
Software Reuse and Configuration Management

cm435 - 1

Objectives

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

cm435 - 2

Software reuse

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

cmse435 - 3

Reuse-based software engineering

- 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.
- Component reuse
 - Components of an application from sub-systems to single objects may be reused. Covered in Chapter 19.
- Object and function reuse
 - Software components that implement a single well-defined object or function may be reused.

cmse435 - 4

Reuse benefits 1

Increased dependability	Reused software, that has been tried and tested in working systems, should be more dependable than new software. The initial use of the software reveals any design and implementation faults. These are then fixed, thus reducing the number of failures when the software is reused.
Reduced process risk	If software exists, there is less uncertainty in the costs of reusing that software than in the costs of development. This is an important factor for project management as it reduces the margin of error in project cost estimation. This is particularly true when relatively large software components such as sub-systems are reused.
Effective use of specialists	Instead of application specialists doing the same work on different projects, these specialists can develop reusable software that encapsulate their knowledge.

Reuse benefits 2

Standards compliance	Some standards, such as user interface standards, can be implemented as a set of standard reusable components. For example, if menus in a user interfaces are implemented using reusable components, all applications present the same menu formats to users. The use of standard user interfaces improves dependability as users are less likely to make 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 should be reduced.

Reuse problems 1

Increased maintenance costs	If the source code of a reused software system or component is not available then maintenance costs may be increased as the reused elements of the system may become increasingly incompatible with system changes.
Lack of tool support	CASE toolsets may 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.
Not-invented-here syndrome	Some software engineers sometimes prefer to re-write components as they believe that they can improve on the reusable component. 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.

cmisc435 - 7

Reuse problems 2

Creating and maintaining a component library	Populating a reusable component library and ensuring the software developers can use this library can be expensive. Our current techniques for classifying, cataloguing and retrieving software components are immature.
! Finding, understanding and adapting reusable components	Software components have to be discovered in a library, understood and, sometimes, adapted to work in a new environment. Engineers must be reasonably confident of finding a component in the library before they will make routinely include a component search as part of their normal development process.

This is the big problem

- Reuse libraries are often "write only"
- Developers paid to build a system. What is their "payoff" to spend effort to make source code more useful for the next system?

cmisc435 - 8

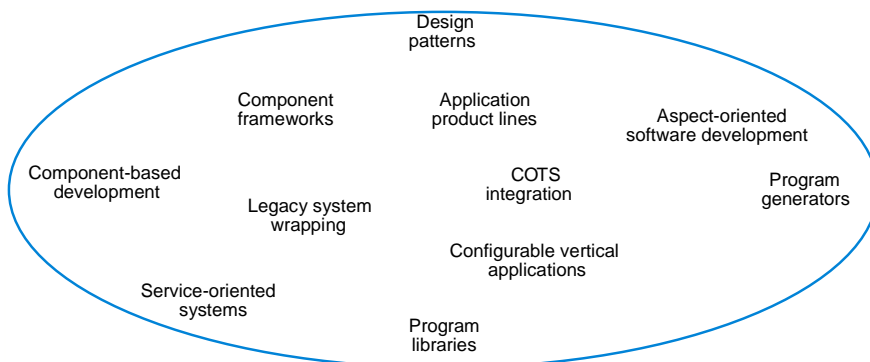
Changes for reusability

- Remove application-specific methods.
- Change names to make them general.
- Add methods to broaden coverage.
- Make exception handling consistent.
- Add a configuration interface for component adaptation.
- Integrate required components to reduce dependencies.

cmsc435 - 9

The reuse landscape

What do we mean by reuse?



cmsc435 - 10

Reuse approaches 1

Design patterns	Generic abstractions that occur across applications are represented as design patterns that show abstract and concrete objects and interactions.
Component-based development	Systems are developed by integrating components (collections of objects) that conform to component-model standards. This is covered in Chapter 19.
Application frameworks	Collections of abstract and concrete classes that can be adapted and extended to create application systems.
Legacy system wrapping	Legacy systems (see Chapter 2) that can be '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 that may be externally provided.

Reuse approaches 2

Application product lines	An application type is generalised around a common architecture so that it can be adapted in different ways for different customers.
COTS integration	Systems are developed by integrating existing application systems.
Configurable vertical applications	A generic system is designed so that it can be configured to the needs of specific system customers.
Program libraries	Class and function libraries implementing commonly-used abstractions are available for reuse.
Program generators	A generator system embeds knowledge of a particular types of application and can generate systems or system fragments in that domain.
Aspect-oriented software development	Shared components are woven into an application at different places when the program is compiled.

Reuse planning factors

- The development schedule for the software.
- The expected software lifetime.
- The background, skills and experience of the development team.
- The criticality of the software and its non-functional requirements.
- The application domain.
- 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.

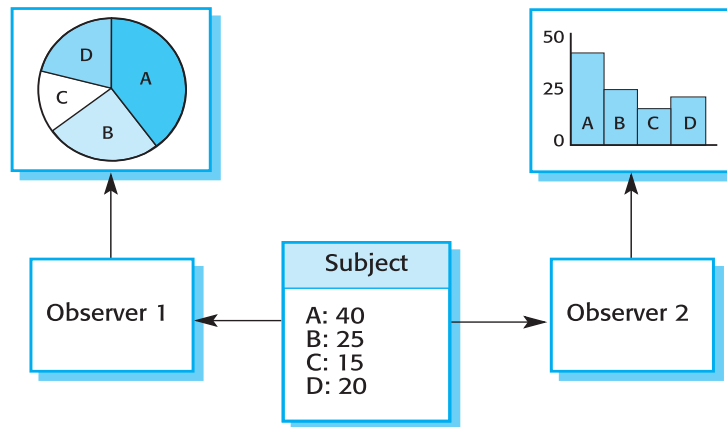
Design patterns

- 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.
- Not discussed in any detail. Part of OO development - you are supposed to be experts in that.

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.

