SOFTWARE REUSE: A COMPANY VISION

Paul E. McMahon
Staff Scientist, CAE-Link Corporation, Binghamton, New York
Adjunct Faculty, Computer Science Dept., Binghamton University

ABSTRACT

Today, many exciting initiatives are underway within the software industry. Structure Modelling technology is growing rapidly through efforts at the Software Engineering Institute (SEI) and ongoing projects. Megaprogramming challenges are being faced on the ARPA STARS project. Open standards including POSIX, X-Windows, and Motif are becoming realities as key software vendors position themselves to support these initiatives.

Reuse library tools and guidelines are also being developed through efforts at the SEI, the Software Productivity Consortium (SPC), and on the STARS project. At the same time, software contractors are moving forward with serious strategies to improve their company software processes in response to industry initiatives including the ISO 9000 requirements and the SEI Process Maturity Model.

All these initiatives share the common objective of cost reduction and most are looking to one form or another of software reuse to achieve this goal.

This paper examines the multi-faceted issues of reuse and the role these current industry initiatives play within reuse technology. Issues discussed include analyzing existing software for reuse, techniques to design for reuse, reusable software architectures, managing variant versions of software, and managing a corporate reuse library. Technical and management issues are presented.

The paper focuses on lessons learned from efforts at CAE-Link to infuse software reuse techniques into the corporate culture. Practical techniques being applied today to meet reuse challenges are discussed. The key roles of reuse criteria, metrics, company software standardization, project-company interaction, management mandates and training and education are discussed.

Experiences and examples are provided from the B-2 ATD project, Independent Research and Development, and a corporate software Process Action Team that was instrumental in providing the focus necessary to move the company forward with an effective and practical reuse initiative.

ABOUT THE AUTHOR

Paul E. McMahon is a Staff Scientist at the Binghamton Operations of CAE-Link Corporation and an Adjunct Faculty member of the Computer Science Department at Binghamton University. Mr. McMahon has been with Link since 1973, holding various technical and management positions within the company. Mr. McMahon has published numerous papers on Ada and software engineering including a paper entitled "Lessons Learned on the Fringe of Ada", which was nominated for best paper at the 1989 Interservice/Industry Training Systems Conference and a paper entitled "Software Metrics, Ada, and the B-2 ATD", which was awarded best paper at the 1991 Interservice/Industry Training Systems and Education Conference. Mr. McMahon teaches Software Engineering at Binghamton University.

SOFTWARE REUSE: A COMPANY VISION

Paul E. McMahon
Staff Scientist, CAE-Link Corporation, Binghamton, New York
Adjunct Faculty, Computer Science Dept., Binghamton University

INTRODUCTION

What is Software Reuse?

To many, software reuse means code. Reuse, however, is multi-faceted. In fact, its greatest potential for cost reduction may be found in other software forms such as requirements, design, documentation, and the development process itself. Most software industry initiatives today are striving for one or more of these forms of reuse. See Table 1.

Form of Reuse	Industry Initiative
Requirements	OOA, CASE Tools, Workstations
Design and Documentation	OOD, Structure Modelling, Megaprogramming, CASE Tools, Workstations, Reuse Library
Code	OOP, Variants, Reuse Library, Autogeneration
Test Cases	CASE Tools, Regression Testing
Development Process	SEI Maturity Model, ISO 9000

Table 1 Forms of Reuse

Tools and Techniques

Object Oriented tools and techniques are popular today largely due to their potential to provide more reusable products. CASE tools provide the potential for standard representations resulting in reusable requirements, designs and documentation. Reuse tools can aid organizations as they transition to software processes with more of a focus on reuse.

Software Architectures

Structure Modelling and Megaprogramming initiatives each provide forms of software architecture reuse. Structure Modelling focuses on reuse through design commonality. Once common design elements are identified, reuse can be enforced through the use of structure model templates.

