Life as a Software Engineer

A software engineer will be responsible for actions taken in the software development stage. They follow principles of the profession and apply these to their work. Whether it be the design, development, testing or the evaluating stages. We can all agree that the world of software engineering is changing, adapting and improving. Within in this analysis I will detail about certain aspects of the developer and their work, how it can be measured and give a view of the ethical responsibilities involved. Nobody likes being watched or judged, especially when they are considered a professional in that field. The question we try to look at, is it right? Should a software engineer be unwillingly measured against themselves and other professionals and why does it matter? Will machine learning improve the software engineering industry, or will it increase the unemployment rate for software engineers?

Why Measure Developers Behaviour or Work[2]

There are many reasons why such information is important. It can all be traced back to the client and their needs. The client is the one looking to invest in the software. The material measured has logical and ethical reasoning. The data is more so for management level in a company to monitor project and forecast expectations.

- **Monitor performance:** Provides the ability to check the progress of the project, also performance of the employees with regards to entitlements of reward or disciplinary action.
- **Diagnosing Problems:** It gives the ability notice if a project is slowing down or help identify problems that arise before they become critical.
- **Innovation:** Analysis of the data can propose for changes to existing procedures and introduce new ones. Can constantly look to improve the business, which can in turn reflect on its employees.
- **Cost:** Can estimate a more accurate cost for the client and find where savings can be made, in effort for increasing profit on projects.
- **Planning & Control:** The ability to plan with respect to a time frame for each section. Understanding the resources required and availability of them.

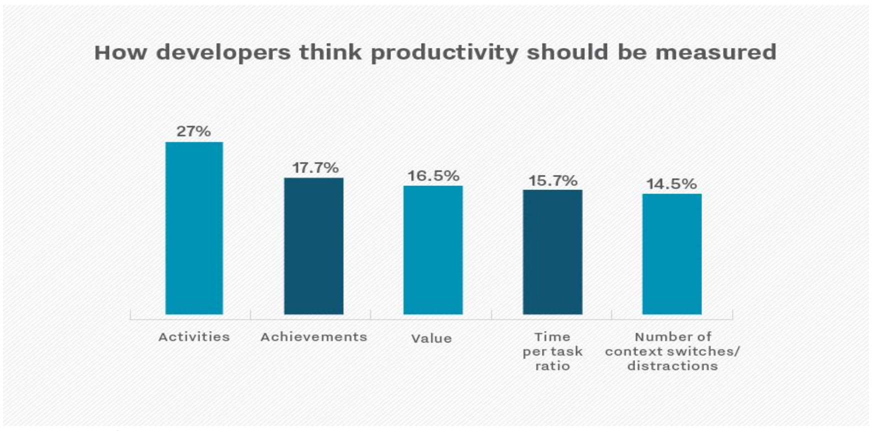


Figure 1.1[9]

Figure 1.1 displays what some developers consider are the more important measurements required.

Measurable Data

From the early days of analysing data, the problem was finding ways to collect enough data and where to look. The problem in recent time is knowing what data to select and what to do with it, there has been a substantial increase in volume of data through the world wide web and enhancements in software development. Many occupations in the world are measured for progress purposes and strategic operations. The management of software developers are responsible for monitoring progress and projects taken on. How does management estimate the length of time that is required to build a various project or how much it is going to cost? This section will focus on some of the measurable data that can be assessed with regards to the progression of software development.

Management by metrics[1]

A list of some measurements that are collected through manual input and through automation of the software. I discuss in more detail the methods later through the report. This section, I will go through the metrics, their value and how management use these analytics.

- Coverage
- Complexity
- Coupling
- Churn
- Size
- Developer Time
- Commits
- Build Time
- Test
- Duplication
- Velocity
- Burndown

Coverage – Indicates the code that is covered when tested. Generally measured as a percent, a developer will look to increase this. The higher this value is, the lower the chances are of having undetected software bugs compared to a program with low code coverage.

Complexity – There are several metrics to measuring software complexity such as: Sneed Metric, Card Metric, Chapin Metric, etc… Individuals that are not within the profession, are often confused to the difference between complicated and complex code. Complicated code is almost like badly written code or uncommented code that is difficult to understand, but can be with time. Levels of complexity regards the entities and interactions with in the code, as the number of entities increase in the developing stage so too will the interactions. It will eventually get to a point where you cannot track the entities and their interactions, and any changes made later in the program can have drastic consequences. This can lead to a form of technical debt, but we will give this its own section.

