Tim Mikkelsen
April 20, 1999

Developing and organizing for modular / reusable software and firmware

Executive Summary

There are two questions that this paper addresses: How do you develop modular / reusable software and firmware? And then, how do you organize an R&D organization so that it can naturally reuse the code? The first step is not to directly answer the questions, but to step back and understand the business needs of the organization. This is necessary to ensure validity, scope and scale of the reuse. In broad terms, reuse is not a technology issue, but a people and process issue. Effectively getting to reuse involves identifying the need and goals and then incrementally moving to these target reuse goals. Engineers, generally, are not pre-disposed to reuse code - they like to build new things. So, 'naturally reusing code' is a bit of an oxymoron. It is possible to make it more normal, but it requires careful management attention that involves direction, metrics, trust and rewards.

Business needs

Large scale or pervasive reuse is very useful and can be extremely cost effective. It is not, however, free or cheap. Before an organization heads off to build products with heavy reuse aspects, the organization needs to determine the business goals and how reuse addresses them. Is there enough commonality, stability, share-ability across the products or product families to make it economically viable. For example, if there is an investigation for a new product line with only limited chance of success, it probably doesn't make sense to gear up a major reuse effort until the product has a reasonable chance of success and successors. The key point is that reuse is a technical tactic to achieve business strategies. Reuse is not an 'end', but a 'means to an end'.

In spite of this 'stepping back' caution, reuse can be tremendously effective. Some of the key benefits of reuse include:

- Cycle time
- Product quality
- Common look and feel, usability
- Supportability
- Minimizing maintenance costs

There are many examples of these benefits in the literature. One of the more common benefits is the improved cycle time (although note that there is an initial set-up cost involved). A specific example is the LDS reuse library in HP/TMO. It was used in over 13,000 shipped products in 1998 resulting in $191 million of revenue. Although the reuse part can't take credit for the full amount, there is real value. Specifically, the product team for one of the products (MID's Falcon - an Economy Signal Generator) indicated that the LDS reuse library enabled them to get to market one year earlier than they would have been able to without LDS. For Falcon, last year's revenue was $51 million. This is a substantial return on investment for the 3-4 engineers associated with LDS.

How to develop reusable software

So, given that reuse is a good and appropriate thing to do, how do you do it. As stated earlier, the problem is not, primarily a technical issue. It is a people and process and organization issue. There is large overlap between how to develop reusable software and how to organize for reuse. (So, there is some overlap between this section and the later section on organization aspects.) However, the basic idea is very simple:

When you build a new product, use previously developed pieces

Even though the idea is simple (uh like … duh…), achieving the result can get to be complex. The key aspects are:

- Train and encourage people to reuse
- As appropriate, build artifacts that can be reused
- Put the reuse artifacts someplace accessible
- Let developers know about the reuse artifacts
- Use the reuse artifacts
- Maintain and support the reuse artifacts

This is all great, but where do you start? Most organizations don't have a full suite of reuse artifacts just waiting for the developers. And further, it is not a good idea to jump into a full-scale reuse program. The best approach is to proceed incrementally. In a lot of ways, this is very much like the SEI Software Capability Maturity Model (CMM) - organizations will start low on reuse maturity, scope and scale and should progress over time to a level appropriate for their business needs.

It is usually disastrous to build reuse components and platforms in a vacuum. The results will tend to be overly ornate and not particularly useful on the intended products. When starting out, it is important to build something in association with an actual product. Within this context, the initial efforts should be in the form of a pilot project. The focus should be fairly narrow, clearly defined with a small number of people involved in a fairly short period of time. The base of the reuse program is built out of a series of reuse pilots.

The intention is, over time, to build up to a minimum viable platform. But as this happens it should be done with a small team on the platform or architecture. It is also important to keep an eye on the development so that it doesn't grow out of control (trying to be all things for all people) or to be perfect (gold plated). Getting something perfect or all-inclusive is very expensive. A natural tendency with reuse work products is to make them 'better'. I feel it is very important to keep the engineering '80/20 rule' in mind - do a good job on the key (80%) part of the reuse functionality, but don't spend all the additional effort for the last (lower priority) aspects. You shouldn't expect reuse work products to perfect, but what you want is for them to be effective.

The Scale and Scope of Reuse

One question that arises from the idea of building up to a platform or architecture, is what is the intended scale and scope of the reuse. This goes back to the basic goals or business needs. But within the business framework, there is a range of outcomes for reuse. It might be appropriate to do limited customization, semi-formal module sharing, or a full-blown reuse program with management and infrastructure. I believe this needs to be considered within the context of the business strategy and return on investment. Martin Griss has a good model to think about the level of formality associated with reuse that he calls a Reuse Maturity Model (derived from some of the SEI/CMM concepts):

Level	description
No reuse	some code sharing, but informal
Informal code salvaging	chunks are copied/adapted, maintenance becomes the problem
Planned black-box code reuse	conscious choices about chunks, some management
Managed work product reuse	explicit management of reuse artifacts and reusers
Architected reuse	higher level of reuse, formal architecture and planning
Pervasive domain-specific reuse	products, components, arch., processes planned/optimized

The idea is to figure out where the organization is currently operating and where the business strategies are driving the organization to be. From there, the organization can start to move through the various phases of more formal software reuse with a clear goal in mind. (This amounts to doing some 'organizational design' with respect to reuse.)

With respect to scope of reuse, it can be very broad. However, it is pretty dangerous to start out to broadly. Keeping with the pilot mentality, an organization should start with a narrow slice of the products and build from there. The other thing to consider about this is that a multi-division reuse program is very difficult to manage because of the differing cultures, strategies and products. It can be done, but it requires additional effort.

