**Surveying Measurements-Oriented Approaches for Software Engineering**

“[Software testing] is not usually a significant part of the CS curriculum in universities, and it is unclear whether this deficiency is ever addressed successfully in organisations.” *– L. Hatton* [1]

In order to understand the effectiveness of a pricing strategy or to gauge the success of a software development effort, metrics and analyses derived from a subset of data about a developmental effort can be used. Measurements relating to a software project are referred to as metrics and models are a subset of the overall system used to make generalizations and, in at least the case of algorithmic models, to derive statistical conclusions about the project. There exist a number of tools and methods to take a measure of different aspects of a project and this paper intends to provide an overview of the types of metrics available to measure various attributes associated with a development effort, an explanation of why and how they may be beneficial, and several scenarios towards which a measurements-oriented approach has been applied in the field of software engineering.

To give shape to the idea that defines usage of metrics and modelling, it is important to understand the circumstances in which their application may be appropriate, as there is a certain level of overhead involved in deriving calculations for assessing software, no matter the phase of the effort at which the project stands and the environmental attributes affect their relevance. Generally, models and metrics are applicable when the software size, software complexity, or level of risk associated with software failure is significant enough to warrant their estimation. This paper does not offer standards in these areas for evaluating circumstances in relation to their appropriateness for using modelling and metrics but provides the reader with some understanding of the types of data necessary for developing some of the more comprehensive approaches to building an estimate, or at a minimum the level of accuracy to be expected given the constraints associated with named modelling approaches that approximate the project environment being studied. Largely due to the fact that metrics are either skewable, ambiguous, or not unquestionably objective, this paper discusses metrics as a means to building estimates versus a means for definitively proving any approach to development superior over another.

The most common parameter type for assessing constraints of a development effort is size of its code base. Aside from variables that can skew this parameter across different project environments, such as level of abstraction of programming language used and complexity of interaction among components within the system, size is often a suitable predictor of other factors, such as number of bugs predicted or time required to adapt the software to meet a specified need. Size is also largely regarded as enabling various objective measures to be determined. Size is a useful measurement for comparing attributes of a given project across the phases of its lifecycle and across many modelling environments as it relates factors, such as productivity, to the effort to further analyze constraints associated with development.

As Somerville explains that what is desired of software is the ability for it to achieve some functionality or quality that is intangible, the objective behind developing software is indirectly related to concrete measures, like size.

Reliability is a measurement type used in a modelling and developmental contexts that can be paired with size for building different estimation types. A concrete measure such as number of defects in a product or estimated time that might elapse before a failure can be expected to occur may be used to assess reliability. Yet reliability can be shown to be a subjective measurement in estimates, whereas reliability is defined as the probability of failure-free operation for a given time in a given environment [2], and assumptions about the environment which may not apply across environments under comparison may exist. Andersson[2] offers that reliability may be related to a system in that it can be used to describe a set of operations a system is designed to perform paired with their probabilities of occurrence. Thus, any hardware differences among environments should be considered to recognize the effect of their difference upon reliability, as the same software run for the same period of time on different hardware could be subject to variability in the likelihood of producing system faults.

When discussing scenarios to which measurements-based approaches have been utilized, later, reliability growth modelling is explained as an estimating technique used to predict the likelihood or confidence a software product will continue improving over time to achieve a determined level of reliability. It can be measured statically or dynamically, using code analysis techniques, also explained.

Productivity is the final measure we examine as being a metric-type often considered in literature and research. Productivity measurements are often made relative to the size of the effort, which again may be subject to interpretation if there rigid rules are not defined for measuring it, such as by amount of added capability achieved versus lines of code produced, whereas productivity may be defined, in a manufacturing environment, as the number of units produced divided by the number of person-hours needed to produce those units [3]. A manufacturing environment offers tangible conceptualization of productivity as a measurement type, yet software development does not afford such an analogous conceptualization. While a most common derivation of the productivity metric is based upon the division of volume of code lines produced by programmer-hours needed to create them, as Boehme[4] asserts, the objective is to assess the quantity of “intellectual work” put into software development, which corroborates the idea of using function points. Function points or object points may serve as relevant metric types for measuring productivity and are often comparable to what is needed to achieve a 1-2 sentence feature explained in the project requirements documentation.

