INSTITUTO SUPERIOR DE TECNOLOGIAS AVANÇADAS DE LISBOA

# Software Engineering

**João Caldeira**

**Invited Professor**

Email. joaocarlos.caldeira@my.istec.pt

Mob. +351 917769544

**January 5, 2023**

INSTITUTO SUPERIOR DE TECNOLOGIAS AVANÇADAS DE LISBOA

# Software Construction

January 5, 2023

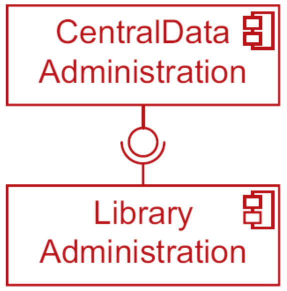
## Software Construction

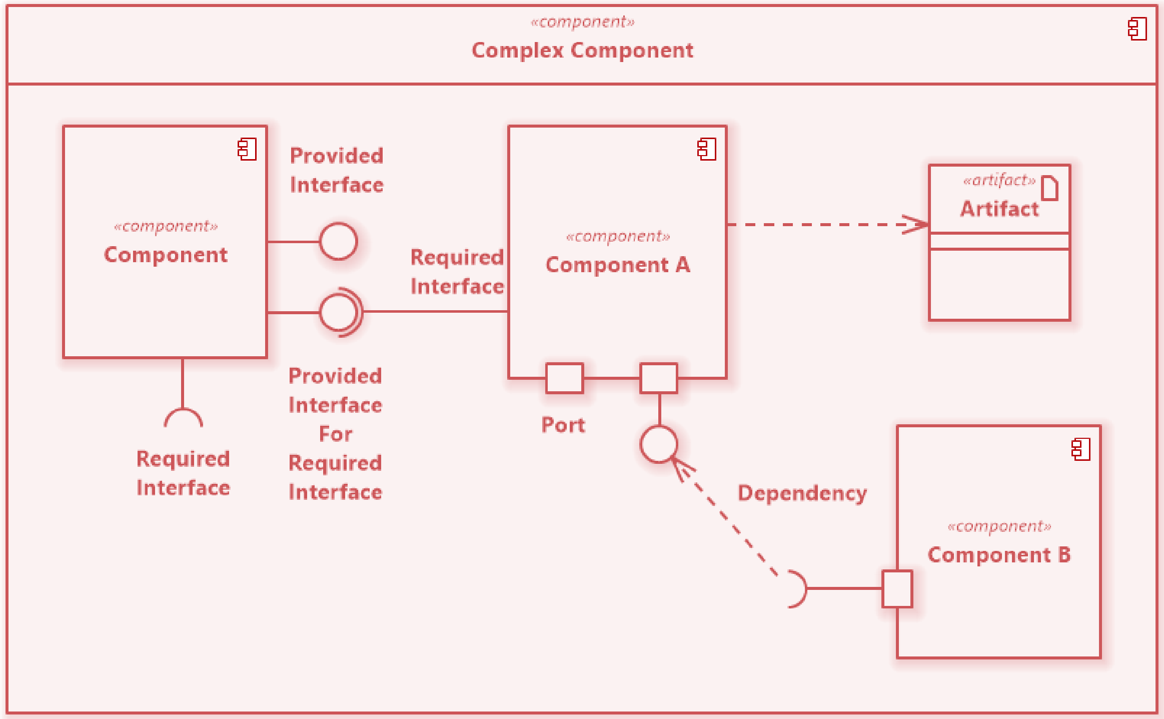
Structural Models

- **Components (Diagrams)**

1. A component is an independent, executable unit that provides other components with services or uses the services of other components

2. When specifying a component, you can model two views explicitly:
   - **The external view.** Which represents the specification of the component
   - **The internal view.** Which defines the implementation of the component





