Architecting for Large-Scale Systematic Component Reuse

Martin L. Griss
Software Technology Laboratories
HPL-98-132
July, 1998

E-mail: griss@hpl.hp.com

systematic reuse, architecture, process, organization, UML, OO, BPR

Organizations building highly complex business and technical systems need to architect families of systems and implement these with large-scale component reuse. Without carefully architecting the systems, components, organizations and processes for reuse, object reuse will not succeed. Experience with software reuse practice and adoption experience at HP and Ericsson led us to a systematic approach to component-based software engineering, based on object-oriented business and system modeling. This article explains how higher-level UML constructs support architected reuse, and describes a systematic process, leading from the business processes of an enterprise, through the system architecture for a family of applications that support these business processes, to the design and use of highly reuseably component systems.

Internal Accession Date Only

To be published in and presented at *Tools USA '98 – Technology of Object-Oriented Languages and Systems*, Santa Barbara, California, August 1998

Architecting for Large-Scale Systematic Component Reuse

Martin L. Griss
Laboratory Scientist
Hewlett-Packard Laboratories
Palo Alto, CA 94304-1126
griss@hpl.hp.com

Abstract

Organizations building highly complex business and technical systems need to architect families of systems and implement these with large-scale component reuse. Without carefully architecting the systems, components, organizations and processes for reuse, object reuse will not succeed. Experience with software reuse practice and adoption experience at HP and Ericsson led us to a systematic approach to component-based software engineering, based on object-oriented business and system modeling. This article explains how higher-level UML constructs support architected reuse, and describes a systematic process, leading from the business processes of an enterprise, through the system architecture for a family of applications that support these business processes, to the design and use of highly reusable component systems.

Keywords: systematic reuse, architecture, process, organization, UML, OO, BPR.

1. Software reuse is strategic

To meet competitive pressures and provide increased value to their customers, many organizations are radically reengineering their business processes. Part of this change is the timely and cost-effective creation of new mission-critical software systems. Dramatically improved software development is a key part of their strategy. Systematic software reuse promises tremendous improvements in time to market, quality and cost.

Simply pursuing "silver bullets," such as CASE, OO, RAD or Java, will not solve the software problem. While many managers adopt OO methods and technology to achieve reuse, experience shows that highly successful reuse programs address a variety of business, organization, process, architecture, cultural and technology issues. And those that do not, fail!

We need a systematic approach to obtaining significant software reuse, meeting business objectives, and overcoming the major organizational impediments to reuse. This article describes our experience at HP and an approach we developed jointly with Rational Software Corporation, described in more detail in the book *Software Reuse: Architecture, Process and Organization for Business Success*[1]. We use a comprehensive and systematic approach to orchestrate the large-scale investment and change needed to establish an effective reuse program. Large-scale reuse, a layered architecture, and a managed organizational change roadmap are of tremendous help in effectively dealing with mission critical business issues such as initiatives to rapidly decrease time to market.

2. Business need motivates reuse

Our direct experience and research at HP and Ericsson, and lessons learned from practical experience at other companies such as AT&T and Netron, shows that a reuse-oriented architecture, process and organization are key to effective reuse. Long-term success and management commitment to a systematic, architected, component-based software reuse program must be aligned with, and driven by, a compelling business reason, such as the critical need to decrease time to market to meet competitive market forces. Effective reuse programs depend strategically on a clear management and organization goal to develop a long term, flexible product line or product portfolio. With appropriate management commitment, evidenced by a strategic statement from the CEO to improve the speed with which new products are developed, change can begin. Energy and resource can be directed to establishing an appropriate reuse-oriented process. People can invest time in creating a software development organization specifically structured to support reuse. Managers and supervisors can then mount programs to address the well-known social and cultural impediments to reuse.

As an example of the issues typically confronting today's corporations developing software-intensive products for rapidly moving Web-enabled markets, the following is a summary of the problems facing several organizations at HP developing families of measurement systems. Similar issues confront anyone contemplating systematic reuse as a strategic or tactical response to business pressures:

- **Time:** Rapid product development and market agility are critical to the development of Web-enabled distributed measurement products. HP management has identified these factors as critical to success. How should this play out in terms of changed processes, new architectures, business measures and organizational structures?
- **Process:** Process changes and improvements will be needed to move from a feature-driven to a schedule-driven organization. What sort of standard processes and process maturity levels (such as SEI CMM level 2 or 3) should be selected? How do we get these into widespread use in the most expeditious way?
- **Organization:** A variety of cultural and organizational issues impede effective ways of working with architected platforms and components. Who owns standards, architectures, and platforms? What stops people from sharing? What encourages NIH? How do we systematically address these issues? What organizational, management and metrics changes are needed to encourage the change?
- **Coordination:** Coordination between new and ongoing technical and process improvement initiatives. What are the connections between new technologies and strategies for development for the Web, improving process predictability, improving quality and decreasing cycle time? What are the priorities? What should we do first? What architectural implications and other standards need to multiple initiatives? Who owns the architecture?
- **Technology:** Common architectures, reusable components and leveragable platforms are key elements to achieving decreased cycle time. What notations, standards and technologies should be used? How should they be introduced? How do standard tools and platforms apply?