Dependent upon the stakeholders to which estimates or projections relying on metrics are to be presented, the most appropriate type of metric or model may vary. The amount of overhead and usefulness of either may be subjective due to a number of factors that are not intrinsic to the field of software engineering but that are possibly a by-product of process variances. The undertaking of using a more commonly reviewed modelling technique, such as COCOMO, offers the advantage of having multiple datasets that may be analogous to the development project under consideration and of offering more refined process guidance to assist in formulating a figure which can be more useful during evaluation.

For modelling-based approaches to software development, the most applicable approach to take may be determined by a specialist, if the financial resources available to the project permit it, as modelling approaches are generally subject to the limitations that scaling factors impose on any project. Modelling approaches imply multiple programmers will be involved and that the project may be subject to non-linearly scaling parameter values, as the effort proceeds. Modelling approaches may take a progressive form such that few programmers are needed initially, while more are added during to complete feature development, and, lastly, fewer programmers are needed during the finalization stage, at which point the system is prepared for deployment.

Metrics and models may be helpful in determining a level of confidence that development will be successful according to any or all of the factors defined by the Project Management Triangle[5] - or the triple constraint. Alternatively, Basili, et al. offers that metrics may measure an aspect of a project according to, simply, product, process, or resource attributes [6]. In this way metrics may be useful to various stakeholders including divisional or corporate managers, line managers or functional analysts wishing to make predictions on, improve upon, or retroactively evaluate a product or process of concern. Regardless of the group of recipients for which the estimate is intended, metrics serve as a basis upon which modelling approaches of a predictive, prescriptive, or extracted type can be evaluated for validity as well as level of fitness to a given environment and may be used to serve in gauging effort involved in re-using, integrating, or adapting existing software to meet an agreed upon objective. However, an algorithmic modelling approach is commonly used to compare ways to reduce project costs and assess the risks of different approaches for completing an effort.

To gauge the effort associated with re-using, integrating, and adapting existing software used in meeting a requirement more efficiently, a group of researchers from the Netherlands and Belgium conducted a case study to identify areas for KBC Bank to make gains in productivity and concluded a form of the COCOMO II model-based approach was useful, even after considering several other model-based approaches [7]. Yet the case study, based on the goal of identifying trends across various projects at KBC versus improving efficiency on a single project and based on the researchers’ determination that SLOC was a better measure for their uses than Function Points after considering tooling and other implications, considered such factors as the direction in which data was being moved between components in the system, from the software- down to the hardware-level. This type of goal may seem a good candidate for an input-domain modelling approach; nonetheless the researchers on the KBC case study found COCOMO II provided valuable insight into making productivity gains. In the KBC study, the researchers used COCOMO II as a SLOC-based approach for ease-of-use and immediacy or reporting capability while noting that a FP-Analysis-based approach would have been more mathematically correct, if it were feasible.

Algorithmic modelling approaches can be described as parametric and based on the level of specificity to which the model details the derivation of the metrics from which it is comprised and allows for adjusting them, relative to the environmental circumstances to which it the model will be applied, the model may be considered superior to other in this regard or not. It is commonly regarded that a benefit to using the COCOMO II model as compared to other models is that it allows adjusting various factors, referred to as cost drivers, that pertain to the environment considered. All parametric models are likely to be of either a homogenous or non-homogenous Poisson (HPP or NHPP) type, in that they monitor a rate at which some circumstance or event will be realized, with the exception that the latter factors the rate at which said occurrence will be realized. Many models, in statistical terms, will be reflective of the Pareto phenomenon will imply their characteristics can be visualized in a recognizable graphical representation form, where their characteristics are familiarly referred to as falling in line with the 80/20 principle.

