation Analyses

When adding new measures to our analyses we needed to evaluate how the measures relate to each other by performing correlation analyses. However, in order to correlate the measures we need to define all the measures for the same entity (e.g. for a file or for a function, see Table 1). The correlation analysis for the telecom product is presented in Table 4.

Table 4. Correlation of measures for telecom product

	File M	File Δ M	Max ΔM	NR	ND
NCLOC	**0.9**	0.27	0.33	0.56	0.47
File M		0.28	0.32	0.48	0.41
File Δ M			**0.77**	0.24	0.25
Max Δ M f				0.35	0.37
NR					**0.92**

The correlations which are over 0.7 are in boldface, since it means that the correlated variables characterize the same aspect of the code. Table 5 presents the Pearson correlation coefficients between measures for the ECU for a truck. The correlations are visualized using correlograms in Figure 3 and Figure 4.

Table 5. Correlation of measures for ECU of truck

	File M	File Δ M	Max ΔM	NR	ND
NCLOC	**0.9**	0.43	0.48	0.61	0.38
File M		0.48	0.5	0.68	0.4
File Δ M			**0.84**	0.13	0.19
Max ΔM f				0.3	0.23
NR					0.46

The tables show that the M change is weakly correlated with NRs for both products. This was expected by the designers as the files with the most complex functions are usually maintained by certain designers and do not need many changes. The files with smaller complexity are not risky since they are easy to be modified. The designers noted that the really risky files are those which contain multiple complex functions that change often.

The strong correlation visible in the tables and diagrams above of NCLOC and M has been manifested by a number of other researchers previously [32], [33], [8].

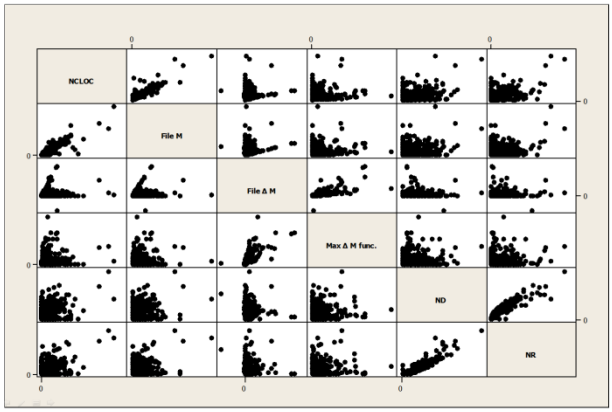

Figure 3. Correlogram of measures for telecom software

The original complexity definition is for a function as a measurement unit, thus we did correlation analyses on function's level. The results were:

- Correl. (M; NCLOC) = 0.76 telecom product
- Correl. (M; NCLOC) = 0.77 truck's software product

The correlation coefficient was weaker compared to correlation between the complexity of a file, which was caused by the fact that we measure the complexity of each file as a sum of complexities of all of its functions. This means that larger files with functions of small complexity will result in higher correlation. Designers claimed that there are many files having moderately complex functions that are solving independent tasks, which did not mean that the file is risky. This resulted in that we used the measure of complexity delta of functions rather than files in our measurement system as a complementary base measure.

Another important observation was the strong correlation between the number of designers and the number of revisions for telecom product Figure 3. Although at the beginning of this study the designers in the reference group believed that a developer of a file might check-in and check-out the file several times which probably is not a problem.

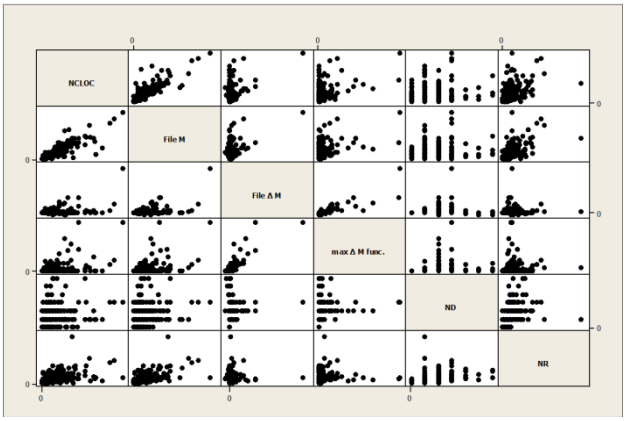

Figure 4. Correlogram of measures for ECU software

They assumed that large number of revisions itself is not as large problem as when many different designers change the file in parallel. This parallel development most likely increase the risk of being uninformed of one another's activities between different developers. The high correlation between **File ΔM** and **max ΔM** shows that the complexity change of the file is mainly due to complexity change of the most complex function in that file. A later observation showed that most of the files contain only one or two complex functions along with many other simple ones.

