Ways in which the software engineering process can be measured:

Brian Whelan – 15315707

Measurable Data:

To aid in the measuring the software engineering process, a number of different aspects of the process can be measured, both manually and automatically, across three main mediums – client-side data, server-side data, and physical data.

Client-Side Data:

Client-side data is data recorded locally on a developer's machine. The advantages of client-side data are that it can be recorded at a high-degree of accuracy (every second or less) and that it can be recorded both when the developer is online or offline. In the case of data recorded offline, it can be stored locally on the developer's machine until the machine is in a position to make contact with the server again. It is important however, for the capturing of client-side data to be unobtrusive, as if the developer is constantly in a state of being interrupted by data collection and transmitting it will impact the developer's workflow and thus the software engineering process – this will also lead to the possibility that the data being collected is not a truthful representation of the developer's workflow and habits. However, a high degree of invisibility also results in some ethical concerns in that the developer may not be aware of the degree to which data they produce is being collected and processed.

Examples of data which can be recorded locally on a developer's machine include buffer transition (implemented in HackyStat), which collects a data instance when a developer changes the active buffer within their IDE between files, allowing the amount of time a developer spends working on a particular file to be logged and measured. As an expansion to this, data can also be logged when a developer makes modifications to different methods within a file, or constructs new test cases or test case scaffolds.

As an extension to the tracking of test cases, the results of local runs of test-case frameworks such as JUnit and NSpec can be logged and used in conjunction with developer data to view progress towards test-completeness in the software engineering process.

Server-Side Data:

With the increasing growth in the use of cloud-based and server-side development tools in Industry, data can now be captured entirely non-intrusively through the daily interactions developers make with these online tools. Through cloud-based Git solutions such as GitHub and BitBucket, information based on a developer's commit habits can be tracked. Using cloud-based services such as Gerrit allow developers to provide reviews and direct feedback on commits before being merged to master – results of reviews and developer feedback can be logged to track code quality of a developer and which aspects of a developer's skills may need to be improved on.

Through cloud-based planning software such as Trello and Atlassian's JIRA, teams can create tasks and assign importance levels (bug/critical/feature) to these tasks. Developers can then be assigned these tasks and manually provide data such as time estimations and completeness statistics which can be accessed via REST APIs and tracked to find out how occupied a developer is, and how much of their time spent developing is allocated, and whether they can take on any additional tasks, and if so which ones. If they become blocked, the provided time estimations will also allow them to switch to a separate task while they wait for the block to clear, ensuring the development pipeline does not slow down.

Physical Data:

According to VersionOne's 11th State of Agile Survey[i], 98% of organisations utilising the Agile development found success in their project. With a huge increase in Agile development adoption in Industry in the last decade, new avenues for data collection have opened.

Through daily SCRUM meetings, developers can relay their progress to the entire team, and the current state of the team in regards to task completeness. While the goal of the SCRUM meeting is not to gather data on individual developers and rather provide the team with an idea of what has been done and what needs to be done, there is an opportunity for data to be logged here through manual data collection through a form based process such as PSP.

Similarly, during a Sprint Review at the end of a Sprint block there is the opportunity for developers to engage in an evaluation process such as PSP, as described in Humphrey's "A Discipline for Software Engineering". In the version of PSP described by Philip M. Johnson in his whitepaper, "Searching under the Streetlight for Useful Software Analytics", developers must fill out 12 forms encompassing the following topics:

- Project plan summary
- Time-recording log
- Defect-recording log
- Process improvement proposal
- Size estimation template
- Time estimation template
- Design checklist
- Code checklist

From my personal experience with Sprint Review in Industry, as the performance of the developer within the sprint, and the results the developer produced during the sprint are fresh on the mind of the developer, it would be a prime time to handle manual data entry for collection, albeit obstrusive and time-consuming.

In contrast to manual form-based data collection, an increase in Smartphone and Smartwatch adoption means that all developers contain a device capable of logging data on their development and workflow habits automatically, even when the developer is not at their desk performing development tasks. Devices such as the Zei allow developers to assign up to 8 tasks to a dice-like device placed on their desk. As they toggle through activities, they rotate the Zei to the task in hand (phone call, meeting, development, break) and the Zei keeps track of the amount of time spent on a task. As the developer is in charge of assigning tasks and tracking their time, they are fully aware of the data being collected and in charge of how much is collected.

