Implementation Issues

Section 7.3 (Sommerville)

Fall Semester 2021

1st Semester 1443 H

Implementation issues

Focus here is not on programming, although this is obviously important, but on other implementation issues:

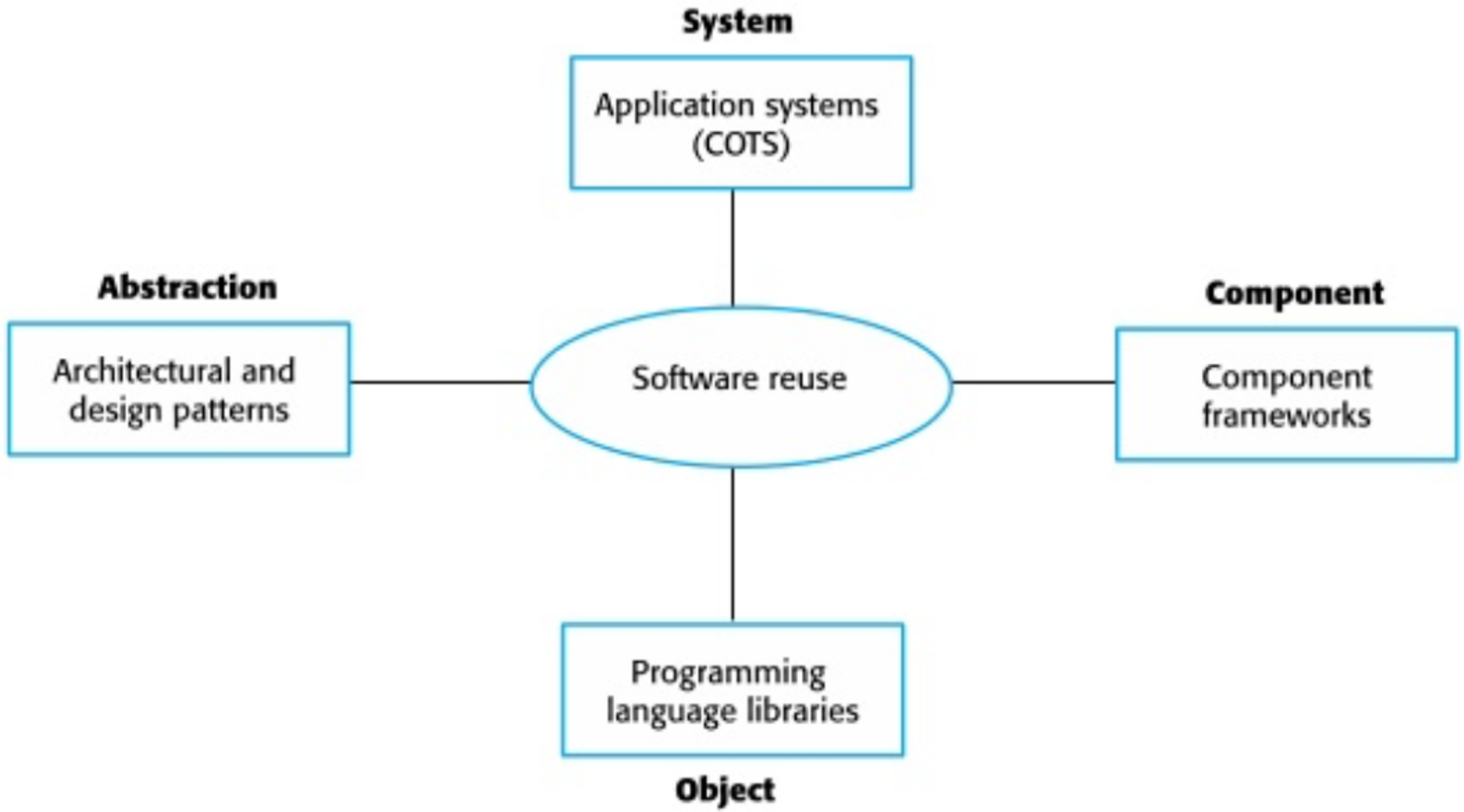
- **Reuse**
- **Configuration management**
- **Host-target development**

Software Reuse

Most modern software is constructed by reusing existing components or systems. When you are developing software, you should make as much use as possible of existing code.

- From the 1960s to the 1990s, most new software was developed from scratch.
  - The only significant reuse was the reuse of functions/objects in programming language libraries.
- **Costs and schedule pressure** mean that this approach became increasingly unviable, especially for commercial and Internet-based systems.
- An approach to development based around the reuse of existing software emerged and is now generally used for business and scientific software.

