***Measuring and Assessing the Software Engineering Process***

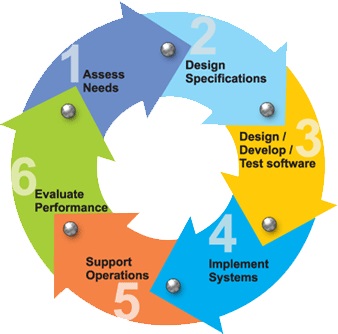
By James Cox 18/11/17

***Table of Contents***

|  |  |
| --- | --- |
| *Introduction / Abstract* | *1* |
| *Measurable Data* | *2, 3, 4* |
| *Computational Platforms* | *4, 5, 6* |
| *Algorithmic Approaches* | *6, 7* |
| *Ethical Concerns* | *8, 9* |

***Abstract***

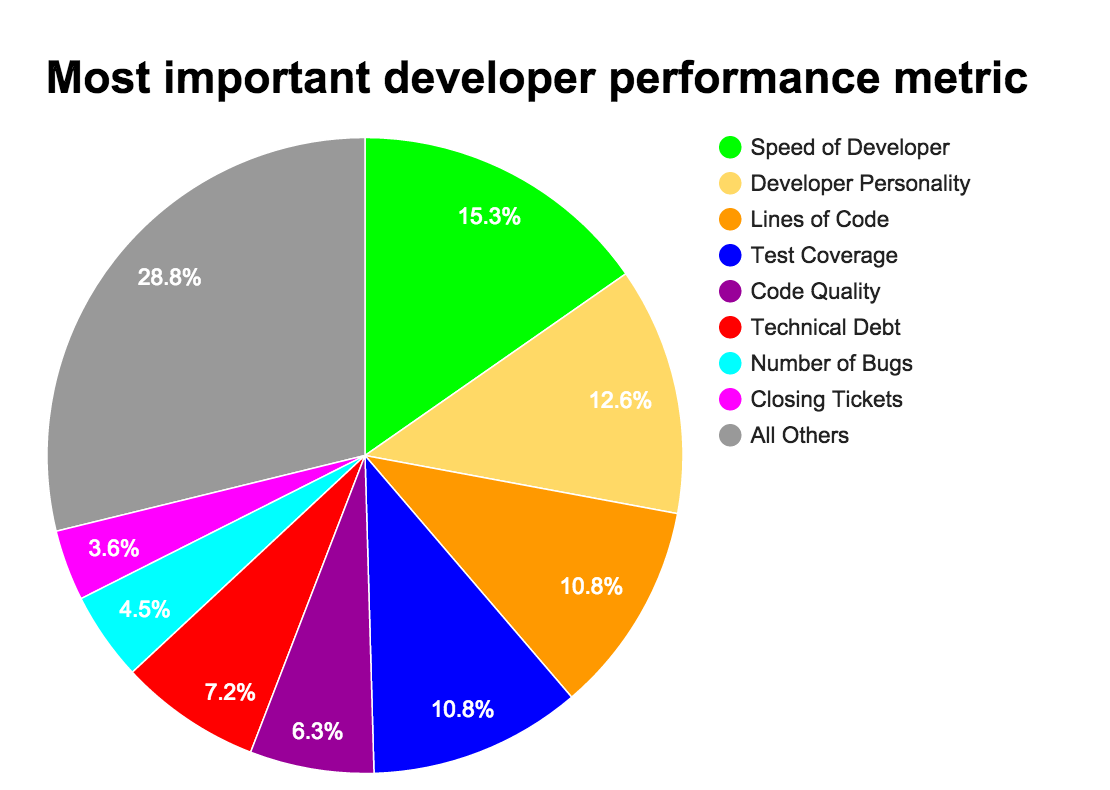
Unlike other disciplines, there is no defined procedure for a software engineer to complete a task. However, Sommerville (2017) believes that the software engineering process is essentially a structured set of activities required to develop a software system. This process involves each step in developing software, from initial customer inception up until the release of the finished product. Although many different process models exist and vary in numerous ways, each one contains the same aspects. These include specification, design and implementation, validation and evolution.



This diagram is a suitable display of the software engineering process and its respective steps. The circular shape explains of how the process has many repeated steps up until the finished product is established.

