**1. Introduction**

Measurement is an important and necessary process in all projects and phases. No matter what the process model is, all stakeholders involved in project development need some indicators to have better understanding and familiarity of the project. For example, customers need measurement to judge correctness and outcome in order to compare them with original requirements and expectations. Developers need measurement to test results and efficiency of their code, functions and modules. Managers need measurement to evaluate quality of development process and products. From these examples, there are multiple reasons, situations, phases and different types of personnel involved in the use of measurement. Therefore, a fixed measurement method cannot be the solution for all measurement requirements. In other words, software metrics selection is the key in software measurement. According to the measurement needs: statistical meaning, hypothese testability, decision making, progress tracking and scales are the requirements of measurement(1). When these requirements are extended to multiple software systems and different phases in software development, software metrics selections become more important to make measurement universal.

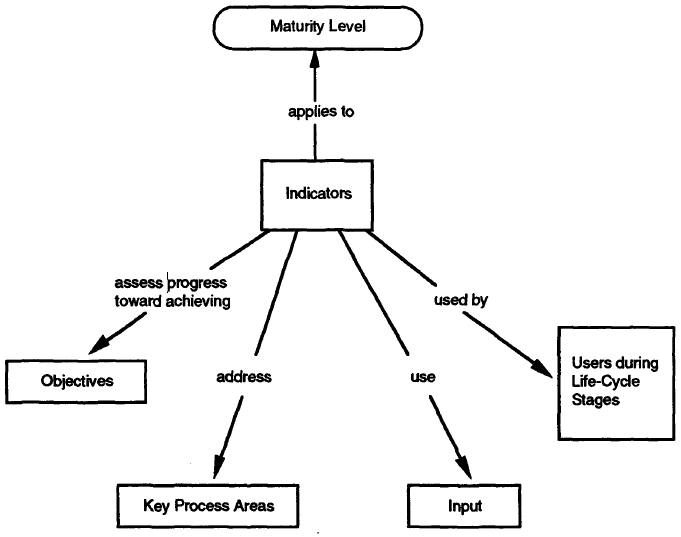
**2. Background**

The capability maturity model is a common model that guides the software project, helps individuals and organizations control and predict the performance of the software system and development. To link measurement with capability maturity model, the structure of the capability maturity model needs to be decomposed.

**2.1 Measurement and capability maturity model**

According to the report of organizations, the benefits of implementing measurement programs included but not limited to: product development insight, quantify tradeoff decisions capability, better projects planning, managing and tracking, better process and environment understanding, potential process and efforts area improvement identification, better communication, etc(2).

To successfully achieve these beneficial goals, the understanding and analysis of the capability maturity model is an important prerequisite. The capability maturity model contains 5 levels: initial, repeatable, defined, managed and optimized(1). For each level transaction, there is a certain process describing phase change: initial to repeatable is disciplined process, repeatable to defined is standard, consistent process, defined to managed is predictable process and managed to optimizing is continuously improving process(2). From this definition, the selection of software metrics must satisfy not only five levels of software process maturity, but also phase change processes.

****

**Figure 1. Relationship of Indicators to CMM Maturity Level and Indicator Discussion Topics(2)**

Figure 1 is a relationship graph of indicators, also known as metrics. It shows for each maturity level, the indicators must be able to connect four fields inside software systems. Furthermore, the indicators must be able to be applied to the corresponding maturity level, while it can also show the expected information from the measurement program.

**3. Software metrics selection in different maturity levels**

Although there are so many software metrics available, the differences between maturity levels determines that correct metric or metrics can optimize measurement outcomes. While preventing repeat measurements which may cause resource waste such as money and time.

**3.1 Ideas and recommendations for different maturity levels**

*Level 1 initial*: ***training***. The training is a software metric that provides visibility about the effectiveness of training programs for individuals and in organizations(2). The purpose of the training program is to evaluate and develop all personnels involved in softwares and systems to meet the skill requirements. In the initial stage of capability maturity model, everything is not defined yet and there are a lot of aspects that need to be finalized such as management, tools, plan, design, etc. The outcomes of the training metric can be used to measure the overall quality and level of the team. For example, if the test skills are proven to be the best performance among indicators, then the development process should prioritize test-driven development. Therefore, training is the recommended software metric in level 1 which can help stakeholders to make final say about the software project or system.

