Reuse R&D: Gap Between Theory and Practice

Chair: Mansour Zand, University of Nebraska
Panelists: Vic Basili, University of Maryland, USA
Ira Baxter, Semantic Designs Inc. USA
Martin Griss, HP-Labs, USA
Even-Andre Karlsson, Q-Labs, Sweden
Dewayne Perry, AT&T Labs, USA

Introduction

Mansour Zand

In SSR97 we presented a panel with the same focus that was very well received [SSR'97]. Although we spend 30 minutes more then allocated time there were many of session audience that were unable to state their opinion or ask questions. Immediately, after the panel, and later on, several people suggested that this panel should have a follow up in the next SSRs or other reuse conference. Indeed after two years still there are a lot of discussions on of future of reuse and the gap between research and practice and the degree of its contribution in software development in the past 15-16 years. There is little doubt on the reuse R&D contribution. However, there are ongoing discussions on how significant the impact have been. Specifically, there are some questions on validity of academic or "basic (pure) research" done in the past decade. Also, there are concerns on duplication, within and outside "reuse" community[Poulin], of works that may implies that reuse R&D is in stalemate. Furthermore, recent development of CBD/CBE/CBA mainly by "practitioners" is a major question mark on how and why researchers in academia and R&D development centers were left behind.

To maintain a continuity on SSR'97 panel the present panel intends to extend discussions on three different prospective on reuse research and practice:

1- Those mainly involved in research present their point of view, from their research on how things have changed, what concepts have been or should be discarded, and what new challenges have emerged
2- Participants from industry: software development practitioners, talk about if reuse research has been relevant and useful in their day-to-day work, whether the tools and methods put out during the past ten years have been useful, and what their expectations are from useful tools and techniques.
3- Not all the "theoretical oriented" and "sophisticated" solutions presented by researcher have not exactly thrilled the practitioners by researchers. They believe that many of those solutions are either based on a set of outdated premises and are either obsolete or are not practical. The researcher, claim that although some of the premises of the problem have changed but there are a large number of fundamental problems that are not being solved scientifically and until we do not have a sound scientific foundation solution are temporary and ad hoc.

We can look at this problem from two different perspective software engineering discipline and reuse community.

In general, existence of such a gap is not inherent property of software engineering or more specifically software reuse. It is not unusual that due to demand for quick solution researchers be unable to provide the solution on a timely manner. Furthermore, if a solution is provided may not have the desirable properties of a sound and reliable solution. In the last decade demand for development and rapid delivery of new software systems have been overwhelming. Software engineer and developers under such immense pressure are looking for quick solutions and can not offered to wait for a comprehensive and reliable solution [SSR97]. It is a known fact that a good number of large projects were pronounced obsolete even before they were completed.

Under such a pressure software engineering community have focused most of its effort on development or use of less formal and more ad-hoc solutions. Therefore, adequate resources are not available to promote development of scientifically sound set of methodologies and models. Furthermore, unlike other engineering

disciplines standards are yet to be established. A large number of software engineers practitioners and some of the researchers are looking forward for one of the giants software companies to forces its practice as de facto standards models. As it is stated by other panelists in the abcense of scientifically sound models and methodologies and standards, as bedrock of the discipline, it would be very difficult to perform an expeditious and reliable research in such a demanding setting. Furthermore, some of the promising researches are not appraised and some of the substantiated results are not being used. This view is shared by other panelists and is considered of one of the major shortcomings in software engineering research and development [Basili and Baxter].

Other perspective is more specific to reuse and "systematic reuse" community. The majority of published works by practitioners are from those large software shops with adequate resources. Moreover, most of the reports are success stories and those software shops that have not been as successful, are not very much forthcoming to describe what have gone wrong. Another major shortcoming of the research in reuse community is the lack of adequate work to address the need of small and medium sized software shops. [zand98]. There is little doubt that great progress has been made to understand the underlying and necessary factors for effective software reuse and conditions or factors of success and failure. [Karlsson and SSR97]. However, because of relative isolation related communities seem to not utilizing findings. They are providing simpler solutions that are more convenient to be utilized in industry [Griss].

