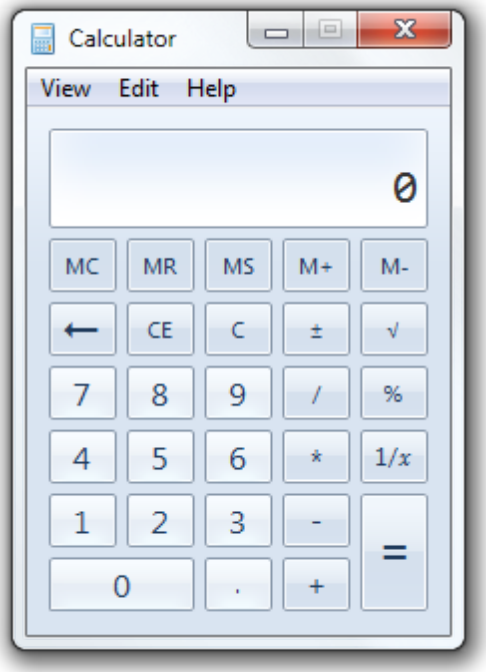
Recap and Objectives

- Last time we looked at Template Method, Dependency Injection, and Cyclomatic complexity[1]
- Today, want to have a look at OO design more generally
  - Review the Booch method[2]
  - Domain Model
  - Software Lifecycle Process

[1] Thomas J McCabe. "A complexity measure". In: *IEEE Transactions on software Engineering* 4 (1976), pp. 308–320.

[2] Grady Booch. "Object-oriented development". In: *IEEE transactions on Software Engineering* 2 (1986), pp. 211–221.

Calculator Example



- A classic windows calculator program (image[3])
- Imagine designing this program from three different points of view: mathematician, computer scientist, and software engineer
- How is your concept of *number* affected by your point of view?

[3] By Microsoft - Own work, MIT, https://commons.wikimedia.org/w/index.php?curid=77199543

Use Software Metrics

- *Software Metrics* or *Code Metrics* measure things about software
  - Number of lines of code
  - Execution time
  - ...
- Metrics exist that quantify OO design principles and have been experimentally validated[4]
- 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

[4] Victor R Basili, Lionel C. Briand, and Walcélio L Melo. "A validation of object-oriented design metrics as quality indicators". In: *IEEE Transactions on software engineering* 22.10 (1996), pp. 751–761.

Cyclomatic Complexity as a Measure of Program Complexity

NAZARBAYEV UNIVERSITY
SCHOOL OF ENGINEERING AND DIGITAL SCIENCES

$$v(G) = e - n + 2$$

- The cyclomatic complexity is a measure of the "size" of the program, like the number of lines of code
- For a *planar* program control graph CC counts the number of distinct regions in the plane
- A program with a higher cyclomatic complexity is expected to require more effort to test

Branch Coverage vs. Path Coverage

- *Code Coverage* is the degree to which a set of tests represents the possible behavior of a program
  - Branch Coverage: each branch of a control structure is executed
  - Path Coverage: every possible execution route has been taken
- Clearly, Branches $\leq$ Paths. We do not expect to achieve path coverage on moderately complex programs
- Cyclomatic Complexity is an upper bound on the number of tests we would need to run for branch coverage

Object-Oriented Metrics

NAZARBAYEV UNIVERSITY
SCHOOL OF ENGINEERING AND DIGITAL SCIENCES

- Standard set of OO Metrics defined by Chidamber and Kemerer[5]
- Mention 3 in particular
  - Depth of Inheritance Tree (DIT)
  - Coupling Between Object Classes (CBO)
  - Lack of Cohesion on Methods (LCOM)
- Experimentally validaded by Basili *et al*[6]

---

[5] Shyam R Chidamber and Chris F Kemerer. "A metrics suite for object oriented design". In: *IEEE Transactions on software engineering* 20.6 (1994), pp. 476–493.

[6] Victor R Basili, Lionel C. Briand, and Walcélio L Melo. "A validation of object-oriented design metrics as quality indicators". In: *IEEE Transactions on software engineering* 22.10 (1996), pp. 751–761.