It is a relatively unsubstantiated postulation that research discussing software reliability growth models may not be as likely to discuss subjectivity of the estimates producible if the rate of failure for software assessed maintains a constant degree of complexity. This is based on the assumption that given the same complexity and hence constant number of paths available during the execution of a software program that, over time, the programmers should be able to isolate fault locations and address fault causes without introducing new faults. However, while this scenario may hold in a manufacturing environment, in software environments, this may not be true, since software may be rarely modelled during the stage of development where this type of behavior might be exhibited - the phase-out stage. Additionally, Andersson notes debugging techniques may be imperfect and that programmers may introduce new bugs during attempts to remedy extant bugs, as revealed in a replication study she conducted for the purpose of generalizing the results of an empirical study on selection of an SRGM appropriate for a given environment, to ensure the results of the original study would hold across different test cases. She notes that several SRGMs consider reliability growth to be non-homogenous in that failure intensity is not constant based on continually improving the reliability of the program, but does not describe the means by which reliability models are selected, in detail. Rather, Andersson proceeds to describe SRGMs as not being categorized according to homogeneity but according to statistical distribution characteristics and describes SRGMs as an estimation tool whose results are unlikely to guarantee much precision. She suggests environmental circumstances that address assumptions used in the SRGM-type applied and which address the test environment factors should be used. Four SRGM-types are evaluated in her study conducted across three projects with system and functional tests applied against each phase. The Andersson replication study uses similar techniques as those suggested by the original case study on which hers was based and concludes the suggested techniques provided meaningful indications for estimating failure data, according to the investigator’s narrative (the explanation of statistical logic she provided seems valid and accurate, although running her figures through an analysis tool and/or assessing the likelihood that the subjective measures were made systematically was not performed). Andersson, interestingly, further contributes towards SRGMs in her replication study by introducing a realistic factor such that locating and correcting a fault in zero time without the possibility of introducing further errors can be incorporated into the modelling approach suggested by the case study on which hers is based.

Researchers, field users and management alike should take fondly to the evidence, revealed by the case study on which Andersson’s study was based (by Musa) and by her replicated study, both suggesting that SRGM’s may be practical, useful, and valid as aid in the reliability estimation process. The research suggests that a model that is simple enough for someone without an extensive background in mathematics will be effective and analyzing the appropriateness of the four model-types used in each of the three projects in the study, as based on numbers, may be as simple as comprehending the level of goodness-of-fit, as based on the R-square test.

To restate cases in which metrics are useful, at any phase of the software lifecycle, a software project may be measured according to reliability, validity of software, efficient usage of personnel and associated programmers’ productivity, and ability to deliver based on other actual or estimated figures. This paper discusses the metrics types most discussed in literature on model-based approaches, but metrics may be described as evaluating project status, broadly, to measure: requirements, risks, source code, tests, defects and documents [9]; volatility metrics check the changes of the requirements; traceability evaluates links between requirements to requirements within a document; and requirements completeness metrics checks whether specified requirements are complete or not [9].

While there is no single metric which can be used to accurately measure the validity of a cost estimate due to the fact that measurable attributes of the software each can be measured to different standards and are relative in and of themselves, it is the opinion of the author that metrics can be used to augment project planning efforts. Based on the results presented in this paper alone, attempts to discredit metrics by and large with broad-reaching, unsubstantiable claims such as that “metrics have rarely been shown to impact the actual outcome of a project in practice,” may be rejected; even if by noting that metrics are relative to one another and that where the value of an outcome is most often subjective and is generated dynamically, such as is the case with software, there may always be some subjectivity but not such that numerical estimates are invalid, in whole.

If a project manager is measured according to the estimate he submits, ultimately the success of his estimation may be evaluated according to the level of accuracy to which he forecasts difficultly foreseeable events and protects the company against the risks associated with said events. In the case the project manager has delivered an estimate that is accurate and the project is seen to complete on time and within budget, then it is likely his estimating techniques will be examined to produce future estimates that are as reliable. If the characteristic of reliability reveals itself in repetition, perhaps the estimation technique will be useful as a technique upon which industry and fellow project managers within the company can model their estimating process. While it is commonly regarded that Boehme’s COCOMO model is useful as an estimation tool, Sommerville [3] insightfully suggests that no matter the estimating tool, technique, or methodology applied to forecast constraints of a development effort, accuracy shall depend to some degree on the level of experience the estimator has with the project environment, including personnel. Boehme has suggested that project managers will be fully successful if and only if they make winners of all the other participants in the software process [8].