Technology

As previously stated, reuse is not particularly a technology issue. However, technology can help. In particular, it is worth making sure that there is reasonable reuse support in the implementation language and its features and the product development environment and tools used in the organization. Obviously, an object-oriented language should be easier for the developers to use for reuse than assembly language. However, keep in mind that OO is not the same as reuse. In particular, it has been pointed out that some OO features and environments are actually hostile to reuse. If there are issues in the technology base, think explicitly about them and build support and infrastructure around issues to make it as is appropriate as possible.

What to reuse

Given that some level of reuse is appropriate, it is worth keeping a somewhat broader perspective as things get started. Many different work products can be reused. In other words - don't just think about code! For example, a development organization should think about the reusability of:

- Software modules/components
- Architecture
- Application templates
- Development tools
- Testing tools
- Process tools
- Documentation modules/components
- Help modules/components
- Process/lifecycle templates

The other aspect often over-looked, is that there are usually commercially available reuse artifacts - i.e. Commercial, Off-The-Shelf (COTS) work products. Obviously, people purchase commercial tools (like compilers and debuggers) and some underlying components (like operating systems). But, there are a wide variety of third party suppliers of components and tools that are worth investigating. This 'build versus buy' orientation to build locally is a traditional HP problem. Note that looking for reuse work products should be somebody's job - otherwise it won't happen. Note also that the real phrase should be 'build versus acquire' - there are lots of work products that could be 'mined' from other HP organizations.

How to organize R&D for reuse

There are a variety of challenges for reuse, from an organizational point of view:

- Engineers like to build new things
- Reuse artifacts are often poorly thought of
- Organizational change is hard
- Organizations tend to under-invest in infrastructure (reuse is an example of infrastructure)
- Organizations have changes in direction, products and strategy

Engineers tend to like to build things themselves. This makes 'natural reuse' a real challenge because it involves behavior change. The key is to encourage engineers to reuse. This involves:

- Communicating the need, vision, direction for the reuse
- Training on reuse technologies and methodologies
- Communicate the existence of the reuse artifacts
- Develop and track metrics on reuse (you get what you measure)
- Reward reuse
- Be consistent over time and across managers
- Give engineers some role or say in the reuse artifacts (review, processes, features, priority, etc.)

Often times, the reuse artifacts are poorly thought of by the developers. This is partially due to the engineer tendency to build something new. The key to address this is to actually build good, useful, high quality work products. This involves:

- Have the right person or people to architect the work products
- Have the right people to build it
- Maintain very high quality standards (within the context of intended use)
- Track and fix defects, quickly (maintain the reuse artifacts)
- Respond to questions and issues (support the reuse artifacts)
- Provide training on the reuse artifacts so people understand use and limitations
- Keep the reuse artifacts current ('old things' are never good, new components 'have to be better')

Note that this implies that the effort is not a part-time or additional effort, but is a major responsibility. The comment about the 'right people' relates to people who really are architects and library developers. Not all developers can develop that type of work product.

Although there are many aspects to the organizational issues, the key is to actively lead the move to reuse. This involves, most importantly, having an identified management champion for the reuse vision who can affect and influence the organization and take it through the necessary changes. The other thing to keep in mind is that components and architectures age. A key role of the champion is to keep an eye on when is the time right to close off one reuse platform and start migrating to the next.

Summary

Reuse is a technical tactic to achieve business strategies. It is best achieved through incremental pilots leading to broader organizational adoption. It requires a real change in the organization development culture. This requires a need, a vision, plans, a champion and action to be successful.

In closing, again drawing from Martin Griss, some key reuse guidelines to remember are:

Ten Commandments Of Effective Reuse

Critical to architect system, process, and organization

1. **Long-term management leadership**
2. **Systematic, incremental organization and process change**

3. **Manage assets as product family portfolio**
4. **Design for domain-specific, architected reuse**
5. **Separate: Create / Support / Utilize**
6. **Objects alone are not sufficient**
7. **Start small pilots, then scale up**
8. **Address culture, experience and organization stability**
9. **Invest in infrastructure, support, and training**
10. **Find dedicated reuse champion**

Bibliography and Additional Resources

1. The Reuse Rabbi's Report: So you want to reuse software and firmware, huh? By Martin L. Griss. *http://swtc.lvld.hp.com/newsletter/sept96/rabbi.htm*
2. The Reuse Rabbi's Report: Well, you have to get organized! By Martin L. Griss. *http://swtc.lvld.hp.com/newsletter/dec96/rabbi.htm*
3. Managing for Reuse: Experiences with the LDS Firmware Reuse Program. By Joe Mueller. *http://swtc.lvld.hp.com/newsletter/dec96/lds.htm*
4. The Reuse Rabbi's Report: The SEI CMM as a Framework. for Adopting Systematic Reuse. Martin L. Griss. *http://swtc.lvld.hp.com/newsletter/dec97/rabbi.htm*
5. The Reuse Rabbi's Report: Systematic Reuse for Fast Cycle Time. By Martin L. Griss. *http://swtc.lvld.hp.com/newsletter/march97/reuse-rabbi.html*
6. The Reuse Rabbi's Report: Processes for Systematic Reuse. By Martin L. Griss. *http://swtc.lvld.hp.com/newsletter/june97/rabbi.html*
7. Software Reuse: Architecture, Process and Organization for Business Success. By Martin L Griss and Patrick Johnsson. Addison-Wesley. Spring 1997.