Development Process Management

- A software **development process** or life cycle is a structure imposed on the **development** of a software product.
- There are several models for such **processes**, each describing approaches to a variety of tasks or activities that take place during the **process**.

- More and more software development organizations implement process methodologies.
- The Capability Maturity Model (CMM) is one of the leading models. Independent assessments can be used to grade organizations on how well they create software according to how they define and execute their processes.
- There are dozens of others, with other popular ones being ISO 9000, ISO 15504, and Six Sigma.

Process Activities/Steps

Software Engineering processes are composed of many activities, notably the following:

- **Requirements Analysis**
  - Extracting the requirements of a desired software product is the first task in creating it. While customers probably believe they know what the software is to do, it may require skill and experience in software engineering to recognize incomplete, ambiguous or contradictory requirements.
- **Specification**
  - Specification is the task of precisely describing the software to be written, in a mathematically rigorous way. In practice, most successful specifications are written to understand and fine-tune applications that were already well-developed, although safety-critical software systems are often carefully specified prior to application development. Specifications are most important for external interfaces that must remain stable.
- **Software architecture**
  - The architecture of a software system refers to an abstract representation of that system. Architecture is concerned with making sure the software system will meet the requirements of the product, as well as ensuring that future requirements can be addressed.
- **Implementation**
  - Reducing a design to code may be the most obvious part of the software engineering job, but it is not necessarily the largest portion.

- **Testing**
  - Testing of parts of software, especially where code by two different engineers must work together, falls to the software engineer.
- **Documentation**
  - An important task is documenting the internal design of software for the purpose of future maintenance and enhancement.
- **Training and Support**
  - A large percentage of software projects fail because the developers fail to realize that it doesn't matter how much time and planning a development team puts into creating software if nobody in an organization ends up using it. People are occasionally resistant to change and avoid venturing into an unfamiliar area, so as a part of the deployment phase, its very important to have training classes for the most enthusiastic software users (build excitement and confidence), shifting the training towards the neutral users intermixed with the avid supporters, and finally incorporate the rest of the organization into adopting the new software. Users will have lots of questions and software problems which leads to the next phase of software.
- **Maintenance**
  - Maintaining and enhancing software to cope with newly discovered problems or new requirements can take far more time than the initial development of the software. Not only may it be necessary to add code that does not fit the original design but just determining how software works at some point after it is completed may require significant effort by a software engineer. About 60% of all software engineering work is maintenance, but this statistic can be misleading. A small part of that is fixing bugs. Most maintenance is extending systems to do new things, which in many ways can be considered new work.
-

Process Models

- A decades-long goal has been to find repeatable, predictable processes or methodologies that improve productivity and quality.

Waterfall processes

- The best-known and oldest process is the waterfall model, where developers follow these steps in order. They state requirements, analyze them, design a solution approach, architect a software framework for that solution, develop code, test, deploy, and maintain. After each step is finished, the process proceeds to the next step.

Iterative processes

- Iterative development prescribes the construction of initially small but ever larger portions of a software project to help all those involved to uncover important issues early before problems or faulty assumptions can lead to disaster. Iterative processes are preferred by commercial developers because it allows a potential of reaching the design goals of a customer who does not know how to define what he wants.

Source code Management

- **Source Code Control System** (SCCS) is a version **control system** designed to track changes in **source code** and other text files during the development of a piece of software. This allows the user to retrieve any of the previous versions of the original **source code** and the changes which are stored.
- In the beginning, there was the spreadsheet, the white board and the release engineer. The release engineer ran from one cubicle to the next, trying to keep track of which developer was working on what module and when, as well as which bugs had been fixed, discovered and introduced. Needless to say, that process was fraught with problems and errors. And so, source control management systems were created.

- Given a **version number** MAJOR.MINOR.PATCH, increment the: MAJOR **version** when you make incompatible API changes, MINOR **version** when you add functionality in a backwards-compatible manner, and. PATCH **version** when you make backwards-compatible bug fixes.

- **Version control** is a system that records changes to a file or set of files over time so that you can recall specific versions later. For the examples in this book, you will use software source code as the files being **version controlled**, though in reality you can do this with nearly any type of file on a computer.

- A component of software configuration **management**, version control, also known as revision control or **source** control, is the **management** of changes to documents, computer programs, large web sites, and other collections of information.