## Software Construction

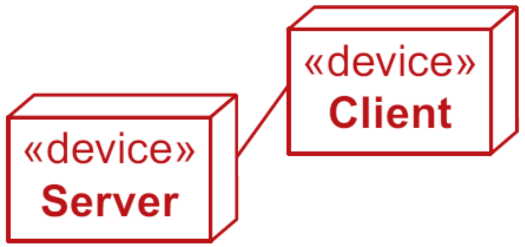
Structural Models

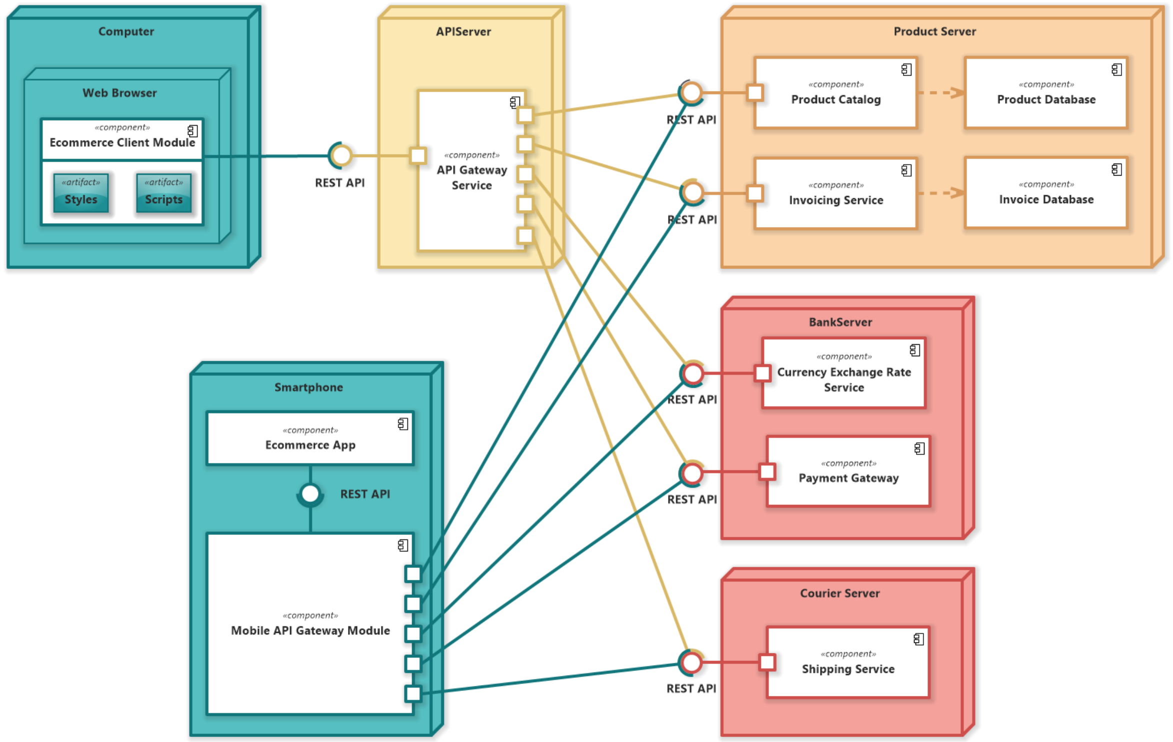
- **Deployment (Diagrams)**

  1. Represents the hardware topology used and the runtime system assigned

  2. The hardware encompasses processing units in the form of nodes as well as communication relationships between the nodes

  3. A runtime system contains artifacts that are deployed to the nodes





## Software Construction

### Implementation Issues

- **Concerns**

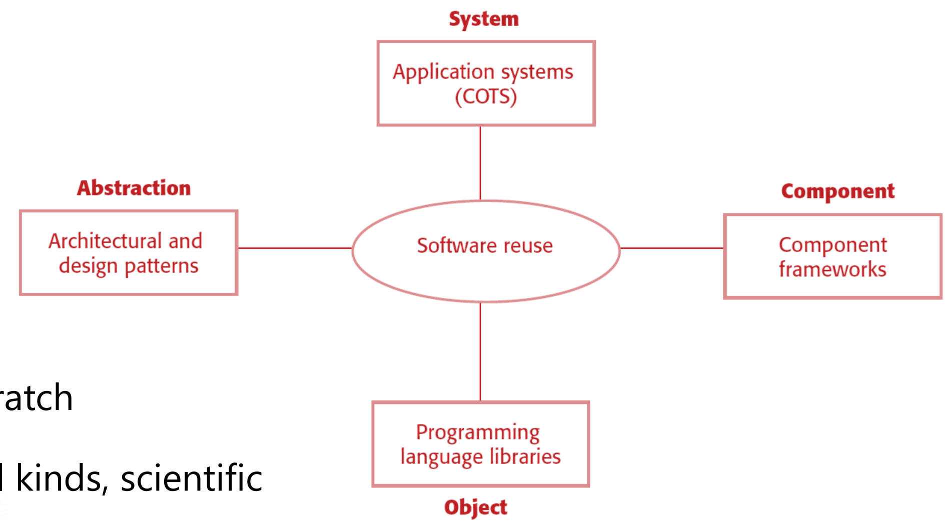
1. **Reuse.** Most modern software is constructed by reusing existing components or systems. When you are developing software, **you should make as much use as possible of existing code.**

