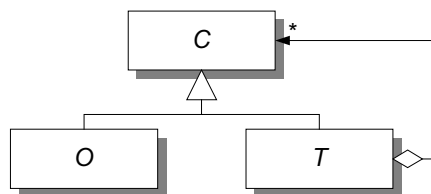


Plan for Introduction of Systematic Reuse
at
Systematic Software Engineering A/S
COT/3-24-V1.2



Centre for Object Technology

*Centre for
Object Technology*

Revision history:	V0.1	19-02-1999	First version
	V0.2	13-06-1999	Submitted for internal case 3 review
	V1.0	28-06-1999	After internal case 3 review
	V1.1	28-09-1999	Incorporating HPJ and OV review comments
	V1.2	12-11-1999	Minor corrections before submission for final approval

Author(s): Henrik Røn, Systematic Software Engineering A/S
Thomas J. Hansen, Systematic Software Engineering A/S

Status: Final

Publication: Public

Summary:

This document consists of two parts:

- A primer on reuse introducing basic reuse concepts and summing up a some of the experience in the reuse field over the past fifteen years.
- An overview of the project methodology and Systematic Software Engineering's plan for introduction of systematic

© Copyright 1999, 2000 Systematic Software Engineering A/S

The Centre for Object Technology (COT) is a three year project concerned with research, application and implementation of object technology in Danish companies. The project is financially supported by The Danish National Centre for IT-Research (CIT) and the Danish Ministry of Industry.

Participants are:

Maersk Line, Maersk Training Centre, Bang & Olufsen, WM-data, Rambøll, Danfoss, Systematic Software Engineering, Odense Steel Shipyard, A.P. Møller, University of Aarhus, Odense University, University of Copenhagen, Danish Technological Institute and Danish Maritime Institute

1. INTRODUCTION	4
1.1 EXECUTIVE SUMMARY.....	4
1.2 DEFINITIONS AND TERMS.....	5
1.3 ORGANISATION OF THE DOCUMENT	5
2. DESCRIPTION OF SSE.....	5
2.1 ORGANISATIONAL LAYOUT	5
2.2 DEVELOPMENT.....	6
3. SYSTEMATIC REUSE – A PRIMER.....	6
3.1 BENEFITS	6
3.2 MISCONCEPTIONS AND PITFALLS	7
3.3 FACETS OF REUSE	10
3.4 LEVELS OF REUSE	11
3.5 INVESTING IN REUSE.....	12
4. CURRENT SITUATION AT SSE	13
4.1 REUSE BETWEEN PROJECTS	13
4.2 MOTIVATION FOR IMPLEMENTING A REUSE SCHEME.....	14
4.3 GOALS AND STRATEGY OF SYSTEMATIC REUSE PROJECT	14
5. PROJECT METHODOLOGY.....	17
5.1 INTRODUCTION TO IDEAL.....	17
5.2 ADAPTING IDEAL FOR SYSTEMATIC REUSE PROJECT.....	19
6. ITERATIONS.....	21
6.1 PROJECT INITIATION	21
6.2 1 ST ITERATION: GROUNDWORK AND PILOT	22
6.3 2 ND ITERATION: GATHERING AND DISSEMINATING.....	24
6.4 3 RD ITERATION: REUSE IN THE DEVELOPMENT PROCESS	24
6.5 4 TH ITERATION: MULTIPLE PROJECTS AND DEPARTMENTS	25
6.6 5 TH ITERATION: IMPROVE METRICS AND PROCEDURES	26
6.7 6 TH ITERATION: REUSING A BROADER RANGE OF ASSETS	26
7. REFERENCES	28
A. ABBREVIATIONS AND CONCEPTS.....	30
B. SYSTEMATIC REUSE PROJECT CORNERSTONES	30
B.I TOOL SUPPORT FOR REUSE.....	30
B.II DOCUMENTATION	31
B.III GUIDELINES	31
B.IV ORGANISATIONAL ISSUES AND PROCEDURES	32
B.V METRICS	32

1. Introduction

This document is an extract of an internal Systematic Software Engineering A/S project management plan for the introduction of systematic reuse. Throughout the rest of this document Systematic Software Engineering A/S will be abbreviated to SSE in order to avoid confusion with the term “systematic reuse” and the project to introduce reuse in SSE, the Systematic Reuse Project, will be referred to as the SR project.

“Reuse” is the process of creating, publishing, and utilising software products in more than one context. The most basic example is the reuse of *source code* from one program in another. Another example would be the reuse of a specific *architecture* (design), which has proven itself useful in one project, to develop similar software in another project.

“Systematic reuse” is doing reuse in a deliberate, well-defined, and well-organised manner [Jacobson et al. 97b]. The SSE interpretation of this is that “systematic reuse” is the act of sharing (or using) software development *assets* produced in one project with another project in a controlled and managed way.

The document is divided into two parts: A primer on reuse and the project plan itself. The target group is anybody with an interest in systematic reuse, but the primer is aimed at people with no or little knowledge about systematic reuse. The project management plan can be of interest and inspiration to anybody how is contemplating introducing systematic reuse in their software process. By “software process” we mean the entire life-cycle of software and surrounding activities.

1.1 Executive Summary

The benefits of reuse are well-known, i.e., shorter time-to-market, better quality, and lower development cost. Many attempts to implement reuse have failed because the focus has been on technical issues rather than organisational and procedural issues. Within the last ten years, however, the number of success stories has increased as the organisational issues have become better understood. This document lists a number of reuse misconceptions and pitfalls that are widespread in the software world today. Introduction of reuse represents a substantial investment and will affect all parts of an organisation and its software process. It furthermore requires strong commitment from the upper management.

Reuse has two basic facets:

- Reuse can be either *vertical* (domain specific), e.g., a framework for Command Control Information Systems (CCIS) within the defence sector, or *horizontal* (generic across domains), e.g., a framework for graphical user interfaces.
- The approach to reuse can either be *generative*, i.e., applications are generated from a specification using a tool, or *compositional*, i.e., applications are constructed using reusable assets that are assembled manually to form an application.

Furthermore the reuse maturity of an organisation can be classified according to five levels ranging from an organisation with uncontrolled opportunistic reuse to an organisation, which is driven by domain-specific reuse.

Some form of systematic reuse will be implemented at Systematic Software Engineering A/S using a problem-oriented, incremental, and iterative methodology. The work method is based on the IDEAL model [McFeeley 96] for software process improvement. Reuse will be introduced in the organisation gradually initially in order to avoid technical and organisational complexities not relating to the core issues of reuse. The technical complexities are multiple hardware platforms, compilers, and implementation languages. The organisational difficulties include (a) knowledge management issues, such as the accumulation and dissemination of knowledge about reusable assets, (b) incorporation of reuse in the software process by establishing policies, procedures, and guidelines for reuse, and (c) resistance from employees to the new policies, procedures, and guidelines.

1.2 Definitions and Terms

“Work product” or “product”	The products or by-products of the software development process.
“Reusable asset”	Synonym for a work product that is considered reusable.

1.3 Organisation of the Document

Section 2 contains a description of SSE as an organisation and is followed by Section 3, which is the primer on reuse. Then in Section 5 the project methodology, which is incremental and iterative, is described, and Section 6 describes the iteration planned initially, i.e., how it is planned to implement systematic reuse. Section 7 contains the references used in this document.

2. Description of SSE

This sections contains a short description of SSE. It is divided into the following sub-sections:

- **Organisational layout.** Description of how the organisation is divided into departments.
- **Development conditions.** Description of the current reality in terms of project size, hardware platforms, programming languages, etc.

2.1 Organisational Layout

Software development is in SSE organised into three departments:

- **Product Department.** The product department is responsible for development and maintenance of the SSE commercial-of-the-shelf (COTS) products.
- **Defence Project Department.** The Defence Project Department is an umbrella organisation for all projects within the defence sector.

- **Industrial Project Department.** The Industrial Project Department is an umbrella organisation for all projects within the public non-defence and private sectors.

