Measuring the Software Engineering Process - Report

Module: Software Engineering

Lecturer: Stephen Barrett

Michaelmas Term 2017
Trinity College Dublin

Caroline Berg

December 6, 2017

1 Introduction

With rising demands in software industries numerous models on how to organize and measure the software engineering process have emerged in the last decades. There are different approaches to this matter while new studies suggesting enhancements emerge on a steady basis. So far none of them provided a satisfactory solution to the multilayered problems that come along with the development process. As to be seen with most areas of software engineering there is no silver bullet, given the high degree of complexity of this subject.
The following report starts with a closer look on how to determining quality in software development processes and introduces two models that aim at capturing the necessary stages for software development. Following sections continue with a short outline of existing metrics in this field. Section three introduces two tools that automatically conduct measurement. The last paragraph discusses the disadvantages in terms of ethical concerns that arise with the process of measuring engineering.

1.1 Defining Quality

Since quality is a ubiquitous term people have to deal with a set of problems when it comes to estimating quality. How we understand quality depends on contextual factors. In software engineering many environmental factors surrounding the development of the product need to be considered and optimized to ensure a high degree of quality of the final product. Therefore it is common practice to plan the process ahead and apply metrics throughout the stages of development. Software engineering is one of the most complex things created by human beings. Typically there are many people with different expectations and goals in mind involved in the process.
To estimate the effectiveness of software metrics and obtained measurements, it is crucial to first define the core components of quality management. Kan[1] mentions the "conformance to requirements" as key factor for the estimation of software quality. During the development process the conformance needs to be checked regularly to ensure that the project it heading into the right direction and no time or resources have to be spend on readjustment to the original requirements. A second major aspect that has to be taken into account is the "fitness for use" of the final product to the real world environment. Problems that arise from unforeseen use of the software by its users greatly diminish the quality of it.
To take into account the crucial aspects at the right stage of development the

[1] Kan p.1

first step is to collect, sum and analyze a clients vision of a system and agree on how his desires can be implemented by the means of software development. The thoroughness and level of detail that is applied to the conduct of the first stage of software development determines the final outcome to a great extent.

1.2 Modelling the Engineering Process

A Software Development Process Model, also known as Systems Development Life Cycle (SDLC) is used in software engineering to provide templates for the organization, implementation and deployment of a software project. Among many other approaches to visualizing and grouping different stages of the process, the *Waterfall Development Model* is one of the widely known concepts. Another well known depiction of the development process is presented in the *Spiral Model*. The next two paragraphs will describe the basic concepts of both models and their advantages and disadvantages.

1.2.1 The Waterfall Model

The Waterfall Model is based on a linear six-step sequential flow, which means that any previous stage of the six steps has to be completed before the process is continued. Each phase can be seen as an individual step without overlap to any other stage.

1. The first step consists of a detailed requirement analysis. After gathering the requirements that need to be met to produce the desired software, they have to be specified in a respective document, called the *Software Requirement Specification (SRS)*. It is therefore common to describe the processes in either a textual document or present a visualization, for example a data flow diagram. For textual descriptions a widely approved template is the *IEEE Software Requirements Specification Template*.
   Besides clearly defining functional and non-functional requirements for the software, the purpose of the *SRS* is to serve as an agreement between the development company and the client. It should be ensured that both sides agree on a realistic assessment of the implementation of the product to prevent future disagreements that in the worst case lead to the failure of the project.

2. The next step is to outline the overall system architecture, thus emphasizing requirements in hardware that result from software requirements.

3. After the definition of the framework the actual implementation of the

software is done in reasonably small units that need to be tested according to their specification. This process is known as unit testing.

4. Given that each individual unit passes extensive testing, the components are put together to build the entire system. Similar to the single unit testing the system's functionality and potential problems are tested extensively.

5. The previous stages culminate in the deployment of the system. This stage may differ depending on whether the software is a standalone product or has to be integrated into an existing frame provided by the customer.

