Lecture Notes
September 26, 2017

2nd team assignment (project proposal) due tonight

3rd individual/pair assignment (finding bugs) due Thursday

4th individual/pair assignment (development technologies) due Tuesday October 3

3rd team assignment (revised proposal) due Thursday October 5

# Use Cases

User stories and their conditions of satisfaction may not provide enough detail for time estimation (necessary to determine how many user stories will fit in a time-boxed iteration) or to break down into development tasks (that can be assigned to developers) or to inform design - necessary to actually implement!

Use cases *expand* user stories to describe the step by step details of how the user role interacts with the software to exercise the feature and achieve the goal or customer value - much longer and more elaborate than user stories

Use cases also try to capture anything and everything that can *go wrong*

Use case elements (not all present in every use case):

Name - verb/noun or actor/verb/noun communicates scope
(cross-reference to title/label of corresponding user story)

Description - paragraph, might just copy user story

Actors - type(s) of users who engage in activity, could copy role from user story, can also refer to whole organization or external system

Preconditions - anything that must be or is expected to be true before use case begins, specify whether can assume true or need to check (with error if not true)

Triggers - if something happens, such as user selects a menu item or the state changes due to some previous activity, that causes initiation of the use case

Basic flow (happy path, sunny day path) - set of steps actor needs to take to accomplish goal, with clear description of what system does in response to each step

Alternate flows - less common user/system interactions, special cases

Exception flows - error cases, things that can prevent the user from achieving their goal

Postconditions - anything that *must* be true when the use case ends

Events - any new triggers (for other use cases) initiated by the completion of the use case

Others...

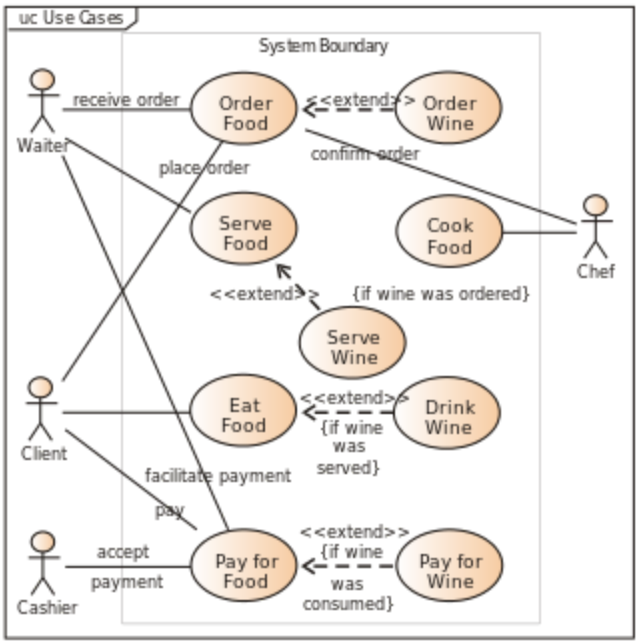
Uses case captures *what* the software needs to do, now *how* the software should do it

The term “use case” is often used more generally to refer to business processes and products that do not necessarily involve software

example use case

We will cover parts of UML (Unified Modeling Language) later on for “class diagrams”, but you do not need to know anything about “use case diagrams” for this course - although future employers may want these

