Curriculum

- Software development lifecycle
- Capture software requirements using UML
- Strike a balance: risk management
- Early bug-finding using model checking
- Maintain traceability in model-based software design
- Software testing

“In a software engineering course, you PREACH, not TEACH.”

-- Frederick P. Brooks, Jr.
UNC Chapel Hill

Why preach instead of teach

- What can be taught?
  - Tools and methodologies
  - Which are different in different industries, and change over time

- Religions are ways of interpreting the world
  - Preaching principles which can change your behavior
  - i.e. You will go to Hell if you don’t donate 1/10 of your wealth (Tithe)
  - Which do not change over time

Key Challenges in Software Engineering

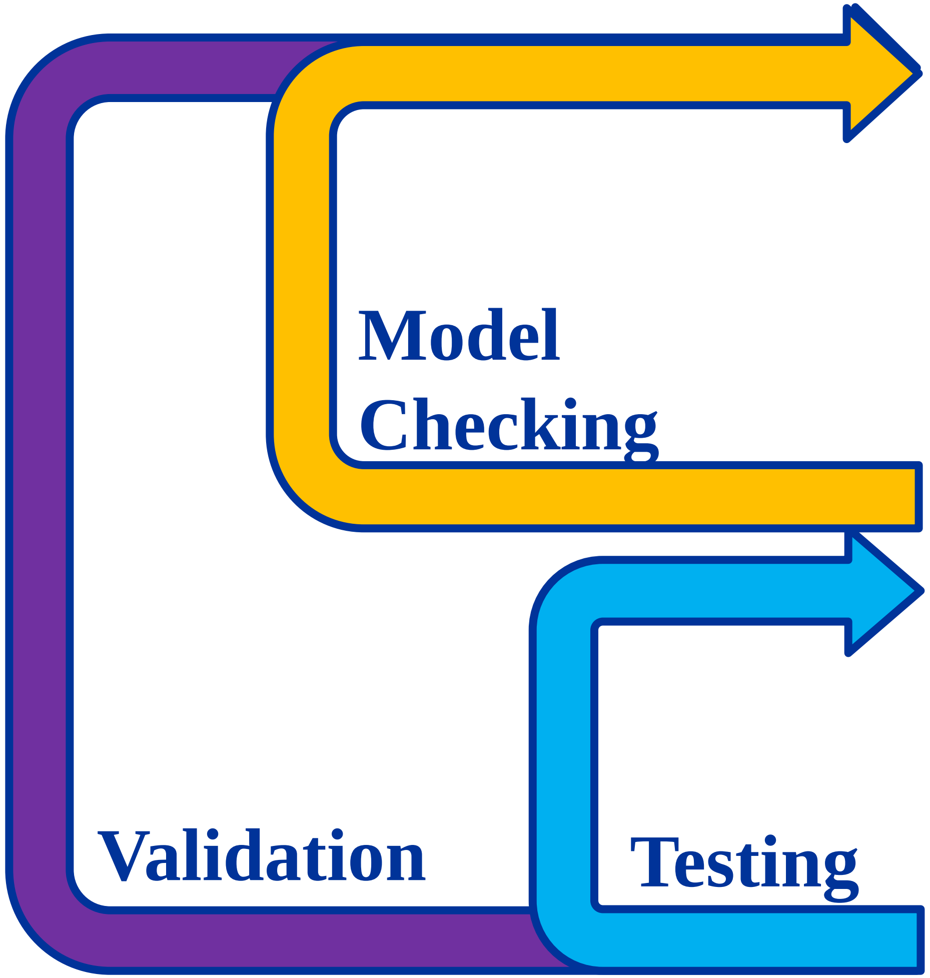
1. Effective communication
   - Between the engineering team and other stakeholders
   - Within the engineering team

2. Risk Management
   - How to balance conflicting judging criteria?

3. Validation
   - How do you know the software is effective/safe/secure?

Lecture 2: Software Life Cycle

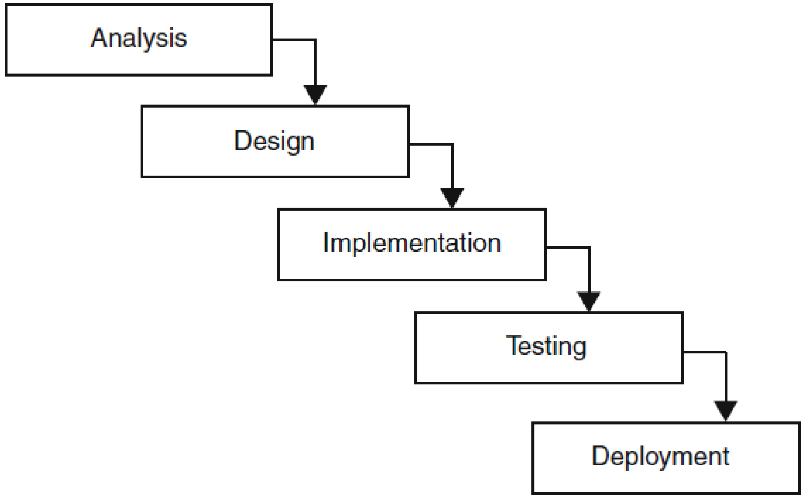
Three Most Important Artifacts



- Requirements **What**
  - i.e. the vehicle should not crash with any other objects
- Specifications **How**
  - i.e. when the vehicle detects an object on its path, apply brake
- Implementation

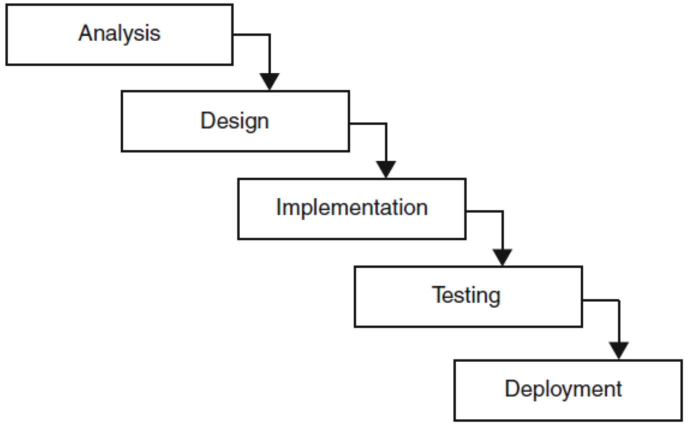
Waterfall Software Development Model

- A new phase begins only when the previous phase has been fully completed
- Intend to ensure full attention on one stage at a time X



Cons: Waterfall Software Development Model

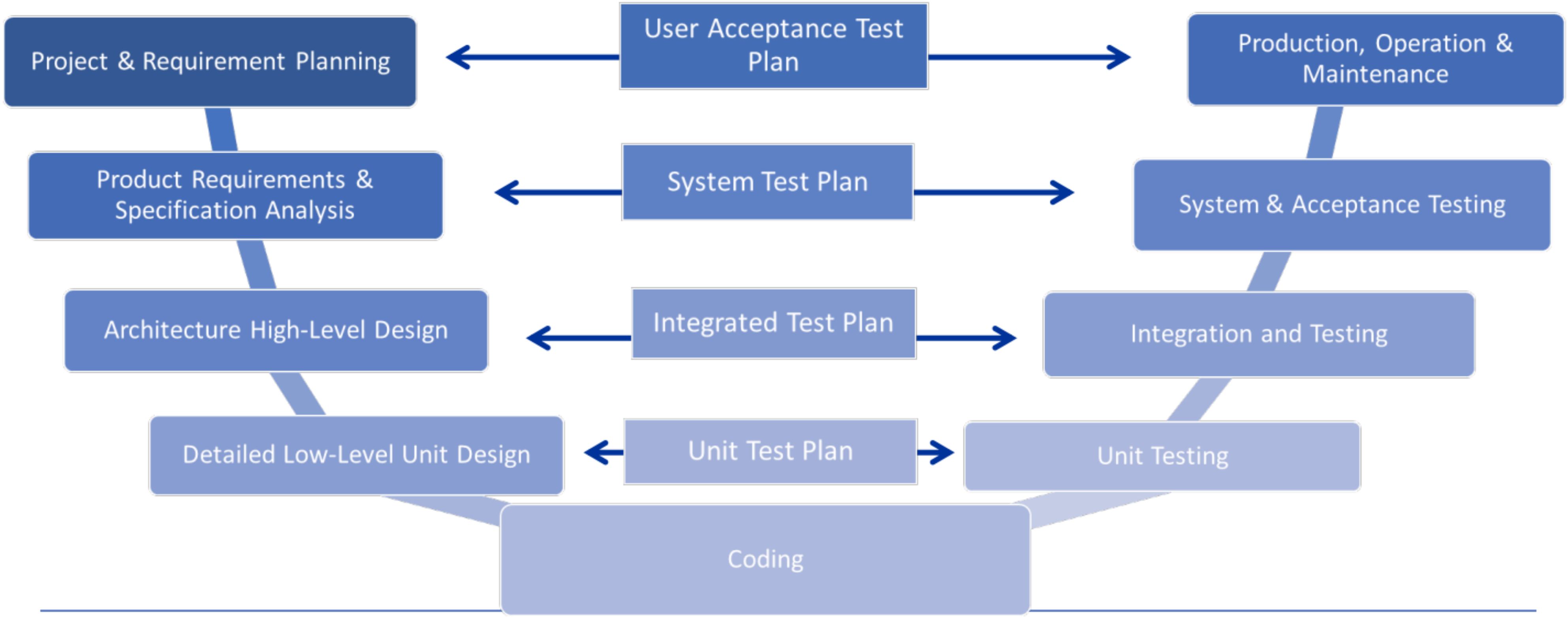
- Inflexible: Assume ideal situation which does not consider
  - Communication failures
  - Human errors
  - Change of requirements
- No feedback: No tangible product available for assessment until very late



