

# Software Reuse and Component-Based SE

ITSE422

---

## Lecture #1: Software Reuse (An Introduction)

# Main References

---

- ▶ Ian Sommerville, *Software Engineering*, 10<sup>th</sup> edition, chapter 15, 16 (Software Reuse & Components and component models)
- ▶ Ivica Crnkovic, Magnus Larsson. *Building reliable component based software systems*, Artech House, 2002.
- ▶ Roger S. Pressman, *Software Engineering: A Practitioner's Approach*, Eighth Edition, McGraw-Hill Higher Education, 2015
- ▶ **Component-Based Software Engineering Methods and Metrics.** Umesh Kumar Tiwari and Santosh Kumar, 2021 CRC Press is an imprint of Taylor & Francis Group, LLC.

# Topics covered

---

- ▶ The reuse landscape
- ▶ Application frameworks

# 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 a design process that is based on systematic software reuse.
- ▶ There has been a major switch to reuse-based development over the past 10 years.

# 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.
- ▶ **Object and function reuse**
  - ▶ Software components that implement a single well-defined object or function may be reused.

# Benefits of software reuse

---

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.

# Benefits of software reuse

---

Benefit	Explanation
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.

# Problems with reuse

---

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.



# Problems with reuse

---

Problem	Explanation
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, 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 include a component search as part of their normal development process.

