Building Software

September 20, 2022

Byung-Gon Chun

(Slide credits: George Candea, EPFL and Armando Fox, UCB)

Announcement

- Project proposal due 9/28 –
  Submit early! I already received proposals.
- HW3 out on 9/21!
- 9/29 - 3~5 selected proposal presentation
  - Selected teams should prepare for slides and present their proposals during class hours. I will announce the teams through a github issue.
- TA assignment

  Teams 1~7: 맹재우
  Teams 8~14: 유준열
  Teams 15~20: 윤종선

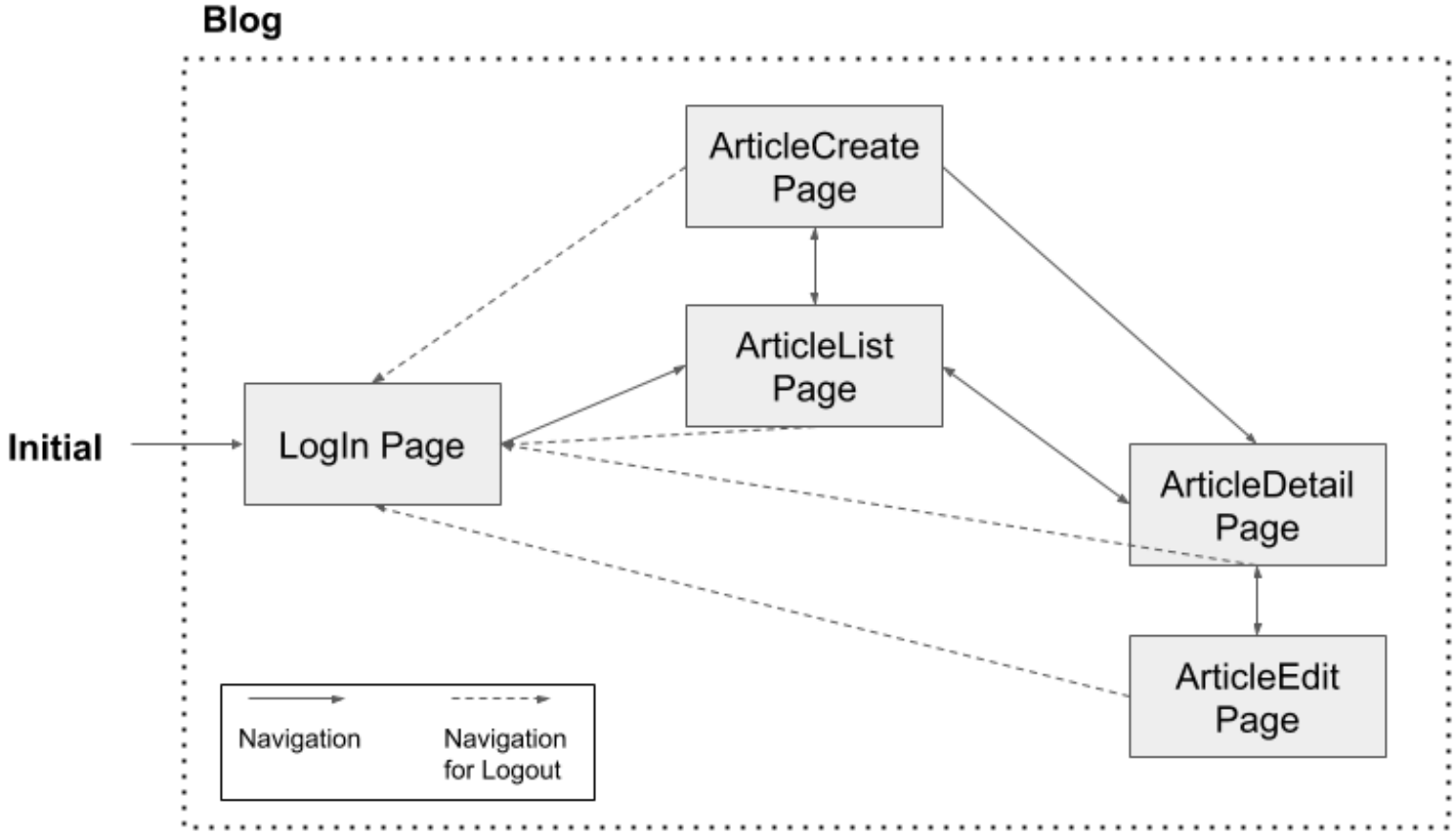
HW3: React Library & Testing

- Make a simple React application before diving into your projects
- Make test suites for the React application you implement
- Let you try out stuff we have learned in our practice sessions

- Two deadlines
  - Feature implementation due
  - Testing implementation due

HW3: Simple Blogging Frontend

- Our blog will support three models: User, Articles, and Comments. You are required to create a total of five pages.