When to use the Waterfall model?

- When the requirements are established hand-on and well known to the team;
- When the technology is mastered by the team;
- The project has a stable plan and product definition;
- When updating or creating a new version of an existing product;
- When porting an existing product to a new platform

V-shape Model



Project & Requirement Planning

User Acceptance Test Plan

Production, Operation & Maintenance

Product Requirements & Specification Analysis

System Test Plan

System & Acceptance Testing

Architecture High-Level Design

Integrated Test Plan

Integration and Testing

Detailed Low-Level Unit Design

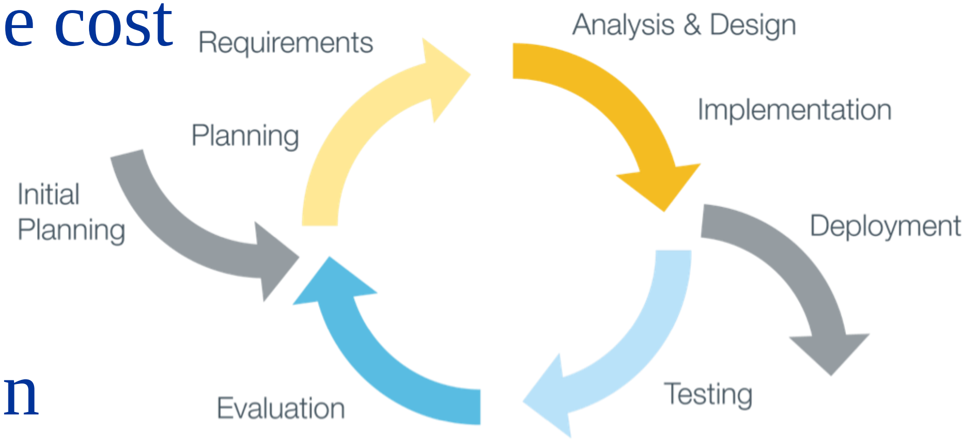
Unit Test Plan

Unit Testing

Coding

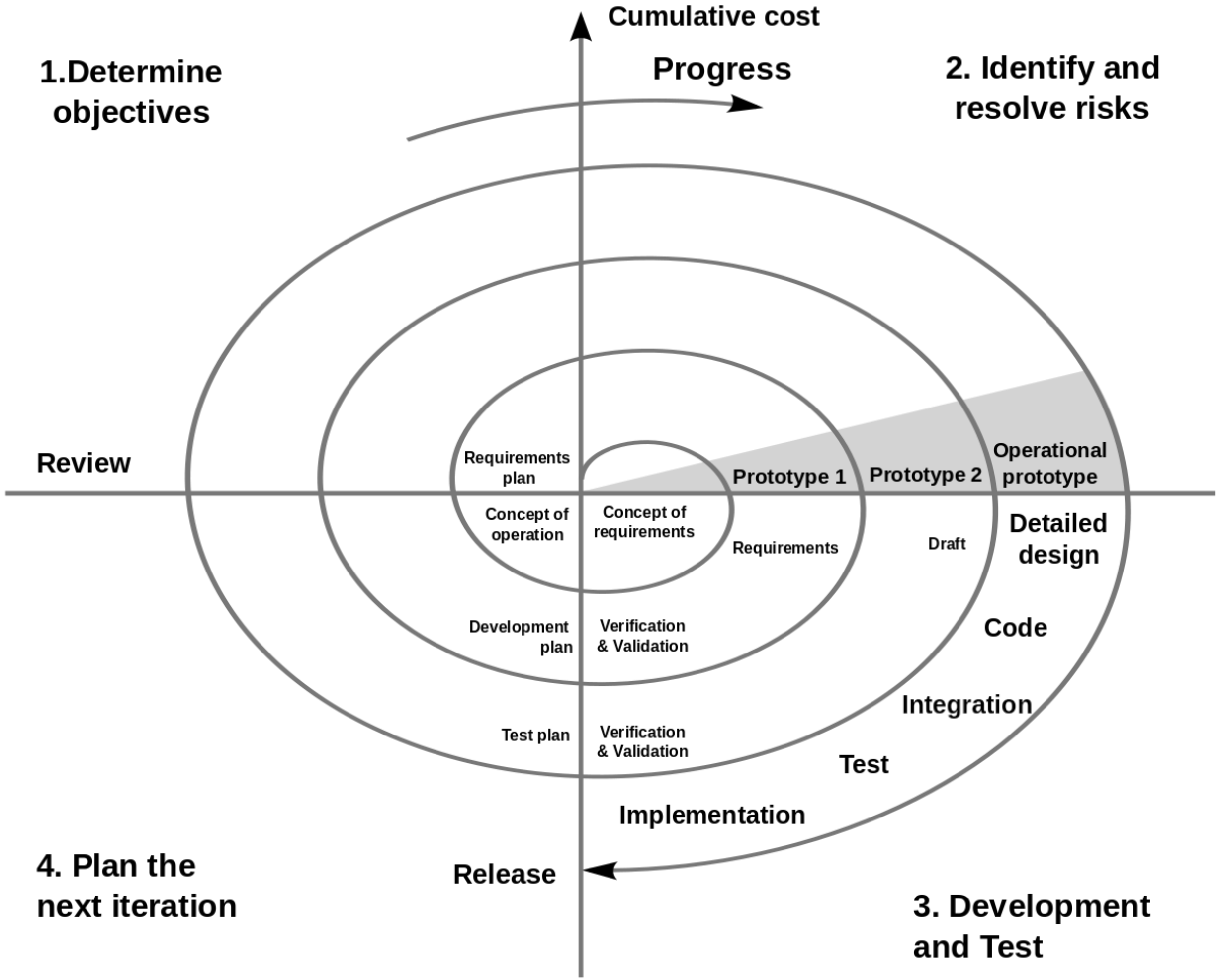
The Importance of Intermediate Artifacts

- Find problems early can significantly reduce cost
  - Tools and methodologies available to analyze intermediate artifacts
- Reduce ambiguity due to miscommunication
  - An executable product is the best communication tool
- What’s in the first prototype?
  - What should be added in each iteration?



Spiral software development model

- Each cycle represents an iteration in the development process
- Client feedback after each iteration
- Iterations guided using risk management



Agile Development

- Individuals and interactions over processes and tools;
- Working software over comprehensive documentation;
- Customer collaboration over contract negotiation;
- Responding to change over following a plan

My Experience in Software Engineering

- Safe software for autonomous medical devices (UPenn)
  - Developed tools and methodologies for software validation
  - Proposed model-based design framework for medical device software
  - Identified physiological requirements with physicians (domain experts)
  - Studied certification of medical device software with regulators

- Software and systems for connected cars (Toyota ITC)
  - Learned the business perspective of software products.
    - Learned how to convert company vision to concrete projects
  - How legacy tools and best-practice affect software design
  - How risk management is used when developing a product
  - How does R&D work? From research to advanced development to production

Why Early Prototyping?