Furthermore there are two non-development departments:

- **Finance and Administration.** Handles economy and secretarial tasks.
- **Business Process Improvement.** Responsible for internal support, but also handles SSE's Software Process Improvement (SPI) and Knowledge Management (KM) projects.

2.2 Development

Several factors make it more difficult to implement systematic reuse:

- **Great variation in manning and project size.** The projects carried out at SSE vary greatly in size both in terms of the number of developers and the size in man-months. The manning in projects varies between 1 and 20 and the size in man-months varies between 2 and 480.
- **Multiple hardware platforms.** Development is done on many different platforms including Wintel, Solaris, Solaris Intel, HP, AIX, and SCO UNIX.
- **Multiple programming languages.** The languages used include C, C++, Java, Object Pascal, and Visual Basic.
- **Multiple development environment.** The development environments include Visual Studio, Delphi, and Emacs combined with the Gnu GCC/g++ compiler.

3. Systematic Reuse – A Primer

Reuse has become the Holy Grail of modern software engineering. While it holds promises of great benefits, it is also surrounded by a number of myths and misconceptions. Based on experiences from other attempts with systematic reuse, this section aims at providing a clear definition of what systematic reuse is, which benefits can be gained, and which pitfalls exist. The information in this section does not represent new research results, it is a limited overview of knowledge accumulated in this field over fifteen years that the authors deemed relevant for introduction of reuse in SSE. The information is in some cases related to examples from SSE.

3.1 Benefits

The three long-term benefits of systematic reuse are reduction of these parameters:

- **Application development cost.** If a library of reusable assets is established for a domain¹ and if these assets are used in projects then development time will be shorter.
- **Time to market.** Lower development cost is due to a lower number of man-hours needed to complete a project and therefore time to market is shorter.

¹ Like, e.g., military command and control information systems, electronic commerce, or financial transaction systems.

- **Defect rate.** As the assets have been used before most of the problems and bugs have been weeded out. Therefore the number of bugs in the application should be significantly smaller than the number of bugs in an application developed from scratch.

This is illustrated in the figure below.

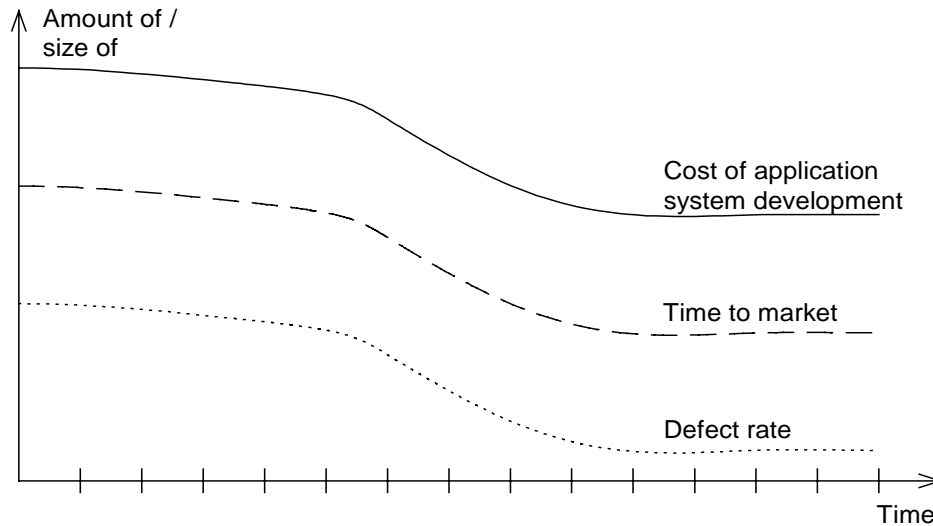


Figure 1 - Long term benefits of reuse [Jacobson et al. 97b]

3.2 Misconceptions and Pitfalls

There exist many misconceptions in the field of (systematic) reuse. Most of the issues below are from [Jacobson et al. 97a] and [Goldberg and Rubin 95]. For more definitions of reuse concepts and terms than in this report, see [Lim 98].

3.2.1 Company Level

- **Reuse is mainly a matter of introducing the right technology.** All the studied sources mention this misconception as the pitfall of reuse. Reuse is more a matter of changing the organisation and the work processes than introducing new technology within the existing organisation and work processes.
- **Insufficient managerial backing.** Introducing reuse in an organisation is a long and hard process with a substantial initial investment and a risk of failure that should not be underestimated. All this in combination can at some point during the process discourage management from retaining its backing.
- **Reuse does not work.** Establishing reuse is difficult, not impossible. Many companies have done so and published their experiences. Some have failed, and fewer have failed and published their problems and experiences. But still the experiences are out there, find them and learn from them.
- **Developers will reuse if assets are available.** A major obstacle to systematic reuse is the “not-invented-here” syndrome: Developers do not reuse because they do not trust code developed outside their own project group. Trust must be established and quality must be ensured.
- **It is too expensive.** Establishing systematic reuse is expensive initially, but will pay off over time as illustrated below.

- **Reuse should be deferred until having reached CMM level 3.** Many of the areas needed for reaching level 2+ in the Capability Maturity Model (CMM) are also prerequisites for introducing systematic reuse, amongst others: configuration management, requirements management, quality assurance, and having a well-defined software process. A systematic reuse project can be started without a software process improvement project, but the two can benefit greatly from being performed in parallel as many activities and areas have to be addressed in both projects in order for them to succeed.
- **Reuse will work without a standardised reuse process.** If the software process does not incorporate procedures and guidelines for reuse it makes it more difficult to establish a systematic reuse program. There *must* be a framework for reuse within the software process to assist and support the developers in their daily work.
- **Reuse should be limited to code components.** Several sources suggest that reusable assets other than code are just as valuable (or more). These assets could be document templates, system design documents, detailed design documents, requirement specification. The important thing when reusing is to go for quality rather than quantity to avoid the not-invented-here syndrome.
- **Reused code is too slow.** If it is too slow, it is a design flaw and the code may have to be redesigned to ensure better performance. This is an example where reuse shows its power, as all users will benefit from this redesign. Reusable assets should be developed under even more rigorous requirements than “ordinary” projects.
- **Setting up a library will automatically get developers to reuse.** This does not work. Developers need to trust the quality of the reusable assets. There needs to be a rigorous quality assurance on reusable assets to ensure a quality that developers will trust. This is necessary to avoid the “not-invented-here” syndrome.
- **Developers can easily select components from a common repository.** Setting up a common repository or database containing the reusable assets and supplying functionality for keyword search is not nearly sufficient. There is a great difference between how the constructor describes the reusable assets and how the reuser describes the asset being looked for. This is explained in more detail in [Henninger 94].
- **Paying developers for contributing to the library will build the library.** Experiences from several sources have shown that this policy does not work. If it has any effect, it is a negative effect on quality as there have been example of developers contributing worthless assets to the library. This can of course be avoided with rigorous quality control.
- **Paying developers to reuse will encourage reuse.** Experiences from this have been negative with examples of developers reusing large chunk of code just to reuse it, not because it was needed. This is of course an extreme example, but as mentioned above reuse should be a natural and integrated part of the software process.

3.2.2 Project Level

- **Allowing project expediency to take priority over reuse and development of reusable assets.** If the projects are allowed to disregard developing for reuse when they are under pressure, there is distinct risk that reuse will be disregarded leading to poor reuse. Reuse *must* be an integrated and mandatory part of the development process if systematic reuse is to be accomplished.
- **Expected reuse from previous projects can always be predicted.** It is very dangerous to assume that the amount of reuse always can be predicted. One should

be wary of setting objectives for reuse for any project that is “First-of-its-kind” or “Legacy-rewrite”. Objectives for reuse for “Done-this-before” or “Variation-on-a-theme” projects have a much larger chance of being correct. The similarity of “Legacy-rewrite” and “Done-this-before” is that it is the same domain. The difference is that “Done-this-before”, in our opinion, implies that a similar system has developed before using a process model involving systematic reuse, i.e., essentially the same model as being used currently. “Legacy-rewrite” on the other hand implies that the legacy system was developed using a process model not involving systematic reuse. It is the old maxim that it is always easier to estimate effort for a well-known domain or business area than for a new and unknown.