HW3 Testing

- All pages/components should have proper unit tests to test its functionalities, using Jest and Enzyme that are covered in the practice session.
- All of your tests must pass.
- Your tests are expected to cover all of your code, and we will give credits according to your coverage results.
  - You can see the coverage information of your application by using npm test -- --coverage.
- We provide a simple json-server backend with our skeleton code.

HW3 Grading: Feature Implementation Score

- Log in page features (4 points)
- Article list page features (5 points)
- Article create page features (10 points)
- Article detail page features when the user is the author of the article (14 points)
- Article detail page features when the user is not the author of the article (13 points)
- Article edit page features (9 points)

HW3 Grading: Testing Score

- Your unit test score will be given based on both test coverage and completeness of your feature implementation. This is to make sure that your test code is faithful to the feature requirements.

Coverage	Points
90% and above	25 * ($score_{feature}$ / 55)
80% and above	20 * ($score_{feature}$ / 55)
70% and above	15 * ($score_{feature}$ / 55)
60% and above	10 * ($score_{feature}$ / 55)
below 60%	0

Note: All your test cases must pass

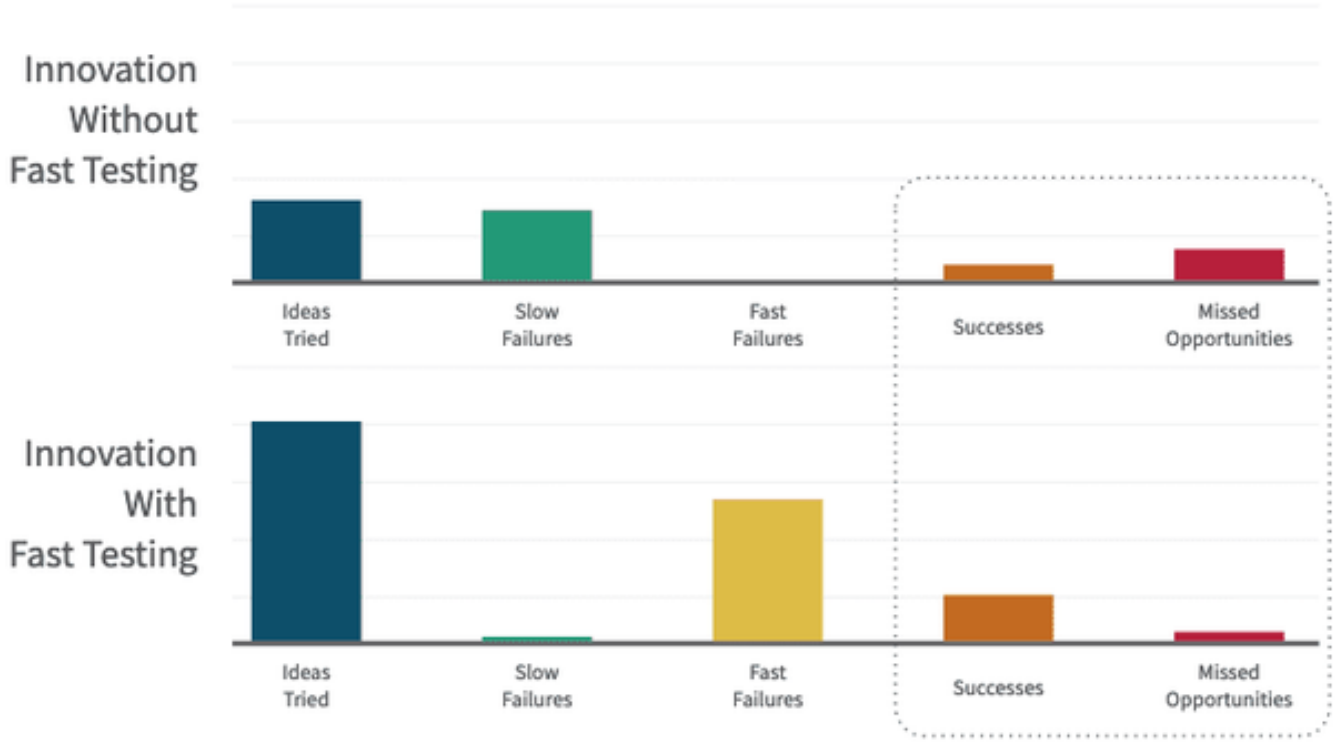
Steps for Building Software

Defining the Problem

- Answers key question #1:
  - *“What problem should the software solve ?”*
- Also known as
  - *product vision, vision statement, product definition, ...*
- Simple formula
  - *brief (no more than 2 pages)*
  - *no reference to possible solutions*
  - *stated in plain language, from the user’s point of view*

How Do You Know You're Solving a Right Problem?

- Most new ideas fail in the market—even if competently executed. We call this the Law of Market Failure. (Alberto Savoia)
