

Reuse Comes in Several Flavors

Achieving a Better Return on Software™ Series



Key Message

Reuse can be practiced in a variety of ways and customized to accommodate different business goals, pace, and investment level.

Principles

Different flavors of reuse vary the amount of:

- Proactive management of assets and process
- Proactive design and architecture of assets to enhance reuse
- Top-down, pre-work, versus bottom-up incremental reengineering

Three Primary Modes of Reuse

A simple reuse framework will distinguish (and appropriately combine) at least three kinds of reuse:

- Facilitated
- Managed
- Designed

Topics of Interest

The choice of mode(s) and the related people, process, methods, and tools needed depend on domain, situation, resource, and readiness. Several issues will be discussed.

- People, process, and tools
- Domain Engineering
- Component mining
- Open Source vs. Systematic Reuse

Martin Griss

Flashline Software Development Productivity Council

The Potential Benefits of Reuse

The general concept of reuse is well known: by developing software assets once, and encouraging others to reuse them rather than developing assets from scratch, a variety of significant quality, time, and cost benefits result. In certain circumstances these benefits can be quite dramatic — a three-fold or greater reduction in time-to-market, for example.

However, in order to maximize the potential of these benefits, it is important to align the reuse model chosen with strategic business goals¹, customizing the reuse model as necessary. Typical reuse-friendly business goals include:

- Improved Business and IT agility
- Reduced development and maintenance time
- Improved quality
- Reduced development and maintenance costs

Other issues that will affect your decision regarding what kind of reuse program to establish include the scope and pace of the program, the nature and availability of the resources necessary to establish the program, and the existence of synergistic or competitive software improvement initiatives.

Selecting a Reuse Flavor

Within the reuse community, several approaches to reuse have emerged. Practitioners can select different policies, processes, and technologies to match a reuse program to the business goals and capabilities of an organization. Some of the learning has been summarized in a variety of Reuse Maturity Models,^{2,3} suggesting that the benefits, scope, and formality of a reuse program grows as more experience is gained.

There are three basic modes² in which software asset reuse can be practiced in an organization; there is also a question of how broadly across the organization a practice should be applied.

- **Facilitated** (or Encouraged): In this first step beyond totally ad-hoc reuse, a small investment is made to make it easier for people to practice reuse, but there is no requirement that they do so. Tools and policies are established to make it easier to submit, publish, find and use reusable assets across the requisite organization. Some support is provided to track usage and inform consumers of changes to the assets they have retrieved. Some evangelism and incentives are employed to encourage participation as either submitter or consumer, but very little extra effort is directed at validating that good assets are created, that the assets are the best to reuse, and that they are in fact reused. Investment is made in a low cost (or no cost) repository solution, with very limited infrastructure and little overt interference with developers. At this level there is modest payback and minimal if any measurement. There may be a reuse advocate or evangelist to help encourage people to practice reuse, but very little explicit support is provided.

Table 1 - Characteristics of the primary reuse flavors

	"Ad hoc" Reuse	Facilitated Reuse	Managed Reuse	Designed Reuse
Theme	Individuals find assets on their own, sharing with colleagues as they choose.	Organization encourages and supports reuse with limited resources, infrastructure, and policies to make reuse easier.	Organization enforces reuse practice through policies, resources, tools, and people.	Organization invests in carefully designing assets for reuse, choosing assets for domain or product line. Assets are architected or reengineered to fit together.
Typical Reuse Level	Varies, usually unknown	5-15%	15-50%	40-90%
Relative Cost	Minimal	Low	Medium	High
People / Roles	Individual effort; no reuse roles.	Evangelist, reuse facilitator/web-master/librarian.	Facilitated roles plus librarians/registrars, certifiers/reviewers, process engineers	Managed roles plus architect(s), domain analyst(s), component developers, component support specialists.
Process & Policies	None	Incentives, asset check-in process, limited review before publication.	Reuse-adapted process, mandated with specific goals. Reuse reviews, and asset documentation, packaging, and certification guidelines, specified metrics to be collected.	Some domain analysis, reuse-oriented architecture, specific steps to design for and with reuse.
Tools & Technology	File system or minimal web site; email.	Self-use repository or Web site, submitters may add some metadata to improve search and evaluation.	Registrar-monitored repository, extensive metadata, multi-project source code control, asset quality assurance, change notification, utilization measurements.	Requirements and portfolio management tools; frameworks, standards, generators; change management tools.