*Level 2 repeatable*: ***effort*** and ***cost***. The repeatable phase is a project conducting software development and management follows a set of defined processes(1). In this stage, the processes are defined so activities and procedures in softwares or systems are fixed, which makes the measurement repeatable as well. The measurement of the minimum unit or cycle can be measured and then multiple the times or quantities in general level. In this case, effort and cost are the two recommended software metrics that fit the characteristics of this level. The effort provides visibility about staff contribution on time costs, schedule correctness and product quality(2). The outcome of the effort metric on a single process, project or system can help evaluate the current staffing cost, predict the future and estimate the whole. It is also a reasonable metric that can help individuals and organizations to make decisions related to time and human resources. Similar to effort, the cost metric provides visibility of tracking actual costs compared to estimated costs, while predicting future costs(2). The focus is changed from staffing effort to resources cost, with similar or the same strategies.

*Level 3 defined*: ***progress***. In the defined phase, projects follow customized templates and processes that have already been designed and decided(1). In this case, the progress is a good software metric that can evaluate the project commitments since those processes are complete. The progress metric provides information about how well the product is performing compared to its expected commitments(2). The outcomes of progress help stakeholders check if the outcomes fits their purpose, as well as help developers and managers check if different processes execute correctly and output logically results. Therefore, progress is a recommended software metric for level 3 to judge the completeness, correctness and timeliness with respect to the software or system commitments.

*Level 4 managed*: ***quality*** and ***stability***. The managed phase has process managed and measured(1), which means the software or system has finished development processes. Therefore, as a viable product, the quality and stability are two software metrics that need prioritized consideration. Unlike other unary software metrics, quality and stability has multiple secondary metrics. The quality contains quality assurance audit results, review results, peer review results, trouble reports and defects, while the stability contains requirements stability, size stability and process stability(2). Although each of them can be treated as a single software metric, combining them together into two major classes makes the result more concise and general, which covers many aspects of the whole software system. The outcome of the quality metric helps the team locate and monitor defects or problems during the measuring, testing and using. Allow them take actions based on the measurement of quality performance. The results of the stability metric guides the team control and monitor the running performance of the software system, as well as help them make decisions on requirements, size or process changes.

*Level 5 optimized*: ***computer resource utilization***. The optimized phase’s main goal is to prevent defects and use innovative new process methods(1). Therefore, the computer resource utilization is a recommended software metric. The computer resource utilization provides information about the software system’s performance on the optimization goal and requirements(2). The outcome of the computer resource utilization metric is exactly the complement of level 5. It tells the stakeholders how well a software system is using its computer resources, where the problematic parts are and help individuals or organizations make optimization decisions. Furthermore, the computer resource utilization metric can constantly provide prediction on resource utilization results after changes are made, as long as the software system is still working. Therefore, the computer resource utilization is a proper metric for measuring the software or system in the optimized stage.

**4. Software metrics selection in different systems**

Similar to the study of measurement and capability maturity model. To link measurement with different systems, the characteristics of different systems, platforms and frameworks are important variables, while simulating and evaluating different metrics selections is a meaningful process.

**4.1 Software metrics used in software systems comparison**

Software systems are always written in some programming languages. Therefore, to choose a proper metric to compare different software systems, look at some indicators in the programming languages is a good start. However, there are countless programming languages and they have created many different software systems. So finding the common grounds in different programming languages and evaluating them if they can be used as metrics is an important step. According to the research, many experts and professionals have implemented this process.

***The number of statements(NOS)***: In programming languages, the statement is a unit that forms a certain behavior execution. It is a very basic component in programming languages. According to the research of comparison of Java and Python:”Finally, the metric NOS revealed that statements are differently used in the two languages, and also that the average number of statement declarations in Java is much larger, even if Java classes feature fewer lines of code on average”(3). From this conclusion, it is not hard to conclude that for the same statement declaration, the size of Java code is higher than Python. In other words, it may result in higher memory cost in the software development process. Therefore, the number of statements is a proper metric that helps individuals and organizations determine which software system to use, estimate the size of the project, predict the hardware and time cost, etc. Even for users, this metric can tell the advantages and disadvantages of two software systems such as wordloard degrees.

***Number of children (NOC)***: For those programming languages have subclasses, those subclasses are a hierarchy of a class and called children. It is a widely used definition method in different software systems so it is a good indicator to show facts in the software systems. According to the research of C# code and its maintainability Assessment:”...for NUnit, clusters 1, 2, 3 and 4 (88% of total population) contain classes with fewer children than those of classes in cluster 5, which means these classes are easier to maintain. Similarly...Classes with high NOC values can potentially become maintenance bottlenecks”(4).

From this conclusion, the number of children metric is able to output data which is the number of subclasses in the system, let stakeholders evaluate the maintainability and find maintenance bottlenecks based on the numerical data.