***Measurable Data***

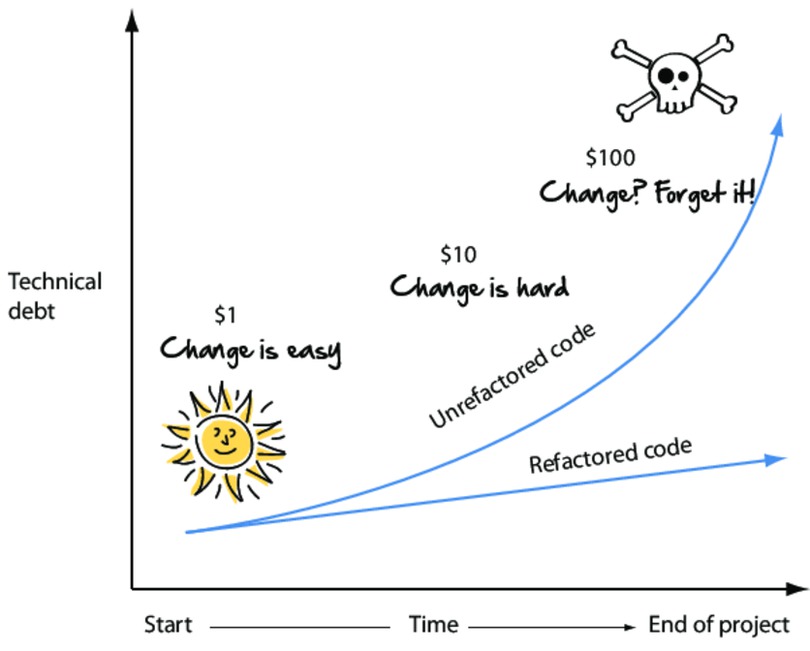
There are two kinds of measures in engineering: product metrics and process metrics. The former describes product's qualities such as dimensions and physical data. Process metrics describes process' qualities such as production time and effort required. In software, the product metrics includes code length, complexity, reusability and maintainability. Whether engineers agree or not, measuring data is a necessary practice in the software engineering process. It gives them something to work with when they are estimating the time it may take to complete a job. Thus, using past experiences as benchmarks for completing new jobs acts as a benchmark for future jobs.

*Speed of Developer*

Essentially, time is money. As a result, measuring the speed of development is another way to measure productivity. The faster that a team delivers, the more productive they seem. A fast development process also allows for the release of software more quickly and efficiently. But speed will only be the most important metric for certain business models. For others, following the specifications would be of greater importance. Overall, speed is a valuable measure of data in the software process but it does also have its flaws. One of them would be that having a development team measured on the speed of the job consistently, will lead to burnout. All in all, it depends on the type of software you write for how beneficial speed of development is as a metric to you.

*Technical Debt*

Technical Debt is a deliberate decision to implement a flawed solution or a lower standard of code in order to release software faster. It is impractical to have zero technical debt as it will lead to the slow delivery of new features. As a means of measuring the software engineering process, technical debt is becoming increasingly popular.  Software will rarely be perfect so treating technical debt as an investment in the future of the software life cycle is becoming a normality.  Reasonable levels of technical debt are now being considered good practice as it allows new features and products to be released without having to revisit old bugs regularly.



*[The Agile Samurai by Jonathan Rasmusson](https://www.safaribooksonline.com/library/view/the-agile-samurai/9781680500066/)*

It is important to note that technical debt is not poor code. Technical debt can often be well written code coming from a good software engineer under unrealistic time constraints. Therefore, an understanding relationship between management and the software engineers is crucial to keep the technical debt at a sustainable level, which ensures that the software engineering process remains as efficient as possible.

*Source Lines of Code and Commits*

When trying to look at Source Lines of Code (SLOC) or number of commits as a positive metric one might try to think of how non-technical people would see it as an easy metric to game. Length is a measure that people understand so either of the metrics mentioned can be a good way of showing someone the work of a project.  These are not the best way to measure the software engineering process.  Many experienced engineers and scientists remark on the ineffectiveness of this methodology and most in the software engineering world do not like this method. These metrics do a very poor job in measuring programming progress and developer’s productivity. A programmer who is being measured by SLOC or number of commits will be rewarded for generating quantity rather than quality.

