REUSE OF SOFTWARE ENGINEERING

M.Sc. Ivanova Milka
Faculty of Mechanical Engineering – Technical University of Sofia, Bulgaria
milkachr@tu-sofia.bg

Abstract: *Reuse-based software engineering is a software engineering strategy where the development process is geared to reusing existing software. The move to reuse-based development has been in response to demands for lower software production and maintenance costs, faster delivery of systems, and increased software quality. More and more companies see their software as a valuable asset. They are promoting reuse to increase their return on software investments.*

Keywords*: Software engineering, reuse of software engineering, applicaton frameworks, software product lines, COST integration, ERP*

The availability of reusable software has increased dramatically. The open source movement has meant that there is a huge reusable code base available at low cost. This may be in the form of program libraries or entire applications.

There are many domain-specific application systems available that can be tailored and adapted to the needs of a specific company. Some large companies provide a range of reusable components for their customers.

Standards, such as web service standards, have made it easier to develop general services and reuse them across a range of applications.

Reuse-based software engineering is an approach to development that tries to maximize the reuse of existing software. The software units that are reused may be of radically different sizes:

- ***Application system reuse*** The whole of an application system may be reused by incorporating it without changing into other systems or by configuring the application for different customers. Alternatively, application families that have a common architecture, but which are tailored for specific customers, may be developed.
- ***Component reuse*** Components of an application, ranging in size from subsystems to single objects, may be reused. For example, a pattern-matching system developed as part of a text-processing system may be reused in a database management system.
- ***Object and function reuse*** Software components that implement a single function, such as a mathematical function, or an object class may be reused. This form of reuse, based around standard libraries, has been common. Many libraries of functions and classes are freely available. You reuse the classes and functions in these libraries by linking them with newly developed application code. In areas such as mathematical algorithms and graphics, where specialized expertise is needed to develop efficient objects and functions, this is a particularly effective approach.

Software systems and components are potentially reusable entities, but their specific nature sometimes means that it is expensive to modify them for a new situation. A complementary form of reuse is 'concept reuse' where, rather than reuse a software component, you reuse an idea, a way, or working or an algorithm. The concept that you reuse is represented in an abstract notation (e.g., a system model), which does not include implementation detail. It can, therefore, be configured and adapted for a range of situations. Concept reuse can be embodied in approaches such as design patterns configurable system products, and program generators. When concepts are reused, the reuse process includes an activity where the abstract concepts are instantiated to create executable reusable components.

An obvious advantage of software reuse is that overall development costs should be reduced. Fewer software components need to be specified, designed, implemented, and validated. However, cost reduction is only one advantage of reuse. In Table.1, they have listed some advantages of reusing software assets.

Table 1

Benefit	**Explanation**
Increased dependability	Reused software, which has been tried and tested in working systems, should be more dependable than new software. Its design and implementation faults should have been found and fixed
Reduced process risk	The cost of existing software is already known, whereas the costs of development are always a matter of judgment. This is an important factor for project management because it reduces the margin of error in project cost estimation. This is particularly true when relatively large software components such as subsystems are reused.
Effective use of specialists	Instead of doing the same work over and over again, application specialists can develop reusable software that encapsulates their knowledge.
Standards compliance	Some standards, such as user interface standards, can be implemented as a set of reusable components. For example, if menus in a user interface are implemented using reusable components, all applications present the same menu formats to users. The use of standard user interfaces improves dependability because users make fewer mistakes when presented with a familiar interface.
Accelerated development	Bringing a system to market as early as possible is often more important than overall development costs. Reusing software can speed up system production because both development and validation time may be reduced.

However, there are costs and problems associated with reuse. There are a significant cost associated with understanding whether or not a component is suitable for reuse in a particular situation, and in testing that component to ensure its dependability. These additional costs mean that the reductions in overall development costs through reuse may be less than anticipated.