Another analysis about the number of children shows:”...shows the impact and code reuse in terms of subclasses. Because change may impact all children, the more children have a class, the more changes require testing. Therefore NOC is a good indicator to evaluate testability but also the impact of a class in its hierarchy”(5). Therefore, the number of children metric can show how subclasses affect a class and be used to evaluate testability and maintainability.

**4.2 Cloudbased and non-cloud based application/framework**

With the rapid development of cloud technologies, individuals and organizations can choose to deploy their applications cloud-based, while legacy software systems are also able to be migrated into cloud computing service. However, the characteristic of the software systems determines the degree of adaptation to the cloud computing service. Therefore, measurement is required to evaluate the suitability of non-cloud based application and cloud based application. According to the research of evaluation on cloud based applications, there are three important metrics in the cloud service models.

***Resource Provisioning***: The duration of virtual resources provision when connected to the application(6). For non-cloud based applications, normally the resources are locally or through their own network, the duration of provision is controlled and reliable. However, for cloud based applications, this means the effectiveness and responsibility of the application. Therefore, resource provisioning metric evaluates the connection performance of the cloud based application.

***System Availability***: The availability of the application over cloud service(6). After deploying applications to cloud based service, the system availability not only depends on the application performance, but also relies on the availability of cloud service platforms. Therefore, system availability metric is not only a measurement for application itself, but also a measurement for overall environment.

***Issue Tracking***: The time needed for tracking the issue and providing a solution(6). When application is cloud based, the issue may come from the migration or integration to the cloud services instead of application. Assume the issue tracking mechanism is ideal in application, the cloud service may not have the same processes. Therefore, issue tracking is a necessary metric for individuals and organizations to judge fault tolerance and problem solving performance in cloud based platforms, as well as make cloud migration decisions.

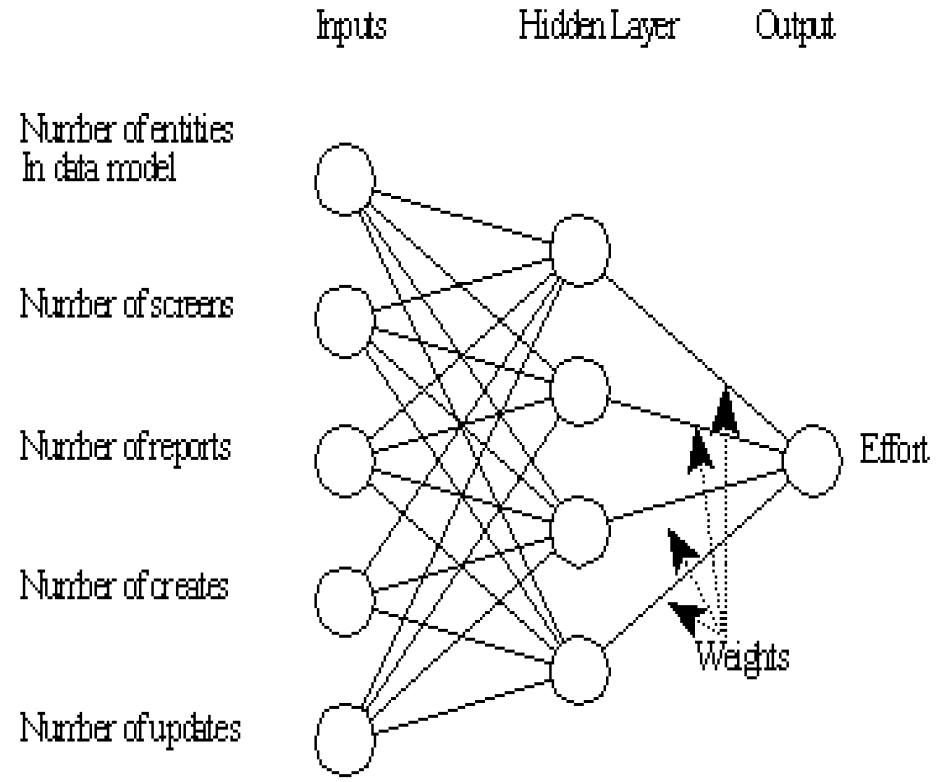
In conclusion, these three software metrics evaluate different aspects of the software performance over cloud based service and provide tradeoff guides for cloud base related decisions.

**5. Modern technologies that may aid software metrics selection**

In the past few years, artificial intelligence, machine learning and data mining became popular topics. The related research, tools and applications were improved and developed dramatically. With the help of these technologies, the software metric selection process may become easier and more accurate.

**5.1 Artificial intelligence**

In predictive models of software metrics, the neural networks model has already been proposed:”Can be used to cluster systems in similar groups. This can help when modelling very different systems, and separate models need to be developed using other techniques”(7).

