CS3012 Report

Title: Measuring Engineering

Name: Andril Khlivniuk

Student ID: 14332864

Why measurements are important for any companies

All companies and organizations wants to know if they achieve the results that they expected, if their customers are pleased with their product, the ways in which they can improve the quality of their products, how to decrease the products price, how to make response to costumers better, how to save competitiveness with other companies and how to improve the growth of the company. To find the answer on questions like these it is necessary to find ways in which process can be measured and assessed. If we take into account software measurement only, then it will not solve all the problems, but it can improve the understanding of it. Moreover, if that measurements are correct they will help to manage processes of improvements. The success of any software engineering company is based on the ability to make predictions and responsibilities for its products. Right measurements allow software engineering companies to understand their abilities better and to make more accurate predictions, which will help to improve their services and products. Also, measurements help companies to detect trends, which later helps to make more modern products. Based on this, it can be concluded that the measurements are necessary for software engineering companies.

To make measurements effective they should be developed for supporting business aims of the particular company or organization. Also, measurements have to give effective economic information. It is not that easy as it might seem at first glance. Sometimes to make right measurements as hard as to develop software development. The difficulty is that there are too many things to measure.

To determine what measurements necessary for status and performance of software processes, it is necessary to identify goals and issues of that company. To find these goals there are software management functions that can be divided into three main classes: project management, process management and product engineering. Each of these classes cover different issues and aims.

What is software process

For different people process means different things, especially in software context, so it is important to clearly understand what term software process means.

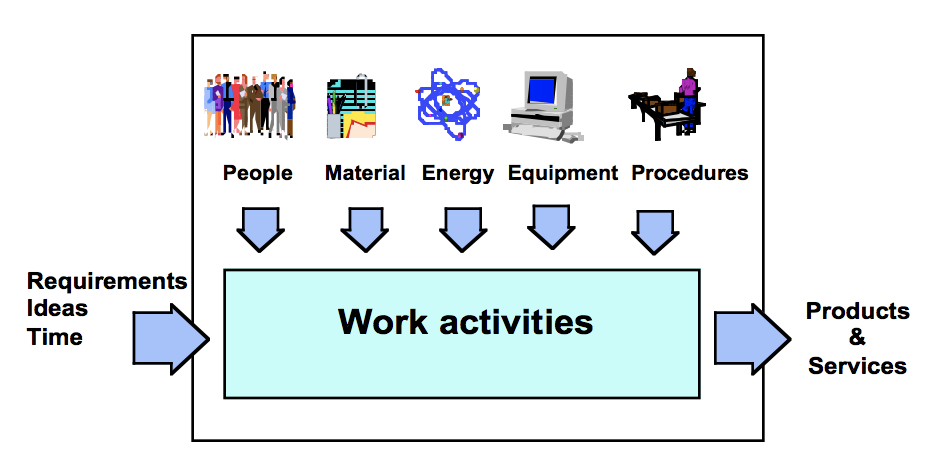
“A process can be defined as the logical organization of people,

materials, energy, equipment, and procedures into work

activities designed to produce a specified end result.”

*Gabriel Pall, 1987*

The illustration of the process is shown on the picture 1. This illustration shows how all processes works, not only software process.



*picture 1.*

Software process composed from activities that leads to production of the software. Such activities include: planning, estimating, designing, implementation (coding testing, reviewing, measuring), verification, validation and evolution of the software.

Software process management

Management of software process implies successful management of work processes, which are responsible for developing, maintaining and supporting software systems and products. Successful management means that all software products and services are correspond to the goals of the organization, fully satisfy clients demands and leads to the predictable results. Predictable results do not mean always the same results, results vary, but in predictable borders. Therefore, the aim of the software process management is to ensure that all processes that you operate and control are meet customer requests and were within predictable limits.

