Software Reuse: Objects and Frameworks are not Enough

Martin Griss
Software Technology Laboratory
HPL-95-03
January, 1995

reuse, objects, systematic process

Software reuse is a widely desired and often-touted benefit of employing object technology (OT), yet too many object-oriented (OO) reuse efforts fail because of too narrow a focus on technology. This column will explore the lessons of systematic software reuse from several perspectives, such as process, organization, management, methods, technology, architecture, tools, patterns, frameworks and practical experience. One lesson is that while OT is in fact neither necessary nor sufficient for effective systematic reuse, it is a most promising enabling technology. My goal is to illuminate how to more effectively use OT to support systematic OO reuse.

Internal Accession Date Only
To be published in *Object Magazine*, February 1995

SOFTWARE REUSE: OBJECTS AND FRAMEWORKS ARE NOT ENOUGH

Martin L. Griss
Hewlett-Packard Laboratories
Palo Alto, California

1 Introduction

One of the most frequently stated reasons for selecting OT is to enhance component and design reuse. Other popular reasons are to improve time to market, flexibility, maintainability and costs, benefits usually attributed directly to reuse itself. While OT is believed to be very important to achieving the long sought-after goal of widespread reuse, many organizations naively equate reuse with objects. They adopt OT, expecting it to automatically ensure reuse. And they are quite often disappointed that they do not actually achieve enough reuse to justify the OT adoption expense. Without an explicit agenda of reuse that includes reuse-supportive organizations, processes, guidelines and mindset, reuse will not be successful. Pittman calls this absence of an explicit reuse agenda by OT adopters the "hidden agenda" of OO reuse [Pittman93]. Many OT prophets reinforce this myth by only discussing techniques such as inheritance and frameworks, and more recently, patterns, without highlighting the importance of systematic reuse planning.

There exist many successful examples of non-OO reuse projects. These include the BaseWorkX software bus-based operation support systems at AT&T (in C), scientific and spacecraft software at NASA (in Fortran), instrument and printer firmware at HP (in C), and financial and insurance software (in COBOL). Several of these are described in more detail in "Software reuse: From library to factory" [Griss93]. More recently, experiences with Visual Basic [Udell94], and large-scale OO reuse at Verilog SA, Brooklyn Union Gas and Ascent Logic have been reported. In almost all cases, a simple architecture, a separate component group, a stable application domain, standards and organizational support are the keys to success. Correct handling of these (largely non-technical) issues is almost always more critical to successful reuse than the choice of specific language or design method, yet too many OT experts choose to ignore these factors.

2 Systematic Reuse: Non-Technical Factors Dominate

By systematic reuse, I mean an institutionalized organizational approach to product development in which software assets are intentionally created or acquired to be reusable. These assets are then consistently used and maintained to obtain high levels of reuse, thereby optimizing the organization's ability to produce quality software products rapidly and effectively.

Over the last ten years, software reuse researchers and practitioners have learned that success with systematic reuse requires that careful attention be paid to both technical and non-technical issues. Furthermore, the non-technical issues are more pervasive and complex than at first realized [Frakes94]. Without a systematic and joint focus on people, process and product issues, a project will not succeed at managing the scope and magnitude of the changes and investment needed to achieve reuse. Simply creating and announcing a reusable class library will not work. Without a "reuse mindset," organizational support, and methodical processes directed at the design and construction of appropriate reusable assets, the reuse investment will not be worthwhile. For this reason, research and practice in the reuse community have developed assessments, guidelines, processes, models, methods and prescriptions that promise more cost-effective reuse.

While attending and participating in numerous conferences and workshops over the last year, I became concerned with the lack of dialogue between the OT and the reuse communities. Unfortunately, it appears that very little of the understanding about systematic reuse seems to have impacted the OT community. Many OT practitioners and researchers seem to be painfully engaged in adopting new methods and adapting basic software engineering techniques to OT, while others are already inventing new methods, without ensuring that they will really improve the software process. When I attended the July '94 San Francisco Object World and the October '94 OOPSLA, I was struck at how many times reuse was mentioned as a major benefit of OT, yet how little was being said about what it takes to achieve effective reuse. Little was said about the importance or cost of domain engineering and organization design. Most of the publishers showcased OT books, totally excluding reuse. Well-known reuse books and collections were not visible. These include those by Tracz, Biggerstaff and Perlis, Hooper and Chester [Hooper91], and Schaefer. Many of these books and other reuse references are discussed in an introductory paper on success factors for systematic reuse by Frakes and Isoda [Frakes94].

Eric Aranow and I developed and presented an OO reuse tutorial at Object World, and were gratified by the attendance, but were only able to reach 100 people. I also attended the November '94 Third International Conference on Software Reuse, to lead a panel on OO reuse. While it was a much smaller

conference than OOPSLA, I was disappointed to see how few attendees viewed themselves as OT experts, and how few had also attended OOPSLA.