Coupling – This is a measure of the degree at which different classes are related, different classes should be independent of one and other. The higher the coupling measurement, the harder it will be to change something in one class without effecting another. Developers will look to keep a low coupling value, this will make classes more independent and easier to change something in the class without causing much conflict.

Churn[3] – This measure's the amount of code that the developer has rewritten within a short time, kind of like how this report was written. Each line written, read, deleted, rewritten, etc… Is this good? That depends, would it be wise to have low churn at the expense of leaving bad code in? It is used by software managers to assess the development process. They use it as an indicator when something might be off with developing stages. If there is a spike in churning and no successful code produced the manager can act on this as it may cause future delays or alter some expectations. Some issues that may arise could be: The developer working on a piece could be under-engaged in the project or the product design team could have the developer wasting time and effort going in circles with ideas.

Size – This is generally the size of the program, but depending on the language, application domain or structural requirements used the measure for size can vary. For example, size could measure the number of lines in a program, but what if the churning level is high and the size keeps changing while the programmer is trying to fix something. Say by deleting 100 and adding 300 lines of code or vice versa. Total count does not consider the new work added here and disregards the aspect of incremental software development. It is more optimal to have a tool that can support varying size measures.

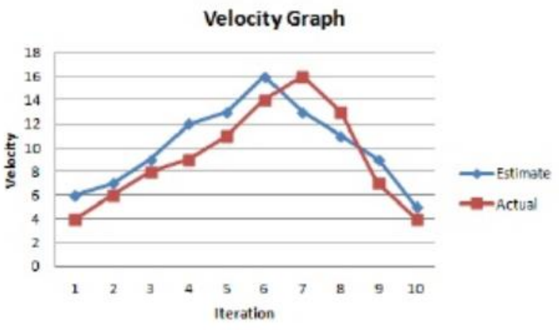
Developer Time – Monitoring the length of time a developer spends on their IDE working on the assigned project. See if it is progressive, they are starting early or leaving it all to the last minute

Commits – Measures the amount of commits they are committing to their repository and the quantity attached.

Build – Monitors how often the developer builds successful code to the system

Test – Lets the manager know if the developer is testing their written code, how often they are testing and if their tests have been successful

Duplication – Checks the work for duplicated code, often code is duplicated because it is easier to manipulate code that has already been tested. However, in the long run can have negative effects by increasing the size of the software or if there is bug testing done on one then it must be done on all duplications which can easily be missed.



Velocity graph – Velocity is used in agile teams, the expected time will be estimated prior to taking on the tasks. Once the developers publish completed work, it is compared against the estimated to give general guidelines to compared against expected completion. It will count the amount of work completed and entered at varying stages.

Burndown Chart – This is a graphical representation of the work involved in a project and the stages in time. The percent of the work on the 'Y-axis' against the time that this work is expected to take, on the 'X-axis'. Generally the work left to do is on the side of the graph as seen in Fig 2, to the right, separated by the colours. From this graph you can see the volume of work each stage will be.

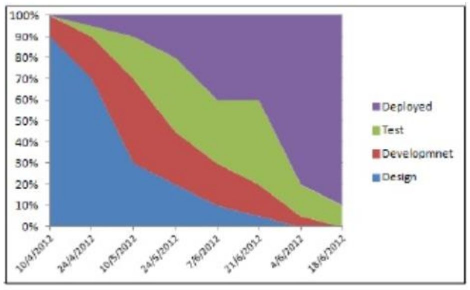


Fig. 2. Cumulative Flow Diagram/Graph

These are not the only measurements available, but are considerably important when looking at the efficiency and productivity levels of the software developers and the work produced. Monitoring expectations and on-the-job factors are just as important as looking back at how it was managed because it can all lead into improvement for future projects. There are other measurements that are almost impossible to account for such as how interruptions effect a project. Considering the amount of interruptions a developer will experience through out the stage of their build can have a huge effect on productivity level.

Interruptions

This is one factor that can be considered tough to measure. If a developer is fully concentrated on designing tasks given to them and they are suddenly interrupted, it can take some time to return to the point they were last at. It is not just about returning to the line they last completed, they must factor in surrounding methods and classes and the idea. Not just for developers, but the worst time to interrupt anyone is when they have their maximum memory load. Ideal times for an interruption is when a developer reaches a milestone or a good breakpoint in the developing stage. It reduces the difficulty for them to turn back on when getting back into it. These are unplanned interuptions, planned interuptions can be just as bad. Knowing that you have something organized later that day, like a meeting or some presentation, can become a complete distraction. A report by Gabriela Motroc called **"And it's gone —The true cost of interruptions"**[4] details the cost an interruption can have on productivity by researching the topic with over 400 programmers. He discusses that having the thought of doing something later in the back of your head can affect the current state of mind, almost like being put under pressure knowing you must get it all done in a smaller timeframe than you would have had given no planned interruption.

