Creating a Research Infrastructure for Reengineering

Spencer Rugaber and Linda M. Wills
Georgia Institute of Technology
{spencer@cc, linda@ee}.gatech.edu

I. State of Reengineering Research

The research area concerned with software re-engineering is maturing. There are several regularly scheduled conferences [22][23][26], several government funding initiatives have recently begun in the area [7][8], special issues of technical journals have been devoted to the topic [1][16][25], and, most importantly, there is recognition in the software industry of the complexity and pervasiveness of the reengineering problem. In response to this maturity, several efforts are underway that reflect on the research area itself: a best practices guide is being edited by the Software Engineering Institute, a tool list has been constructed [28], terminological convergence is underway [5], and case studies are being collected by the Software Technology Support Center, Hill Air Force Base. Furthermore, panel sessions at conferences have considered research issues and future directions [6][27][31]. This position paper considers the maturation process and proposes some steps that we can take to make further progress.

A. Obstacles to Achieving Impact

Despite all of the activity described above, reengineering research has had notably little effect on actual software reengineering practice. Most of the published papers in the field present techniques supported by prototype tools; few of which have actually been used on real projects. The typical software developer on such a project is still using code reading and test runs as the primary ways of learning about an existing system and an editor and compiler as the primary ways of changing it. We see several reasons for the research area's lack of impact.

1. **Difficulty of communicating value:** The foremost reason why research results have not made a significant difference to practice is the difficulty the research community has had in communicating the role of reengineering in the software lifecycle. Whereas it is now well-understood that software maintenance is a dominant cost factor, the reengineering alternative to continued maintenance, if it is considered at all, is thought of as an expensive, time consuming detour from the clear path to the next maintenance release. We have not effectively communicated the long term benefits made possible by taking the detour.

2. **Difficulty of assessing value:** Of course, a major reason why we have difficulty communicating the value of reengineering is that we have difficulty measuring its costs and benefits in the first place. Although several cost models have been proposed [2][4][20], there has been little validation, and there are certainly no generally accepted decision procedures for planning reengineering efforts. If we cannot effectively predict reengineering costs and benefits, we cannot expect a manager to be willing to invest the significant resources required to completely analyze and restructure a system. And without a clear understanding of costs and benefits, there can be no effective decision procedure to help a manager judge alternatives such as starting over, reengineering, continuing maintenance, or abandoning a product.

3. **No validated reengineering process:** Another difficulty confronting the maintenance manager is that there is no accepted reengineering process. Straightforward questions like the following do not have agreed upon answers: how much up-front time to spend on program comprehension

before recoding begins, what form of representation should the results of the comprehension take, how does testing a reengineered system differ from testing one under maintenance or one built from scratch. Because the individual steps are not well-understood, it is difficult for the manager to measure progress, further increasing project risk.

4. **Technical difficulties:** There are also stumbling blocks at the technical level. The diversity of programming languages, compilers, and hardware platforms compounds the difficulty in effecting tool penetration throughout the marketplace. Furthermore, information useful to research tools and produced by commercial compilers is not accessible through published interfaces, thereby forcing tool developers to divert energy better spent on exploring new techniques. Even well-understood technical ideas, such as program slicing [31], have not penetrated the commercial market due to the difference in scale between a research tool and a commercial product.

In summary, we need to have a clearer understanding of what our area's goals are, how those goals relate to customers' needs, how we can more effectively satisfy those needs than we have in the past, and how we can communicate what we have done to practitioners and management.

II. A Research Infrastructure

There are some steps that we as a community can take to address these issues and to leverage our efforts. Some of these are already underway, some are understood but need to be implemented, and some are themselves research questions that need to be examined. All of them need contributions from the research community.

A. Taxonomy

One fairly mature effort is the IEEE Reverse Engineering Taxonomy project. Initiated by Elliot Chikofsky and James Cross, this effort has resulted in a published description [5] of our field's terminology. This document is now the *de facto* source of definitions and should serve as a basis for extensions and elaborations as the field matures.