- **Reuse can be proved prospectively.** It is not possible to estimate beforehand how much of the newly developed code in a project that will be reused. It can only be demonstrated how much has been reused, it is not possible to “prove” how much will be reused. Domain analysis does to some extent identify prospects for reusable assets, but these are only prospects. They can only be considered reusable, when they have been reused, i.e., domain analysis gives an *estimate* of how much can be reused, it does not *prove* how much can be reused.
- **Architecture does not have to be introduced as a specific task.** When assets are to be reused, the architecture of a system becomes extremely important as it defines the structure of the system and the interfaces between the different components in the system, which is what the reusers are dependent upon when reusing the asset.

3.2.3 Reuse and Object-Oriented

While OO holds the promise of supporting systematic reuse the introduction of OO simultaneously with systematic reuse can make it more difficult to establish systematic reuse. The following pitfalls are from [Fichman and Kemerer 97]

- **Adopting object-orientation will lead to systematic reuse.** Object-orientation (OO) may support reuse as OO languages have several features suited very well for developing reusable software:
 - *Encapsulation.*
 - *Inheritance.*But OO will *not* lead to reuse without an integrated, supporting framework in the software development process.
- **OO itself takes time to learn.** When introducing OO it takes time for developers to get familiar with the concepts through learning-by-doing and even longer for developers to become good designers as this is the most difficult part of OO.
- **Neglecting OO analysis and modelling.** Some sources view systematic reuse as most effective within a domain, e.g., CCIS, rather than across several domains. If domain based reuse is employed, then the analysis and design phases become the foundation for later reuse and therefore an absolute necessity. If the analysis and modelling is done superficially then architecture of the system will not be stable thus making reuse harder.
- **Providing no library function.** If it is believed that OO automatically induces reuse there could be a tendency to neglect the establishment of a library of reusable assets. This is lethal to the establishment of a framework for reuse as assets must have a home of some kind.

3.3 Facets of Reuse

Two facets of reuse are:

1. Which approach is used to construct systems comprised of reusable assets constructed?
 - **Generative.** System construction is “accomplished via the use of application generators to build new applications from high level descriptions” [Ogush 92]. In [Bassett 96] there is a thorough coverage of such a generative approach.
 - **Compositional.** System construction is accomplished by assembling existing reusable assets. This approach is most often associated with repositories of assets and standard interfaces.
2. What is the domain scope?
 - **Horizontal reuse.** Reusable assets are used across different business domains. Examples of such horizontally reusable spell-checkers, framework for database access, and a gray-box [Lim 98, p. 397] framework for searching and filtering [COT/3-20]. This is also called generic reuse.
 - **Vertical reuse.** Reusable assets are only used in the business domain in which they were developed. Domain specific reuse is developing for reuse within one specific domain, e.g., CCIS. A domain analysis is performed initially and it is more general than a system analysis. The domain analysis covers the entire domain, i.e., it covers several projects within the domain while the system analysis focuses on a specific project and its requirements. The main objective of the domain analysis is to discover which parts of a family of system are the same, *commonalities*, and which parts that vary from family member to family member, the *variabilities* [Coplien et al. 98]. The domain analysis is updated after each project within the domain to cover insights gained during the project.

In [Lim 98, p. 397] several other facets are listed, but these are outside the scope of this report. A number of sources describe domain specific reuse as the most promising and profitable form of reuse for several reasons:

1. **The reusable assets tend to be bigger.** The domain specific assets are identified during domain analysis and correspond to entities in that domain. Examples of domain specific assets could be parsers for different message formats or a workflow manager. Domain specific assets tend to have a higher granularity than generic assets because they represent an entity in the problem domain and therefore their size corresponds to the size of the entity in the domain.
2. **It is easier to develop domain specific reusable assets.** As the domain specific assets only have to be used in one domain they do not have to be as general and versatile as the generic reusable assets, which can be used in any domain.
3. **There are fewer generic reusable assets.** Experience has shown that there only are a limited number of generic components and that these are hard to develop as each user will have specific needs.

A system can of course be constructed using a both vertical and horizontal reusable assets.

3.4 Levels of Reuse

In [Jacobson et al. 97b] the experiences from introducing reuse at Ericsson and Hewlett Packard are described. The authors advocate an incremental approach to introducing reuse as shown below:

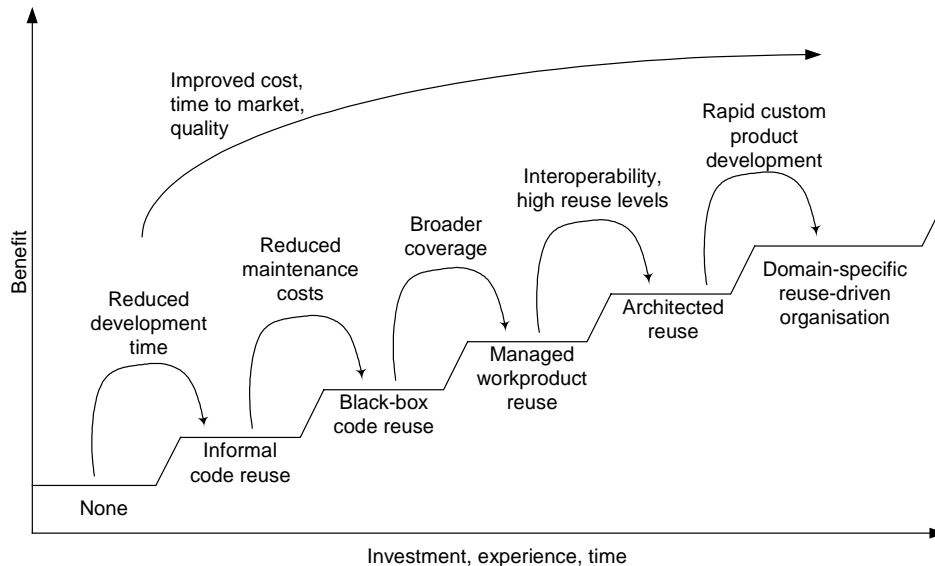


Figure 2 - Incremental introduction of reuse at HP [Jacobson et al. 97b]

Each level has its own characteristics:

- **None.** No reuse whatsoever.
- **Informal code reuse.** Often called cloning or leverage. Code is copied, adapted, and modified but these modifications are not merged into the code originally copied, i.e., each defect will have to be fixed in each of the clones, i.e., multiple times.
- **Black-box code reuse.** A few carefully chosen instances of code are reengineered, tested, and documented for reuse. All projects use the code without modifying or adapting it.
- **Managed work product reuse.** On this level the creation and reuse of reusable assets is explicitly managed and supported by a distinct organisation.
- **Architected reuse.** Assets and systems are explicitly architected to ensure that they fit together. This involves introduction of domain analysis to describe the business domain of the project.
- **Domain-specific reuse-driven organisation.** Here the entire software process has been modified by encompass reuse and the company has established a number of business domains where domain analysis and domain engineering are used to construct reusable assets. For each project within a domain it is decided which parts should be build using reusable assets, which should be developed from scratch, and which should be developed for reuse.

The model above is but one of many models for maturity regarding reuse, see [Lim 98] for other models. It should be noted that the above model is not about going through predefined steps to reach the next level as for example the Capability Maturity Model (CMM) does. Furthermore a company can be at several levels at the same time. SSE for

example is generally on the “Informal code reuse” level, but some initiatives for “Black-box code reuse” can be found several places within the organisation.

3.5 Investing in Reuse

Initially, reuse requires an investment, which will be returned in the course of time. The initial investment is necessitated by a number of key activities:

- **Define reuse architecture.** The analysis of the existing organisation and the design of new procedures, workflow, etc. This is a one-time investment, but it is time-consuming.
- **Reorganise organisation.** The organisational structure has to be adapted to meet the new requirements of the reuse process.
- **Analyse business domain.** It is an initial investment to analyse the business domains for the projects where reuse is established.
- **Train employees.** To achieve the best level of reuse the employees should be trained in how to design and program for reuse.
- **Create reusable assets.** It requires extra effort to create reusable assets especially during the initial establishment of the library.
- **Employees learning curve.** Creating “normal” assets can be difficult; creating reusable assets is even more difficult. Of the programming related activities design is probably the most difficult and most challenging.