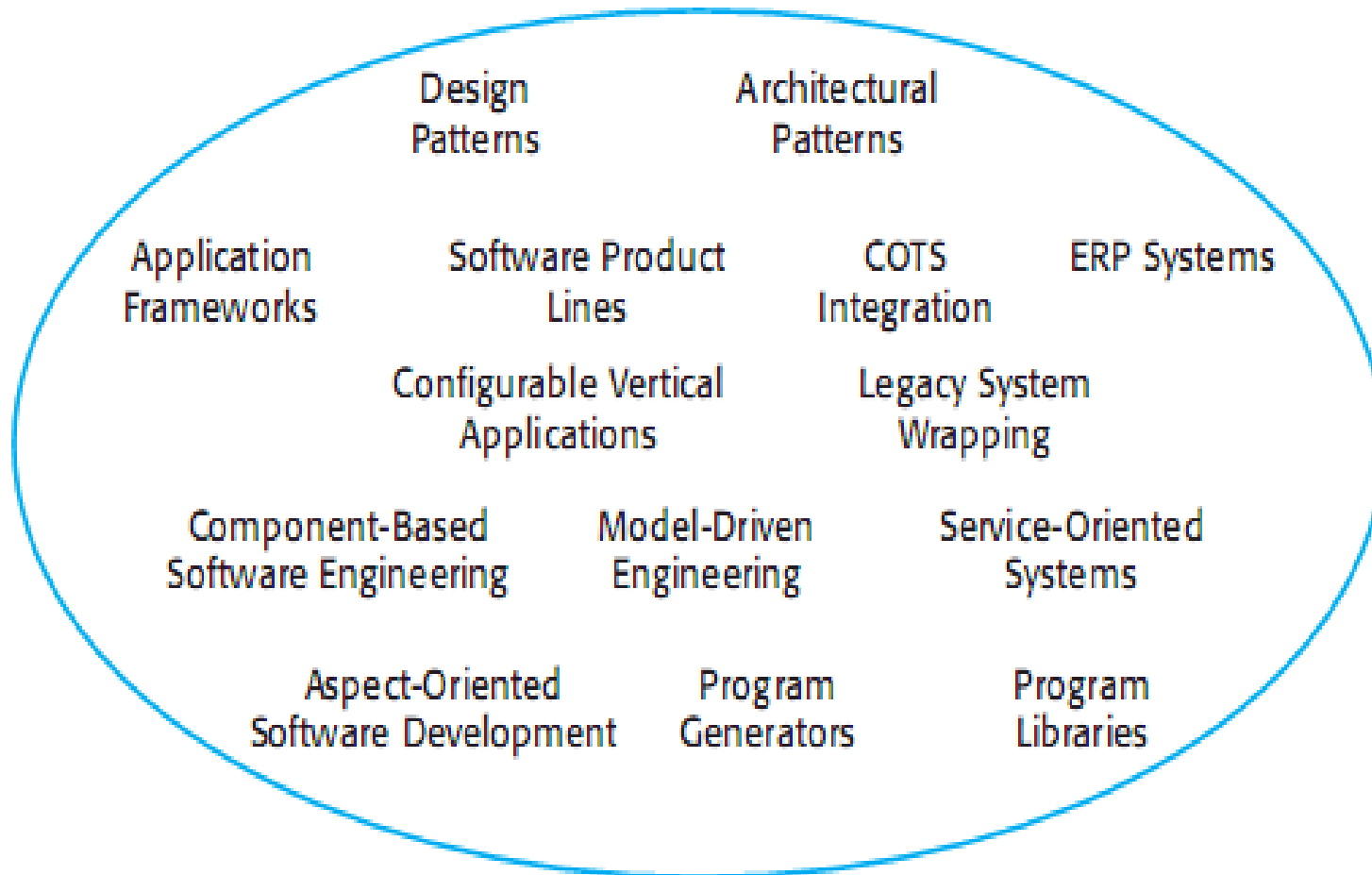
# The reuse landscape

---

- ▶ 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

---



# Approaches that support software reuse

---

Approach	Description
Architectural patterns	Standard software architectures that support common types of application systems are used as the basis of applications. Described in Chapters 6, 11, and 17.
Design patterns	Generic abstractions that occur across applications are represented as design patterns showing abstract and concrete objects and interactions. Described in Chapter 7.
Component-based development	Systems are developed by integrating components (collections of objects) that conform to component-model standards. Described in Chapter 16.
Application frameworks	Collections of abstract and concrete classes are adapted and extended to create application systems.
Legacy system wrapping	Legacy systems (see Chapter 9) are 'wrapped' by defining a set of interfaces and providing access to these legacy systems through these interfaces.

# Approaches that support software reuse

---

Approach	Description
Service-oriented systems	Systems are developed by linking shared services, which may be externally provided. Described in Chapter 18.
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.

# Approaches that support software reuse

---

Approach	Description
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. Described in Chapter 5.
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. Described in Chapter 31.

# 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.

---

# Application frameworks





# Framework definition

---

*“..an integrated set of software artefacts (such as classes, objects and components) that collaborate to provide a reusable architecture for a family of related applications.”*

# Application frameworks

---

- ▶ Frameworks are moderately large entities that can be reused. They are somewhere between system and component reuse.
- ▶ 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.

# 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.

# Web application frameworks

---

- ▶ Support the construction of dynamic websites as a front-end for web applications.
- ▶ WAFs (Web application frameworks) are now available for all of the commonly used web programming languages e.g. Java, Python, Ruby, etc.
- ▶ Interaction model is based on the Model-View-Controller composite pattern.