6. For it is very unlikely that the software will run without complaints, maintenance is done to resolve issues. Sooner or later alterations and improvements will likely result in a new version of the original product.

Since each phase requires precise outlining and completion before moving on to the next part, the *Waterfall Model* can be successfully deployed for smaller, straightforward projects. It can be easily understood and its simplicity leaves little potential for confusion or trade-offs that might conflict with the initial requirements. This implies that especially the initial stages need to be well thought out and scheduled thoroughly. Subsequent alterations in requirements or unforeseen problems cause a cumbersome readjustment process that is not taken into account in the *Waterfall Model*. Once the *SRS* is set, it is final and hence immune to alterations, which can be a curse or a blessing depending on the type of the project. The approach of the *Waterfall Model* is therefore less suitable for long term projects that are partly dependent on unpredictable components. According to the timeline provided by the model the integration of the final product is done at the very end of the project, leaving little possibilities for identification of technical obstacles in the first place.

1.2.2 The Spiral Model

The *Spiral Model* is to a great extend based on the concept of the *Waterfall Model* while taking into account potential risks that might arise during the development process. This is done by continuously releasing versions of the product ensuring additive refinement. During the entire process steps are organized in a spiral manner accounting for the name of the model.
Starting with the definition of the system's requirements, a provisional design concept is established. At an early stage the first prototype of the system is released based on the preliminary software design, forming a first version that may differ to a great extend from the final product.
To enhance the first prototype an inherent four-step procedure is applied. Firstly

the existing prototype is evaluated with respect to its advantages and disadvantages. The requirements for the second version of the prototype are formed from these insights and a new prototype is implemented respectively. Finally it is tested against prospective shortcomings.
The rather unspecified approach of this model bears the risk that costs and deadlines spin out of control. It is therefore up to the client to make the decision to quit the project at any stage of the development process. The client's view plays a leading role during the entire process and it should be ensured that he or she is constantly involved in the decision making part of the project.
The step of evaluation and refinement in the fourfold manner is repeated until the customers requirements are satisfied.
After the release of the final version maintenance is done on the software product to ensure long term functionality.
The *Spiral Model* approach ensures customer satisfaction and prevents risky measures by continuous readjustment and early detection of potential bottlenecks. While it improves the shortcomings of the *Waterfall Model* by enabling flexibility in time management and the concrete design of the product, it is very vague in its description of the spiral phases and therefore not suitable for inexperienced developers or short-term projects that need to be released before a certain deadline. The *Spiral Model* has proven to be a good choice for internal software projects given that the flexibility which this approach demands can be granted by the management.

2 Metrics

The basic requirement of a metric is that it precisely depicts the desired information about the ongoing development process. It should be thoroughly evaluated whether the conducted measurements actually reflect the information they intent to capture. Reliability and validity[2] are the most important factors when it comes to evaluating the benefits of a metric.
The degree of consistency of derived data from a certain metric, that is consistency in outcomes of the same metric applied to the same input, is an indicator for reliability. Reliability is defined as $\frac{standard \quad deviation}{mean}$, low results indicating a high reliability. To enable consistency the description of the procedure that produces measurements should be as precise as possible, leaving little possibilities for variations.
As opposed to reliability, validity is very difficult to compute when it is applied

[2]Kan p.4

to abstract concepts. It is usually grouped into different types, such as coverage of the concepts or construct validity, which describes how well a metric captures the information that we are trying to obtain.
It is not unusual to agree on a trade off between reliability and validity. In many cases a distinct definition of the metric leads to high reliability, while limiting the means to represent the concept it is aimed at, thus resulting in a low degree of validity.