Multiple displays



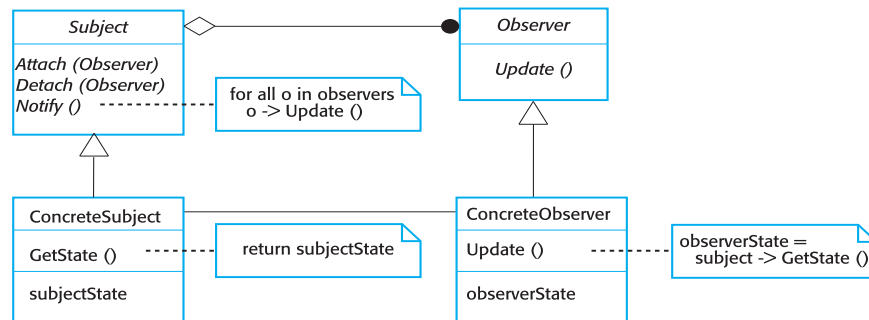
cmsc435 - 17

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.

cmsc435 - 18

The Observer pattern



cmse435 - 19

Generator-based reuse

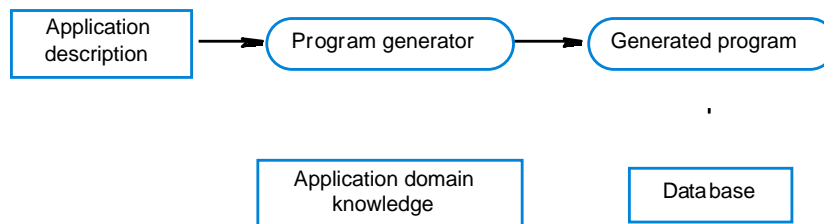
- Program generators involve the reuse of standard patterns and algorithms.
- These are embedded in the generator and parameterized by user commands. A program is then automatically generated.
- Generator-based reuse is possible when domain abstractions and their mapping to executable code can be identified.
- A domain specific language is used to compose and control these abstractions.

cmse435 - 20

Types of program generator

- Types of program generator
 - Application generators for business data processing;
 - Parser and lexical analyser generators for language processing;
 - Code generators in CASE tools.
- Generator-based reuse is very cost-effective but its applicability is limited to a relatively small number of application domains.
- It is easier for end-users to develop programs using generators compared to other component-based approaches to reuse.

Reuse through program generation

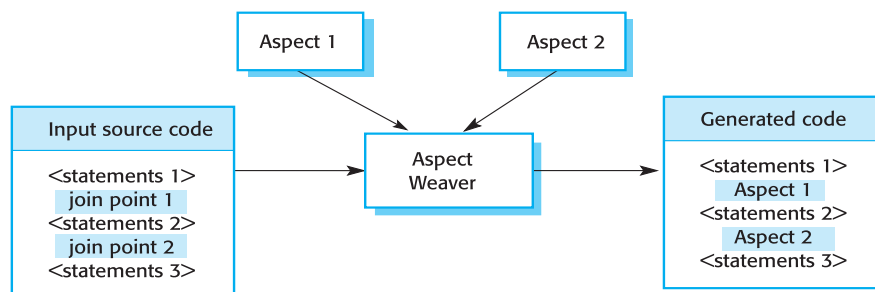


Aspect-oriented development

- Aspect-oriented development addresses a major software engineering problem - the separation of concerns.
- Concerns are often not simply associated with application functionality but are cross-cutting - e.g. all components may monitor their own operation, all components may have to maintain security, etc.
- Cross-cutting concerns are implemented as aspects and are dynamically woven into a program. The concern code is reused and the new system is generated by the aspect weaver.

cmse435 - 23

Aspect-oriented development



cmse435 - 24

Example: Aspect Oriented Programming

- Task: Add debugging code on entry and exit of a procedure.
- Traditional way: Do it manually.

```
public void doGet(JspImplicitObjects theObjects) throws
    ServletException
{
    logger.entry("doGet(...)");

    JspTestController controller = new JspTestController();
    controller.handleRequest(theObjects);

    logger.exit("doGet");
}
```

cmssc435 - 25

Example: Aspect Oriented Programming

- Use aspects:

```
public aspect AutoLog
{
    pointcut publicMethods() : execution(public *
        org.apache.cactus.*(..));
    pointcut logObjectCalls() : execution(* Logger.*(..));
    pointcut loggableCalls() : publicMethods() &&
        ! logObjectCalls();
    before() : loggableCalls(){
        Logger.entry(thisJoinPoint.getSignature().toString());
    }
    after() : loggableCalls(){
        Logger.exit(thisJoinPoint.getSignature().toString()); }
}
```

cmssc435 - 26

Aspect Oriented Programming

- Languages like AspectJ provide a preprocessor for languages like C and Java for adding additional code to each function.

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.
- **Question:** How do frameworks differ from reference models (e.g., ECMA toaster model)?

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.

Extending frameworks

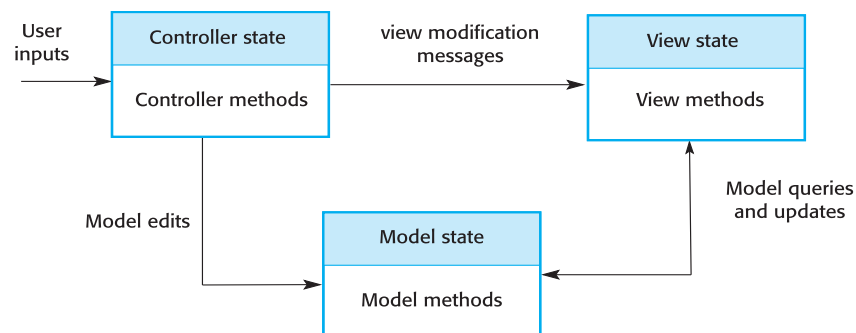
- Frameworks are generic and are extended to create a more specific application or sub-system.
- Extending the framework involves
 - Adding concrete classes that inherit operations from abstract classes in the framework;
 - Adding methods that are called in response to events that are recognised by the framework.
- Problem with frameworks is their complexity which means that it takes a long time to use them effectively.

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).

cmisc435 - 31

Model-view-controller



cmisc435 - 32

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.
- Two approaches covered here:
 - COTS product integration;
 - Product line development.