Definition of DIT[7]

Maximum depth of the inheritance graph for each class

- A well-designed OO system will be structured as a "forest of classes" instead one huge tree
- The deeper a class is in the inheritance hierarchy, the more definitions and behaviors it will receive from its ancestors, the more likely it is to be error prone

[7] Victor R Basili, Lionel C. Briand, and Walcélio L Melo. "A validation of object-oriented design metrics as quality indicators". In: *IEEE Transactions on software engineering* 22.10 (1996), pp. 751–761.

Definition of CBO[8]

A class is coupled to another if it uses its member functions and/or instance variables. CBO provides the number of classes to which a given class is coupled.

- It is expected that highly coupled classes are more error prone than weakly coupled classes
- Lower CBO means classes are more weakly coupled

[8] Victor R Basili, Lionel C. Briand, and Walcélio L Melo. "A validation of object-oriented design metrics as quality indicators". In: *IEEE Transactions on software engineering* 22.10 (1996), pp. 751–761.

Definition of LCOM[9]

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.

- Number of pairs of methods is a binomial coefficient $\binom{N}{2}$ where N is the number of methods
- Higher LCOM $\rightarrow$ less cohesion, class has multiple responsibilities

[9] Victor R Basili, Lionel C. Briand, and Walcélio L Melo. "A validation of object-oriented design metrics as quality indicators". In: *IEEE Transactions on software engineering* 22.10 (1996), pp. 751–761.

LCOM Example

NAZARBAYEV UNIVERSITY
SCHOOL OF ENGINEERING AND DIGITAL SCIENCES

```
public class Person {
  String name;
  String address;
  int age;

  public String getName() { return name.toUpperCase(); }
  public int getAge() { return age; }
  public String getDescription() {
    return getName() + " " + address;
  }
}
```

- `getName()` and `getAge()`: do not access any of the same instance variables
- `getAge()` and `getDescription()`: do not access any of the same instance variables
- `getName()` and `getDescription()`: both depend on `name`
- There are 2 pairs without shared instance variables and 1 pair with shared instance variables, LCOM equals $2 - 1 = \mathbf{1}$
- *Note*: we may choose to exclude getter and setter methods from the definition of LCOM

Why care about software metrics?

- Time available for testing is a finite resource
  - Know where to focus our resources
  - Estimate the amount of testing we need to do
- By being quantitative, metrics can help to establish *evidence* in Software Engineering
  - As in, something stronger than anecdote
- You can experiment with CC (in Javascript) at `jshint.com`

Low Representational Gap

- Object-orientation is about mental models
- Objects are supposed to be "like" the representations human beings use to understand the world
- *Low Representational Gap* is the idea that *conceptual* categories and *programming* categories can converge

Grady Booch, 1986[10]

Abstraction and information hiding are actually quite natural activities. We employ abstraction daily and tend to develop models of reality by identifying the objects and operations that exist at each level of interaction.

[10] Grady Booch. "Object-oriented development". In: *IEEE transactions on Software Engineering* 2 (1986), pp. 211–221.

Booch Method

- Booch[11] presented one of the first systems of OO design
  - He called it a *partial lifecycle* method, because it is only concerned with the design and implementation activities of the software lifecycle
- Booch highlights five major steps
  1. Identify the objects and their attributes
  2. Identify the operations suffered by and required of each object
  3. Establish the visibility of each object in relation to other objects
  4. Establish the interface of each object
  5. Implement each object

[11] Grady Booch. "Object-oriented development". In: *IEEE transactions on Software Engineering* 2 (1986), pp. 211–221.

Domain Model

- "Identify the objects and their attributes" is a crucial part of OO design and results in what is called the *domain model*
- According to Larman[12] the domain model is a "visual representation of conceptual classes or real-situations in a domain"
  - conceptual classes and their relationships
  - Domain: sphere of knowledge, application area
- Booch's design case study involves software for controlling a system of computerized buoys
  - Collect weather data, interact with vessels and sailors

[12] Craig Larman. *Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Interative Development*. Prentice Hall PTR, 2012.