To Summarize recommendations of the panelists:

- Methods and models are needed to present body of knowledge in a scientific form that can be used with great confidence
- Researcher need to work more closely with practitioners to understand real problems, develop and verify their practicality of their scientific solutions.
- "Historical software research" - Evaluation and study of past failure and success are essential. This study should include non-technical factors such as organizational and cultural factors.
- Leadership of research community needed to move industry beyond different forms of code reuse.
- Need for protocols and standards.
- Time to get a research into practice is curtailed.
- Reuse community needs to more aggressively publicize its research finding by taking it to main stream software engineering communities.

Reference

[Poulin]Poulin, J. " The Foundation of Reuse", *Proceedings of WISR'9,* Austin, TX, 1999.
[SSR'97] M. Zand, G. Arango, M. Davis, R. Johmson, A. Watson, "Reuse R&D: is it on the right track", *Proceedings of SSR'97,* pp 212-216, Software Engineering Notes, ACM, May 1997.
[Zand] M. Zand, "Organizational and Management Issues: Are we there yet," *Proceedings of WISR'9, Austin, TX, January 1999.*

Reuse R&D: Gap between Theory and Practice

Victor R. Basili

One of the criteria for reuse of knowledge in any discipline is that the knowledge is credible, packaged in some usable form, and that the risks associated with use of that knowledge are clear. It implies there is a certain maturity associated with the information, i.e., the strengths and weaknesses of the knowledge package have been made clear, based upon analysis and empirical study.

Common wisdom, intuition, speculation and proofs of concepts are not reliable sources of credible knowledge. On the contrary, progress in any discipline involves building models that can be tested, through empirical study, to check whether the current understanding of the field is correct. Credibility comes when what is actually true can be separated from what is only believed to be true. To accomplish this, the scientific method supports the building of knowledge through an iterative process of model building, prediction, observation, and analysis.

The scientific method has contributed to the progress of fields such as physics, medicine, and manufacturing. Unfortunately, in software engineering, the balance between evaluation of results and development of new models is still skewed in favor of unverified proposals. A body of evidence has not yet been built that enables a project manager to know with great confidence what software processes produce what product characteristics and under what conditions.

We also need to build bodies of knowledge by classifying and integrating research results. Too often, promising software research goes unevaluated. Lacking a proper understanding of the usability of an idea, overloaded, risk-averse applications organizations rarely pay attention to research results. This lack of engagement means that the research community does not receive feedback on the viability of new approaches.

To address this issue, we need change they way we to view software engineering research and development. We

need to study and classify past development successes and failures and understand the parameters that limit our progress. This means that researchers need to work with practitioners to understand the real problems, build and validate their solutions in practical situations, and package them with the relevant caveats and limits, so they can be applied appropriately. This allows the researcher to test the new models and the practitioner to move forward in a risk-averse way.

References:

Victor Basili, Forrest Shull, and Filippo Lanubile, Using Experiments to Build a Body of Knowledge, Proceedings of the 23rd SEL Workshop, NASA Goddard Space Flight Center, December 1998.

NSF Workshop On a Software Research Program For the 21st Century, Greenbelt, Maryland, October 15–16, 1998.

Reuse Theory/Practice Gap: Where to Build Bridges?

Ira D. Baxter

There is community concern that Reuse Research is less focused or less effective than it might be, in terms of how much impact it is having on industry, and whether in fact industry seems to be contributing more in practice than the R&D community. We argue that in the short term, major industry players are harnessing some of the better R&D ideas in a more visible fashion than the community, but they still need strong leadership to move beyond simple code reuse. The research community must focus on the next generation of reuse technologies of domain analysis/engineering and generative reuse, and deliver an integrated set of processes and tools.

1 Generic Reuse

Reuse of any set of artifacts must follow two basic phases:

- **Investment phase**
- Define *problem specification language*
- Define component *structure*
- Define component *instantiation/composition technique*
- Define *component specification language*
- Define *component selection process*
- Acquire reusable *components* to solve domain problems
- *Classify* methods using domain specifications
- **Payoff phase**
- *Specify* instance problem to solve
- *Select solution(s)* based on problem specification
- *Compose/instantiate* selected solution(s)

Each reuse research project must somehow address the mechanics or economics of some these issues, and so this list establishes a research agenda by itself. Note that none of these say "code" specifically, allowing the reuse of many types of artifacts. This author is dismayed that most reuse work does not classify itself carefully as to its category, or gauge the impact of that work on the category. This makes it more difficult to tell where work has been done.

2 Research vs. Practice

Reuse research has had mixed impact, as summarized b the following table. However, most of the impact has been via function libraries, black-box component reuse, scripting languages and OO.

