MATA KULIAH : Rekayasa Piranti Lunak

KODE MATA KULIAH/SKS : TI1014– 4 SKS

TAHUN : **2013**

VERSI : **1.0**

KALBIS Institute

Science • Technology • Business

EDUCATION FOR A BETTER LIFE

KEMAMPUAN AKHIR YANG DIHARAPKAN

Mahasiswa dapat menerapkan prinsip-prinsip Rekayasa Piranti Lunak dalam proyek pengembangan Piranti Lunak

MATERI POKOK

- Penggunaan kembali (reuse) lanskap
- Desain pola
- Penggunaan kembali berbasis Generator
- Kerangka Piranti Lunak penggunaan kembali sistem

Sumber Pustaka

1. A. S. Rosa, Salahuddin M., Rekayasa Perangkat Lunak Terstruktur dan Berorientasi Objek, Informatika, 20013.
2. Ian Sommerville, Software Engineering 9th Edition, Addison-Wesley, 2011.
3. Roger S. Pressman, Software Engineering: A Practitioner's Approach Seventh Edition, McGraw-Hill, 2010.
4. Yasin Verdi, Rekayasa Perangkat Lunak Berorientasi Objek, Mitra Wacana Media, 2012.

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 recognised 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

- 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.

Reuse benefits 1

Increased dependability	Reused software, that has been tried and tested in working systems, should be m ore 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.

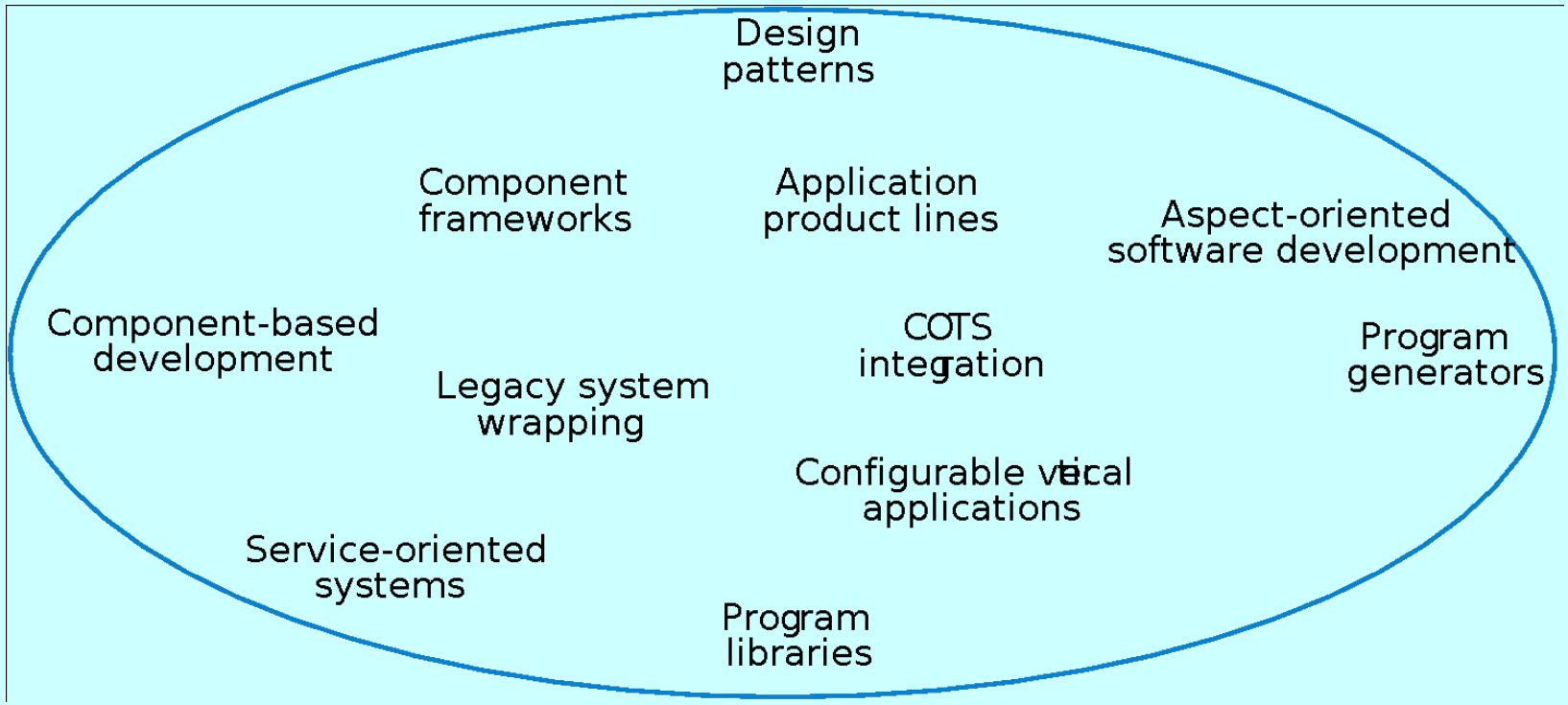
Reuse problems 1

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 peopleÕs software.

