Yitpin Chin
15335586
CS3012
16 November 2017

Measuring Software Development Process

What is software engineering?

Software engineering is a discipline of engineering. Software engineers have to apply the principle and techniques of engineering to the design, development, testing, deployment, maintenance and management of software systems.

Software engineering includes a number of fields that covers the process of software development such as: defining the requirements, designing the software, constructing the software, maintaining the software, software configuration management, software development process and creation, software engineering models and methods, checking software quality and software engineering professional practices. Software engineering focuses more on the techniques of developing quality software for use in industry compared to computer science which is more focused on the theory and algorithms behind software development.

Like 343many other types of businesses and organisations, a software engineering company wants to be able to gather data and metrics on the performance of its various stakeholders, especially the ones that directly contribute to the process of developing new software and systems. They can then use the results to further improve its offerings and reduce on any inefficiencies. However, the metrics these firms require are harder to gather than in other engineering fields due to the intangible nature of software development. In this report, I am going to explore the history behind the measuring and analysing of data that goes on in the software engineering field.

Measuring and analysing the process

In the early days of trying to capture measurable data about software engineering process, there was much resistance to the practice as developers were reluctant to do it as developers did not view it as an important activity and would rather focus their efforts into coding and refining their software. Data captured early on in this field required the developers to manually input data into a simple spreadsheet and the analysis had to be done manually as well. All this required substantial effort on the developers' and managers' part and thus were deemed not to be a worthy pursuit by most software engineering firms. Manual collection of data tended to be expensive and unreliable due to human error factors. These were the problems facing the original version of the Personal Software Process (PSP). *"The manual nature of the PSP makes its analytics fragile, in the same way a candle flame is easily extinguished. On the other hand, the manual nature also makes the PSP's analytics flexible"* [1]. Although PSP can yield high quality analytics, its overheads was way too high to justify integrating PSP into the software development process.

There are two types of measures needed in software development: product metrics and process metrics. Product metrics is concerned with things like code length, maintainability, reusability, complexity etc. Process metrics covers number and type of changes in class or in a file, editing time, commits etc. There is no consolidated way to measure these in the software engineering due to its intangible nature. "*Most of software development costs are human resources costs: experience, skills, etc. Moreover, the productivity of very good programmers is ten times better than average. For these reasons, it is very important to understand how top developers work and to encourage all developers to adopt a process that helps them to achieve the best possible results.*" [2]

In order to increase the adoption rate of measuring the software development process, other methods had to be introduced to overcome the issue of developers having to manually entering data. The whole process had to be changed to overcome the resistance to this practice by both developers and companies. The obvious solution to this is to automate the process. The data collection process has to be as unobtrusive as possible to the developers to enable them to focus on the important parts of their job. It also has to have low overheads to convince the money men inside the companies to take it up.

A popular method of doing this is to attach plug-ins to the developers' toolset that would enable the collection of various types of data as the developers coded away.

As this is done in the background it doesn't take precious time away from the developers. The data collected is then sent on to be analysed by a metrics analysis software. This is also automated resulting in lower overheads and costs.

The findings of the analysis is then accessible by both managers and developers. What kind of analysis is available to each party is dependent on the software they use to do this. The findings can be leveraged by developers to help them in improving their development process. For managers, the benefit is easier cost managements and control of projects.

One paper advocate "*Software Intelligence (SI) as the future of mining software engineering data, within modern software engineering research, practice, and education. SI offers software practitioners (not just developers) up-to-date and pertinent information to support their daily decision-making processes. SI should support decision- making processes throughout the lifetime of a software system not just during its development phase.*" [3]

Today, many decisions regarding software systems are based on pst experience and intuition. This non-exact decision process can lead to wasted resources, inefficiency and higher cost when building large, complex software systems. The vision of SI is not yet a reality but advances in the field of Mining Software Repositories shows great promise and give credence for realising SI in the future.

Computational platforms available to measure and analyse

There has been a few computational platforms throughout the years to collect metrics on the software engineering process. Researchers at the Collaborative Software Development Laboratory (CSDL) at the University of Hawaii have looked for analytics that help developers understand and improve development process for the past 15 years. They looked at different platforms to do this, with these platforms improving in capabilities as the years goes on. According to them, "*the easier an analytic is to collect and the less controversial it is to use, the more limited its usefulness and generality*." [1]

They first looked at the Personal Software Process (PSP). With this platform, developers must fill in forms that about what they did and what went wrong. These forms required substantial effort to fill in.