An interesting discussion is how much of the widely adopted technology is due to the reuse community, vs. other subdisciplines of computer science or software engineering, and how active that part of the reuse community is. We believe that much of the widely-practiced part is not due mainly to the reuse research community, as the base technologies are relatively mature.

	Used?	Wide Practice?
General Libraries	✓	
Function libraries	✓	✓
Component reuse	✓	✓
Scripting languages	✓	✓
Parametric programs	✓	
O-O programming & frameworks	✓	✓
Generative Reuse	✓	
Domain-Specific Perspective Analysis/Engineering.	✓	
Architectures	✓	
Economic Models	✓	

3 Why isn't Reuse Research more visible/effective?

The practicing engineering community has bought, without consideration, a particular model of reuse. This model assumes an informal problem and component specification language (English), black-box component structure consisting of ***code***, component-composition by programmers-coding-glue, a completely ad-hoc

component acquisition and selection process, given very weak descriptions.

This approach offers modest success, on the order of 10% productivity enhancement. *Reuse R&D will not have any impact unless it offers a significant enhancement beyond this.* An obvious place to start looking is in the failures of the conventional model:

- **Component Interoperability failures**
- **Index scheme failures**
- **Overcommitment to code reuse**

In the vast sea of available components, the first problem is that arbitrary sets of such components cannot even communicate between themselves (CORBA components vs. DCOM components) nor do so with high overhead (CORBA to CORBA, etc.). To define interoperability, one must define *protocols* of interactions, not signatures, and data must be defined abstractly.

Even given potential compatibility, current schemes have no scalable indexing methods. No library with a million components can be successful without it.

The current model is overcommitted to the reuse of code. This has the severe defect of forcing all reasonable configurations (language, data structure, speed, and safety...) of a concept, e.g., STACK, to have an implementation. This is impractical for STACK, and impossible for anything of larger complexity. Consequently, we end up with only a few configuration instances.

Large organizations, such as Microsoft, will appear to offer more to reusers in the near future than research, because of their ability to dictate standards. Such standards alleviate many problems, by defining standards for interfacing (e.g., DCOM), defining the component libraries manually (e.g., the Win32 API), and implementing the base technologies for a large audiences.

4 How to enhance reuse theory impact

The engineering community does not want to use anything new, unless it is well proven. Much of reuse research produces papers saying, in effect, "this idea sounds good, and I have some preliminary evidence". More convincing evidence is needed to achieve adoption.

One approach is to show that one is a delighted consumer of one's own tools; nobody will eat what the cook won't eat. Secondly, one must find a means to evaluate the actual value of the technique. This means that production of the idea is not enough; it must be followed up with an evaluation. That part isn't fun, what with finding real customers, performing experiments, and writing up the results. Funding agencies need to condition their grants on research *and evaluation* to make sure this gets done.

It is a bitter tech-transfer lesson that you can't push technology on a recipient. He has to want it, and its adoption costs must be outweighed by benefits. Often, a single tool has a difficulty justifying this. A reasonable possibility is for multiple, related tools to be integrated into a single package proving collective benefit. Research groups might do well to integrate their results, both to make testing easier, and to enhance the customer pull.

5 Next big theory impact areas: Domain Engineering + Generative Reuse

To achieve more effective reuse, we must solve the Indexing problem. This is the only way that large collections of components have any chance of becoming useful. A key to the indexing problem is domain analysis [Neighbors84], as it provides the basic terminology from which descriptors can be made. More research into domain analysis and indexing is needed.

Overcommitment of reusable artifacts also limits their reusability. Generative reuse technologies, such as Draco [Neighbors84], Batory's JTS [Batory98], and Biggerstaff's AOP [Biggerstaff98], promise the generation of a concrete component designed to fit into the context in which it is needed. This can provide interfaces of the right type, and ensure that the interface overhead is low, allowing even small components to be reused effectively. Domain engineering has the nice synergy of providing the raw materials needed by such systems.

6. Eating the cook's Soup: The Design Maintenance System

We are believers in our own medicine. Assuming that Domain Engineering and Generative Programming are the next big impact areas, we have set out to build The Design Maintenance System™, (DMS™), a domain-based generative system intended to help maintain large-scale software systems, by reusing design information. [BaxPidg97], consisting of records of transformations chosen for an implementation. DMS is an integrated set of tools, including problem domain parsers, design navigation aids, and change management machinery.