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.

The reuse landscape



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 ÔwrappedÕ 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.

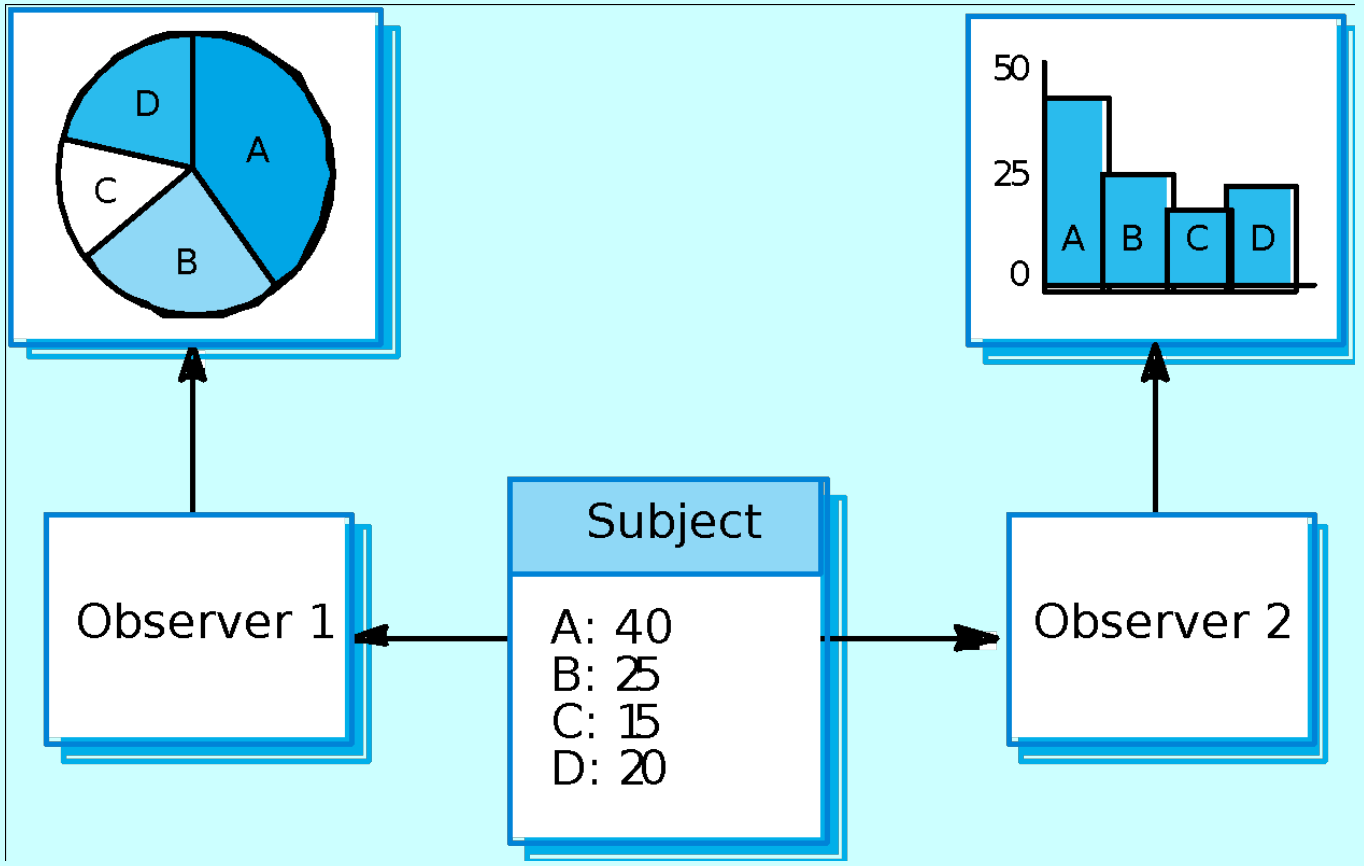
Design patterns

- 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.

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.