■ **Managed** (or Mandated): In this mode, formal structures, policies, and tools for enforcement or control are implemented to ensure that reuse is practiced and results measured. Additional policies, reviews, process steps, and tools are used to ensure an adequate supply of high-quality assets. Similarly, reviews, process steps, and release checkpoints are required to ensure that an appropriate effort is expended to reuse the recommended assets as-is, rather than developing a new asset from scratch, finding some alternative asset, or radically modifying an asset. Reuse metrics are captured and reuse program ROI is measured.

■ **Designed** (or Architected): In this mode proactive measures are taken to ensure that assets are developed to meet specific needs. Architecture and domain engineering techniques are used to determine which assets should be developed to cover more of the application space. "Design for reuse" guidelines, standards, and additional reviews ensure that high-quality, compatible assets are produced. This strategy works best when developing for a specific application family or product line within a relatively stable domain.

A key factor in the decision to adopt either the Managed or Designed mode is the increased expenditure of resources and management effort. More time and more investment may be required to design, develop, and certify the necessary high-quality assets, and increased management is required to ensure that people will use these assets. Additional training and organizational change may also be necessary, as well as a significant pilot demonstration program. The Managed and Designed modes also allocate the resources necessary to develop only those assets whose quality and flexibility

ensure a high probability of reuse. This additional effort is repaid in repeated reuse and the rapid development of higher quality applications. This strategy also minimizes the need to alter the assets, further enhancing quality by decreasing the repeated testing and maintenance needed with multiple versions of similar assets.

Key difference between Facilitated and Managed is a more active style of management and larger commitment of resources.

Key difference between Managed and Designed is amount of proactive development of reusable assets.

The key difference between the Facilitated and Managed reuse modes rests in the chosen form of reuse program **governance**: the level of overt control exerted to enforce the practice of reuse in accordance with organizational policies and guidelines. At one extreme of Facilitated reuse there is minimal infrastructure (e.g., a self-publishing Website using WIKI or WebDav) and no designated reuse advocate or evangelist. At the other extreme you'll find a corporate reuse office with design-for-reuse guidelines and an incentives program. In either extreme, or at any point in between, the Facilitated mode leaves the decision to practice reuse to the individual developer. In the Managed mode, however, the practice of asset reuse is a requirement of the job. Enforcement is accomplished with either a light or heavy hand, with lightweight, Agile or Open Source processes at one end of the spectrum, and at the other, heavier RUP or CMM methodologies that have been extended for reuse. To support the Managed reuse objective, the governance model will require more organizational structure, monitoring, and measurements.

These flavors (or modes) of reuse do not constitute a strict "reuse maturity model." It is important to note that it is quite possible for an organization to choose Designed reuse, but to run it in Facilitated mode, rather than Managed mode. Effort could be proactively expended to design and construct high quality assets, or to establish a roadmap and targets for the discovery and reengineering of a domain of assets, but still leave it to individuals and projects to choose to use these assets. Encouragement, facilitation, incentives, and evangelism could be used to market these assets to developers in order to expand their use.

Aspects of each reuse flavor can be mixed to achieve specific business goals.

It is also worth contrasting **ad hoc** reuse from the other modes that require more resource and management commitment. Many individuals and some project teams practice an informal style of reuse — looking for assets on the public or corporate Web; sending email to colleagues asking for, or telling of, useful assets or starting-point code for "cut and paste" reuse; and looking for code in prior projects. **Ad hoc** reuse is important because it is the baseline against which the organizational reuse initiatives are compared. **Ad hoc** reuse is widely practiced today. The Internet, code collection sites such as AlphaWorks and SourceForge, and peer-to-peer sharing tools make informal reuse much easier. The key difference is that **ad hoc** reuse is driven by the individual (though an organization could discourage the practice by locking out the Internet, making a fuss over code provenance, and so on.), while the other three are clearly conscious, proactive organizational decisions.

Ad hoc reuse – no proactive organizational involvement.

The words **Tactical** and **Strategic** are frequently used to talk about the intent and scope of a reuse program. The terms do not describe flavors of reuse organization, but rather are used to describe how and why an organization chooses to practice reuse, and how important reuse is to the business.