It is my goal in this series of columns to bring to your attention these issues, relevant reuse learning and experience, and information about reuse events and activities. In my related research and reuse community activities, I am working with other reuse and OT workers to help integrate these communities. From time to time, I will invite some of these people to co-author columns with me.

3 Systematic Reuse Needs A Systematic Approach

Mounting a large reuse program, perhaps with corporate-wide impact, requires a systematic approach. It is a significant effort to change culture, organization, processes and infrastructure. These changes are not evolutionary and have greater complexity than those associated with incremental and ongoing changes for Continuous Process Improvement (CPI). The scope of these changes is effectively a Business Process Reengineering (BPR) of the software development process and organization. Reuse IS a business issue: We have to change the way we view software at a fundamental level. It is now a corporate asset that needs to be invested in, improved, and leveraged effectively and consistently.

Often, sweeping changes in the software development organization are needed to institute large-scale, systematic reuse. These include business, process, management and organizational changes to:

- fund product family design and construction that are optimized to business goals like improved time to market, decreased development cost, or inter-product compatibility;
- create, maintain and manage reusable assets and repository over the long term;
- separate asset creators from asset users who build applications;
- introduce new management roles and responsibilities, funding models, and development processes;
- create and deliver training, technology and tools.

While some of these changes can be introduced incrementally (see a future discussion on reuse adoption and maturity), the magnitude of the changes and the issues encountered are quite similar to those encountered when doing BPR. Many of the systematic methods used to design and implement BPR changes must also be applied to engineering reuse-based software organizations. Certainly the underlying ideas of organizational modeling, systematic change management, and socio-technical systems design are useful to implement a business' reuse program. This observation prompted my research group at HP Laboratories to define and pilot a BPR-inspired reuse organization design and reuse adoption process as part of our Flexible Software Factory research, and to employ some BPR-like techniques within our reuse program in HP's Corporate Engineering software initiative [Griss93]. As an example, we used the Software Engineering Institute (SEI) change management training course as an essential part of a workshop for reuse team leaders and champions.

Like many BPR efforts (50-70% of which fail due to lack of attention to the "soft factors"), most reuse adoption programs fail because too simplistic a solution is taken. Jacobson [Jacobson94] summarizes key factors and impediments facing BPR efforts, while Frakes and Isoda [Frakes94] discuss analogous barriers to large scale reuse. In both cases prescriptions focus on management support, incremental adoption, explicit process change, attention to soft factors, and pilot projects.

The process of becoming more systematic about reuse introduction has several levels of increasing rigor. These are related to the incremental adoption of reuse and corresponding expectations. Figure 1 illustrates the following steps towards achieving systematic reuse:

1. Reuse is desired, but nothing is done to make it happen. Ad hoc reuse is encouraged, object technology may be introduced, but no formal reuse program is visible.

2. An attempt is made to formally introduce reuse. A code or class library may be set up and publicized, but little attention is paid to handling the "soft factors" or to managing the organization through its adoption of reuse.

3. The importance of managing organization change is recognized. A reuse management team is given authority and encouragement to use incentives, begin training, make organizational changes and introduce new processes to support reuse.

4. Significant effort is devoted to use of systematic change management. Organizational change is planned and instituted. Some form of systematic reuse adoption process may be followed.