So if you plan to submit anything that looks like this … don’t



Instead submit something that looks like this

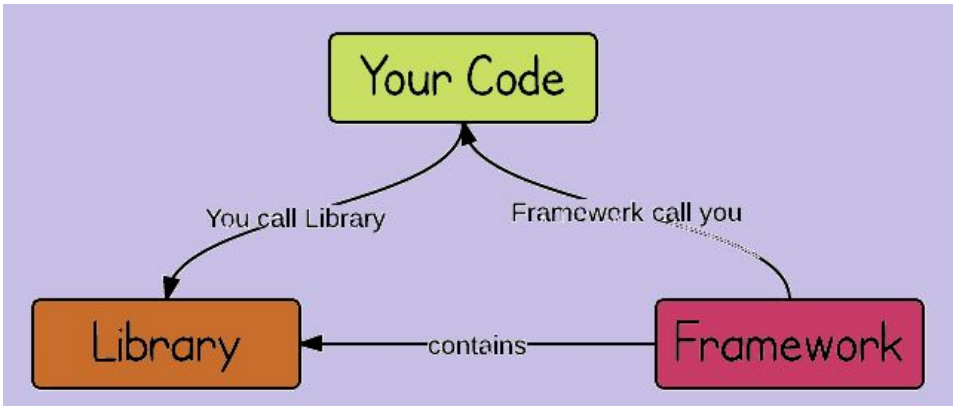
| | |
|---|---|
| **Name** | Save item for purchase. |
| **ID** | UC_001 |
| **Description** | While browsing items in the eStore, a user finds an item he is not ready to purchase yet, but he wants to save it to a list so that he can later find the item that he was previously interested in. |
| **Actors** | eStore customer. |
| **Organizational Benefits** | Increase sales by helping the customer remember products he was previously interested in. |
| **Frequency of Use** | 20% of users save an item to be bought later each time they visit the site. 50% of saved items are purchased within one year of the saved date. |
| **Triggers** | The user selects an option to save an item. |
| **Preconditions** | User is viewing an item in the catalog. |
| **Postconditions** | The item selected to be saved is visible to the user when he views his saved items.<br>The item selected to be saved is reflected as a saved item when the user views his eStore search and browse results. |
| **Main Course** | 1. System prompts user to confirm saving selected item instead of purchasing it right away.<br>2. User confirms to save now (see EX1).<br>3. System determines user is not logged in and redirects user to log on (see AC1).<br>4. User logs on (see AC2, AC3).<br>5. System stores the saved item (see EX2).<br>6. System redirects the user to their saved items list to view the full list. |
| **Alternate Courses** | AC1 System determines user is already logged on.<br>1. Return to Main Course step 5.<br><br>AC2 User logs off again.<br>1. Return user to Main Course step 3.<br><br>AC3 User does not have an account already.<br>1. User creates an account.<br>2. System confirms account creation.<br>3. Return user to Main Course step 4. |
| **Exceptions** | EX1 User decides to purchase the item now.<br>1. See "Purchase item" Use Case.<br><br>EX2 System fails on saving item to list.<br>1. System notifies user that an error has occurred.<br>2. Return user to Main Course step 1. |

# Development Frameworks

Using an application development framework is almost always better, faster, cheaper than developing an application from scratch or combining third-party libraries on your own

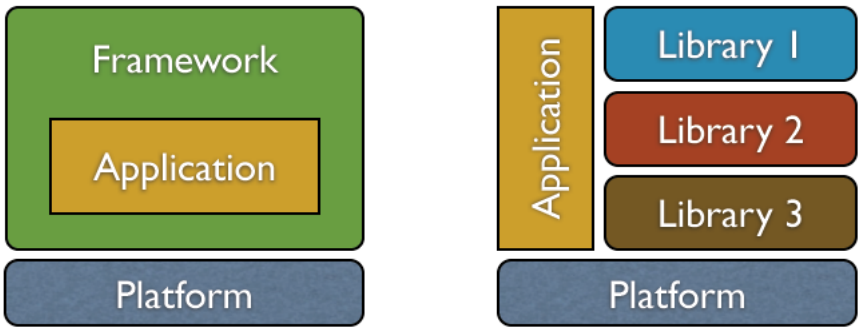
A framework provides set of generic components (some of which may indeed be third-party libraries) that work together to accomplish common development tasks and functionalities for a popular target domain, such as web applications

What is the difference between a *framework* and a *library*?




The conventional definition is based on the concept of "*Inversion of control*"
(Hollywood Principle = don't call us, we'll call you)

- When your code calls a library, your code is in control
- In contrast, the framework calls your code, and the framework is in control, the framework plays the role of the main program in coordinating and sequencing application activity



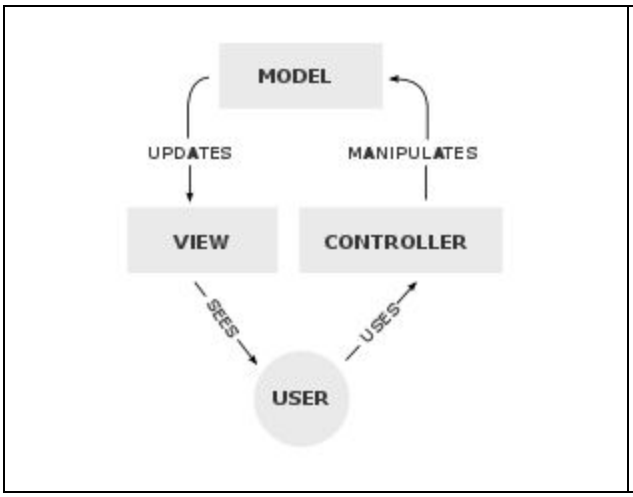
Not all collections of code referred to as frameworks follow this definition, and there are also conceptual frameworks with no specific code (e.g., agile is a conceptual framework for a wide range of real-world software processes)