Megaprogramming focuses on reusing complete software infrastructures. "Megamodules are independently maintained software systems managed by a community with its own terminology, goals, knowledge, and programming traditions."[1]

Distributed Interactive Simulation (DIS) is a form of Megaprogramming that reuses complete training devices to face new training needs despite the fact that these devices were never envisioned to operate in this fashion. Communication and control through standardized protocols are key attributes within this emerging technology.

Open Standards and Process Improvement

Efforts geared toward industry open standards and company process improvements provide other forms of reuse.

Open standards means software is reusable across vendor platforms.

Company process improvement efforts provide standardization and repeatability of software processes across company projects. This results in reuse of procedures, tools, training and even corporate knowledge.

Software Development efforts of the past relied heavily on the knowledge and opinion of subject matter experts to keep projects on course. Today, company initiatives are moving away from reliance on individuals toward defined and managed processes that are repeatable, cost-effective, and independent of the skills of particular individuals.

Why is Software Reuse Important?

The primary reason software reuse is key today is the need to improve cost-effectiveness. Finding better ways to reuse software to meet our needs means that we can do more for less. This translates into

reduced development cost, reduced risk and increased reliability and maintainability.

REUSE TECHNIQUES

Due to the many forms of reuse, there are also a number of implementation techniques.

We have identified six key reuse techniques at Link.

1. Analyzing Existing Software
2. Designing For Reuse
3. Structure Modelling
4. Variants
5. The Corporate Reuse Library
6. Management Support and Mandate

Analyzing Existing Software

The fact that software exists does not imply its suitability for reuse. Existing software must be analyzed against established criteria to determine its reuse potential. This analysis includes answering the following key questions:

1. What do we want to reuse (design, documentation, test cases, code)?
2. Is the design compatible with the planned software architecture?
3. Do we want to reuse only the algorithmic design or more?

It is currently believed that the most valuable reusable software products may be analysis and design. In many cases, math models may be reusable, but coded modules may not be compatible with modern software architectures. Reuse analysis is necessary to answer these questions.

Designing For Reuse

Designing software to be reusable may cost more. This is because our development model and techniques change when we are designing for reuse. This is an investment that will begin to pay off as the reuse library is populated. The reuse library is discussed later in the paper.

We have identified four key factors in designing reusable software. It is recommended that these factors be employed as a review criteria for determining acceptability of software for the reuse library.

1. Reuse Existing Software First

When designing reusable software, our development model changes. The first step is to look to reusable components to meet needs. Our goal is to build solutions from existing library components rather than reinvent new solutions. Technical trade-offs may be necessary.

This may mean early discussions with the customer refining requirements to support maximum reuse to reduce overall cost. Customers will need to be aware of these changes in contractor reuse processes.

2. Follow Standards

Follow company software standards to ensure new software meets reuse criteria. In a reuse development environment key standards include software naming conventions, standard software packages, and established architecture guidelines.

3. Isolate Device Specifics

Device specific data must be isolated minimizing the effort required to adapt reusable software to changing requirements. The management of modified reusable software is discussed further in the section on variants.

4. Use Object Oriented Techniques

Current studies and our experience to date indicate object oriented techniques result in more reusable and maintainable software than traditional methods.

Structure Modeling

Work at the SEI and experiences on the B-2 ATD indicate that structure modeling techniques are effective at enforcing reusable common designs, simplifying training, and reducing schedule and computational resource risks.

Engineering productivity may also be enhanced through autogeneration of structure model components. CASE tools may be used to enforce common design decisions supporting the structure model.

Variants

The Variant aids in the management of reuse during software modifications. A variant is a software component with a special relationship to a "parent" software component. Both the parent and the variant

are independently managed and tracked, but the variant reuses a significant amount of the parent software.

There are similarities between the concept of variant and the concept of inheritance found within object oriented languages. Inheritance provides a form of reuse, but may include a run-time penalty.

