Measuring Engineering

Thomas Erlis

Student Number: 15324746

The software engineering process can be measured and assessed in terms of measurable data in many different ways, and there exists many computational platforms and algorithmic approaches to both gathering and analysing this data, although there are many ethical concerns with this kind of analytics, depending on the framework implemented. In this report, I intend to cover a few of the more popular and interesting methods for measuring the process that I have found during my research and to cover some of the ethical concerns that go along with each.

The trend I noticed during my research was that there is a trade-off between easily obtained analytics, which are less controversial but with more limited usefulness and more generality, and richer more in-depth analytics which can raise privacy and overhead concerns.

I found there to be a split between two kinds of measures in engineering, product metrics and process metrics. Product metrics are the products qualities, the physical data describing the product, which in software includes the code length, complexity, reusability and maintainability. Process metrics describes the qualities of the process, such as effort required, production time, etc. In software, this includes processes such as editing time, number and type of changes in a class or in a file, etc.

Measures are difficult to obtain in software engineering due to two problems. Collecting metrics can be a time intensive task, most software projects are restrained and limited by their deadline already so in a lot of cases there is no time to spend on work that does not produce any immediate benefits to the project. Secondly, manual data collection is unreliable. There are errors and missing data which will affect the analysis.

Any analysis looking for correlation between the data and the product is hampered by low quality manual data. There are some automated tools that seek to both improve data quality, reduce manual effort and perform analysis. These tools were the focus of my research.

Personal Software Process (PSP) (1)

As I considered these automated tools I found that they were somewhat based off an old manual method known as the Personal Software Process (PSP). The technique was first outlined in the book ‘A Discipline for Software Engineering’ by Watts S. Humphrey. The book had three main innovations, it showed how to adapt organisational software process analytics for developers, how these analytics could help developers improve and presented the practices in such a way that they could be adopted by both the academic and professional world. This was the PSP. It can yield rich, high impact analytics, but it does have significant overhead costs for developers and ethical implications.

The books version of PSP uses spreadsheets and manual collection and analysis of data which is a substantial effort and time investment. This manual approach makes the analytics fragile and data may be prone to errors, but it is also extremely flexible as it allows its users to find the analytics best suited to their needs at the time. As outlined in *Searching Under the Streetlight for Useful Software Analytics* by Philip M. Johnson, the LEAP toolkit was designed to rectify this.

LEAP Toolkit (Lightweight, Empirical, Antimeasurement Dysfunction and Portable Software Process Measurement)

The toolkit aimed to address data quality problems which plagued the PSP by automating data analysis. The data is still manually entered, but the toolkit automatically performs PSP analyses. It attempted to avoid dysfunction in measurement by giving developers control over their data files; it keeps data only about the individual developers’ activities and doesn’t reference their name in the files.

Data from the toolkit is also extremely portable, creating a repo of data that developers can keep with them as they move between projects and organisations. The introduction of automatic analysis makes some analytics easy to collect but others more difficult, as it’s much more difficult to create analytics suitable to your needs compared to when the process was entirely manual. Johnsons team decided that the negatives of the toolkit outweighed the positive and set out to develop a new tool, which became Hackystat.

Hackystat

Hackystat was developed with focus on ways to collect software process and product data with the least amount of overhead for developers. It implements a service oriented architecture where sensors on development tools gather process and product data and send it to a server, where other services can query the data to build higher level analyses.

Hackystat has 4 main design features. The first is that it collects data on both the server side and the client side, this is to make the project useful in the modern development environment where a developer works on both their local workstation as well as on server or cloud based activities. Second is unobtrusive data collection, removing the loop and wasted time from manual data collection where the developer must stop working to record data about their work. The third feature is in depth data collection; Hackystat can collect data on a minute by minute or a second by second basis if the organisation wishes. Finally, it has features for both personal and group development. It can collect personal development data, but can also define projects and shared artefacts to represent group work, allowing Hackystat to track cooperation between developers when they edit the same file, work on the same project etc.

It has a lightweight architecture and is completely based on open-source components and standard protocols such as Apache Tomcat, XML, SOAP, etc. It does not make use of any back-end databases as it uses XML to store information.

However, Johnsons own research found significant ethical problems with the Hackystat tool. Some developers did not like the unobtrusive data collection, as they didn't want to install instruments that would collect data about their activities without notifying them. Secondly, the fine-grained data collection can cause discord in a group, with one user calling it "hacky-stalk", due to the transparency it gave regarding the working styles of individual members. This fine-grained data also led to the largest barrier to adoption of Hackystat, as developers weren't comfortable with management having access to such in depth data despite management promises to use it appropriately.

In essence, the tool focuses on individual data collection and use. The structure allows a reasonably high level of privacy, but it also has limited advantages for workgroups or entire companies.

PROM (PRO Metrics) (2.)

PROM was designed to help developers and to help managers keep projects under control. It does this by providing a different view to the user depending on their role in the project. Developers can only access their own data, including software metrics, PSP data and analysis results. This data shows them their inefficiencies and allows them to improve their approach to software development. A developer can also give another user access to his personal data to help him achieve better results. Managers cannot access the data for a single developer due to privacy concerns, but they can get the same information in an aggregated form to get the status of the whole project. This allows managers to get the important data that they need without looking at unnecessary details or affecting developer privacy.

Similar to Hackystat, the data collection procedure is completely automated to address the problems in acquiring data, and the process is unobtrusive so as not to interrupt the developers' main activity to collect data.