- Tools to be used ,SVN,Git etc

- Source Control Management (SCM) is all about the way software changes are made. It has a number of goals which are fundamentally geared toward ensuring that development teams can deliver higher quality code changes at faster speeds. By improving tracking, visibility, collaboration and control throughout the release lifecycle, SCM tools provide more creativity, freedom and possibilities for developers when undertaking complex and challenging work. Moreover, SCM can protect the original source files from any kind of mishap and enables all team members to look at who has made what changes at what point.

Continuous Integration

- **Continuous Integration** (**CI**) involves producing a clean build of the system several times per day, usually with a tool like Cruise Control, which uses Ant and various source-control systems. **Agile** teams typically configure **CI** to include automated compilation, unit test execution, and source control **integration**.

- **Continuous integration** (**CI**) is a software engineering practice in which isolated changes are immediately tested and reported on when they are added to a larger code base. ... **Continuous integration** software **tools** can be used to automate the testing and build a document trail.

- Jenkins is an open-source **CI** tool written in **Java**. It originated as the fork of Hudson when the Oracle bought the Sun Microsystems. Jenkins is a cross-platform **CI** tool and it offers configuration both through GUI interface and console commands.

- **Continuous integration** is a **DevOps** software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. ... With **continuous integration**, developers frequently commit to a shared repository using a version control system such as Git.

Software Testing

- ***Software Testing*** is the process of identifying the ***correctness and quality*** of software program.
- The purpose is to check whether the software satisfies the specific requirements, needs and expectations of the customer. In other words, testing is executing a system or application in order to find software ***bugs, defects or errors***.

Ways of Software Testing

- ***Manual Testing:*** *Test Cases executed manually.*
- ***Automation Testing:*** *Testing performed with the help of automation tools.*

Who does Testing?

- Software Tester
- Software Developer
- Project Lead/Manager
- End User

When to Start Testing?

- During the requirement gathering phase, the analysis and verification of requirements are also considered as testing.
- Reviewing the design in the design phase with the intent to improve the design is also considered as testing.
- Testing performed by a developer on completion of the code is also categorized as testing.

ISO/IEC 9126 standards

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

Software Testing Tools

- HP Quick Test Professional
- Selenium
- IBM Rational Functional Tester
- SilkTest
- TestComplete
- Testing Anywhere
- WinRunner
- LoadRunner
- Visual Studio Test Professional
- WATIR

Testing Method

Black-Box Testing

- The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code.

White-Box Testing

- White-box testing is the detailed investigation of internal logic and structure of the code. White-box testing is also called **glass testing** or **open-box testing**. In order to perform **white-box** testing on an application, a tester needs to know the internal workings of the code.

Grey-Box Testing

- Grey-box testing is a technique to test the application with having a limited knowledge of the internal workings of an application. In software testing, the phrase the more you know, the better carries a lot of weight while testing an application.

Level of Testing

Functional Testing

- This is a type of black-box testing that is based on the specifications of the software that is to be tested. The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for.
- **Non-functional testing** involves testing a software from the requirements which are nonfunctional in nature but important such as performance, security, user interface, Load balancing between servers

etc.

Software Documentation (UML Diagram and Tools)

- UML is a way of visualizing a software program using a collection of diagrams.
- The notation has evolved from the work of Grady Booch, James Rumbaugh, Ivar Jacobson, and the Rational Software Corporation to be used for object-oriented design, but it has since been extended to cover a wider variety of software engineering projects.
- Today, UML is accepted by the Object Management Group (OMG) as the standard for modeling software development

What is UML?

- UML stands for Unified Modeling Language. UML 2.0 helped extend the original UML specification to cover a wider portion of software development efforts including agile practices.

- Improved integration between structural models like **class diagrams and behavior models** like activity diagrams.

- Added the ability to define a hierarchy and **decompose a software system into components and sub-components.**