- **Tactical**: Reuse is used primarily as a technique to reduce time and cost, along with other software development improvement initiatives. Depending on the explicit improvement desired, this can be achieved through Facilitated, Managed, or Designed reuse.
- **Strategic**: Reuse plays a vital role in achieving the overall business vision and goals, and is a key corporate competitive strategy. Companies invest in time-to-market and flexibility tools and processes as part of their market growth initiatives. Typically,

achieving Strategic reuse requires that the reuse program be run as Designed and Managed, to support a specific domain (such as customer service or e-commerce) or to produce a specific product line (such as in financial services or travel-related services).

How to Find and Use Assets

There are a variety of different ways to create or acquire reusable assets; each approach has pros and cons. In practice, these coexist in most projects and organizations. No single method offers any distinct advantage over the others, so it is wise to be conscious of the choices, actively adjusting the strategy and investment profile as needed.

Assets can be acquired via any of the following methods:

- With the utility asset library supplied with the operating system, compiler or application development environment (e.g., Microsoft®).
- Through purchase from a company or broker, usually without source (e.g., RogueWave®, ILOG®).
- With source code, from an Open Source community (such as SourceForge®), a university, or IBM® AlphaWorks®.
- From an internal company repository.
- Through “mining” of legacy code or previous versions of products in the product-line.
- Through a swap, barter, or purchase agreement or contract with another internal group.
- Created as part of the regular product or application development process, then packaged and published to a corporate or local repository.
- Proactively defined to meet the needs of new projects, then developed prior to an application development project, or as a sub-project.
- Through the evolution, reengineering, or wrapping of existing assets obtained from another source, in order to be more appropriate to current and future projects.
- Created by a distinct component development and maintenance team that develops components and Web services only for release to others.

Any of these sources can be used with either Facilitated or Managed reuse — the key difference is how much quality control is exerted before publishing and recommending the use of any asset. In some cases, an assessment of merit or grade is associated with assets of different quality and value. Assets can then be promoted to higher grades and their use encouraged or mandated. Also, in Managed reuse, some effort will be expended to control the proliferation of multiple versions of the same or similar assets, so that redundant maintenance and support can be avoided.

Similarly, Designed reuse can use several of these sources. The key idea is to have a plan or roadmap of which domains are considered important to the organization, and how assets from disparate sources are meant to operate together. This plan can then be used to guide the harvesting, reengineering, purchasing, and/or purposeful development of the required assets. Keep in mind that Designed reuse does not automatically imply expensive new development; rather it is a strategic choice to limit and focus investment and energy on the important assets.

Major approaches to asset development and ownership organizational structures:

- Project teams develop and own assets specifically for their use. Costs borne by the “developed for” project. Assets shared where possible.
- Project teams develop and own assets with some awareness of needs of other projects. Development and maintenance costs shared with other projects.
- Centralized team (e.g. and architecture team) or shared services group develops and maintains assets for use by multiple projects. Costs can be absorbed or shared on a “per use” basis.

Domain engineering – produces reusable assets for an application family or product line.

Domain analysis – determines the common needs and feature set of an application family or product line.

Domain analysis produces a roadmap for future component development or harvesting

There are several orthogonal dimensions that can be used to analyze your situation.

- Planned, proactive design, and development of assets vs. ad hoc/byproduct discovery/mining.

Do you perform an explicit domain analysis and asset inventory to identify promising candidates, perhaps for new development?

Or...

Do you determine at the end of a project which code or other artifacts are the most reusable? Do you go hunting for reusable pieces as needed?

- Top-down vs. bottom-up design/development/discovery of components.

Do you develop a large-scale use case and architecture design first, leading to subsystem design, and then create new components?

Or...

Do you perform an asset inventory, followed by (mechanical) mining from existing systems or ongoing projects?

- Waterfall vs. incremental/iterative vs. concurrent development of components vis-à-vis development of the applications using them.

Do you develop components first, then applications?

Or...

Do you develop a cycle of components then a cycle of product?

Or...

Do you engage in parallel development of both components and applications? (This is easier with multiple teams)

- Same vs. different project teams produce components and applications.

A key differentiator is the existence of a plan to develop and release components, and specifically to proactively look for specific kinds of components for subsequent reengineering, re-documentation, and/or re-packaging, rather than just finding them along the way.

Asset and Reuse Initiative Support

In a similar vein, there are a variety of ways in which the reusable assets can be supported, maintained, and evolved.

- Supported: Assistance in using an asset is provided through email, consulting, training, additional documentation, and mentors.
- Maintained: Reported/discovered defects to assets are corrected.
- Evolved: Work is performed to add requested or planned features to assets.