What is streetlight effect

The streetlight effect is some form of observation bias, that was named in honor of following statement:

*A drunk has lost his keys and is looking for them under*

*a streetlight. A police officer comes over and asks what*

*he’s doing. “I’m looking for my keys,” he says.*

*“I lost them over there.” The policeman looks puzzled.*

*“Then why are you looking for them all the way over here?”*

*“Because the lighting here is so much better.”*

Researches from Collaborative Software Development Laboratory in University of Hawaii at Manoa for more than 15 years tried to find materials that will help software developers to understand and improve products and processes of software development. This research shows that “searching under the streetlight” statement helps to understand better researches in this sphere. Searching under the streetlight research includes collection and analyzing information that was obtained with social, political and developmental impact. The easier is to get information, the worse and limited this information will be. The good example of this, is collecting information from configuration management repository. The repository is public, that makes collecting of analytics easier and most of the developers do not mind so that their data is used. The result of such analytics would be very small range of development activity. Therefore, it could be very difficult to get analytics that will provide extensive results.

Personal Software Process

Before first research on analytics will be considered it is important to introduce term Personal Software Process (PSP). Personal Software Process comprises of methods, forms and scripts that shows for software developers how to plan, measure and control their work. PSP can be used with any programming language or design methodology and it can be used with most of aspects of work with software development, including grammar requirements, tests implementation, determination of processes and defects removing. When software developers use PSP, the aim of the process is to produce products with: minimum defects, in time and within the planned costs.

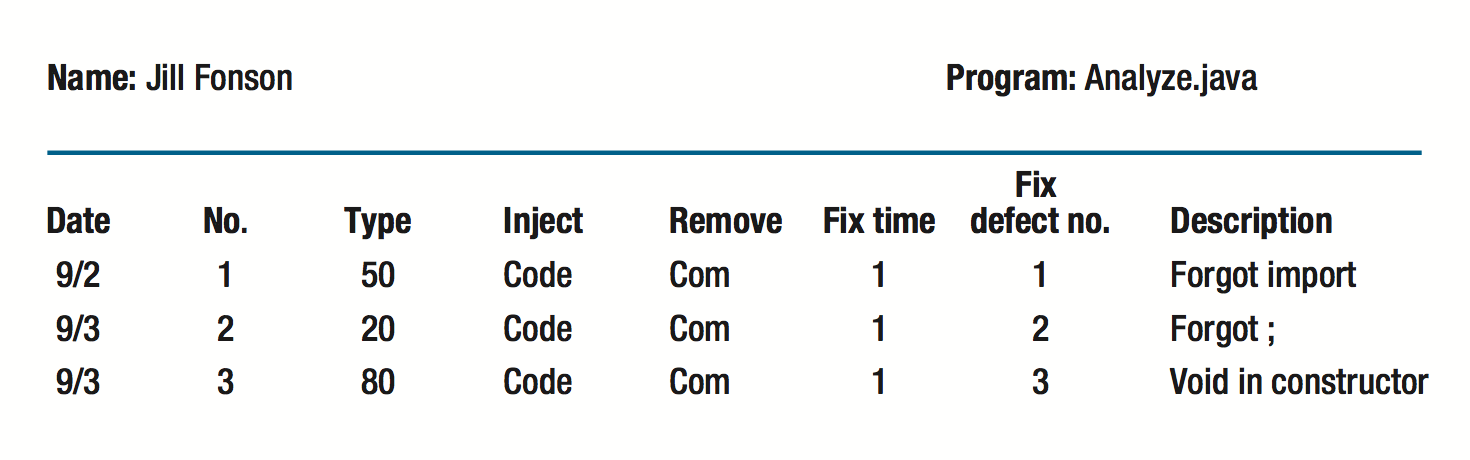
Team Software Process

Team software process (TSP) in combination with Personal software process is a guide for teams of software developers on how to develop software-intensive products. TSP helps to make secured, reliable and lower costs software.

Candle Light vs Streetlight

There is a Watts Humphrey's book that describes CSDL research on collecting of analytics with using Personal Software Process in 1996. This research was described in three global approaches. First approach showed how to adapt analytics of the software development processes for each programmer. Then, these analytics showed how programmers can achieve improvements. And this incremented professional and academic adoption. In this research PSP uses digital tables, data collection and analytics. In one of the versions of such PSP programmers should write their: project plan summary, time-recording log,

defect-recording log, process improvement proposal, a size estimation template, a time estimation template, design checklist, and a code checklist. Example of defect-recording log is shown on picture 2.