Variants, on the other hand, are managed through an off-line configuration management system eliminating run-time overhead. The off-line system manages the relationships between "parent" units and variant offspring tracking and measuring variant reuse.

Variants provide a powerful mechanism to manage large collections of reusable software components across many similar, but functionally modified systems.

Corporate Reuse Library

A corporate reuse library is critical to the success of a company-wide reuse effort. However, equally important to its functional capabilities, the library processes and procedures must be integrated with company existing software processes and tools. This includes approvals, reviews and change notifications.

While many large companies do not yet have reuse technology as part of their software process model, they may have well-established software configuration management tools and procedures. The infusion of reuse technology into the company must minimize redundant engineering effort. In particular, reuse processes and tools should be integrated closely with existing software configuration management processes, ensuring that identification, statusing, approvals, and notifications are processed as efficiently as possible.

Management Support and Mandate

Changing a company's software process model to effectively support reuse requires more than a reuse library and a technical understanding of the issues. Software engineers frequently prefer to reinvent rather than reuse unless clear direction to do otherwise is provided.

For reuse technology to effectively take hold in a corporation, it is essential to educate all levels of software management in the latest reuse principles and strongly support the company reuse effort. A management mandate to follow the principles of reuse must be clear to all involved in software production. To manage this effort successfully, reuse objectives should be established. A reuse program should include metrics and feedback of results, taking corrective action where necessary. Through the reuse library, integrated closely with a disciplined configuration management system and strong management backing, the capabilities exist to measure, manage, and succeed with reuse technology.

Past attempts at Corporate level reuse have failed largely for four reasons. First, criteria, control, and approval were unclear. This resulted in product changes initiated at the project level that were inconsistent with the company vision. Second, adequate resources were not supplied to support the company perspective. Third, motivation from the organization continued to be the project rather than the company. Fourth, the notion of reuse as code only was a barrier. Change in each of these areas is essential to the success of a corporate reuse effort.

CAE-LINK INITIATIVES BACKGROUND

In the first half of 1992, an outside consulting organization was placed under contract by CAE-Link to facilitate Company-Wide process improvements. A software process action team (PAT) consisting of CAE-Link senior engineers and software consultants was formed. The objective of the team was to study current software policies and processes at Link and implement improvements necessary to reduce software life-cycle cost.

One and one-half years earlier an effort was initiated to export the B-2 ATD Ada software process and tool-set, making it available for other CAE-Link projects. Since that time, enhancements to both the tools and the software process have continued on the B-2 project and through Independent Research and Development at Link. The information presented in this section is based on these activities.

Software Process Action Team (PAT) Lessons Learned

The Software Process Action Team (PAT) conducted interviews throughout the

Company with both junior and senior software engineers and software managers to identify key areas for potential improvement. Seven target areas for improvement were identified as a result of this activity. These include:

1. Eliminate process redundancies and non-value added work.
2. Do not release (put under formal configuration control) software until fully tested.
3. Improve the off-line test environment and tools.
4. Focus on early error detection.
5. Improve software training.
6. Establish company level standard metrics.
7. Establish a Corporate software reuse library with defined procedures for reviews, approvals, and reuse criteria.

Process Action Team Results

In parallel with the interviews, the PAT members and the consultants met periodically to brainstorm potential improvement strategies. Presentations from experienced senior technical engineers frequently provided the focus for discussions.

As a result, action plans were established and carried out. The company Software Standards and Procedures Manual was modified addressing targeted areas for improvement.

Formal company software training classes were prepared and conducted in support of these initiatives. This training included:

1. Object Oriented Techniques
2. Criteria for:
   a. Releasing software
   b. Design level of detail
   c. Risk assessment
3. Standard company design representation
4. Standard company Structure Model
5. Standard company software metrics
6. Standard company software process
7. Software management techniques (i.e., algorithmic cost estimating).

See Figure 1.