Who looks after measuring?[1]

Having all the data is good, but surely it is not collected magically. The collection of data comes at a cost itself, depending on how it is conceived can make the difference. Like most data collection methods or calculations, it was done by hand before we found some way of automating it. An article by Philip Johnson speaks about a topic known as the streetlight effect. It was named after a joke, which has similar reasoing:

I'M LOOKING FOR MY QUARTER I DROPPED!
DID YOU DROP IT HERE?
NO, I DROPPED IT TWO BLOCKS DOWN THE STREET!
THEN WHY ARE YOU LOOKING FOR IT HERE?
BECAUSE THE LIGHT IS BETTER HERE!
?

As humurous as it is, I consider it relative to some of the examples given in his article. He found many were happy to use the data that was easier to obtain and less troublesome to use at the expense of its usefulness and meaning. The data collected was more limited to helping with improvement. When collecting data, it is important to collect data that can be relative to the next project if you are looking for ways to improve and give better estimations to clients. I will go through some processes for manual input data and some automated data collection services that are currently available. From reading the article[1], it gives an indication that some developers are more concerned with the development of the automated processes due to the monitoring of developers behaviour while working on projects.

Personal Software Processes(PSP)

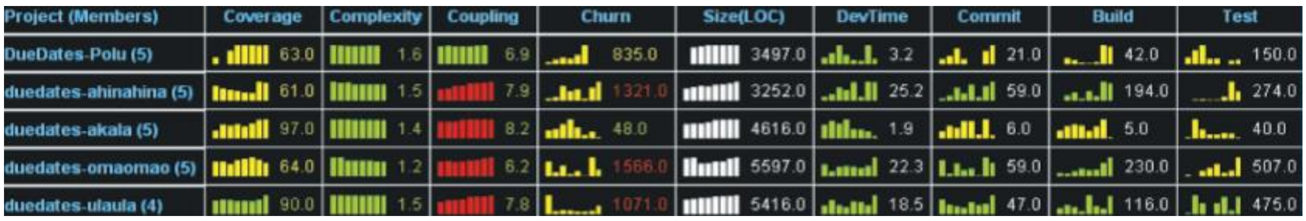
PSP has been developed by the Software Engineering Institute(SEI) to look at the possible improvements of individual software engineers. A software engineer, named Watts Humphrey, wrote a book that detailed some aspects of the PSP. It mainly focused on a manual input process. In the book were many spreadsheet templates that had a range of inputs that needed to be recorded through out a developers work. It would be looking for log times and estimations made by the developer, along with some errors found. The process is flexible on the input and requires personal judgement for the developer, which can help to better suit their needs(such as inputing the interruptions we spoke about earlier, logging these would help understand where can it be cut down). Like any manual input, there is human error or incorrect judgements made which can hinder analytics and data quality. Much of it was self reflective, which would be easier for that individual developer to learn best from their own experience. It is through these errors there has been search for improvements,. Research for solving this problem came the Leap toolkit.

Leap Toolkit

The Leap toolkit looked at some of the quality problems found in PSP and addressed them by automating and normalizing the data analysis process. This new technique still requires some manual input of data from the developer but the machine will produce some analysis that the PSP would not provide. It will perform various forms of regression analysis on the data. Between the manual input and automated calculations of the software, here are some of the records that are available from its use: "the size of the work product; the time it takes to develop it; the defects that the user or others find in it; the patterns that they discover during the development of it; checklists that they use during or design as a result of the project; estimates for time or size that they generate during the project; and the goals, questions, and measures that the user uses to motivate the data recording." The list of records that the Leap could provide were provided at a software engineering conference in county Limerick, Ireland during the month June of year 2000.[5]

Hackystat

Users of hackystat attach this software to their development tools, kind of like sensors, from there it will unobtrusively collect raw data about current development and send this to the Hackystat server. This open source framework can analyze and make interpretations of the collections, it can create visualizations to better understand the raw data that is extracted. The idea of Hackystat is to try and prevent manual inptut, to prevent distractions from the development. Allow the developer to remain working while the data is automatically collected. There is also collectings of interplay among developers, it will collect data of individual developers that work together on the same file.