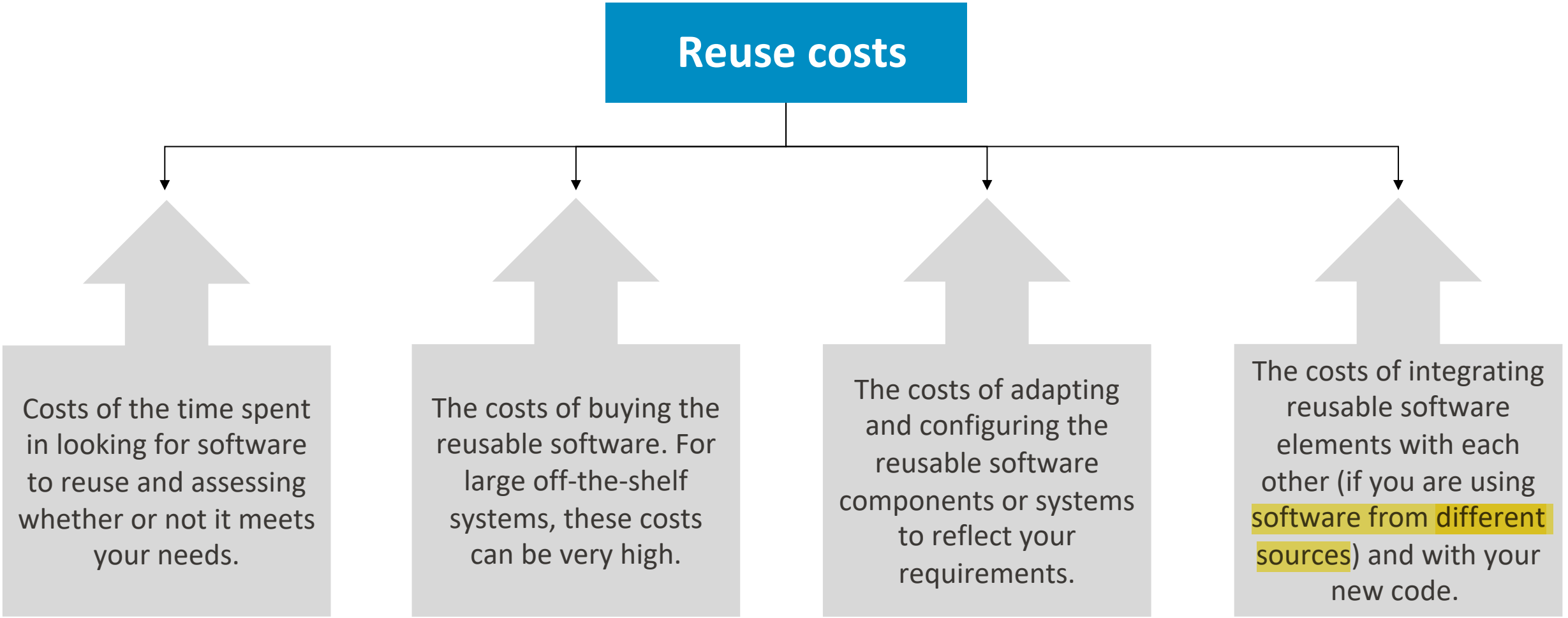
Software Reuse levels



Software Reuse levels

Reuse level	Definition	Example
Abstraction level	You don't reuse software directly but use knowledge of successful abstractions in the design of your software.	Architectural patterns
Object level	You directly reuse objects from a library rather than writing the code yourself.	Using objects and methods from a JavaMail library to process mail messages.
Component level	Components are collections of objects and object classes that you reuse in application systems.	User interface
System level	You reuse entire application systems.	Online shopping system

Reuse Costs


Reuse costs
Costs of the time spent in looking for software to reuse and assessing whether or not it meets your needs.
The costs of buying the reusable software. For large off-the-shelf systems, these costs can be very high.
The costs of adapting and configuring the reusable software components or systems to reflect your requirements.
The costs of integrating reusable software elements with each other (if you are using software from different sources) and with your new code.

