Software Reuse: Nemesis or Nirvana?

Panel Session

Steven Fraser, Nortel (Chair); Maggie Davis, Boeing; Martin Griss, HP Labs; Luke Hohmann, SmartPatents; Ian Hopper, Nortel; Rebecca Joos, Motorola; Bill Opdyke, Lucent Technologies

Context

As we reach the end of the millennium, the concept of software reuse and the practice of software engineering has existed for almost thirty years. Both concepts were introduced into the literature in the late sixties. To set the context for this panel, Nirvana is taken to mean a state of great happiness, while Nemesis is understood to be a state of great frustration. The panelists will share their experiences and perspectives with the audience in a dialogue initiated by the following questions and positions:

- What are the most challenging cultural barriers to software reuse?
- What are the *best-in-class* successes achieved with software reuse?
- How is software reuse success recognized and sustained?

Perhaps the key determination of success for software reuse is how it can — as a combination of cultural and technology practices — deliver customer valued systems in "webtime"? What are the challenges of: education; team communication; design technology; economics; and leadership? What is the role of: the customer; the manager; the designer; and the corporate executive?

Steven Fraser
Panel Chair
Nortel, Santa Clara, CA, USA.
sdfraser@nortel.com

Steven Fraser is manager of the Software Design Process Engineering team at Nortel's Meridian Systems Headquarters in Santa Clara, California. He is also Chair of the Nortel Design Forum, an annual three day proprietary interactive video conference with more than 25 participating sites world-wide. Previously, he spent four years at Nortel's Computing Research Laboratory in Ottawa, Canada. In 1994 he was a Visiting Scientist at the Software Engineering Institute (SEI) collaborating with the Application of Software Models project on the development of team-based domain analysis techniques. Since joining Nortel in 1987, Fraser has contributed to the development of OO-based CASE-Design Tools and to software development processes. Fraser completed his doctoral studies at McGill University in Electrical Engineering. He holds a Master's degree from Queen's University at Kingston in applied Physics and a Bachelor's degree from McGill University in Physics and Computer Science. He is an avid operatunist and photographer.

Maggie Davis
Boeing, Seattle, WA, USA.
maggie.davis@pss.boeing.com

Maggie Davis will discuss what barriers she has encountered working first in the large defense contractor culture of Boeing where products are extremely large, long-lived systems and now in the information systems culture supporting manufacturing and assembly of complex airplanes.

Davis is now a Computing Systems Architect using object and domain modeling techniques to support development of a factory computing system architecture for Boeing Commercial Airplane Group. She was Principal Investigator (1995-1997) of the Boeing internal research and development project titled *Enabling Technology for Product Reuse,* which concentrated on architecture frameworks as a motivating factor in systematic reuse adoption. Davis was Reuse Technology Area Lead for the Boeing STARS program from 1988-1995, participating in the joint development of the STARS Conceptual Framework for Reuse Processes (CFRP) and the Reuse Strategy Model (RSM).

Martin L. Griss
HP Laboratories, Palo Alto, CA, USA.
griss@hpl.hp.com

Technology is Not Enough - Cultures in Conflict

Objects are believed to be crucial to achieving the long-sought after goal of widespread reuse. Unfortunately, many people naively equate reuse with objects, libraries and other tools, expecting reuse to *automatically* emerge, but often do not get much reuse. Based on our experience with reuse at HP, and with our many customers, we know that without extensive process and organizational changes to support systematic reuse, objects will not succeed in giving users reuse.

In almost all cases of successful reuse, architecture, a dedicated component development and support group, management support, and a stable domain were the keys to success. These largely non-technical issues seem more important to successful reuse than the specific language or design method chosen. As Simos has pointed out, architecting for reuse is a social process, involving numerous stakeholders with disparate personal, political and technical agendas. Teams and individuals may choose to work with each other, or avoid collaboration and sharing based on trust, prior history, credibility or perceived importance of role. Management must employ systematic organizational design techniques and even business-process re-engineering techniques to ensure a successful transition.

Martin Griss is a senior 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, object-oriented reuse, and measurement system kits. He has had a defining role as senior reuse consultant within HP's Professional Services Organization. As HP's *reuse rabbi*, he led research on software reuse process, tools, and software factories; the creation of an HP Corporate Reuse program; and the systematic introduction of software reuse into HP's divisions. He was director of the Software Technology Laboratory at Hewlett-Packard Laboratories, and has over 25 years of experience in software engineering research. He was previously an associate professor of computer science at the University of Utah, where he is currently an adjunct professor. He is a member of the SIGSOFT executive committee, and the UML 1.1 semantics and revision taskforces. He has authored numerous papers and reports on software engineering and reuse, writes a reuse column for Object Magazine, and is active on several reuse program committees.