Overview of the Computational Platforms Available

Physical Data Collection:

PSP:

PSP (The Personal Software Process) is a software development process which aids software engineers to better understand and expand upon their development performance by tracking aspects of their predicted and actual code development metrics. It also works to aid developers in improving their estimation and planning skills, manage the quality of their projects and avoid regressions and defects in the work they produce. It acknowledges that each engineer is different, and that to be most effective they must plan based off of their own data rather than generic data. It also acknowledges that to produce quality products, software engineers must feel personally responsible for the quality of their projects

The PSP is integrated into the workflow of a developer over three main stages – PSP0, PSP1 and PSP2.

In the first stage, PSP0, engineers are given a planning script. While performing work, engineers follow this script and record time spent working, alongside time defect data over the course of the time period. At the end of the task, engineers enter the post-mortem phase, in which they summarise these defects against the amount of work completed and log a summary form, which covers estimated and actual time values, estimated and actual defects injected and estimated and actual defects removed from the codebase over planning, design, development, compilation and testing.

In the second stage, PSP1, engineers start planning by defining the work that needs to be completed for the next task through requirements capturing, conceptual design and product size/resource estimations. This can be done using PROBE (PROxy Based Estimating), which represents portions of the product as objects. A product is broken up into object representations, and based on historical data from similar sized objects, the likely size of the finished product can be estimated.

Once the size of the product has been estimated, software engineers can make use of previous historical data to develop time estimations and spread the task time over scheduled hours to better manage the completion of each task.

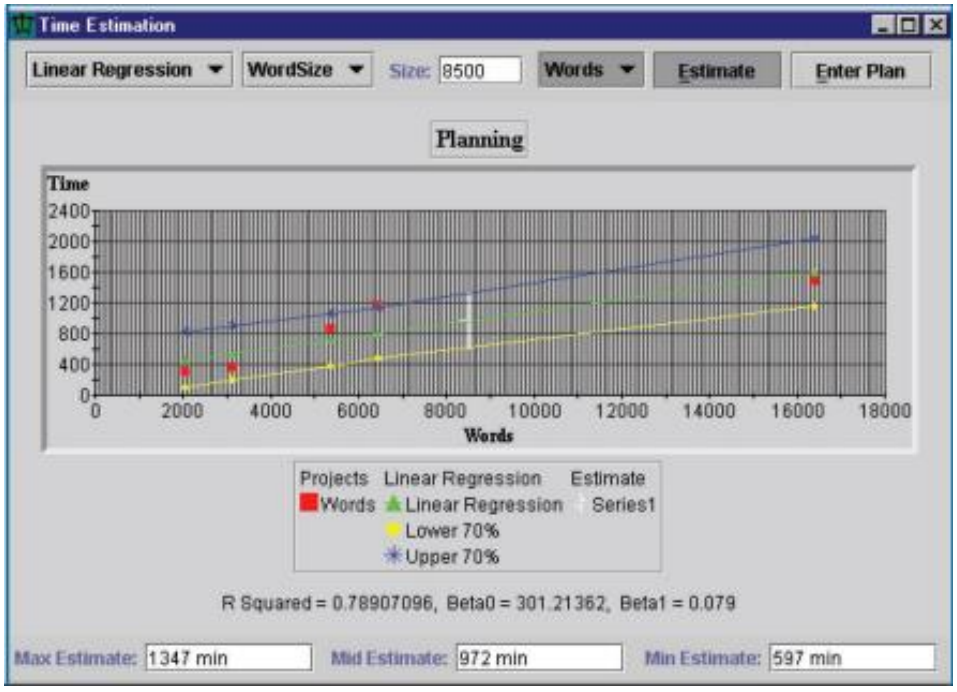
In the final stage of PSP, PSP2, software engineers focus on design review and code review. Defect/Regression removal and prevention are focused on, and software engineers learn to evaluate and improve their process by measuring task length and defect injection/removal levels in each phase.

Leap:

The Leap (lightweight, empirical, antimeasurement dysfunction and portable software measurement) toolkit was developed by Philip M. Johnson and his team at the University of Hawaii at Manoa[ii]. Afer two years of teaching predefined PSP processes, they began to suspect that the manual nature of the PSP led to significant data quality problems, as the consistency of data collected by the PSP varied.

The Leap toolkit attempted to address the data quality problems encountered by the PSP by normalizing data analysis. Unlike the PSP, rather than prescribing planning scripts it enables developers to maintain control of their data files, and records data only on the individual developer's activities, while maintaining a degree of anonymity through the exclusion of developer names. Although developers still manually enter data, the Leap toolkit automates analysis based on entered data and provides details such as indicating forms of regression without relying on the developer to discover these regressions themselves.