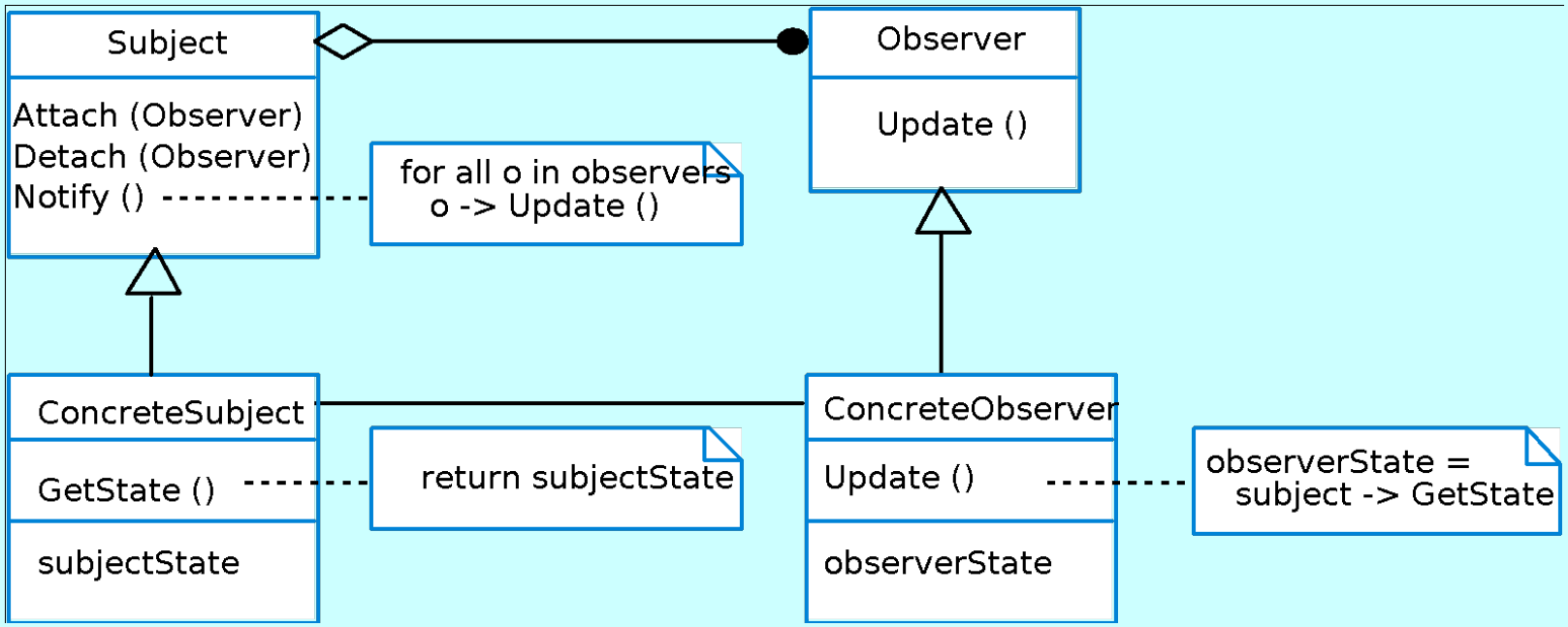
Multiple displays



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
  - Optimisations to enhance display performance are impractical.