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

Here is an example of the kind of data that is visually represented by Hackystat.

Sonar[6]

Sonar is an open source platform used to manage the quality of source code. It is designed to manage code quality with minimal effort to the user. It can be used for once-off audits or continuous observation, and management also use it for improving strategy on code quality. Some of the monitoring development aspects covered by Sonar are:

- Duplicated code
- Coding standards
- Unit tests
- Complex code
- Potential bugs
- Comments

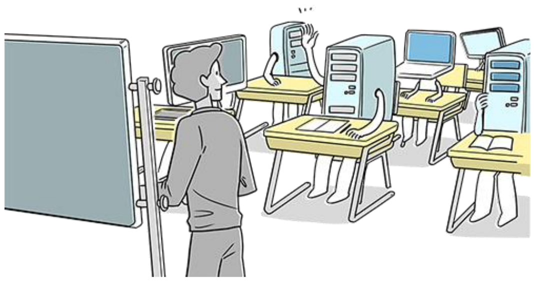
The main language that Sonar covers is Java, but with the use of some plugins you can adopt others like Flex, PHP, PL/SQL and Cobol. It has multiple products depending on the service required. It can help companies manage technical debt issues and can display error descriptions to help developers detect them.

Here are a few more companies that supply software that help collect and analyze data during development stage, they also allow for visualization to represent the data in a user friendly way:

- Semmle
- Teamscale
- CAST
- Code Climate
- Codebeat
- Black Duck

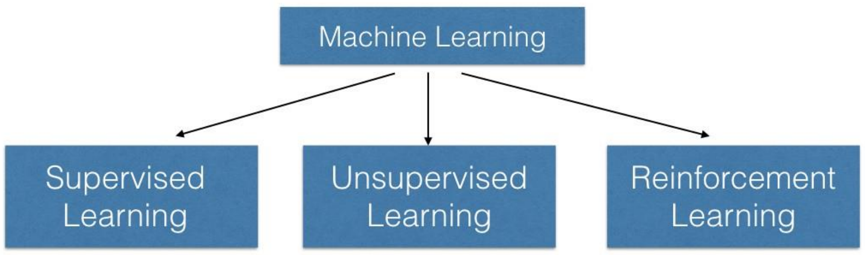
There are many companies out there that are managing the analytics and automating features to improve productivity levels in software development. Data collected entirely through automation certainly saves time, but there can be restrictions due to privacy and overhead concerns. This is why manual input is still important, depending on the requirements. Advances in machine learning can help decrease the manual input.

Wait! The computer did that, is that machine learning?



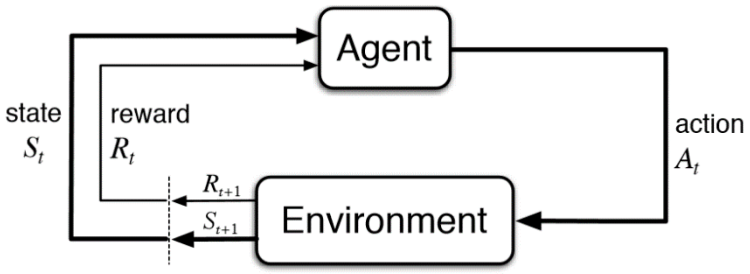
There are a lot of costs involved in todays developing of software, a big proportion of this is involved in the testing and maintenance of the software. This is down to the imperfections due to human error. It is these imperfections that make us human, it is just not ideal for the software development. This could be the lack of quality during the design stages and the coding of the software. It can be much more beneficial if many of these faults are found early on in the development process. By using many of the measurement techniques and the analytics available, it opens a new era for software development. Through the use of the historical input and efforts made, a machine can be programmed to read off these and predict results. Finding adequate quality of software development to measure from will be difficult. This new development is known as machine learning. The machine will be given learning algorithms of fault prone and non fault prone methods to develop predictive models of quality. There are mainly three types of machine learning algorithms but there is many machine learning algorithms under these categories, and as software engineering advances more will be produced.

Types of Machine Learning Algorithms



1. **Supervised Machine Learning**[7]
   This would be the more popular in machine learning. For a supervised algorithm, there will be an input of variables and an expected output such as ‘y = f(x)’. The idea is not to only suit the current ‘x’ inputs but to allow for varying ‘x’ inputs in the future that will conclude a respective response ‘y’ output, where ‘y’ is a function of ‘x’. It is called supervised because we know the answer expected. While the algorithm is making its predictions on the input data it is being monitored and corrected where needs be. There is a constant search to minimize the sum of square errors to allow for better predictions.