However, the downside to this automation is that the addition of new areas of analysis must be implemented into the Leap toolkit. As given by an example in Philip M. Johnson's whitepaper, "Searching under the Streetlight for Useful Software Analytics", if a developer suspected that interruptions (from other developers, phone-calls etc) are affecting their productivity, they would have to develop a new Leap toolkit module, rather than simply creating a new spreadsheet to track interrupts as per PSP.



1. Time Estimation Component of the Leap Toolkit

HackyStat

HackyStat is a toolkit which was developed in the opposite direction to both PSP and Leap[iii], in that it focuses on collecting data on the software process unobtrusively and with little overhead to developers, and provides software engineering goals based on what can be gathered via analysis on this data. It implements a service-orientated architecture, which provides monitors which hook on to development tools and gather data, which is sent to a server and exposed to external servers which can query this data to provide higher-level analysis. It features both client-side and server-side data collection, with tooling for data collection from editors, build tools and test tools, and on the server-side, build server and repository data.

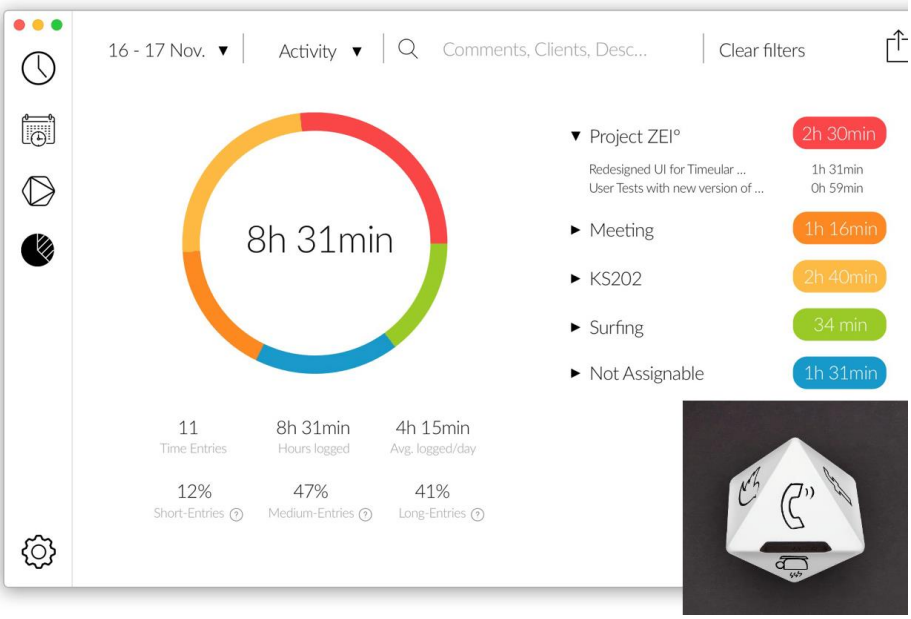
To provide streamlined, unobtrusive data collection, HackyStat collects data locally in the background and caches it while a developer works offline. Once a connection is re-established, it sends the gathered data to the HackyStat data repository. Data is collected in a second-by-second timeframe. Organisations using HackyStat can then build tools (such as dashboards) which can build upon and provide powerful high-level analysis on the data gathered by HackyStat.

2. A Software Dashboard based on HackyStat, which provides at-a-glance details on the health of a project

Hardware Based Solutions - Timeular Zei:

For time-based tracking, organisations can provide developers with hardware based solutions such as the Timeular Zei[iv]. The Timeular Zei is a small dice-esque product which features 8 blank sides. Developers can label each side using a permanent marker as to which productivity task this side should be associated it, and configure this side within the mobile app or desktop application to which the device is paired.

Once setup, developers can rotate the Zei to match their current status, for example, if a developer has to take a call, they may rotate the device to a call face, while in a meeting they may set the face to a meeting face, and while at their desk they can set the face to development. This allows developers to keep an accurate track of how they spend their time, and to learn and produce better plans. Through publicly exposed APIs it also allows organisations to track employee statistics, how individual developers function and in which conditions these developers thrive, in association with other data measurements.