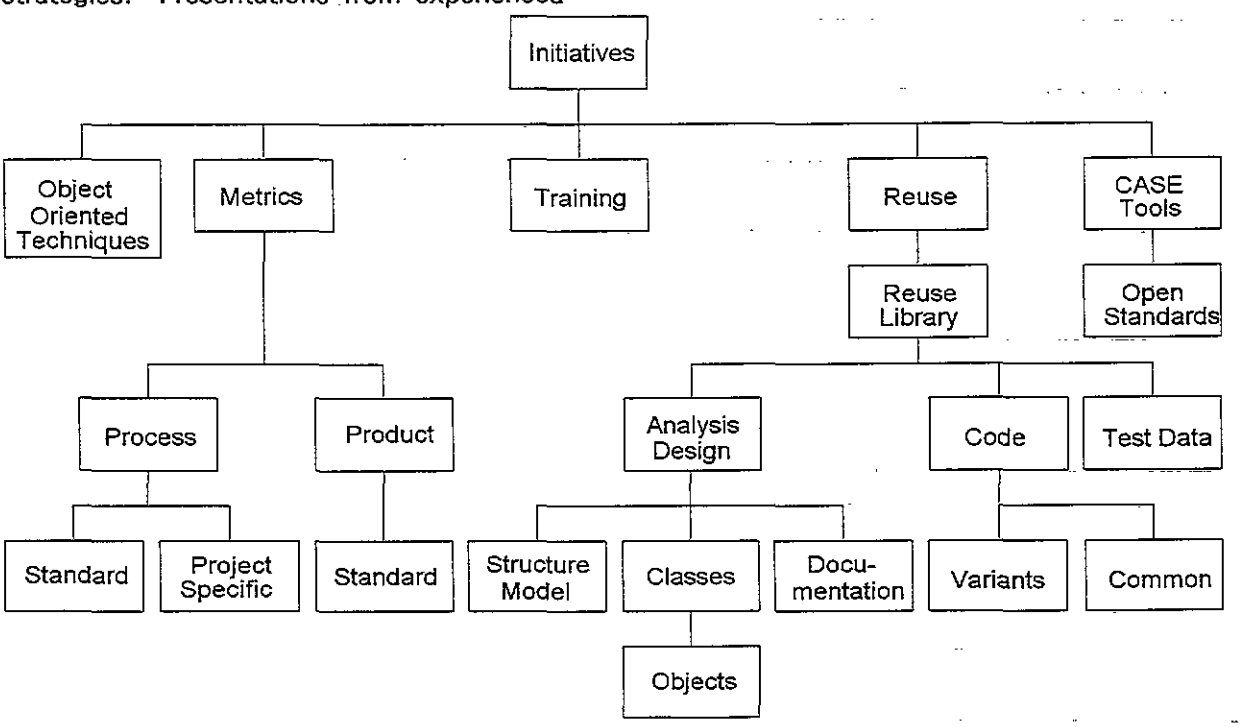

Figure 1 Process Improvement Initiatives

Reuse Through Process Improvement

Making the process repeatable is another form of software reuse. The SEI Capability Maturity Model defines five process levels. See Table 2.

Level	Capability	Description
1	Ad Hoc	Process not repeatable
2	Repeatable	Repeatable, but dependent on people
3	Defined	Procedures and Process Training
4	Managed	Metrics collected
5	Self-Improving	Metrics feedback for improvement

Table 2 SEI Process Maturity Model

Our objective is to reuse training, tools, procedures, and standards company wide. However, it is critical that any methods reused company wide be as cost-effective as possible. This implies that the company standard method must be self-improving. Repeatable and defined is the first step. However, a self-improving process must be our ultimate objective (SEI Level 5).

To achieve level five, metrics are necessary. Selecting the "right" set of standard metrics for your organization is a key to success. The right set for one organization may not be right for another. The metrics chosen must add real value to each organization's own process, while, at the same time, adding minimum burden to the software engineer.