2. **Unsupervised Machine Learning**[7]
Given the supervised version, I'm sure you know what to expect here but we'll go through it anyway. For unsupervised machine learning there will be a set of input variables but no corresponding output variable expected. This models the underlying structure in the data so that it can be more understood. There is no constant monitoring and no attempts to adjust depending. From this algorithm, the output we are looking for is designed to give us groupings of the data. They can be grouped by clustering and associations. Clustering will group the variables with similarity and disimilarity behaviours. The association will try to find relationships between these clusters, as some similarities may be stronger than others meaning they will be more than likely grouped together but this does not take away the fact that there are some in each group that will have conflicting disimilarities too. Example: people like product 'x' and those that like product 'z', two clustered groups, but there are some that like both 'x' and 'y' but prefer 'x' more.
3. **Reinforcement Machine Learning**[8]
Reinforcement learning makes use of the dynamic programming techniques. It focuses on learning a sequence of actions that lead to long-term reward. With that saying, their will come a time when the machine is put in a position where it is given choices to make, like option 'a' or 'b', if it actions on 'a' and returns a negative response(known as "punishment") the machine will have remembered the steps it took to get here and return to the point when it had option 'a' or 'b'. From there the machine will take option 'b' until it gets to a positive reward and will learn from these actions. It may not always be just the two options, there could be n options(n meaning unknown amount but greater than 1). Basically the machine will make its own decisions and collect the information as each step is taken until it comes to long term positive reward which is goal orientated.



Machine Learning Algorithms

Supervised Learning	Unsupervised Learning	Reinforcement Learning
• K-Nearest Neighbours	• K - Means	• Markov Decision Process
• Linear Regression	• Sens Clusters	
• Logisitic Regression	• Aproiri	

- K-Nearest Neighbours(Supervised Learning) – Is a non-parametric machine learning algorithm, it makes no assumptions about the functionality of the problem being solved. It will be given a new point that we are looking to solve a prediction for, so we need to classify the point using the given data. Will need to give a number of k, it will select k points nearest to it and the dominating group will be its new classification. For example: with two variables and multiple observations for each, and looking for K=3. We will choose which ever variable dominates with the choice of the 3 nearest observations to the new point.

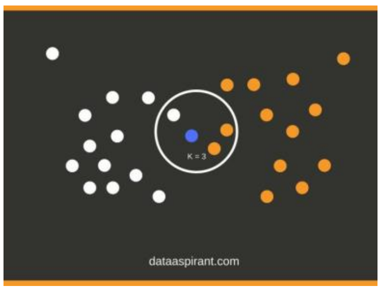
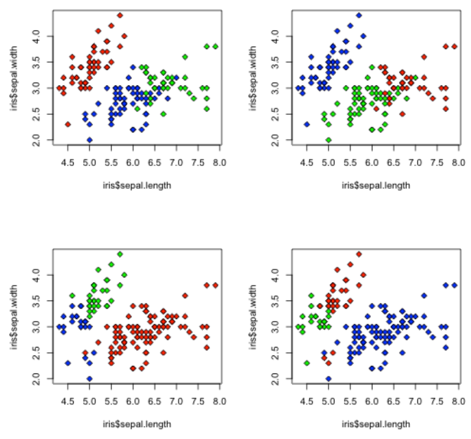
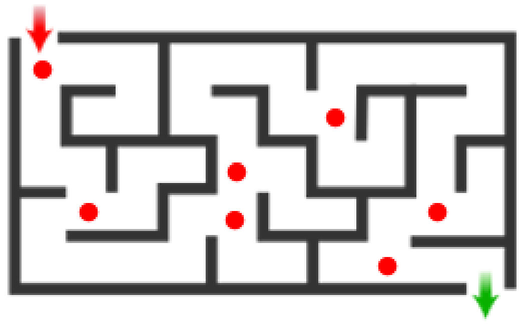


Figure 3.1[10]

- K-Means(Unsupervised Learning) – Will be given an input of K with points in a random position known as a centroid. Centroids will be placed apart from each other as to expect where such clusters will gather. Then it will go through the dataset and which ones have the smallest distance to each centroid and assign those data points to that centroid. Once the dataset has been assigned to a centroid, it will take an average of all the classified points and place the new centroid in this position. This step is repeated until no points in the dataset are changing classification and this is the final position of the centroid. This is the end of the algorithm running.
  Taken from the famous 'Iris' dataset, here is an example of the observed points changing classification(The colours indicate each centroid classification):



