* The Software Crisis 1968

- Capability of software lags behind that of the hardware

-General difficulty of developing complex software systems

* 1968 NATO Conference: first use of the term “Software Engineering
* The first software crisis led directly to many of the innovations that we take for granted today Structured Programming and High-level languages;
* What is failure (for software projects)?: cancellation, over-budget, final product lacks a requested feature
* What are the reasons for failure?

Incomplete or changing requirements

Lack of user input and involvement

* Code of Ethics
  + PUBLIC act in the public interest
  + CLIENT AND EMPLOYER act in the best interests of the client
  + PRODUCT maintain high quality standards in software products
  + JUDGEMENT integrity in professional judgments
  + MANAGEMENT ethical approach to management of software projects
  + PROFESSION maintain reputation of the profession
  + COLLEAGUES treat your peers and colleagues with respect
  + SELF continual learning over the lifetime in the profession
* Low Representational Gap is the idea that conceptual categories and programming categories can converge
* The Unified Modeling Language (UML) is a modeling language that can help you to document and communicate designs in Software Engineering
* Different types of diagrams to represent different views of the design: Behavior, Structure, Communication
* Class Diagrams in UML: Provides the static relationships between different classes
* Two of the most important kinds of relationships between classes are inheritance or composition (aggregation)
* Inheritance: one class is a specialization or generalization of another class (IS-A) Cat → Animal
* Composition: one class contains or uses another class (HAS-A) Car → Engine
* “Composition over Inheritance” is often considered helpful design advice
* Polymorphism: Any particular instance of a class may be treated as having a different type depending on its subtyping relationships. With polymorphism we can Program to an Interface
* Delegating Responsibilities: One class will typically contain references to other classes for the purpose of achieving more complex behavior. Class delegates its responsibilities to other objects
* Java pass-by-value
* Design Patterns: an idea from architecture that was applied to software development. Design patterns indicate how classes should be structured to solve a particular problem (don’t provide implementation)
* Types of Pattens:
  + Creational: singleton, builder
  + Structural: composite, decorator
  + Behavioral: observer, strategy, state, template method
* The Observer Pattern: Way to implement a publisher/subscriber relationship between objects
* UML Sequence Diagrams allow us to model object interactions ordered in time
* Aim for low coupling and high cohesion; Coupling refers to how much one object has to know about another object to interact. Cohesion refers to how much a class does a single thing
* Strategy Pattern: about being able to reconfigure algorithms at runtime
* The Decorator Pattern: Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.
* Open-Closed Principle: Classes should be open for extension but closed for modification
* Builder Pattern: “Separate the construction of a complex object from its representation so that the same construction process can create different representations.”
* The State Pattern: “Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.”; Class involved: Context, State Interface, Concrete State Classes
* The Template Method Pattern: “Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm’s structure.” Class involved: Abstract class, Concrete Class
* Dependency Injection refers to the way a class receives its dependencies from other parts of the application
* Coupling: Coupling Between Object Classes (CBO), counts number of classes to which one class is coupled (use of member functions or instance variables)
* Cohesion: Lack of Cohesion on Methods (LCOM), measures how much the instance variables of a class are shared, compares number of pairs of member functions with and without shared instance variables
* LCOM: The number of pairs of member functions without shared instance variables, minus the number of pairs of member functions with shared instance variables. However, the metric is set to zero whenever the above subtraction is negative.
* Low Representational Gap is the idea that conceptual categories and programming categories can converge
* the domain model is a “visual representation of conceptual classes or real-situations in a domain”
* Use Case: description of a sequence of interactions between the system and one or more external actor, the actor derives some benefit or achieves some goal by interacting with the system
* Extensions capture the scenarios other than the main success scenario
* Requirements are captured as items in a list of “product backlog”