Software Project

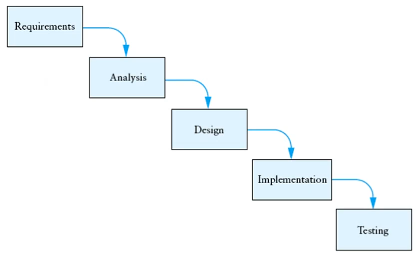
Başlangıçta tüm projenin gereksinimleri tespit etmek mümkün olmaz.

Olabildiğince bütün gereksinimleri çıkartırsın, bunlara göre sistemi dizayn edersin, kodlamaya başlandığında kendini gelişimlere kapatmazsın.

The Software Life Cycle Models

Waterfall Model:

* static
* unworkable
* assumption that each stage can and must be completed before the next one occurs
* user may need to see finished product to express true requirements!



Requirements: The requirements for the software system are determined and documented. İsteyen tarafın sağlayan tarafa yapmasını istediği tüm özellikler requirementstır. System analyst works with users to clarify the detailed system requirements.

Analysis: Requirements çalışılır, aydınlatılır ve çözümün genel yapısına karar verilir.

Design: Methods and data fields are defined for classes. Detailed algorithms for the methods are defined.

Architectural design: The architecture of the solution is determined. This breaks solution

into different components, which are allocated to one or more processing resources.

(which components do I need)

Component design: For each component, classes are identified, with specified roles and

responsibilities. (how components work)

Detailed design: Methods and data fields are defined for classes. Detailed algorithms for

the methods are defined. (input, output, properties of methods)

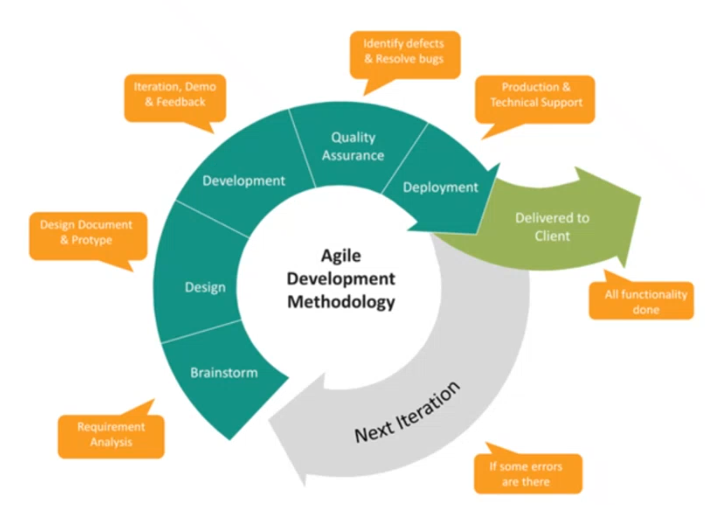
Implementation: The individual classes and methods are coded in the target programming language.

Testing: The methods of each class are tested in isolation and as a class (unit testing). The methods and classes are tested together (integration testing) to verify that they work together and meet the requirements. The product as a whole is tested (acceptance test) against its requirements to demonstrate that the product meets its requirements.

Installation: The product is installed in its end-use (production) environment.

Maintenance: Based upon experience with the software, enhancements and corrections are made to the product.

Agile Model:



* dynamic
* Bütün sistem gereksinimleri belirlenir ve tüm süreçler işlenir. Testlerden sonra talep eden taraf olmuş mu diye sorulur, olmamışsa en baştan aynı süreçler ilerletilir. Bu olana kadar yapılır.

Design

Top-down: break system into smaller subsystems

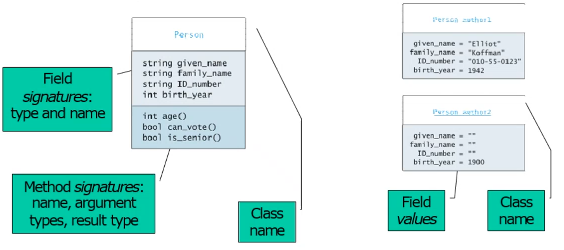
Object-oriented: identify objects and their interactions

UML diagrams: tool to show interactions between classes and external entities

Class diagram:

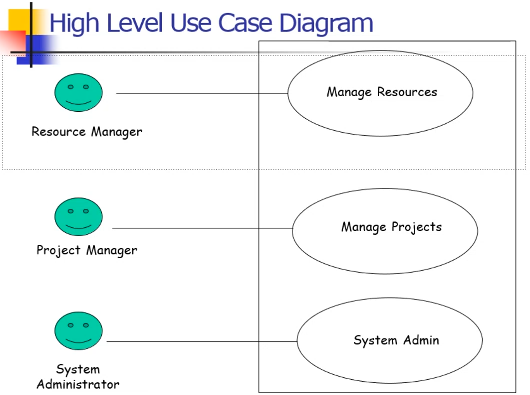
* describe the static structure of a system or how it is structured rather than how it behaves
* contains:
  + Classes
  + Associations which represent relationship that relate 2 or more other classes where the relationships have common characteristics or features. These attributes and operations.

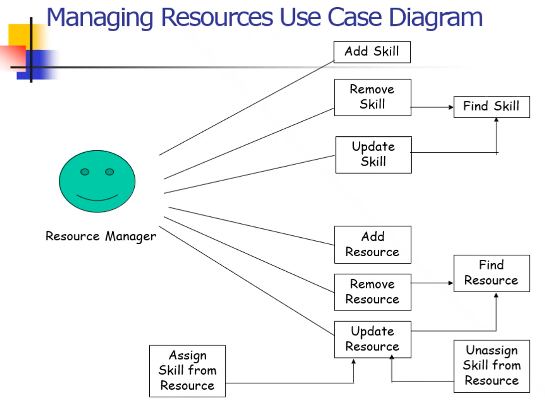
UML:



Use case diagrams:

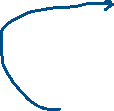
* describe the functionality of a system and users of the system.
* contain:
  + Actors which represent users of a system, including human users and other systems
  + Use cases, which represent functionality or services provided by a system to user



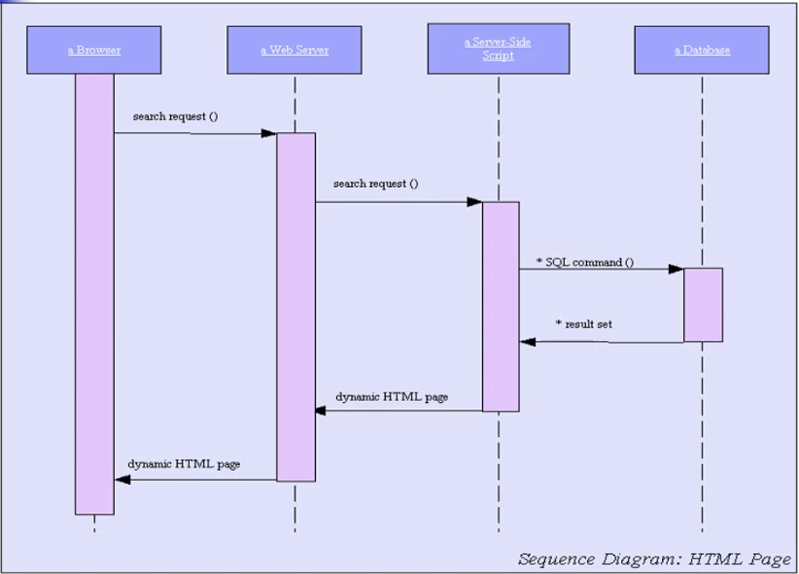


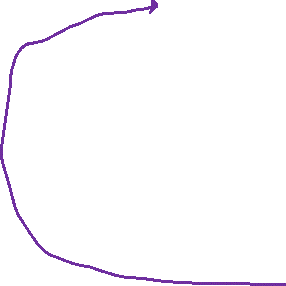
Sequence diagrams:

* focus on classes and the messages they exchange to accomplish some desired behavior
* type of interaction diagrams
* describe interactions among classes
* interactions are modeled as exchange of messages
* contain:
  + Class roles, which represent roles that objects may play within the interaction
  + Lifelines, which represent the existence of an object over a period of time
  + Activations, which represent the time during which an object is performing an operation