The Observer pattern



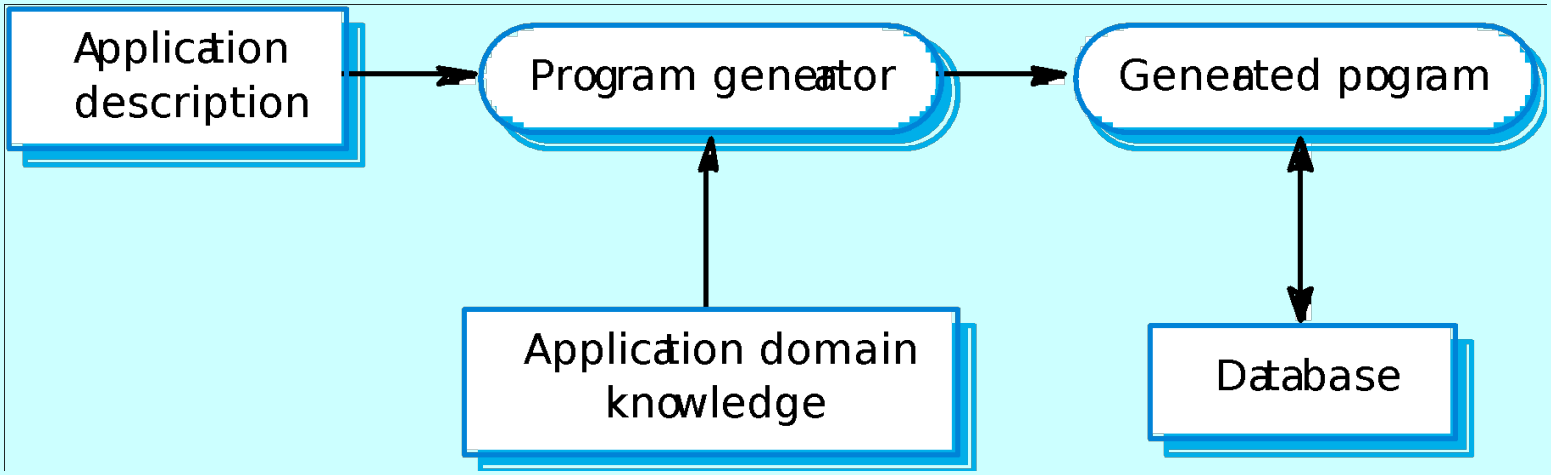
Generator-based reuse

- Program generators involve the reuse of standard patterns and algorithms.
- These are embedded in the generator and parameterised 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.

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



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

- 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.

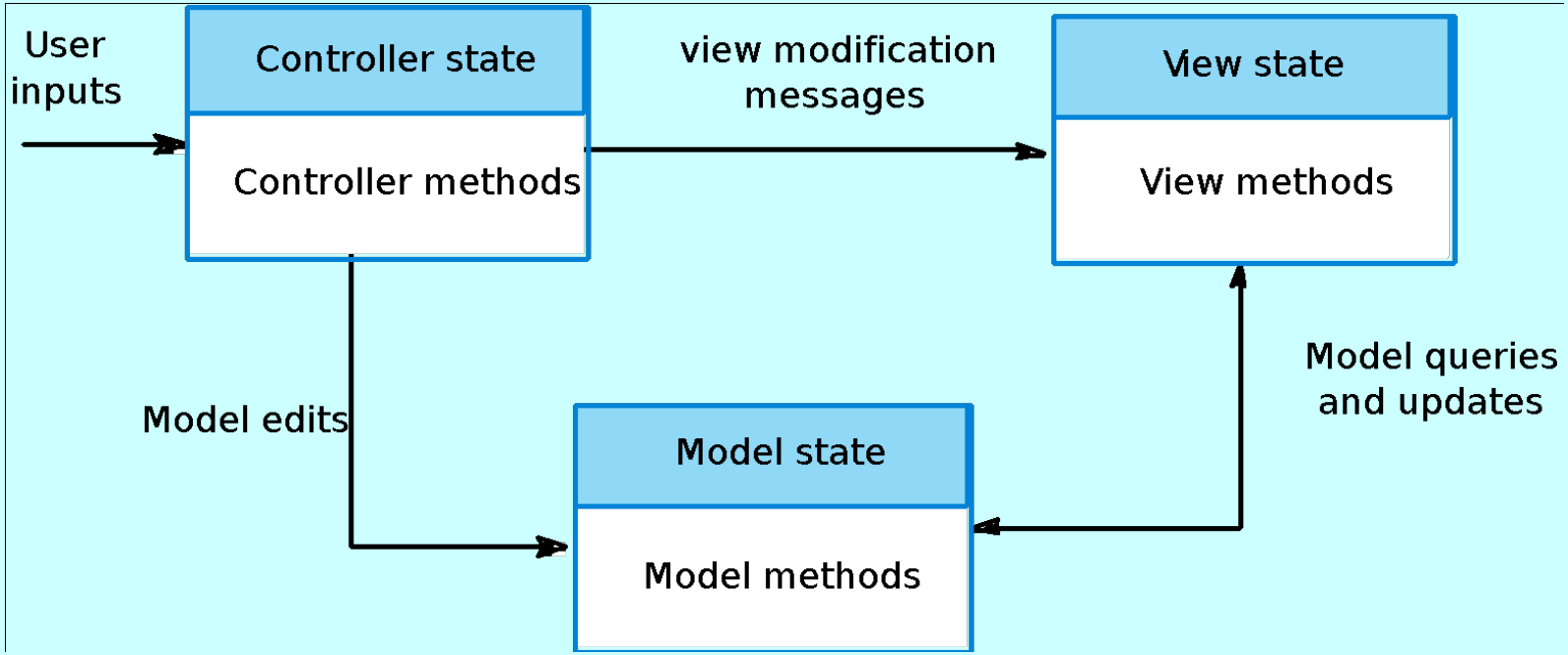
Extending frameworks

- 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).

