A

Seminar report

On

"Software reuse"

Submitted in partial fulfillment of the requirement for the award of degree
of Bachelor of Technology in Computer Science

SUBMITTED TO:
www.studymafia.com

SUBMITTED BY:
www.studymafia.com

Preface

I have made this report file on the topic **software reuse**; I have tried my best to elucidate all the relevant detail to the topic to be included in the report. While in the beginning I have tried to give a general view about this topic.

My efforts and wholehearted co-corporation of each and everyone has ended on a successful note. I express my sincere gratitude to ……..who assisting me throughout the preparation of this topic. I thank him for providing me the reinforcement, confidence and most importantly the track for the topic whenever I needed it.

Introduction of Software Reuse

Software reuse is the process of creating software systems from existing software rather than building software systems from scratch. Something that was originally written for a different project and implementation will usually be recognized as reuse. Code reuse is the idea that a partial or complete computer program written at one time can be, should be, or is being used in another program written at a later time. The reuse of programming code is a common technique which attempts to save time and energy by reducing redundant work. Software assets, or components, include all software products, from requirements and proposals, to specifications and designs, high level designs, data formats, algorithms to user manuals and test suites. Anything that is produced from a software development effort can potentially be reused. Software developed and used repeatedly by the same people on the same project and implementation, Product maintenance and new product versions, use of operating systems, database management systems, and other system tools doesn't amount to reuse. Software engineering has been more focused on original development but it is now recognized that to achieve better software, more quickly and at lower cost, we need to adopt a design process that is based on systematic software reuse.

For systematic reuse to succeed organizations must recognize that good components, frameworks, and software architectures require time to design, implement, optimize, validate, apply, maintain, and enhance. Creating reusable software assets requires a mature organization whose developers and architects can distinguish key sources of variability and commonality in their application domain. Identifying and separating these concerns for complex networked applications requires an iterative development process since it's hard to design reusable artifacts correctly the first time using a top down "waterfall" software lifecycle model.

Why Software Reuse

A good software reuse process facilitates the increase of productivity, quality, and reliability, performance and the decrease of costs, effort, risk and implementation time. An initial investment is required to start a software reuse process, but that investment pays for itself in a few reuses. In short, the development of a reuse process and repository produces a base of knowledge that improves in quality after every reuse, minimizing the amount of development work required for future project and implementations, and ultimately reducing the risk of new project and implementations that are based on repository knowledge

Reusing code saves programming time, which reduces costs. If one person or team has already solved a problem, and they share the solution, there's no need to solve the problem again (with some potential caveats see Drawbacks).

• Sharing code can help prevent bugs by reducing the amount of total code that needs to be written to perform a set of tasks. Generally, the more code a system contains the more bugs it's likely to have. The shared code can also be tested separately from the applications which use it.

• Separating code into common libraries lets programmers specialize in their particular strengths. A security library, for example, can be built by security experts while a user interface which uses the library can let UI experts focus on their tasks.

• Repeatedly, separating code into specialized libraries lets each be tuned for performance, security, and special cases. For example, a Python application might delegate graphics functionality to a C library for performance.

Objectives

1. To explain the benefits of software reuse and some reuse problems
2. To discuss several different ways to implement software reuse
3. To explain how reusable concepts can be represented as patterns or embedded in program generators
4. To discuss COTS reuse
5. To describe the development of software product lines

Software reuse

1. In most engineering disciplines, systems are designed by composing existing components that have been used in other systems.

2. Software engineering has been more focused on original development but it is now recognized that to achieve better software, more quickly and at lower cost, we need to adopt a design process that is based on systematic software reuse.

Reuse-based software engineering

1. Application system reuse

The whole of an application system may be reused either by incorporating it without change into other systems (COTS reuse) or by developing application families.

2. Component reuse

   Components of an application from sub-systems tosingle objects may be reused.

3. Object and function reuse

   Software components that implement a single well-defined object or function may be reused.

