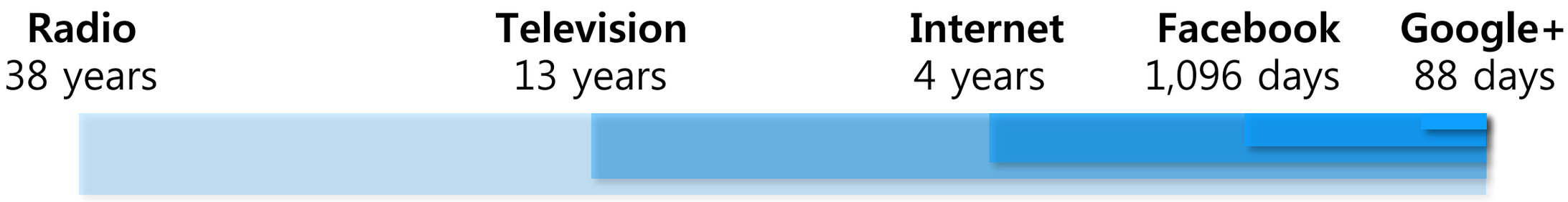
Code Review

류석영

Most material developed with Prof. Sooel Son at KAIST

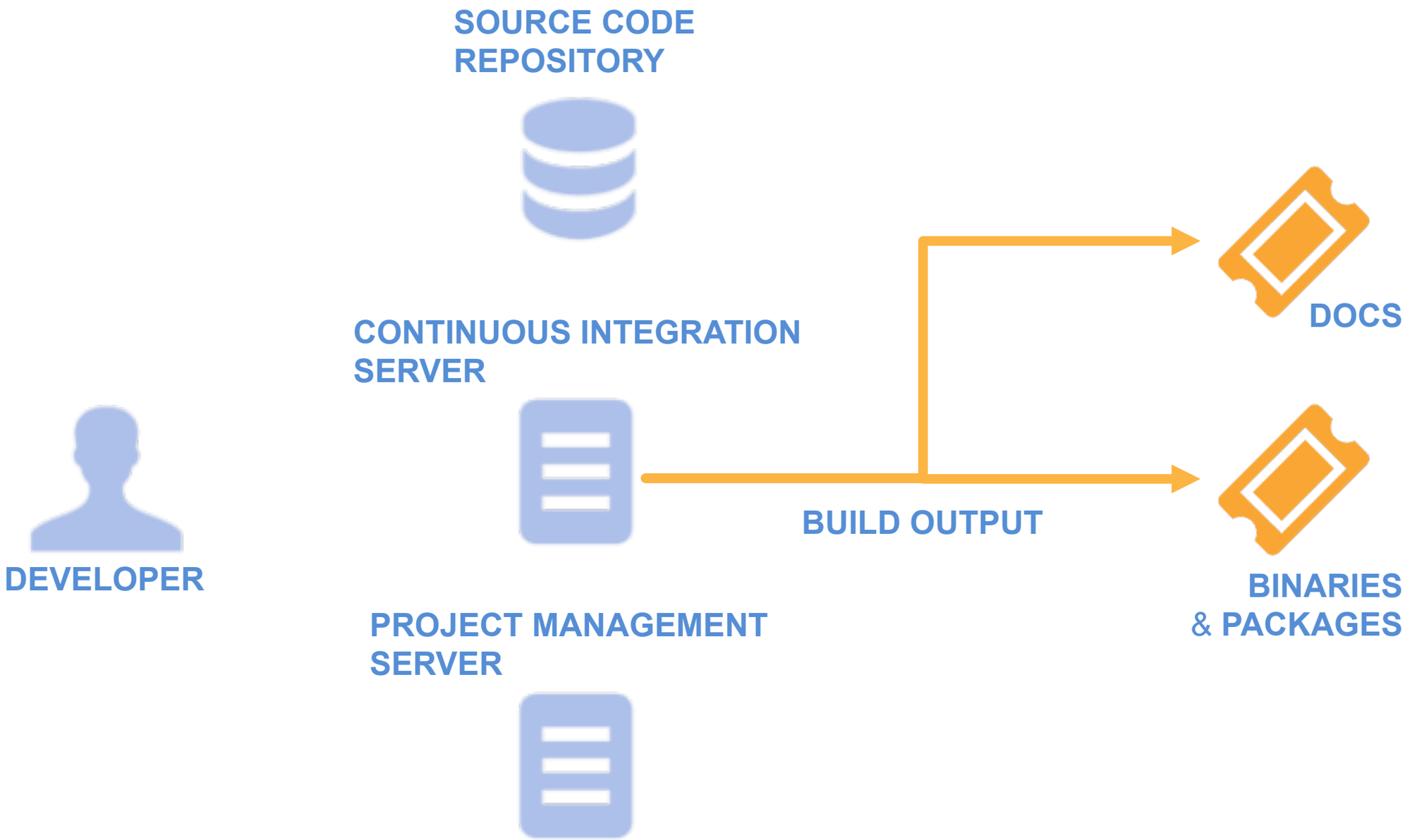
Software Development Models



- Waterfall model
- Careful code design
- Minimal updates

- Agile model
- Continuous integration
- Continuous deployment

Amazon AWS



From DevOps on AWS
Continuous Integration and Deployment Best Practices on AWS

Facebook

Feature: Devops

Development and Deployment at Facebook

Internet companies such as Facebook operate in a "perpetual development" mindset. This means that the website continues to undergo development with no predefined final objective. Also, new developments are deployed in small increments, so engineers can experiment with them to find out what users like the best and determine the costs. To support this, Facebook uses both technical approaches, such as peer review and extensive automated testing, and a culture of personal responsibility.

Google

DOI:10.1145/2854146

Google's monolithic repository provides a common source of truth for tens of thousands of developers around the world.

BY RACHEL POTVIN AND JOSH LEVENBERG

Why Google Stores Billions of Lines of Code in a Single Repository

Google

- Test-Driven Development
  - Separation of requirements and implementation
  - Tests are executable document, which are always synchronized with the code.
  - No source code without tests
- Continuous integration
  - Run tests before any code commit
- Short iterations
  - Fast feedback cycles

Google

- Pair programming
  - Two developers with one machine
  - Fast knowledge transfer
- Code review
  - Second set of fair eyes for every code
- LinkedIn: Continuous deployment
  - Kevin Scott from Google closed the LinkedIn site for 2 months to switch to continuous deployment.

What Is Code Review?

- One stage in a software development workflow
- Gets feedback from your colleagues on your code to submit