Model-view-controller



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.

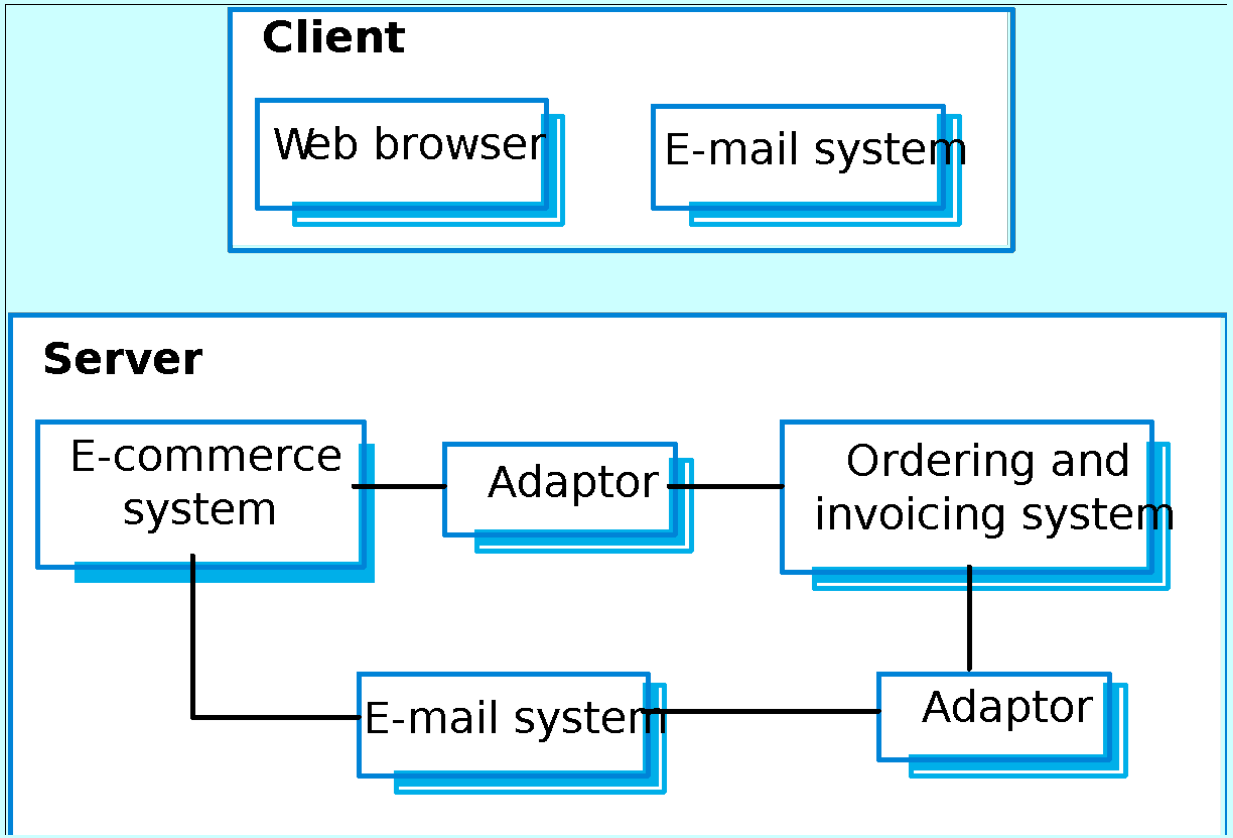
COTS product reuse

- 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.

E-procurement system



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.

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
- Support from COTS vendors
  - COTS vendors may not offer support over the lifetime of the product

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.