****

**Figure 2. An example neural network structure for effort prediction(7)**

The other techniques may refer to artificial intelligence technology because the sample neutral network structure is very similar to the artificial neural network(ANN). By placing the neural networks into artificial intelligence tools, the new model may produce more precise metric selections and better metric predictions.

**5.2 Machine learning**

Based on a research about empirical assessment of machine learning:” We have compared different machine learning models for...We have shown that there is no particular learning technique that performs the best for all the data sets...We also showed that “size” and “complexity” metrics are not sufficient attributes for accurate prediction”(8). Since the result is based on software defect prediction technique and the software system is real-time. It is very similar to the metric selection process: the complexity, unpredictability and sophisticated requirements of the projects and different phases increase the difficulty of choosing proper metrics. Therefore, machine learning is still a vague concept in aiding software metric selection.

**5.3 Data mining**

According to the summary of the research, data mining in software metrics databases:”In this paper, we have for the first time combined fuzzy clustering and data mining for the analysis of software metrics data....Our work provides a new perspective on these three datasets, and points the way toward the use of data mining technology in the context of software process control. For organizations at the higher levels of the Capability Maturity Model, there is a need to analyze software quality data and apply this information to process improvement activities...the more powerful techniques of machine learning and data mining are viable, useful tools for software quality analysis”(9). The data mining technology reminds individuals and organizations to reconsider their position in capability maturity model as well their metrics data. Furthermore, this research confirms that machine learning and data mining are powerful tools, they are viable and can be useful in analyzing softwares and systems.

**6. Conclusion**

Source lines of code metric were widely used before, after more and more people realized the defects of it such as lack of accountability, lack of cohesion with functionality, adverse impact on estimation, difference in languages, etc(10). New metrics are proposed and implemented based on different needs, models and phases, as well as different metrics are used in different systems and platform comparisons. However, just like source lines of code, all metrics have their suitable situation and problematic scenes. Choosing the right metric is the key to the measurement, as well as help making decisions and predictions. So software metric selection should be a process of scientific analysis. On the other hand, some research shows modern technologies such as artificial intelligence and data mining are also powerful tools that can help software metric selection. Therefore, the software metric selection methods are still changing and improving.

**7. References**

1. Peter Grillo, *636 Wk 3 Lecture V2*, Mar 04, 2021.
2. Baumert. John, and McWhinney. Mark, "*Software Measures and the Capability Maturity Model,*" Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, Technical Report CMU/SEI-92-TR-025 , 1992.
3. G. Destefanis, M. Ortu, S. Porru, S. Swift and M. Marchesi, "*A Statistical Comparison of Java and Python Software Metric Properties,*" 2016 IEEE/ACM 7th International Workshop on Emerging Trends in Software Metrics (WETSoM), Austin, TX, USA, 2016, pp. 22-28, doi: 10.1109/WETSoM.2016.012.
4. Shumail Arshad and Christos Tjortjis. 2016. *Clustering Software Metric Values Extracted from C# Code for Maintainability Assessment*. In Proceedings of the 9th Hellenic Conference on Artificial Intelligence (SETN '16). Association for Computing Machinery, New York, NY, USA, Article 24, 1–4.
5. Balmas, Françoise & Bergel, Alexandre & Denier, Simon & Ducasse, Stéphane & Laval, Jannik & Mordal-Manet, Karine & Abdeen, Hani & Bellingard, Fabrice. (2010). *Software metric for Java and C++ practices (Workpackage 1.1)*.
6. Sheetal, A. & Kongara, Ravindranath. (2018). *Software metric evaluation on cloud based applications*. International Journal of Engineering and Technology(UAE). 7. 13-18. 10.14419/ijet.v7i1.5.9071.
7. Andrew R. Gray, Stephen G. MacDonell, *A comparison of techniques for developing predictive models of software metrics*, Information and Software Technology, Volume 39, Issue 6, 1997, Pages 425-437, ISSN 0950-5849.
8. V. U. B. Challagulla, F. B. Bastani, I-Ling Yen and R. A. Paul, "Empirical assessment of machine learning based software defect prediction techniques," 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, Sedona, AZ, USA, 2005, pp. 263-270, doi: 10.1109/WORDS.2005.32.
9. Dick, Scott & Meeks, Aleksandra & Last, Mark & Bunke, Horst & Kandel, Abraham. (2004). *Data mining in software metrics databases*. Fuzzy Sets and Systems. 145. 81-110. 10.1016/j.fss.2003.10.006.
10. Bhatt, Kaushal & Tarey, Vinit & Patel, Pushpraj. (2012). *Analysis Of Source Lines Of Code(SLOC) Metric*. IJETAE. 2.