Name: Jill Fonson **Program:** Analyze.java

Date	No.	Type	Inject	Remove	Fix time	Fix defect no.	Description
9/2	1	50	Code	Com	1	1	Forgot import
9/3	2	20	Code	Com	1	2	Forgot ;
9/3	3	80	Code	Com	1	3	Void in constructor

FIGURE 1. A sample defect-recording log. In the Personal Software Process (PSP), even compiler (syntax) errors are recorded. Developers typically find this aspect of the PSP to be onerous.

PSP's manual nature has benefits and disadvantages. It allowed users to only search for analytics best suited to their needs. On the other hand, its manual nature resulted in data quality concerns which In turn lead to incorrect conclusions being drawn. To address these problems, the LEAP toolkit was developed.

LEAP addresses the data quality problem by automating the data analysis part but developers still had to manually enter most of the data required. LEAP creates a portable repository of personal process data that developers can bring with them as they move from projects to projects. "*But by introducing automation, the LEAP toolkit makes certain analytics easy to collect but others increasingly difficult. After several years of using the Leap toolkit, we came to agree with Humphrey that the PSP approach could never be fully automated and would inevitably require significant manual data entry."* [1]

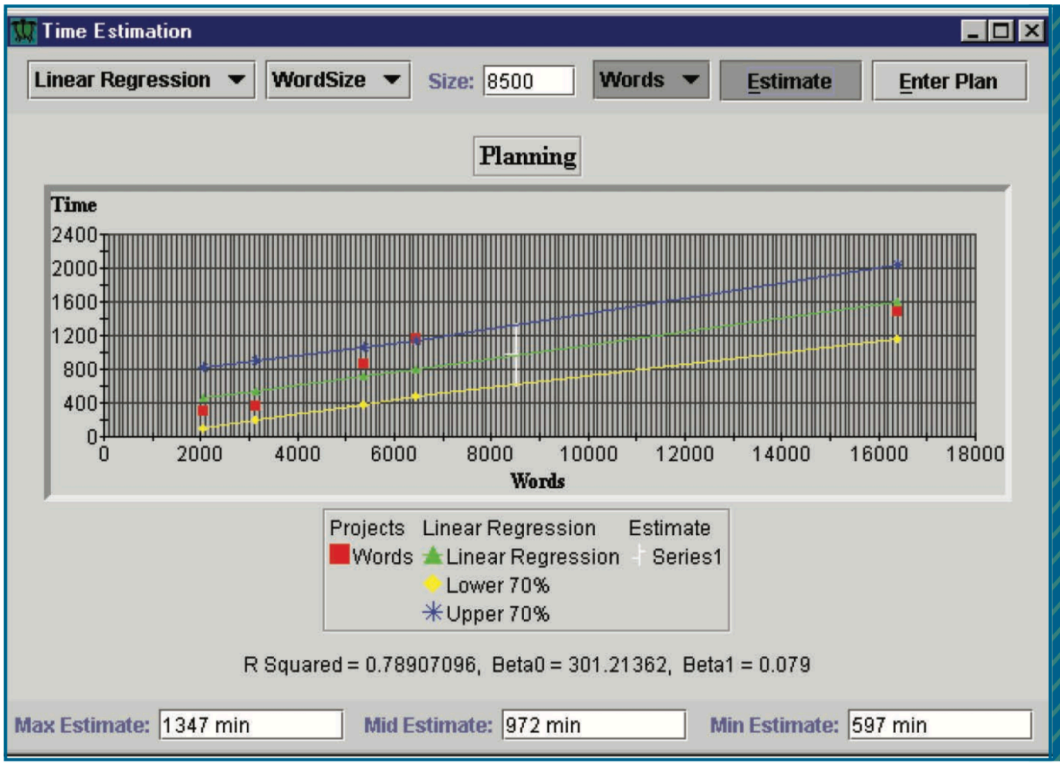
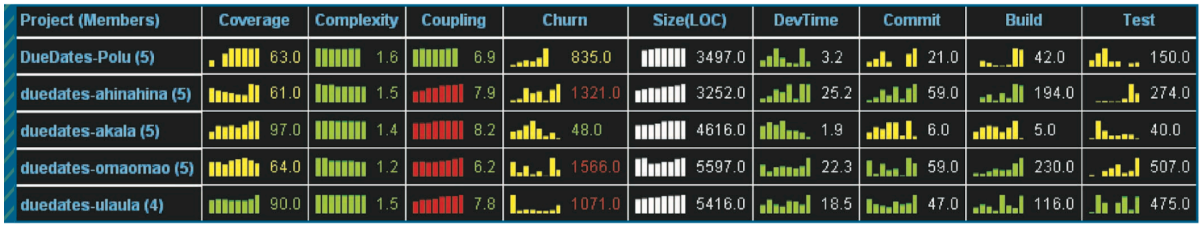