COTS product reuse

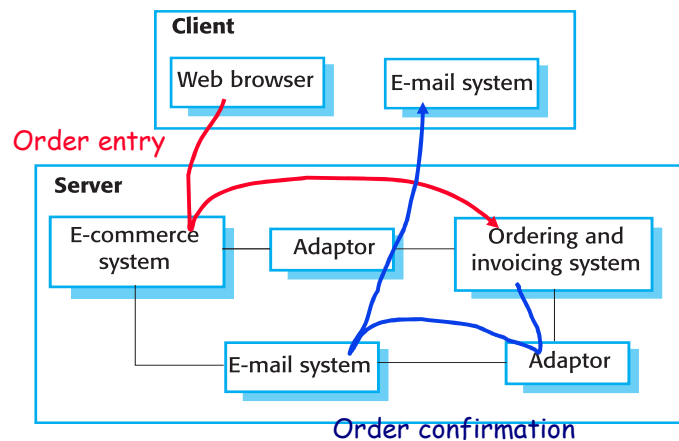
- COTS - Commercial Off-The-Shelf systems.
- COTS systems are usually complete application systems that offer an API (Application Programming Interface).
- Building large systems by integrating COTS systems is now a viable development strategy for some types of system such as E-commerce systems.
- The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

COTS design choices

- Which COTS products offer the most appropriate functionality?
 - There may be several similar products that may be used.
- How will data be exchanged?
 - Individual products use their own data structures and formats.
- What features of the product will actually be used?
 - Most products have more functionality than is needed. You should try to deny access to unused functionality.

cmssc435 - 35

E-procurement system



cmssc435 - 36

COTS products reused

- On the client, standard e-mail and web browsing programs are used.
- On the server, an e-commerce platform has to be integrated with an existing ordering system.
 - ❑ This involves writing an adaptor so that they can exchange data.
 - ❑ An e-mail system is also integrated to generate e-mail for clients. This also requires an adaptor to receive data from the ordering and invoicing system.

cmisc435 - 37

COTS system integration problems

- Lack of control over functionality and performance
 - ❑ COTS systems may be less effective than they appear
- Problems with COTS system inter-operability
 - ❑ Different COTS systems may make different assumptions that means integration is difficult
- No control over system evolution
 - ❑ COTS vendors not system users control evolution
 - ❑ Timing issues:
 - What if COTS vendor out of business?
 - What if need new functionality from COTS vendor?
 - What if COTS vendor update cycle not same as yours?
- Support from COTS vendors
 - ❑ COTS vendors may not offer support over the lifetime of the product

cmisc435 - 38

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.

COTS product specialization

- Platform specialization
 - ❑ Different versions of the application are developed for different platforms.
- Environment specialization
 - ❑ Different versions of the application are created to handle different operating environments e.g. different types of communication equipment.
- Functional specialization
 - ❑ Different versions of the application are created for customers with different requirements.
- Process specialization
 - ❑ Different versions of the application are created to support different business processes.

COTS configuration

- Deployment time configuration
 - A generic system is configured by embedding knowledge of the customer's requirements and business processes. The software itself is not changed.
- Design time configuration
 - A common generic code is adapted and changed according to the requirements of particular customers.

Product line architectures

- Architectures must be structured in such a way to separate different sub-systems and to allow them to be modified.
- The architecture should also separate entities and their descriptions and the higher levels in the system access entities through descriptions rather than directly.

Component-based development

- Component-based software engineering (CBSE) is an approach to software development that relies on software reuse.
- It emerged from the failure of object-oriented development to support effective reuse. Single object classes are too detailed and specific.
- Components are more abstract than object classes and can be considered to be stand-alone service providers.

CBSE essentials

- **Independent components** specified by their interfaces.
- **Component standards** to facilitate component integration.
- **Middleware** that provides support for component inter-operability.
- **A development process** that is geared to reuse.

CBSE and design principles

- Apart from the benefits of reuse, CBSE is based on sound software engineering design principles:
 - ❑ Components are independent so do not interfere with each other;
 - ❑ Component implementations are hidden;
 - ❑ Communication is through well-defined interfaces;
 - ❑ Component platforms are shared and reduce development costs.

cmse435 - 45

CBSE problems

- **Component trustworthiness** - how can a component with no available source code be trusted?
- **Component certification** - who will certify the quality of components?
- **Emergent property prediction** - how can the emergent properties of component compositions be predicted?
- **Requirements trade-offs** - how do we do trade-off analysis between the features of one component and another?

cmse435 - 46

Components

- Components provide a service without regard to where the component is executing or its programming language
 - A component is an independent executable entity that can be made up of one or more executable objects;
 - The component interface is published and all interactions are through the published interface;
- Interfaces
 - Provides interface - Defines the services that are provided by the component to other components.
 - Requires interface - Defines the services that specifies what services must be made available for the component to execute as specified.

cmisc435 - 47

Component characteristics 1

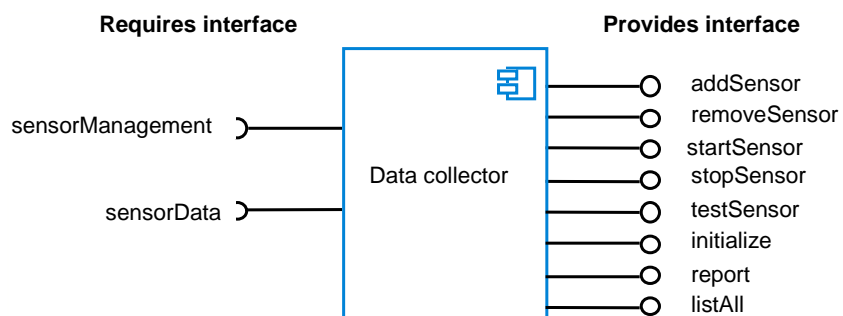
Standardized	Component standardization means that a component that is used in a CBSE process has to conform to some standardized component model. This model may define component interfaces, component meta-data, documentation, composition and deployment.
Independent	A component should be independent – it should be possible to compose and deploy it without having to use other specific components. In situations where the component needs externally provided services, these should be explicitly set out in a 'requires' interface specification.
Composable	For a component to be composable, all external interactions must take place through publicly defined interfaces. In addition, it must provide external access to information about itself such as its methods and attributes.

cmisc435 - 48

Component characteristics 2

Deployable	To be deployable, a component has to be self-contained and must be able to operate as a stand-alone entity on some component platform that implements the component model. This usually means that the component is a binary component that does not have to be compiled before it is deployed.
Documented	Components have to be fully documented so that potential users of the component can decide whether or not they meet their needs. The syntax and, ideally, the semantics of all component interfaces have to be specified.

A data collector component



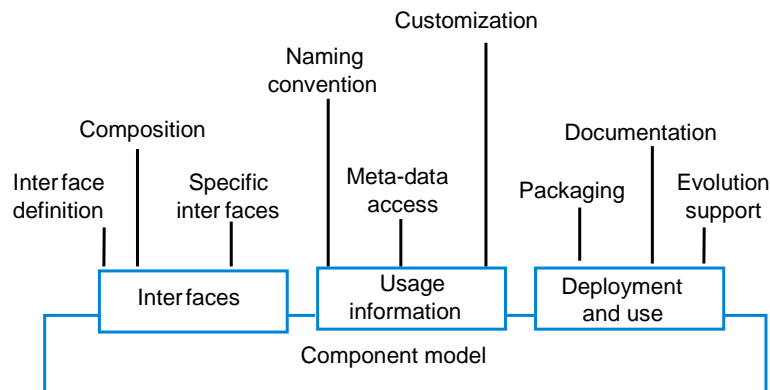
Components and objects

- Components are deployable entities.
- Components do not define types.
- Component implementations are opaque.
- Components are language-independent.
- Components are standardized.