- It is **NOT** about
  - Getting comments on overall software designs
  - Planning requirements or features for a product

Code Review Workflow

- Step 1: A software engineer (SWE) implements a **CL (change list)**.
- Step 2: The CL owner sends the CL to his/her colleagues for comments.
- Step 3: The designated colleagues leave comments on the received CL.
- Step 4: The CL owner changes his/her CL based on the received comments and iterates Steps 2 & 3 until the owner gets **LGTM (Looks Good To Me)**.
- Step 5: When all reviewers say **LGTM**, the CL owner submits the CL to the repository.

Code Review Example

spostman Update practice_case1.cpp

1 contributor

18 lines (12 sloc) | 247 Bytes

```cpp
#include <stdio.h>

#include <vector>

using namespace std;

int main() {
  vector<int> vec_numbers;
  vec_numbers.push_back(1);
  vec_numbers.push_back(2);

  for (auto number: vec_numbers) {
    printf("=> %d\n", number);
  }

  return 0;
}
```

Code Review Example

spostman Update practice_case1.cpp

1 contributor

18 lines (12 sloc) | 247 Bytes

```cpp
#include <stdio.h>

#include <vector>

using namespace std;

int main() {
  vector<int> vec_numbers;
  vec_numbers.push_back(1);
  vec_numbers.push_back(2);

  for (auto number: vec_numbers) {
    printf("=> %d\n", number);
  }

  return 0;
}
```

codereview-avatar 4 minutes ago

Please leave comments what this main file is designed for.

Reply...

Code Review Example

spostman Update practice_case1.cpp

1 contributor

18 lines (12 sloc) | 247 Bytes

```cpp
#include <stdio.h>

#include <vector>


using namespace std;

int main() {
  vector<int> vec_numbers;
  vec_numbers.push_back(1);
  vec_numbers.push_back(2);

  for (auto number: vec_numbers) {
    printf("=> %d\n", number);
  }

  return 0;
}
```

codereview-avatar 4 minutes ago

Please leave comments what this main file is designed for.

