“[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* [3]

In order to strike a balance between understanding the effectiveness of a pricing strategy or programming associated with a software development effort, metrics and analyses can be used, and there exist a number of tools to affect such measuring. This paper intends to provide a brief overview of the types of metrics that are available and to provide some mathematical/graphy-theory based illustrations of the effectiveness of the metrics available, as described:

* Per Leveson, metrics for any given software project may serve as a measure of:

1. **Reliability**: as computed by SCA or Dynamic Analysis to determine:

- Bugginess - # bugs left in code (defect density – via static or dynamic analysis)

- Future failure times(MTBF)/ operational reliability – (via dynamic analysis, using input-domain model, which approximates the current/prospective situation to a previous instance vs. reliability growth modelling)

1. **SW Requirements validity**: can be measured by requirements tracing[1] - (more change requests more often may mean requirements possibly weren’t adequate in first place)
2. **Programmer productivity:** a pretty ‘intangible’ property (due to feasibility/practicality, this probably has not been ‘scientifically’ addressed to a level of ‘fine granularity’ as human productivity will always remain somewhat subjective and intangible)
3. **Cost projections –** using ie: Algorithmic cost modelling (uses historical data to project costs of current based on analogy to past project) / winning-bid strategy (build out price est. towards expected final budget) / top-down (decomposition-oriented) / bottom-up (cost accumulates)

* Reliability of Metrics – are we comparing like to like? … maybe it’s not so reliable – is a line of code just one line if it contains two statements.
* Verification may be hindered by “Observer Effect” (may refer to *definitions* to discover how this relates) – a point I intend to highlight in my paper is: types of conditions that are measurable when assessing software (-metrics). As dynamic-analysis methods may realize effects of this phenomenon I may focus on it minimally and transition from it into broader picture, in terms of what are alternative approaches that are not subject to it.
* Output-generating statements can affect location of variables in memory

* can be improved by asserting invariant (pre-)conditions are adhered to[2](pg1) – this point may be addressed minimally but is not intended to be a focal point in my paper.
* There exist a variety of programs that can be useful as tooling for post-mortem debugging:

1. For java, there are post-mortem debugging tools that have been developed incl/ JDM,JMAP
2. In python & ruby, a debugger can be attached to the interpreter or run simultaneously and interactively alongside the program during execution
3. In C/C++/Objective-C, there are extensive facilities for handling crashes, saving dumps, and analyzing them; while in some newer dynamic languages, these facilities may not be available or applicable such as by way of presenting merely a list of global symbols, pointers to thread stacks, and all of a program’s virtual memory – rather, users may need to gain insight into abstractions of the state of the program at it’s time of crash in the context of the environment such as interpreter/virtual machine/
4. For javascript, there is a node.js facility developed by David Pacheco, Pacheco also references Eric Schrock’s AJAX strategies

**Definitions:**

Cyclomatic Complexity:

Software metric that directly measures the number of linearly independent paths through a program's source code. Often measured by conditionals.

Tokens:

In programming, a [token](http://www.techterms.com/definition/token) is a single element of a programming language. There are five categories of tokens: 1) constants, 2) identifiers, 3) operators, 4) separators, and 5) reserved words. For example, the reserved words "new" and "function" are tokens of the JavaScript language.

[Glossary of Graphing terminology](http://en.wikipedia.org/wiki/Glossary_of_graph_theory) (ie: concise reviews of ‘subgraph’, ‘DAG’, ‘[Eulerian circuit](http://en.wikipedia.org/wiki/Eulerian_path)’, etc):

Requirements Traceability:

*Requirements tracing is the process of documenting the links between the user requirements for the system you're building and the work products developed to implement and verify those requirements. Sometimes, this is a matter of tracing a requirement described in one document to its measure described in another* [5]

Reliability Growth Model:

*A* [*reliability growth model*](http://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/DepSecAssur/RGM.html) *is a model of how the system reliability changes over time during the testing process*

(Heisenbug is sensitive to..)Observer Effect:

*Attempts to find the bug by inserting debugging code or running in a debugger often disrupt the sequence of events that led to the bug, making it go away.[6]*

# Bibliography

[1] Etestinghub, “No Title.”

[2] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin, “Dynamically discovering likely program invariants to support program evolution,” in *Proceedings of the 21st international conference on Software engineering*, 1999, pp. 213–224.

[3] 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.

[5] Y. Shin, J. H. Hayes, and J. Cleland-Huang, “A framework for evaluating traceability benchmark metrics,” 2012.

[6] E. D. Berger, “Software needs seatbelts and airbags,” *Commun. ACM*, vol. 55, no. 9, pp. 48–53, Sep. 2012.