B. Common portable and interoperable intermediate representations

Because there have been numerous efforts within the larger computer science community to establish standard forms for intermediate representations (IRs), we currently suffer from an over abundance of representations. To name a few, not necessarily equivalent forms, we have IDL (two varieties) [15] [18], IRIS [13], DIANA [11], and generic Abstract Syntax Trees (ASTs). Although they are intended for slightly different purposes, it is not clear that their special features compensate for the resulting reduction in interoperability. Thus, we are left with the unpleasant choice of somehow deciding on one or finding a way to facilitate interoperation.

C. Vendor requirements

If we can agree on a plan for IR convergence, then we can ask compiler and tool vendors to cooperate. One preliminary step would be to convince them to publish application programmer interfaces (APIs) or intermediate file formats for their IRs. Researchers could then take advantage of commercial progress in dealing with industrial scale issues, thereby freeing energy for exploring research questions.

D. Reengineering resources

The advent of the World Wide Web has encouraged individual research groups to construct home pages containing references to resources related to the field of reengineering. Such pages list conferences, vendor and tool descriptions, and bibliographies describing relevant literature. Many of these pages are accessible from the IEEE Committee on Reverse Engineering's web site [12], which includes a resource repository. The advantages of the World Wide Web for supporting the research area include its eclecticism, its dynamic nature, and its cross linking. Disadvantages include the possibility of being out of date, redundancy, and lack of quality control.

E. Reengineering task descriptions

As another condition of maturity, the field of reengineering must define itself. This includes not only resolving the remaining naming and terminology issues, but also composing a systematic description of when and how reengineering activities can contribute to the software development process. Such a description must take the form of a list of specific reengineering activities or tasks, including a definition of the activities, a description of their inputs and outputs, their costs and benefits, and any other collected wisdom the field can offer to practitioners. There are several benefits to such a list. These include communicating to practitioners situations with which we might be able to help, the initial steps to a process model, and the sanity checks engendered by such reflections.

F. Graduated series of milestones/challenges

Another step we can take is to define a series of specific research challenges to strive for, similar to the Grand Challenges in High-Performance Computing and Communications [17] and the goal of satisfying the Turing Test [29] in Artificial Intelligence. We need tangible targets ranging from what we are within sight of now (for example, an industrial strength slicer for the C language), to medium-term challenges (such as measurable diagnostic assistance in finding the location of bugs, given failing test data), to true understanding (such as domain-specific program documentation generators).

G. Standard Data Sets/Benchmarks

Coordinated and measurable progress in our field depends critically on establishing standard data sets and benchmarking tasks that can be used to quantitatively evaluate and compare reengineering tools and techniques. This will help to guide and validate research efforts in the field, analogous to the use of the SPECmark suite for computer performance evaluation [10]. To start, we need to collect information about publicly accessible software systems, including their associated documentation, modification history, test cases, and other available information. The standardization of these data sets, in conjunction with efforts to establish clear reengineering task descriptions, will provide the basis for benchmarking task scenarios representative of important, common reengineering challenges. This will be instrumental in evaluating and communicating the value of our work.

H. Repository

The rapid growth in the field of reverse engineering over the past few years has generated a wealth of tools, techniques, methodologies, publications, and other resources. To enable sharing of resources and to avoid unnecessary duplication of effort, it is important to collect and synthesize our products. Such products include the following:

1. **Data sets** - specific source code and test data using which researchers can compare results.

2. **Scripts and other code** - such as Refine language programs for specific analyses.

3. **Libraries of cliches and transformation rules** - in support of program recognition and tranformational analysis.

4. **Grammars** - machine processable programming language grammars.

5. **Experiments** - descriptions of repeatable program comprehension experiments and the materials to replicate them, such as described by [14] and [24].

6. **Case studies** - a systematic collection of case studies, uniformly documented so as to provide guidance to development managers, such as those performed by [9]. This would serve as a first step to a cost model, such as Cocomo [4], and for a reengineering decision procedure.

7. **Public domain utilities** - research tools and basic utilities (such as parsers visualization tools, analyzers, graph layout and editing facilities), including documentation supporting automatic applicability qualification.

8. **Literature references -** up-to-date references and literature in our field such as texts, handbooks, and papers.

There are many difficulties, including technical, legal, logistical, and even theoretical issues that make constructing this repository difficult. For example, for the repository to be truly useful, it should be systematically documented and indexed—which requires a substantial effort. But we do not think that the problems are insurmountable, and the value of such a repository, both to practitioners and to researchers, warrants pursuing the possibility.