3. Success factors for systematic reuse

While this article can not address all of these questions, our work offers a framework for approaching these kind of issues. Architecture, reuse, standards, process, organization and business issues are closely related, and must be addressed together within a common model.

A successful, large-scale reuse effort must be:

1. **Business-driven**
   - The organization must have a visible, articulated and compelling need for dramatic improvements in cycle time, cost, productivity and/or agility.
   - The organization must be producing software (applications, embedded systems or key subsystems) that form an obvious product-line, application family or coherent "domain."
   - These two factors provide a context for organizational commitment in which some form of reuse can be justified economically, technically and strategically. Component-based development becomes of strategic importance. Management will pay attention.

2. **Architected**
   - Applications, systems and components must be purposefully designed and structured to ensure that they fit together, and that they will cover the needs of the family or domain.
   - A well defined, layered, modular structure, and various design and implementation standards are needed. Object-oriented modeling and implementation are extremely useful vehicles.
   - Ad hoc reuse of existing software, or use of common frameworks, do not ensure the coherence of a set of compatible, reusable, maintainable parts. A system architecture for reuse must be defined and maintained as an organizational asset.

3. **Process-oriented**
   - Distinct software development and maintenance processes for architecture, components and applications must be defined and followed.
   - These processes must explicitly incorporate reuse. They systematically identify and express commonality and variability, and include standards for designing, packaging, managing and using components for reuse.
   - Processes will evolve as the situation changes and experience and adoption grows. This will need careful management to standardize a set of reusable process elements that can be flexibly combined and customized.

4. **Organized**
   - Long-term management leadership and commitment will ensure that the organization is structured, trained and staffed to follow the processes and conform to standards.
   - The organization will have several distinct subgroups responsible for creating, reusing and supporting reusable components. These teams must be trained and willing to follow the specific processes.
   - The organization must be ready for the requisite level of change, and be willing to "stay the course."

- People must have well defined reuse-oriented jobs, with appropriate training, skills and rewards for effective reuse performance.

4. Engineering the solution systematically

As we came to understand the magnitude of the changes needed to install pervasive organizational reuse, it became clear that we should adopt a Business Process Reengineering (BPR) approach to restructuring a software engineering organization for large-scale reuse[2]. A standard architectural design notation and a compendium of common, reusable architectures and designs provide a base for more reusable components and frameworks. Used consistently within a reuse-oriented process, architected components and frameworks enable high levels of reuse. And high levels of reuse can lead to dramatic reductions in product development time. As we looked at how an OO software modeling method should be used for the architecture, we also looked for compatible ways to model processes and organizations. This motivated us to build upon a powerful, well-known OO method that could deal not only with the modeling of architecture and components, but also with software process and organization design.

This led to a collaboration with Ivar Jacobson of Rational Software Corporation with the goal of integrating reuse process and pragmatics into an OO development method[3]. We have worked together since 1994 to develop a coherent approach to structuring architecture, process and organization for effective object-oriented and component-oriented reuse[2,3,4]. We have integrated techniques of systematic software reuse into Jacobson's Object-Oriented Software Engineering process (OOSE)[5], building on the new Unified Modeling Language (UML) for object-oriented systems[6]. Since the goal of reuse is pervasive, we call our systematic approach to practical, large-scale reuse the *"Reuse-driven Software Engineering Business"* (RSEB). An RSEB is a software development organization specifically structured to employ systematic reuse-based software engineering and management techniques as a key business strategy.

We based our collaborative work on Jacobson's use-case-driven architecture and process modeling frameworks — Object-Oriented System Engineering (OOSE)[5] and Object-Oriented Business Engineering, described in "The Object Advantage"[7]. By so doing, we have integrated the key aspects of successful reuse programs into a coherent model. Our approach relates all processes (activities and supporting infrastructure) and products (architecture and components) to the business goals of a reuse-driven software engineering business.