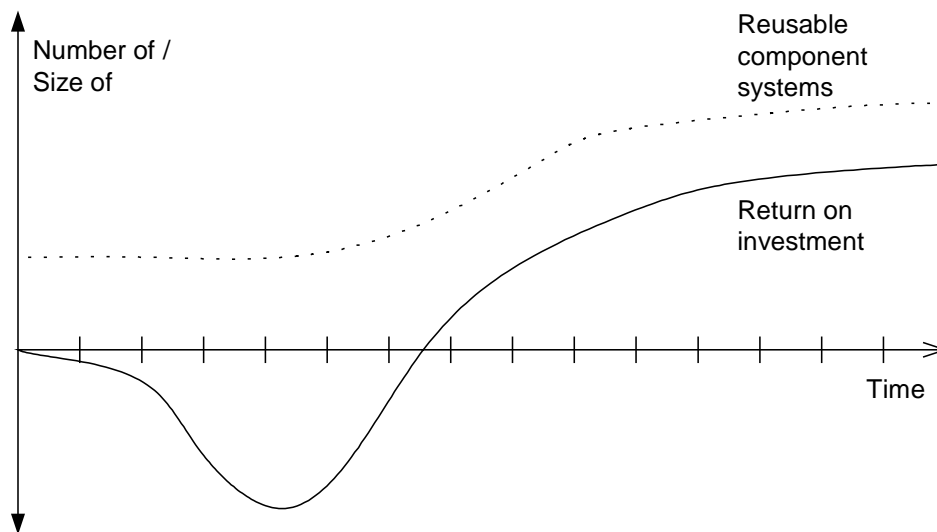


Figure 3 - Return on investment [Jacobson et al. 97b]

When the organisation is transformed during the transition to systematic reuse, tension within the organisation will rise initially as the transition can make employees insecure, dissatisfied with new business structure etc., but tension should decline again over time, hopefully to a lower level than before.

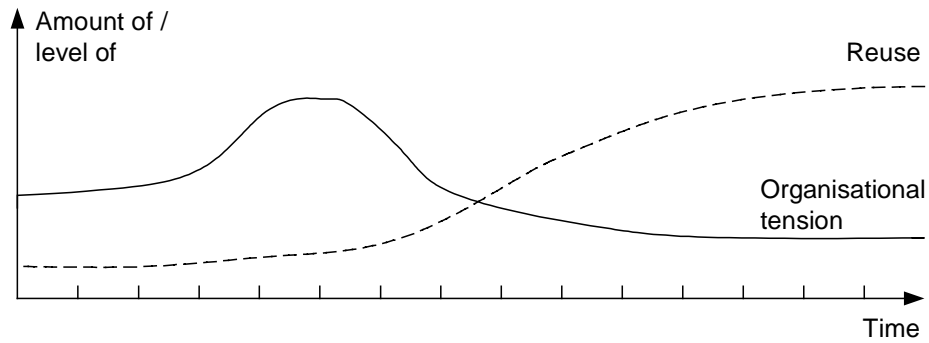


Figure 4 - Tension during transition [Jacobson et al. 97b]

In the literature the establishment of some kind of parts department is recommended by most sources in the literature, amongst others, [Jacobson et al. 97, Goldberg and Rubin 95]. The parts department, called “domain engineering organisation” by Jacobson et al. and “expert services” by Goldberg and Rubin, is the *producer* of reusable assets, and the projects are the *consumers* of reusable assets. There are several reasons for this partitioning:

- Explicit allocation of resources keeps the parts department free of delivery pressure as projects often are. If reuse experts are part of a project team then the project work will often take precedence over the reuse work.
- Some developers will be experts in constructing reusable assets, others will specialise in assembling applications from reusable assets.

4. Current Situation at SSE

The situation in the projects range from a) no reuse at all through b) copying and modifying to c) a limited form of managed reuse. Some of the past problems have been lack of version control and procedures for management of code. As these problems were remedied new problems were discovered, such as lack of overview and documentation.

4.1 Reuse Between Projects

The reuse between projects is mostly based on hearsay where one developer in the reusing project has heard something about code used in another project. The code is then usually copied to the reusing project’s work space and modified there, i.e., pure copy’n’paste (also called cloning) reuse. This form of reuse has several severe drawbacks:

- **Separate versions.** The same code exists in each of the projects in which it is used, i.e., it exists in two (or more) versions within the company.
- **Bug fix.** When a bug is discovered and fixed in the code of one project, it also has to be fixed in the other project.
- **New functionality.** When one project adds new functionality, this functionality will not be available to the other project(s).

In our opinion there are several reasons why things are done this way:

- **Easiness.** The reuser takes responsibility for the copied code, which places no extra responsibility on the provider.
- **Not invented here.** Although the developers in general are flexible and open-minded there are still sign of the “not invented here” syndrome. At SSE the “not-invented-here” syndrome manifests itself in the form of “this is not the optimal solution for my needs, I can develop a better one”. The developers often discard a 80 percent solution which has been implemented and tested, in favour of a 100 percent solution which they implement themselves.
- **Independence.** The reused code becomes independent of the project in which it was developed.
- **Risk control.** It is easier for the reusing project to control risks if it has full control and responsibility over the reused code.
- **Cheap initially.** There are no extra costs connected to this form of reuse, i.e., it is initially cheap, but will gradually cost more and more.

4.2 Motivation for Implementing a Reuse Scheme

At SSE the following incentives for establishing a systematic reuse scheme exist besides the traditional benefits mentioned previously.

Employee satisfaction poll. During a recent poll of employee satisfaction the employees were asked to rate on a scale from one to five i) the importance they attributed to the statement and ii) their estimate to which degree the statement reflected reality. Two of the questions were:

- a) “We learn from the experiences from other projects/departments”. Attributed importance = 4.2 and reflection of reality = 2.3
- b) “We reuse code and components”. Attributed importance = 3.8 and reflection of reality = 2.7

These statements were both in the top three in terms of discrepancy between attributed importance and reflection of reality. This indicates that this area is ripe for change and that the employees consider this area as being important.

4.3 Goals and Strategy of Systematic Reuse Project

The SR project aims at promoting reuse of software *components*, software *frameworks*, and software *architecture* within and among SSE’s projects. The strategy is *incremental*, *bottom-up* introduction of reuse in the development process, project management, and the company culture by *participatory implementation* where a regular SSE project is involved in each iteration. Throughout the project, a number of project *cornerstones* will server to focus the attention on critical areas within the domain of software reuse.

4.3.1 Components, Frameworks, and Architecture

By *components* we mean reusable pieces of software, characterised by low context dependencies, i.e., pieces that can be reused in a wide variety of software products. The goal is to promote systematic development, publication, and reuse of components inside and among projects.

By *framework* we mean a platform on which software products within the same domain can be based. A framework will typically provide a kind of infrastructure or foundation of services for the software products. The goal is to promote systematic development, publication, and reuse of frameworks inside and among projects.

By *architecture* we mean "...the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them" [Bass et al. 98, p. 23]. Reusable architecture provides proven solutions to reoccurring problem in analysis, design, and implementation of software products. Generalisation describing a set of such solutions are often called and are also called architectural patterns [Buschmann et al. 96] or architectural styles [Shaw and Garlan 96], [Bass et al. 98]. However less general but still reusable architectures are also feasible. The goal is to promote systematic development, publication, and reuse of architecture inside and among projects.

4.3.2 Incremental, Bottom-Up Approach

By incremental we mean gradually achieving higher and higher levels of systematic reuse, but also that the project gradually should cover larger and larger parts of the organisation. The primary reason for selecting an incremental approach is that promotion of systematic reuse necessitates changing the behaviour of the organisation, and change is easier to achieve in steps. Also, it is not possible precisely to describe *how* systematic reuse can best be promoted in SSE. Hence, experimentation is needed.