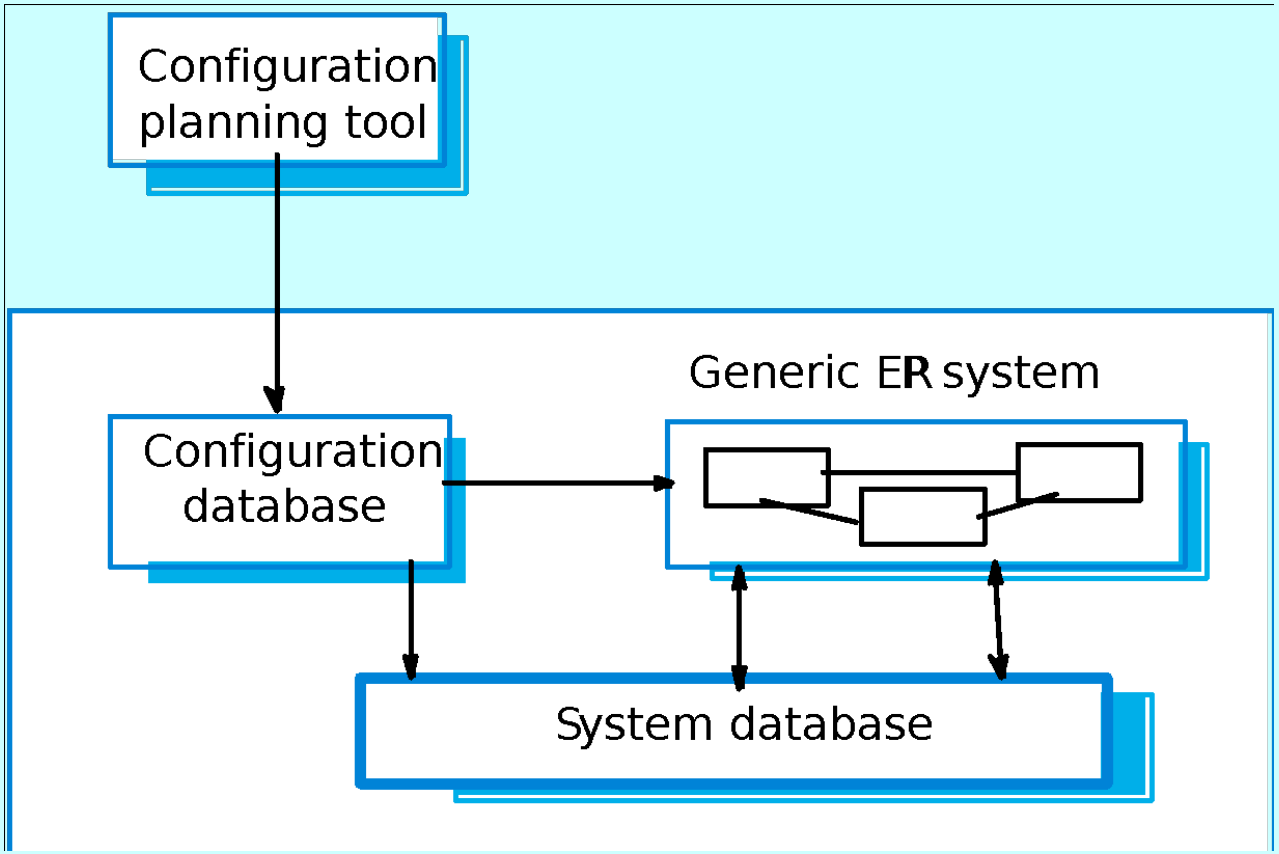
COTS product specialisation

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

ERP system organisation



ERP systems

- An Enterprise Resource Planning (ERP) system is a generic system that supports common business processes such as ordering and invoicing, manufacturing, etc.
- These are very widely used in large companies - they represent probably the most common form of software reuse.
- The generic core is adapted by including modules and by incorporating knowledge of business processes and rules.