4.3 Design of the Measurement System

Based on the results that we obtained from investigation of complexity evolution and correlation analyses, we designed two indicators based on M and NR measures. These indicators capture the evolution of complexity and highlight potentially problematic files over time. These indicators were designed according to ISO/IEC 15959. An example definition of one indicator is presented in Table 6.

Table 6. ISO/IEC 15939 definition of the complexity growth indicator

Information Need	Monitor cyclomatic complexity evolution over development time
Measurable Concept	Complexity development of delivered source code
Relevant Entities	Source code
Attributes	McCabe's cyclomatic complexity of C/C++ functions
Base Measures	Cyclomatic complexity number of C/C++ functions – M
Measurement Method	Count cyclomatic number per C/C++ function according to the algorithm in CCCC tool
Type of measurement method	Objective
Scale	Positive integers
Unit of measurement	Execution paths over the C/C++ function
Derived Measure	The difference of cyclomatic number of a C/C++ function in one week development time period
Measurement Function	Subtract old cyclomatic number of a function from new one: ΔM = M(week) – M(week-1)
Indicator	Complexity growth: *The number of functions that exceeded McCabe complexity of 20 during the last week*
Model	Calculate the number of functions that exceeded cyclomatic number 20 during last week development period
Decision Criteria	If the number of functions that have exceeded cyclomatic number 20 is different than 0 then it indicates that there are functions that have exceeded established complexity threshold. This suggests the need of reviewing those functions, finding out the reasons of complexity increase and refactoring if necessary

The other indicator is defined in the same way: the number of files that had NR > 20 during last week development time period.

The measurement system was provided as a gadget with the necessary information updated on a weekly basis (Figure 5). The measurement system relies on two previous studies carried out at Ericsson [34, 35].

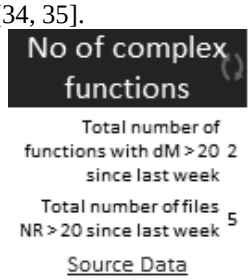

Figure 5. Information product for monitoring ΔM and NR metrics over time

For instance the total number of files with more than 20 revisions since last week is 5 (Figure 5). The gadget provides the link to the source file where the designers can find the list of files or functions and the color-coded tables with details.

We visualized the NR and ΔM measures using tables as depicted in Table 3. Presenting the ΔM and NR measures in this manner enabled the designers to monitor those few most relevant files and functions at a time out of several thousands. As in Streamline development the development team merged builds to the main code branch in every week it was important for the team to be notified about functions with drastically increased complexity (over 20). This table drew the attention of designers to the potentially problematic functions on a weekly basis – e.g. together with a team meeting.

5 Threats to Validity

In this paper we evaluate the validity of our results based on the framework described by Wohlin et al. [36]. The framework is recommended for empirical studies in software engineering.

The main external validity threat is the fact that our results come for an action research. However, since two companies from different domains (telecom and automotive) were involved, we believe that the results can be generalized to more contexts than just one company.

The main internal validity threat is related to the construct of the study and the products. In order to minimize the risk of making mistakes in data collection we communicated with reference groups at both companies to validate the results.

The limit 20 for cyclomatic number established as a threshold in this study does not have any firm empirical or theoretical support. It is rather an agreement of skilled developers of large software systems. We suggest that this threshold can vary dependent on other parameters of functions (block depth, cohesion, etc.). The number 20 is a preliminary established number taking into account the number of functions that can be handled on weekly basis by developers.

The main construct validity threats are related to how we match the names of functions for comparison over time. The measurement has been in the following way: We measured the M complexity number of all functions for two consequent releases, registering in a table function name and file name that the function belongs to. We register the class name of the functions also if it is a C++ function. Then we compare

the function's, file's and class' names of registered functions for two releases. If there is a function that has the same registered names in both releases we consider that they are the same functions and calculate the complexity number variance for them.