In particular, we expect that the payback for an adopter will be a factor of two or better in software maintenance, to make up the price of acquiring and using a complex tool. Building such a system is about 10 percent research (design capture, transform replay, etc). and 90% construction of infrastructure (parsers and prettyprinters for real languages, etc.). We have about 20 man-years invested, and the tools are just starting to show evidence of utility. We hope to be able to provide early evaluative reports in the next year.

References

[Batory98] JTS: Tools for Implementing Domain-Specific Languages, Proceedings of 5th International Conference on Software Reuse, 1998, IEEE.

[BaxPidg97] I. Baxter and C. Pidgeon. Software Change Through Design Maintenance. Proceedings of International Conference on Software Maintenance, 1997, IEEE

[Biggerstaff98] T. Biggerstaff, Anticipatory Optimization in Domain-Specific Translation, Proceedings of 5th International Conference on Software Reuse, 1998, IEEE.

[Neighbors84] J. Neighbors. The Draco Approach to Constructing Software from Components. IEEE Transactions on Software Engineering 10(5), 1984.

Reuse Technology -Why Is Adoption So Slow

Martin L. Griss

Abstract

We have made tremendous progress in understanding the critical technology, methods, processes, and organizational factors that lead to effective reuse. Research into new methods and technologies continues unabated. However, far too few software organizations and schools consider systematic reuse as a key part of their programs. Recent developments such as CBSE and OO patterns do not incorporate key learning from the reuse community. Why does reuse technology transfer seems to be so slow and ineffective? How might we improve the situation?

1 Background

For the last 16 years, I have worked on software engineering and software reuse methods, process improvement, education and accreditation at HP, at the University of Utah[kessler97] and as a member of the ACM/IEEE software engineering education project[griss98a]. Far too few organizations understand, practice or teach systematic reuse (or even systematic software engineering!). Feedback on my reuse book[jacobson97] and from many reuse panels, workshops and tutorials confirms this.

We have lots of promising technology, methods, and guidelines. We understand how issues and choices in several areas could influence critical success or failure of reuse:

- Technology - OO; architecture; patterns; components; interfaces; generators; library systems and classification schemes; ....
- Process - domain analysis, CFRP, DFR-DWR, incremental pilots, process, product and reuse metrics; process maturity models; economics, ...
- People: distinct create, reuse, manage, support organizations; explicit high-level management leadership; domain- and component- engineering skills; roles; ...

Far too few software practitioners, and too few researchers in allied fields, such as OO, architecture and CBSE seem aware of our results. Very few software engineering books contain adequate introductions to software reuse, and far too few students are not exposed to the notion that software reuse can be approached systematically. Very little of reuse is mentioned in the merging "software engineering as a profession" body of knowledge[IEEE98] or accreditation guidelines [griss98a,tracz98].

Why is this? It is rather surprising, since presentations on systematic reuse or component-based engineering have been made by several of us, including myself, at several software engineering and object-oriented conferences. Many reports have been issued on reuse [DOD96]. Many of the techniques being popularized in CBSE or product-line approaches could be seen as those of systematic reuse, though typically do not reference earlier work on reuse, generators, domain engineering, or the like.

This may be just a consequence of the long time between the creation of an idea and its widespread recognition and adoption (ala Kuhn), although other developments in object technology, and languages such as Java, have had widespread impact more rapidly. It may be that we have not focused enough of our writing and technology packaging on the "chasm" that separates early adopters from the mainstream (ala Moore). While the reuse research community holds research conferences and workshops, we have not been very visible or successful in reaching other communities.

One possible cause is that the reuse community has worked on complex technologies and methods with high ceremony, yet most of the software community seems to be looking for simpler solutions, perhaps even silver bullets. High ceremony methods require an organization with high process maturity to achieve success. SEI's claim that systematic reuse and product lines should only be addressed at CMM levels 3 and 4, has certainly discouraged many engaged in systematic process improvement from looking at reuse -- and may accurately reflect a real issue that reuse cannot be effective without an appropriate level of process maturity. Yet we know that significant reuse can and should happen earlier [griss98].