*Communications*

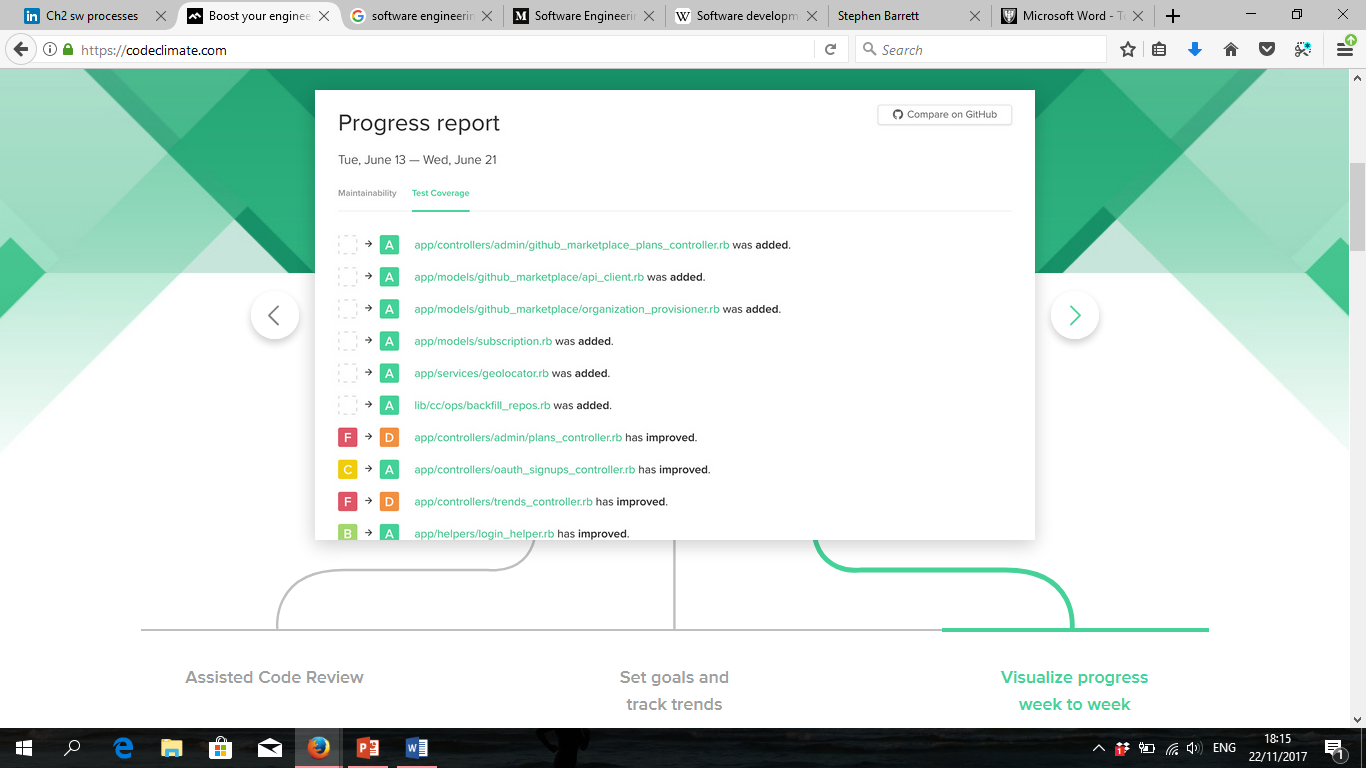
Dissimilarly to the metrics mentioned above, communications isn’t necessarily a quantitative measurement. This metric is non-code based but is regularly considered just as valuable as some of the other metrics.  Software engineers are continuously collaborating across instant messaging, emails, reading designs, reviewing code and going to meetings. Sufficient communication skills are beginning to become more and more relevant with many projects failing from simple errors or lapses in communication.  Experienced software engineers such as Flood (2015) have even gone as far as saying that “all things being equal, the better you are at communication the further your software engineering career will advance and it will do so a faster pace.” It is difficult to measure how good someone is at communicating compared to how many lines of code they write per day, but the better the communications between software engineers and their clients, the better the end solution. As a result, a high level of communications is an extremely useful piece of data to analyse when measuring an engineer's performance.

***Overview of Computational Platforms Available***

When measuring and assessing the software engineering process, the platform used can be of huge importance. Computational platforms have various ways of displaying results and measurements and as a result, one must make an educated decision on which one to use. Analytics as a service refers to the practice of using web-based technologies to carry out analysis of big data, opposed to the traditional method of developing an onsite hardware warehouse to deal with the data.  The new method allows clients to use a particular analytics software for as long as needed, and can be more cost effective and less labour intensive when compared to the traditional way of building the necessary infrastructure in-house.  Outsourcing the bulk of this work to reduce cost is becoming an increasingly popular option but hybrid options, like using tailored hardware combined with web-based infrastructure, are allowing companies to create even more suitable solutions for analysing their big data needs.