5. Full-scale BPR of the software development process and associated product development process is undertaken.

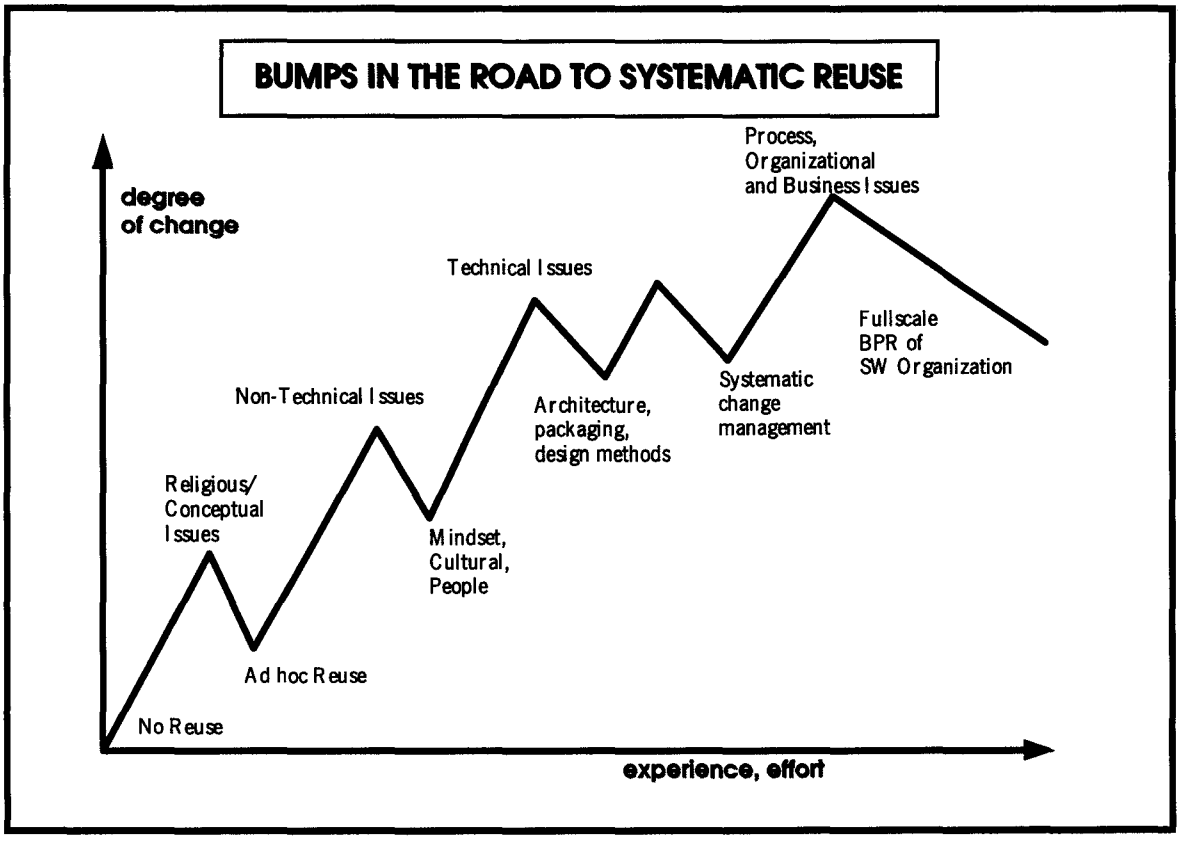

Figure 1.

4 Systematic OO Reuse: Changing OT

To obtain a systematic OO reuse process, we need to augment today's OT methods with a reuse-oriented process and guidelines to produce an effective OO reuse method. Several aspects of OT may have to be modified, or used very carefully (e.g., inheritance can be harmful, and is not always best for robust reuse [Lorenz93]). Most of today's OOA/OOD methods seem to be reuse-neutral or even reuse-hostile. Starting from an appropriate method, modifications and additions will be needed to better support domain engineering, product family design, and object model factoring and clustering. Specific process steps will be needed to support component and

framework creation, including explicit design and implementation guidelines, and access to catalogs of patterns, designs and mechanisms. Processes for application and system construction must include explicit access to an object repository during design and implementation, with specific reuse-oriented guidelines.

While several popular methods may be used as a starting point, my current favorite and base for my own experimentation is Objectory [Jacobson94]. I find Objectory's pervasive use-case driven design and traceability, object model factoring, subsystem extraction and packaging, development-case based process customization, and recent OO BPR approach as most consistent with my vision of successful systematic OO reuse. I will provide more details in a later column. The ESPRIT-funded REBOOT (Reuse Based on Object-Oriented Techniques) project illustrates the kind of changes and integration needed.

Work at HP on domain-specific kits [Griss93] suggests that we need to better understand how to use OT in combination with other technology to produce flexible, layered, composible, modular architectures if we wish to ensure high levels of manageable reuse. To handle the complexity inherent in subclassable frameworks with large numbers of classes and methods, we need to determine how and when to augment these frameworks with software-bus architectures, problem-oriented scripting languages, and system and component generators.

5 Conclusion

I have come to the end of my first column, and will continue my observations in the next column. I plan to address reuse processes, incremental adoption, reuse organizations, and reuse technologies. I will summarize recent or upcoming reuse events, and describe practical OO reuse experience.

I hope I have whet your appetite (or peaked your ire) sufficiently so that you might enjoy looking at the September 1994 IEEE Software special issue on systematic reuse [Frakes94] and my essay describing reuse experience at HP [Griss93]. I would also like to bring to your attention the upcoming Symposium on Software Reuse, to be held in conjunction with ICSE in Seattle, April 1995.

It is also my hope that these columns will not only bring to your attention the best knowledge and insights of OT and reuse experts, but will also stimulate discussion. Letters and email are welcome.

REFERENCES

[Frakes94] Bill Frakes and Sadahiro Isoda. Success factors of systematic reuse. IEEE Software, 11(5):15-19, September 1994.

[Griss93] Martin L. Griss. Software reuse: From library to factory. IBM Systems
Journal, 32(4):548-566, November 1993.

[Hooper91] James W. Hooper and Rowena Chester. Software Reuse - Guidelines and Methods. Plenum Press, New York, 1991.

[Jacobson94] Ivar Jacobson, Maria Ericsson, and Agneta Jacobson. The Object
Advantage - Business Process Reengineering with Object Technology. Addison-Wesley Publishing Company, 1994.

[Lorenz93] Mark Lorenz. Facilitating reuse using OO technology. American Programmer, pages 44-49, August 1993.

[Pittman93] Matthew Pittman. Lessons learned in managing object-oriented development. IEEE Software, 10(1):43-53, January 1993.

[Udell94] Jon Udell. Component software. BYTE, 19(5):46-55, May 1994.