Reuse benefits and problems

Benefits

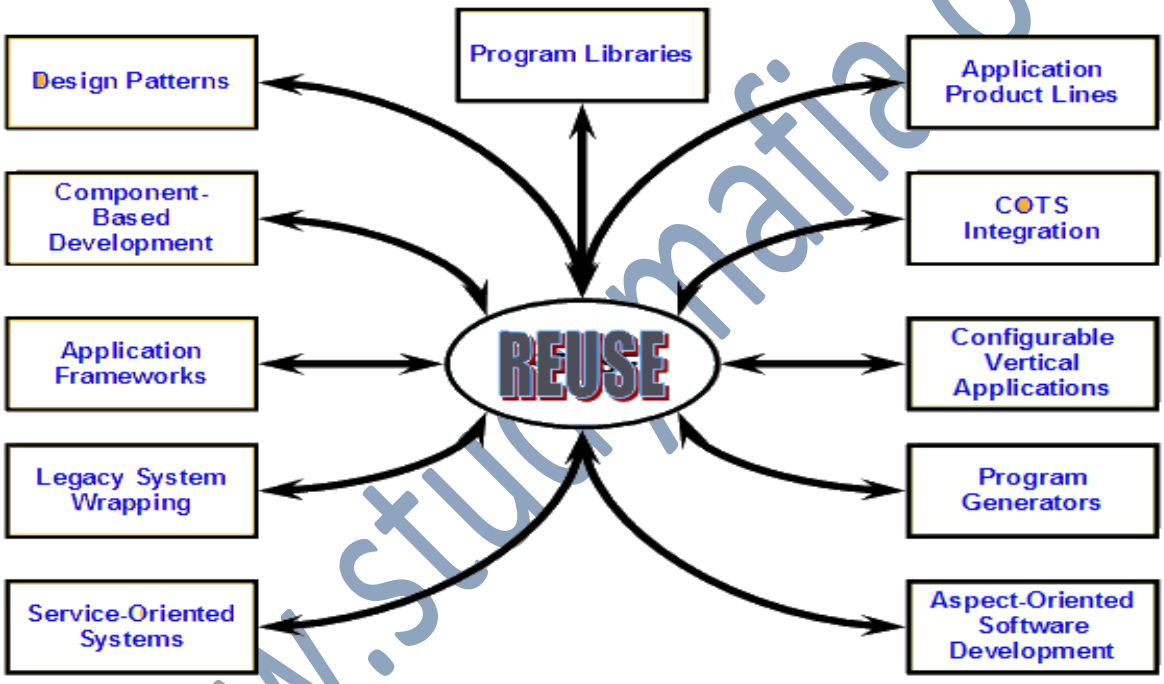
- Increased dependability
- Reduced process risk
- Effective use of specialists
- Standards compliance
- Accelerated development

Problems

- Increased maintenance costs
- Lack of tool support
- Not-invented-here syndrome
- Creating and maintaining a Component library
- Finding, understanding and Adapting reusable components

The reuse landscape

• Although reuse is often simply thought of as the reuse of system Components, there are many different approaches to reuse that may be Used.
• Reuse is possible at a range of levels from simple functions to complete application systems.
• The reuse landscape covers the range of possible reuse techniques.



Reuse planning factors

1. The development schedule for the software.
2. The expected software lifetime.
3. The background, skills and experience of the development team.
4. The criticality of the software and its non-functional requirements.
5. The application domain.
6. The execution platform for the software.

Concept reuse

• When you reuse program or design components, you have to follow the Design decisions made by the original developer of the component.
• This may limit the opportunities for reuse.
• However, a more abstract form of reuse is concept reuse when a Particular approach is described in an implementation independent way And an implementation is then developed.
• The two main approaches to concept reuse are:

- Design patterns
- Generative programming

Design patterns

• A design pattern is a way of reusing abstract knowledge about a problem and its solution.
• A pattern is a description of the problem and the essence of its solution.
• It should be sufficiently abstract to be reused in different settings.
• Patterns often rely on object characteristics such as inheritance and Polymorphism.