An analogy from the movie industry

Stakeholders

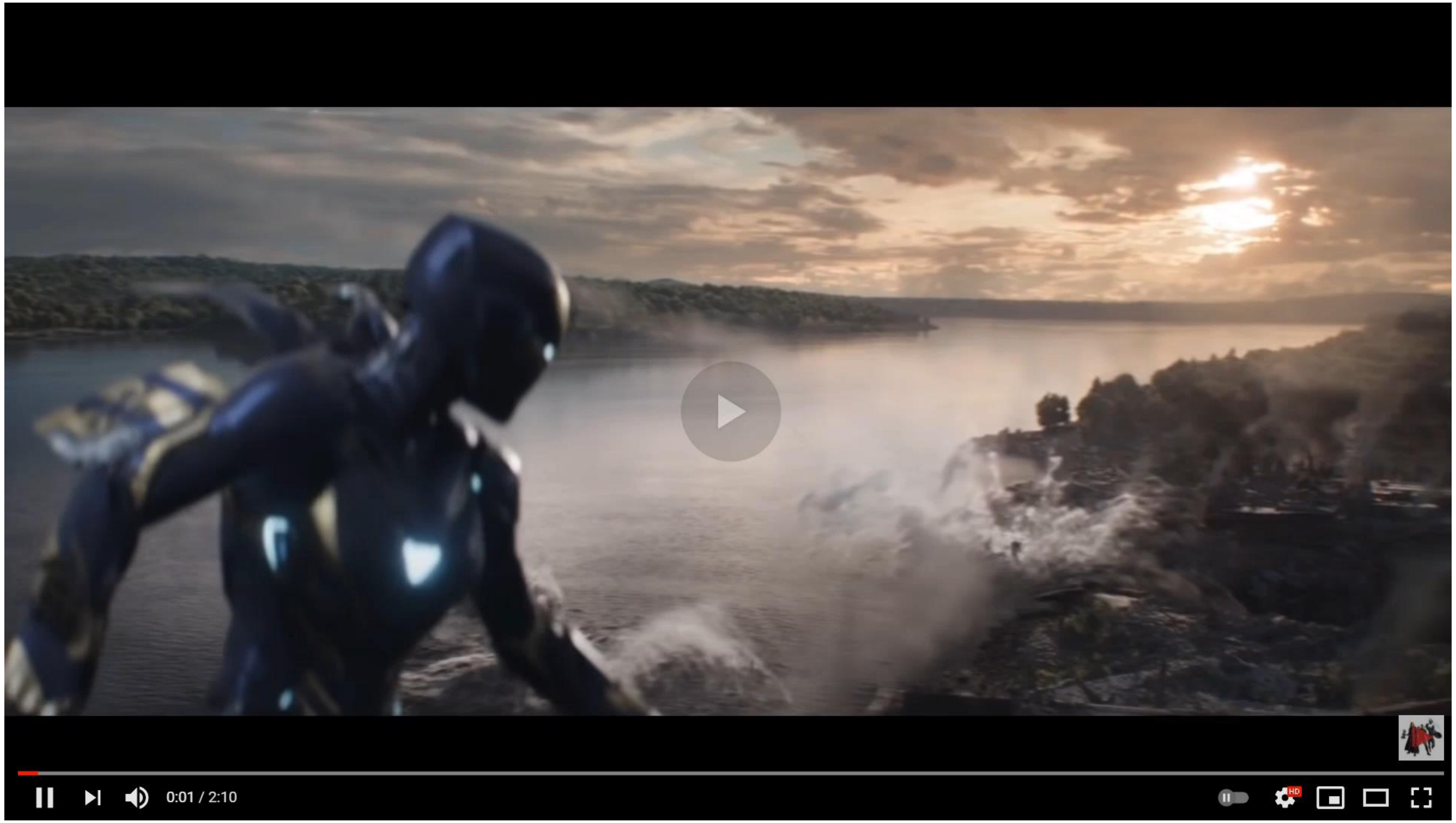
- Investor
- Production Team
  - Director
  - Actors
- Audience

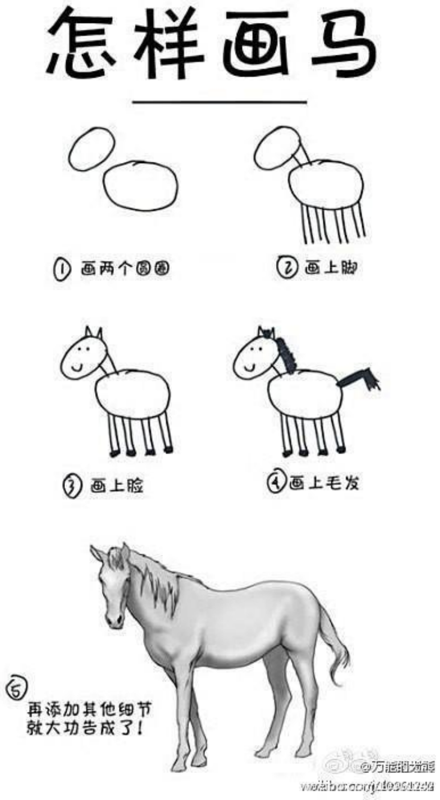
How can the production team convince the investor that they can make a good movie that makes profit?

Scripts – The Avengers Endgame

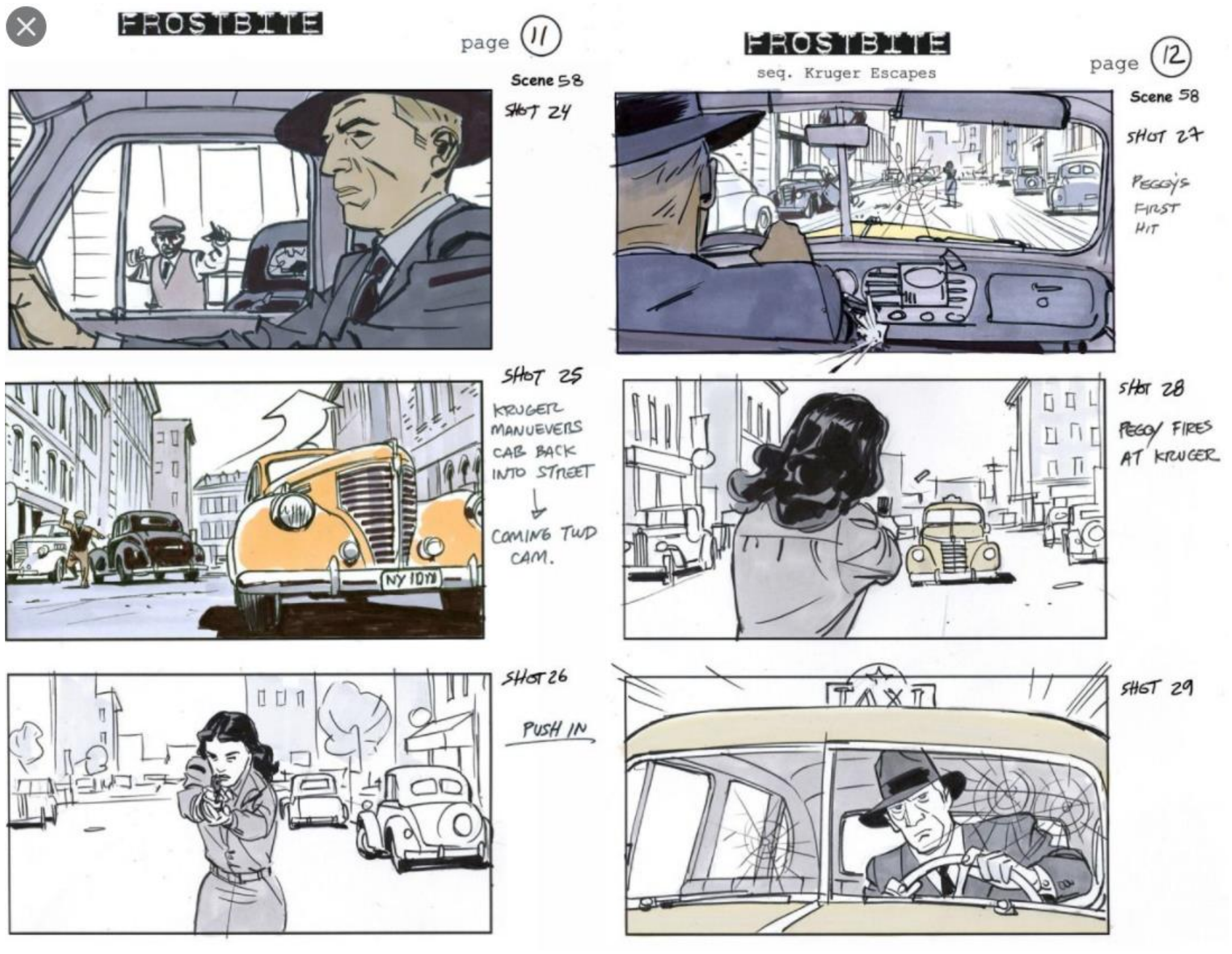
- The barrage destroys many on the battlefield
- The Barrage opens the riverbank and threatens to flood the battlefield, Dr. Strange and the other sorcerers have to hold the floodwater back
- Peter Parker and the Gauntlet is about to be overwhelmed by enemy forces BUT Steve hurls Mjolinir
- Peter catches a ride on it then with Valkyrie, BUT the ship's cannon fire knocks both Peter and Valkyrie to the ground
- The ship's fire is going to KILL THEM ALL. THE FIRE IS CLOSING IN ON THEM WITH NO ESCAPE. ALL IS LOST…