Table 2

Problem	**Explanation**
Increased maintenance costs	If the source code of a reused software system or component is not available, then maintenance costs may be higher because the reused elements of the system may become increasingly incompatible with system changes.
Lack of tool support	Some software tools do not support development with reuse. It may be difficult or impossible to integrate these tools with a component library system. The software process assumed by these tools may not take reuse into account. This is particularly true for tools that support embedded systems engineering, less so for object-oriented development tools.
Not-invented-here syndrome	Some software engineers prefer to rewrite components because they believe they can improve on them. This is partly to do with trust and partly to do with the fact that writing original software is seen as more challenging than reusing other people's software.
Creating, maintaining, and using a component library	Populating a reusable component library and ensuring the software developers can use this library can be expensive. Development processes have to be adapted to ensure that the library is used.
Finding, understanding,	Software components have to be discovered in a library, understood and, sometimes, adapted

and adapting reusable components	to work in a new environment. Engineers must be reasonably confident of finding a component in the library before they include a component search as part of their normal development process.

Software development processes have to be adapted to take reuse into account. In particular, there has to be a requirements refinement stage where the requirements for the system are modified to reflect the reusable software that is available. The design and implementation stages of the system may also include explicit activities to look for and evaluate candidate components for reuse.

Software reuse is most effective when it is planned as part of an organization-wide reuse program. A reuse program involves the creation of reusable assets and the adaptation of development processes to incorporate these assets in new software.

Over the past years, many techniques have been developed to support software reuse. These techniques exploit the facts that systems in the same application domain are similar and have potential for reuse; that reuse is possible at different levels from simple functions to complete applications; and that standards for reusable components facilitate reuse. Figure 1 sets out a number of possible ways of implementing software reuse, with each described briefly in Table 3.

Figure 1

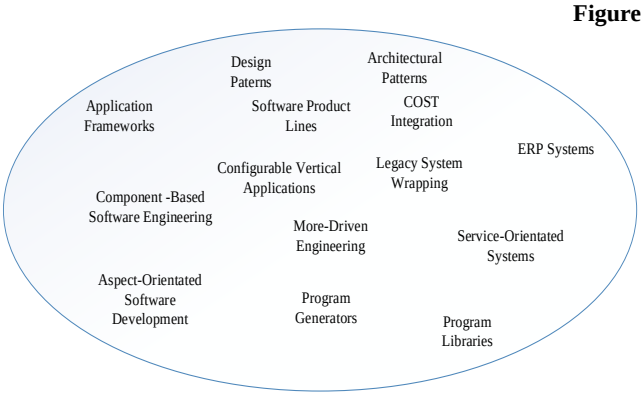


Table 3

Approach	Description
Architectural patterns	Standard software architectures that support common types of application systems are used as the basis of applications.
Design patterns	Generic abstractions that occur across applications are represented as design patterns showing abstract and concrete objects and interactions.
Component-based development	Systems are developed by integrating components (collections of objects) that conform to component-model standards.
Application frameworks	Collections of abstract and concrete classes are adapted and extended to create application systems.
Legacy system wrapping	Legacy systems are 'wrapped' by defining a set of interfaces and providing access to these legacy systems through these interfaces.
Service-oriented systems	Systems are developed by linking shared services, which may be externally provided.
Software product lines	An application type is generalized around a common architecture so that it can be adapted for different customers.
COTS product reuse	Systems are developed by configuring and integrating existing application systems.
ERP systems	Large-scale systems that encapsulate generic business functionality and rules are configured for an organization.
Configurable vertical applications	Generic systems are designed so that they can be configured to the needs of specific system customers.
Program libraries	Class and function libraries that implement commonly used abstractions are available for reuse.
Model-driven engineering	Software is represented as domain models and implementation independent models and code is generated from these models.
Program generators	A generator system embeds knowledge of a type of application and is used to generate systems in that domain from a user- supplied system model.
Aspect-oriented software development	Shared components are woven into an application at different places when the program is compiled.

Given this array of techniques for reuse, the key question is "which is the most appropriate technique to use in a particular situation?" Obviously, this depends on the requirements for the system being developed, the technology and reusable assets available, and the expertise of the development team. Key factors that you should consider when planning reuse are:

- ***The development schedule for the software*** If the software has to be developed quickly, you should try to reuse off-the-shelf systems rather than individual components. These are large-grain reusable assets. Although the fit to requirements may be imperfect, this approach minimizes the amount of development required.
- ***The expected software lifetime*** If you are developing a long-lifetime system, you should focus on the maintainability of the system. You should not just think about the immediate benefits of reuse but also of the long-term implications. Over its lifetime, you will have to adapt the system to new requirements, which will mean making changes to parts of the system. If you do not have access to the source code, you may prefer to avoid off-the-shelf components and systems from external suppliers; suppliers may not be able to continue support for the reused software.
- ***The background, skills, and experience of the development team*** All reuse technologies are fairly complex and you need quite a lot of time to understand and use them effectively. Therefore, if the development team has skills in a particular area, this is probably where you should focus.
- ***The criticality of the software and its non-functional requirements*** For a critical system that has to be certified by an external regulator, you may have to create a dependability case for the system This is difficult if you don't have access to the source code of the software. If your software has stringent performance requirements, it may be impossible to use strategies such as generator-based reuse, where you generate the code from a reusable domain- specific representation of a system. These systems often generate relatively inefficient code.
- ***The application domain*** In some application domains, such as manufacturing and medical information systems, there are several generic products that may be reused by configuring them to a local situation. If you are working in such a domain, you should always consider these as an option
- ***The platform on which the system will run*** Some components models, such as NET, are specific to Microsoft platforms. Similarly, generic application systems may be platform-specific and you may only be able to reuse these if your system is designed for the same platform

The range of available reuse techniques is such that, in most situations, there is the possibility of some software reuse. Whether or not reuse is achieved is often a managerial rather than a technical issue. Managers may be unwilling to compromise their requirements to allow reusable components to be used. They may not understand the risks associated with reuse as well as they understand the risks of original development. Although the risks of new software development may be higher, some managers may prefer known to unknown risks

Application Frameworks & Software Product Lines

It has become clear that object-oriented reuse is best supported in an object-oriented development process through larger-grain abstractions called frameworks. A framework is a generic structure that is extended to create a more specific subsystem or application.

Frameworks provide support for generic features that are likely to

be used in all applications of a similar type.

Frameworks support design reuse in that they provide a skeleton architecture for the application as well as the reuse of specific classes in the system. The architecture is defined by the object classes and their interactions. Classes are reused directly and may be extended using features such as inheritance.

Frameworks are implemented as a collection of concrete and abstract object classes in an object-oriented programming language. Therefore, frameworks are language-specific.

Frameworks are often implementations of design patterns. For example, an MVC framework includes the Observer pattern, the Strategy pattern, the Composite pattern, and a number of others. The general nature of patterns and their use of abstract and concrete classes allows for extensibility. Without patterns, frameworks would, almost certainly, be impractical.

Web application frameworks usually incorporate one or more specialized frameworks that support specific application features. Although each framework includes slightly different functionality, most web application frameworks support the following features: ***Security*** WAFs may include classes to help implement user authentication (login) and access control to ensure that users can only access permitted functionality in the system; ***Dynamic web pages*** Classes are provided to help you define web page templates and to populate these dynamically with specific data from the system database; ***Database support*** Frameworks don't usually include a database but rather assume that a separate database, such as MySQL, will be used. The framework may provide classes that provide an abstract interface to different databases; ***Session management*** Classes to create and manage sessions (a number of interactions with the system by a user) are usually part of a WAF; ***User interaction*** Most web frameworks now provide AJAX support which allows more interactive web pages to be created.

However, frameworks are usually more general than software product lines, which focus on a specific family of application system. For example, you can use a web-based framework to build different types of web-based applications. One of these might be a software product line that supports web-based help desks. This 'help desk product line' may then be further specialized to provide particular types of help desk support.

Frameworks are an effective approach to reuse, but are expensive to introduce into software development processes. They are inherently complex and it can take several months to learn to use them. It can be difficult and expensive to evaluate available frameworks to choose the most appropriate one. Debugging framework- based applications is difficult because you may not understand how the framework methods interact. This is a general problem with reusable software. Debugging tools may provide information about the reused system components, which a developer does not understand.

A software product line is a set of applications with a common architecture and shared components, with each application specialized to reflect different requirements. The core system is designed to be configured and adapted to suit the needs of different system customers. This may involve the configuration of some components, implementing additional components, and modifying some of the components to reflect new requirements.