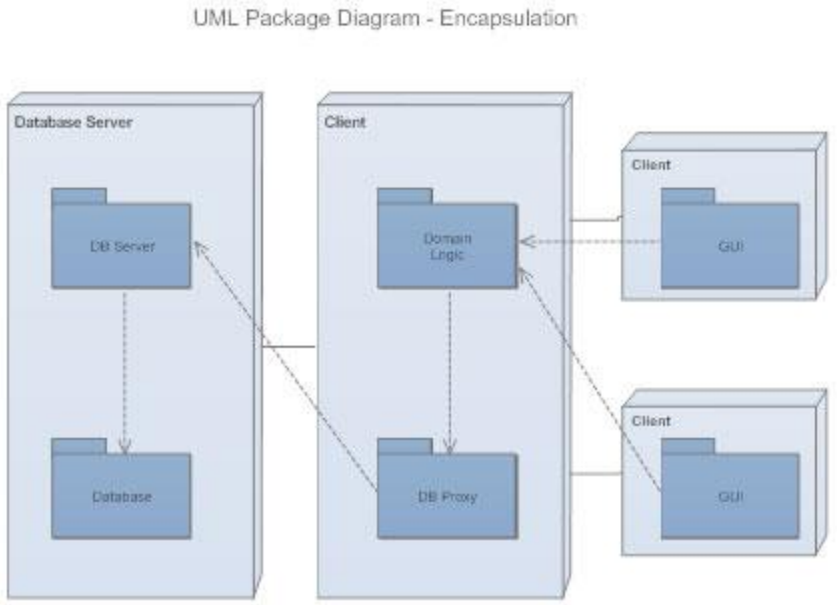
Structural UML diagrams

- Class diagram
- Package diagram
- Object diagram
- Component diagram
- Composite structure diagram
- Deployment diagram

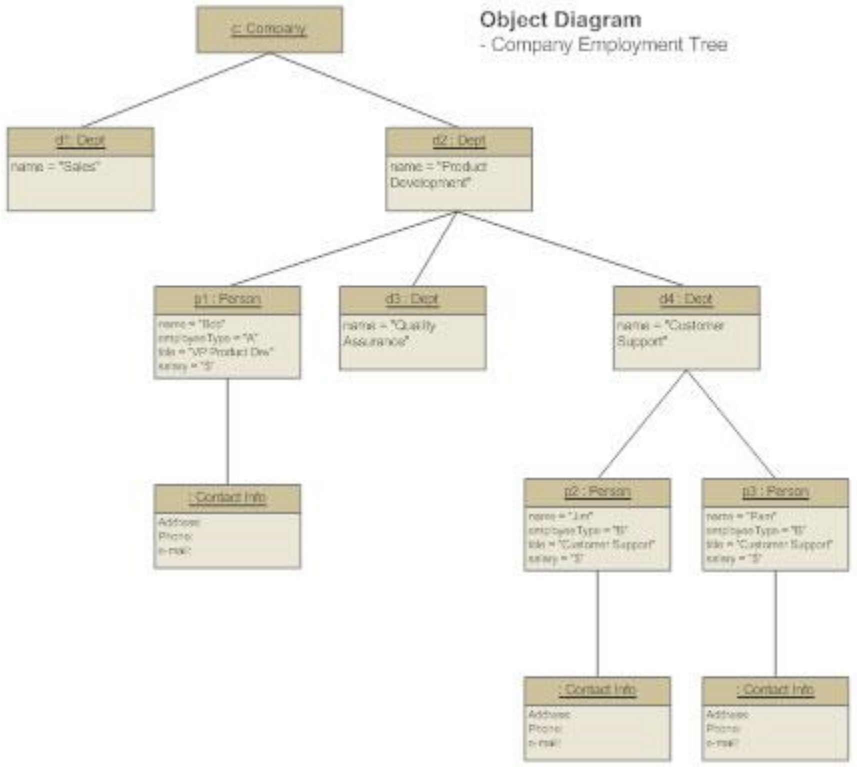
Behavioral UML diagrams

- Activity diagram
- Sequence diagram
- Use case diagram
- State diagram
- Communication diagram
- Interaction overview diagram
- Timing diagram