The Final Movie Clip



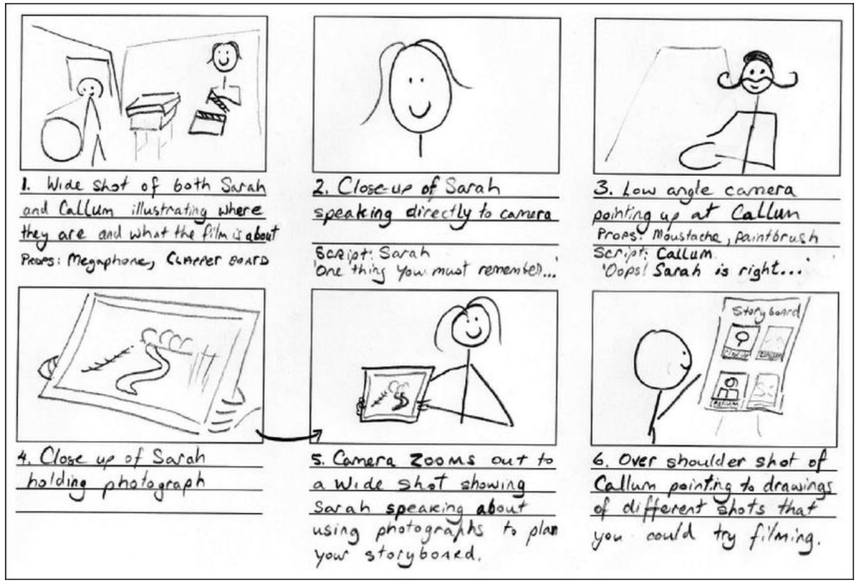


Storyboard


FROSTBITE
page 11
Scene 58
SHOT 24
SHOT 25
KRUGER MANUEVERS CAB BACK INTO STREET
COMING TWD CAM.
SHOT 26
PUSH IN
FROSTBITE
seq. Kruger Escapes
page 12
Scene 58
SHOT 27
PEGGY'S FIRST HIT
SHOT 28
PEGGY FIRES AT KRUGER
SHOT 29
TAXI

LUCKY STAR
CAB COMPANY
0:00 / 3:58

More Storyboards



1. Wide shot of both Sarah and Callum illustrating where they are and what the film is about
Props: Megaphone, Clapper board

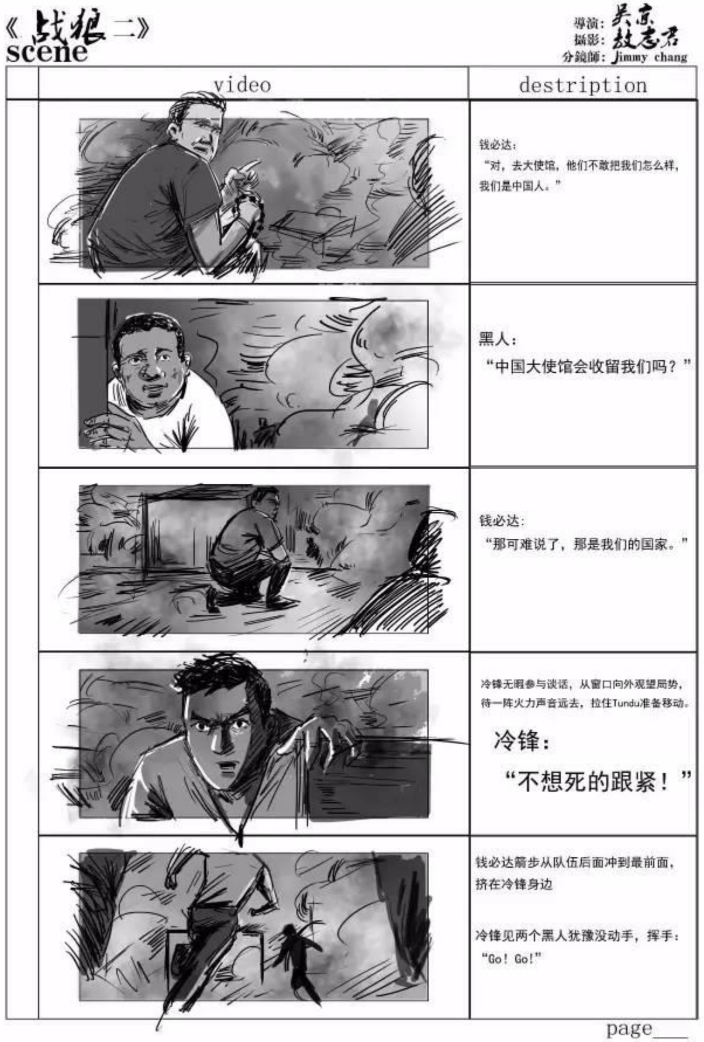
2. Close-up of Sarah speaking directly to camera
Script: Sarah
'One thing you must remember...'