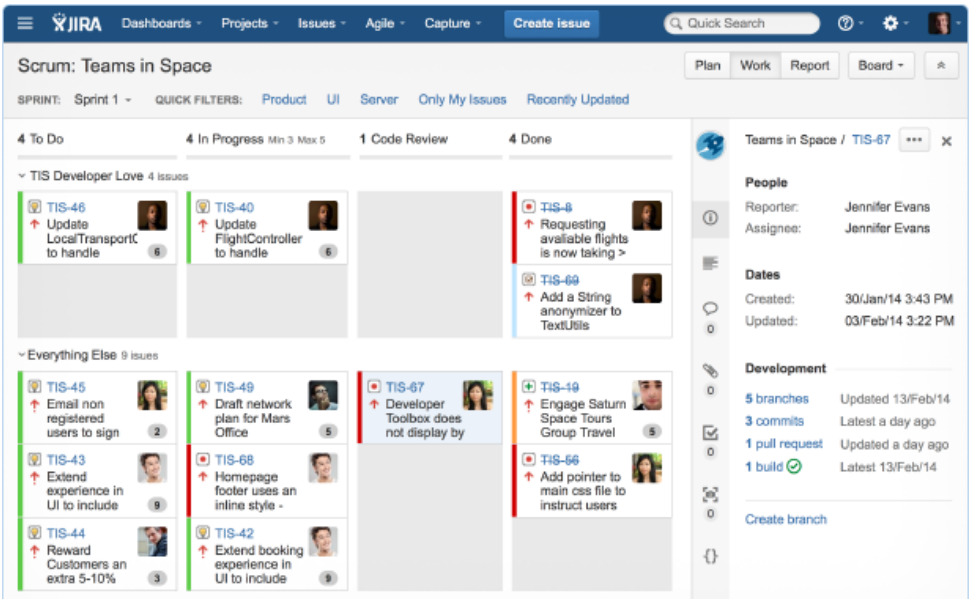


3. The Timeular Zei software and hardware device

Cloud-Based Project Management Systems:

Cloud-based project management applications such as Trello and Jira provide immense detail into the workflow of a team and how each member reacts within the team. Combined with AGILE methodologies, each new development task can be represented as a task card in a project management application, and these cards can link data such as the criticality of a task, dependant tasks and the containing sprint of a task.

Jira, Atlassian's closed-source tool for project management, provides a card-based structure for tasks. Each task is represented as a card on a board, and each board defines a Sprint. Cards can have issue links, which represent sub-issues (for example a larger task may be broken down into smaller tasks, and the parent task would provide issue links to each new child task). Tasks can also be assigned custom values, which allows additional statistics, such as time estimations, to be tracked.



4. Atlassian JIRA - Example of card based Sprint views

In my experience in Industry, after accepting a task in Jira, we would record the estimated time we intended to spend on a particular task. This data would be logged against our available time during the week (which would factor in time-off and meetings as unavailable time) so that we could be allocated tasks in such a way as to reach maximum capacity for the sprint. Once the task had been completed and was flagged for review, the actual time taken would be logged with the task on Jira, which allowed us to improve our planning skills by viewing charts based on estimated and actual development times. We could also produce charts to see the capacity of the team as a whole and whether or not we could take on additional tasks, or if a previously underestimated task was taking more time than expected or blocked, which tasks we could push forward into the next sprint so as to free time for current tasks which were underestimated time-wise, or if unexpected leave or meetings led to interruptions in task-estimations.

HR Software such as Workday also provide deep insight into management and leadership in teams, which directly integrate with employees through a web-based control panel. Through the built-in organisation chart page, organisations can get a top-down view of the state of the organisation and which developers work together and interact with each other. Alongside this, Workday is also used to manage time-off and can be used to integrate wellbeing programs into the company. Through Workday's Data-as-a-Service [v]organisations can access employee data centred around turnover, leadership effectiveness and even happiness – for example if an employee is involved in a wellbeing program through a device such as a Fitbit, the activity of this employee can be tracked and correlated with other workplace activity results to predict the happiness of the employee, and target whether employee happiness needs to be improved., which can directly correlate to productivity in the workplace. A high turnover rate would indicate a degree of unhappiness among employees and also lead to problems within the software engineering flow, and that action must be taken in order to reduce the turnover rate. In Hitachi's report on Measuring

Happiness Using Wearable Technology, it is reported that employees who are considered to be happy experience 37% higher work productivity.

In ResearchGate's whitepaper on the Network Effects on Worker Productivity[vi], it is shown that a productivity increase of 10% in a co-worker results in a 1.7% increase in productivity

The Algorithmic Approaches Available

In order to accurately process the data gathered by organisations on the software engineering process, the computational platforms employed to gather and process this data will generally make use of statistical algorithms to identify outliers and areas for improvement.