By bottom-up we mean starting by addressing systematic reuse issues close to the software implementation process, moving upwards towards project planning and possibly organisational restructuring.

4.3.3 Participatory Implementation

A major reason for selecting a bottom-up approach, is that one of the key factors in succeeding with reuse is *motivated participants*. By first addressing reuse issues which are recognised by the participant as problems, the *purpose* and *usefulness* of promoting reuse becomes self-evident. More specifically, we hope to ensure that:

1. the participants are motivated for changing current procedures and old habits regarding software development. A crucial step is overcoming the "not-invented-here" syndrome.
2. the quality of participation will be high, given that the involved individuals know the problems first hand.

It can not be stressed strongly enough how important for the success of the SR project it is to involve the regular SSE projects. Benefits of this are testing of tools and methods in a real setting, dissemination of knowledge into the entire organisation, and hopefully immediate feedback.

4.3.4 Cornerstones

Given the experimental nature of the SR project, it is difficult to foresee the exact sequence of activities, which will lead to a successful completion of the project. Instead, a number of project cornerstones have been identified. It is the strategy of the SR project to try to achieve the project's goals by focusing on developing these areas:

- **Tool Support** - It is the intention to develop tools to support the workflow surrounding a library of reusable assets. As an example, it must be possible for a developer to search the library for reusable components that match the requirements of his current project or problem. However experience has shown that this should not have too much focus as the number of assets tends to be small and does not warrant a special purpose built repository. This is discussed in [Doublait 97].
- **Documentation** – One of the keys to successful reuse is the right documentation of the reusable assets. It is the intention to develop and test efficient documentation standards.
- **Guidelines** – It is the intention to produce guidelines for use and construction of reusable assets. As an example, a guideline might explain how software components could be designed with reuse in mind but without introducing significant overhead.
- **Organisational Issues** – It is the intention to produce input to the ongoing Software Process Improvement (SPI) activities. To incorporate reuse properly in the development process, process descriptions and concrete procedures may have to be altered.
- **Metrics** – It is the intention continuously to develop and refine metrics to measure the impact and degree of reuse in the participating projects. This way, statistic material will be available to track the progress of the SR project and to support planning of succeeding iterations. The standard reference on reuse metrics is [Poulin 96].

Please refer to appendix B. for a detailed description of the cornerstones.

4.3.5 A Reuse Scenario

The following example shows what the software process might be when systematic reuse has become part of the everyday routines at SSE. The scenario is included to give a rough idea of how things *could* be, *not* necessarily how they *will* be. The scenario could more or less fit any of the three Departments in SSE. The activities described below are generally handled by the projects with assistance and guidance from one or more company reuse experts.

Already during the preparation of the tender reuse is considered and it is estimated approximately how much will be reused in the project. As mentioned earlier this approach is only feasible for projects within well-known domains. The library is scanned for assets relating to the project domain and a domain analysis is retrieved. Using the domain model, i.e., the architecture for the domain, the components relevant to the project at hand are identified. It is checked that the identified components match the project requirements. It is investigated which of these components are a) present in the library and can be used as-is, b) present in the library but must be expanded, c) should be developed from scratch for reuse, and d) should be developed from scratch, but not reused. The distinction between a) and b) happens during the system and detailed design phases described below.

During the requirement specification it is ensured that a) the requirements for the system do not conflict with the requirements for the components one intends to reuse. The requirements for the reusable components are described in a separate requirement specification which of course still is part of the requirement specification for the project. The requirements for the reusable component are described in a separate document so

that they easily can be added to the documentation when the component is added to the library.

The system design phase then deals with the integration of the reusable components and how the other parts of the system should interface with these components. If the system design phase identifies the need for adjustments to the domain model these are incorporated in it. A rough API for the components that will be implemented for reuse in the project should be part of the System Design Document (SDD). In addition to the normal review a special review for the proposed reusable components is carried out as part of quality assurance. If the system design of one or more reusable components is rejected the components are degraded to being normal non-reusable components.

The system design phase is succeeded by the detailed design where the components are split up into sub-components and the API is defined in detail. This is done especially thoroughly for the reusable components that are to be developed. Again the Detailed Design Document (DDD) is subject to a special reuse review to ensure quality.

Then the design is implemented and tested. As part of the project follow-up the reusable components are made independent from the rest of the implemented code, documented, and moved from the project repository to the library containing the reusable assets. This work should be done in co-operation between the developers from the project and the company reuse expert(s). The resources for this work should come from a Department independent resource and not from the specific projects. Furthermore candidates for reusable components are identified within the rest of implemented code. This is done because the participants, utilising the knowledge gained during the project, may have identified components suitable for reuse in hindsight.

5. Project Methodology

The SR project does not use SSE's customary project model, but the IDEAL model, which is basically an incremental, iterative approach for software process improvement, see [IDEAL] for a primer and [McFeeley 96] for a thorough treatment. The introduction contains a walkthrough of the phases in the IDEAL model and their sub-phases. This incremental and iterative methodology was chosen for several reasons:

- **We need to experiment.** The goal is clear, but the road is not.
- **Feedback loop.** The iterations form a feedback loop, where information and user feedback gathered during an iteration is used as input for the following iterations.
- **Industrial Ph.D.** As a part of the SR project is an industrial Ph.D. study, it should be possible to make changes in the project plans if an area requires special attention or is especially interesting.

IDEAL is also used in the other SSE improvement projects, i.e., SPI and KM.

5.1 Introduction to IDEAL

IDEAL was originally a model for software process improvement project based upon the Capability Maturity Model (CMM) [Raynes 98], but has been generalised for any type of improvement project.

IDEAL is designed to provide a systematic, well-ordered, practical framework for improvement projects and consists of five major phases:

I – Initiating. Laying the groundwork for a successful improvement effort.

D – Diagnosing. Determining where you are relative to where you want to be.

E – Establishing. Planning the specifics of how you will reach your destination.

A – Acting. Doing the work according to the plan.

L – Learning. Learning from the experience and improving your ability to adopt new technologies in the future.

Each of the five phases is made up of several activities designed to ensure that the improvement project is kept on track. The phases and activities are described below.

5.1.1 Initiating

The initiation phase is concerned with the groundwork for an iteration and consists of the following sub-phases:

- **Stimulus for change.** Identify reasons for changing the business.
- **Set Context.** How the effort in the iteration affects the business and how it is related to other improvement projects.
- **Build Sponsorship.** Establish how the work in the iteration is funded and who is the sponsor.
- **Charter Infrastructure.** Setting up how the improvement effort is managed and which infrastructure is needed.

5.1.2 Diagnosing

Building upon the previous phase this phase is concerned with deepening the understanding of the improvement area.

- **Characterise current and desired states.** The current and desired state of the area of improvement are described. The focus is on elements close to the improvement rather than the entire business process.
- **Develop Recommendations.** The second part of the diagnosing phase is concerned with developing recommendations on how to implement the desired state based on the current state.

5.1.3 Establishing

In the establishing phase a detailed work plan is developed based on the preceding phases. Specific actions, milestones, deliverables, and responsibilities are incorporated into an action plan.

- **Set Priorities.** The tasks of the work are prioritised.
- **Develop Approach.** Development of a strategy for accomplishing the work and identifying resource availability under consideration of both technical and non-technical factors.
- **Plan Actions.** A detailed implementation plan is developed. This plan includes schedule, tasks, milestones, decision points, resources, responsibilities,

measurement, tracking mechanisms, risks and mitigation strategies, and any other elements required by the organisation.

5.1.4 Acting

The actual implementation of the plan developed in the previous phases.

- **Create Solution.** Creating a first shot at a solution.
- **Pilot/Test Solution.** The first shot implementation is tested and evaluated.
- **Refine Solution.** Based on the test results and evaluation the implementation is improved.
- **Implement Solution.** The solution is rolled out in the organisation.

5.1.5 Learning

The learning phase is for reviewing the work carried out in the iteration and assess whether the goals state in the first phases has been completed.