Our systematic approach has the following elements:

1. OO model-driven component and application development

- We use the Unified Modeling Notation (UML 1.0), to model components and system.
- We build upon a UML-based OOSE architectural style and method. OOSE defines the constructs, models and steps which are needed to transform requirements into models and ultimately software. (OOSE is the base method for the Rational Objectory process)
- We have added UML-based extensions to model layered architectures, reusable components, component interfaces, variability and frameworks.
- We use incremental, iterative software development processes, which allows us to build up the architecture, reusable components and applications in stages.

2. OO Business Engineering to model reuse process and organization

- We use OO models, called business models, to precisely define the details of the software processes, and how they interact.
- We use these models to also define the roles of various workers in the software organization, and how they can be effectively grouped and managed.
- Finally, these models can be used to define the workflow (or process scripts) that guide these workers.

3. OO Business Engineering as transition framework

- This provides a standard set of steps and questions that identify the scope of the transition, and determine the best organization structure to match the needs, level of commitment and existing situation.
- A specific organization change roadmap is designed, involving an assessment of scope, readiness for change and incorporates a variety of change management pragmatics, such as reuse pilots and staged changes.

5. Integrating architecture, process and organization in the RSEB

The RSEB is run as a software engineering business. This means that the software engineering goals are key to accomplishing the organization's business goals, and that as a consequence, the software organization itself is operated as a business, with well-defined customer and financial objectives. In many organizations, a dominant and compelling business goal is to reduce product development times, yet retain market agility within and across product families. Examples of these families are applications and systems which are built to meet different customer and country needs by combining and customizing reusable components: measurement systems built from common instrument components, and Telecom switches (such as Ericsson' AXE) which are configured to a variety of situations.

If deemed technically feasible, a strategic decision must be made to establish one or more reuse-driven business units within these organizations. As a reuse-driven software organization, such business units produce multiple, related applications, centered and optimized around the production and reuse of components. These applications commonly form a product-line or product family. As a business, this organization must understand its customers, and serve their needs, while at the same time effectively achieving its profit and expense objectives. Business tradeoffs must be managed using well defined economic, product and process measures.

Figure 1 summarizes the key elements of the RSEB, using an informal OO business engineering notation. The ellipses are business use cases, representing software engineering processes, while the rectangles are systems, represented by sets of UML models. The stick figures are actors, representing people or organizations with which the RSEB interacts.

Each system in Figure 1 is expressed as a set of OO models using OOSE and UML notation. Reusable components are public elements drawn from the various models, designed and packaged for reuse, including use cases, patterns, analysis types and design and implementation classes (code). Components are not used alone, but in groups called *Component Systems.* Component systems are constructed by the *Component System Engineering* process. In additon to the public reusable components, component systems include the various supporting models and code to implement the components. Applications are constructed by *the Application System Engineering* process as a set of connected models and software, called an *Application System.*

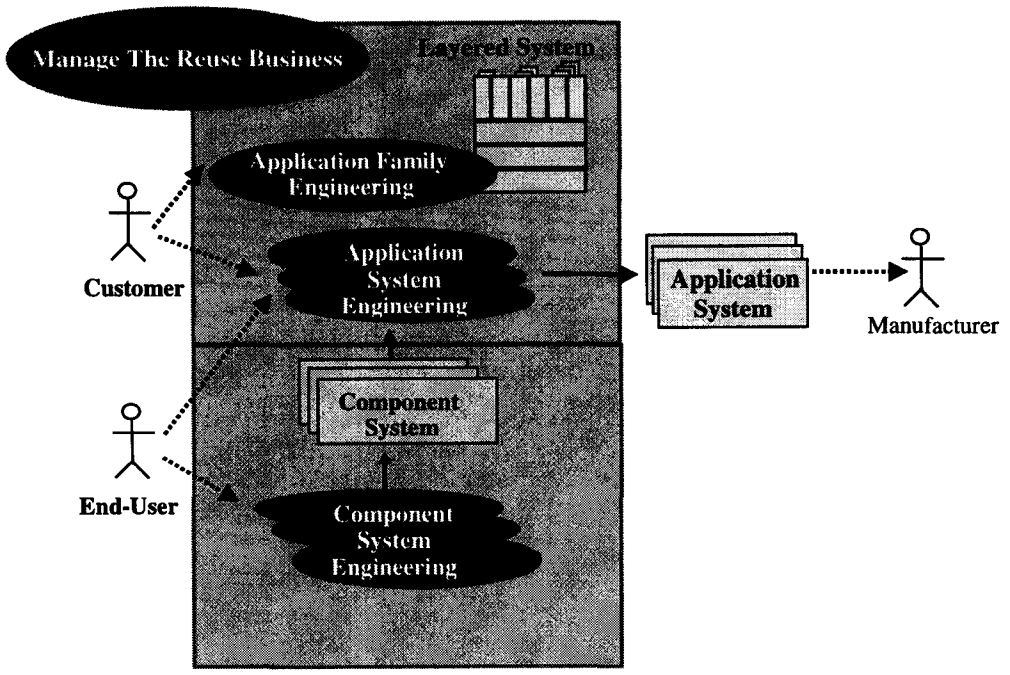

Figure 1. The elements of the Reuse Business. The ellipses represent processes, the rectangles represent systems, and the stick figures represent the environment within which the Reuse Business functions.