*picture 2.*

It was very hard to get these analytics, because there are more than 500 values that programmers have to calculate manually. The author of the book Watts Humphrey say: “It would be nice to have a tool to automatically gather the PSP data. Because judgement is involved in most personal process data, no such tool exists or is likely in the near future.”. Humphrey invoke developers to modify forms and procedures, that was described in his book, to solve various tasks. In terms of streetlight statement, that was presented earlier, this version of Personal Software Process can be compared with a candle light, because this method makes specific analytic for given situation and the fact that data for PSP calculating manually makes it as brittle as candle light.

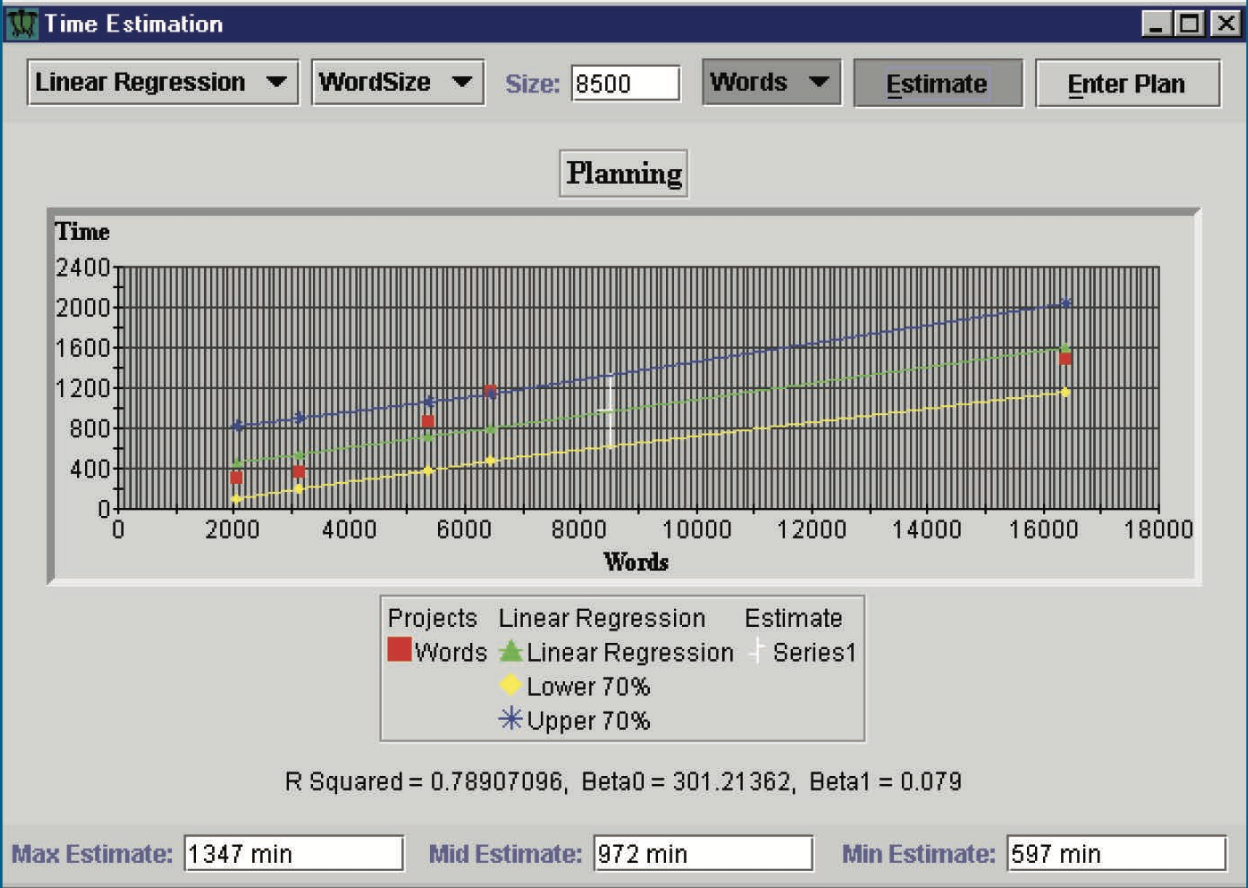
Another comparison with candle light could be the fact that manual calculations make PSP flexible, because developers can search for the most suitable analytics, which could be compare to the candle light, as you can move it in the darkness. In spite of all good sides of manual nature of PSP, researches showed that this approach of using Personal Software Process created serious danger for quality of data. A test was conducted in which more than 30000 data results were obtained using PSP. Manual calculations sometimes gave wrong results, although the coefficient of the wrong calculations was really low (less than 5 percent). To decrease amount of wrong calculations Leap toolkit was developed. LEAP - (lightweight, empirical, anti-measurement dysfunction, and portable software process measurement).

