# [Handout for L11P2]
# Less work, more results: reusing existing artifacts

## Platforms, frameworks and libraries

### Frameworks

Reusability is a major theme in software engineering practices. By reusing tried and tested "components", we can enhance the robustness of a new software system, while reducing the manpower and time requirement. Reusable components come in many forms; it can be reusing a piece of code, a subsystem, or the whole software architecture. In this section, we take a look at *software framework*, as an example of reusable design.

In everyday life, we can find a number of *entities that share a general structure*, but specialized/modified in specific ways to suit different purpose. One simple example is of a *formal letter*. For a formal letter, information like sender/recipient address, date, reference number, subject title, etc are placed in specific positions. Following the format, we can create a *template* to facilitate this task easily. By supplying the necessary information, e.g. the address, date etc, one creates a *customized* letter that conforms to the format. Software framework is similar to the letter template in this example.

A software framework is a general skeleton (system architecture with partial implementation) that allows user to supply code to adapt and specialize it to provide specific functionality. In a specific domain, the overall structure and execution flow of software systems can be very similar. A software framework supplies an abstraction of the architecture and behaviors to facilitate software development. For some frameworks, the *default behavior* is already provided with complete implementation, which allows rapid deployment.

Below, we give three examples of popular software frameworks.

**Web application frameworks**
Most web-applications have the following components:

- *Database*: information to be stored/retrieved through web based user interface. The data are commonly stored in a database.
- *Web UI*: Web pages that present information to a user and accept user input.
- *Request processor*: Intermediary between the above two components.

The interaction between the components is also similar in most web based applications. For example, a web page will request for data through the request processor, which retrieves the required information from data source. Upon receiving the requested data, the web page will then present the data as needed. Instead of coding the components and managing the interaction between them for every new web application, software developer can utilize a software framework that supplies the general architecture and concentrate on specializing/adapting to suit the requirement. *Ruby on Rails* is an example of one such software frameworks designed for web based application development. *Drupal* is another web application framework specifically for CMS-type web applications (CMS = Content Management System)

**An automated testing framework**
A typical API testing requires following steps:

- Setup the component to be tested
- Supply input to the component under test
- Capture output and compare with expected answer
- Report and collate statistics.

By extracting the common flow of the testing process, we can have an automated testing framework. Examples of automated testing framework include e.g.  JUnit for Java, and cppUnit for C++.

**An IDE framework**
Eclipse is an IDE framework. Not only it is a ready-to-use IDE for Java, it is a framework for developing IDEs for other languages as well as adding more IDE features.

### Frameworks VS Libraries

Although framework and library are both designed with the purpose of reusability, they are not the same. Library is a collection of modular code that is general and can be used in many independent programs.

Here are some characteristics of frameworks that differentiate them from libraries:

- We use libraries 'as is'. In contrast, most frameworks are highly configurable to suit your needs. (using configuration files, or as part of the installation). E.g. Drupal can be configured during installation or by editing configuration files.  Furthermore, frameworks are meant to be extended by writing plugins, sub-classing, etc.You can write plugins to Eclipse, so that it can be used to edit different languages (C++, PHP, etc.).  You can add modules, and themes to Drupal. You need to add test cases to JUnit .
- Frameworks use a technique called *inversion of control*, also called the "Hollywood principle" (i.e. don't call us, we'll call you!). You write code that will be called by the framework. E.g. your test methods will be called by the JUnit framework. In contrast, your code calls libraries.

### Platforms

A platform provides a runtime environment for applications. While technically an operating system can be called a platform, in practice a platform is bundled with various libraries, tools, and technologies in addition to a runtime environment. For example, Windows PC is a platform for desktop applications while iOS is a platform for mobile apps. .NET is considered a framework to write enterprise applications on the Windows platform. Note that 'enterprise applications' means software applications used at organizations level and therefore has to meet much higher demands (such as in scalability, security, performance, and robustness) than software meant for individual use. However, note that .NET is much more general framework that can be used to write any enterprise application whereas the framework examples given previously apply to much narrower scope.

JavaEE (Java Enterprise Edition, previously called J2EE) is both a framework and a platform for writing Java enterprise applications. JavaEE comes with a set of libraries, tools, technologies, techniques for developing any kind of enterprise applications.