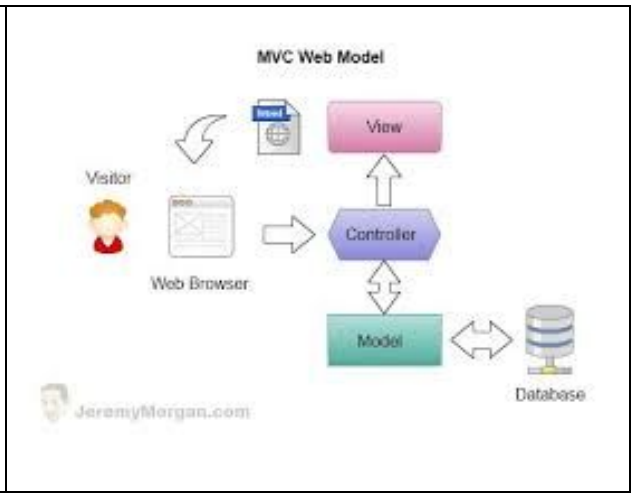
What are the common facilities of web application frameworks? (Not every web app framework will do/have all of these.)

Standard interface between applications written using the framework and conventional web servers and/or browsers (some frameworks primarily/only frontend or backend)

Usually imposes *Model View Controller* (MVC) structure - covered later on

- There are several variants of MVC, here's two:




URL (Uniform Resource Locator) *routing* maps incoming HTTP (HyperText Transfer Protocol) request to resource - the code that will handle the request and receive any query parameters

> URL = <protocol or scheme> : protocol-specific format
>
> URL = http[s]:// host [:port] / path / to / resource [# fragment ] [? query name1=value1 & name2 = value2 & … ]

Request and response objects for information to be received from or sent to browser, resp.

> In general, the framework receives the client's request and constructs the request object (e.g., HTTPRequest) and provides it to your code, your code constructs the response object (e.g., HTTPResponse) and provides it to the framework to respond to the client

*Template engine* for separating business logic from output HTML or other produced content

```
{% extends "layout.html" %}
{% block body %}
  <ul>
  {% for user in users %}
    <li><a href="{{ user.url }}">{{ user.username }}</a></li>
  {% endfor %}
  </ul>
{% endblock %}
```

Interface(s) to persistent store(s), SQL and NoSQL databases

> NoSQL is any persistent store that is not a conventional file system and not a conventional SQL relational database (but might use some SQL notation)
>
> Some “free” databases

Web services via REST, SOAP, and other kinds of APIs - covered later on

> REST = HTTP with GET (simple URL) or POST (form)
>
> SOAP = XML-format RPC (remote procedure call)

Various security-related mechanisms

- encode/sanitize user/network inputs, including form parameters
- Cookies and session tokens
- userid/password management
- Encryption
- …

General application support

- Sockets, network protocols
- Caching
- Process/thread management
- Resource pooling (objects, threads, DB connections)
- Scalability support (load balancing, replication)
- ...

Development web server that updates itself as code changes so can test quickly/easily

May also be closely tied via plugins to IDE and/or build, testing, etc. tools

https://github.com/showcases/web-application-frameworks provides starter web applications for lots of server-side (backend) and “full stack” web application frameworks in various languages

https://github.com/showcases/front-end-javascript-frameworks provides starters for lots of client-side (frontend) Javascript frameworks

You will definitely need a backend framework for this class, you probably do not need a separate frontend framework and you do not need to use Javascript (but if you do use Javascript, probably easiest to choose a full stack framework entirely in Javascript)

AJAX (Asynchronous Javascript and XML) toolkit for dynamic webpages, single page applications - client-side code interacts with server code to update parts of web page in-place without reloading page

Often uses JSON instead of XML (object representation instead of markup)

| | |
|---|---|
| {<br>"Paragraphs": [<br>{<br>"align": "center",<br>"content": [<br>"Here ", {<br>"style" : "bold",<br>"content": [ "is" ]<br>},<br>" some text."<br>]<br>}<br>]<br>} | <Document><br><Paragraph Align="Center"><br>Here <Bold>is</Bold> some text.<br></Paragraph><br></Document> |
| {<br>"firstName": "Homer",<br>"lastName": "Simpson",<br>"relatives": [ "Grandpa", "Marge", "The Boy", "Lisa", "I think that's all of them" ]<br>} | <Person><br><FirstName>Homer</FirstName><br><LastName>Simpsons</LastName><br><Relatives><br><Relative>Grandpa</Relative><br><Relative>Marge</Relative><br><Relative>The Boy</Relative><br><Relative>Lisa</Relative><br><Relative>I think that's all of them</Relative><br></Relatives><br></Person> |