The Leap toolkit

Leap is a special toolkit that is solving a problem with wrong calculations that developer do manually (as was observed before). This happens due the fact that Leap toolkit doing automation and normalization of data analytic. This approach is easy, because it tries to avoid dysfunctions, which allow developers to control their data. It creates repository with data of the particular process and developer can transfer this data from one project to another. For example, one of the Leap toolkit components is component that support time estimations based on the private historical data and regression analysis selection. This example is illustrated on picture 3.

Leap toolkit makes some analytic easier, but other more difficult. For example, if the developer thinks that interrupts in his work have influence on his productivity. I this case he or she will need to create new Leap toolkit component instead of just doing simple spreadsheet form.

After many years of researches, it was established that Personal Software Process cannot be fully automatic and needs a lot of manual calculations. Also, it was found out that this system does not pay off the investments.



*picture 3*

Hackystat research

Next research called Hackystat, which lasted 10 years. The main idea of this research was to find what useful analytics it is possible to obtain from the software developing if the collection of this analytics would be free. PSP and Leap toolkit, unlike Hackyastat, although they gave a good analytics results, better planning and reduced defects, but it required high costs. For data analysis Hackystat were developed different ways of data collection for software developers.

Data collection was implemented by service-oriented architecture where sensors are connected to the development tools and they are collecting process and product data, then send this data to the server and from this server other services can request data for higher level analysis. Collection of the data occurs on both server and client sides. For data collection, different tools where developed. For the client side servers were developed tools, such as: editors, build tools, test tools. For the server side, such tools as: build servers and configuration management repositories.

Another feature of Hackystat system was inconspicuous data collection. As shown by practice one of the most disappointment aspects for software developers was interrupts during their work for recording of work that they did already. So, in Hackyastat system developers are not interrupted to record what they worked on. For the client side data collection is working in following way. When developer is working on his project, this system locally caches all the data and then, when developer reconnect, it sends all data collection to the Hackystat repository.

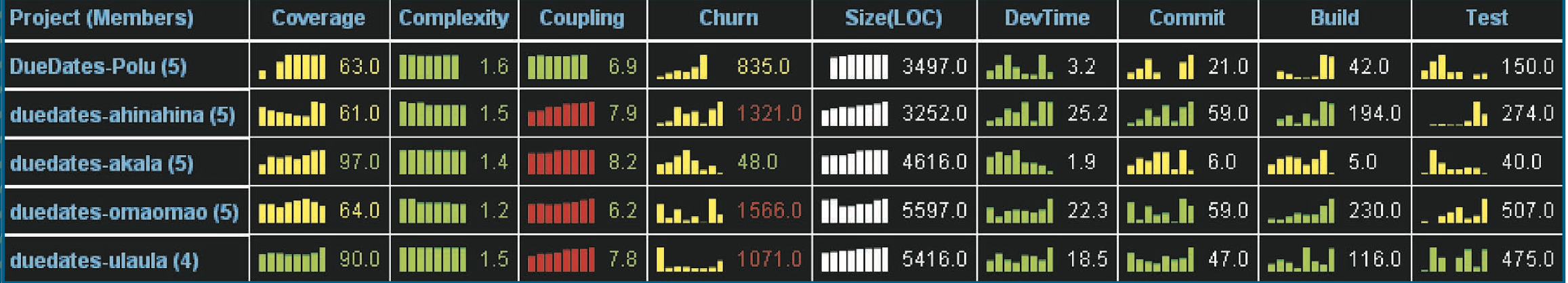
Next feature of Hackystat system is collecting data when certain actions were taken. For example, data could be sent to the Hackystat repository in certain time period (every 1 minute or 1 second), or data could be collected when developer changed certain method, or when he or she is testing this method.