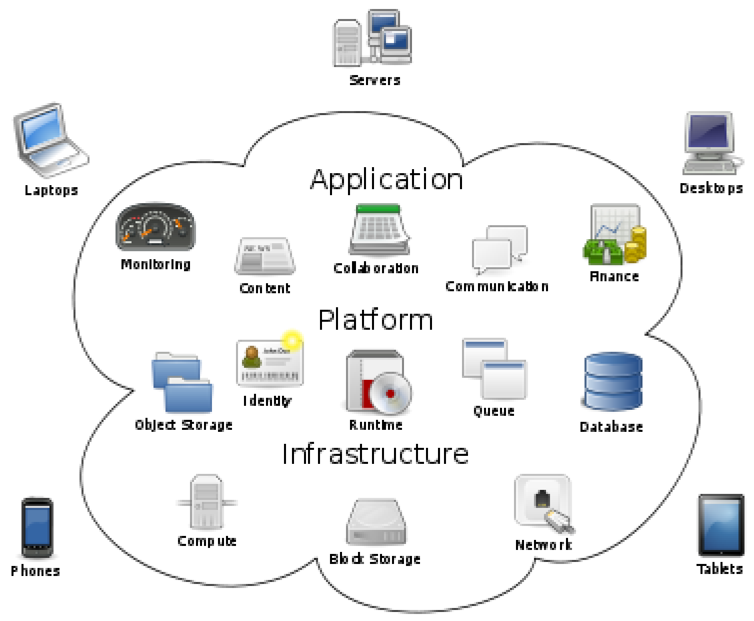
Infrastrcture services such as connection pooling, load balancing, remote code execution, transaction management, authentication, security, messaging etc. are done similarly in most enterprise applications. Both JavaEE and .NET provide these services to applications in a customizable way without developers having to implement them from scratch every time.

For example, JavaEE

- Is built on top of JavaSE (Java Standard Edition).
- Includes technologies like JSP, Servlets, JMS, EJB, RMI, etc.
- Provides configurable infrastructure services. For example, EJB technology lets you automatically persist objects in a database.

## Cloud computing

A topic that is loosely related to the reuse of existing artifacts is *Cloud computing.* Cloud computing is the delivery of computing as a service over the network, rather than a product running on a local machine. This means the actual hardware and software is located at a remote location, typically, at a large server farm, while users access them over the network. Maintenance of the hardware and software is managed by the cloud provider while users typically pay for only the amount of services they use. This model is similar to how we use electricity; the power company manages the power plant while we pay them only for the electricity we use. The cloud computing model optimizes hardware and software utilization and reduces the cost to consumers. Furthermore, users can scale up/down their utilization at will without having to upgrade their hardware and software. The traditional non-cloud model of computing is similar to everyone buying their own generators to create electricity for own use.



As shown in the above diagram (taken from Wikipedia), the cloud contains infrastructure, platform elements, and applications. Accordingly, the cloud can deliver computing services at three different levels:

i. **Infrastructure as a service (IaaS)**: Iaas delivers computer infrastructure as a service. For example, a user can deploy virtual servers on the cloud instead of buying physical hardware and installing server software on them. Another example would be a customer using storage space on the cloud for off-site storage of data. Rackspace is an example IaaS cloud provider. Amazon Elastic Compute Cloud (Amazon EC2) is another.

ii. **Platform as a service (PaaS)**: PaaS provides a platform on which developers can build applications. Developers do not have to worry about infrastructure issues such as deploying servers or load balancing as is required when using IaaS. Those aspects are automatically taken care of by the platform. The price to pay is reduced flexibility; applications written on PaaS are limited to facilities provided by the platform. A PaaS example is the Google App Engine where developers can build applications using Java or Python whereas Amazon EC2 allows users to deploy application written in any language on their virtual servers.

iii. **Software as a service (SaaS)**: This is when applications can be accessed over the network instead of installing them on a local machine. For example, Google Docs is a SaaS word processing software while Microsoft Word is a traditional word processing software.

## Worked examples

**[Q1]**

One of your teammates is proposing to use a recently-released "cool" UI framework for your class project. List pros and cons of this idea.

**[A1]**

Pros

- The potential to create a much better product by reusing the framework
- Learning a new framework is good for the future job prospects

Cons

- Learning curve may be high
- May not be stable (it was recently released)
- May not allow us to do exactly what we want
- Performance penalties
- Might interfere with learning objectives of the module

Note that having more cons does not mean we should not use this framework. Further investigation is required before we can make a final decision.

---End of Document---