Buoy Example

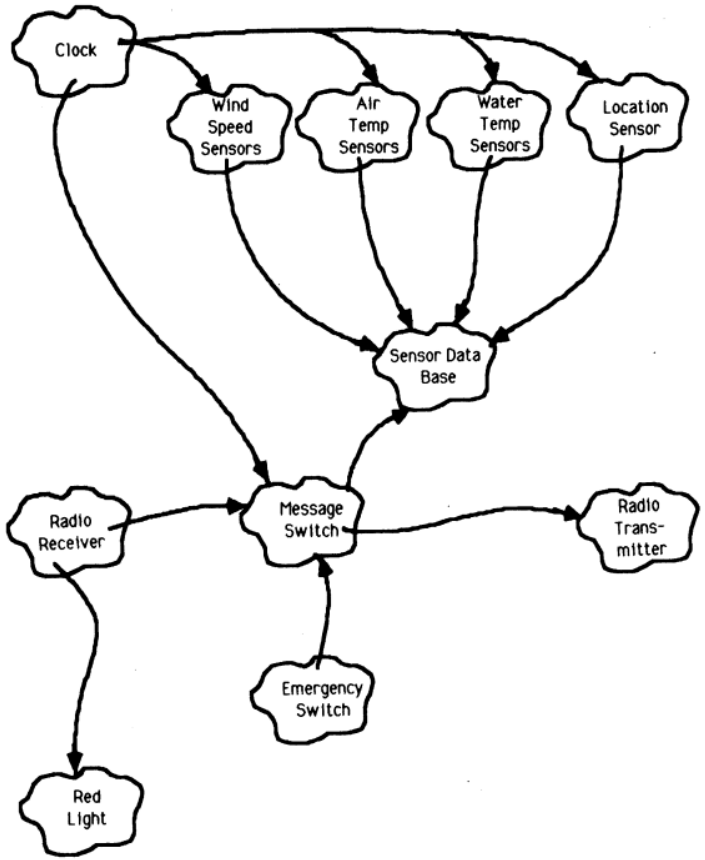
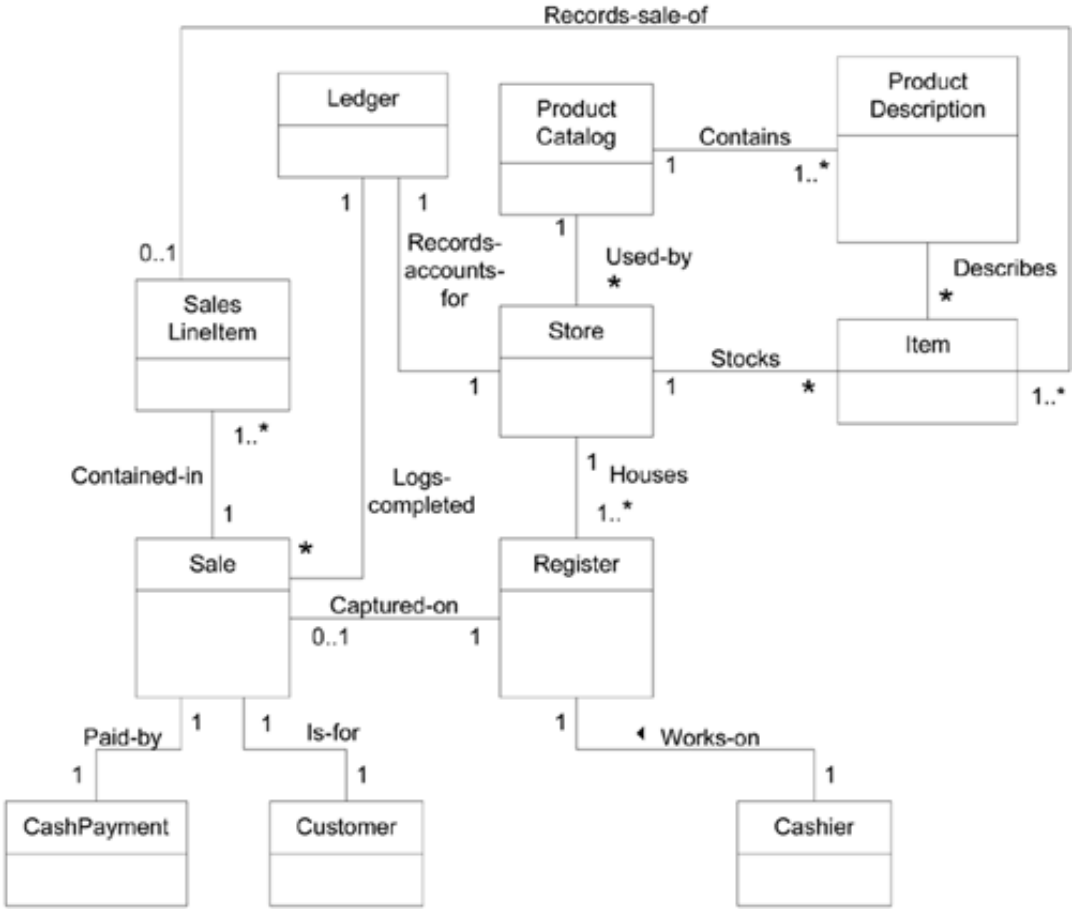