Configuration Management

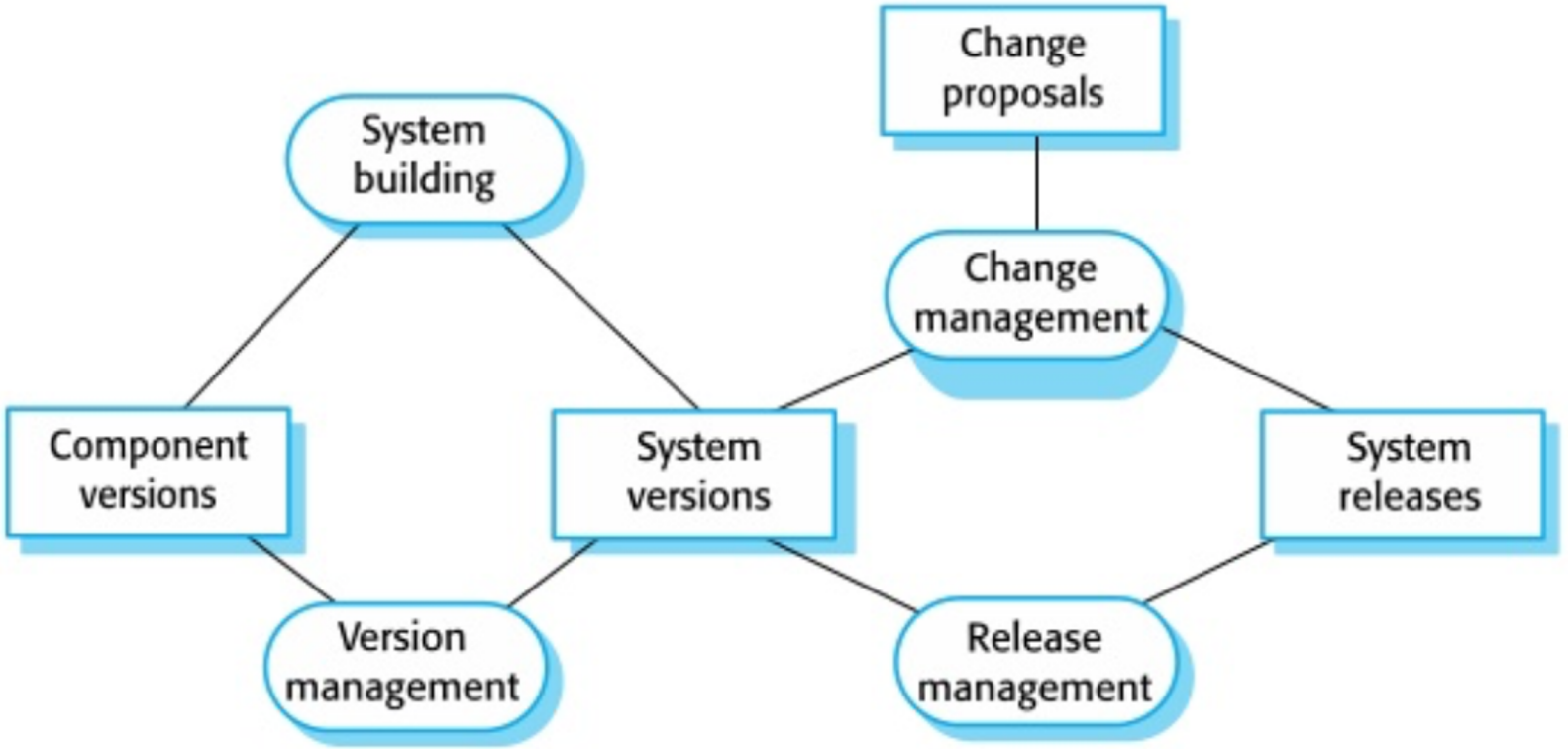
During the development process, you have to keep track of the many different versions of each software component in a configuration management system.

- Configuration management is the name given to the general process of **managing a changing software** system.
- The aim of configuration management is to support the system integration process so that **all developers can access the project code and documents in a controlled way**, find out what changes have been made, and compile and link components to create a system.