2. **Configuration management.** During the development process, many different versions of each software component are created. If you don't keep track of these versions in a configuration management system, **you are liable to include the wrong versions of these components in your system.**

3. **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). **The host and target systems are sometimes of the same type, but often they are completely different.**

## Software Construction

### Software Reuse


System
Application systems (COTS)
Abstraction
Architectural and design patterns
Software reuse
Component
Component frameworks
Programming language libraries
Object

- **Reuse**
  - From the 1960s to the 1990s, most new software was developed from scratch
  - A reuse-based approach is now widely used for web-based systems of all kinds, scientific software, and, increasingly, in embedded systems engineering

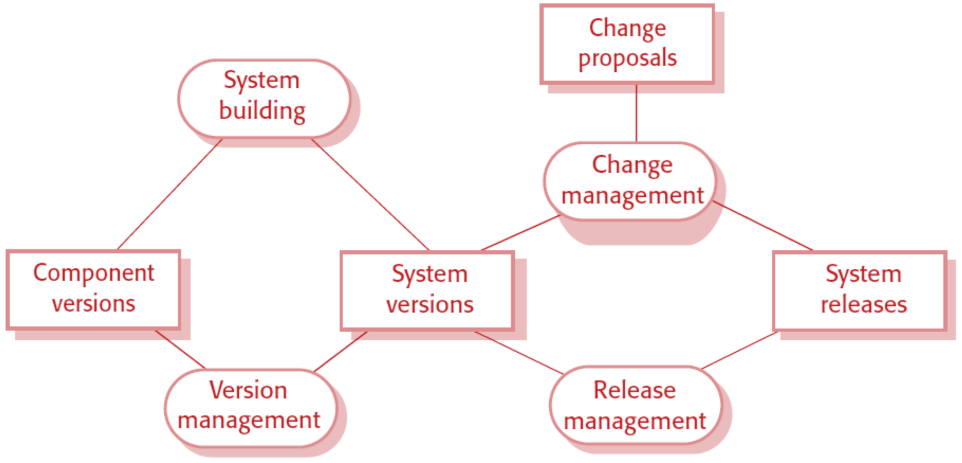
1. **The abstraction level.** At this level, you don’t reuse software directly but rather use knowledge of successful abstractions in the design of your software
2. **The object level.** At this level, you directly reuse objects from a library rather than writing the code yourself.
3. **The component level.** Components are collections of objects and object classes that operate together to provide related functions and services. You often have to adapt and extend the component by adding some code of your own
4. **The system level.** At this level, you reuse entire application systems. This function usually involves some kind of configuration of these systems

## Software Construction

Configuration Management

- **Activities**

1. **Version management.** Where support is provided to keep track of the different versions of software components
2. **System integration.** Where support is provided to help developers define what versions of components are used to create each version of a system.
3. **Problem tracking.** Where 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.
4. **Release management.** Where new versions of a software system are released to customers. It's concerned with planning the functionality of new releases and organizing the software for distribution.