Fig. 9. Host at sea buoy objects.

- This diagram[13] shows the classes in the domain of the case study
- These notations are the precursor to UML

[13] Figure from Booch, "Object-oriented development", op. cit.

Domain Model in UML



- The POS Terminal Domain Model from Ch .9 of the text[14]

[14] Craig Larman. *Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Interative Development*. Prentice Hall PTR, 2012.

Domain Models and Data Models

- A domain model is distinct from a data model
- *Data model*: model of data for persistent storage. For example, in a relational database system, drawn in IE (information engineering) notation
- Domain models deal with *conceptual classes*, do not necessarily need to be persisted
- Conceptual class[15]
  - Symbol
  - Intension
  - Extension

[15] Craig Larman. *Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Interative Development*. Prentice Hall PTR, 2012.

Creating a Domain Model

- Larman[16] recommends a three step process
  1. Find the conceptual classes (same as Booch)
  2. Draw as a UML Class diagram
  3. Fill in the associations and attributes
- How to find the conceptual classes?
  - Identify the nouns
  - Use a list[17] of general categories
  - Adapt existing models (including data models)
- Attributes versus Classes
  - Attributes should usually be primitive data types
  - Conceptual classes are to be related with an association

---

[16] Craig Larman. *Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Interative Development*. Prentice Hall PTR, 2012.

[17] Such a list is provided on p. 140

Identifying Operations

NAZARBAYEV UNIVERSITY
SCHOOL OF ENGINEERING AND DIGITAL SCIENCES

- The second step in the Booch method[18] is to identify operations
  - Operations *suffered by*: data flow into an object
  - Operations *required of*: data flow out of an object
- The directed dependecies in Booch's system generally follow the *required of* data flow

[18] Grady Booch. "Object-oriented development". In: *IEEE transactions on Software Engineering* 2 (1986), pp. 211–221.

Buoy Example[19] (again)