Hackyastat system allow developers to work in groups. In this way developers can share same project and see what changes made another developer. All changes of each individual developer are tracking by system.

Turning to shortcomings of Hackyastat system, in spite of all good sides of data collection without interrupting developer, some people found this technology unacceptable. They did not want to install software that collect their data without their knowledge.

Next disadvantage of the system was data collection on the client side. Collecting data that is not interconnected may cause a discord in a development group. Also, developers have reported that it is not comfortable to manage client side, fine-grained data, despite that data provides good analytics about system development.

These problems can be explained if we will get acquainted with software intensive care unit (ICU). Software ICU collecting software artifacts structural metrics based on Hackyastat system. Results of that collection is displayed on picture 4.

**

*picture 4*

As shown on left side of the Software ICU on picture 4 displays such metrics as: coverage, complexity, coupling and churn. The most recent values are colored in green, red or yellow, because they display health status. Next metric is size, it has a white color, because it does not relate to health status. Health metrics collect data of the project and then use it for comparison.

The right side of the picture 4 especially interesting, because it shows information about every single developer. There are 4 metrics that describes his activity. First metric “DevTime” shows how much time developer spent in integrated development environment (IDE) and how much time developer spent working on specific project. Second metric is “Commit” that show how often developer committed his code and how many lines of code was written between commits. Next metric is “Build” that indicates how many times developer built his project and how many of that builds was successful. Last metric is “Test”, it shows how many tests was written for certain method and indicates was that test successful or not. It is possible to open each metric to see more detailed information about that metric.

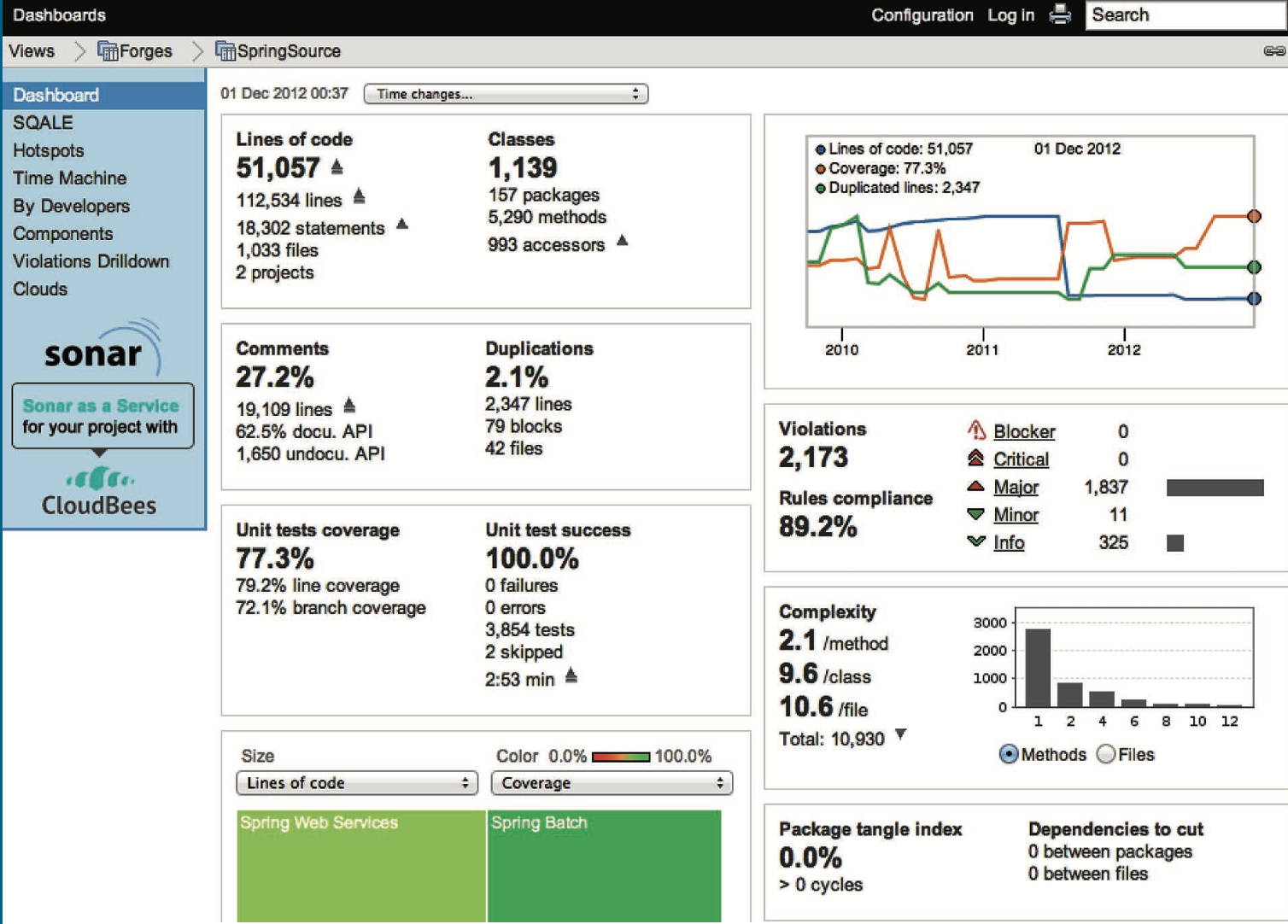
Some developers say that this data collection makes them feel uncomfortable, but there is no another way to see statistic. Main principle of agile software development is build early and often. And this data makes a full picture of how developers adhere to this principle.