Our software processes at Link are based on a combination of modern software engineering principles, and experiences and lessons learned from real-world projects. Metrics provide an example of this blend of real-world experience and textbook theory.

METRICS

To ensure our reusable processes, procedures, and tools are as efficient as possible we must measure. In establishing our company approach to metrics, we listened to the people from our ongoing projects.

We found from our experience in applying metrics on the B-2 ATD project that there are two distinct kinds of metrics necessary to support real process improvement. We refer to these two types as Standard and Non-Standard metrics.

Standard Metrics History

Standard metrics are those that are applicable to all software developed at Link. Standard metrics are collected periodically and are used as feedback for process improvement.

As part of our PAT interviews, we asked managers and engineers for feedback on the effectiveness of standard metrics. These interviews provided us with some key insights.

First, we found that standard metrics were not being collected consistently across all projects. Second, we found that the lower one moved into the organization, the less value was reported with the use of these metrics. Metrics were not being collected consistently because the engineers and their immediate supervisors found minimal value in this data.

Project engineers, however, reported that software metric reports were found to be useful. Project engineers used metrics to identify trends and potential problem areas early.

Standard Metrics Lesson Learned

Engineers and first level managers tend to be close to the problems on a daily basis. As a result, the value of metrics as a management aid at this level is low. Metrics are trend indicators. The higher one's perspective or span of accountability, the more valuable they become.

We found that metrics are particularly valuable to those concerned with multiple projects. Metrics can provide a common ground for comparison leading to better understanding of the root cause of problems.

The Process Action Team established a company set of standard software metrics, and initiated training of all personnel including engineers and managers in why it is important to collect this data accurately from a company perspective.

We found that people respond positively to initiatives when they comprehend the

motivation. This means education. Metrics is becoming an integral part of our training program as well as our company software vision.

CAE-Link Standard Metrics

1. Cost
2. Schedule
3. Manpower
4. *Execution Time*
5. Memory
6. Complexity
7. Source Lines of Code (SLOCS)
8. Stability (Rate of Change)
9. Problem Category.

Non-Standard Metrics

The value of metrics rests in process improvement. Optimum process improvements can only take place if the "right" information is available to detect process weaknesses. This may at times require collection of "non-standard" metrics.

Non-Standard Metrics can include any data necessary to understand a perceived problem. They can include such measures as the length of time for an engineer to complete or learn a sequence of process steps perceived to be inefficient. Non-Standard metrics are only gathered for the period of time necessary to isolate a problem, and implement a solution. Once resolved, for efficiency, collecting of non-standard metrics should cease.

On the B-2 ATD, in response to a problem report on the build process efficiency, the following non-standard metrics were collected over a period of several months and analyzed weekly:

1. Elapsed wall clock time of each load
2. Number of changed software units per load
3. Number of lines compiled
4. Number of tasks linked
5. Number of load build process problems
6. Categories of build process problems
7. Elapsed time of segments of build process.

As a result of this analysis, a process improvement plan was initiated. Lessons, rules, and guidelines resulting from this activity were communicated to the company Process Action Team for approval and incorporation into the company standard process. It was the collection of specific non-standard metrics on a project that provided the needed insight leading to key process improvements for the company.

SOFTWARE CONFIGURATION MANAGEMENT

Earlier in this paper key areas targeted by the Company Process Action Team were identified. One area identified was Configuration Management.

Interviews with engineers indicated that certain projects may have applied too much control too soon resulting in unnecessary process inefficiency. As a result, the Process Action Team collected data, examining a sampling of systems across multiple programs. We found that certain systems had been placed under configuration control prematurely and some programs were requiring too high a level of change approval too early. Releasing software before it was adequately tested was found to be the result of inadequate education and training concerning the relative cost impact of detecting errors prior to (versus after) release.

The Cost To Detect and Fix Errors

The cost to detect and fix errors during integration is significantly greater than the cost to detect these errors prior to release.