III. Infrastructure Implementation

A. Committee on Reverse Engineering

Many of the discussions leading to this paper have taken place in the context of the Committee on Reverse Engineering (CORE) of the IEEE Computer Society's Technical Council on Software Engineering. Since the taxonomy project was begun under the aegis of CORE and the IEEE Computer Society sponsors (in part) the conferences mentioned earlier, CORE represents an appropriate organizing vehicle for implementing this proposal. CORE has created a web site [12], where a repository has been established to collect reengineering resources, the taxonomy of terminology, pointers to relevant conferences and literature, and the many products of our research. In addition, for several years, CORE has published a newsletter [19] that facilitates communication in our area and helps form ties to related areas.

B. Volunteers

The effort described in Section II is ambitious, and it will take considerable time to realize. Moreover, the implementation will be entirely voluntary. We suggest that an initial step is for individuals to volunteer to organize the various components by presenting an organizational plan including a time and effort estimate, resources required, issues to be resolved, etc. On-line discussion would raise awareness and hopefully stimulate a synthesis.

C. Outreach

Unfortunately, even an outpouring of volunteer effort from the reengineering research community will not suffice to implement the proposal. We need external input from at least three other research and industrial areas.

1. **Test and evaluation.** The test and evaluation research area has been concerned for some time with questions similar to ours, such as common representations and tool repositories. In fact, there are more areas in which we overlap with them than in which we differ. As a separate but related effort, we suggest discussion aimed at moving our communities closer together.

2. **Compilers.** Of course, successfully implementing this proposal depends on support from the compiler community, both its researchers and the industry itself. Not only are they concerned with many of the same questions, but access to the intermediate results of currently available commercial and research tools can mitigate redundant effort.

3. **Domain analysis.** The area of domain analysis [21] promises to play an important role in reengineering in the future. Comprehensive reengineering of a software application requires extensive knowledge of the application's domain. While little technology currently exists to support these efforts, we should remain aware of what is going on in this newly formed area and attempt to communicate our needs to them.

D. Advisory board

Just as success for the reengineering research community requires active communication with our customers, so must successful implementation of this proposal involve wide participation. We propose the establishment of an advisory board to which we would report progress and from which we would solicit advice and contacts. Among the participants would be the following.

1. **Industrial practitioners / customers** - an enlightened software development middle manager, whose expertise and contacts would facilitate beta testing of our technology.

2. **Government funding agents** - members of the Department of Defense, NSF, ONR, and other government funding agencies would be invaluable, particularly since they are supporting much of the research in the area.

3. **IEEE representative** - a member of CORE to facilitate repository logistics and provide other organizational access would be of value.

4. **Academic / industrial researcher**.

5. **Tool vendors (reengineering, compiling)**.

6. **Consultant** - as many reengineering efforts are guided by advice from external consultants, it would be useful to have one as a member of the board.

IV. Summary

We have proposed several steps which we believe would encourage maturation of the research efforts in the area of software reengineering. These include increased interaction with industry, the development of a repository of research artifacts, and convergence of intermediate representations. The paper is a proposal which requires considerable discussion and consensus before it can be realized. We encourage your thoughts and your participation.

V. References

[1] "Special Issue on Reverse Engineering." *Automated Software Engineering*, 3(1-2), 1996.

[2] Lowell Jay Arthur. *Software Evolution*. John Wiley & Sons, 1988.

[3] "A Bibliography on Reengineering." http://www.informatik.uni-stuttgart.de/ifi/ps/reengineering/reengineering.html.

[4] Barry W. Boehm. *Software Engineering Economics*. Prentice Hall, 1981.

[5] Elliot J. Chikofsky and James H. Cross II. "Reverse Engineering and Design Recovery: A Taxonomy." *IEEE Software,* 7(1):13-17, January 1990.

[6] James H. Cross II and Spencer Rugaber. "Speaker Cites Standard Data Sets as a Major Challenge Facing Software Reverse Engineering Researchers." *Computer*, IEEE Computer Society, 26(11):83-84, November 1993.