Reply...

codereview-avatar 2 minutes ago

Consider initializing vec_numbers with the variable declaration like vector vec_numbes = {1, 2};

Code Review Example

spostman Update practice_case1.cpp

1 contributor

18 lines (12 sloc) | 247 Bytes

```
1   #include <stdio.h>
2
3   #include <vector>
4
5   using namespace std;
6
7   int main() {
8     vector<int> vec_numbers;
9     vec_numbers.push_back(1);
10    vec_numbers.push_back(2);
11
12    for (auto number: vec_numbers) {
13      printf("=> %d\n", number);
14    }
15
16    return 0;
17  }
```

codereview-avatar 4 minutes ago

Please leave comments what this main file is designed for.

Reply...

codereview-avatar 2 minutes ago

Consider initializing vec_numbers with the variable declaration like vector vec_numbes = {1, 2};

codereview-avatar 3 minutes ago

Using auto keyword is not good for readability. Specify its type.

Reply...

Code Review Example

spostman Update practice_case1.cpp

1 contributor

18 lines (12 sloc) | 247 Bytes

```cpp
#include <stdio.h>

#include <vector>

using namespace std;

int main() {
  vector<int> vec_numbers;
  vec_numbers.push_back(1);
  vec_numbers.push_back(2);

  for (auto number: vec_numbers) {
    printf("=> %d\n", number);
  }

  return 0;
}
```

spostman Revise the CL to reflect received comments

1 contributor

18 lines (12 sloc) | 329 Bytes

```cpp
// The file is implemented for the code review purpose. The reviewer sh
// points code insistent with our C++ style guideline.

#include <stdio.h>

#include <vector>

using namespace std;

int main() {
  vector<int> vec_numbers = {1, 2};

  for (int number: vec_numbers) {
    printf("=> %d\n", number);
  }
  return 0;
}
```

Why Is Code Review Necessary?

- 당신의 CL을 이해하기 쉽게 만듬.
  - 당신의 동료가 당신의 코드를 이해할 수 있어야 함.
- 당신의 CL에 동료가 남긴 의견으로부터 tips & lessons을 배움.
  - 숙련된 개발자로부터 코딩 스타일과 팁을 배움.
  - 당신의 팀이 공통된 코딩 스타일을 공유할 수 있음.
- 결함을 줄일 수 있음.

Why Is Code Review Necessary?

- 당신의 coding decision에 대한 개발 역사를 보관함.
  - 동료의 의견은 코드 설계와 결정 사항에 대해 이해하는 데 매우 큰 도움이 됨.
  - 새로 온 개발자는 committed logs와 의견으로부터 코드의 구조와 결정 사항을 이해할 수 있음.
- 당신의 코드에 일관적인 코딩 스타일을 유지할 수 있음.
  - 새로 온 개발자가 기존의 코딩 스타일을 따를 수 있도록 도와줌.
  - 일관적인 코딩 스타일은 이후에 코드를 refactoring하거나 디버깅할 때 큰 도움이 됨.

Possible Downsides of Code Review

- 거칠고 무례한 의견 때문에 CL owners may get discouraged.
- 리뷰가 늦어지면 개발 기간이 늦어짐.
- 코드 리뷰를 제대로 하려면 시간이 걸림.
- 경험이 부족한 개발자의 잘못된 CL을 리뷰하느라 숙련된 개발자의 시간이 허비될 수 있음.
- 코드 리뷰를 위해서는 어느 정도 숙력된 개발자가 필요함.

Code Reviews Are Cultural

- Code review process requires cultural supports.
  - Respect your colleagues.
  - Leave constructive feedback.
  - It takes time to grow code review culture.
- Code review helps maintaining healthy and scalable code base.
  - Code review prompts transparency.
  - Code review invites collaboration.
  - Code review gradually elevates the coding standards.
  - Code review is all about teamwork.