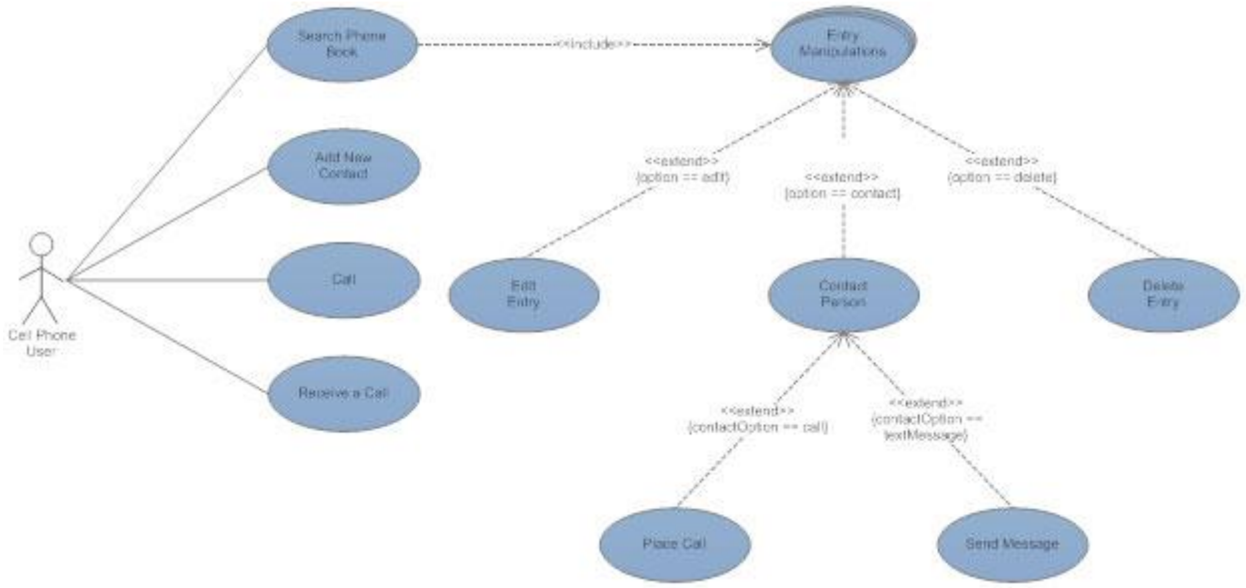
- **Class Diagram**
  Class diagrams are the backbone of almost every object-oriented method, including UML. They describe the static structure of a system.
- **Package Diagram**
  Package diagrams are a subset of class diagrams, but developers sometimes treat them as a separate technique. Package diagrams organize elements of a system into related groups to minimize dependencies between packages.
-



- **Object Diagram**
  Object diagrams describe the static structure of a system at a particular time. They can be used to test class diagrams for accuracy.