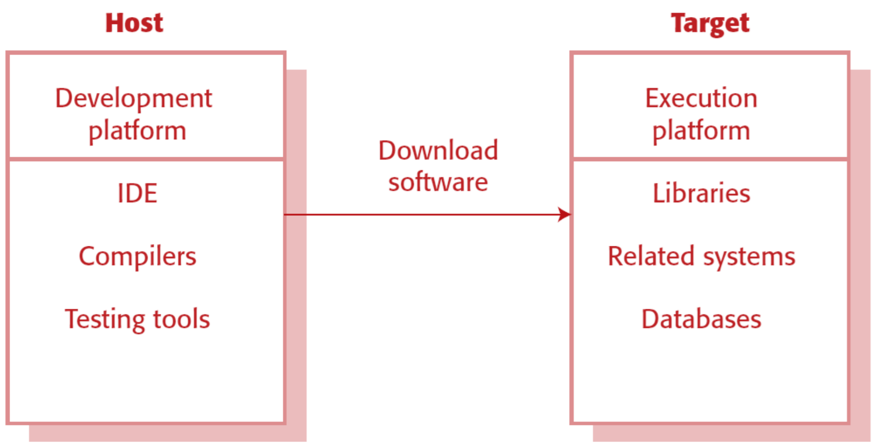


## Software Construction

Host-Target Development

- **Software Development Platform**

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, such as JUnit, that can automatically run a set of tests on a new version of a program
5. Tools to support refactoring and program visualization
6. Configuration management tools to manage source code versions and to integrate and build systems



## Software Construction

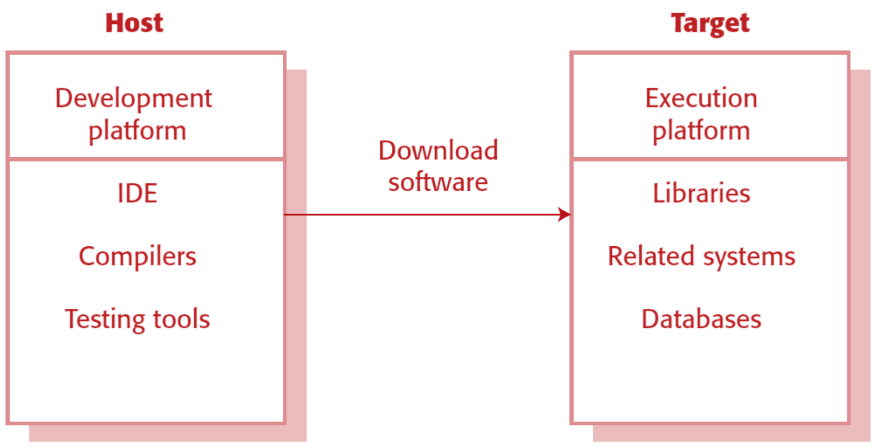
Host-Target Development

- **Target Platform**

1. **The hardware and software requirements of a component**
   - 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.

2. **The availability requirements of the system**
   - High-availability systems may require components to be deployed on more than one platform.

3. **Component communications**
   - If there is a lot of intercomponent communication, it is usually best to deploy them on the same platform or on platforms that are physically close to one another.



## Software Construction

Open-Source Development

- **Definition**

1. Open-source development is an approach to software development in which the source code of a software **system is published, and volunteers are invited to participate** in the development process

2. Its roots are in the **Free Software Foundation (www.fsf.org)**, which advocates that source code **should not be proprietary** but rather should always **be available for users to examine and modify** as they wish

3. There was an assumption that the code would be controlled and developed by a small core group, rather than users of the code

- **Licensing**
    1. A fundamental principle of open-source development is that source code should be freely available
    2. It  does not mean that anyone can do as they wish with that code
    3. Legally, the developer of the code (either a company or an individual) owns the code
    4. They can place restrictions on how it is used by including legally binding conditions in an open-source software license

## Licensing Variants (3 general models)

1. **GNU General Public License (GPL).** This is a so-called reciprocal license that simplistically means that if you use open-source software that is licensed under the GPL license, then you must make that software open source.

2. **GNU Lesser General Public License (LGPL).** This is a variant of the GPL license where you can write components that link to open-source code without having to publish the source of these components. However, if you change the licensed component, then you must publish this as open source.

3. **Berkley Standard Distribution (BSD) License.** This is a  nonreciprocal license, which means you are not obliged to re-publish any changes or modifications made to open-source code. You can include the code in proprietary systems that are sold. If you use open-source components, you must acknowledge the original creator of the code. **The MIT license is a variant of the BSD license with similar conditions.**

## Software Construction

Open-Source Development

- **Licensing Issues**

  1. If you use open-source software as part of a software product, then **you may be obliged** by the terms of the license **to make your own product open source**

  2. If you are trying to **sell your software, you may wish to keep it secret**. This means that you **may wish to avoid using GPL-licensed opensource** software in its development

  3. If you are building software that **runs on an open-source platform** but that does not reuse open-source components, **then licenses are not a problem**

  4. However, if you **embed open-source software in your software**, you need processes and databases to keep track of **what's been used and their license conditions**