Software product lines usually emerge from existing applications. That is, an organization develops an application then, when a similar system is required, informally reuses code from this in the new application. The same process is used as other similar applications are developed. However, change tends to corrupt application structure so, as more new instances are developed, it becomes increasingly difficult to create a new version. Consequently, a decision to design a generic product line may then be made. This involves identifying common functionality in product instances and including this in a base application, which is then used for future development. This base application is deliberately structured to simplify reuse and reconfiguration.

Application frameworks and software product lines obviously have much in common. They both support a common architecture and components, and require new development to create a specific version of a system. The main differences between these approaches are as follows:

- Application frameworks rely on object-oriented features such as inheritance and polymorphism to implement extensions to the framework. Generally, the framework code is not modified and the possible modifications are limited to whatever is allowed by the framework. Software product lines are not necessarily created using an object-oriented approach. Application components are changed, deleted, or rewritten. There are no limits, in principle at least, to the changes that can be made.
- Application frameworks are primarily focused on providing technical rather than domain-specific support. A software product line usually embeds detailed domain and platform information. For example, there could be a software product line concerned with web-based applications for health record management.
- Software product lines are often control applications for equipment. This means that the product line has to provide support for hardware interfacing. Application frameworks are usually software-oriented and they rarely provide support for hardware interfacing.
- Software product lines are made up of a family of related applications, owned by the same organization. When you create a new application, your starting point is often the closest member of the application family, not the generic core application.

COST-solution Systems & ERP

A commercial-off-the-shelf (COTS) product is a software system that can be adapted to the needs of different customers without changing the source code of the system. COTS products are adapted by using built-in configuration mechanisms that allow the functionality of the system to be tailored to specific customer needs. For example, in a hospital patient record system, separate input forms and output reports might be defined for different types of patient. Other configuration features may allow the system to accept plug-ins that extend functionality or check user inputs to ensure that they are valid.

This approach to software reuse has been very widely adopted by large companies over the last years, as it offers significant benefits over customized software development:

- As with other types of reuse, more rapid deployment of a reliable system may be possible
- It is possible to see what functionality is provided by the applications and so it is easier to judge whether or not they are likely to be suitable. Other companies may already use the applications so experience of the systems is available.
- Some development risks are avoided by using existing software. However, this approach has its own risks, as discuss below.
- Businesses can focus on their core activity without having to devote a lot of resources to IT systems development.
- As operating platforms evolve, technology updates may be simplified as these are the responsibility of the COTS product vendor rather than the customer.

Of course, this approach to software engineering has its own problems:

- Requirements usually have to be adapted to reflect the functionality and mode of operation of the COTS product. This can lead to disruptive changes to existing business processes.
- The COTS product may be based on assumptions that are practically impossible to change. The customer must therefore adapt their business to reflect these assumptions.
- Choosing the right COTS system for an enterprise can be a difficult process, especially as many COTS products are not well documented. Making the wrong choice could be disastrous as it may be impossible to make the new system work as required.
- There may be a lack of local expertise to support systems development. Consequently, the customer has to rely on the vendor and external consultants for development advice. This advice may be biased and geared to selling products and services, rather than meeting the real needs of the customer.
- The COTS product vendor controls system support and evolution. They may go out of business, be taken over, or may

make changes that cause difficulties for customers.

Software reuse based on COTS has become increasingly common. There are two types of COTS product reuse, namely COTS-solution systems and COTS-integrated systems. COTS-solution systems consist of a generic application from a single vendor that is configured to customer requirements. COTS-integrated systems involve integrating two or more COTS systems (perhaps from different vendors) to create an application system. In Table 4 they summarize the differences between these different approaches:

Table 4

COST-solution Systems	COST-integrated Systems
Single product that provides the functionality required by a customer	Several heterogeneous system products are integrated to provide customized functionality
Based around a generic solution and standardized processes	Flexible solutions may be developed for customer processes
Development focus is on system configuration	Development focus is on system integration
System vendor is responsible for maintenance	System owner is responsible for maintenance
System vendor provides the platform for the system	System owner provides the platform for the system

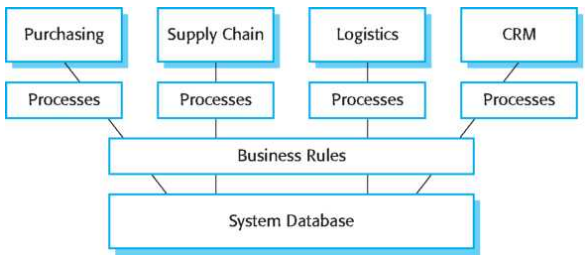
COTS-solution systems are generic application systems that may be designed to support a particular business type, business activity, or sometimes, a complete business enterprise. For example, a COTS-solution system may be produced for dentists that handles appointments, dental records, patient recall, etc. At a larger scale, an Enterprise Resource Planning (ERP) system may support all of the manufacturing, ordering, and customer relationship management activities in a large company.