Component models

- A component model is a definition of standards for component implementation, documentation and deployment.
- Examples of component models
 - EJB model (Enterprise Java Beans)
 - COM+ model (.NET model)
 - Corba Component Model
- The component model specifies how interfaces should be defined and the elements that should be included in an interface definition.

Elements of a component model



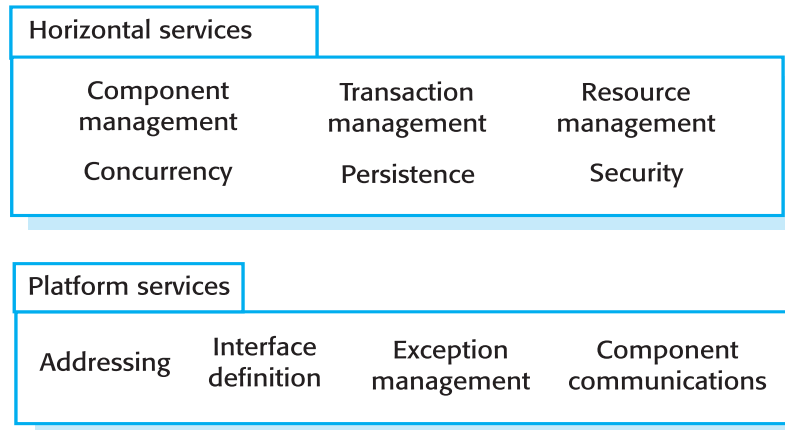
cmse435 - 53

Middleware support

- Component models are the basis for middleware that provides support for executing components.
- Component model implementations provide:
 - Platform services that allow components written according to the model to communicate;
 - Horizontal services that are application-independent services used by different components.
- To use services provided by a model, components are deployed in a **container**.

cmse435 - 54

Component model services



cmse435 - 55

Component development for reuse

- Components developed for a specific application usually have to be generalized to make them reusable.
- A component is most likely to be reusable if it associated with a stable domain abstraction (business object).
- For example, in a hospital stable domain abstractions are associated with the fundamental purpose - nurses, patients, treatments, etc.

cmse435 - 56

Component development for reuse

- Components for reuse may be specially constructed by generalizing existing components.
- Component reusability
 - ❑ Should reflect stable domain abstractions;
 - ❑ Should hide state representation;
 - ❑ Should be as independent as possible;
 - ❑ Should publish exceptions through the component interface.
- There is a trade-off between reusability and usability
 - ❑ The more general the interface, the greater the reusability but it is then more complex and hence less usable.

Legacy system components

- Existing legacy systems that fulfill a useful business function can be re-packaged as components for reuse.
- This involves writing a **wrapper** component that implements provides and requires interfaces then accesses the legacy system.
- Although costly, this can be much less expensive than rewriting the legacy system.

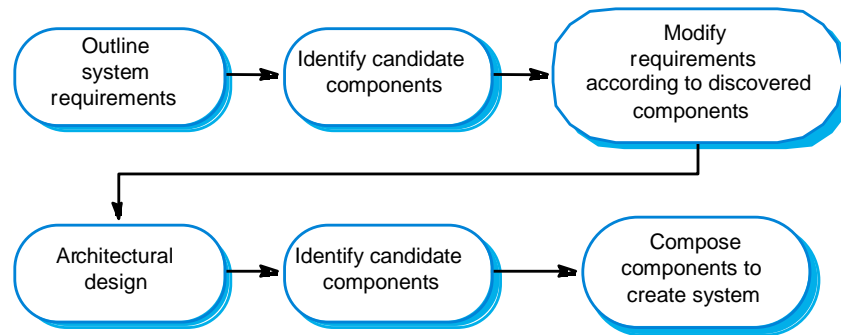
Reusable components

- The development cost of reusable components may be higher than the cost of specific equivalents. This extra reusability enhancement cost should be an organization rather than a project cost.
- Generic components may be less space-efficient and may have longer execution times than their specific equivalents.

The CBSE process

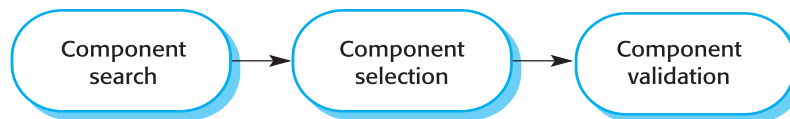
- When reusing components, it is essential to make trade-offs between ideal requirements and the services actually provided by available components.
- This involves:
 - Developing outline requirements;
 - Searching for components then modifying requirements according to available functionality (*Easier said than done*).
 - Searching again to find if there are better components that meet the revised requirements.

The CBSE process



cmse435 - 61

The component identification process



Big problem: How?

cmse435 - 62

Component identification issues

- **Trust.** You need to be able to trust the supplier of a component. At best, an untrusted component may not operate as advertised; at worst, it can breach your security.
- **Requirements.** Different groups of components will satisfy different requirements.
- **Validation.**
 - ❑ The component specification may not be detailed enough to allow comprehensive tests to be developed.
 - ❑ Components may have unwanted functionality. How can you test this will not interfere with your application?

cmse435 - 63

Ariane launcher failure

- In 1996, the 1st test flight of the Ariane 5 rocket ended in disaster when the launcher went out of control 39 seconds after take off.
- The problem was due to a reused component from a previous version of the launcher (the Inertial Reference System) that failed because assumptions made when that component was developed did not hold for Ariane 5.
- The functionality that failed in this component was not required in Ariane 5.
- **But was it really a software component failure?**

cmse435 - 64

Ariane failure

Design issues: In order to save funds and ensure reliability, and since the SRI is used for prelaunch activities, and since the French Ariane 4 was a successful rocket, the SRI from Ariane 4 was reused for the Ariane 5. [Reuse - good]

Operations: On June 4, 1996 the Ariane 5 launch vehicle failed 39 seconds after liftoff causing the destruction of over \$100 million in satellites [Use for mission with no test flight - bad]

Cause of failure: The Inertial Reference System (SRI), which controls the attitude of the vehicle by sending aiming commands to the rocket nozzle, sent a bad command to the rocket causing the nozzle to move the rocket toward the horizontal.