Metrics in software engineering can be grouped into three mayor fields. Process Metrics help to analyze the quality and efficiency of individual procedures happening during the entire development process. Project Metrics are useful to keep track of financial expenses, schedule delays and interim and final deliverables. Product Metrics are meant to ensure the quality of the product.
This grouping is only reasonable to a certain extent and the three categories are closely interrelated. For example a delay in schedule is an indicator that something is not going according to plan and will probably effect the quality of the final product.
Metrics within the different categories are aimed at capturing effort, quality and progression of the software development process. In addition to that size denoted as *Thousand Lines per Code (KLOC)* is a common measure that is deployed in metrics. It is important to note that *KLOC* was introduced at a time where programming languages like FORTRAN or COBOL where commonly used, explaining the emphasis on the number of lines of code, while nowadays it must be individually evaluated whether many lines of code are favorable for the respective project.
A prominent metric for size oriented measurements is the *number of pages of documentation per KLOC*. Good documentation is a mayor requirement that not only influences the degree of replicability but also gives insights into the conscientiousness which is employed by the developer when working on an assignment. A rather simple and patent metric is *KLOC per person month*, while factors such as programming language and the degree of complexity of the respective assignment need to be taken into account.
When it comes to estimating effort *Function Points per month* are a popular choice. Function points measure the functionality that a code unit provides to the entire software product. They bear the advantage of being code independent and are regarded as a more sophisticated metric compared to *KLOC*.
Quality is often defined as the number of defects in a product. Exemplary metrics are *defect density*, which is defined as $\frac{total\ \ number\ \ of\ \ bugs}{total\ \ lines\ \ of\ \ code}$ and *defect removal efficiency*, which is *the number of defects resolved* divided by *the number of defects at any given point*.

Progression of the process is regarded as the conformity of the schedule and the actual state. Two approaches to capturing progression are described in the following section, which introduces the *Personal Software Process*.

3 The Personal Software Process

The *Personal Software Process (PSP)* is a template for the development process invented in 1995 by *Watts Humphrey*. In combination with the *Team Software Process (TSP)* it is meant to lead each team member through the software development process, by providing guidelines and setting standards, thus organizing group work. Scripts provide orientation on necessary steps that have proven to be worthwhile in software engineering processes and different metrics aim at giving objective feedback on how well different aspects of the process are proceeding.

Tools for estimating the size and time cost of a requirement before implementation starts are a key component of the *PSP.Proxy-based Estimating* is the most common concept applied when it comes to predicting future costs. The idea here is to base the estimation on former observations in similar processes. When specifying requirements in the early stages of the project, it is typical to predefine software components such as objects, functions or logical entities, depending on the programming. These smaller software units are called proxies and an estimation on how many lines will be necessary to implement them can be derived from existing projects, that implemented related features.
The *PSP* model puts high emphasis on measuring progression as a fundamental component of the software engineering process. This is employed via the concept of *Earned Value Management (EVM)*. *EVM* is intended to measure the variance from scheduled milestone to actual states. The central formulas applied in *EV* management are *Planned Value (PV)* and *Earned Value (EV)*.
PV monitors the supposed progress that should be achieved at any arbitrary point of the process. It can be subdivided into *Current PV*, which describes the costs approved for the current part of the project, and *Cumulative PV* which indicates the current total of *PV* at any given state. Hence the *PV* gives an indication of the "value" that should have been delivered at any state of the project.
On the other hand the *EV* keeps track of the actual value delivered to a certain point. Analogous to the *PV* it can be expressed in terms of *Current EV* and *Cumulative EV*.
Apart from other indicators the *Schedule Performance Index (SPI)* and *Schedule Variance (SV)* can easily be computed from the values that *PV* and *EV* provide.

$$SPI = \frac{EV}{PV}$$
$$SV = EV - PV$$

SPI is supposed to indicate, whether the actual state of the project is ahead or behind the scheduled achievements, while *SV quantifies* how far behind or ahead you are.