Luke Hohmann
SmartPatents, Mountainview, CA, USA
lhohmann@acm.org

In this position statement I'd like to talk about a fundamentally different approach to reuse than my fellow panelists. Instead of talking about reuse from the perspective of components, objects, subsystems, or the like, I'd like to talk about the reuse of knowledge. (Where does reuse start? Humans are reuse machines. We work at finding a solution to a problem *what is the fastest way to drive to work?*). Once found, we *reuse* this solution as much as possible (*we drive to work the same way every day*). When the *reusable* solution fails us, we try something new. Through this process we *learn* something. We learn what didn't work (don't reuse that) and we learn what did (remember — reuse — that).

Where does software reuse start? Software development is a special kind of problem solving, one that involves creating software systems to meet some number of defined needs. We know from empirical evidence that what makes an *expert* developer different than the *novice* is that the expert has amassed, inside their head, a rich cognitive library of reusable knowledge. It is this knowledge, traditionally acquired through years of hard work, that enables the expert to be an expert in problem solving. But we also know that expert-level knowledge is often tightly coupled to a specific problem domain, such as real-time embedded system programming or the design of large relational databases.

Note that this kind of *reuse* is very different from the kind of reuse most commonly referenced in the literature. I'm not talking about reusing an object, an OCX, a subsystem, or a framework. Instead, I'm talking about the kind of reuse that is more commonly referred to as *experience.*

How can we reuse expert knowledge? A central question in the reuse literature is how we can bring *forth* expert-level knowledge in such a way that it can be shared with novices in order to improve their performance quickly. This is the process of separating the *cost* (or pain) of creating expert-level analysis or design knowledge with the cost of (re)using it. While many things have been tried over the years, the current patterns movement has proven beneficial as a means of sharing expert knowledge in an efficient, cost-effective means. Patterns exist for general design issues (such as the *Builder* pattern, which guides us in the creation of complex run-time object structures), to teaching a complex subject in a classroom. Finally, we are starting to see more and more pattern languages, or sets of inter-related patterns that deal with solving problems in a well-defined problem domain.

What is the reuse from patterns? Although many patterns and pattern languages move far beyond mere *reuse* in their sharing of expert knowledge, the practical software developer in me focuses primarily on how they directly impact my and my teams' ability to write great software. Viewed in this light, patterns represent a form of reuse in which the (re)user engages in the intelligent customization of expert-level knowledge to meet the specific demands of their problem. Pattern reuse is not the same kind of reuse that one achieves from using remove_if() from the standard C++ library, nor is it the kind of reuse that one achieves when they purchase an ActiveX component. Both of these forms of reuse are powerful in their own right, but they are qualitatively different than pattern-based reuse. Instead, pattern reuse is a deeper kind of reuse, one that also enables both the novice and the expert to do more than simply *reuse* a component or a subsystem, for in *reusing* patterns, we learn as humans.

Luke Hohmann is the Vice-President of Engineering at SmartPatents, Inc., the world-wide leader in the development of analytical software systems that enables companies to automate the process of analyzing, protecting, and maximizing the value of intellectual property assets. Mr. Hohmann has extensive experience in object-oriented analysis and design, software engineering, user interface design, and project management. He is the author of *Journey of the Software Professional: A Sociology of Software Development* (Prentice Hall). Mr. Hohmann is currently working on two books, *GUIs with Glue: Creating Usability Through Lo-Fi Design* and *Pattern Vignettes: Using Design Patterns in the Real World.* He has authored numerous papers on the sociology of software development.

Ian Hopper
Nortel, Santa Clara, USA.
ihopper@nortel.ca

Carry-Over Reuse: *Hoisting*

Reuse appears to be in the realm of the real world; neither nemesis nor nirvana. Our solutions are very much in the *rubber bullet* category: insufficient to solve the biggest problems, but useful in some cases. Software engineering research has resolved most of the smaller issues and is moving on to the larger issues, which involve soft-sciences like economics and psychology. Progress on programming in the large will be slow.

As we explore new avenues and technologies for higher productivity, we tend to forget the basics. Old (existing) software is frequently a good-enough basis for the development of new software. In this discussion, I will review cultural and economic factors that may lead away from software reuse and suggest a practical middle course. Old software is difficult to sustain over time due to incremental development encroaching on architecture limitations, challenges in maintaining a high-productivity development environment and difficulties retaining staff. However, replacing the software of an existing product often forces double development of features until the new software completely covers all existing capabilities. Finally, new software for a new product is higher risk due to vague requirements and resulting unpredictable schedules. The new product may be awkward in the marketplace due to functionality overlap and integration expectations.