3. Low angle camera pointing up at Callum
Props: Moustache, paintbrush
Script: Callum
'Oops! Sarah is right...'

4. Close up of Sarah holding photograph

5. Camera zooms out to a wide shot showing Sarah speaking about using photographs to plan your storyboard.

6. Over shoulder shot of Callum pointing to drawings of different shots that you could try filming.



《战狼二》
scene

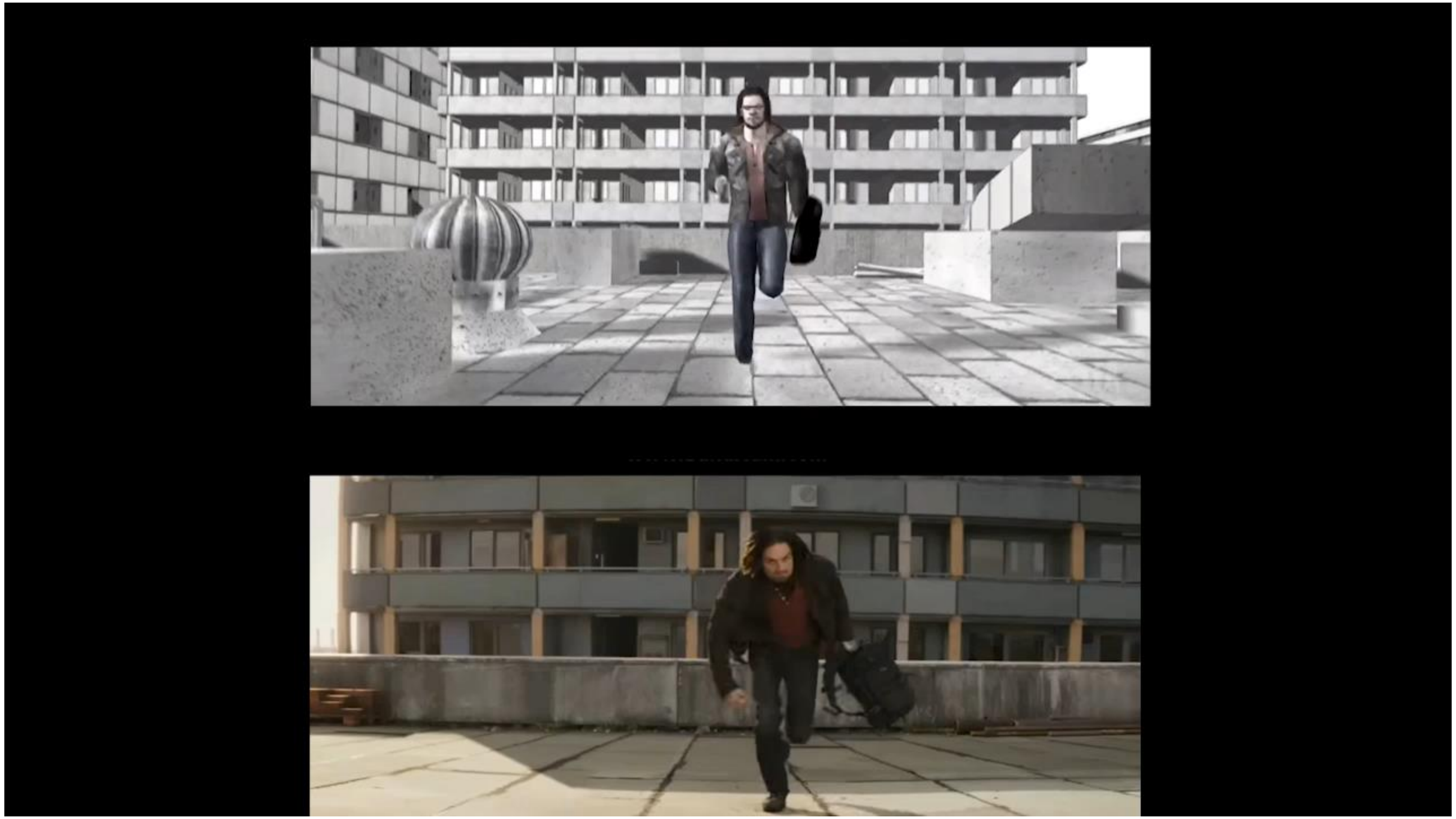
導演：吴京
攝影：赵志君
分鏡師：Jimmy chang

	video	destription
		钱必达： “对，去大使馆，他们不敢把我们怎么样，我们是中国人。”
		黑人： “中国大使馆会收留我们吗？”
		钱必达： “那可难说了，那是我们的国家。”
		冷锋无暇参与谈话，从窗口向外观望局势，待一阵火力声音远去，拉住Tundu准备移动。 冷锋： “不想死的跟紧！”
		钱必达箭步从队伍后面冲到最前面，挤在冷锋身边 冷锋见两个黑人犹豫没动手，挥手： “Go! Go!”

page___

Previs

Previs (Pre-visualization)



What we learned

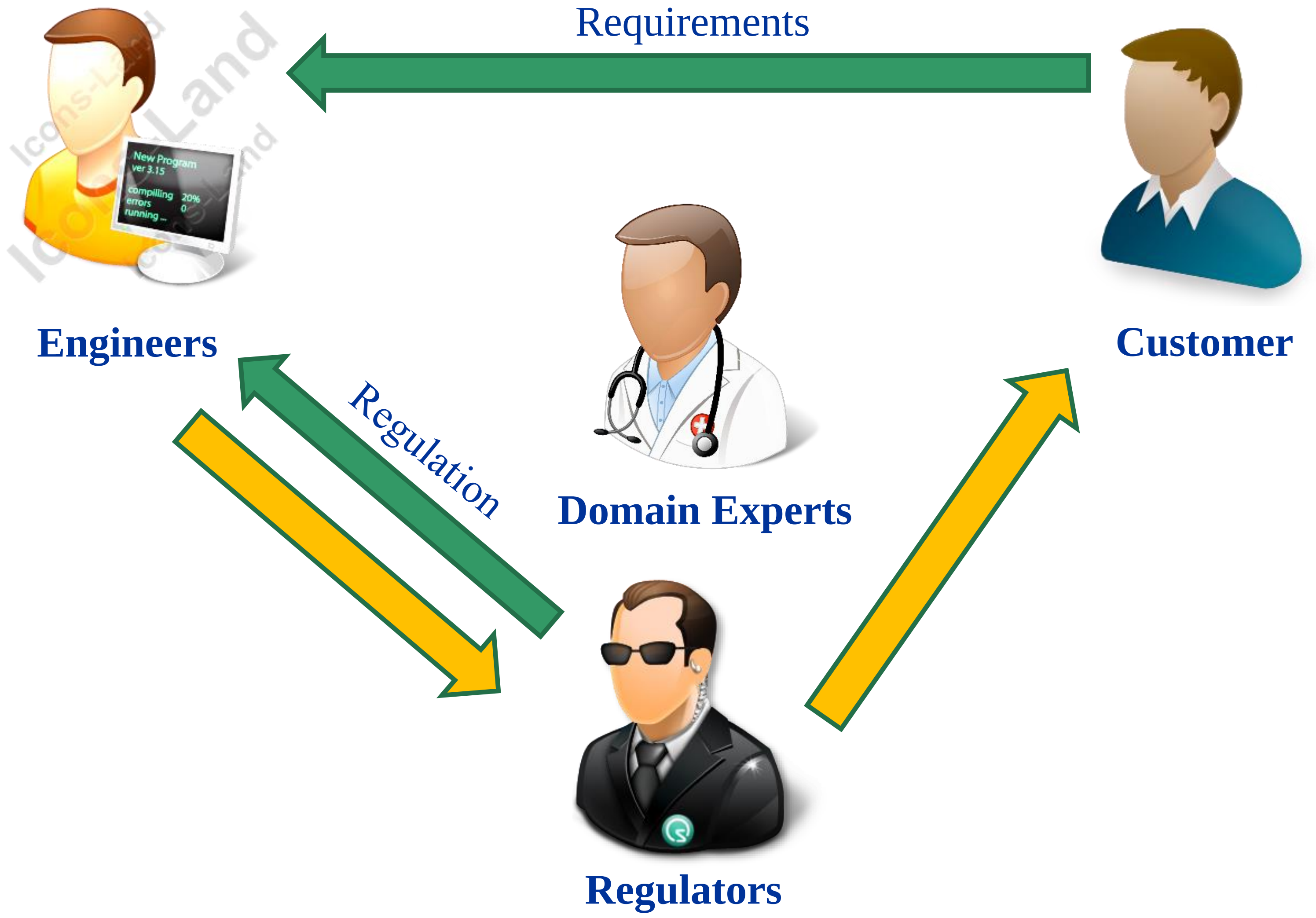
- Quick prototyping
  - Gets feedbacks early
  - Saves money
  - Earns trust
- Just having a good idea is not enough
- Mastering the new tools is very important

The Analogy

Movie Making	**Software Development**
• Script	• Requirement document
• Storyboard	• UML
• Previs	• Models->Prototypes
• Techvis	• Model translation & Code generation

Step 1: Software Requirement

Stakeholders for software



Requirements

Engineers

Customer

Regulation

Domain Experts

Regulators

Composition of an Engineering Team

- Business analyst
  - In charge of developing requirements
  - Interacts with customer and domain experts
- Developer
  - In charge of developing specifications that satisfy the requirements
- Tester
  - In charge of validating the design and implementation
  - Interacts with regulators

Software Requirement

- Requirements: expected services of the system and constraints that the system must obey

- Functional Requirements
  - What the system must achieve

- Non-functional Requirements
  - Software quality: How well the system can do its job, etc

- Domain Requirements
  - Easy to omit as domain experts may think they are “obvious”

Functional Requirements

- Functions, tasks, or behaviors the system must fully support.
  - How user of the system use the system
- The “skeleton” of the system requirements
  - Should be captured in early iterations
- Need to distinguish “core functions” from “features”

Non-Functional Requirements

- Constraints placed on various attributes of system functions or tasks
- Equally important compared to functional requirements
  - Separate software products from software practices
- Sources
  - Domain: i.e. Human can tolerate up to 150ms delay in voice communication
  - Legacy: i.e. QWERTY keyboard
  - User: i.e. User want to operate the interface with one hand
  - Regulation: The system should switch to backup and resume within 1ms after the primary program crashes

Examples of Non-Functional Requirements

- User interface and human factors:
  - What type of user will be using the system?
  - Will more than one type of user be using the system?
  - What sort of training will be required for each type of user?
  - Is it particularly important that the system be easy to learn?
  - Is it particularly important that users be protected from making errors?
  - What sort of input/output devices for the human interface are available, and what are their characteristics?