*Code Climate*

One of the main computational platforms available for measuring this process is Code Climate, which is essentially an open source, extensible platform for ensuring code health and analysing software (thoughtbot, 2017). Code Climate allows the user to take full control of their code quality by offering configurable test coverage and maintainability data throughout a software engineering project. Such opportunities ensure that quality improvement is explicit, ubiquitous and continuous. The platform is used by over 100,000 projects and manages 2 billion lines of code daily (About Us, 2017).



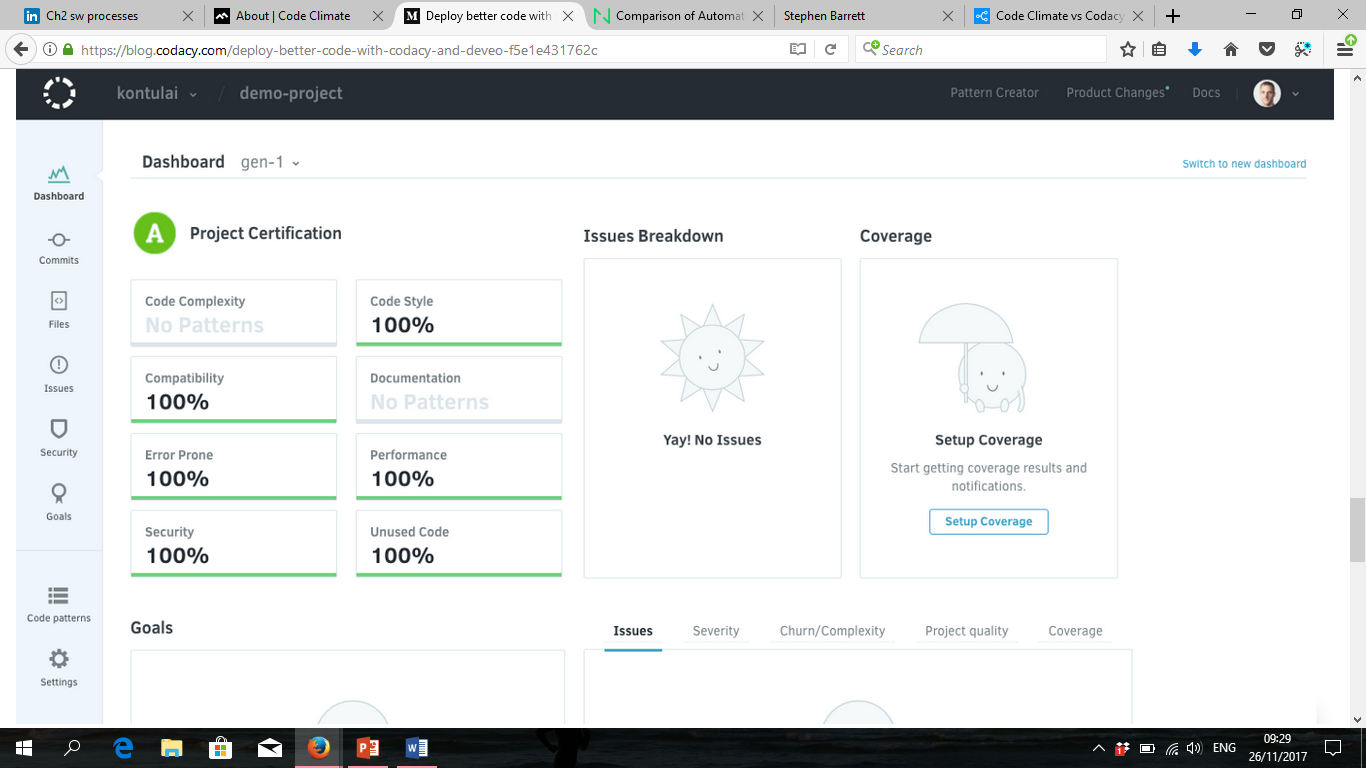
The adjacent diagram depicts a vivid example of Code Climate analysing a project. It displays each change and whether or not there has been an improvement in code quality.