- Test the market for your ideas objectively, rigorously, and quickly before you invest to develop them.
- Pretotyping: *The Right it*



The IBM Speech-to-Text Experiment

Three decades ago

“They put potential customers of the speech-to-text system, people who said they’d definitely buy it, in a room with a computer box, a screen and a microphone – but no keyboard. They told them they had built a working speech-to-text machine and wanted to test it to see if people liked using it. When the test subjects started to speak into the microphone their words appeared on the screen: almost immediately and with no mistakes! The users were impressed: it was too good to be true – which, as it turns out, it was.” (The Right it)

Formulating Requirements

- Answers key question #2:
  - *“What should the software do to solve the problem ?”*
- Also known as
  - *requirements doc, requirements definition, functional specification,...*
- Makes user’s requirements explicit
  - *acts as a contract between user and developer*
  - *avoids arguments with user and with other developers*
  - *error in functional spec → discard code and tests*

Good Functional Specs

- All user tasks
  - *description, expected response time, success vs. failure*
- All system inputs
  - *source, accuracy, value range, frequency of arrival*
- All system outputs
  - *destination, accuracy, value range, format, etc.*
- All interfaces with the rest of the world
  - *hardware, software, communication interfaces*

Good Functional Specs

- Requirements are verifiable
- Competing attributes can be resolved
  - *clear guidance on how to make tradeoffs*
- Clear connection to problem definition
  - *each item contributes to solving the problem*

Requirements Change

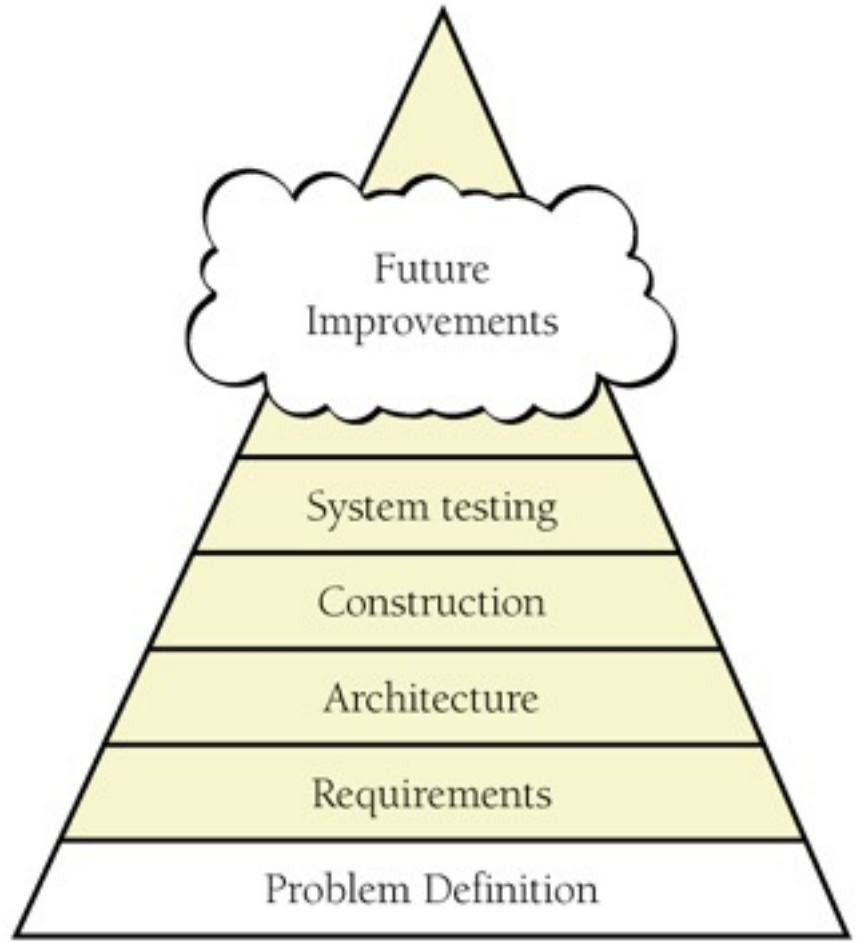
- Customer does not know what (s)he wants
  - *the development process helps clarify requirements*
- 25% of requirements change during development
  - *these account for 70%-85% of amount of rework*