Popular developments such as patterns, which are a way of capturing reusable design, are remarkably simple and were developed by the OO community, and not the reuse community. Similarly, we have concentrated on fairly complex generator and transformation technology, when simple wizards and C++ templates have had much more impact on the community. We need to develop and publish a consensus on best practices that can be immediately adopted, outside the reuse community. We need to more vocally share this "reuse body of

knowledge" outside the reuse community. This will guide several activities in developing standards and courseware, and provide advice to other national strategy setting bodies. We need to better coordinate our efforts in reuse conferences and workshops (ICSR, WISR, SSR, ...), both with our own community of reuse research and practices, and more importantly, in contact with other software engineering activities, such as architecture, objects, software engineering, etc. (ISAW, ICSE, OOPSLA, FSE, TOOLS98...).

References

[DOD96] Reifer, Don, *"Reuse Technology Roadmap"*, Department of Defense, 1996.

[griss98] Martin Griss,CMM as a Framework for Adopting Systematic Reuse, Object Magazine, March 1998.

[griss98a] Griss, Martin, "Letter from the Executive Committee," *Software Engineering Notes,* Vol. 23, No. 5, Sept. 1998, pp. 1-2.

[IEEE98] Bourque, Pierre at. al., "Guide to the Software Engineering Body of Knowledge (SWEBOK)", Strawman Version, IEEE Computer Society, September 1998. (See http://www.ieee.org/).

[jacobson97] Jacobson, Ivar, and Martin Griss and Patrik Jonsson, *"Software Reuse: Architecture, Process and Organization for Business Reuse,"* Addison-Wesley-Longman, 1997.

[kessler97] Kessler, Robert R.,"CS451-CS453 - Software Engineering Laboratory", Computer Science Department, University of Utah, Salt Lake City, UT, 1997. See http://www.cs.utah.edu/~cs451.

[tracz98] Tracz, Will, and Mary Shaw, and Martin Griss, and Don Gotterbarn, "Panel: Views on the State of Texas Licensure of Software Engineers", *Proceedings of FSE-6: ACM SIGSOFT 6th International Symposium on the Foundations of Software Engineering*, Nov 3-5, Orlando, ACM SIGSOFT, 1998, pp. 203-208.

What software reuse research is needed?

Even Andre Karlsson

Software reuse is perhaps one of the least well understood areas of software engineering - and we have a low understanding of what really constitutes success and failures in this area? There are several questions, which makes this hard, I will here list some of them:

1) The success of reuse can only be proven over time, i.e. we can only then see if something is really reused, and what savings it gave. Why was this asset reused, and what made it hard or easy to reuse? Also why other assets where not reused?

2) There are soft lines between use of standard components, reuse and good design (i.e. anticipating future product changes) that tends to marginalize reuse, i.e. we can't distinguish these things.

3) Reuse is largely dependent on organizational factors, i.e. product strategies, marketing and how development is organized.

All of these issues need real objectives in real organizations to study over a longer period of time. It also requires a quite deep understanding of the organization and product where reuse is happening. We need to be very precise in describing what is really reused and how it is or is not reused, the current studies mentioning a reuse percentage are not very helpful to really understand what has happened.

Currently we have too few such studies, i.e. it does not seems to attract academia. It could be too hard or too much work, and the payback may not be substantial enough. Probably we will find just a lot of ordinary good or bad decisions in the organization that resulted in the current reuse status.

It will not be research in the form of "new ideas", but rather what I will call "historical software engineering", i.e. a careful examination of the past, not through experiments but through examining real life, i.e. system evolution, organization and politics. This sort of "research" has not been very popular, but I think is needed to give us a better understanding.

We have seen some such studies, but more in the organizational and process area, i.e. Microsoft Secrets, which have been very popular in industry, but probably is not considered as real research. There is a need for such forms of researches that are related to product issues.

One such study that I would have found very interesting is described below:

Software architecture is recognized as one of the most important factors for a successful product. There has been several successful products in the telecommunication domain for the last 20 years, e.g. 5ESS, System 12, AXE 10, etc. The architectures of these systems are rather different, but these architectures have each been basically constant over this period, and all have managed to incorporate all the external changes in the systems. The purpose of this study is to analyze how the systems have been able to adapt to the external changes in markets and technologies, to learn something about which factors of the architecture was important, and the process of evolving an architecture. We will also try to understand why some changes are more difficult in some systems than in other. Our assumption is not that any of the architectures were better or worse than the other, but they are different, and we want to learn how these differences have impacted the last 20 years.

So maybe what I would say is that software researchers are more interested in their own ideas - and particular small nice ideas which can easily be written into a nice research paper, than real life - and I would like that to change.