FIGURE 2. The time estimation component in the Leap (*l*ightweight, *e*mpirical, *a*ntimeasurement dysfunction, and *p*ortable software process measurement) toolkit. Unlike the PSP, no Leap analytics are paper-based.

The next step in CSDL's research was the Hackystat project. "*Hackystat implements a service-oriented architecture in which sensors attached to development tools gather process and product data and send it to a server, which other services can query to build higher-level analysis.*" [1] Hackystat has four major design features. The first is the collection of both client and server side data. Modern development usually includes local activities and server or cloud based activities. Hackystat collects data from both side for a more comprehensive analysis. The second feature is unobtrusive data collection so users shouldn't notice when data is being collected. It also allows for offline data collection which will send the data when the developer reconnects. The third feature is fine-grained data collection. Data could be collected on a minute to minute basis or even as developers is editing their code in real time. The fourth feature is both personal and group-based development. As well as tracking personal development, Hackystat can track collaborative efforts between a group.

Three major social or political problems were discovered during the CSDL's research of Hackystat. First, the unobtrusive nature of the data collection were not welcomed by developers as they did not want data about their activities to be collected without them being aware of it. Second, the fine-grained nature of the

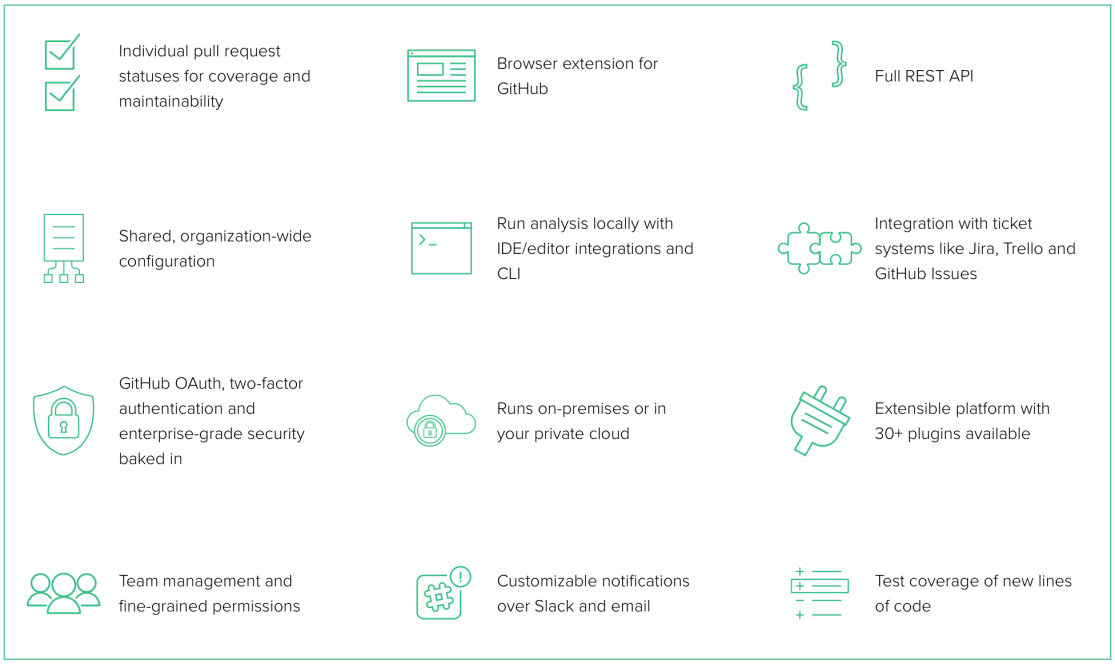
data collection can create disharmony amongst a group of developers as they can view what and how the other members of the group contributed to a project. Third, developers were uncomfortable with management's access to such fine grained data, even if it is being used appropriately by management. "*CSDL research suggests that such a representation of individual developer behaviour makes some developers uncomfortable; however, it's necessary to provide certain kinds of insight. For example, a principle of agile software development is "build early and often." The Software ICU can measure the ex- tent to which developers adhere to this.*" [1] These issues have proven to a barrier to industrial adoption of Hackystat technology.