Design: Software Architecture

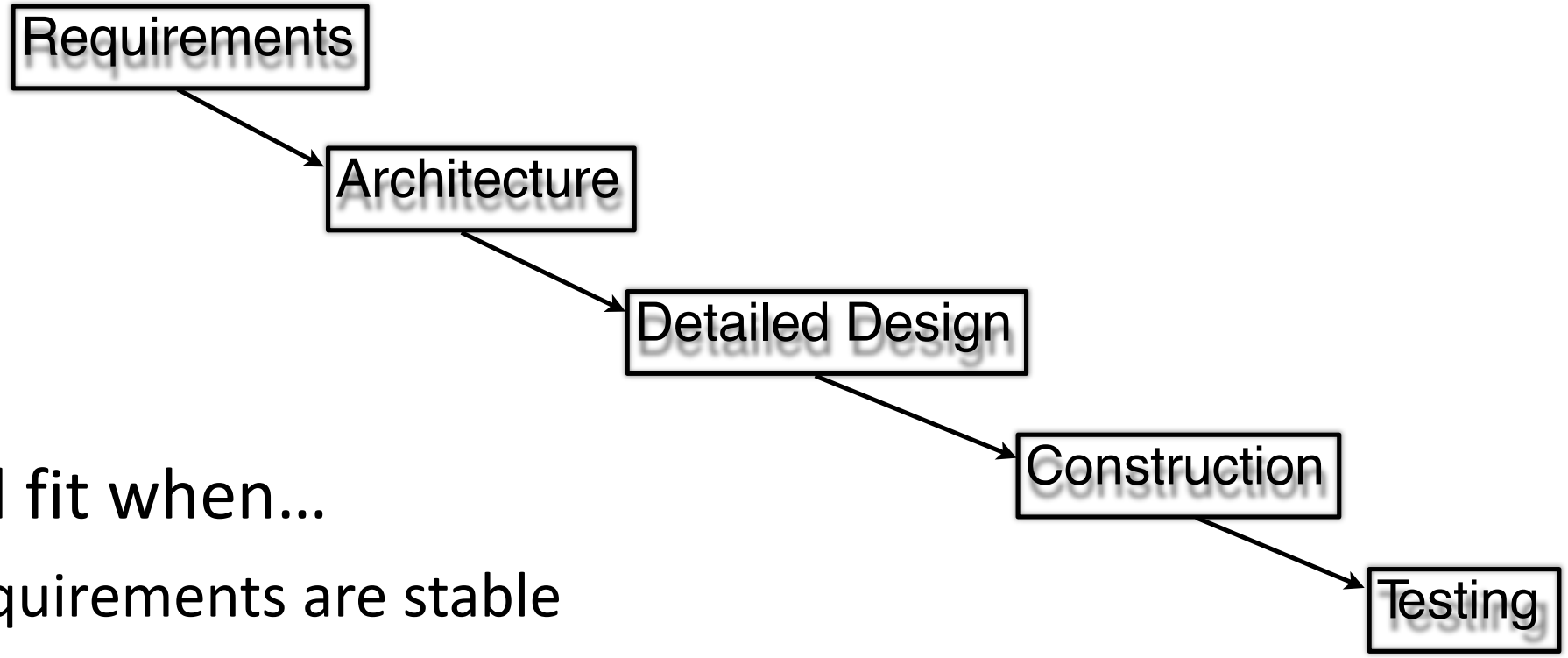
- Data design
  - *describe main data structures and files/DB tables*
  - *interoperability with other systems*
- User interface
  - *Web pages, GUI, CLI that enable user tasks from requirements*
  - *keep UI replaceable (useful for evolution, testing, etc.)*
- Resources
  - *estimates of maximum and average needs (e.g., for memory, disk space, threads, connections, network bandwidth, ...)*