Domain-specific COTS-solution systems, such as systems to support a business function (e.g., document management), provide functionality that is likely to be required by a range of potential users. However, they also incorporate built-in assumptions about how users work and these may cause problems in specific situations.

ERP systems are large-scale integrated systems designed to support business practices such as ordering and invoicing, inventory management, and manufacturing scheduling. The configuration process for these systems involves gathering detailed information about the customer's business and business processes, and embedding this in a configuration database. This often requires detailed knowledge of configuration notations and tools and is usually carried out by consultants working alongside system customers.

A generic ERP system includes a number of modules that may be composed in different ways to create a system for a customer. The configuration process involves choosing which modules are to be included, configuring these individual modules, defining business processes and business rules, and defining the structure and organization of the system database. A model of the overall architecture of an ERP system that supports a range of business functions is shown in Figure 2.

Figure 2



The key features of this architecture are: Number of modules to support different business functions; Defined set of business processes, associated with each module, which relate to activities in that module; Common database that maintains information about all related business functions; Set of business rules that apply to all data in the database.

Both domain-specific COTS products and ERP systems usually require extensive configuration to adapt them to the requirements of each organization where they are installed. This configuration may involve:

- Selecting the required functionality from the system
- Establishing a data model that defines how the organization's data will be structured in the system database.
- Defining business rules that apply to that data.
- Defining the expected interactions with external systems.
- Designing the input forms and the output reports generated by the system.
- Designing new business processes that conform to the underlying process model supported by the system.
- Setting parameters that define how the system is deployed on its underlying platform.

COTS integration can be simplified if a service-oriented approach is used. Essentially, a service-oriented approach means allowing access to the application system's functionality through a standard service interface, with a service for each discrete unit of functionality. Some applications may offer a service interface but, sometimes, this service interface has to be implemented by the system integrator

SUMMARY:

Most new business software systems are now developed by reusing knowledge and code from previously implemented systems. There are many different ways to reuse software. These range from the reuse of classes and methods in libraries to the reuse of complete application systems. The advantages of software reuse are lower costs, faster software development, and lower risks. System dependability is increased. Specialists can be used more effectively by concentrating their expertise on the design of reusable components. Application frameworks are collections of concrete and abstract objects that are designed for reuse through specialization and the addition of new objects. They usually incorporate good design practice through design patterns. Software product lines are related applications that are developed from one or more base applications. A generic system is adapted and specialized to meet specific requirements for functionality, target platform, or operational configuration. COTS product reuse is concerned with the reuse of large-scale, off-the-shelf systems. These provide a lot of functionality and their reuse can radically reduce costs and development time. Systems may be developed by configuring a single, generic COTS product or by integrating two or more COTS products. Enterprise Resource Planning systems are examples of large-scale COTS reuse. You create an instance of an ERP system by configuring a generic system with information about the customer's business processes and rules. Potential problems with COTS-based reuse include lack of control over functionality and performance, lack of control over system evolution, the need for support from external vendors, and difficulties in ensuring that systems can interoperate.

BIBLOGRAPHY:

1. I 1016-2009 IEEE standard for Information Technology-Systems Design-Software Design Descriptions, ed, 2009.
2. Budgen D., Software Design, 2nd ed., New York: Addison-Wesley, 2003
3. IEEE/ISO/IEC, IEEE/ISO/IEC Systems &Software Engineering Vocabulary, 1st ed. 2010
4. Mili H., A. Mili, S. Yacoub and E. Addy, Reuse-based Software Engineering. A comprehensive discussion of different approaches to software reuse, John Wiley & Sons, 2002.
5. Sommerville I, Software Engineering 9th ed. New York: Addison-Wesle1y 2010