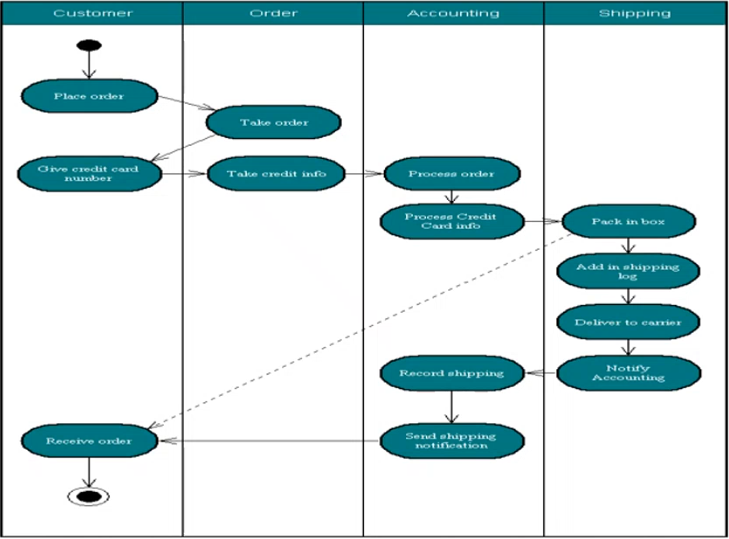
* + Messages, which represent communication between objects





Activity diagrams:

* describe the activities of a class
* contain:
  + Swimlanes, which represent responsibilities of one or more objects for actions within an overall activity; that is, they divide the activity states into groups and assign these groups to objects that must perform activities
  + Action states, which represent atomic, or noninterruptible, actions of entities or steps in the execution of an algorithm
  + Action flows, which represent relationshios between the different action states of an entity





Activities(fiil olmalı!)

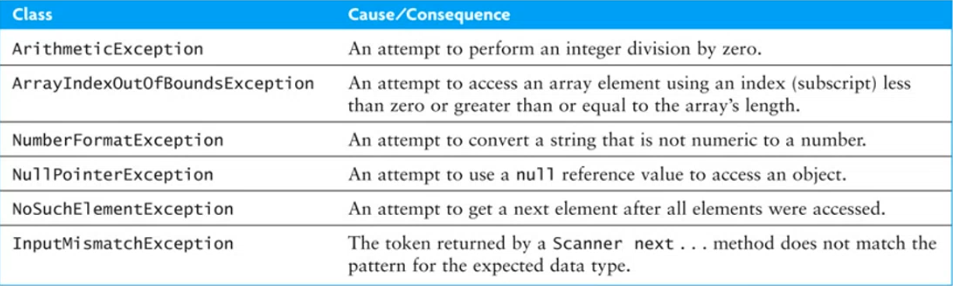
Method calls



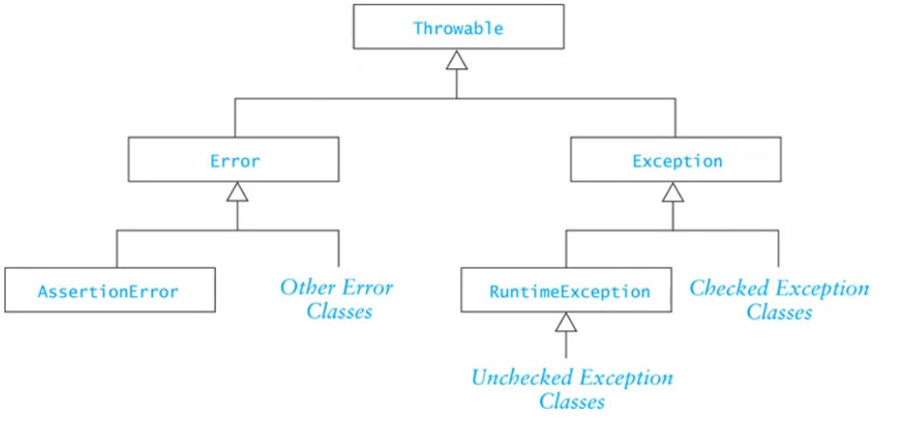
Classes

Major Categories of Defects

1. Syntax and other in-advance errors
   1. Parantez, yazım yanlışı vs.
2. Semantic errors
   1. A class’ında olmayan bir fonksiyonu çağırmak, variable tanımlamadan kullanma, multiple declarations of a variable, performing incorrect operation on a primitive type value, failure to include a library header vs.
3. Run time errors and exceptions
   1. 0’a bölme vs. Genelde yanlış inputlardan kaynaklanır
   2. occur when the JVM detects an operation that it knows to be incorrect
   3. cause the JVM to throw an exception