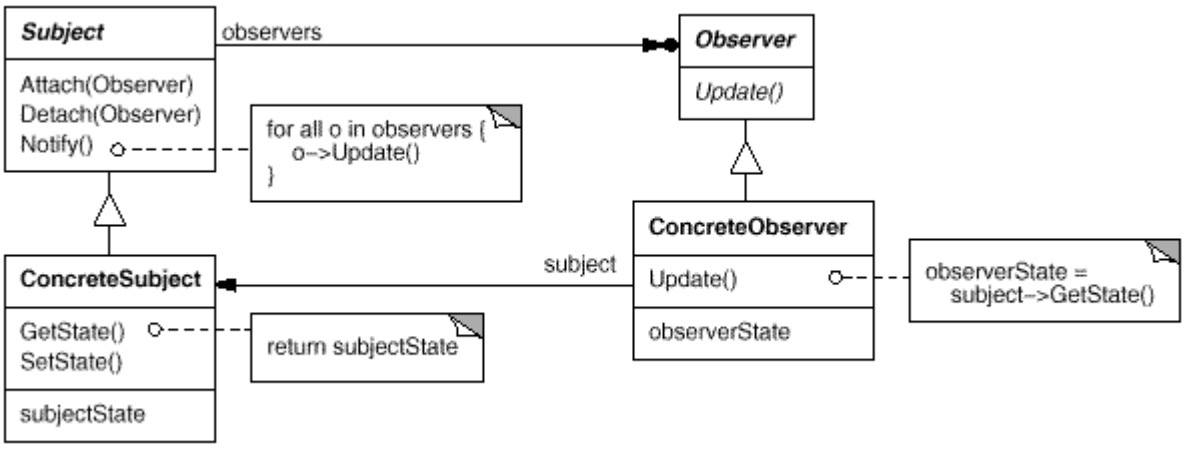
Pattern elements

- Name

– A meaningful pattern identifier.
• Problem description.
• Solution description.
– Not a concrete design but a template for a design solution that can be instantiated in different ways.
• Consequences
– The results and trade-offs of applying the pattern.

The Observer pattern

•Name
–Observer.
•Description
–Separates the display of object state from the object itself.
•Problem description
–Used when multiple displays of state are needed.
•Solution description
–See slide with UML description.
•Consequences
–Optimizations to enhance display performance are impractical.



Application frameworks

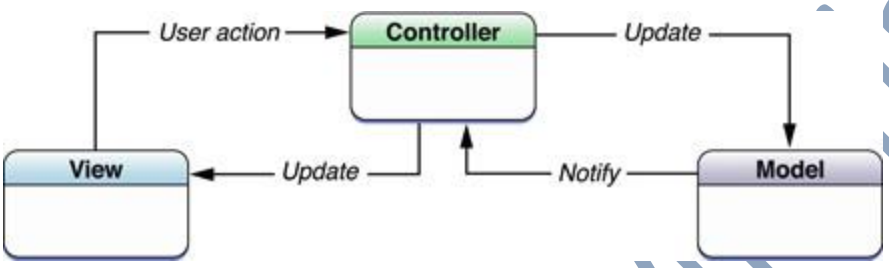
- Frameworks are a sub-system design made up of a collection of abstract and concrete classes and the interfaces between them.
- The sub-system is implemented by adding components to fill in parts of the design and by instantiating the abstract classes in the framework.
- Frameworks are moderately large entities that can be reused.

Framework classes

•System infrastructure frameworks
–Support the development of system infrastructures such as communications, user interfaces and compilers.
•Middleware integration frameworks
–Standards and classes that support component communication and information exchange.
•Enterprise application frameworks
–Support the development of specific types of application such as telecommunications or financial systems.

Model-view controller

• System infrastructure framework for GUI design.
• Allows for multiple presentations of an object and separate interactions with these presentations.
• MVC framework involves the instantiation of a number of patterns (as discussed earlier under concept reuse).



Application system reuse

• Involves the reuse of entire application systems either by configuring a system for an environment or by integrating two or more systems to create a new application.

• COTS - Commercial Off-The-Shelf systems
- COTS systems are usually complete application systems that offer an API (Application Programming Interface).
- Building large systems by integrating COTS systems is now a viable development strategy for some types of system such as E-commerce systems.
- The key benefit is faster application development and, usually, lower development costs.

• Software product lines
- Software product lines or application families are applications with generic functionality that can be adapted and configured for use in a specific context.
- Adaptation may involve:
- Component and system configuration;
- Adding new components to the system;
- Selecting from a library of existing components;
- Modifying components to meet new requirements.

Key points

• Advantages of reuse are lower costs, faster software development and lower risks.
• Design patterns are high-level abstractions that document successful design solutions.
• Program generators are also concerned with software reuse – the reusable concepts are embedded in a generator system.
• Application frameworks are collections of concrete and abstract objects that are designed for reuse through specialization.
• COTS product reuse is concerned with the reuse of large, off-the-shelf systems.
• Problems with COTS reuse include lack of control over functionality, performance, and evolution and problems with inter-operation.
• ERP systems are created by configuring a generic system with information about a customer's business.
• Software product lines are related applications developed around a common core of shared functionality.