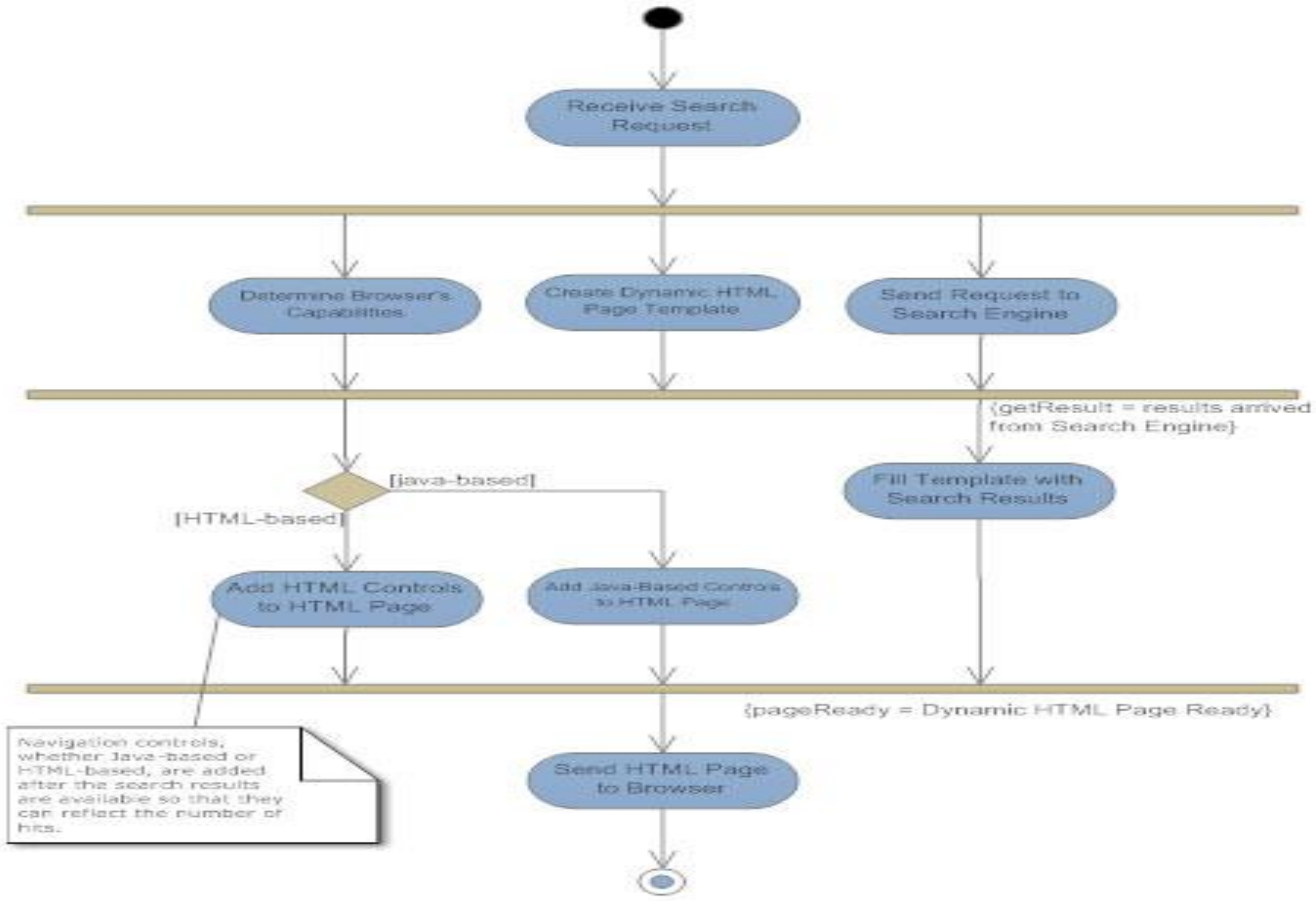


- **Use Case Diagram**
  Use case diagrams model the functionality of a system using actors and use cases.


Search Phone Book
<<include>>
Entry Manipulations
Add New Contact
Call
Receive a Call
Cell Phone User
<<extend>> {option == edit}
<<extend>> {option == contact}
<<extend>> {option == delete}
Edit Entry
Contact Person
Delete Entry
<<extend>> {contactOption == call}
<<extend>> {contactOption == textMessage}
Place Call
Send Message

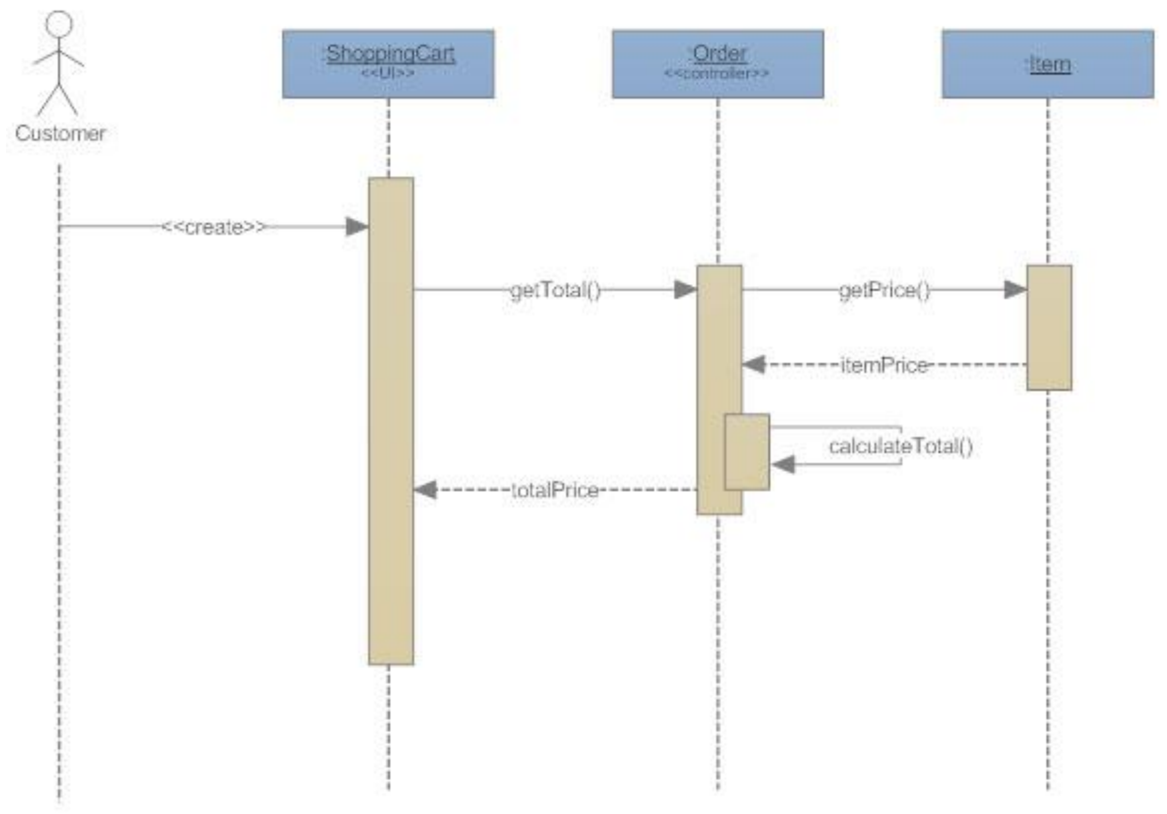
- Activity diagrams illustrate the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation.