1. Logic errors
   1. Örneğin karakter 20 cm zıplayabiliyor. Sen 30 cm engel koydun. Compile veya run time’da tespit edilemeyecek bir hata. Tahmin etmesi en zor hatalardır.



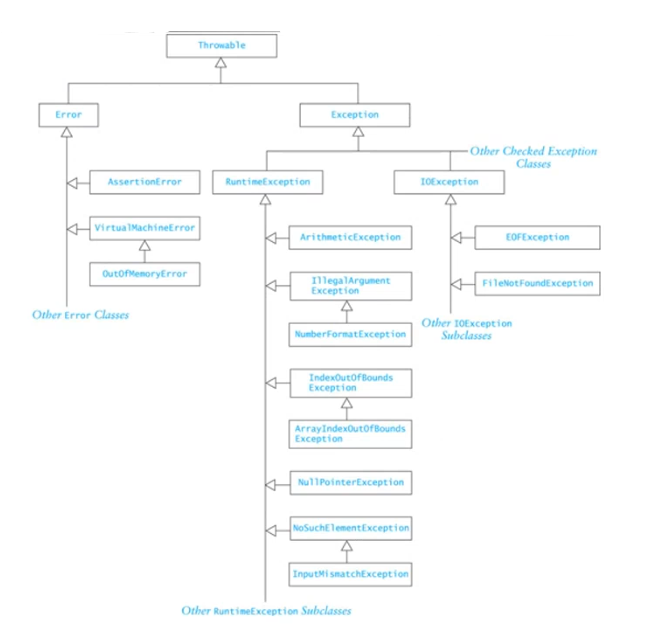
Throwable is the superclass of all exceptions.

Checked exceptions:

* beyond control of programmer
* hazır kütüphane kullanırken doğabilecek general exceptionlardır
* biliyorum ki bu problem birileri tarafından handle edildi
* examples: IOException, FileNotFoundException

Unchecked exceptions:

* geri kalanlar
* examples: NullPointerException, ArrayIndexOutOfBoundsException



Testing Programs

Genelde testing logic hatalar için yapılır.

Must logic errors:

* Come from the design phase
* Result from an incorrect algorithm

For example (logic error):

if (0 >= x && x >= 10) { … }

One way to test: hand-trace algorithm before implementing.

The Designer:

* Explains the algorithm to other team members
* Simulate its execution with them looking on

The Team:

* Verifies that it works
* Verifies that it handles all cases

Unit testing:

* checking the smallest testable piece
* a method or class

Integration testing:

* the interactions among units

System testing:

* testing the program in context

Acceptance testing:

* system testing intended to show that the program meets its functional requirements

Black-box testing:

* Tests item based only on its interfaces and functional requirements
* Assumes no knowledge of internals
* Sistemi bilmiyorum, sadece ne iş yaptığını biliyorum. Çok sayıda inputla test edilir.

White-box testing:

* Tests with knowledge of internal structure

Preparing to test:

* Develop test plan early, in the design phase:
  + How to test the software
  + When to do the tests
  + Who will do the testing
  + What test data to use
* Early test plan allows testing during design & coding

Carefully document:

* Each method parameter
* Each class attruibute (instance and static variable)
* As you write the code!

A driver program:

* Declares necessary instances and variables
* Provides values for method inputs
* Calls the method
* Display values of method outputs

-----------------------------------------------------------

* Object’i üret.
* Metodlarını dene.
* Örneğin karakter isteyen yere numeric bir şey girince olmadığını da göster.

Testing framework:

* Software that facilitates:
  + writing test cases
  + organizing the test cases into test suites
  + running the test suites
  + reporting the results