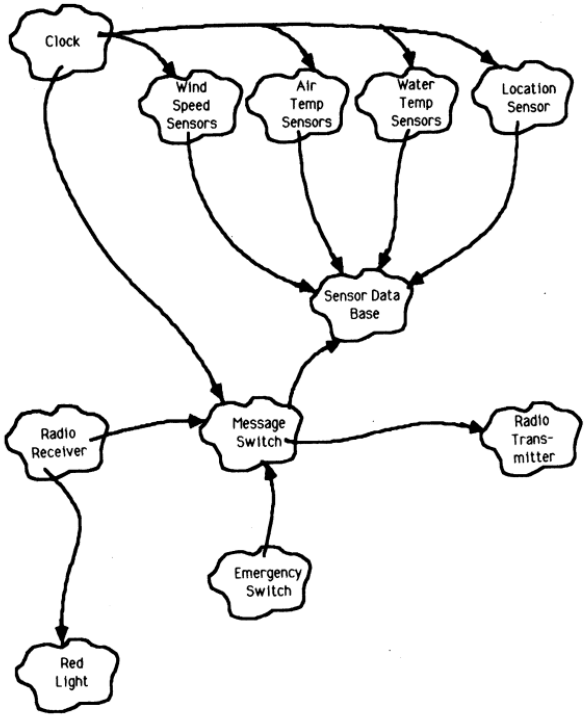


Fig. 9. Host at sea buoy objects.

- *Clock* only requires operations of other objects
- *Red Light* only has in data flows

[19] Grady Booch. "Object-oriented development". In: *IEEE transactions on Software Engineering* 2 (1986), pp. 211–221.

Implementation

- The next two steps of the Booch Method establish visibility and interfaces of objects
  - Implementation: conceptual classes $\rightarrow$ programming classes
- Transition facilitated by low representational gap
- *Low Representational Gap* is the idea that *conceptual* categories and *programming* categories can converge
- A **Domain Layer** consists of business objects and business logic
  - Not necessarily one-to-one correspondence with the domain model (but should be close and share names)

SOLID Principles

- Martin[20] came up with what are known as the SOLID princples (SOLID is an acronym)
- **Single-Responsibility Principle**: "a class should have only one reason to change"
- **Open/Closed Principle**: *decorator* and *template method*
- **Liskov Substitution Principle**: "subtypes must be substitutable for their base types"
- **Interface segregation principle**: "Clients should not be forced to depend on methods they do not use"
- **Dependency-Inversion Principle**: "High level modules should not depend on low-level modules."

[20] Robert C. Martin. *Agile Principles, Patterns, and Practices in C#*. Prentice Hall, 2006.

Reading

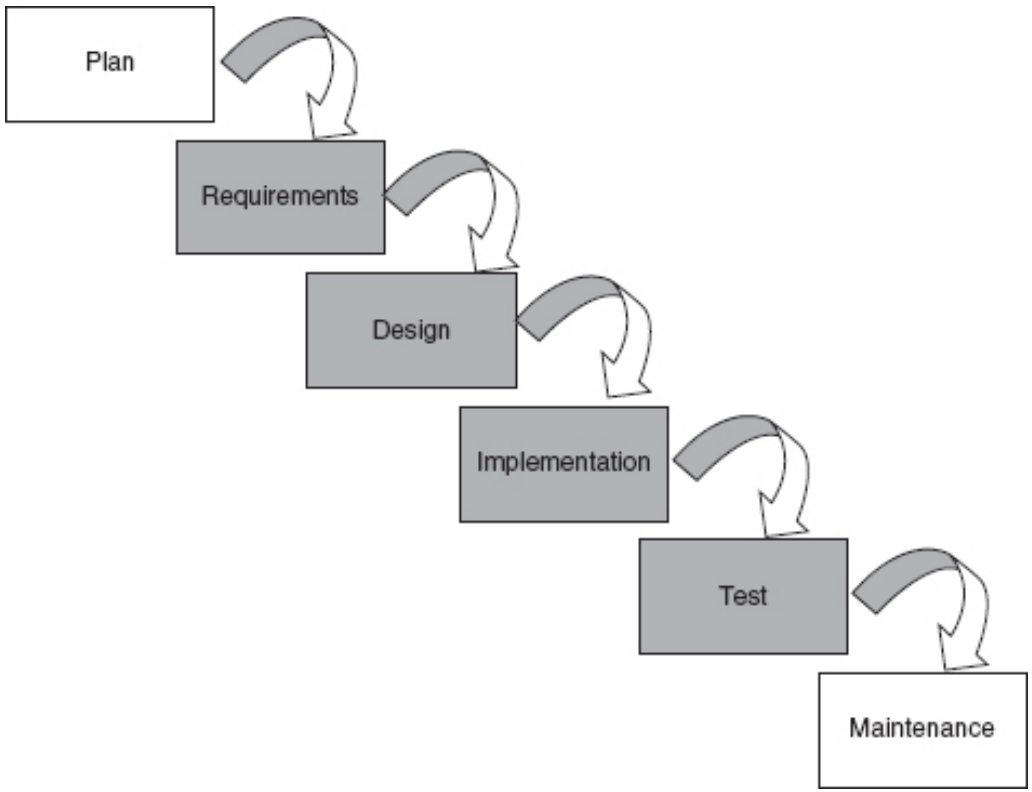
- Please review Chapters 9 (Domain Models), 17 (GRASP), and 18 (Design Examples with GRASP) from the text[21]