A concept that comes along with *PSP* is the *Team Software Process (TSP)*. It is formed out of team members that commit themselves to operating according to *PSP*. Team roles are assigned to the members and there is a high emphasis on regular meetings and discussions, where members give a report on their progress. Software engineering groups that operate according to *TSP* have reportedly been more successful at accomplishing milestones in projects and meeting requirements in a given time.
Coming to terms on how well the *PSP* manages to improve and facilitate the work of software engineers, many papers and academic articles report a growth in efficiency and higher success rates. It is said to provide a solid framework, which can easily be adapted and enhanced with additional metrics for specific projects if necessary. While it is often reported that developers would not use this methodology if it was not imposed on them, they note an improvement of their capabilities when it comes to estimating their work and successfully meeting long-term goals.
Although the concept and aspirations of the *PSP* are widely praised, there is some criticism coming mainly from software developers who have made experiences with the methodology in practice . Some describe it as a process that is all about data being "captured, recorded, followed and measured", inducing team leaders to focus on presenting good figures at the end of the month instead of ensuring high-quality outcomes. It is also said to put a lot of pressure on developers leading them to manipulate numbers that can then be "sold to the management" in order to fulfill the demands of *PSP*.

4 Measurment Tools

A disadvantage of the *PSP* is the high overhead resulting from the requirement of collecting data manually. Users complain about the amounts of time that is spent on filing documents and keeping track on the time they spend on specific tasks. There are a lot of *PSP*-compatible tools available, many of them open-source projects such as *Process Dashboard* or *Hackystat* that provide a solution

to this problem. Tools usually support automated data collection as well as data tracking and analysis. The following subsections introduce two of them.

4.1 Jasmine

With *Jasmine* the data needed for *PSP* analysis is collected automatically. To provide guidance through the different phases of the software development process *Jasmine* offers an *Electronic Process Guide (EPC)* combined with the *Experience Repository (ER)*. It is highly focused on the individual developer's struggles and efforts.
As it is common for most software process analysis tools data is collected by *Sensors* that can be installed within the editors in use, such as *Eclipse* or *JBuilder*. The data is then sent to a server to analyze and retrieve the desired measures. *Jasmine's* two main components are the *Personal Process Management Tool (PPMT)* and the *PSP Guide/Experience Repository (PSPG/ER)*. *Sensors* collect the data automatically and send it to the *PPMT Client engine*. The data is then forwarded to the *PPMT Server* where the analysis is done. The results are displayed in a web interface, which can also be supplied by developers with manual data or schedules that are in return forwarded to the *PPMT Server*. The results are also forwarded to the *PSPG/ER*-component which also comes with a web interface that lets the user submit personal experiences and provides the *PSP Guide* for orientation.
More specifically the *Sensor*-based data collection retrieves data regarding time spent on tasks, growth of software size, and defects as well as the efforts spent on solving them. Failed unit tests, compile and runtime-errors are stored as defects in a log file, simultaneously logging additional data such as defect type, found date and fix time. The analysis done by the *PPMT client* is visualized for the users in trend charts or tables and reports that summarize the progresses made over time.
The *PSPG/ER* interface provides an overview of what has been achieved so far and what is yet to be accomplished. Each section of the project can be selected and provides a description of the assignments grouped under it and related links concerning artifact pages or submitted experiences concerning it the assignment. An additional part of the *ER* is a chat functionality that enables software engineers to comment and discuss problematic features.
Jasmine provides big enhancements when it comes to the time consuming process of documenting and submitting data to *PSP* tools. Comparing it to other *Sensor*-based data collection tools such as *Hackystat* a clear advantage is that *Jasmine* provides functionality for the *PSP* aspects of planning and estimation. In addition *Jasmine* provides a more precise data analysis when it comes to

monitoring defects, as it automatically keeps track of the time spent on testing and fixing errors.

4.2 PROM

PROM is a component-based software tool that collects a broad spectrum of metrics, such as object oriented metrics and *PSP* related metrics. It is entirely written in JAVA and hence platform independent, using plug-ins to ensure compatibility with a wide range of IDEs. *PROM* is mainly composed of the *PROM Database*, the *PROM Server*, the *Plug-in Sever* and the *Plug-in component*. Similar to the architecture of *Jasmine* the database stores collected data while the *PROM Server* provides an interface to the database. It enables clients with web service extensions to directly analyze and examine the retrieved data. The *Plug-in Server* provides a cache thereby enabling offline working. The data is being collected and filtered by plug-ins and then forwarded to the *PROM Server* as soon as a connection is established. *PROM* puts a high emphasize on privacy. Data that is generated by an individual developer can not be directly accessed by the project manager. Its creators argue that personal data is of no use for the management division since the aggregated data of the whole development group provides enough insight into current states.