Code Review Course

- Code review
- Code review workflow with Git
- Code review etiquette
- Coding style guidelines
- Testing
- Code review case studies
- Code refactoring
- Secure coding
- Clean code

Google Style Guide: C++

Google C++ Style Guide

https://google.github.io/styleguide/cppguide.html

Google C++ Style Guide

Background

C++ is one of the main development languages used by many of Google's open-source projects. As every C++ programmer knows, the language has many powerful features, but this power brings with it complexity, which in turn can make code more bug-prone and harder to read and maintain.

The goal of this guide is to manage this complexity by describing in detail the dos and don'ts of writing C++ code. These rules exist to keep the code base manageable while still allowing coders to use C++ language features productively.

Style, also known as readability, is what we call the conventions that govern our C++ code. The term Style is a bit of a misnomer, since these conventions cover far more than just source file formatting.

Most open-source projects developed by Google conform to the requirements in this guide.

Note that this guide is not a C++ tutorial: we assume that the reader is familiar with the language.

Goals of the Style Guide

Why do we have this document?

Goals

- All submitted code should conform to the guidelines.
  - A few exceptions are allowed.
- They are designed for readers, not for writers!
- Be consistent with your existing code.
- Avoid complicated code that leverages all language features.
- Concede to optimization if necessary.

How to Enforce the Style Guide

- Code review
  - Your reviewers will point out violations of the style guide.
- Automatic tool
  - Code checking tool that runs in background will continuously monitor the style.
- Readability & ownership
  - A readability master for a programming language will point out inconsistencies.
  - A project owner will check the correctness of your code and consistencies.

General Naming Rule

- Name should be descriptive.
  - Avoid abbreviation.
  - Give as descriptive a name as possible.

GOOD

```
int price_count_reader;      // No abbreviation.
int num_errors;              // "num" is a widespread convention.
int num_dns_connections;     // Most people know what "DNS" stands for.
int lstm_size;               // "LSTM" is a common machine learning abbreviation.
```

General Naming Rule

- Name should be descriptive.
  - Avoid abbreviation.
  - Give as descriptive a name as possible.

BAD

```
int n;                      // Meaningless
int nerr;                   // Ambiguous abbreviation
int n_comp_conns;           // Ambiguous abbreviation
int wgc_connections;        // Only your group knows what this stands for.
int pc_reader;              // Lots of things can be abbreviated as "pc".
int cstmr_id;               // Internal letters are deleted.
FooBarRequestInfo fbri;     // Not even a word
```

General Naming Rule (Type Name)

- Have a capital letter for each new word with no underscores(_).

```
// classes and structs
class UrlTable { ... }
class UrlTableTester { ... }
struct UrlTableProperties { ... }

// typedefs
typedef hash_map<UrlTableProperties *, string> PropertiesMap;

// using aliases
using PropertiesMap = hash_map<UrlTableProperties *, string>;

// enums
enum UrlTableErrors { ... }
```

General Naming Rule (Variable Name)

- Variable names are all lowercase, with underscores between words.
- Data members of classes have trailing underscores.

```
string table_name; // OK

class TableInfo {
 private:
  string table_name_;  // OK - underscore at end
  string tablename_;   // OK
  static Pool<TableInfo>* pool_; // OK
};

struct UrlTableProperties {
  string name;
  int num_entries;
  static Pool<UrlTableProperties>* pool;
}
```

Conditionals

- **Prefer no space inside parentheses.**
- if **and** else **keywords belong on separate lines.**
- **The if statement on one line is not allowed when there is an** else **clause.**

```
if (condition) { // no spaces inside parentheses
  ... // 2 space indent.
} else if (...) { // The else goes on the same line as the closing brace.
  ...
}

// Not allowed - The if statement on one line when there is an else clause.
if (x) DoThis();
else DoThat();
```

Function

- Write small and focused functions.
  - A function that performs multiple operations is hard to maintain as a project grows.
- What is your opinion about the code?

```
bool AddTwoForEachItem(const string& filename) {
  ifstream infile(filename);
  if (!infile.is_open())
    return false;

  string line;
  int quantity_sum = 0;
  map<int, double> line_quantity_map;
  while (getline(infile, line)) {
    // Parsing line ...
    line_quantity_map[line_no] = quantity;
    quantity_sum += quantity;
  }

  for (auto kv: line_quantity_map) {
    kv->second = kv->second/quantity_sum;
  }

  // Writing back line_quantity_map

  return true;
}
```

Summary

- Do not need to follow the suggested style.
  - But, it is **important to have a consistent coding style** within a team and even in the company.
- After style guidelines are established, stick to the guidelines.
  - The style guidelines can improve over time.
  - Allow exceptions only if they are critical for your services and performance.