Analyse and Validate. The work and results are analysed and it is assessed whether the objective of the iteration has been met.

Propose Future Actions. Based on the analysis and validation recommendations for future improvements are developed.

5.2 Adapting IDEAL for Systematic Reuse Project

This sub-section describes how IDEAL is interpreted in the context of the SR project. It consist of two parts:

- **The initiation model.** This consists of the I and D phases from the IDEAL model to state a vision and make a preliminary plan. These phases are performed on a larger and more general scale than it will be done in each iteration, where these phases will focus on a specific improvement area.
- **The iteration model.** For each iteration all the five phases in IDEAL will be carried out. The I-phase will be carried out for the specific area of improvement, not the entire project. The entire I-phase is carried out as resources to perform the work have to be allocated for each iteration.

After the initial phase the general approach is to have iterations that last three months succeeded by a follow-up phase lasting a month. This is illustrated in the figure below.

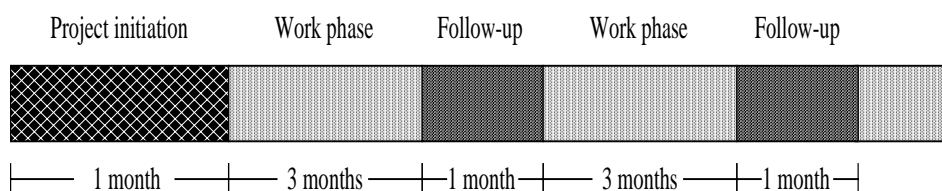


Figure 5 - The basic project model

The SR project does *not* follow IDEAL to the letter. We are following the basic intention of the model, and doing the same work as described in the phases and sub-

phases, we have merely structured the sub-phases a little different. The IDEAL model is the reference model for our own iterative model.

5.2.1 Initiation Model

The purpose of this period is to analyse the current situation at SSE carefully, and then write a requirement specification for the systematic reuse system, as envisioned at that point in time. It is the intention that time-boxing will be used in the project. The concrete activities are:

- **Define boundaries.** Define boundaries with other SSE improvement projects, i.e., KM and SPI.
- **Literature study.** Do limited literature study to get a approximate idea of state of the art in reuse.
- **Establishing funding.** What is the role of the SR project and how is it funded.
- **Characterise current state.** The current situation at SSE with respect to reuse is assessed. This assessment is carried out in order to know the starting point for the improvement project.
- **Describe desired state.** A goal for the SR project is outlined in a short document.
- **Write recommendations.** How should the outlined goal be reached through a number of iterations.

5.2.2 Iteration Model

In each iteration the five phases of the IDEAL model are carried out. In the following it is shortly described which activities we have assigned to the phases, i.e., what meaning we have attached to the sub-phases of IDEAL.

5.2.2.1 Initiation Phase

The “Initiation” phase is used to pinpoint a problem, study existing material on the problem area and state hypothesis for iteration, i.e., formulate a paragraph stating how the work done in the iteration will solve (address) a particular problem.

- **Stimulus for change.** Identify the problem that should be attacked in this iteration.
- **Set Context.** Study existing work in the problem area and state hypothesis for iteration.

5.2.2.2 Diagnosing and Establishing phase

The diagnosing and establishing phases have been joined and are concerned with establishing the current state within the problem area, describing the desired solution, and planning how to implement the solution.

- **Characterise current and desired state.** Describe current state of the problem area and how the identified problems will be addressed.
- **Prepare plan.** A plan containing a work break down schedule (WBS) for the work is written and approved.
- **Allocate resources.** The resources needed for an iteration are allocated according to the work plan. This also involves identifying participants in the iteration.

5.2.2.3 Acting Phase

The acting phase is concerned with carrying out the work described in the plan.

- **Create solution.** Not all the sub-phases described below are relevant for every iteration and may therefore be omitted.
- **Test solution.** Usually we try to use ourselves as guinea pigs before the solutions are tried in a broader scope, i.e., outside the SR project.
- **Refine solution.** Based on the results and experiences of the test the solution is improved.
- **Field solution.** The solution is fielded in a broader context.
- **Observe effect.** After the fielding data are gathered for the learning phase.

5.2.2.4 Learning Phase

- **Analysis and reflection.** The data gathered is analysed and worked through. If deemed relevant a report describing the iteration is written.
- **Update overall plan.** The overall project goal is adjusted to reflect the knowledge gained in the iteration.

6. Iterations

This section contains a description of the iterations carried out in accordance with the IDEAL model that is the project methodology for the SR project. The section is extended along with scheduling of new iterations.

Each iteration has its own subsection which contains purpose, description of involved cornerstones and the tasks within them, work products of the iteration, and a WBS.

After each iteration is completed, a project follow-up is carried out and an associated project follow up report is produced.

The first iterations will be carried out within the Product Department. A number of aspects of the Product Department make it an ideal subject:

- The amount of code (approximately 700KLOC) in the (IRIS Message Formatting System) IRIS/MFS project has become so large that it has become hard to maintain an overview of the different modules comprising the product.
- Informal reuse is already part of the everyday work in the Product Department. Migration to systematic reuse should yield greater benefits.
- There are three more or less independent development teams, each working on their own part of the system, and each team produces a number of potentially reusable components and functions, but has a hard time making them known to the other developers.
- The developers themselves brought up the problems at a follow-up meeting on a survey of employee satisfaction.

6.1 Project Initiation

Purpose:

Initiation of the project and formation of a base for the work in the iterations of the SR project.

- Project management plan for incremental introduction.
- Analysis of the current software process within the Product Department.
- Description of current roles in the software process.
- Description of future reuse scenario.
- Description of future roles in the software process.
- Initiating establishment of terminology.

Involved cornerstones:

None, as this step solely deals with start-up activities that form the base for the rest of the work.

- **Analysis of current situation in SSE.** The purpose of the analysis of the current state is to introduce non-SSE COT participants to the software process in the Product Department and at SSE. This base will be used to identify existing problems, obstacles and shortcomings.
- **Statement of a goal for SR projects.** The statement of an overall goal is essential because it is hard to control a project if there is no clear goal. The formulation of a goal involves modelling future work processes involving reuse and describing the future roles that act in the software process.
- **Plan for incremental introduction of reuse.** An overall plan for the introduction must be established. The plan describes how to accomplish the transition from the current situation, as described in the analysis, to the future goal. It will be updated continuously throughout the project as the goal gets clearer and experiences are accumulated in the iterations.
- **Reuse terminology.** The SR project groups, i.e., the monitoring and the work group need to establish a common terminology so that they can collaborate successfully.

6.1.1 Deliverables

Four documents will be the results of this iteration:

- This PMP forms the plan for the incremental introduction.
- An analysis of the current software process in the Product Department with focus on reuse. The analysis is a combination of descriptions of roles in the software process and scenarios describing the current software process.
- A description of the future software process with focus on reuse. It will also be a combination of future roles and scenarios describing the future software process focusing on reuse.
- Terminology document for the SR project. An important part of a project is the development of a common terminology.

All four items are not finished documents but dynamic working documents which will evolve over time as insights and experiences are gained.

6.2 1st Iteration: Groundwork and Pilot

Purpose:

The purpose of the first iteration is twofold

- **Groundwork.** Establishing the foundation for the next iterations. This includes establishing standards for documentation of reusable components (which may

involve modification of existing SDD and DDD template), design and implementation of a simple facility for storing and retrieving components.

- **Reuse of filter component in EWare.** The filter facility has been designed for IRIS/MFS, but with reuse in EWare as an objective. However the code must be made independent of IRIS/MFS and there are a few bad design decisions which will have to be eliminated. Then the component will be documented using the templates developed during the groundwork after which the component and its sub-components will be added to the library. Last but not least the component will be reused in EWare.

The EWare project is used as guinea pig for several reasons:

- **Finish pilot project.** The pilot project started April 1998 and phase one was completed November 1998. Completing phase two and thus wrapping up the pilot would be an additional benefit of using EWare for a first evaluation of the templates and library.
- **Intimate knowledge of EWare.** One of the persons in the SR project group has worked on EWare previously and has intimate knowledge of the code. He also participated in the design of the search component and thus has some knowledge of this as well. These two facts minimise the risk(s) of unforeseen problems.