The platform works by breaking the analysis up into two segments, test coverage and maintainability. Test coverage specifications can be customised in order to prioritise different issues. Along with efficient GitHub integration, the test coverage displays line-by-line analysis on the software. The maintainability is a mix of duplication, complexity and structure. The trends page allows the user to look at the overall scale of things. With graphs, charts and summaries, analysis has been simplified to its optimum level.

The benefits of using this static analysis tool include the fact that it is open source, extensible and can be ran anywhere. In addition to this, it has a great UI and a simple grade system that motivates the user to keep code clean. By making this platform open source, Code Climate allows its users to understand exactly how their code is being measured and assessed, which ensures that the analysis is credible and users therefore trust the feedback that they are given. In addition, the extensibility means that users can write their own static analysis engines to run on the Code Climate servers so that an even greater range of languages can be used. This platform offers users the chance to efficiently measure and assess the software engineering process by assessing their needs, improving and developing code and tests, and evaluating performance through highly detailed analysis. However, when compared with other analytic platforms such as Codacy, Scrutinizer and Codebeat, Code Climate appears to be the most expensive tool. Also, some believe that other platforms place greater effort into extending the tool based on customer demands (Łukasz Ozimek, 2017).

*Codacy*

An alternative to using the Code Climate platform is Codacy. Codacy is another easy-to-use code quality review solution. The platform gives users an automatic feedback solution to code and utilizes static analysis to track project quality and notify when new issues are found. Codacy is flexible and constantly adapts to project updates. The platform pushes results as comments in the users pull requests or can also send them as notifications with supporting channels (Slack or Hipchat). Like Code Climate, Codacy works efficiently with continuous integration tools and serves as an ideal complement to code unit tests.



This is an example of a user’s dashboard and project summary based on the code that has been submitted to Codacy.

Once a user selects the files to prioritise, Codacy offers advanced statistics such as churn, complexity and duplication to assist in the optimization of the code. What differentiates this analytical service from the rest of the market is the ability that it gives the user to customize their code analysis. Code standards can be applied to projects and required code analysis patterns can be selected. The graphs and trends available make analysing code much easier and the security and performance checks ensure that no stone is left unturned in regards to customer confidence in the platform.

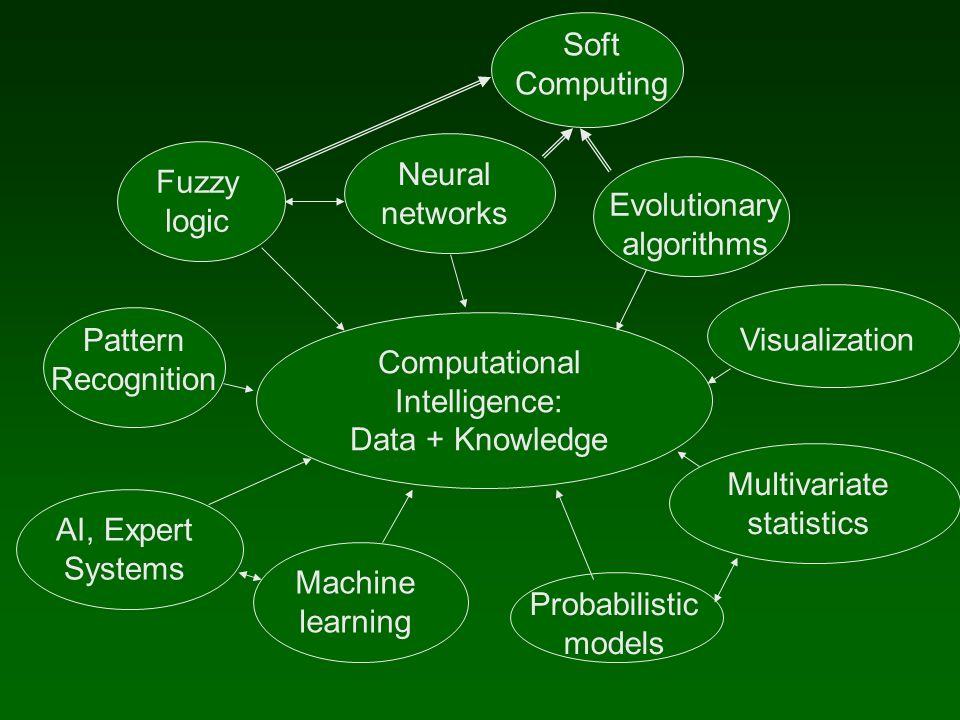
The benefits of using Codacy far exceed the negative issues associated with this platform. Many complement its easy setup and customization. In addition, the duplication detector, reduced technical debt and the fact that it is free for open source allow Codacy to remain as one of the best software analysis tools around. Where Codacy excels over Code Climate is its ability to connect with far more integration tools. While both platforms are in association with GitHub, Slack and Hipchat, only Codacy integrates with BitBucket, GitLab and Jenkins. This provides Codacy with greater opportunities and the ability to attract a larger market audience. Companies currently using Codacy include Firecracker, Telepath and Abibao.