UML Activity Diagram: Web Site

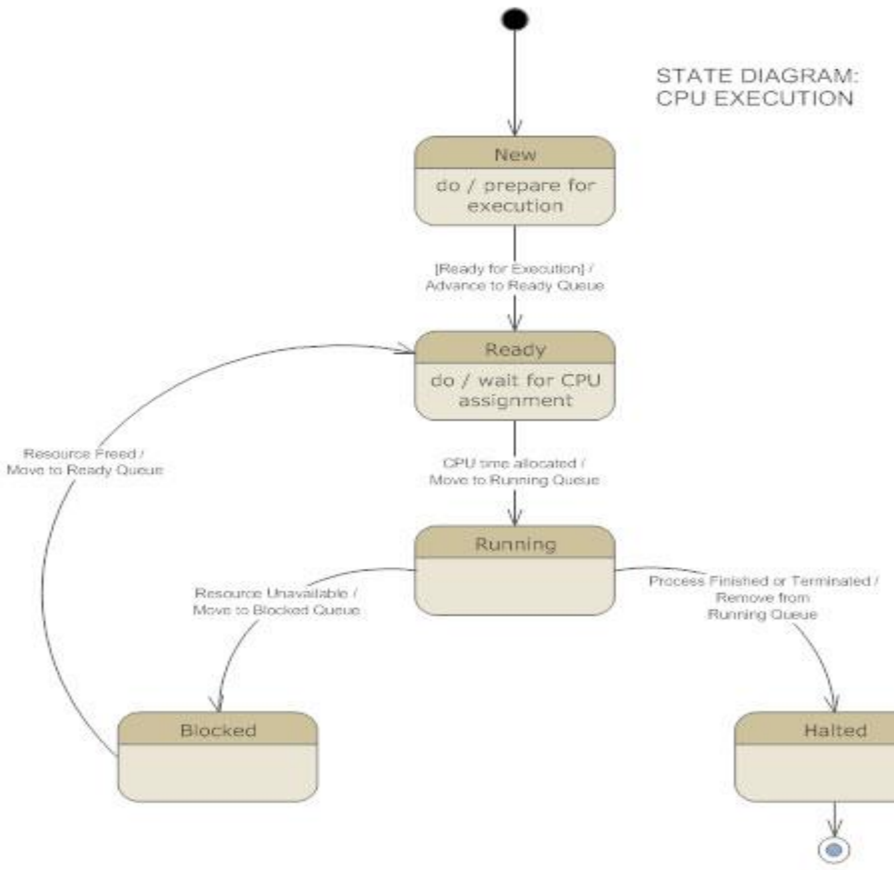

Receive Search Request
Determine Browser's Capabilities
Create Dynamic HTML Page Template
Send Request to Search Engine
{getResult = results arrived from Search Engine}
[java-based]
[HTML-based]
Fill Template with Search Results
Add HTML Controls to HTML Page
Add Java-Based Controls to HTML Page
{pageReady = Dynamic HTML Page Ready}
Send HTML Page to Browser
Navigation controls, whether Java-based or HTML-based, are added after the search results are available so that they can reflect the number of hits.

- **Sequence Diagram**
  Sequence diagrams describe interactions among classes in terms of an exchange of messages over time.