Examples of Non-Functional Requirements

- Performance characteristics
  - Are there any speed, throughput, or response time constraints on the system?
  - Are there size or capacity constraints on the data to be processed by the system?
- Error handling and extreme conditions
  - How should the system respond to input errors?
  - How should the system respond to extreme conditions?

Examples of Non-Functional Requirements

- Quality issues
  - What are the requirements for reliability?
  - Must the system trap faults?
  - What is the maximum time for restarting the system after a failure?
  - Is it important that the system be portable (able to move to different hardware or operating system environments)?
- System Modifications
  - What parts of the system are likely candidates for later modification?
  - What sorts of modifications are expected (levels of adaptation)?
  - Might unwary adaptations lead to unsafe system states?

Identifying Non-functional Requirements

- Certain constraints are related to the design solution that are unknown at the requirements stage.
- Certain constraints are highly subjective and can only be determined through complex, empirical evaluations.
- Non-functional requirements tend to conflict and contradict.
- There is no ‘universal’ set of rules and guidelines for determining when nonfunctional requirements are optimally met.

Requirement Elicitation

- Step 1: (Business analyst) develops common understanding of the problem domain with (customers) and (domain experts)

- Step 2: (Business analyst) explains the problem to (the development team) and develop a design strategy

- Step 3: (Business analyst) presents the design strategy to the customer, and agree on technical solutions

Business analysts

- Need to be familiar with the problem domain and development techniques
- The bridge between the customers and the development team
  - To the customers:
    - Explain in domain language what can/cannot be achieved with existing constraints
    - Hide technical details when explaining the technical solution to the customers
    - Create user manual
  - To the development team:
    - Reformulate the domain problem as mathematical problems

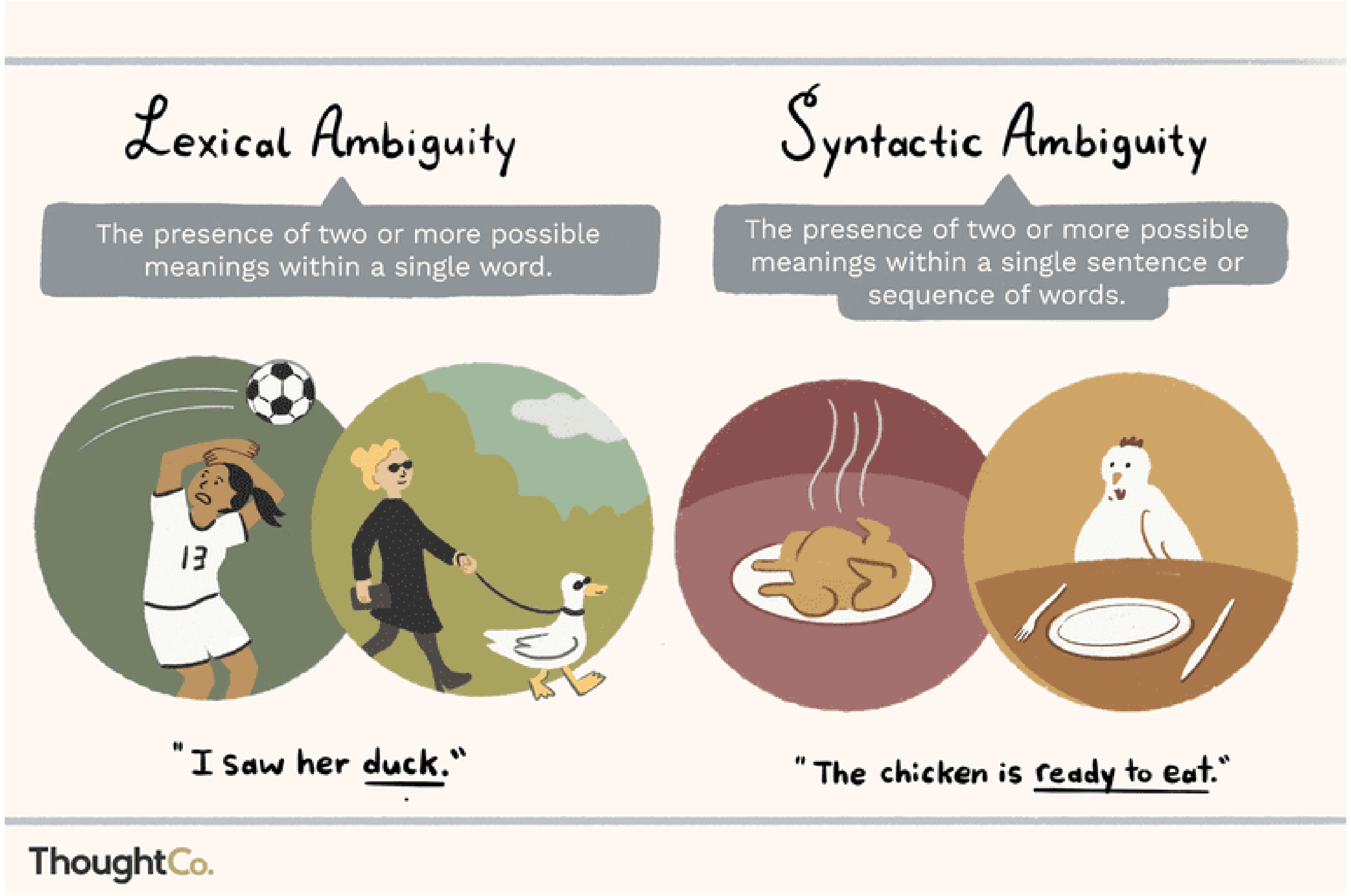
Common Problems During Requirement Elicitation

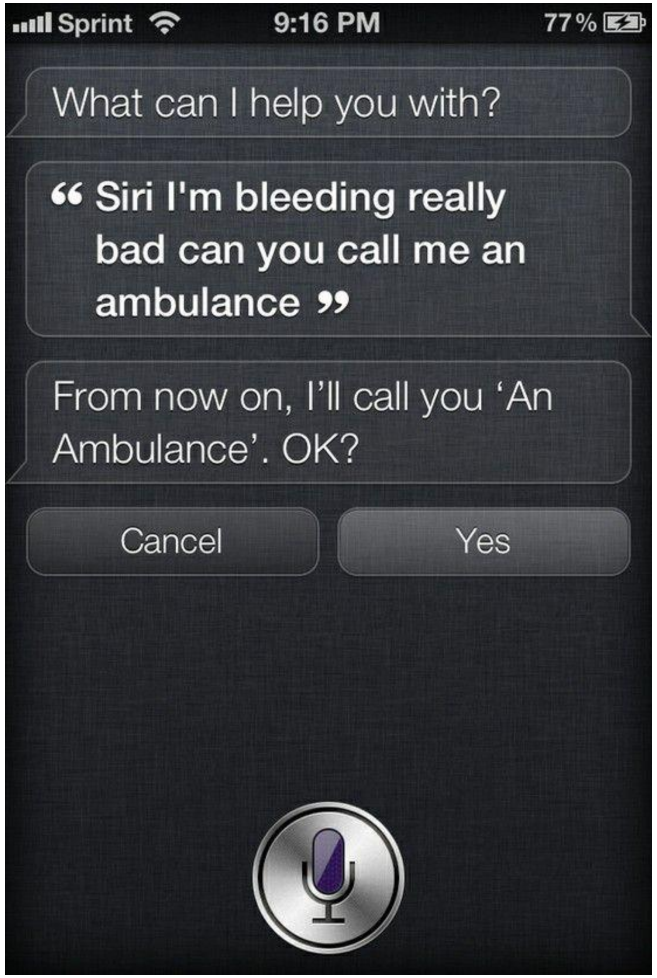
- Problem of scope
  - What environmental condition the system will operate in?
- Problem of understanding
- Problem of volatility
  - User needs evolve over time

Problem of Understanding

- The customer fails to explain their needs well.
  - Need a common language
- The analyst may not understand the customer's need.
  - Need to study the problem domain
- The customer may not know what he/she wants
  - The team should identify customer needs from the problem domain
- The analyst may not clearly convey the requirements to the development team
  - Problem abstraction

Natural Languages Are Prone to Ambiguities


Lexical Ambiguity
The presence of two or more possible meanings within a single word.
Syntactic Ambiguity
The presence of two or more possible meanings within a single sentence or sequence of words.
13
"I saw her duck."
"The chicken is ready to eat."
ThoughtCo.