In general, this additional work can be provided centrally, regionally or locally by the individual asset consumer, by the original asset producer, or by specially chartered support teams. The specifics depend on the scope of the reuse initiative, on the amount of support, on the organizational culture, and amount of control, and, to some degree, on the mode of reuse.

Also, some care and feeding of the overall reuse initiative will be required, including quality assurance, training, process enhancements, acquisition, and operation of a repository, reporting, and administration of an incentive program. These activities and

budgets can be centralized or regionalized into one or more reuse support organizations, or largely distributed to developer or asset consumer individuals or teams. Here again, the specifics depend on the scope, resources, and culture of the organization, and the level of control.

Typically, in Facilitated reuse there will be little investment in the support, maintenance, and evolution of assets, other than that provided by volunteers who contribute or use assets. Similarly, there will be minimal investment in administering the reuse *program*, other than perhaps the care and feeding of a Web site, some incentives, and periodic promotional events to encourage participation in the program.

An important task in Managed reuse is to decide how much to invest in support, maintenance, evolution, and program administration, and to ensure that the investment yields an appropriate return. This is one of the primary functions of reuse metrics.

In Designed reuse, additional effort is expended in proactively identifying, reengineering, and/or developing assets, so even more effort is needed to encourage/ensure that the right assets are developed and used.

The Role of Domain Analysis and Domain Engineering

Domain engineering (DE) is a process that produces reusable assets (components, Web services, generators, frameworks, models, documents) for subsequent use in the development of applications or product line. Domain analysis (DA) is the front part of this process, which analyzes the anticipated applications, technology trends, standards, and existing assets to develop a model of commonality, variability, and initial “chunking” of features into reusable assets.

Domain analysis should not be confused with waterfall/upfront development. It is a process, perhaps concurrent with other development processes, that is used to identify appropriate reusable assets. Some form of this process is an important part of, and most appropriate for, Designed reuse, but can also be used in Managed or even Facilitated reuse. DA can be used to create a roadmap of required variability and reusable assets, which can then be used to drive the creation, either upfront or incremental, of components as applications are developed. DA can also help to identify which components to harvest from existing applications.

Domain Analysis, Engineering, and Architecting will be addressed in greater detail in a future white paper.

Component Mining

As mentioned above, component mining (or harvesting) can be a useful way of finding assets in any of the flavors, though more systematic mining will be performed primarily as part of Managed reuse (wherein a directive to find and use assets is issued) and Designed reuse (wherein the objective is the identification and development of appropriate, domain-specific assets).

Component mining can be either a bottom-up or a top-down process. With a bottom-up approach, developers use tools based on cohesion/coupling metrics⁴ to find candidate assets within existing legacy systems. They would also find promising candidates while doing a purposeful inventory of code used by others, or believed to be potentially reusable. Experience shows that it is easy to find roughly 10% of code that is already modular enough, and is (re)used a sufficient number of times in the legacy systems to justify a small amount of re-packaging, re-documenting or re-engineering to extract these components and make these into reusable assets.

Top-down component mining proceeds to find and reengineer candidate assets as a conscious by product of domain analysis while analyzing exemplars. As the candidate domain model and architecture is evolved, a domain dictionary and asset inventory is produced, which provides a guide as to which assets are needed and how much reengineering they will need before they appropriately “fit” the target asset architecture.

Open Source as a Form of Reuse

See <http://www.opensource.org> for more information on the Open Source Initiative.

Without going into great detail, it is worth noting that the encouragement and practice of open source techniques can be viewed as a form of Facilitated reuse, and perhaps even as a part of a Managed reuse program. It is also worth noting that a number of organizations have advocated the use of Open Source techniques⁵, either through participation in external Open Source efforts, or by setting up some form of Open Source community and repository inside the organization (so called Corporate Source⁶). The key idea here is large-scale collaborative development and sharing of source code, and is therefore a form of organized reuse.

Depending on the strength of the roles of architects and gatekeepers, this operates as a form of Facilitated reuse, with some small flavorings of Managed and Designed reuse. Fundamentally, people are encouraged to participate through strong community building, but there are distinct policies about which people can modify and contribute which kinds of code, and how code should be tested and documented before acceptance for reuse.

The key observation is that if a policy of Facilitated reuse has been adopted, the organization should be careful to appropriately align the reuse program and the open source program.

Some Experience in Adopting and Combining Reuse Modes

Let's say we choose an overall strategy of reuse-driven improvement, but we want to proceed incrementally to avoid “organizational shock” and over-investment. The key is to view Facilitated reuse and the supporting repository at its core as just the first step towards the Managed and Designed modes.