- Markov Decision Process(Reinforcement Learning) – This was mentioned in the above under reinforcement learning. It is basically and algorithm that will be placed in a certain condition or state, from being in the state it will be given certain actions it can take with consequences for each. It will make its decision and define the new state it is in. This algorithm in machine learning will do this over and over with a goal objective. For each deadend or bad action taken it will learn from this and alternate until it reaches the goal.(getting through a maze is a perfect example)
  Each red point is a state at which the algorithm can be put in. It must make a decision and learn from it to get out of the maze successfully which is the goal in this objective.



It is not that machine learning is getting easier, just that is becoming more easily accessible. These algorithms are the computational algorithms that we as humans use in our own analysis and have done so without the use of computers. Nowadays we use computers to calculate the end results making it easier and less time consuming on the analyst. Its from good use of these algorithms, can the machine learn to use them more effectively, by using good historical experiences. This is why the software engineer is still so important during this development. Only the developers can determine if the code being analyzed by the machine is worthy. Machines aren't taking over, they're opening new doors and making it easier for developers to increase productivity and better their research.

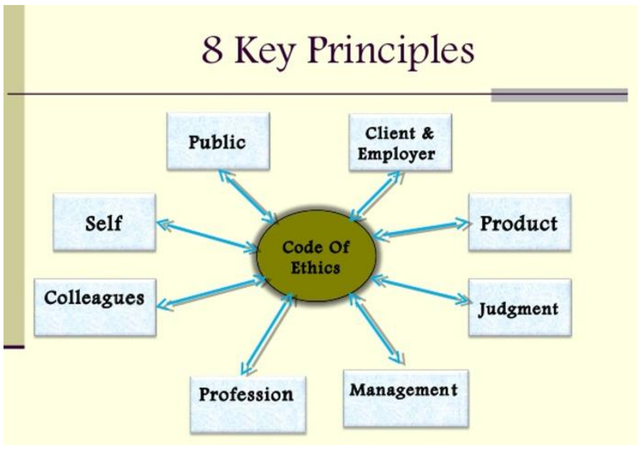
Are the efforts ethical?

What is does ethics involve in the computing world?

Ethics are moral principles that an individual must apply when using their skills. They should use their professionalism to make decisions with regards to professional and social conduct. There is a code of ethics that software engineers abide by, they are approved by the ACM and the IEEE-CS as a competent standard for learning and current software engineer professionals.
The recommended ethics can be found at the following website:

https://www.computer.org/web/education/code-of-ethics

Much of the ethics for a software engineer can be considered universal to many traits. Many of the main principles is understanding what is good practice and what is not. By not abiding by these code of ethics, can tarnish a professional career. Which is only right in the sense that your lack of consideration for the rules put in place can have a knock-on effect for something worse. It is not only to be ethical in software engineering, but everything you do.

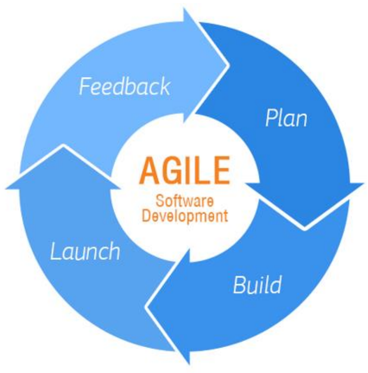


Are the measurements of developer behaviour and work ethical?

Like most professions and disciplines, there is a code of ethics that must be adhered to. The work carried out must be within a social and legal framework which limits the freedom of people in that area, such as software engineers. A developer must be aware of these ethics and understand that they have more responsibilities than using their skills in software development. There must be honesty and integrity in ones work, and should not use your skills and abilities to behave in a dishonest manner. Respect should be portrayed towards the skill of software engineering as a profession. Some that are not bound by law are considered responsibilities to uphold. Such as respecting the confidentiality of your employer or the clients, you should remain within the limits of your skills and capabilities and respect intellectual property rights by being more aware to any patents or copyrighted work that may appear. Accidents can happen, but it is the responsibility of the individual to prevent them to the best of their ability.