Codacy’s ability to support operations and also to assist in implementing systems and optimizing code ensure that it is a prime platform for assessing and measuring the software engineering process. At this moment in time it is difficult to choose a more successful computational platform as both Code Climate and Codacy have unique and interesting features respectively. In a survey recently completed by stackshare.io, Code Climate claimed 269 votes, while Codacy earned 147 votes based on users’ favourite platforms (Code Climate vs Codacy vs PullReview, 2017). Although people may have contradicting opinions on these platforms, both Code Climate and Codacy improve code and with it, the software engineering process.

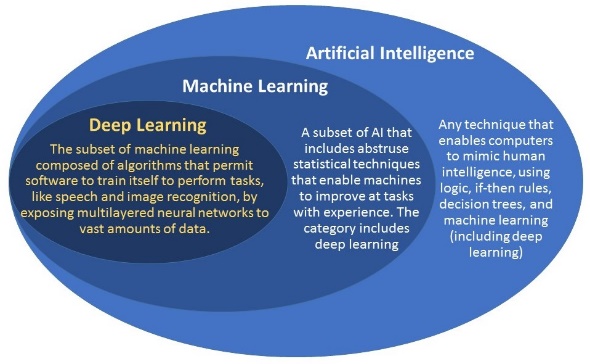
***Algorithmic Approaches Available***

A great deal of software engineering is centred on daily decisions and questions. Unfortunately, many decisions related to a software system are based on instinct and gut feeling.  These hastily made decision processes lead to wasted resources and increased costs when developing and maintaining large complex software systems. Software Intelligence (SI) intends to deal with this common mistake.  Much like Business Intelligence uses statistical models to make accurate forecasts and improve decision making in the business landscape, software intelligence is starting to be implemented into measuring and assessing of the software engineering process.  It offers software developers continuous and accurate information to support their daily decision-making processes.

*Computational**Intelligence*

Computational Intelligence (CI) is the study of the design of intelligent agents. An agent is something that acts in an environment. In this case CI refers to the ability of a computer to learn a specific task from data.  The goal of this is to understand the principles that make intelligent behaviour possible. The systems that are designed using CI tend to be flexible with changing environments and changing goals, learning from past experiences and adapting dynamically to solve particular problems. The system is somewhat of a black box; taking in prior knowledge, past experiences, goals and observations and leading to the creation of an end product to solve a problem.

***Machine Learning Algorithms***

Machine learning is essentially the process of getting computers to act without being explicitly programmed. In the past decade, machine learning has given us many revolutionary products.  Machine learning is so pervasive that we use it dozens of times a day without realising it.

There are two primary machine learning algorithms being used by companies today.

*Supervised Learning*

Supervised Learning is the most common type of machine learning process.  The term refers to the fact that we give the algorithm a data set in which the output variables are given.   The goal is to approximate a mapping function so precisely that when you have new input data you can predict the output variables for that data with a supervised learning algorithm.  This can become incredibly useful when trends begin to arise and the process gives an engineer the advantage with data from the past.

*Unsupervised Learning*

Unsupervised learning is where you only have input data and no corresponding output variables.  The goal for this is to model the underlying structure or distribution in the data in order to learn more about said data.  The process is called unsupervised learning because in contrast with supervised learning, there is no correct answer and there is no teacher. Algorithms are left to their own devises to discover and structure the data.  Unsupervised learning algorithms use the technique of clustering to make use of the input data presented to them.

*Algorithm Decision Making*

Algorithms have come to replace humans in making decisions that affect many aspects of the software engineering process.  Algorithms that most wouldn’t think of affect certain software all the time.  We rarely realize how much blind trust is placed into algorithmic decision-making. One improvement may be that algorithms don’t think when they are approving credit cards, counting votes, or determining financial aid.   Therefore, assuming that we provide data to the machines with high integrity they should be able to make a fair decision from a tailored supervised algorithm for the situation.