In a hypothetical example, we would first move to establish and then expand the Facilitated mode with a repository and initial policies to encourage participation. At the same time, we could introduce Inspections and some PSP or XP training, refining experience in several pilot projects. Then, in the context of a larger CMM improvement initiative, we could expand the reuse effort to a Managed mode, improve configuration management of assets and products, and at the same time increase the use of Inspections and systematic asset review, and implement increased testing and documentation to improve the quality of *recommended* and *required* assets. Finally, we would begin RUP training and adoption as we design more of the assets and define the architecture of the supporting framework.

As a real example, consider that in one multi-divisional HP instrument business unit a crucial goal was to significantly decrease time to market, at the same time that the variety of products was increasing. A lightweight, incremental Designed reuse strategy was adopted, followed by an evolving mixture of Facilitated and Managed reuse. First, in less than a year they developed initial core instrument architecture and reengineered supporting components within several projects. These were made available for optional use, and an active program of encouragement saw the launch a number of pilot projects. After demonstrating initial success, the number of components was

expanded over a period of several years. Use of these components was mandated, and a proactive approach to the architecting and implementation of targeted components and frameworks was adopted. As time-to-market shrank (from over 18 months to as little as 5 months for new development) and reuse of components expanded (with levels ranging from 25% to over 50%) to include more projects, the reuse program expanded to a Managed mode, with an explicit component support group.

Summary and Conclusions

Reuse can be practiced in a variety of ways, and customized to suit several different business goals, time frames, and investment levels. In choosing and adopting a Facilitated, Managed and/or Designed mode it is most important to understand the potential costs and payoff associated with each mode in order to correctly assess the situation and need with respect to domain, available resources, and organizational readiness, and to move incrementally to make the organization, process and technology changes.

References

1. M. Griss. 2002. *Ranking IT Productivity Improvement Strategies*, Flashline, Inc.
2. I. Jacobson, M. Griss, P. Jonsson. 1997. *Software Reuse: Architecture, Process, and Organization for Business Success*. Addison-Wesley.
3. W. Lim. 1998. *Managing Software Reuse: A Comprehensive Guide to Strategically Reengineering the Organization for Reusable Components*, Prentice Hall Professional.
4. V. Basili, G. Caldieri. 1991. Identifying and Qualifying reusable software components. *IEEE Computer*.
5. J. Feller, B. Fitzgerald, E. Raymond. 2001. *Understanding Open Source Software Development*. Addison-Wesley. See also: <http://www.opensource.org/> and <http://www.osdn.com/>.
6. Pj Garg, J. Dinckelacker. 2001. *Applying Open Source Concepts to a Corporate Environment*. 1st ICSE International Workshop on Open Source Software Engineering.

f1 This 3 mode model is a simplification of the 5 step "reuse maturity model" stair- diagram in my book² and similar reuse maturity models discussed in Wayne's book³.

The word asset suggests a corporate investment and suggests that more than just code components will be reused.

Author Profile

Martin Griss is highly regarded as one of the leading authorities on software reuse, having spent nearly two decades as Principal Laboratory Scientist at Hewlett-Packard and as Director of HP's Software Technology Laboratory. At HP, Griss was widely known as the company's "Reuse Rabbi" where he helped introduce software reuse into the company's development processes and defined a reuse consultancy. His research has covered software reuse processes and tools, software agents, software factories, component-based software engineering, and software-bus frameworks. Griss has written extensively on software reuse for a number of industry publications, and is co-author of the widely read *Software Reuse: Architecture, Process and Organization for Business Success*. He is currently an Adjunct Professor of Computer Science at the University of California, Santa Cruz, and at the University of Utah.

Mr. Griss earned a B.Sc. from the Technion in Israel in 1967 and a Ph.D. in Physics from the University of Illinois in 1971.

About the Flashline Software Development Productivity Council

Martin Griss is a member of Flashline's Software Development Productivity Council (SDPC), an unprecedented assemblage of knowledge and experience covering the broad spectrum of software development areas including software engineering process and environments that affect today's corporations. The members of the SDPC are individually recognized authors and consultants in a wide range of subjects covering the entire software development lifecycle and its relation to the corporate bottom line. Their expertise guides the evolution of Flashline's products and services. Through direct consultation with Flashline customers, the SDPC aids in the realization of the goal of greater return of software.