What can I help you with?

“ Siri I'm bleeding really bad can you call me an ambulance ”

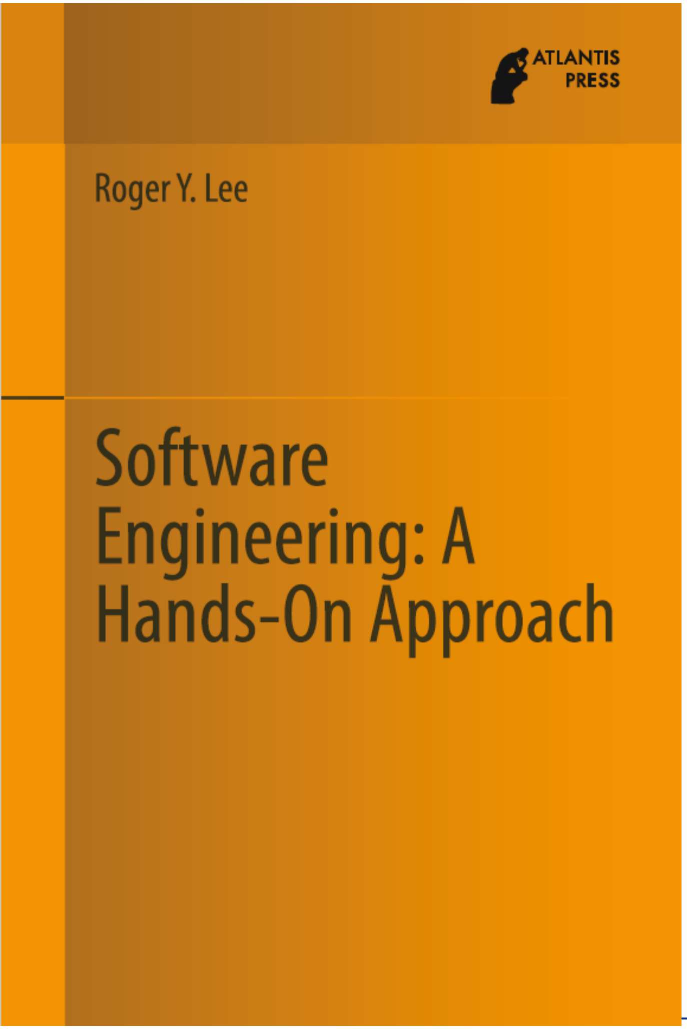
From now on, I’ll call you ‘An Ambulance’. OK?

We need a widely used formal language

Communications among various stakeholders

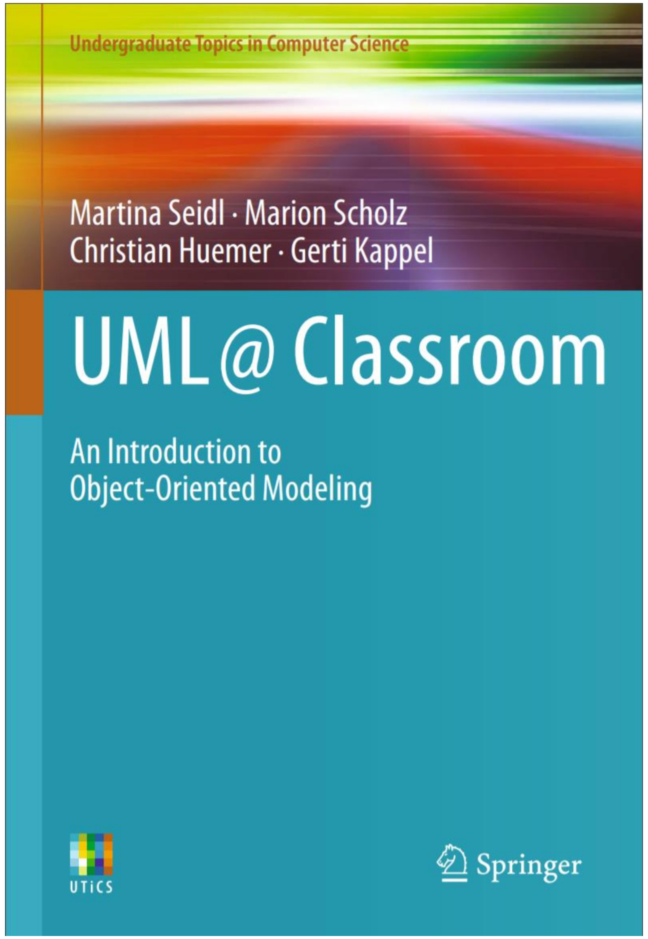
- Need a common language for communication
- Unified Modeling Language (UML)
- Recognized as an international standard
- It’s just a tool, not a solution

Reference Book



Reference for UML

- Freely available online
- Search from our library website


Undergraduate Topics in Computer Science
Martina Seidl · Marion Scholz
Christian Huemer · Gerti Kappel
UML @ Classroom
An Introduction to
Object-Oriented Modeling
UTiCS
Springer

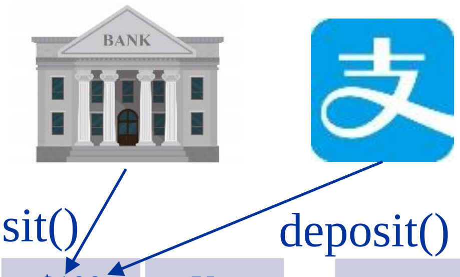
Procedure-Oriented Software Design

- Describe problems in terms of functions: y=f(x)
- Behaviors hard to describe as procedure

Procedure-Oriented Software Design

- Sensitive to requirement changes
- Nothing reusable
- Less intuitive (Communication problems)
- No information hiding

```
graduate()
{
    returnCafe();
    dropClass();
    returnBook();
}
```



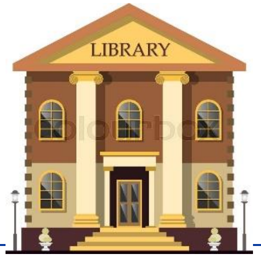
deposit()

deposit()

Joe	202001	$100	Yes
Jane	202002	$200	No

Joe	202001	CS132
Jane	202002	CS233

Joe	202001	Book 1
Jane	202002	Book 2



DIY Community in Electrical Engineering

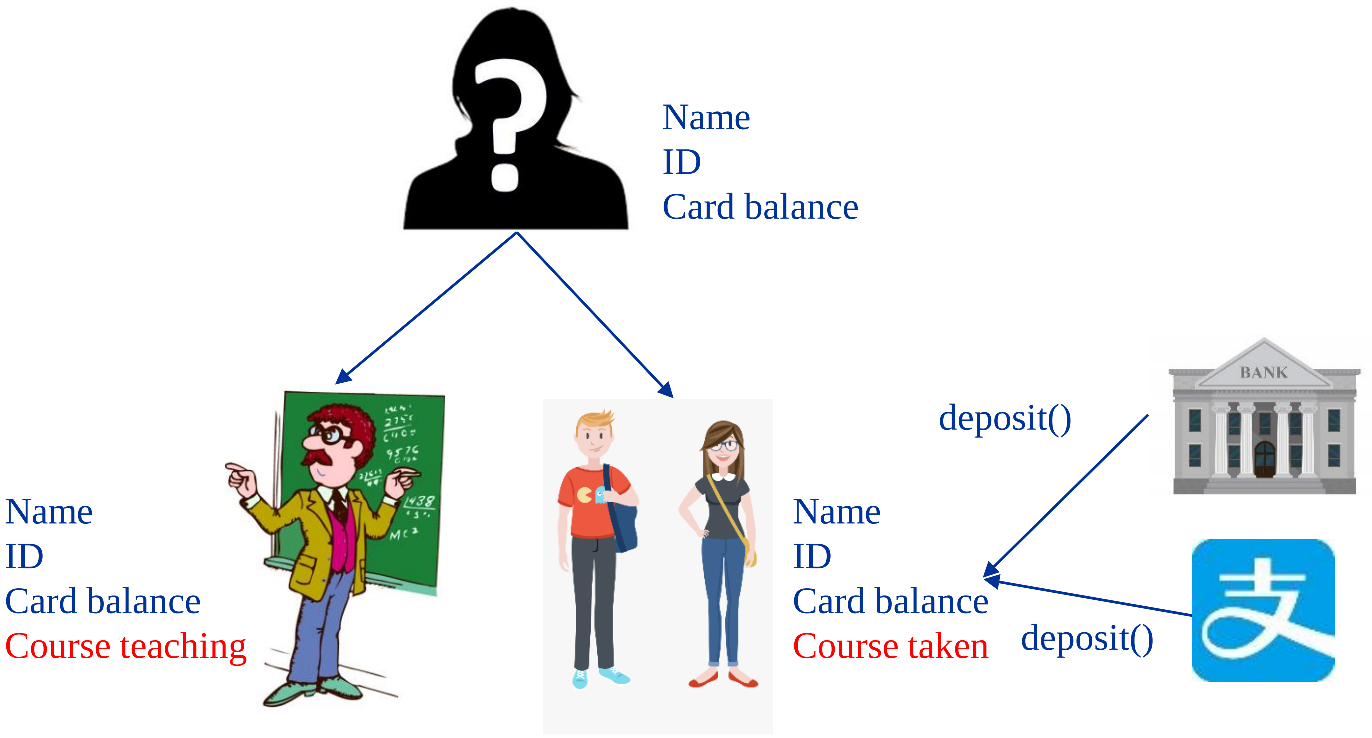
- Standardized “building blocks”
  - Easily accessible