The vehicle tried to switch to the backup SRI (computer), but that failed for the same reason 72 millisecond earlier. [Did they really have redundancy with 2 computers? - bad]

The vehicle had to then be destroyed.

cmssc435 - 65

Inquiry Board Findings

- SRI tried to convert a floating point number out of range to integer. Therefore it issued an error message (as a 16 bit number). This 16 bit number was interpreted as an integer by the guidance system and caused the nozzle to move accordingly. [OV→number. Bad interface specs]
- The backup SRI performed according to specifications and failed for the same reason. [Assumed hardware reliability theory not software]
- The Ada range checking was disabled since the SRI was supposedly processing at 80% load and the extra time needed for range checking was deemed unnecessary since the Ariane 4 software worked well. [Disabled checks in compiler. - Software covering for inadequate hardware]
- The ultimate cause of the problem was that the Ariane 5 has a more pronounced angle of attack and can move horizontally sooner after launch. The "bad value" was actually the horizontal speed of the vehicle. [Requirements for Ariane 5 not same as Ariane 4.]
- The SRI was used for prelaunch activities. It could have been turned off as soon as launch began. [Bad design]

cmssc435 - 66

Was this a software failure?

No failure mode on SRI. Besides "crash only," needed "omission" and "non malicious behavior" modes (state tables?)

No range checking on data. No validation that OBC gets correct data in face of SRI failure (verification?).

System engineering specifications - guidance on Ariane 5 different from Ariane 4. Reuse is not free. Just because Ariane 4 SRI was bug free, reuse of code is not relevant if it is not the right product.

Pair of SRI systems. Assumed only random hardware errors - no account for any software problems. 1 out of 2 not sufficient.

Inquiry Board claimed this was a software engineering failure, but others claimed it was a systems engineering failure. Software Engineers did all they were supposed to be able to do.

Comments?

Component composition

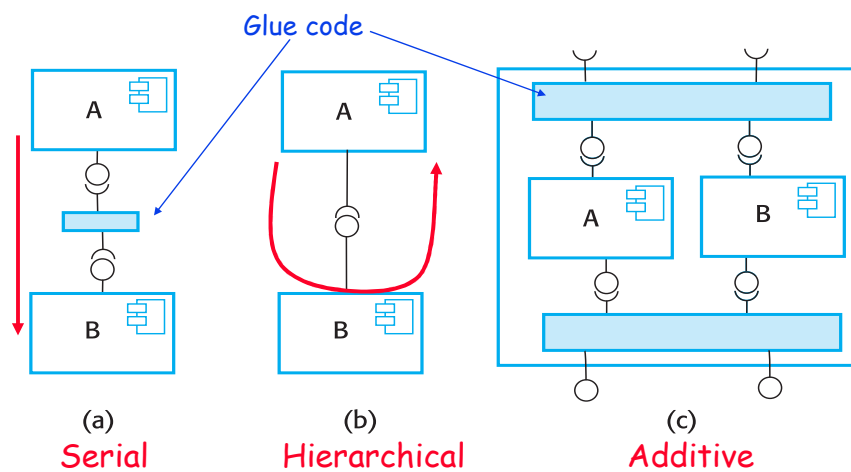
- The process of assembling components to create a system.
- Composition involves integrating components with each other and with the component infrastructure.
- Normally you have to write 'glue code' to integrate components.

Types of composition

- **Sequential composition** where the composed components are executed in sequence. This involves composing the provides interfaces of each component.
- **Hierarchical composition** where one component calls on the services of another. The provides interface of one component is composed with the requires interface of another.
- **Additive composition** where the interfaces of two components are put together to create a new component.

cmisc435 - 69

Types of composition



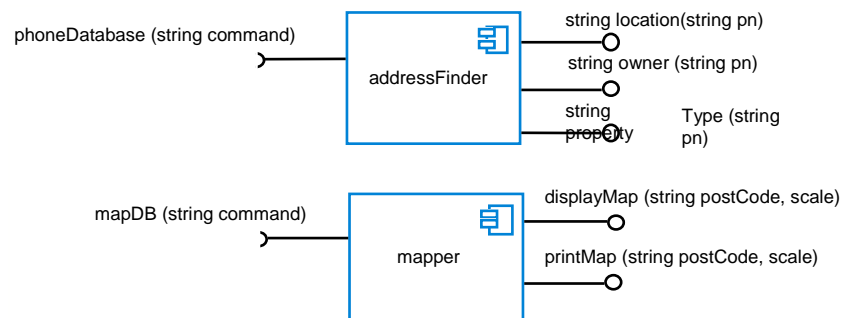
cmisc435 - 70

Interface incompatibility

- **Parameter incompatibility** where operations have the same name but are of different types (e.g., cause of Ariane 5 failure).
- **Operation incompatibility** where the names of operations in the composed interfaces are different.
- **Operation incompleteness** where the provided interface of one component is a subset of the requires interface of another.

cmisc435 - 71

Incompatible components



cmisc435 - 72

Adaptor components

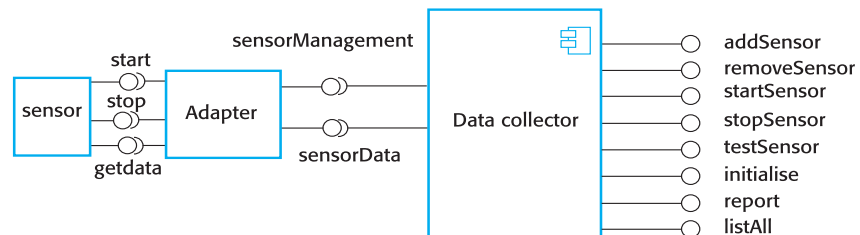
- Address the problem of component incompatibility by reconciling the interfaces of the components that are composed.
- Different types of adaptor are required depending on the type of composition.
- An addressFinder and a mapper component may be composed through an adaptor that strips the postal code from an address and passes this to the mapper component.

