

Scientific Reverse Engineering

Spencer Rugaber
College of Computing
Georgia Institute of Technology
(spencer@cc.gatech.edu)

Part of a science of software design should be learning from former and current systems. The study of these systems is called *reverse engineering*, the systematic extraction and recording of design information. In addition to analyzing the artefacts themselves, we should uncover as much as we can about their context of use: the requirements that they were designed to satisfy, their history of changes, their failures and their successes.

What would it take to make such a study of former and current software designs scientific? *Science* is the controlled discovery of knowledge. Science progresses through the coevolution of theory and experimentation. The following are some examples of research that exhibit scientific characteristics by including both empirical study of existing artefacts and theoretical implications on software design.

Belady and Lehman’s study of system evolution dynamics [2]: This is a classical study of a series of releases of a software system. Measurements were made of systems size, modules handled and changes made over time. A predictive model was developed and validated on other systems. Empirical laws of system evolution were proposed.

Baker’s study of clones [1]: A *clone* is a structural phenomenon in which duplicate or near duplicate design elements appear in code. In this paper, a theoretical explanation was proposed for the occurrence of clones, empirical data was obtained on the extent of the phenomenon, and a precise measurement technique was provided in the form of an algorithm.

Murphy et al.’s study of call graph extractors [3]: A *call graph* is a representation of a certain class of design information—which subprograms invoke each other. The authors studied a collection of call graph extraction tools applied to

software artefacts, pointing out differences among them. They discussed practical and theoretical difficulties with the process.

Clearly, we have much to learn from the study of software artefacts, but the study should be scientific in the sense that precise and repeatable measurements can be made. Researchers studying reverse engineering have proposed a variety of techniques and representations. But there is no overall basis for measuring and comparing the quality of such approaches. What might such an overall basis look like? Several characteristics seem desirable.

- **Repeatability:** a measurement should be well-defined so that different researchers could measure the same artefact and get the same results.
- **Predictability:** the amount of effort required to produce a measurement should be predictable so that the measurement process can be planned and managed.
- **Precision and accuracy:** the limitations (error bounds) of a measurement should be included with the measurements made.

What factors should be measured? Measurements of reverse engineering representations can be extrinsic or intrinsic. *Extrinsic measures* reflect the extent to which a representation is useful for accomplishing some task, such as debugging a program. Other uses to which a representation can be put include documentation, detection of reuse candidates and support for reengineering and refactoring.

Intrinsic measures are taken on the reverse engineering representation itself. There are a variety of such representations each exhibiting one or

more relationships among design elements. In order to scientifically judge a representation, quality attributes must be identified. Classes of attributes include those related to breadth, depth, and intentionality.

- **Breadth:** the extent to which a representation covers all aspects of a program. For example, it is relatively easy to produce a representation of a single subprogram. But such a representation provides less knowledge than a set of such representations, one for each subprogram in a program. Breadth is an indication of coverage or completeness of a representation.
- **Depth:** the level of abstraction at which a representation describes a program. Software design comprises levels of refinement. Design representations are tailored to particular levels, such as system, architectural, or module. Measures of representation depth indicate the nature and precision of the information provided.
- **Intentionality:** the extent to which the representation sheds light on the purpose of the program. Most representations are program based, deriving directly from a program's syntactic structure. These representations should be complemented by representations that reflect program requirements. The extent to which this is accomplished by a representation is measured by its intentionality. To measure intentionality, a separate model of a program's application domain can be used. A reverse engineering representation is intentional to the extent that each program construct can be mapped to a corresponding domain concept. The domain concept provides an answer to what role that construct plays in accomplishing the program's purpose.

Well-defined measures of reverse engineering representations would engender a variety of benefits.

- Comparisons could be made between representations, particular with regard to how they support various maintenance tasks.

- Similar comparisons could be made between representation tools. To the extent that research and commercial vendors have targets to pursue, competition will improve the breed.
- Data could be gathered on how long it takes to produce a representation with certain characteristics. Such data would enable managers to better plan maintenance efforts.
- Finally, measurements of design representations can inform theory. Measurements can point out anomalies or confounding factors that can serve to extend or simplify theories.

Design is inherently evolutionary; we learn from the deficiencies of our products. Scientific measurement of software artefacts can serve to improve the feedback, ultimately yielding better software.

The author is a faculty member in the College of Computing at Georgia Tech. His research interests are in the areas of software engineering and human-computer interaction. His specific interests include program understanding, reverse engineering, software maintenance, software design, and model-based user interface generation. Last year, he served at the National Science Foundation as Program Director for the Software Engineering and Languages Program.

References

- [1] Brenda Baker, "On Finding Duplication and Near-Duplication in Large Software Systems," *Working Conference on Reverse Engineering*, 1995.
- [2] L. A. Belady and M. M. Lehman. *Program Evolution. Processes of Software Change*. Academic Press, 1985.
- [3] Gail C. Murphy, David Notkin and Erica S.-C. Lan. "An Empirical Study of Static Call Graph Extractors." *Proceedings of the 18th International Conference on Software Engineering*, Berlin, Germany, March 25-29, 1996.