5 Conclusion

The measurements of software engineering processes enable people who are directly involved in the process of development and their coordinators to obtain a feedback on their work and make adjustments if necessary to ensure the quality of the final product.
The process of software project development is undeniably complex and often comes with many hardly accessible parameters of all kinds. Unforeseen problems that pose a potential risk to the results of the process need to be detected at an early stage to minimize damages.
However there are many concerns when it comes to weighting out the benefits of these metrics against the downsides that developers are facing. When reading through recommendations on how to interpret and use the data obtained via these metrics, three pieces of advice are mentioned continuously:

- Do not use the data to assess the individual developer.
- Do not use the observations to intimidate the developer.
- Do not consider a high error rate as a negative indicator, but see it as a way to ensure improvement.

Obviously a lot of ethical concerns arise with the introduction of software process monitoring. Arguably it is of no use to put developers under pressure but at the same time it is necessary to provide objective feedback to ensure improvement of the process.
It has been reported that these methods are often opposed on the development team by the management board and developers are usually less than delighted find themselves constantly tracked during their working hours. The fear of being stripped of one's right to privacy often goes along with the introduction of these tools.
Since each software developer has his or her own ways of approaching tasks, they should not be forced to follow a strict schedule. However it can not be denied that concepts like the *PSP* provide guidance and mitigate shortcomings in time management and documentation. Tools to conduct software engineering metrics and process models should either be optionally provided for a developer or, as in the case of *PROM*, only the developer him- or herself should gain insight into personal analysis and a report should be provided to the management facilities in form of a summation of the whole development department.

Sources[3]

- Kan, Stephen H.: Metrics and models in software quality engineering; 2.ed - Boston: Addison-Wesley, 2002.
- Noopur, Davis; Mullaney, Julia L.: The Team Software Process (TSP) in Practice: A Summary of Recent Results. Pittsburgh, PA: Software Engineering Institute, Sept. 2003.
- Khan, Abdul Kadir: Impact of Personal Software Process on Software Quality. IOSR Journal of Computer Engineering (IOSRJCE), May-June 2012.
- Shin, Hyunil; Choi, Ho-Jin; Baik, Jongmoon: Jasmine: A PSP Supporting Tool. ICSP'07 Proceedings of the 2007 international conference on Software process, May 2007.
- Sillitti, Alberto; Janes, Andrea; Succi, Giancarlo; Vernazza, Tullio: Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data. Conf. Proc. EUROMICRO, 2003.
- Živadinović, Jovan; Medić, Zorica; Maksimović, Dragan; Damnjanović, Aleksandar; Vujčić, Slañana: Methods of Effort Estimation in Software Engineering. I International Symposium Engineering Management And Competitiveness, June 2011.
- https://softwareengineering.stackexchange.com/questions/25728/anybody-use-the-team-software-process-tsp-and-or-personal-software-process-ps
- https://stackoverflow.com/questions/28197/do-you-follow-the-personal-software-process-does-your-organization-team-follow
- https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm
- https://en.wikipedia.org/wiki/Software_requirements_specification
- https://www.tutorialspoint.com/sdlc/sdlc_spiral_model.htm
- http://searchsoftwarequality.techtarget.com/definition/spiral-model
- http://www.zyxware.com/articles/4134/how-to-calculate-the-defect-density-in-software-products
- https://swtestingconcepts.wordpress.com/test-metrics/defect-removal-efficiency/
- https://en.wikipedia.org/wiki/Personal_software_process
- https://en.wikipedia.org/wiki/Team_software_process
- http://www.chambers.com.au/Sample_p/co_proxy.htm
- https://www.pmi.org/learning/library/earned-value-management-systems-analysis-8026

[3]retrieved 4th December 2017