We have had success with software evolution using a pattern that I call *hoisting*. It is practical to put a new series of layers below an entire software structure. The classic and lowest risk variant is to port to a more sophisticated operating system, which allows new development to exploit the new platform and to be marketed easily as new features for a proven product. In the more general case, the new layers creatively interpret the formerly low-level operations to allow new capabilities to be controlled by the older software.

Ian Hopper is the Software Architect for the Nortel Meridian 1 PBX product in Santa Clara, California. He joined the Nortel, Ottawa lab in 1983 as a software designer on multi-media and data communications systems. He worked on early voice-data integration and twisted pair LAN technology (802.9). Hopper has been exploring and applying the object-oriented computation model since 1981. He is an Honors B.Math./CS graduate of the University of Waterloo. His weekends are often spent wind surfing on San Francisco bay.

Rebecca Joos
Motorola, Arlington Heights, IL, USA.
joos@cig.mot.com

When is Too Much Enough?
and
When is It Too Little?

The first hurdle of introducing software reuse is the *selling and support* phase. The success of this phase depends upon getting the right kind of support from the right level of management. Once that is accomplished the next phase, or hurdle, is *selling* the troops i.e., the engineers. Joos will address some of the issues (a what has worked and what hasn't approach) of getting reuse into the engineering culture.

Rebecca Joos is a Principal Staff Engineer at the Cellular Infrastructure Group looking into new and better methods, techniques, and tools to more accurately and efficiently develop software in the switching division. Her concentration on software reuse has been extended to basic software engineering and quality issues. She has been instrumental in introducing and supporting process (SEI capability matrix) and promoting reuse as a productivity and quality enhancer. Her immediate concern is how this technology can be made a cultural part of the engineering environment.

William F. Opdyke
Lucent Technologies/Bell Labs,
Naperville, IL, USA.
wopdyke@lucent.com

Nemesis? Nirvana? Neither! Neglected! Needed!

Let's consider what the terms nemesis and nirvana really mean (based on their definitions in the *New American Heritage Dictionary*) and whether either term comes close to characterizing software reuse. Then, let's reflect on the current state - and speculate about the future - of software reuse.

Is software reuse a nemesis - something that inflicts relentless destruction on organizations that attempt a reuse program? No - at least not relentlessly. Organizations have pursued reuse efforts for many reasons and in many different ways. If the organizational culture is supportive of reuse (as per the areas well enumerated by Griss in his position statement), software reuse can play a supportive (rather than destructive) role in enabling an organization to achieve or at least come closer to achieving its cost, quality and schedule targets for application developments. If the culture doesn't support reuse, most organizations will eventually abandon their reuse program and thus the pain won't be relentless.

Is software reuse nirvana - providing blissful freedom from pain and care? No, at least not for long. Software applications must function in the *real world*, where change is rapid along several dimensions, including technologies, user needs, external and component interfaces, and designers' understanding of the family of applications they are supporting. I have yet to meet a software developer who could credibly claim omniscience; hence, software that is intended to be reusable must be able to adapt to unforeseen changes. A culture supporting reuse will support the adaptation of existing components in a way that minimizes - but doesn't eliminate - the pain.

Geoffrey Moore, in his book *Crossing The Chasm*, notes that many ideas and products are embraced by the *innovators* and *early adopters* (the folks who embrace new technologies, have visions of new paradigms, etc.), but ultimately fail because they are never embraced by the mainstream - the *early and late majorities*. That audience cares about reliability, technological maturity, cost and customer support. They also want to see how others, with problems similar to theirs, have already successfully applied whatever new idea or product is being marketed to them. For decades, the champions of software reuse have been unsuccessful in delivering solutions that meet the needs of this mainstream audience - and, hence, software reuse has been neglected by the mainstream.

Software reuse is now being viewed afresh in industries such as telecommunications, where competition for both customers and technical talent is fierce, and where cost pressures are relentless and increasing. The mainstream is taking more notice of software reuse. Technologies such as object-oriented technology (with its focus on reusable components and cross-application frameworks) and domain analysis (with its focus on supporting application families) are among the tools being successfully applied to reduce costs across related applications. Some of the critical, non-technical issues are being addressed in organizational cultures that are embracing systematic reuse. Success stories are beginning to appear (while others are kept secret, to retain a competitive advantage) - and I predict that we will see more success stories in the future.

Bill Opdyke has conducted several software reuse and platform related projects at Lucent Technologies/Bell Labs, including cross-application design and advanced development, consulting with product development teams on platform technologies and vendor component selection, publishing a newsletter that focuses on experience reports, technologies and other issues related to reuse. Opdyke's research at the University of Illinois (which Ralph Johnson supervised) focused on refactoring object-oriented frameworks to support evolution and reuse. Opdyke's most recent research related to organization issues and reuse was presented at WISR '97 (the International Workshop on Software Reuse) and PLoP '97 (Pattern Languages conference).