Health of the System:

Through graphing the number of lines of code present in a software project against test coverage over a time period, excluding quarantined test-cases, the level of coverage and thus the health of the current state of the system can be tracked, and whether or not the health of the platform has been improving over time or if regressions have been introduced causing tests to fail.

Method complexity can also be calculated through a combination of class file counts, methods per class file and method lengths over the entire scope of the project. This can provide insight into locations within the codebase in which code smells such as Bloaters[vii] (long methods, large classes and data clumps) that have increased to such large sizes that maintenance and development on these objects becomes hard to manage, leading to a decrease in productivity. may be occurring, and can thus be remedied to improve the health of the system.

Employee Productivity:

In Hitachi's whitepaper, "Measuring Happiness Using Wearable Technology", the duration of physical activity (in a state of activity, or inactive) against frequency is used to develop a frequency distribution of employee activity. They found that quantitatively, the probability of a person changing from active to inactive is inversely proportional to the duration of the activity (the 1/T rule).

Employees with a high level of happiness were found to have a low degree of deviation from the 1/T rule while groups with a lower level of happiness had a greater degree of deviation. As happiness is shown to directly impact productivity, it is shown that activity/wellbeing programs in organisations can contribute and improve the productivity of workers.

Ethics concerns

While current U.S. laws mandate that employee monitoring is legal, ethical questions arise from the use of this data in regards to employee effectiveness and happiness.

The largest three questions surrounding the ethics of employee monitoring in my opinion are that of how data collected from employees is being used, who has access to the data (privacy issues), and how aware employees are of the data collection and monitoring occurring.

The risks to participants in the process of data collection should be minimised, and it is important that data collected should either b e anonymised, or in the event that this is not possible, as confidential as possible. Developers may not feel comfortable with the feeling that they are constantly being monitored, and this may induce a nervousness in developers which would impact employee happiness and lead to a splintered relationship being leadership positions who have access to employee data. With HackyStat, it is mentioned that one user even referred to the product as “HackyStalk”, and that there was a high degree of uncomfortableness recognised amongst users.

It may also be the case that the data gathered by these systems is analysed incorrectly, and resultingly makes incorrect assumptions about employees and their productivity, which could lead to unfair terminations or other accidental and unwarranted procedures being enacted.

It could also be the case that the results of the analysis of captured data do not accurately reflect the work ethic of an employee, as individuals often have different working styles, or may be performing work outside of the scope of the data being collected, and as such the data collected may reflect badly on their character and the importance of their role within the software engineering process.

Finally, it is important that developers are aware of the degree to which their data is being collected. It can be assumed that data such as commit history and time estimations will be logged by the cloud-based platforms which manage them, as they are publicly available. However, software such as HackyStat performed data collection on more obscure data such as which files were modified, the time spent by the developer on the code, and the methods and test cases they implemented, which the developer was not aware of. It is important to question whether or not this information is truly of importance enough to warrant the violation of trust which occurs when the developer is not aware that this data is being captured

References

[i] https://explore.versionone.com/state-of-agile/versionone-11th-annual-state-of-agile-report-2

[ii] http://www.citeulike.org/group/3370/article/12458067

[iii] http://www.citeulike.org/group/3370/article/12458067

[iv] https://timeular.com

[v] https://www.workday.com/en-us/applications/data-as-a-service.html

[vi]https://www.researchgate.net/profile/Jan_Sauermann2/publication/285356496_Network_Effects_on_Worker_Productivity/links/565d91c508ae4988a7bc7397.pdf

[vii] https://sourcemaking.com/refactoring/smells/bloaters

PSP Details:
https://www.sei.cmu.edu/reports/00tr022.pdf

Measuring Happiness using Wearable Techology
http://www.hitachi.com/rev/pdf/2015/r2015_08_116.pdf

Legal and Ethical Issues of Employee Monitoring.
http://www.iiakm.org/ojakm/articles/2013/volume1_2/OJAKM_Volume1_2pp44-55.pdf

The Algorithms That Tell Bosses How Employees Are Feeling
https://www.theatlantic.com/technology/archive/2016/09/the-algorithms-that-tell-bosses-how-employees-feel/502064/

New Directions for Evaluations
http://onlinelibrary.wiley.com/doi/10.1002/ev.1136/pdf