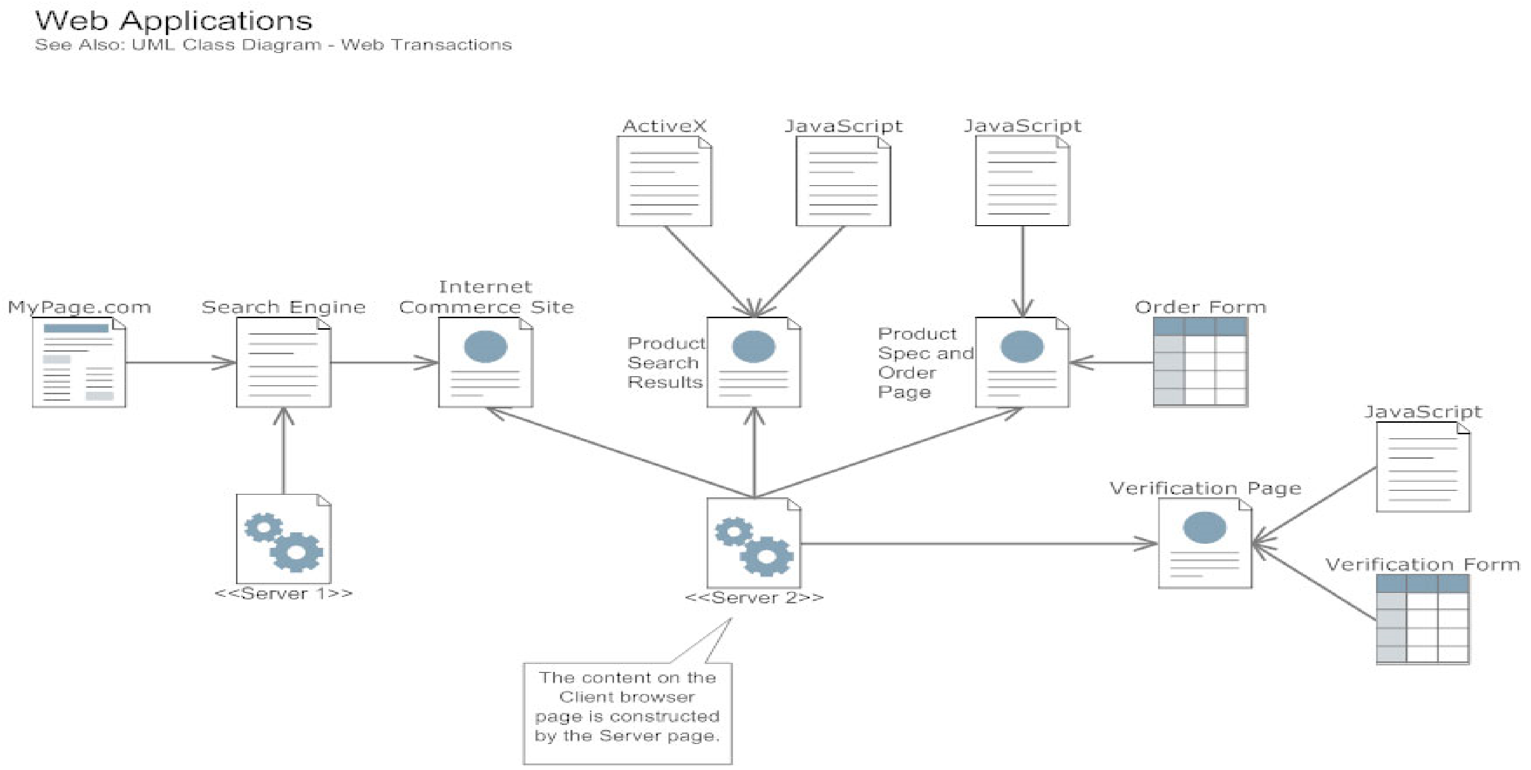
Sequence Diagram: Shopping Cart



- Statechart diagrams, now known as state machine diagrams and state diagrams describe the dynamic behavior of a system in response to external stimuli. State diagrams are especially useful in modeling reactive objects whose states are triggered by specific events


STATE DIAGRAM:
CPU EXECUTION
New
do / prepare for execution
[Ready for Execution] / Advance to Ready Queue
Ready
do / wait for CPU assignment
CPU time allocated / Move to Running Queue
Running
Resource Freed / Move to Ready Queue
Resource Unavailable / Move to Blocked Queue
Process Finished or Terminated / Remove from Running Queue
Blocked
Halted

- Component diagrams describe the organization of physical software components, including source code, run-time (binary) code, and executables.



Why Do We Use UML?

- A complex enterprise application with many collaborators will require a solid foundation of planning and clear, concise communication among team members as the project progresses.
- Tools can be use – UMLet,smartdraw **draw**.io, Lucidchart etc.