Composition through an adaptor

- The component `postCodeStripper` is the adaptor that facilitates the sequential composition of `addressFinder` and `mapper` components.

```
address = addressFinder.location (phonenumber) ;  
postCode = postCodeStripper.getPostCode (address);  
mapper.displayMap(postCode, 10000)
```

Adaptor for data collector



cmse435 - 75

The Object Constraint Language

- The Object Constraint Language (OCL) has been designed to define constraints that are associated with UML models.
- It is based around the notion of pre and post condition specification - similar to the approach used in Z

cmse435 - 76

Formal description of photo library

```
-- The context keyword names the component to which the conditions apply
context addItem

-- The preconditions specify what must be true before execution of addItem
pre:   PhotoLibrary.libSize() > 0
        PhotoLibrary.retrieve(pid) = null

-- The postconditions specify what is true after execution
post:  libSize () = libSize()@pre + 1
        PhotoLibrary.retrieve(pid) = p
        PhotoLibrary.catEntry(pid) = p photodesc

context delete

pre: PhotoLibrary.retrieve(pid) <> null ;

post: PhotoLibrary.retrieve(pid) = null
        PhotoLibrary.catEntry(pid) = PhotoLibrary.catEntry(pid)@pre
        PhotoLibrary.libSize() = libSize()@pre - 1
```

cmisc435 - 77

Photo library conditions

- As specified, the OCL associated with the Photo Library component states that:
 - ❑ There must not be a photograph in the library with the same identifier as the photograph to be entered;
 - ❑ The library must exist - assume that creating a library adds a single item to it;
 - ❑ Each new entry increases the size of the library by 1;
 - ❑ If you retrieve using the same identifier then you get back the photo that you added;
 - ❑ If you look up the catalogue using that identifier, then you get back the catalogue entry that you made.

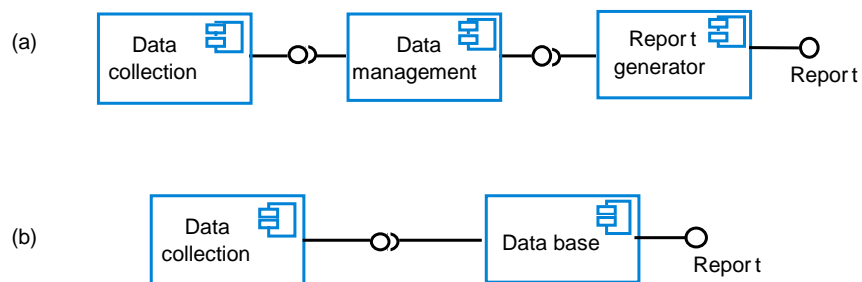
cmisc435 - 78

Composition trade-offs

- When composing components, you may find conflicts between functional and non-functional requirements, and conflicts between the need for rapid delivery and system evolution.
- You need to make decisions such as:
 - ❑ What composition of components is effective for delivering the functional requirements?
 - ❑ What composition of components allows for future change?
 - ❑ What will be the emergent properties of the composed system?

cmisc435 - 79

Data collection and report generation



cmisc435 - 80

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.

Key points

- CBSE is a reuse-based approach to defining and implementing loosely coupled components into systems.
- A component is a software unit whose functionality and dependencies are completely defined by its interfaces.
- A component model defines a set of standards that component providers and composers should follow.
- During the CBSE process, the processes of requirements engineering and system design are interleaved.
- Component composition is the process of 'wiring' components together to create a system.
- When composing reusable components, you normally have to write adaptors to reconcile different component interfaces.
- When choosing compositions, you have to consider required functionality, non-functional requirements and system evolution.

Configuration management

cmse435 - 83

Objectives

- To explain the importance of software configuration management (CM)
- To describe key CM activities namely CM planning, change management, version management and system building
- To discuss the use of CASE tools to support configuration management processes

cmse435 - 84

Configuration management

- Set of procedures that track
 - ❑ requirements that define what the system should do
 - ❑ design modules that are generated from requirements
 - ❑ program code that implements the design
 - ❑ tests that verify the functionality of the system
 - ❑ documents that describe the system

cm435 - 85

Configuration management

- New versions of software systems are created as they change:
 - ❑ For different machines/OS;
 - ❑ Offering different functionality;
 - ❑ Tailored for particular user requirements.
- Configuration management is concerned with managing evolving software systems:
 - ❑ System change is a team activity;
 - ❑ CM aims to control the costs and effort involved in making changes to a system.

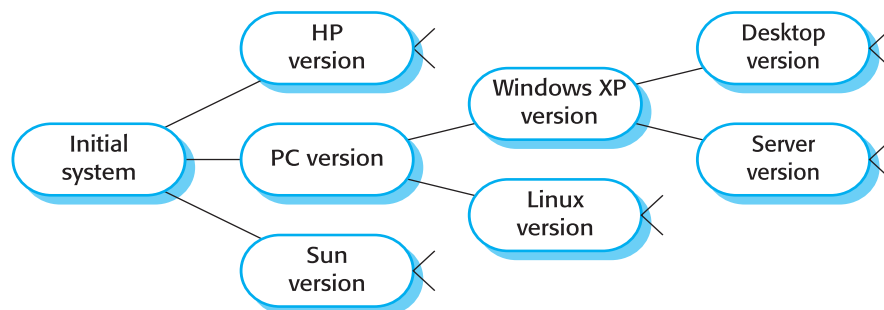
cm435 - 86

Configuration management

- Involves the development and application of procedures and standards to manage an evolving software product.
- CM may be seen as part of a more general quality management process.
- When released to CM, software systems are sometimes called *baselines* as they are a starting point for further development.

cmssc435 - 87

System families



cmssc435 - 88

Concurrent development and testing

- A time (say 2pm) for delivery of system components is agreed.
- A new version of a system is built from these components by compiling and linking them.
- This new version is delivered for testing using pre-defined tests.
- Faults that are discovered during testing are documented and returned to the system developers.

Frequent system building

- It is easier to find problems that stem from component interactions early in the process.
- This encourages thorough unit testing - developers are under pressure not to 'break the build'.
- A stringent change management process is required to keep track of problems that have been discovered and repaired.

Configuration management planning

- All products of the software process may have to be managed:
 - Specifications;
 - Designs;
 - Programs;
 - Test data;
 - User manuals.
- Thousands of separate documents may be generated for a large, complex software system.

The CM plan

- Defines the types of documents to be managed and a document naming scheme.
- Defines who takes responsibility for the CM procedures and creation of baselines.
- Defines policies for change control and version management.
- Defines the CM records which must be maintained.