Finally the main threat to conclusion validity is the fact that we do not use inferential statistics to monitor relation between the code characteristics and project properties, e.g. number of defects. This was attempted during the study but the data in defect reports could not be mapped to individual files, this jeopardizing the reliability of such an analysis. Therefore we chose to rely on the most skilled designers' perception of how fault-prone and unmaintainable code is delivered.

6 Conclusions

In Agile and Lean software development quick feedbacks on developed code and its complexity is crucial. With small software increments there is a risk that the complexity of units of code or their size can grow to unmanageable extensions through small increments.

In this paper we explored how complexity changes by studying two software products – one telecom product at Ericsson and one software for electronic control unit at Volvo GTT. We identified that in short periods of time a few out of tens of thousands functions have significant complexity increase. In large products software development teams need automated tools to identify these potentially problematic functions. We also identified that the self-organized teams should be able to make the final assessment whether the "potentially" problematic is indeed problematic.

By analyzing correlations we found that it is enough to use two measures – McCabe complexity and number of revisions – to draw attention of the teams and to designate files as "potentially" problematic.

The automated support for the teams was provided in form of a MS Sidebar gadget with the indicators and links to statistics and trends with detailed complexity development. The method was validated on a set of historical releases.

In our further work we intend to extend our validation to products under development and evaluate which decisions are triggered by the measurement systems. We also intend to study how the teams formulate the decisions and monitor their implementation.

Acknowledgment

The authors thank the companies for their support in the study. This research has been carried out in the Software Centre, Chalmers, University of Gothenburg and Ericsson AB, Volvo Group Truck Technology.

References

[1] B. Boehm, "A view of 20th and 21st century software engineering," in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 12-29.

[2] T. Little, "Context-adaptive agility: managing complexity and uncertainty," *Software, IEEE*, vol. 22, pp. 28-35, 2005.

[3] J. Bosch and P. Bosch-Sijtsema, "From integration to composition: On the impact of software product lines, global development and ecosystems," *Journal of Systems and Software*, vol. 83, pp. 67-76, 1// 2010.

[4] S. Henry and D. Kafura, "Software structure metrics based on information flow," *Software Engineering, IEEE Transactions on*, pp. 510-518, 1981.
[5] T. J. McCabe, "A complexity measure," *Software Engineering, IEEE Transactions on*, pp. 308-320, 1976.
[6] B. Curtis, "Measuring the psychological complexity of software maintenance tasks with the Halstead and McCabe metrics," *IEEE Transactions on Software Engineering*, vol. SE-5, p. 96.
[7] M. H. Halstead, *Elements of software science* vol. 19: Elsevier New York, 1977.
[8] G. K. Gill and C. F. Kemerer, "Cyclomatic complexity density and software maintenance productivity," *Software Engineering, IEEE Transactions on*, vol. 17, pp. 1284-1288, 1991.
[9] R. P. L. Buse and W. R. Weimer, "A metric for software readability," in *Proceedings of the 2008 international symposium on Software testing and analysis*, 2008, pp. 121-130.
[10] Y. Wang, "On the Cognitive Complexity of Software and its Quantification and Formal Measurement," *International Journal of Software Science and Computational Intelligence (IJSSCI)*, vol. 1, pp. 31-53, 2009.
[11] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 452-461.
[12] T. M. Khoshgoftaar, E. B. Allen, K. S. Kalaichelvan, and N. Goel, "Early quality prediction: A case study in telecommunications," *Software, IEEE*, vol. 13, pp. 65-71, 1996.
[13] B. Ramamurthy and A. Melton, "A synthesis of software science measures and the cyclomatic number," *Software Engineering, IEEE Transactions on*, vol. 14, pp. 1116-1121, 1988.
[14] M. Shepperd and D. C. Ince, "A critique of three metrics," *Journal of Systems and Software*, vol. 26, pp. 197-210, 9// 1994.
[15] T. Fitz. (2009). *Continuous Deployment at IMVU: Doing the impossible fifty times a day*. Available: http://timothyfitz.wordpress.com/2009/02/10/continuous-deployment-at-imvu-doing-the-impossible-fifty-times-a-day/
[16] T. Chow and D.-B. Cao, "A survey study of critical success factors in agile software projects," *Journal of Systems and Software*, vol. 81, pp. 961-971, 2008.
[17] G. I. U. S. Perera and M. S. D. Fernando, "Enhanced agile software development - hybrid paradigm with LEAN practice," in *International Conference on Industrial and Information Systems (ICIIS)*, 2007, pp. 239-244.
[18] H. Zhang, X. Zhang, and M. Gu, "Predicting defective software components from code complexity measures," in *Dependable Computing, 2007. PRDC 2007. 13th Pacific Rim International Symposium on*, 2007, pp. 93-96.
[19] A. Monden, D. Nakae, T. Kamiya, S. Sato, and K. Matsumoto, "Software quality analysis by code clones in industrial legacy software," in *Software Metrics, 2002. Proceedings. Eighth IEEE Symposium on*, 2002, pp. 87-94.
[20] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*, 2008, pp. 181-190.
[21] D. Wisell, P. Stenvard, A. Hansebacke, and N. Keskitalo, "Considerations when Designing and Using Virtual Instruments as Building Blocks in Flexible Measurement System