Project (Members)	Coverage	Complexity	Coupling	Churn	Size(LOC)	DevTime	Commit	Build	Test
DueDates-Polu (5)	63.0	1.6	6.9	835.0	3497.0	3.2	21.0	42.0	150.0
duedates-ahinahina (5)	61.0	1.5	7.9	1321.0	3252.0	25.2	59.0	194.0	274.0
duedates-akala (5)	97.0	1.4	8.2	48.0	4616.0	1.9	6.0	5.0	40.0
duedates-omaomao (5)	64.0	1.2	6.2	1566.0	5597.0	22.3	59.0	230.0	507.0
duedates-ulaula (4)	90.0	1.5	7.8	1071.0	5416.0	18.5	47.0	116.0	475.0

FIGURE 3. A Software ICU (intensive care unit) display based on Hackystat. The Software ICU assesses a project's health both alone and in relation to other projects.

Over the past few years, specialised services for software product analytics have been on the rise. CodeClimate, DevCreek, Ohloh, Atlassian, CAST, Parasoft, McCabe, Coverity, Sonar, and others offer these services. "*These services' analytics are typically built from one or more of three basic sources: a configuration management system, a build system, and a defect-tracking system*." [1] Example of the features that these services offer (example is CodeClimate):



These systems have automated data collection. Analytic techniques is applied to the data and the results are displayed to the users in a simple, easy-to-use graphical interface. Since the data is gathered automatically from a repository, there is low overhead for management. The data collected focuses on product metrics and not on the developers' behaviour that produce them. This sort of approach will help to put developers' privacy concerns at ease. A downside of this it that this approach doesn't offer insight into developer practices as not behavioural data is collected.

In summary, these different approaches make different trade-offs so the chosen approach to gathering and analysing metrics is dependent on what sort of data they want to collect and the amount of overhead they can afford. If they want to metrics about developer behaviour and practices, an approach based on PSP should be chosen but at the cost of manual entry and a higher overhead. If they want low overheads and little manual entry, they should opt for the specialised analytics software that is available in the market such as CodeClimate or Sonar but they won't get as comprehensive an analysis.