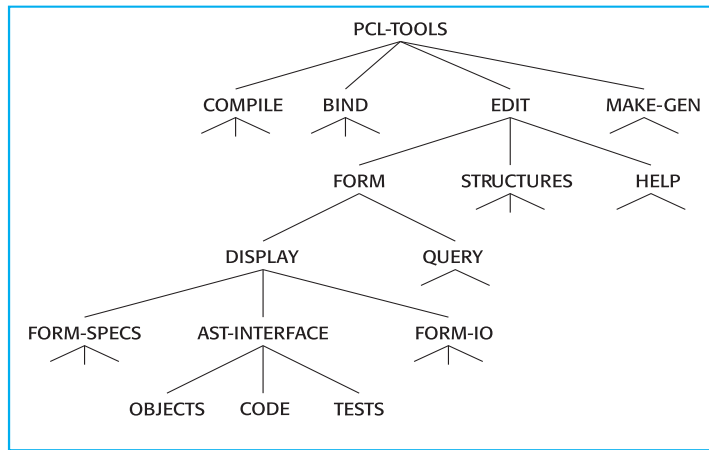
The CM plan

- Describes the tools which should be used to assist the CM process and any limitations on their use.
- Defines the process of tool use.
- Defines the CM database used to record configuration information.
- May include information such as the CM of external software, process auditing, etc.

Configuration item identification

- Large projects typically produce thousands of documents which must be uniquely identified.
- Some of these documents must be maintained for the lifetime of the software.
- Document naming scheme should be defined so that related documents have related names.
- A hierarchical scheme with multi-level names is probably the most flexible approach.
 - PCL-TOOLS/EDIT/FORMS/DISPLAY/AST-INTERFACE/CODE

Configuration hierarchy



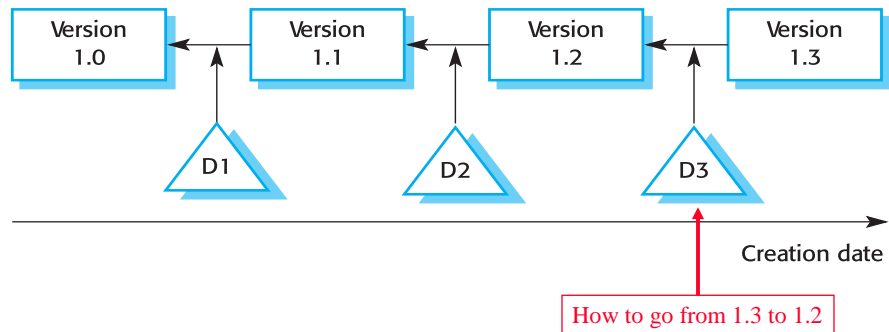
cmsc435 - 95

The configuration database

- All CM information should be maintained in a configuration database.
- This should allow queries about configurations to be answered:
 - Who has a particular system version?
 - What platform is required for a particular version?
 - What versions are affected by a change to component X?
 - How many reported faults in version T?
- The CM database should preferably be linked to the software being managed.
 - Use of CVS or RCS on dc cluster
 - Create RCS directory
 - `co [-l] filename, ci [-u] filename`

cmsc435 - 96

Delta-based versioning



cmisc435 - 97

Change management

- Software systems are subject to continual change requests:
 - From users;
 - From developers;
 - From market forces.
- Change management is concerned with keeping track of these changes and ensuring that they are implemented in the most cost-effective way.

cmisc435 - 98

Change request form

- The definition of a change request form is part of the CM planning process.
- This form records the change proposed, requestor of change, the reason why change was suggested and the urgency of change (from requestor of the change).
- It also records change evaluation, impact analysis, change cost and recommendations (System maintenance staff).

cmsc435 - 99

Example change request form

Change Request Form	
Project: Proteus/PCL-Tools	Number: 23/02
Change requester: I. Sommerville	Date: 1/12/02
Requested change: When a component is selected from the structure, display the name of the file where it is stored.	
Change analyser: G. Dean	Analysis date: 10/12/02
Components affected: Display-Icon.Select, Display-Icon.Display	
Associated components: FileTable	
Change assessment: Relatively simple to implement as a file name table is available. Requires the design and implementation of a display field. No changes to associated components are required.	
Change priority: Low	
Change implementation: Estimated effort: 0.5 days	
Date to CCB: 15/12/02	CCB decision date: 1/2/03
CCB decision: Accept change. Change to be implemented in Release 2.1.	
Change implementor:	Date of change:
Date submitted to QA:	QA decision:
Date submitted to CM:	
Comments	

This is not the same as a defect tracking system. How?

cmsc435 - 100

Change control board

- Changes should be reviewed by an external group who decide whether or not they are cost-effective from a strategic and organizational viewpoint rather than a technical viewpoint.
- Should be independent of project responsible for system. The group is sometimes called a change control board.
- The CCB may include representatives from client and contractor staff.

Change control process

- Problem discovered by or change requested by user/customer/developer, and recorded
- Change reported to the change control board
- CCB discusses problem: determines nature of change, who should pay
- CCB discusses source of problem, scope of change, time to fix; they assign severity/priority and analyst to fix
- Analyst makes change on test copy
- Analyst works with librarian to control installation of change
- Analyst files change report

Change control issues

- *Synchronization*: When was the change made?
- *Identification*: Who made the change?
- *Naming*: What components of the system were changed?
- *Authentication*: Was the change made correctly?
- *Authorization*: Who authorized that the change be made?
- *Routing*: Who was notified of the change?
- *Cancellation*: Who can cancel the request for change?
- *Delegation*: Who is responsible for the change?
- *Valuation*: What is the priority of the change?

Impact analysis

- **Workproduct**: any development artifact whose change is significant
- **Horizontal traceability**: relationships of components across collections of workproducts
- **Vertical traceability**: relationships among parts of a workproduct

Derivation history

- This is a record of changes applied to a document or code component.
- It should record, in outline, the change made, the rationale for the change, who made the change and when it was implemented.
- It may be included as a comment in code. If a standard prologue style is used for the derivation history, tools can process this automatically.

cmisc435 - 105

Component header information

```
// BANKSEC project (IST 6087)
//
// BANKSEC-TOOLS/AUTH/RBAC/USER_ROLE
//
// Object: currentRole
// Author: N. Perwaiz
// Creation date: 10th November 2002
//
// ©Lancaster University 2002
//
// Modification history
// Version      Modifier Date      Change      Reason
// 1.0          J. Jones          1/12/2002   Add header   Submitted to CM
// 1.1          N. Perwaiz       9/4/2003   New field    Change req. R07/02
```

cmisc435 - 106

Version and release management

- Invent an identification scheme for system versions.
- Plan when a new system version is to be produced.
- Ensure that version management procedures and tools are properly applied.
- Plan and distribute new system releases.