Testing

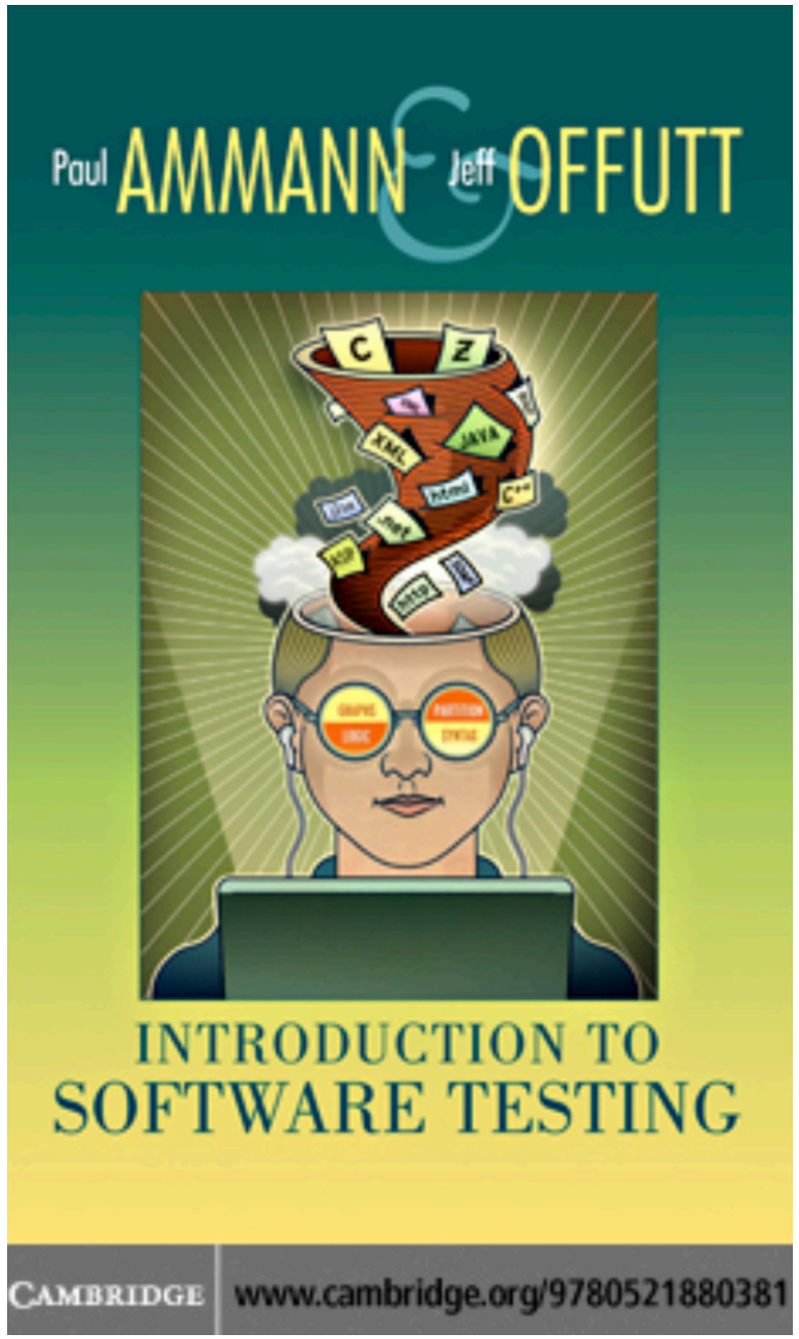
Most material borrowed from Prof. Shin Yoo at KAIST

Why Is Testing So Important?