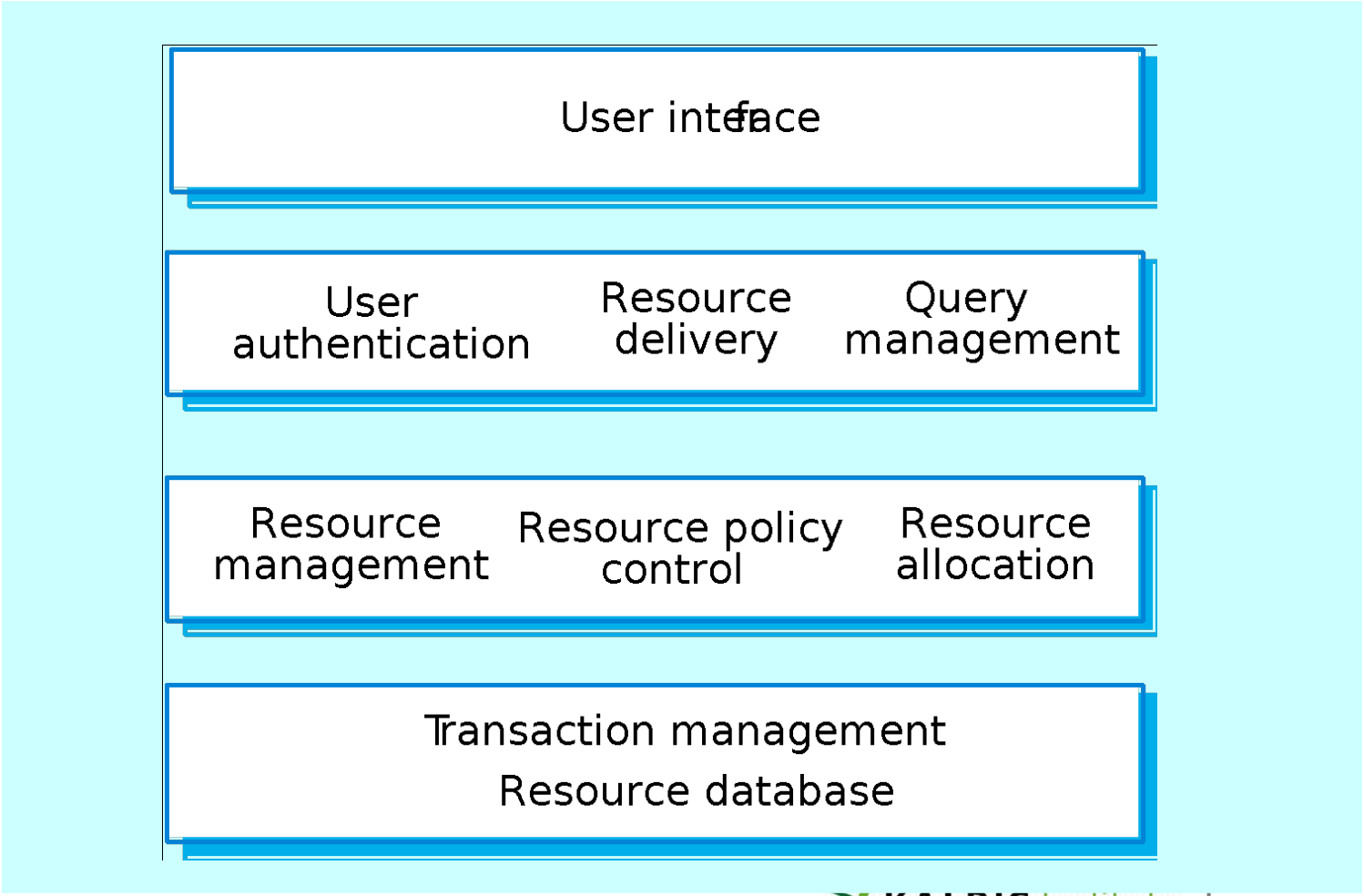
Design time configuration

- Software product lines that are configured at design time are instantiations of generic application architectures as discussed in Chapter 13.
- Generic products usually emerge after experience with specific products.

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.