Many companies are now adopting an accounting method called Activity-Based Costing (ABC) to manage costs. This methodology is extremely difficult to apply in a software engineering context as human activities are hard to track. PROM aims to assist managers implementing this method by collecting relevant data.

PROM is an automated data acquisition and analysis tool that can collect code and process measures. Its main focus is on comprehensive acquisition and analysis to provide suitable metrics to improve products. The data that is collected is based on a wide range of metrics, including all PSP metrics, procedural and object-oriented metrics, and ad-hoc metrics to measure activities that a developer might do apart from coding, such as writing a requirements document. This tool collects and analyses data at multiple levels; the personal, workgroup and enterprise level. This is the key differentiation between it and Hackystat, as it gives a picture of the software company as a whole and preserves developers' privacy when providing data to managers.

The architecture design has three main constraints; it should be extensible to support new IDEs, new kinds of data and new types of analysis tools, IDE dependent plug-ins need to be as simplistic as possible and developer need to be able to work off-line. These constraints are satisfied in the PROM architecture. The PROM core is written entirely in Java, using open source technologies and standard protocols like SOAP and XML. Developers can write plug-

ins in any language that they desire but they must communicate to the plug-ins server using the SOAP protocol.

PROM is also component-based, based on the Package-Oriented Programming development technique. There are 4 main components in PROM. The database, which stores all the acquired data and information on users and projects. The server, which provides an interface to the database through SOAP services. This hides the low-level data model and provides functionalities through the web, allowing clients such as Microsoft Excel to access the data and perform custom analysis. The implementation is based on the Apache Tomcat application server. The plug-ins server, which collects data from the plug-ins, reduces redundant data and sends the source code to the WebMetrics tool for metrics extraction and collects the results. It then sends the results to the PROM server to be stored. And finally, the plug-in, which is an IDE dependant plug-in that listens to application events, collects data and sends it to the plug-ins server.

The data collected shows the effort spent in writing code and correlates it with metrics taken from the same piece of source code to determine its properties. It also tracks older versions of the source code to analyse code evolution, providing a full view of the development process. PROM dynamically creates web pages that perform data visualisation using Apache Tomcat, showing both tables and bitmaps containing graphs. Any PROM administration, such as project management and system status reports, that needs to be performed is managed through web pages also. The tool, also realising the importance and flexibility of manual data, supports manual data insertion through a webpage, allowing data collection on non-computer activities such as documentation.

Gamification (3.)

Gamification is yet to be fully utilised in a software engineering environment, but it is an area of great interest to me, so I considered research on how it's currently being used in the software engineering environment.

Gamification attempts to improve a user's engagement, motivation and performance when carrying out a task by incorporating game mechanics and elements to make the task more attractive. Research work has studied the application of gamification in software engineering for increasing the engagement and results of developers. Gamification has experienced significant popularity in the last few years, it uses the philosophy, elements and mechanics of game design in non-game environments to attempt to induce a certain behaviour in people and improve both their motivation and engagement. Some existing commercial tools that support software engineering have started to incorporate basic gamification elements, e.g. Visual Studio Achievements, however it has yet to be implemented on a large scale so I instead focused on research into the subject of gamification in software engineering.

From the studies examined in *Gamification in software engineering – A systematic mapping* by Oscar Pedreira et al., the most popular software engineering processes in research into gamification are software requirements, development and testing, followed by project management and other support areas. This isn't surprising, as development and testing share commonalities that make them more suitable for application of gamification. These features include their difficulty when compared to other tasks, and the need for collaboration. These features are a clear target for gamification, as it can make the activity both more and fun and attractive, it can also foster collaboration and competitiveness among players, enhancing their performance.

In areas like software requirements, gamification was used to overcome the problems of elicitation and analysis, such as a lack of user involvement. One study proposed a collaborative requirements elicitation environment, using a point based system to reward the participants for completing actions such as registering a new requirement, scoring an existing requirement, or commenting on requirements to clarify their meaning for all stakeholders.

In Pedreiras mapping, system implementation was the process area that was considered most in the studies they found, some proposals include gamification of code quality to reward developers, incorporating gamification mechanics into a crowd development scenario, and a gamified environment where the concepts of iterative software development process are

mapped to the aspects of a game, the different releases of a product under construction being levels, and the iterations of each build being mapped to different quests that are part of that level.

The gamification elements that were being applied across the studies were awards, points, badges, levels, quests, voting, ranking and betting.

There are of course ethical concerns when it comes to gamification. Gamification can lead to tense relationships between colleagues, e.g. due to increased competitiveness when a leaderboard has been implemented. It can lead to rating people and creating classes within the group. It can also create tension on the person as it could be seen as a monitoring system on their performance. Gamification also captures a lot of personal data; privacy policies and data protection would need to be augmented by ethical awareness. It can expose information users would prefer not to be known, for instance a leaderboard would reveal if someone was never a top performer/leader. Gamification ethics depend highly on the norms and culture of the organisation, and should be seen on a case by case basis, as the same mechanic for the same task may be seen from a different ethical perspective by a different user.

References:

1. Johnson, P. M. (2013) *"Searching Under The Streetlight for Useful Software Analytics"*
2. Sillitti, A. et al. (2003) *"Integrating and Analyzing Software Metrics and Personal Software Process Data"*
3. Pedreira, O. et al. (2015) *"Gamification in software engineering – A systematic mapping"*
4. Shahri, A. et al (2016) *"Towards a Code of Ethics for Gamification at Enterprise"*