Software Architecture

- What level of fault tolerance (1 fault, 2 faults, ...) ?
- Detection + handling + containment of errors/exceptions
- Build vs. reuse
  - *identify all reusable libs (GUI controls, communication, ...)*
- Internationalization
- Design for change / extensibility

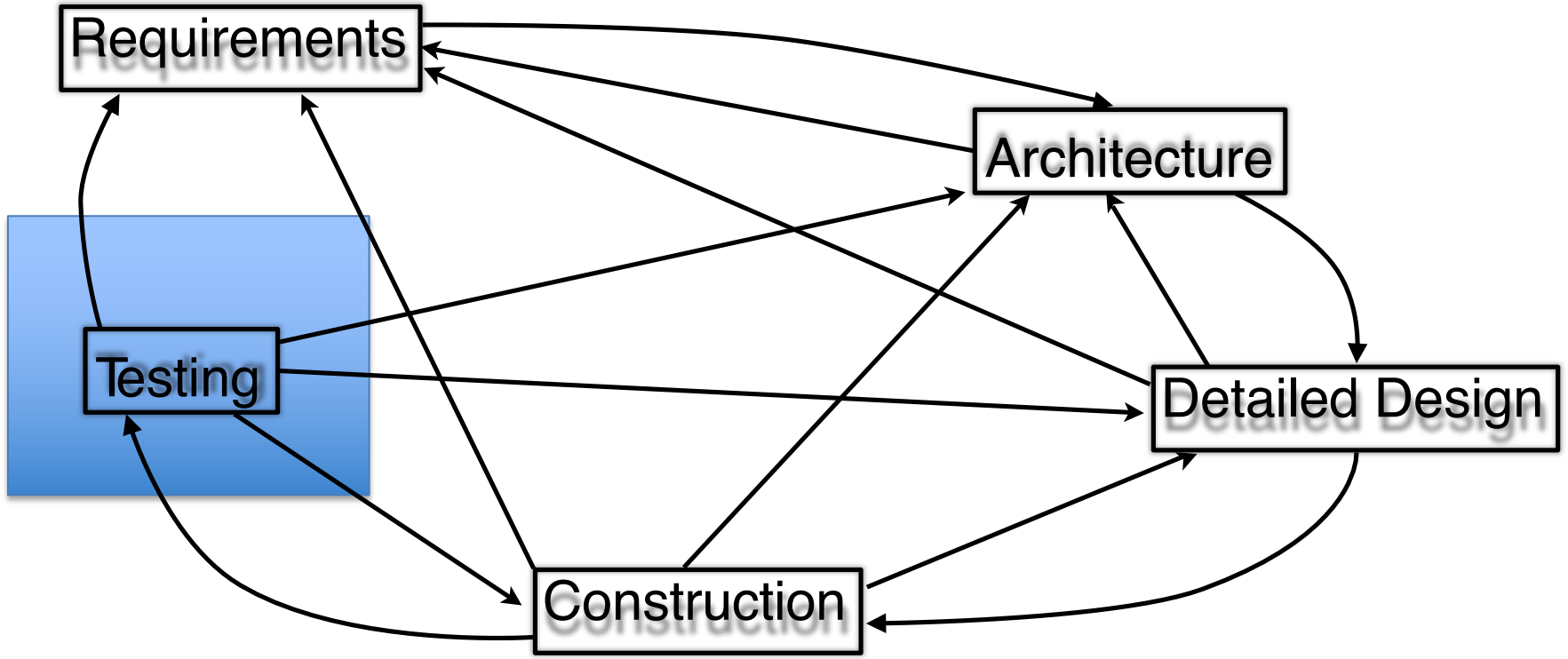

Future
Improvements
System testing
Construction
Architecture
Requirements
Problem Definition

Block Approach



- Good fit when…
  - Requirements are stable
  - Design is straightforward and well understood
  - Development team need not “experiment”

Agile Approach



- Good fit when…
  - Requirements are not fully fleshed out
  - Design is complex and challenging
  - Development team not familiar with the class of software being built

SWPP Project Sprints

- **Sprint 1**
  - Requirements and Specification
- **Sprint 2**
  - Design and Planning (Start Development and Testing!)
- **Sprint 3**
  - Repeat the same procedure
- **Sprint 4**
  - Repeat the same procedure
- **Sprint 5**
  - Repeat the same procedure
- **Demo Poster & Final Report**