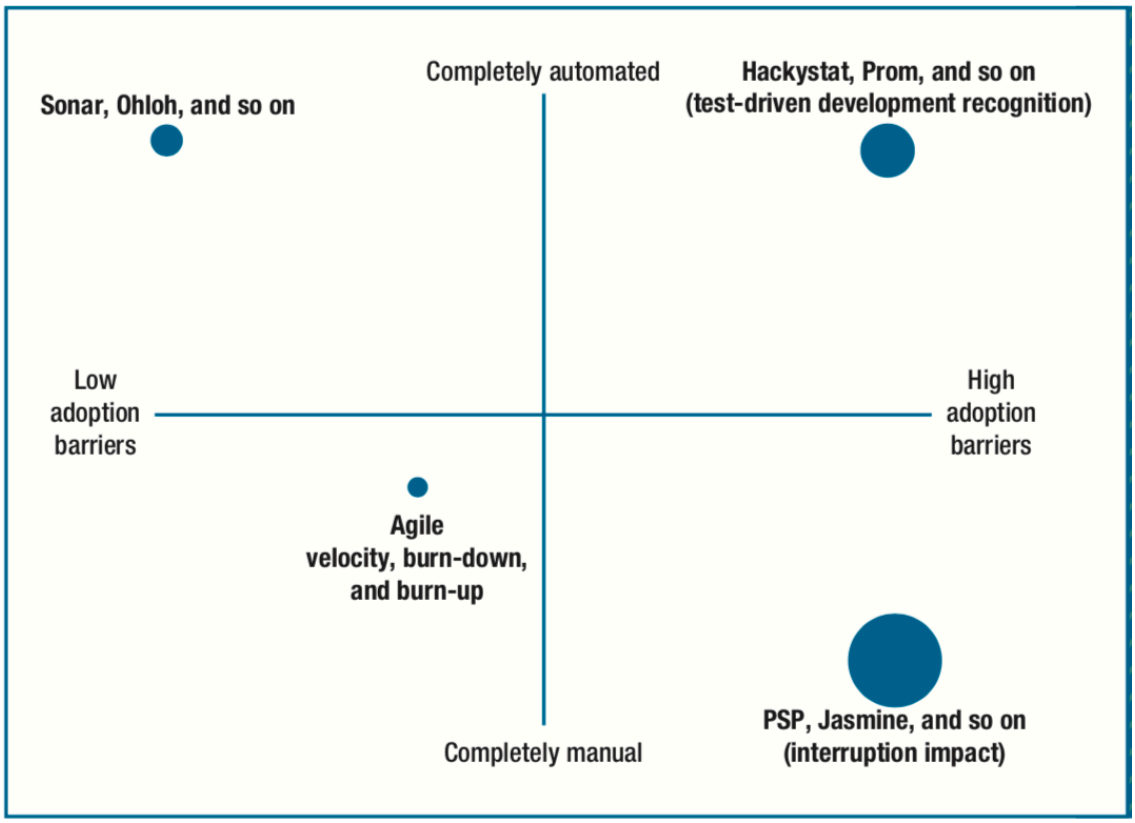


FIGURE 6. A classification for software analytics approaches, including automation, adoption barriers, and the breadth of possible analytics the approach supports (indicated by the circles' size). In the parentheses are analytics that would be difficult to implement with techniques or technologies in the other quadrants.

Algorithmic approaches available

One interesting field that could make a big difference to the gathering and analysis of software engineering data is Computational Intelligence (CI). Things like AI and CI are becoming increasingly more common in this day and age and will definitely become even more influential in all walks of life in the future. These two fields have many applications that will change the way we live and work. I'm going to explore CI in relation to the subject of interest of this report.

CI is the ability of a machine to learn a specific task from a set of data and/or from experimental observations. CI is a new way to solve modern, complex problems where traditional mathematical techniques are not adequate anymore. CI is a set of computational methods and approaches to address real world problems where the process might be too complex for mathematical reasoning, where the process is of a random nature and where there is an amount of uncertainty during the process. These are characteristics of many real world problems that makes it hard to translate into mathematical language. CI is able to deal with inexact and incomplete knowledge as it tries to approximate human reasoning.There are five main components to CI.

Fuzzy Logic is measurements and process modelling made for real world problems. It deals with incompleteness. Because of this it has a wide range of application such as decision making. It doesn't have learning abilities however.

Neural Networks has three components that work in tandem. One processes the information, one sends the signal and one controls the signals that are sent. It is a distributed information system that learns from experimental data. It accounts for fault tolerance like human beings.

Evolutionary Computation is based on the theory of natural selection where only the strongest survive. Applications of this are found in the area of optimisation and multi-objective optimisation where traditional mathematical techniques are not adequate to solve a wide range of problems.

Learning Theory in CI tries to approximate as closely as possible the reasonings of humans. It helps understanding how cognitive, emotional and environmental effects and experiences are processed and then making predictions off it.

Probabilistic Methods tries to evaluate outcomes of systems defined by randomness, like the real world. It provides possible solutions to a reasoning problem, based on past experiences and knowledge.

These five components work together to enable CI to be applied to a wide range of applications. An example of this is in the cost estimation of a future projects where the cost modelling is based on a historical database of past projects. However many of these historical databases contains missing data so the common practice has been to ignore observations with missing data but this results in inaccurate estimates. The fuzzy logic component of CI would helpful in this case as it deals with incompleteness of data. Neural networks connects all the needed data and it improves the CI as more experimental data is fed to it. By feeding it with a vast amount of data, it can be trained to give accurate results/decisions depending on the application. It also accounts for fault tolerances (which occurs regularly in real life) which enables CI to be use for real life applications such as software fault tolerance. Evolutionary computation can be applied to a problem to find the optimal solution. This could be optimising the number of developers needed for a project so that no resources is wasted, finding the optimal budget to assign to a project, which developer to fire or optimising the software development process. Learning theory is the the component of CI that tries to approximate behaviour and reasoning. As CI is used more and more to a problem, learning theory helps to make future predictions based on past experiences. For example, if a project went horribly wrong, this component will try to understand why it happened and use it to improve the CI model so that better decisions are made in the future. Probabilistic methods deals with randomness which is pretty much a factor in most if not all real world problems.