[21] Craig Larman. *Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Interative Development*. Prentice Hall PTR, 2012.

Software Lifecycle Process Model

- Booch called his method a "Partial Lifecycle" model
- What would be a "full" lifecycle model
- The process model is an abstract representation of the tasks required to achieve the goals of the project
- The SPM defines
  - Which tasks are to be performed
  - Input and output of each task
  - How the tasks should sequence and flow

Waterfall Model



- One of the earliest published *software processes*[22]
- Development occurs in distinct phases, with the output of one activity serving as the basis for the next

[22] Winston W Royce. "Managing the development of large software systems". In: *proceedings of IEEE WESCON*. vol. 26. 8. Los Angeles. 1970, pp. 328–338.

Waterfall Discussion

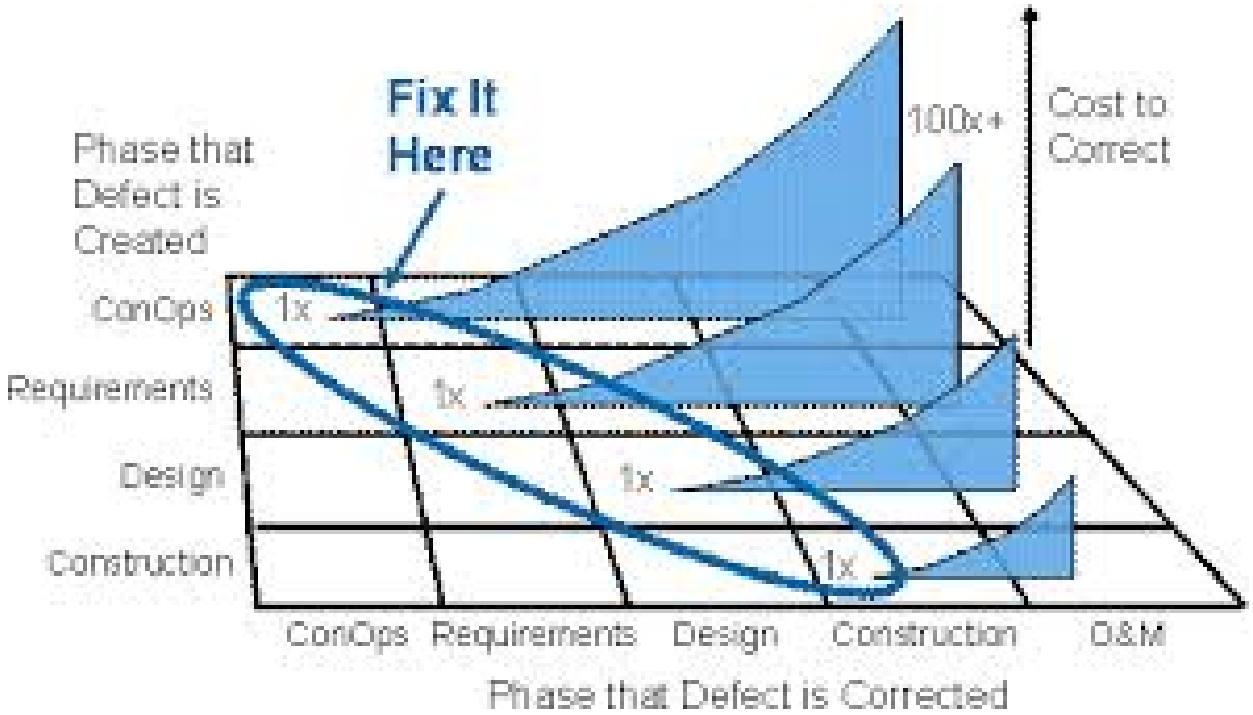
- Waterfall is also called the *document-driven* approach due to the many documents that must be produced while executing it
- Activities occur in a strict linear sequence
- Early electronic computers could occupy an entire room
  - Even simple programs required extreme amounts of labor
  - Not conducive to iterating, making changes
- "Big Design Up Front" process
- Different specialists work in different "silos"
- Formal plan and timeline can be disrupted if one phase runs beyond schedule