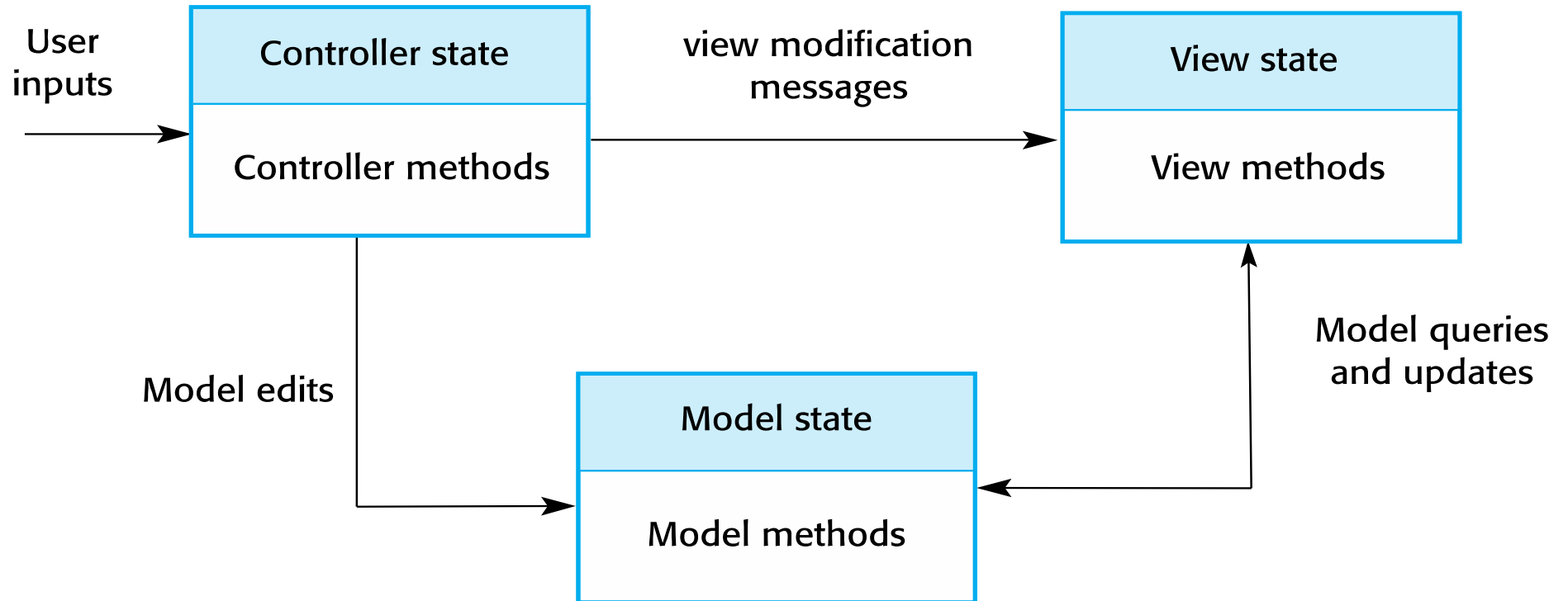
# Model-view controller

---

- ▶ 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.

# The Model-View-Controller pattern

---



# WAF features

---

- ▶ *Security*
  - ▶ WAFs may include classes to help implement user authentication (login) and access.
- ▶ *Dynamic web pages*
  - ▶ Classes are provided to help you define web page templates and to populate these dynamically from the system database.
- ▶ *Database support*
  - ▶ 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.

# Extending frameworks

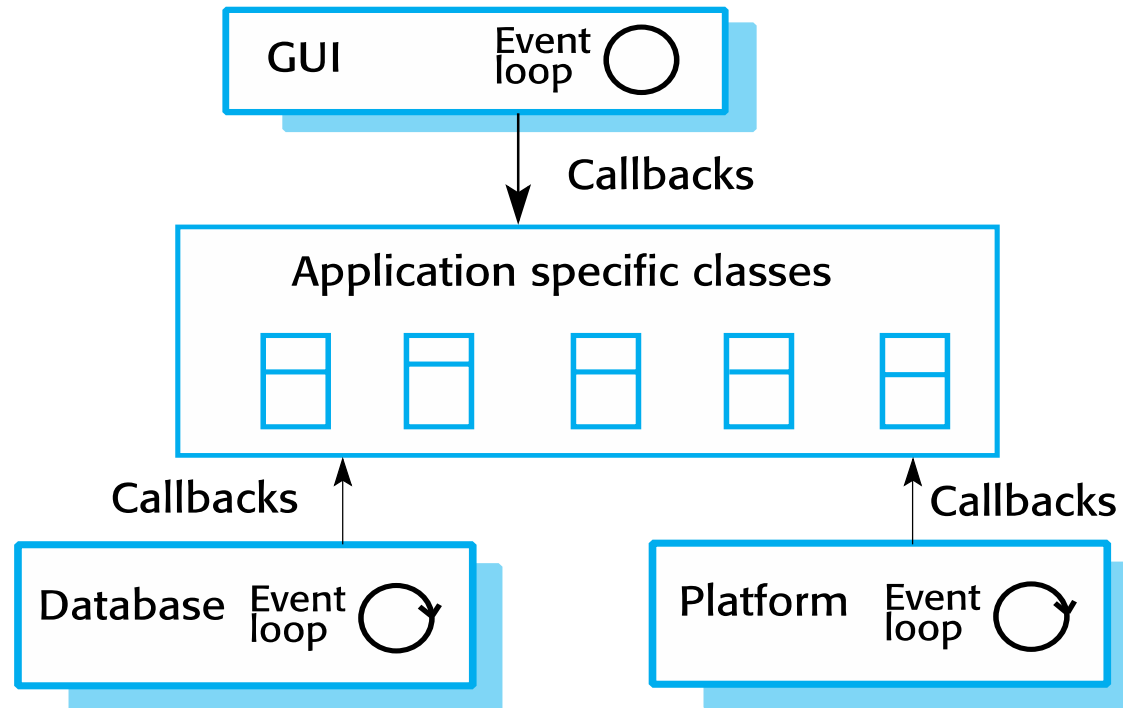
---

- ▶ Frameworks are generic and are extended to create a more specific application or sub-system. They provide a skeleton architecture for the 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.



# Inversion of control in frameworks

---



# Key points

---

- ▶ 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.

---

▶ **Questions?**