When design errors are not detected until integration, the impact is great for the following reasons:

1. Load build time is extended. This affects many engineers.
2. The rigor of the change process (approvals, etc.) is more costly.
3. Having software in the load that has not been fully tested can cause testing rework to interfacing systems.
4. Possibly the most significant, frustration caused by all of the above leads to low engineering morale.

Company Action Plan

The Process Action Team determined that the root cause of reported inefficiencies

was not the company software configuration management policy or procedures, but rather a misapplication of the process and inadequate training and education in the consequences of premature release and excessive early controls. This education became a key part of our formal software engineering training program.

It was the key feedback from metrics that initiated the actions leading to these key improvements in our process.

PROJECT AND CORPORATE INTERACTION

Project feedback to a company focal point is critical to effective company level process improvement and reuse.

As a result, a corporate level Software Engineering Process Group (SEPG) was permanently established at Link. The SEPG listens to project concerns and lessons, approving, where appropriate, Company software process changes. Any changes to the Company standard software process and supporting tool-set must be approved by the SEPG. See Figure 2.

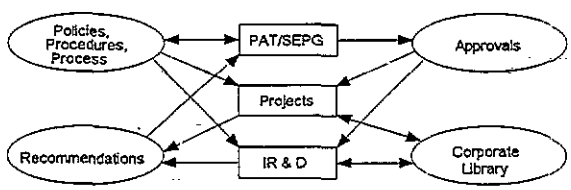

Figure 2 Project and Corporate Interaction

SEPG decisions are based on the company software vision. Software process changes or tool modifications are approved only if the change is in the best interest and consistent with the Company software principles and vision.

In the past, individual projects have modified software tools and processes based on shortsighted project issues. This resulted in overall increased software development and training costs.

Each directorate within CAE-Link with software responsibility has a representative on the company SEPG. It is through this vehicle that we are affecting the changes necessary to make reuse technology integral to our software development process.

WHERE ARE WE GOING?

Today we are recognizing the needs of the full life-cycle of software. We are using front-end analysis and design tools and object oriented techniques to better communicate with our customers and ourselves. These techniques and tools also provide more reusable software.

Reuse techniques, methods, and tools are certain to play a key role in reducing future software costs, allowing us to do more for less. We envision a reuse library integrated with configuration management systems providing disciplined and efficient software processes supported by metrics and feedback leading to continual process improvement.

We are positioning ourselves to grow through open standards and commercial off-the-shelf software products. We are moving toward improved off-line test environments supporting earlier less costly error detection.

Today, all the tools required to support our vision are not yet commercially available. We plan to support our company process with the necessary tools developed and managed through the corporate library and the SEPG.

As improved products become commercially available we intend to maximize our use of commercial products supporting open standards.

BALANCING STANDARDIZATION AND GROWTH

Companies that fail to change will not survive. At the same time, too much change or change of the wrong kind can be equally detrimental to the success of a software organization.

Each company must establish its own vision of the future with clear software objectives and change guided by its own objectives.

CONCLUSIONS

Early reuse programs will not be capable of addressing all associated issues. Companies seeking to institute reuse programs must look closely at their own process and products to determine where the greatest gains are to be made and how to best integrate reuse technology into current software cultures.

Successful reuse initiatives today demand a selective and focused strategy coupled with management mandates, training, and education.

Software reuse is not limited to our products. Perhaps the greatest potential for cost savings is found in the reuse of efficient processes. The path to better processes and methods is through metrics.

Without measurement, we tend to work bottom-up with decisions being based on nearsighted perceptions. It is frequently these decisions that produce the products that are difficult to reuse and maintain.

Effective reuse programs must be integrated with process improvement continually providing feedback through metrics. Process metrics should be viewed from the company perspective. We need to change to survive, but change must be consistent with our principles and our vision.

REFERENCES

1. Wiederhold, Gio, & Wegner, Peter & Ceri, Stefano (Nov. 1992). Toward Megaprogramming. Communications of the ACM.