There is another system called Zorro, based on Hackyastat system, allows to see even more behavior information about every single developer. It is providing second by second information about what developer doing, it can determine each step of the developer in project. This system, as well as the previous one, made some developers feel uncomfortable.

In general, Hackyastat system that collects data automatically have its own advantages and disadvantages. On the one hand, it collects data and provides most detailed information about every single developer, wherein without distracting him or her. Moreover, developers do not need to make analysis about themselves. On the other hand, Hackyastat and similar to it systems are makes to feel some developers uncomfortable, because they know that everything that they do is recorded.

Recent systems

Nowadays systems for software development analytics became very popular. Some of the most popular companies that provide such systems are: DevCreek, Ohloh, Atlassian, CAST, Parasoft, McCabe, Coverity, Sonar. These systems usually collecting data to build analytics based on 3 main sources: systems of configuration management, system building, and system that tracks defects. A good example of such service is Sonar. Display of that system is shown on picture 6.



*picture 6*

Systems like that have some strong sides. Data collection is fully automatic and data is already available. System just doing analytics of this data and presenting it on the screen. Costs for managers and developers are low, because data collection and analytics are fully automatic. And finally, results are uncontroversial, because system make analytics from product characteristic, instead of monitoring developers behavior.

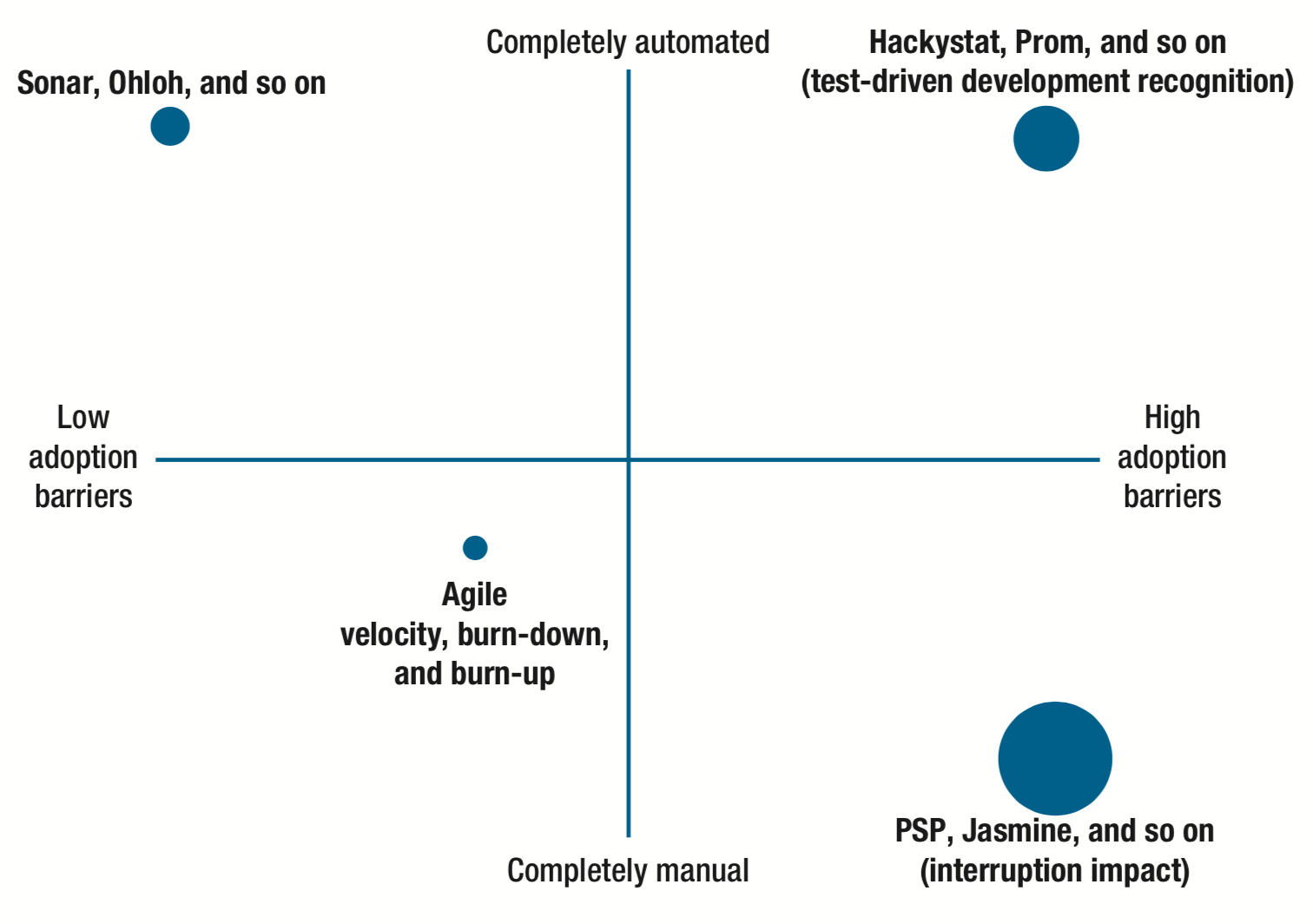
These systems, also, have some disadvantages. Firstly, for example, developer might think that distractions during his work can have influence on final analytics, but these systems do not support data collection on the client side. Secondly, system cannot show analytics that based on methods developing and testing.

Summary