Configuration Management Activities

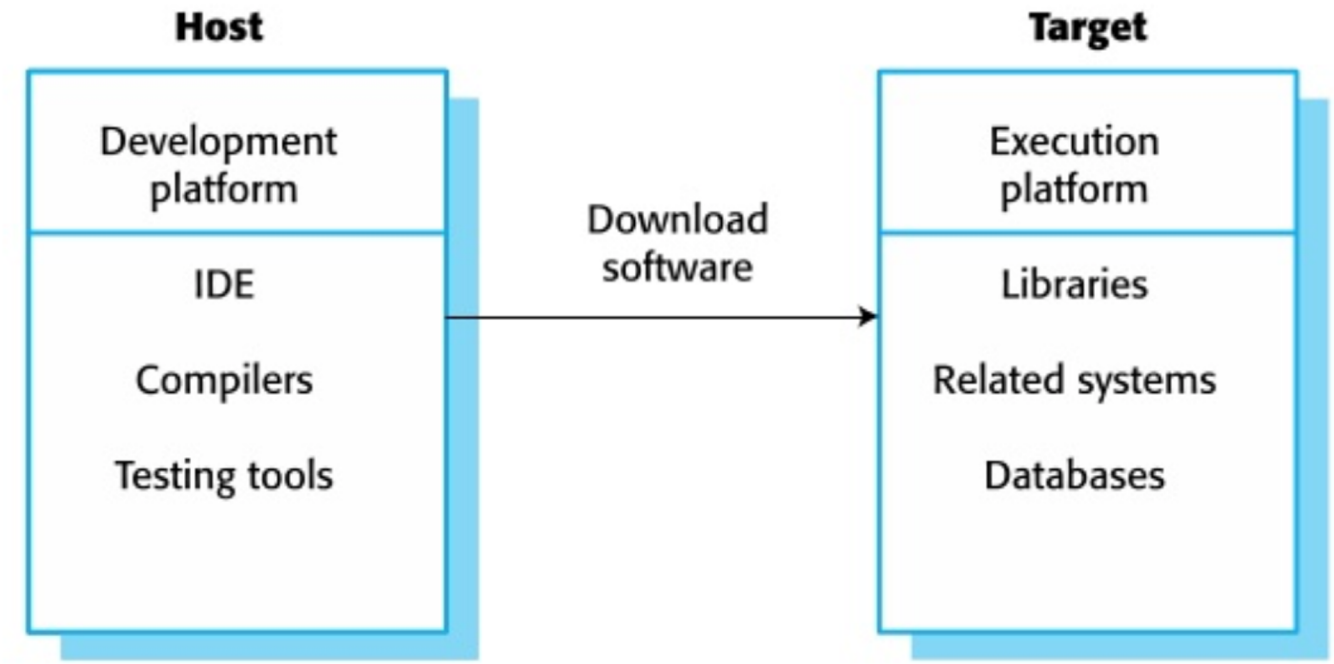
Activity	Description
Version management	Support is provided to keep track of the different versions of software components. Version management systems include facilities to coordinate development by several programmers.
System integration	Support is provided to help developers define what versions of components are used to create each version of a system. This description is then used to build a system automatically by compiling and linking the required components.
Problem tracking	Support is provided to allow users to report bugs and other problems, and to allow all developers to see who is working on these problems and when they are fixed.

Configuration Management Tool Interaction



Host-target Development

Production software does not usually execute on the same computer as the software development environment. Rather, you develop it on one computer (the host system) and execute it on a separate computer (the target system).



Host-target Development

- Most software is developed on one computer (the host) but runs on a separate machine (the target).
- More generally, we can talk about a **development platform** and an **execution platform**.
- A **platform** is more than just hardware.
  - It includes the installed operating system plus other supporting software such as a database management system or, for development platforms, an interactive development environment.
  - Development platform usually has different installed software than execution platform; these platforms may have different architectures.

Software Development Platform Tools

1. An integrated compiler and syntax-directed editing system that allows you to create, edit and compile code.

2. A language debugging system.

3. Graphical editing tools, such as tools to edit UML models.

4. Testing tools that can automatically run a set of tests on a new version of a program.

5. Project support tools that help you organize the code for different development projects.

Integrated Development Environments (IDEs)

- Software development tools are often grouped to create an integrated development environment (IDE).
- An IDE is a set of software tools that supports different aspects of software development, within some common framework and user interface.
- IDEs are created to support development in a specific programming language such as Java. The language IDE may be developed specially, or may be an instantiation of a general-purpose IDE, with specific language-support tools.

Component/System Deployment Factors

- If a **component is designed for a specific hardware architecture**, or relies on some other software system, it must obviously **be deployed on a platform that provides the required hardware and software support**.

- **High availability systems** may require components to be **deployed on more than one platform**. This means that, in the **event of platform failure, an alternative implementation** of the component is available.

- If there is a **high level of communications** traffic between components, it usually makes sense to deploy them on the **same platform or on platforms** that are physically close to one other. This **reduces the delay** between the time a message is sent by one component and received by another.

Thank you