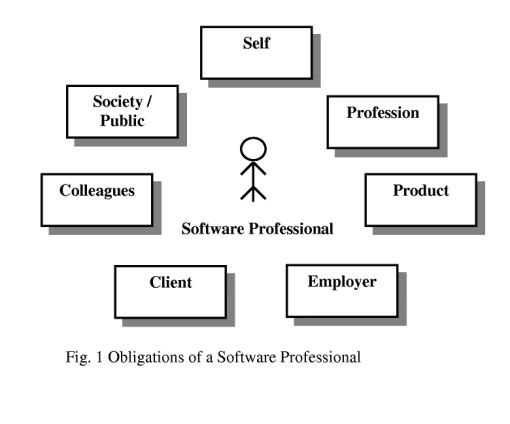
***Ethical Concerns***

*Openness and Availability*

In this day and age, the amount of information available to software engineers is completely underestimated. Software projects tend to store huge amounts of real data and it is not always sufficiently protected. We have a worrying amount of information available at the click of a mouse and with that comes ethical issues and regulations. The most concerning part of software engineering is what we, or others, could do with the data generated from a project.  We place so much trust in people we don’t even know and as a result, sensitive data could easily be placed in the wrong hands.  Chacko (2016) talks about how “the role of a software engineer is not limited to designing, developing and deploying software projects. He/she should also possess sound knowledge and perseverance in all such ethical concerns without losing the scope and quality of products that are delivered”. No software engineering projects should cause any hindrance to privacy of individuals or organizations, yet it frequently happens. This is a major ethical concern that most of the software engineering firms experience. Software or cloud systems can be used as the key tool in accessing confidential information on clients which forces engineers to take caution of what is stored and readily available. Another concern for companies would be social implications and cyber crimes.  The involvement of the public in all major social software projects is appreciable. They are also concerned about the cyber crimes that can occur after the deployment of their software.

*Professional Ethics*

Professionals plays a significant role in today’s society. Software engineers are hired by clients based on the above statement and it goes without saying that a client expects their project team to act in a professional manner always. Whether this is on a personal level or when physically working on a project, it is vital that the process remains free of any major ethical concerns.



“The Obligations of a Software Professional” (meherchilakalapudi, 2009). Displays the basic ethical responsibilities of a software engineer.

*Transparency of Data Measurement*

The transparency of how we acquire and use the data is key when we talk about the ethics concerns surrounding this kind of analytics.  Transparency facilitates accountability, verifying that services perform as they should, and that data is used according to contract.  Fortunately, new methods to increase the transparency of data measurement are being introduced more and more.  These approaches include personal information management and program verification.

*Codes of Ethics*

Because of the fundamental impact computing has on our lives, it is necessary to integrate computing technology and human values in such a way that the technology protects and advances human values. Codes of Ethics express the consensus of the profession on ethical issues. At the same time they are a means of educating the general public about the ethical norms and values of software engineering projects and the processes involved. This can give members of the public, and hence new clients, confidence in software engineering, while also ensuring that their information remains private through efficient and effective processes.

*Bibliography*

Ahmed E. Hassan, (2017). The Future of Mining Software Engineering Data. Available at: https://pdfs.semanticscholar.org/e23e/287baf50b5b9a19774de6d6fb356b6bac212.pdf [Accessed at: 25/11/2017].

Code Climate: About Us, (2017). Available at: https://codeclimate.com/about [Accessed at: 21/11/2017].

Code Climate vs. Codacy vs. PullReview, (2017). Available at: https://stackshare.io/stackups/codacy-vs-code-climate-vs-pullreview [Accessed at 20/11/2017].

Comparison of Automated Code Review Tools: Codebeat, Codacy, Codeclimate and Scrutinizer, (2017). Available at: https://www.netguru.co/blog/comparison-automated-code-review-tools-codebeat-codacy-codeclimate-scrutinizer [Accessed at 21/11/2017].

Dubakov, (2014). Available at: https://www.targetprocess.com/articles/speed-in-software-development/ [Accessed at: 25/11/2017].

Professional and Ethical Issues of Software Engineering Curricula, (2017). Available at: https://pdfs.semanticscholar.org/5ad1/c1acba4e43446d751dee3d72fee667809ade.pdf [Accessed at: 27/11/2017].

Thoughtbot.com, (2017). Available at: https://thoughtbot.com/work/code\_climate [Accessed at: 20/11/2017].