It is not only the developer that must follow the code of ethics, but also management and client alike. For years there has been research done to try and improve software development, not only in sense of code writing but as reliability and professional approaches to development stages. Like any profession, process or procedure, if there is room for improvement it is ideal to do so. More so than not, the enhancements are through experience and what is experience without historical data. This is where conflict arises between the developers and management. Software developers are concerned about the monitoring of their behaviour on work and their work efforts. Some feel it goes against moral status and intrudes on their rights. This would be the case if their every move was being monitored and not just their work and effort put in, and from this it would be unethical and illegal. The monitoring, as we have seen, is used to help management make decisive manoeuvres on current and future projects. Some platforms require manually calculated input data, and others are automated. The manual input is more flexible to the developers' judgement, whereas automated platforms read directly from the computer and monitor their stages at each moment in time and report statistics with respect to the data analysed.

In the article, by Philip Johnson, he explains how many developers do not like the platforms used to measure their behaviour during software development. There still lies some concern from developers being monitored for their work, their work effort and the approaches they take. Perhaps, they feel that it is intrusive and unnecessary. More than not, each project that is being designed is not about the developer, it is not about how the project is designed regarding which step comes first and how they got to the final testing stages. At the end of the day, it is what the client has asked for and is looking for and this is the main concern managers have. Monitoring the behaviour can also pick up on aspects the developer struggle with or if there is a consistent disruption amongst many developers, this could lead to a structural change in process. It is all about the 'bigger picture'(improving the software development stages).



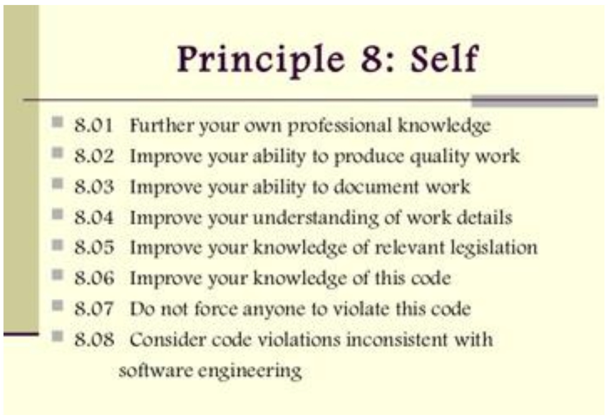
Some approaches taken by software developers are plan-driven or agile processing. Plan-driven is about setting out a plan prior to development and during development measure progress against the plan. Whereas, agile software development is about planning incrementally throughout the development. This method is more flexible and adaptable to changes. Often it is preferred to find a mix of both. Whichever method is chosen, management need a way of analysing the progress. The capability of measuring the project, the developer and any other anomalies will help management develop more and better strategic planning during the process. They can allocate more preferred resources and match the developer with more suited projects. They can promote changes in the organizational structure of the project early on and help prevent any disasters in the future of the project.

With the agile software development approach, it is possible to present current work to the client to make sure it matches their desires or allows them to change early. It is not only legal to monitor developer's, but it is also ethical too. While monitoring employees, it is important not to discriminate and this is where it can become tricky because there are so many individual ways to develop software. Each developer may have a different approach to a specific problem or an alternative method to writing a piece of code. How can you monitor such anomalies? The way to manage this, is by setting rules and procedures to approaching projects and problems. One rule that can be applied to all is:

"Build early and often"

This is an effective procedure set for developers. They ask that their developers start the project early and work on it often. This can promote quality and reduce risk factors. The lack of appropriate and effective measurement practices can also increase the risk of a project.

Is this a matter of opinion?



Principle 8: Self

- 8.01 Further your own professional knowledge
- 8.02 Improve your ability to produce quality work
- 8.03 Improve your ability to document work
- 8.04 Improve your understanding of work details
- 8.05 Improve your knowledge of relevant legislation
- 8.06 Improve your knowledge of this code
- 8.07 Do not force anyone to violate this code
- 8.08 Consider code violations inconsistent with software engineering

Are you aware of the principles that are included within the code of ethics for software engineers or was the last page just me, giving my bias opinion on the matter…
Kind of both, but that is because I strongly believe in the code of ethics and that it is there for a reason. Principle 8 highlights the main aspects behind the purposes of measuring the analytics that can be found on each software development. As a software engineer, you are encouraged to improve in the profession. From a management perspective, they too must improve but for them to improve they will need you to improve. Is that right? The collection of your work and your behaviours can help them improve, this information can help you as a software developer improve too. The effects of analysing development stages can bring some undervalued staff members to the light. It can help recognition for the work done, but at the same time with some it can do the opposite. It can cause some to be anxious or stressed because they know they are constantly being monitored. This should not be the case, it should be used as learning tool to help improve. Unfortunately, control over peoples' thoughts and views is not possible and therefore some negatives can arise. It may lead to misconduct with developers trying to cheat the system to get it working in their favour. Such as the example about the churning mentioned earlier, where one may ignore bad code just to reduce their churning rate.