Most important to shaping the applications, and ensuring high levels of reuse is the architecture of the reusable components and applications built from these components. Applications and components interact with each other in a layered, modular architecture. We visualize the *Layered System* architecture as illustrated in Figure 2. This layered system is developed by an *Application Family Engineering* process. The RSEB systematically engineers a layer of related application systems, at the top. Behind the front row of applications are related variants of the applications. The application-specific software is engineered in the Application System Engineering process by extensively reusing component systems, which we imagine as existing on several lower layers. Below the application layer are components reusable only for the specific business or application domain area, such as banking systems or microwave instruments. This layer includes components usable in more than a single application. The third layer of cross-business middleware components includes common software and interfaces to other established entities, such as graphical user interfaces, ORB's, etc. The lowest layer of system software components includes interfaces to hardware, such as an operating system interfacing to the computer (or to its built-in instruction set).

Application systems are not free to reuse any UML element within a component system, but only those made publicly available through carefully defined and documented *facades*. Components and component systems are specializable in various ways, and explicitly indicate where they can be specialized through a UML extension called *Variation Points*. Figure 3 shows an application system (**Payment**) importing components from a component system (**Account Management**) via the façade (**Transfers**). Figure 3 also shows several variation points (black dot) and several attached variants.

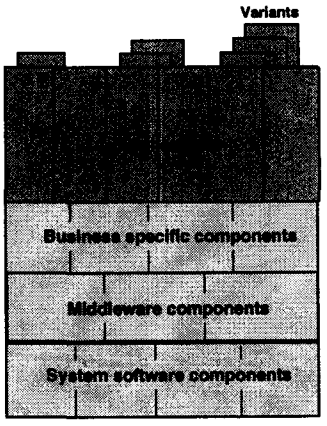

Figure 2. The basic layered system architecture consists of three layers of components that go into the application systems at the top of the diagram.

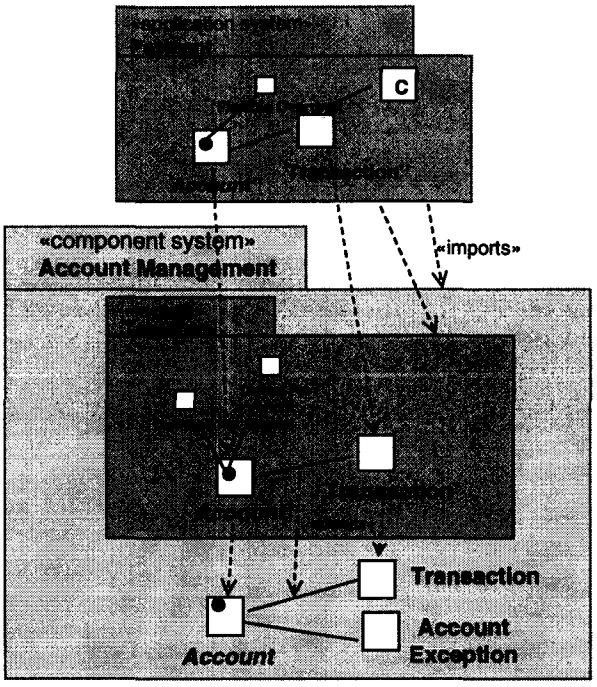

Figure 3. An application system importing objects from a component system, via the façade. Some components have variation points; some of these come with supplied variants

Business use-cases are used to model the interaction of an actor (person or external organization) with the business organization[7]. These business use-case models help identify

core processes within the organization. When these models are mapped onto business object models, they display business workers as objects, and the entity objects they manipulate. They can be used to define the responsibilities of the workers, the workflows they enact, and the information systems and tools they use.

6. Incremental transition

The systematic transition of an existing software organization into an RSEB is based on an OO Business Engineering framework[7]. We execute Business Process Reengineering (BPR) incrementally, using explicit business models expressed in UML notation. Business use cases model the interaction of an actor (person or external organization) with the subject organization. These business use case and object models help identify core organizational processes. These models are further refined to produce a robust business object model showing workers, and entity objects. We then map these onto responsibilities, workflows and information systems and tools. Factoring and subsystems model appropriate organizational structure.