Solutions," in *IEEE Instrumentation and Measurement Technology Conference*, 2007, pp. 1-5.

[22] International Bureau of Weights and Measures., *International vocabulary of basic and general terms in metrology = Vocabulaire international des termes fondamentaux et généraux de métrologie*, 2nd ed. Genève, Switzerland: International Organization for Standardization, 1993.

[23] J. Lawler and B. Kitchenham, "Measurement modeling technology," *IEEE Software*, vol. 20, pp. 68-75, 2003.

[24] Predicate Logic. (2007, 2008-06-30). *TychoMetrics*. Available: http://www.predicatelogic.com

[25] F. Garcia, M. Serrano, J. Cruz-Lemus, F. Ruiz, M. Pattini, and ALARACOS Research Group, "Managing Software Process Measurement: A Meta-model Based Approach," *Information Sciences*, vol. 177, pp. 2570-2586, 2007.

[26] Harvard Business School, *Harvard business review on measuring corporate performance*. Boston, MA: Harvard Business School Press, 1998.

[27] D. Parmenter, *Key performance indicators : developing, implementing, and using winning KPIs*. Hoboken, N.J.: John Wiley & Sons, 2007.

[28] A. Sandberg, L. Pareto, and T. Arts, "Agile Collaborative Research: Action Principles for Industry-Academia Collaboration," *IEEE Software*, vol. 28, pp. 74-83, Jun-Aug 2011 2011.

[29] R. L. Baskerville and A. T. Wood-Harper, "A Critical Perspective on Action Research as a Method for Information Systems Research," *Journal of Information Technology*, vol. 1996, pp. 235-246, 1996.

[30] G. I. Susman and R. D. Evered, "An Assessment of the Scientific Merits of Action Research," *Administrative Science Quarterly*, vol. 1978, pp. 582-603, 1978.

[31] P. Tomaszewski, P. Berander, and L.-O. Damm, "From Traditional to Streamline Development - Opportunities and Challenges," *Software Process Improvement and Practice*, vol. 2007, pp. 1-20, 2007.

[32] G. Jay, J. E. Hale, R. K. Smith, D. Hale, N. A. Kraft, and C. Ward, "Cyclomatic complexity and lines of code: empirical evidence of a stable linear relationship," *Journal of Software Engineering and Applications (JSEA)*, 2009.

[33]M. Shepperd, "A critique of cyclomatic complexity as a software metric," *Software Engineering Journal*, vol. 3, pp. 30-36, 1988.

[34] M. Staron, W. Meding, G. Karlsson, and C. Nilsson, "Developing measurement systems: an industrial case study," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 23, pp. 89-107, 2011.

[35] M. Staron and W. Meding, "Ensuring reliability of information provided by measurement systems," in *Software Process and Product Measurement*, ed: Springer, 2009, pp. 1-16.

[36] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslèn, *Experimentation in Software Engineering: An Introduction*. Boston MA: Kluwer Academic Publisher, 2000.