<u>Conclusion</u>

Its not a matter of being right or wrong, but about taking the ethical approach. We have seen that it is an ethical responsibility for developers to improve their abilities and help advance the industry of computer science. Furthermore, it is the responsibility of the software managers how this information is obtained and processed. There are methods available that give the developer insight to work as they have developed, but then there are the unobtrusive methods which can raise some concern of the developer being watched. Better communication, is perhaps the better approach, rather than leaving it until some review for reward or punishment. It seems to require more of a phycological approach, dealing with these kinds of matters. Not that the developers have mental issues, but from the aspect of how to approach individuals to tell them they are doing a good job or need to pick it up a notch without having a negative impact on their productivity levels (they are professionals too and are in higher demand when half way through development stage).

This can lead to the worry of machine learning, does this advance affect their employment in the future or does it give the opportunity to upskill in the industry. Rather than fearing the inevitable, there should be a positive approach taken for machine learning. It may make some areas of software engineering redundant, but this only gives them a better reason to upskill. Machine learning is not perfect, and nor will it be for some time. The computer science industry needs good software developers to improve its capability. Therefore, measuring is a necessity in the advances of software development. Each software developer needs to be at the standard that is expected as a professional. Their responsibilities should go beyond the code of ethics, as this is just a base line. Like in many professions, through the advances of technology, their work has been made easier. Such as JCB tractors saves time on shovelling, computers to spot defects saves time on a mechanic searching for the issue, etc… These are all advances in a trade/profession that have not made the professional redundant but has saved time and resources.

References

8 - Analytics Vidhya. (2017). *Essentials of Machine Learning Algorithms (with Python and R Codes)*. [online] Available at: https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/ [Accessed 22 Nov. 2017].

7 - Brownlee, J. (2017). *Supervised and Unsupervised Machine Learning Algorithms - Machine Learning Mastery*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/ [Accessed 20 Nov. 2017].

1 - Csdl.ics.hawaii.edu. (2017). *Cite a Website - Cite This For Me*. [online] Available at: http://csdl.ics.hawaii.edu/techreports/2012/12-11/12-11.pdf [Accessed 20 Nov. 2017].

Figure 3.1 - Dataaspirant. (2017). *Knn Classifier, Introduction to K-Nearest Neighbor Algorithm*. [online] Available at: http://dataaspirant.com/2016/12/23/k-nearest-neighbor-classifier-intro/ [Accessed 23 Nov. 2017].

2 - Harris, D. (1994). *Organizational linkages*. Washington, D.C.: National Academy Press, p.107, Chapter 6. Available at: https://www.nap.edu/read/2135/chapter/6#107 [Accessed 23 Nov. 2017].

5 - Ieeexplore.ieee.org. (2017). *Lessons learned from teaching reflective software engineering using the Leap toolkit - IEEE Conference Publication*. [online] Available at: http://ieeexplore.ieee.org/document/870464/ [Accessed 26 Nov. 2017].

4 - JAXenter. (2017). *And it's gone —The true cost of interruptions - JAXenter*. [online] Available at: https://jaxenter.com/aaaand-gone-true-cost-interruptions-128741.html [Accessed 14 Dec. 2017].

3 - Metrics for Software Engineering Management | GitPrime. (2017). *5 Developer Metrics Every Software Manager Should Care About | GitPrime Blog*. [online] Available at: https://blog.gitprime.com/5-developer-metrics-every-software-manager-should-care-about/ [Accessed 29 Nov. 2017].

Figure 1.1 - Searchsoftwarequality.techtarget.com. (2017). *The secrets of software developer job satisfaction - Measuring software developer productivity one task at a time*. [online] Available at: http://searchsoftwarequality.techtarget.com/photostory/4500271450/Measuring-software-developer-productivity-one-task-at-a-time/2/The-secrets-of-software-developer-job-satisfaction [Accessed 26 Nov. 2017].

6 - Sonarsource.com. (2017). *Continuous Code Quality | SonarSource*. [online] Available at: https://www.sonarsource.com/ [Accessed 20 Nov. 2017].