From all these researches that were discussed above we can draw a conclusion that software process and products are can involve 3 stages:

1. Analytics is obtained from the degree of automatization, overhead costs and management of developer.
2. Analytics could be wrong because of the wrong measurements, which could be corrupted by social or political technique or technology.
3. The technique or technology catholicity. In other words, is how wide is the range of analytics.

The degree of these three stages for different systems that was discussed above is displayed on picture 7. The width of the circles shows how wide is the range of analytics.



*picture 7*

Basis on information that was obtained from the experiments using different software measuring systems we can conclude that there is no perfect system to get analytics from software developers, all of them have its own advantages and disadvantages. Some of them fully automated and do not require large investments, some are time and costs consuming. Most likely only collection of data from different systems can give most wide range of analytics.

Future researches should take into account some aspects.

* Older technologies and systems does not have to be worse than newer, they just might give different types of analytics. The example of such systems are Sonar and PSP.
* Software developers who think that interruptions have influence on their productivity would not give right analytics.
* Some functions of old systems, such as PSP, are useless. For example, recording syntax errors are unnecessary, due to new IDEs, such as Eclipse that shows the errors before compilation the program.

References

(PSP), T. (2017). *The Personal Software Process (PSP)*. [online] Resources.sei.cmu.edu. Available at: https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=5283

A. Florac, W., E. Park, R. and D. Carleton, A. (1997). Practical Software Measurement: Measuring for Process Management and Improvement. pp.1-12.

Johnson, P. (2013). Searching under the Streetlight for Useful Software Analytics. *IEEE Software*, 30(4), pp.57-63.

Sei.cmu.edu. (2017). *Team Software Process | Overview*. [online] Available at: https://www.sei.cmu.edu/tsp/