- Standardized interface
  - Interchangeable components

- Can we define a software system as a collection of objects of various types that interact with each other through well-defined interfaces?

Object-Oriented Software Design

- Describe problems as objects and interactions between objects
- Much more intuitive



Name
ID
Card balance

Name
ID
Card balance
Course teaching

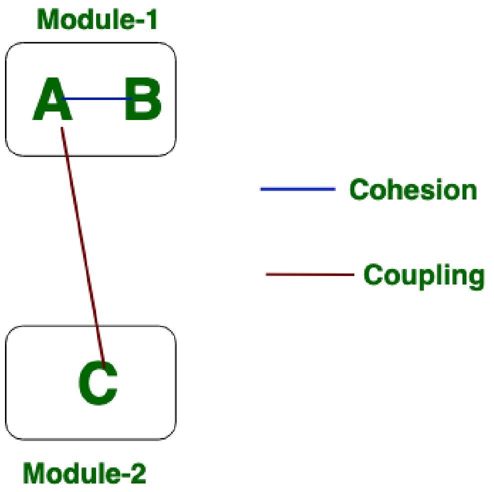
Name
ID
Card balance
Course taken

deposit()

deposit()

Benefits of OO

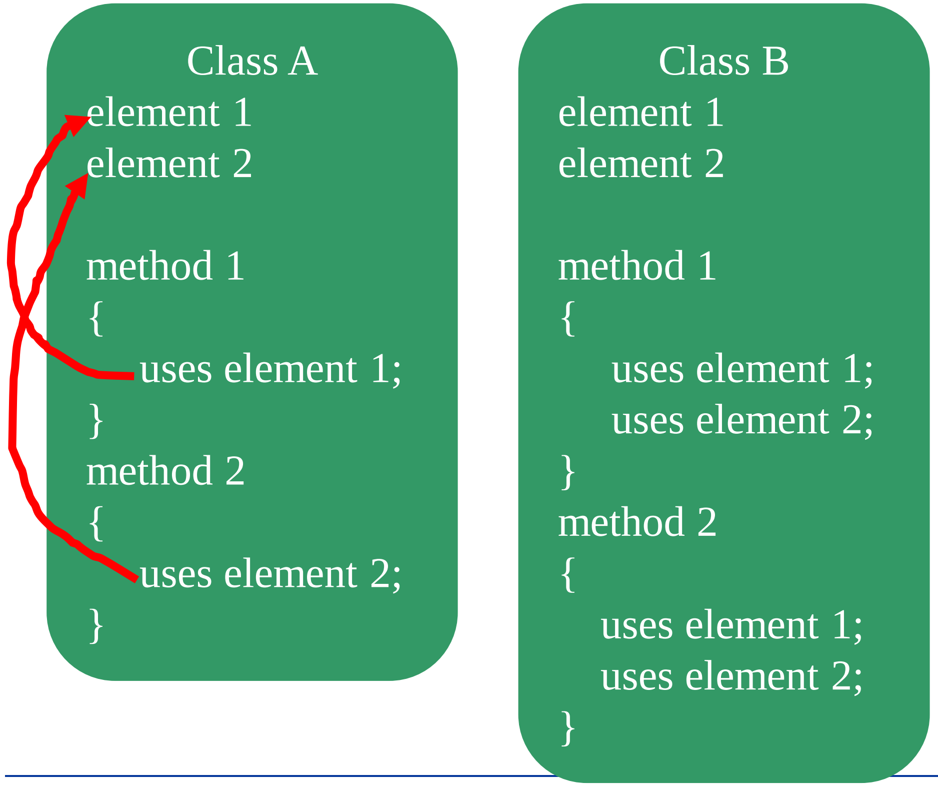
- Modularity: Decompose a system into a set of cohesive and loosely coupled modules
  - Reusability
    - Accidental vs. deliberate reuse
  - Encapsulation and information hiding
    - Interfaces
  - Access levels
    - Reduce coupling



- Inheritance: a relationship between different classes in which one class shares attributes of one or more different classes

Cohesion vs. Coupling

- Low vs. high cohesion



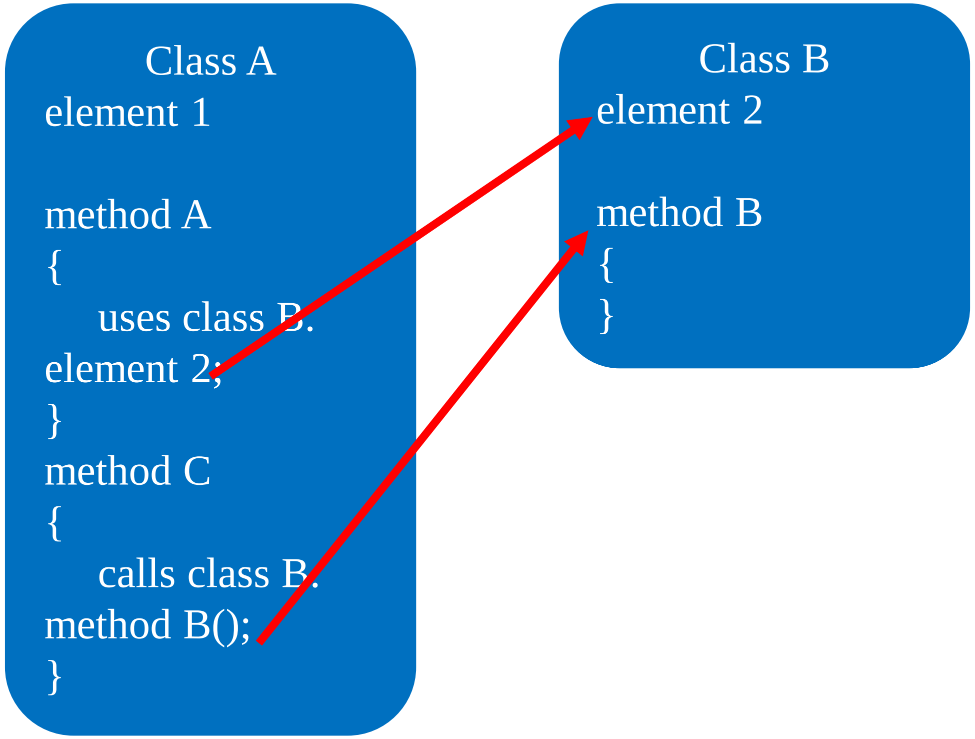
```
Class A
element 1
element 2

method 1
{
    uses element 1;
}
method 2
{
    uses element 2;
}
```

```
Class B
element 1
element 2

method 1
{
    uses element 1;
    uses element 2;
}
method 2
{
    uses element 1;
    uses element 2;
}
```

- Tight Coupling (avoid)



```
Class A
element 1

method A
{
    uses class B.
element 2;
}
method C
{
    calls class B.
method B();
}
```

```
Class B
element 2

method B
{
}
```

Design Choices

- A method of an object may only call methods of:
  - The object itself.
  - An argument of the method.
  - Any object created within the method.
  - Any direct properties/fields of the object.
- Don’t talk to strangers!
- When one wants a dog to walk, one does not command the dog’s legs to walk directly; instead one commands the dog which then commands its own legs.