Involved cornerstones:

- **Documentation**
 - *Templates for documenting reusable assets.* The reusable assets must be documented to ease reuse.
- **Guidelines**
 - *Salvaging reusable assets.* As the search component must be salvaged from IRIS/MFS, some experience will be gained in this area.
- **Organisational Issues and Procedures**
 - *Problems when maintaining reusable assets.* As two projects will be using the same asset experiences with maintenance of such assets will be gained. This includes procedures for error reporting and correction, notification about new versions of assets, and general problems pertaining to maintenance and management of the assets.
 - *Education and training of employees.* The EWare project member must receive training in some patterns in order to understand the design of the search component and implement the code necessary to use it.
- **Metrics**
 - *Gather data.* Data will be gathered about the effort needed to salvage the reusable assets.

6.2.1 Deliverables

This iterations results in several deliverables:

- Templates for documenting reusable components.
- A report describing the experiences of the second phase of the pilot project.

6.3 2nd Iteration: Gathering and Disseminating

Purpose:

To gather assets and knowledge about potential assets, support dissemination of knowledge about assets and potential assets. This is done within the Products Department.

Involved cornerstones:

• **Tool support.**

- *First version of library tool.* In collaboration with the KM project a tool for management and dissemination of knowledge is produced. Knowledge refers to assets in the form of documents (KM) or code (SR project).
- *Mediation of information about reusable assets.* One thing is placing reusable assets in a repository, another is creating awareness about the components present and notifying users when they are modified. The mediation task involves setting up mechanisms and procedures for notification about new or modified assets.
- *Candidate list.* A list of already implemented components that might (should) be made reusable in the future.
- *List of wishes for components and features.* Developers can register wishes for new components and addition of features in existing components.
- *Versioning of assets.* When assets are modified or expanded, a new version is created. The tools must support versioning of assets.

• **Documentation.**

- *Updated documentation template for reusable code.* The templates for describing reusable components will also need updating as more knowledge is gained.

• **Guidelines.**

- *Salvaging reusable assets.* The experience that will be gained during the salvaging of the code will be joined with the experiences of the previous iteration into guidelines describing how to salvage code for reuse.
- *Identification of potentially reusable assets.* Some of the code in IRIS/MFS is investigated and described using the template for describing assets in the library.
- *Types of reusable components.* The reusable components are identified above are examined to gain knowledge about the different types of reusable components.

6.3.1 Deliverables

This iterations results in several deliverables:

- Improved documentation templates.
- Prototype implementation of a library tool.
- Guideline document on the salvaging of components.
- A report describing the experiences of the iterations.

6.4 3rd Iteration: Reuse in the Development Process

Purpose:

The role of reuse in the software development process is defined and established.

Involved cornerstones:

• **Organisational Issues and Procedures**

- *Preparation of tender.* The amount of reuse in a project will have effects on the estimates when preparing a tender for a project.
- *Project Planning.* When planning the project, reuse will have effect on the time scheduled for analysis, design and implementation.
- *Training and education of participants.* The participants in the project must have a sufficient base of knowledge for reusing and developing for reuse.
- *Requirement Specification.* When doing requirement specification, reuse must be considered both when reusing components that have requirements and when developing new components for reuse.
- *System Design.* Reuse places certain constraints on the system design as the reused parts of the system are black-boxes, i.e., their interfaces are known to the designers, but their internals are not. Furthermore designers will have to deal with the fact that the interfaces of some parts, i.e., the reused parts, are more or less fixed.
- *Detailed Design.* Same problems as in the system design phase.
- *Implementation.* Same problems as in the system design phase.
- *Maintenance.* There are several questions concerning maintenance that need to be answered when reuse is introduced:
 - What happens if reused code is altered, e.g., a component is modified and thus gets a new version number. Should all projects that use the components upgrade?
 - When components are modified, e.g., because of bug fixes, what should be tested?

• **Guidelines.**

- *Design of reusable code.* This could, for example, address the use of pre- and post-conditions and “design by contract”.
- *Implementation of reusable code.* Guidelines such as patterns and anti-patterns for development of reusable components, i.e., how to do it and how not to do it.
- *Integration and usage of reusable assets.* Information on how to integrate, instantiate, and use reusable components present in the library.
- *Improvement of existing guidelines.* The guidelines already written are improved as deficiencies are detected and experiences are gained.

• **Documentation**

- *Improvement of existing documentation templates.* The templates are improved as deficiencies are detected and experiences are gained.

6.5 4th Iteration: Multiple Projects and Departments

Purpose:

Introducing systematic reuse across several SSE departments. This is done by adding more and more projects gradually.

Involved cornerstones:

- **Activities outside cornerstones.** As the Industrial and Defence Projects Departments are added, some of the activities from the SR project initiation phase have to be performed for these department as they were for the Product Department.
 - *Analysis of current reuse situation at Industrial and Defence Departments.* As projects in these departments often are one-shot projects there may be differences in the way reuse is done. The analysis will examine the way projects are carried out in the two departments.
 - *Refinement of the SR project goal.* Again the differences between the Product Department and the two other departments may necessitate a refinement of the overall goal for the SR project.
- **Organisational Issues and Procedures**
 - *Refining implemented procedures.* The procedures implemented in the two previous iterations are improved.
 - *Implementation of additional procedures.* The procedures that have not yet been addressed are designed and implemented.
- **Metrics**
 - *Degree of reuse in a project.* When more projects are added, it becomes interesting to examine the degree of reuse both in terms of input and output.
 - How much does a project reuse?
 - What kinds of assets have been reused?
 - What is the maturity of the reused assets?
 - How many reusable assets has a project produced?

6.6 5th Iteration: Improve Metrics and Procedures

Purpose:

As reuse is becoming more and more widespread in the organisation, the exact nature of metrics and organisational procedures and issues becomes clearer.

Involved cornerstones:

- **Organisational Issues and Procedures**
 - *Refining of implemented procedures.* The procedures implemented in the two previous iterations are improved.
- **Metrics**
 - *Maturity of a reusable asset.* How stable and mature is each asset? There might be several levels of maturity.
 - *Quality assurance.* A more formal way to decide if something is good enough for reuse.
 - *Overall reuse grade.* The “overall reuse grade” expresses the assets’ overall degree of reusability.
- **Tool Support for Reuse**
 - *Reuse metrics.* The tool should to some degree support the automatic computation of a number of reuse metrics. See [Poulin 96] for reuse metrics.

6.7 6th Iteration: Reusing a Broader Range of Assets

Purpose:

Until now the SR project has focused on software reuse. This iteration expands the reuse scope to other types of reusable assets, e.g., requirement specifications, SDDs, DDDs, test specifications, etc.

Involved cornerstones:

- **Tool Support for Reuse**
 - *Classification of assets.* As new assets are added, there must be defined classification schemes for these assets.
 - *Mediation of information about reusable assets.* New types of assets may require new kinds of mediation techniques.
- **Organisational Issues and Procedures**
 - *Modification of existing procedures.* As more types of assets are added, it may become necessary to adjust the software process so that it incorporates the new possibilities.