The OO Business Engineering approach has several stages that prepare, define and execute a transition:

- **Directive:** Management issues a directive to reengineer, articulates scope and commitment, and empowers a reengineering team.
- **Envision**: The reengineering team envisions the new organization, and establishes what sort of RSEB it might be.
- **Reverse Engineer**: The team uses reverse engineering to determine essential aspects of the existing organization and its processes.
- **Forward Engineering**: Forward engineering is employed to develop a detailed model by customizing and extending a generic RSEB process and organizational model.
- **Install**: Finally, several teams work together to install the new processes and organization

Each of these steps includes specific organizational change management guidelines, such as the use of champions and pilots, metrics, and some reuse pragmatics, such as the use of incremental, pilot driven reuse adoption, and distinct reuse-maturity stages[2,3]. To develop the appropriate roadmap, it is important to assess both the clarity and urgency of the business and domain drivers, as well as the organization and process maturity. Standard assessment and maturity model templates are provided.

7. Conclusion and Next Steps

The RSEB approach provides a systematic framework for the definition and transition to a reuse-driven organization. Clear business goals and management commitment are key to making the process and organization changes needed for architected reuse. UML as a common modeling language for OO system and business modeling, provides a widely accepted software "blueprinting language" for architects and engineers to describe complex systems and precisely define their reusable components. Large-scale reuse and an organization change roadmap are critical for systematically improving software and meeting business needs.

We are now developing a more comprehensive approach to integrating explicit traditional domain engineering methods into the RSEB[8]. Starting from a business model, today's RSEB includes a use-case driven analysis of commonality and variability for an application family, carried out in Application Family Engineering and Component Systems Engineering.

Variability is expressed in terms of variation points and variants. Others kinds of features that characterize alternative applications are mentioned, but not modeled explicitly. Our new work integrates elements of the well-known FODA[9] method, adding an explicit feature model and new domain analysis steps, based on exemplers. Some features directly represent use cases, while others deal with performance alternatives or implementation alternatives. Some features are mandatory, while others are optional or correspond to variation points. This should make RSEB approach applicable to an even broader class of problems, in which partially incomplete knowledge of the application family (the "domain") can be handled systematically. We are also exploring the use of tools that more flexibly represent components, patterns, and features, with explicit, traceable linkages between the various models.

8. References

[1] I Jacobson, M Griss, P Jonsson, *Software Reuse: Architecture, Process and Organization for Business Success*, Addison-Wesley-Longman, May 1997.

[2] ML Griss, Software reuse - a process of getting organized, *Object Magazine,* May 1995. (See http://www.hpl.hp.com/reuse for other columns and references.)

[3] ML Griss, Systematic OO software reuse - a year of progress, *Object Magazine,* February 1996.

[4] I Jacobson, Succeeding with Objects: Reuse in Reality, *Object Magazine,* July 1996.

[5] I Jacobson et al, *Object-Oriented Software Engineering: A Usecase-driven Approach*, Addison-Wesley, 1992.

[6] G Booch, I Jacobson and J Rumbaugh, "The Unified Modeling Language," Technical Report, Version 1.0, Rational Software Corporation, January 1997. (See http://www.rational.com/ot/uml).

[7] I Jacobson, M Ericsson and A Jacobson, *The Object Advantage: Business Process Reengineering with Object Technology*, Addison-Wesley 1994.

[8] M Griss, J Favaro and M d'Allesandro, *Integrating Feature Modelling with the RSEB,* Proc. 5th International Conference on Software Reuse, Victoria, BC, IEEE, June, 1998.

[9] K Kang et al, *Feature-Oriented Domain Analysis Feasibility Study*, SEI Technical Report CMU/SEI-90-TR-21, November 1990.

9. Biography

Martin L. Griss is a Principal Laboratory Scientist at Hewlett-Packard Laboratories, Palo Alto, California, where for the last 15 years he has researched software engineering processes and systems, systematic software reuse, and object-oriented development. He created and led the first HP corporate reuse program. He led HP efforts to standardize UML for the OMG. He was previously director of the Software Technology Laboratory at HP Laboratories, and an Associate Professor of Computer Science at the University of Utah. He has over 25 years of experience in software engineering, is co-author of the best-selling book "Software Reuse: Architecture, Process and Organization for Business Success," writes a column for the "Object Magazine", has written over 40 articles and lectures widely on systematic reuse and software process improvement. He received a Ph.D. (Physics) from the University of Illinois.