- Testing Rocks! Debug Sucks!
  - 디버깅은 보통 문제를 찾는 데 엄청 시간이 오래 걸림.
  - 테스팅은 새로 작성한 코드에서도 결함을 검출할 수 있음.
  - 테스팅은 테스트 코드를 필요로 하기 때문에, 유지보수 부담을 줄임.
- Project Scalability
  - 새로 온 개발자도 테스트 코드를 잘 작성해서 프로젝트에 기여할 수 있음.
  - 동료나 외부 기여자에게서 도움을 받기에 가장 적합함.

Testing Types

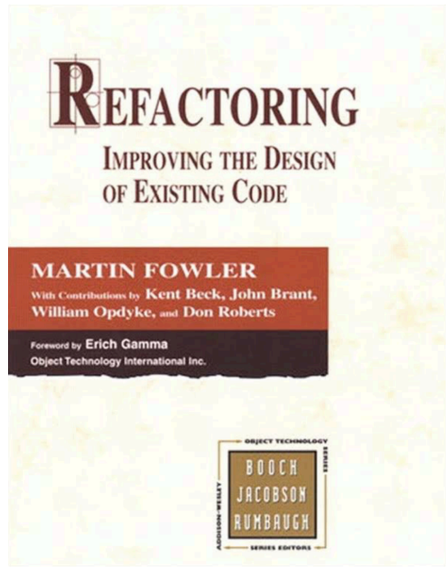
- Unit Testing
  - Test an individual function, interface or class.
  - Mostly validate execution behaviors of a single function.
- Integration Testing
  - Test execution behaviors when two or more modules interoperate.
- Regression Testing
  - Validate whether an existing/developed software performs the same way even after its source code is changed.
- End-to-End (E2E) Testing
  - Test the entire flow of a developing system from start to finish.



From "Introduction to Software Testing" by Paul Ammann & Jeff Offutt

Code Refactoring

From "Refactoring: Improving the Design of Existing Code" by Martin Fowler



Code Refactoring

- Refactoring은 SW의 동작을 바꾸지 않으면서 내부 구조를 개선하는 것, 즉, 코드의 구조를 **잘 정해진 규정대로 수정하는** 기술임.
- SW를 더 이해하기 쉽게 만들고 수정하는 비용을 줄임.
- 꽤 오랫동안 숙련된 개발자들 사이에 전해내려오는 노하우로, 정돈되지 않고 일관적이지 않았음.
- Refactoring is an overhead activity.

Why Refactor?

- SW 설계 개선
  - Refactoring이 없으면 프로그램의 설계는 낡아짐.
  - 설계가 좋지 않은 코드는 보통 같은 일을 하는데 코드가 길고, 같은 일을 여러 곳에서 함.
- 이해하기 쉬움
  - 대부분의 SW 개발 환경에서, 누군가 언젠가는 당신의 코드를 읽어야 할 때가 오기 때문에, 그들을 위해 이해하기 쉬운 코드를 작성해야함.
- 결함 검출할 때도 있음.
- 프로그램 속도 향상
  - 프로그램에 대해 더 잘 이해하기 때문에

A Few Refactorings

- Add Parameter
- Change Association
- Reference to value
- Value to reference
- Collapse hierarchy
- Consolidate conditionals
- Procedures to objects
- Decompose conditional
- Encapsulate collection
- Encapsulate downcast
- Encapsulate field
- Extract class
- Extract interface
- Extract method
- Extract subclass
- Extract superclass
- Form template method
- Hide delegate
- Hide method
- Inline class
- Inline temp
- Introduce assertion
- Introduce explain variable
- Introduce foreign method

Bad Smells in Code

- Duplicated code
- Long method
- Large class
- Long parameter list
- Divergent change
- Shotgun surgery
- Feature envy
- Data clumps
- Primitive obsession
- Switch statements
- Parallel interface hierarchies
- Lazy class
- Speculative generality
- Temporary field
- Message chains
- Middle man
- Inappropriate intimacy
- Incomplete library class
- Data class
- Refused request

KAIST

Sukyoung Ryu

Associate Professor

School of Computing, KAIST

sryu.cs@kaist.ac.kr

+82 42 350 3538

plrg.kaist.ac.kr

카이스트 포용성 위원회

KAIST Committee on Social Inclusion

inclusion.kaist.ac.kr