[7] "Dear Colleague Letter." National Science Foundation solicitation for Evolutionary Design of Complex Software initiative. http://www.cise.nsf.gov/cise/ccr/SEDearColleague1.html.

[8] "Evolutionary Design of Complex Software." DARPA Program. http://www.ito.darpa.mil/ResearchAreas/EDCS.html.

[9] P. Fiore, F. Lanubile, and G. Visaggio. "Analyzing Empirical Data from a Reverse Engineering Project." *2nd Working Conference on Reverse Engineering*, Toronto, Ontario, Canada, July 14-16 1995, 106-114.

[10] Ran Giladi and Niv Ahituv. "SPEC as a Performance Evaluation Measure." *IEEE Computer*, August 1995, pp. 33-42.

[11] G. Goos and W. A. Wulf. "Diana Reference Manual," CMU-CS-81-101, Department of Computer Science, Carnegie-Mellon University, 1981.

[12] "IEEE Computer Society, Technical Council on Software Engineering, "Committee on Reverse Engineering" home page. http://www.tcse.org/revengr/.

[13] "The IRIS Toolset." http://laser.cs.umass.edu/tools/iris.html.

[14] A. Lakhotia. "Understanding Someone Else's Code: Analysis of Experiences." *Journal of Systems and Software*, Elsevier Science, Volume 23, December 1993, 269-175.

[15] David Alex Lamb. "IDL: Sharing Intermediate Representations." *ACM Transactions on Programming Languages and Systems*, July 1987, 9(3):297-318.

[16] "Maintenance, Reverse Engineering & Design Recovery," Special Issue of *IEEE Software*, 7(1), January 1990.

[17] National Research Council. *Computing The Future*. National Academy Press, Washington, D.C., 1994, (see also http://www.hpcc.gov/blue96).

[18] Object Management Group. "Common Object Request Broker Architecture and Specification," OMG Document Number 91.12.1.

[19] Mike Olsem, editor. “Reverse Engineering Newsletter.” Committee on Reverse Engineering, Technical Council on Software Engineering, IEEE Computer Society. http://www.stsc.hill.afb.mil/~red/IEEE.html.

[20] Mike Olsem, Chris Sittenauer, John Clark, Alan Giles, and Dennis Barney. *Software Reengineering Assessment Handbook*. Software Technology Support Center, Hill Air Force Base.

[21] Ruben Prieto-Diaz and Guillermo Arango. *Domain Analysis and Software Systems Modeling*. IEEE Computer Society Press, Los Alamitos, California, 1991.

[22] *Proceedings of the 1994 International Conference on Software Maintenance*, Victoria, British Columbia, Canada, September 19-23, 1994.

[23] *Proceedings of the 3rd Workshop on Program Comprehension*, Washington, D.C., November 1994.

[24] Vaclav Rajlich, James Doran, and Reddi T. S. Gudla. “Layered Explanations of Software: A Methodology for Program Comprehension.” *Third Workshop on Program Comprehension*, IEEE Computer Society, November 14-15, 1994, Washington D. C., 46-52.

[25] “Reverse Engineering.” Special Issue of *Communications of the ACM*, 37(5), May 1994.

[26] *Second Working Conference on Reverse Engineering,* Toronto, Ontario, Canada, July 14-16 1995.

[27] P. G. Selfridge, R. C. Waters, and E. J. Chikofsky. “Challenges to the Field of Reverse Engineering--A Position Paper.” *Working Conference on Reverse Engineering*, IEEE Computer Society Press, Baltimore, Maryland, May 21-23, 1993, 144-150.

[28] Chris Sittenauer, Mike Olsem, and Daren Murdock. “Re-engineering Tools Report.” Software Technology Support Center, Hill Air Force Base, May 1992.

[29] Alan Turing. “Computing Machinery and Intelligence.” *Computers and Thought*, E. Feigenbaum and J. Feldman, editors, McGraw-Hill, 1-35, 1963.

[30] Mark Weiser. “Program Slicing.” *Proceedings of the 5th International Conference on Software Engineering*, San Diego, California, March 9-12, 1981, IEEE Computer Society, pp. 439-449.

[31] Linda Wills and James H. Cross II “Recent Trends and Open Issues in Reverse Engineering.” *Automated Software Engineering*, 3(1-2), 1996.