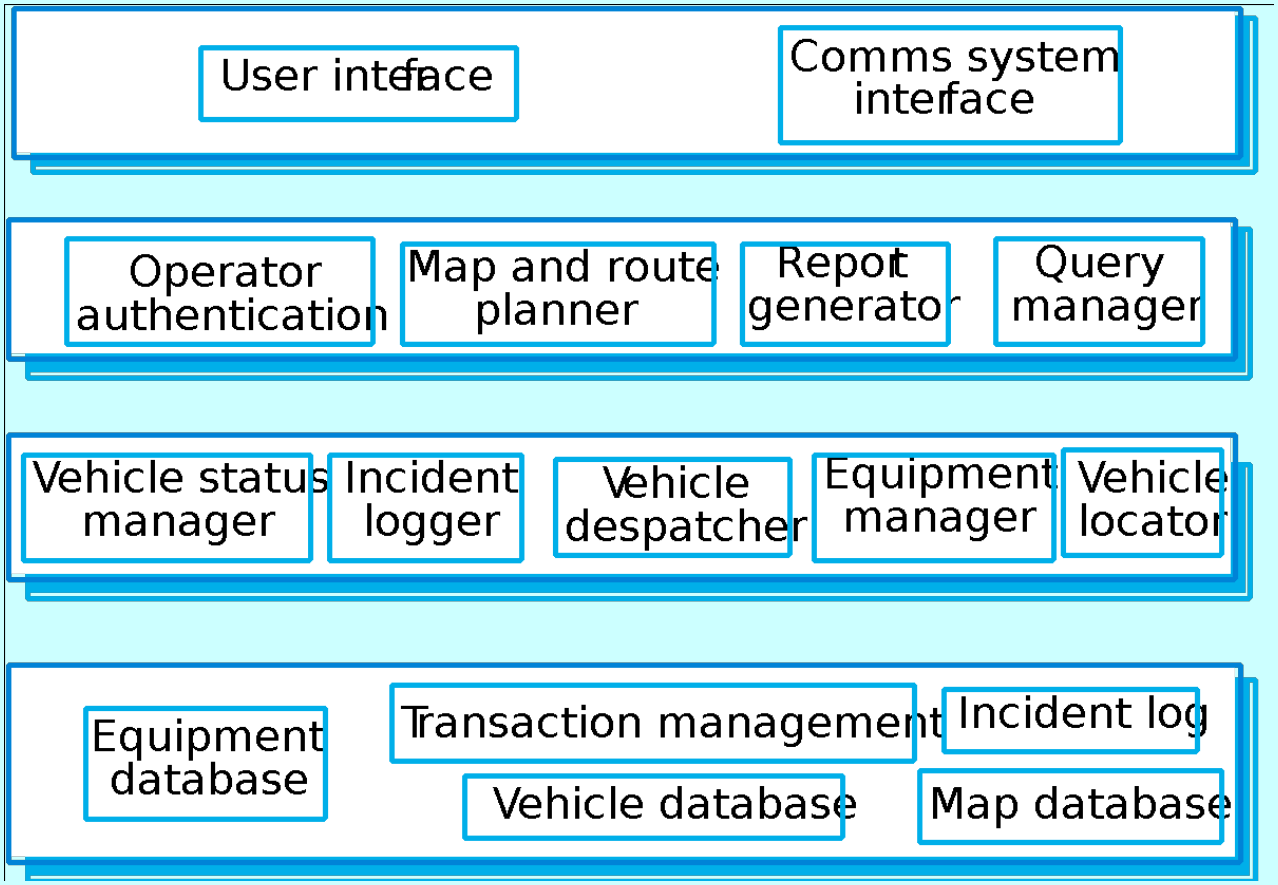
A resource management system



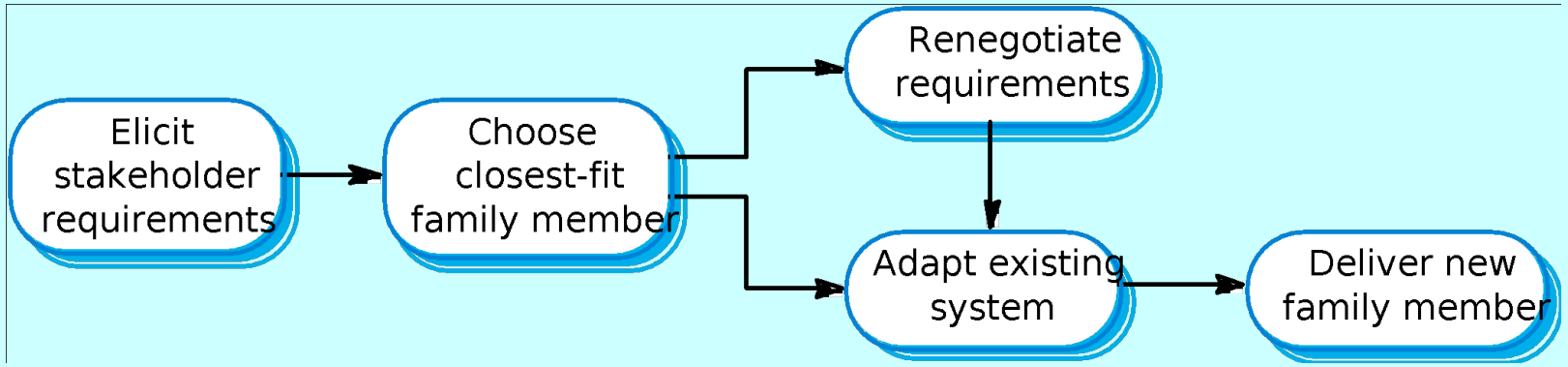
Vehicle despatching

- A specialised resource management system where the aim is to allocate resources (vehicles) to handle incidents.
- Adaptations include:
  - At the UI level, there are components for operator display and communications;
  - At the I/O management level, there are components that handle authentication, reporting and route planning;
  - At the resource management level, there are components for vehicle location and despatch, managing vehicle status and incident logging;
  - The database includes equipment, vehicle and map databases.

A despatching system



Product instance development



Product instance development

- Elicit stakeholder requirements
  - Use existing family member as a prototype
- Choose closest-fit family member
  - Find the family member that best meets the requirements
- Re-negotiate requirements
  - Adapt requirements as necessary to capabilities of the software
- Adapt existing system
  - Develop new modules and make changes for family member
- Deliver new family member
  - Document key features for further member development

SUMMARY

KALBIS Institute

Science • Technology • Business

EDUCATION FOR A BETTER LIFE

KALBIS Institute

Science • Technology • Business

EDUCATION FOR A BETTER LIFE