Versions/variants/releases

- **Version** An instance of a system which is functionally distinct in some way from other system instances.
- **Variant** An instance of a system which is functionally identical but non-functionally distinct from other instances of a system.
- **Release** An instance of a system which is distributed to users outside of the development team.

Version numbering

- Simple naming scheme uses a linear derivation
 - V1, V1.1, V1.2, V2.1, V2.2 etc.
- The actual derivation structure is a tree or a network rather than a sequence.
- Names are not meaningful.
- A hierarchical naming scheme leads to fewer errors in version identification.

Release management

- Releases must incorporate changes forced on the system by errors discovered by users and by hardware changes.
- They must also incorporate new system functionality.
- Release planning is concerned with when to issue a system version as a release.

System releases

- Not just a set of executable programs.
- May also include:
 - ❑ Configuration files defining how the release is configured for a particular installation;
 - ❑ Data files needed for system operation;
 - ❑ An installation program or shell script to install the system on target hardware;
 - ❑ Electronic and paper documentation;
 - ❑ Packaging and associated publicity.
- Systems are now normally released on optical disks (CD or DVD) or as downloadable installation files from the web.

Release problems

- Customer may not want a new release of the system
 - ❑ They may be happy with their current system as the new version may provide unwanted functionality.
- Release management should not assume that all previous releases have been accepted. All files required for a release should be re-created when a new release is installed.

Release decision making

- Preparing and distributing a system release is an expensive process.
- Factors such as the technical quality of the system, competition, marketing requirements and customer change requests should all influence the decision of when to issue a new system release.

System release strategy

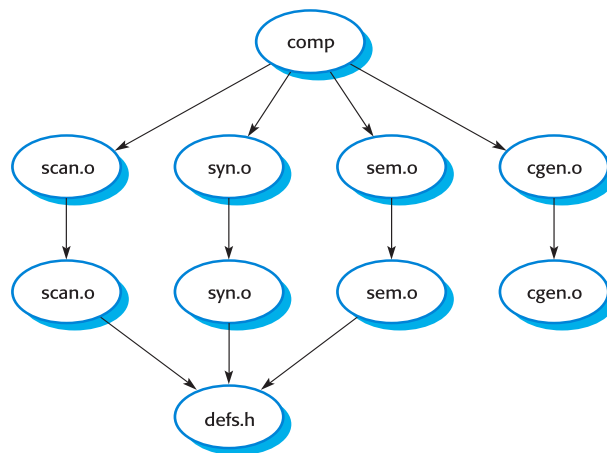
Factor	Description
Technical quality of the system	If serious system faults are reported which affect the way in which many customers use the system, it may be necessary to issue a fault repair release. However, minor system faults may be repaired by issuing patches (often distributed over the Internet) that can be applied to the current release of the system.
Platform changes	You may have to create a new release of a software application when a new version of the operating system platform is released.
Lehman's fifth law (see Chapter 21)	This suggests that the increment of functionality that is included in each release is approximately constant. Therefore, if there has been a system release with significant new functionality, then it may have to be followed by a repair release.
Competition	A new system release may be necessary because a competing product is available.
Marketing requirements	The marketing department of an organisation may have made a commitment for releases to be available at a particular date.
Customer change proposals	For customised systems, customers may have made and paid for a specific set of system change proposals and they expect a system release as soon as these have been implemented.

Release creation

- Release creation involves collecting all files and documentation required to create a system release.
- Configuration descriptions have to be written for different hardware and installation scripts have to be written.
- The specific release must be documented to record exactly what files were used to create it. This allows it to be re-created if necessary.
- System building is now always supported by automated tools that are driven by 'build scripts' (e.g., make).

cmisc435 - 115

Component dependencies



cmisc435 - 116

System building problems

- Do the build instructions include all required components?
 - ❑ When there are many hundreds of components making up a system, it is easy to miss one out. This should normally be detected by the linker.
- Is the appropriate component version specified?
 - ❑ A more significant problem. A system built with the wrong version may work initially but fail after delivery.
- Are all data files available?
 - ❑ The build should not rely on 'standard' data files. Standards vary from place to place.

System building problems

- Are data file references within components correct?
 - ❑ Embedding absolute names in code almost always causes problems as naming conventions differ from place to place.
- Is the system being built for the right platform
 - ❑ Sometimes you must build for a specific OS version or hardware configuration.
- Is the right version of the compiler and other software tools specified?
 - ❑ Different compiler versions may actually generate different code and the compiled component will exhibit different behaviour.

CASE tools for configuration management

- CM processes are standardized and involve applying pre-defined procedures.
- Large amounts of data must be managed.
- CASE tool support for CM is therefore essential.
- Mature CASE tools to support configuration management are available ranging from stand-alone tools to integrated CM workbenches.

CM workbenches

- Open workbenches
 - Tools for each stage in the CM process are integrated through organizational procedures and scripts. Gives flexibility in tool selection.
- Integrated workbenches
 - Provide whole-process, integrated support for configuration management. More tightly integrated tools so easier to use. However, the cost is less flexibility in the tools used.

Change management tools

- Change management is a procedural process so it can be modelled and integrated with a version management system.
- Change management tools
 - ❑ Form editor to support processing the change request forms;
 - ❑ Workflow system to define who does what and to automate information transfer;
 - ❑ Change database that manages change proposals and is linked to a VM system;
 - ❑ Change reporting system that generates management reports on the status of change requests.

Version management tools

- Version and release identification
 - ❑ Systems assign identifiers automatically when a new version is submitted to the system.
- Storage management.
 - ❑ System stores the differences between versions rather than all the version code.
- Change history recording
 - ❑ Record reasons for version creation.
- Independent development
 - ❑ Only one version at a time may be checked out for change. Parallel working on different versions.
- Project support
 - ❑ Can manage groups of files associated with a project rather than just single files.

System building

- Building a large system is computationally expensive and may take several hours.
- Hundreds of files may be involved.
- System building tools may provide
 - ❑ A dependency specification language and interpreter;
 - ❑ Tool selection and instantiation support;
 - ❑ Distributed compilation;
 - ❑ Derived object management.

cmisc435 - 123

Key points

- Configuration management is the management of system change to software products.
- A formal document naming scheme should be established and documents should be managed in a database.
- The configuration data base should record information about changes and change requests.
- Version identification should be established using version numbers, attributes or change sets.
- System releases include executable code, data, configuration files and documentation.
- CASE tools may be stand-alone tools or may be integrated systems which integrate support for version management, system building and change management.

cmisc435 - 124