7. References

- [Bass et al. 98] Len Bass, Paul Clements, and Rick Kazman:
“*Software Architecture in Practice*”,
Addison-Wesley, 1998, ISBN 0-201-19930-0
- [Bassett 1996] Paul G. Bassett: “*Framing Software reuse*”,
Prentice Hall, 1996, ISBN 0-133-27859-X
- [Buschmann et al. 95] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter
Sommerlad, Michael Stal: “*A system of Patterns*”,
Wiley, 1996, ISBN 0-471-95869-7
- [Coplien et al. 98] James Coplien, Daniel Hoffman, and David Weiss:
“*Commonality and variability in Software Engineering*”,
IEEE Software, November/December 1998
- [COT] Centre for Object Technology, <http://www.cit.dk/COT>
- [COT/3-20] Henrik Røn:
“*Development of a Component for Searching and Filtering*”,
Centre for Object Technology Report, 1999,
<http://www.cit.dk/COT/> under the “Report Series” report
COT/3-20 v1.1
- [Doublait 97] Stephane Doublait:
“*Standard Reuse Practices: Many Myths vs. a Reality*”,
ACM Standard View, Vol. 5, No. 2, June 1997
- [Fichman and Kemerer 97] Robert G. Fichman and Chris E. Kemerer:
“*Object Technology and Reuse: Lessons from Early
Adopters*”, IEEE Computer, October 1997
- [Goldberg 98] Adele Goldberg: “*A reuse business model*”,
Software – Tools & Concepts 1998 19:11-13, Springer
Verlag
- [Goldberg and Rubin 95] Adele Goldberg and Kenneth Rubin
“*Succeeding with Objects*”,
Addison-Wesley, 1995, ISBN 0-201-62878-3
- [Henninger 94] Scott Henninger:
“*Using Iterative Refinement to Find Reusable Software*”,
IEEE Software, September 1994
- [IDEAL] <http://www.sei.cmu.edu/ideal/ideal.bridge.html#overview>

- [Jacobson et al. 97a] Ivar Jacobson, Martin Griss, and Patrick Jonsson
“*Making the Reuse Business Work*”,
IEEE Computer, October 1997
- [Jacobson et al. 97b] Ivar Jacobson, Martin Griss, and Patrick Jonsson:
“*Software Reuse – Architecture, Process and
Organization for Business Success*”,
Addison-Wesley, 1997, ISBN 0-201-92476-5
- [Karlsson 95] Even-André Karlsson (Ed.): *Software Reuse – A Holistic
Approach*”,
John Wiley & Sons, 1995, ISBN 0-471-95819-0
- [Lim 98] Wayne C. Lim:
“*Managing Software Reuse*”,
Prentice Hall, 1998, ISBN 0-13-552373-7
- [McFeeley 96] Robert McFeeley:
“*IDEAL : A User’s Guide for Software Process
Improvement*”
Software Engineering Institute at Carnegie Mellon
University, CMU/SEI-96-HB-001, 1996
- [Ogush 92] M. Ogush:
“*Terms in transition: A software reuse lexicon*”,
Crosstalk: The Journal of Defense Software Engineering,
No. 39, pp. 41-45, December 1992
- [Poulin 96] Jeffrey Poulin:
“*Measuring Software Reuse*”, Addison-Wesley, 1996,
ISBN 0-201-63413-9
- [Raynes 98] Joseph Raynes:
“*Software Process Improvement With CMM*”,
Artech House, 1998, ISBN 0-890-06644-2
- [Shaw and Garlan 96] Mary Shaw and David Garlan:
“*Software Architecture – Perspectives on an Emerging
Discipline*”,
Prentice Hall, 1996, ISBN 0-131-82957-2

A. Abbreviations and Concepts

AU	University of Aarhus
CCIS	Command Control Information System
CIT	Centre for Information Technology
CMM	Capability Maturity Model. A model for measuring the maturity of an organisation developing software in software process improvement. It was developed at the Software Engineering Institute (SEI) at Carnegie Mellon University.
COT	Centre for Object Technology
COTS	Commercial-Of-The-Shelf
DDD.....	Detailed Design Document
DTI.....	Danish Technological Institute
EWare	A SSE product for management of information for electronic warfare
FAT	Factory Acceptance Test
IRIS/MFS.....	IRIS Message Formatting System. IRIS refers to the mythological figure.
KM.....	Knowledge Management
MRS.....	Master Requirement Specification
OO	Object-Orientation
PMP	Project Management Plan
SDD	System Design Document
SEI	Software Engineering Institute
SPI	Software Process Improvement
SR	Systematic Reuse
SRS	System Requirement Specification
SSE	Systematic Software Engineering A/S
WBS.....	Work Breakdown Schedule

B. Systematic Reuse Project Cornerstones

This section contains a description of the five cornerstones of the SR project. The cornerstones are a way to group the tasks necessary to introduce systematic reuse into focal areas. The cornerstones identified have been cross-referenced against the way the tasks are grouped in [Goldberg and Rubin 95] to ensure that no tasks or groups have been left out.

B.I Tool Support for Reuse

This cornerstone deals with the establishment of a system for description and publication of reusable code. It could be considered using the system for publication of SDDs, DDDs, etc., as well, so that the system would act as a kind of library or repository.

The reuse tools should deal with as different things as

- **Classification of assets.** The assets stored in the library must be classified according to a classification scheme yet to be determined. From the tool it should be possible to:

- Classify assets.
- Search on classifications.
- **Mediation of information about reusable assets.** One thing is placing reusable assets in a repository, another is creating awareness about the components present and when they are modified. The mediation task involves setting up mechanisms and procedures for notification about new or modified assets.
- **Candidate list.** A list of already implemented components that might (should) be made reusable.
- **List of wishes for components and features.** Developers can register wishes for new components and features in existing components.
- **Versioning of assets.** When assets are modified or expanded, a new version is created. The tools must support this versioning of assets.
- **Reuse metrics.** The tool should to some degree support the automatic computation of a number of reuse metrics.

B.II Documentation

When developing for reuse a number of the existing SSE document templates must be modified

- **Documentation of reusable code.** This includes templates for describing:
 - SDD and DDD for reusable components.
 - Architecture of the reusable component.
 - API of the reusable component.
 - An asset when it is stored in the library.
- **Documentation of usage and integration of reusable code.**
 - The SDD and DDD templates must be updated.

B.III Guidelines

One of the most difficult tasks when developing software is design, especially designing for reuse. There tend to be a number of pit-falls that every programmer encounters on the learning curve. This cornerstone consists of guidelines for:

- **Design of reusable code.** For example by using pre- and post-conditions and “design by contract”.
- **Implementation of reusable code.** Guidelines such as patterns and anti-patterns for development of reusable components, i.e., how to do it and how not to do it.
- **Integration and usage of reusable assets.** Information on how to integrate, instantiate, and use reusable components present in the library.
- **Identification of potentially reusable assets.** This deals with identification of reusable assets among existing software products.
- **Salvaging reusable assets.** When a potentially reusable asset has been identified, it must be generalised and made context independent.
- **Types of reusable assets.** What kinds of reusable assets exist.
- **Rules of thumb.** Rules of thumb for design of object-oriented code in general.

B.IV Organisational Issues and Procedures

SSE has a well-defined software process, which is under improvement as part of SSE's SPI project. As reuse can have impact throughout the software process, a number of phases will be affected by a systematic approach to reuse:

- **Preparation of tender.** The amount of reuse in a project will have effects on the estimates when preparing a tender for a project.
- **Project Planning.** When planning the project reuse will have effect on the time scheduled for analysis, design and implementation.
- **System Design.** Reuse places certain constraints on the system design as the reused parts of the system are black-boxes, i.e., their interfaces are known to the designers, but their internals are not. Furthermore designers will have to deal with the fact that the interfaces of some parts, i.e., the reused parts, are more or less fixed.
- **Detailed Design.** Same problems as in the system design phase.
- **Implementation.** Same problems as in the system design phase.
- **Maintenance.** There are several questions concerning maintenance that need to be answered when reuse is introduced:
 - What happens if reused code is altered, e.g., a component is modified and thus gets a version number. Should all projects that use the components upgrade?
 - When components are modified, e.g., because of bug fixes, what should be tested?

B.V Metrics

Metrics for reuse are important for several reasons:

- **Maturity of a reusable asset.** How stable and mature is each asset? There might be several levels of maturity.
- **Degree of reuse in a project.** Both in terms of input and output.
 - How much does a project reuse?
 - What kinds of assets have been reused?
 - What is the maturity of the reused assets?
 - How many reusable assets has a project produced?
- **Quality assurance.** How to decide if something is good enough for reuse?

The most used metric will be the

- **Overall reuse grade.** The overall reuse grade expresses the assets' overall degree of reusability.

This will be a number expressing the reusability of an assets. It will most likely be computed as an average of other weighted metrics.