Waterfall Thinking

- What do people really mean by *waterfall*?
- It is a kind of rigid thinking that says requirements, scope, design, can all be known in advance.
- Evidence shows that "...poor practice for most software projects, rather than a skillful approach...associated with high rates of failure"[23]

[23] Craig Larman. *Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Interative Development*. Prentice Hall PTR, 2012.

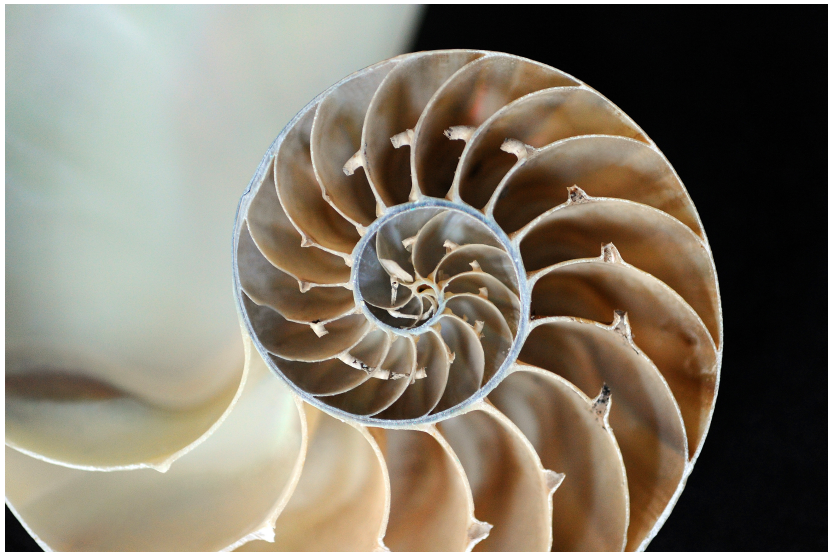
Cost of Fixing a Bug



- The total cost to correct a bug (time spent, monetary, lost business, etc.) increases the later the bug is detected (image credit[24])
- A process that delivers working software early can help to mitigate the effects of *defect cost increase*

[24] `https://ops.fhwa.dot.gov/publications/seitsguide/section3.htm`

Iterative and Incremental Development[25]



- Two important notions in software development are *increments* and *iterations*
- *Iteration*: Is one pass through the waterfall process
- *Incremental*: Delivering additional features in small units
- *Evolutionary Model*: Iterative model where the software gradually evolves over time, allows for changes to requirements and design based on customer feedback

[25] C. Larman and V. R. Basili. "Iterative and incremental developments. a brief history". In: *Computer* 36.6 (2003), pp. 47–56.

Process Iteration

- Changing requirements are a basic fact of developing systems
- *Incremental Delivery*: software delivered in increments, each new increment adds a small piece of the required functionality
- Most important requirements are delivered in early increments
- Requirements are frozen once work has started on a given increment, this is called a *timebox*
  - Typical timebox might be 2-3 weeks
- A fundamental principle of *agile* development is that working software should be delivered as early as possible