If CI is indeed the future of measuring and analysing the software development process, there are clear tangible benefits to it. These models can be trained by feeding vast amounts of data that is too much for humans to comprehend and because these models take more into account more data, it should in theory lead to better decision making and optimising.

Ethics of this kind of analytics

Measuring and monitoring employee performance metrics is very common in all kinds of businesses. Whether it is a retail job or an office job, the vast majority of companies partake in this practice for many reasons but the main reason is to gain insights into the productivity and contribution of their employees so that they can reduce inefficiency where it exists and become more efficient. Basically to improve on their bottom line and please shareholders. While the benefits are potentially game changing, there are ethical questions to consider.

My stance on this issue is kind of neutral. I have absolutely no problem with the data collection of employees as long as it is used for the appropriate reasons and doesn't cross the line where data about an employee's personal life is collected. The data collected should be purely related to the role of an employee within an organisation. As long as businesses are only using the analysis of said data to improve on business decision making and to support employees so that they can perform better in their roles, it is fine by me.

An example of where I think a company is going too far is Humanyze. Humanyze provide their clients with biometric badges that look like normal employee badges but with a catch. These special badges are equipped with radio frequency identification (RFID) and near field communication (NFC) sensors, bluetooth for proximity sensing, infrared to detect face to face interaction, an accelerometer and two microphones. That is a lot of monitoring technologies in one device if you ask me. The function of the Humanyze is to collect data in and analyse it in real time to provide useful information to management. To achieve this the badge is basically tracking everything an employee does, where they go in the office, who they were interacting with etc. This is crossing the line in terms of how much a company is collecting about their employees. It's fine that they are only collecting information while the employees are on campus but the fact that they know exactly where an employee is and who they are talking to at any time is a bit creepy in my opinion.

I'd like to argue that constantly monitoring the workforce may actually have the opposite effect than was intended. When people feel under-pressure (in this case it's because they know everything they do is being monitored) they make more mistakes or do things that they normally wouldn't do. This added stress of being watched undoubtedly contributes negatively to a person's mental wellbeing. I have personal experience of this in my current job working as a computer salesman in an electronics shop. Although I am not a software engineer, the same logic is the same.

This also applies to the majority of my colleagues not just myself. Management have 11 KPI targets that is tracked for each colleague and we have to hit these targets every week and our bonus is dependent on this. Because of the pressure that is constantly on us to hit these targets, some colleagues may be too pushy in trying to get customers to buy the add-ons or in some cases I've seen some of my co-workers tell unknowledgeable customers false information so they would buy the add-ons that would ensure the targets are hit. I've also seen managers gaming the system so that they hit their targets as well. So what I am saying is that by constantly monitoring employees, management might be indirectly encouraging unethical and undesirable behaviours from their subordinates.

Another problem with the monitoring of employees is that it could lead to discriminatory hiring/promotion decisions. With the amount of data that managers have about each employee they will undoubtedly find out information about them that they normally wouldn't have known. For example, a woman is being highly considered for a promotion. However, the manager finds out that she is pregnant through the data that they have captured and decides to change his mind about her promotion because she would have to take maternity leave. This is an example of how the data collected can and will probably be misused.

This is why I like the idea of third parties offering data collection and analysis services as they act as a neutral arbitrator, ensuring that the data is kept safe and used ethically. This separation between employee and employer is important and prevents the employers from knowing too much about their workforce. So if employee data collection is to be a big part of organisations in the future (which it will be), I think this approach of a third party being a arbitrator is the best way to go about it.

Sources

[1] *Searching Under The Streetlight For Useful Software Analytics IEEE Software (July 2013)* by Philip M. Johnson

[2] *Collecting, integrating and analyzing software metrics and personal software process data (2003)* by A. Sillitti, A. Janes, G. Succi, and T. Vernazza

[3] *Software Intelligence: The Future of Mining Software Engineering Data (2010)* by Ahmed E. Hassan, Tao Xie

[4] *http://whatis.techtarget.com/definition/software-engineering*