In this paper, types of metrics and the benefits and considerations that couple them in practical model-based approaches are provided at a high level. Metrics in the context of various model-based approaches used for estimating cost and reliability are provided, along with a personal reflection as to perceived advantages and ambiguities remaining in the area of SRGM-based approaches, as largely determined based on a replication study assessing the SRGM-selection approach by Musa, in which three project environments are used to determine the best-fit SRGM. The reader should have been made familiar with considerations bearing impact on project estimation from the perspective of estimators, developers, and integrators, and should be better able to assess the type of model-based approach that is more applicable to a context, given factors inherent to it; additionally, the reader should have a basis from which to form an opinion as to the usefulness or inapplicability of metrics to a given project environment.

# Appendix

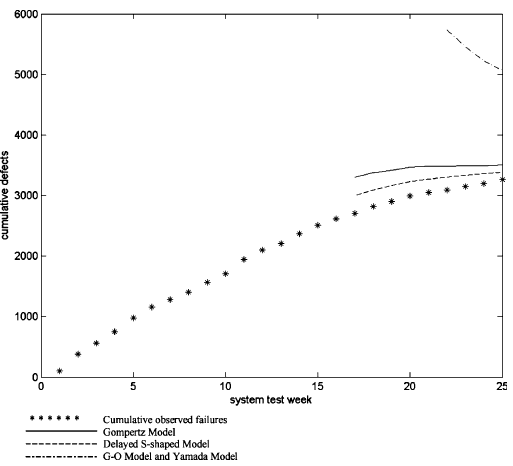


Figure 1 GOF FOR THREE SRGM-TYPES

AS PER THE REPLICATION STUDY

Four common statistical models are used in the original study replicated by Andersson: the basic Musa or Goel-Okumoto (G-O) model (Goel and Okumoto 1979; Musa et al. 1987), the delayed S-shaped model (Yamada et al. 1986), the Gompertz model (Kececioglu 1991), and the Yamada exponential model (Yamada et al. 1986). **Shown above,** fitness of three SRGM-types is depicted relative to the actual defects to graphically reveal reliability growth.

# Bibliography

[1] L. Hatton, “Defects, Scientific Computation and the Scientific Method,” in in *Uncertainty Quantification in Scientific Computing SE  - 8*, vol. 377, A. Dienstfrey and R. Boisvert, Eds. Springer Berlin Heidelberg, 2012, pp. 123–138.

[2] C. Andersson, “A Replicated Empirical Study of a Selection Method for Software Reliability Growth Models,” *Empir. Softw. Engg.*, vol. 12, no. 2, pp. 161–182, Apr. 2007.

[3] I. Sommerville, *Software Engineering 7*. New York : Pearson/Addison-Wesley, 2004, p. 759.

[4] B. W. Boehm, *Software Engineering Economics*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1981.

[5] R. C. Nienaber and A. Barnard, “A Generic Agent Framework to Support the Various Software Project Management Processes,” *Interdiscip. J. Information, Knowledge, Manag.*, vol. 2, pp. 149–162, 2007.

[6] R. van Solingen, V. Basili, G. Caldiera, and H. D. Rombach, “Goal Question Metric (GQM) Approach,” John Wiley & Sons, Inc., 2002.

[7] L. De Rore, M. Snoeck, G. Poels, and G. Dedene, “Cocomo II as productivity measurement: a case study at KBC,” *Available SSRN 1369122*, 2008.

[8] B. W. Boehm and R. Ross, “Theory-W Software Project Management Principles and Examples,” *IEEE Trans. Softw. Eng.*, vol. 15, no. 7, pp. 902–916, Jul. 1989.

[9] S. I. Shahid Iqbal and M. N. A. K. M. Naeem Ahmed Khan, “Yet another Set of Requirement Metrics for Software Projects,” *Int. J. Softw. Eng. Its Appl.*, vol. 6, no. 1, pp. 19–28, 2012.