Advanced Programming in Java
CM3: the Software Development Process

Arthur Bit-Monnot

INSA 4IR

Section 1

Software Life Cycle

Phases of Software Development (outdated)

1. **Requirements Gathering**
   - Identify and document the software's purpose, features, and constraints.
2. **Planning**
   - Create a project plan, including timelines, resources, and budget.
3. **Design**
   - Define the software architecture, user interfaces, and data structures.
4. **Implementation**
   - Write code, develop features, and integrate components.
5. **Testing**
   - Verify that the software meets the specified requirements and is free from defects.
6. **Deployment**
   - Release the software to users or production environments.
7. **Maintenance**
   - Continuously update, enhance, and fix issues in the software.

Software Development

Previous slide: a bit dated

- software was monolithic, produced by single company
- Selling a given version
- Releasing a new version every few years

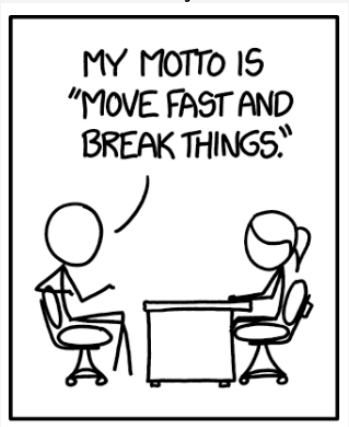
Adapted from existing industrial dev process

Changing factors:

- complexity increase (open source as enabler)
- ease of delivery: paper $\Rightarrow$ CD-ROM $\Rightarrow$ Network

Software development: the Agile Era

Nowadays...

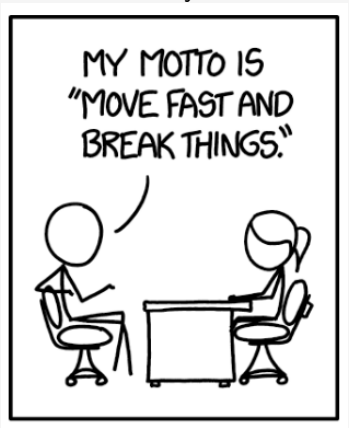


(xkcd/1428)

... software development is much more iterative

Software development: the Agile Era

Nowadays...



(xkcd/1428)

... software development is much more iterative

Specificities of **software** industry:

- Cost of replacing a defective piece in your product:
  - Google vs Volkswagen

Ensuring quality in Increasingly Complex Software

- **decouple** concerns into independent modules
  - large scale: micro-services
  - medium: libraries
  - small packages/class/function
- agree on services provided by each module

Guidelines

1. Identify the responsibility of each module (class/package/library)
   - non overlapping concerns
2. Document the module's contract
   - type system (everything that is public)
   - textual documentation (limitations, perf guarantees, ...)
3. Hide implementation details
   - private / protected methods and classes

Your code provides a Service

Would your threading code keep working if the JVM providers were to:

- Change the name of the `Thread` class?

Your code provides a Service

Would your threading code keep working if the JVM providers were to:

- Change the name of the `Thread` class? $\rightarrow$ No
- Remove a (public) method?

Your code provides a Service

Would your threading code keep working if the JVM providers were to:

- Change the name of the `Thread` class? → No
- Remove a (public) method? → No
- Add a method

Your code provides a Service

Would your threading code keep working if the JVM providers were to:

- Change the name of the `Thread` class? → No
- Remove a (public) method? → No
- Add a method → Yes
- Change the return type of a method

Your code provides a Service

Would your threading code keep working if the JVM providers were to:

- Change the name of the `Thread` class? → No
- Remove a (public) method? → No
- Add a method → Yes
- Change the return type of a method → No (or to something more specific)
- Change a method's parameter name

Your code provides a Service

Would your threading code keep working if the JVM providers were to:

- Change the name of the `Thread` class? → No
- Remove a (public) method? → No
- Add a method → Yes
- Change the return type of a method → No (or to something more specific)
- Change a method's parameter name → Yes (but not in all languages)
- Change the parameter type

Your code provides a Service

Would your threading code keep working if the JVM providers were to:

- Change the name of the `Thread` class? → No
- Remove a (public) method? → No
- Add a method → Yes
- Change the return type of a method → No (or to something more specific)
- Change a method's parameter name → Yes (but not in all languages)
- Change the parameter type → No (or to something more general)
- Change the name of the main thread?

Your code provides a Service

Would your threading code keep working if the JVM providers were to:

- Change the name of the `Thread` class? → No
- Remove a (public) method? → No
- Add a method → Yes
- Change the return type of a method → No (or to something more specific)
- Change a method's parameter name → Yes (but not in all languages)
- Change the parameter type → No (or to something more general)
- Change the name of the main thread? → It should (but does it?)
- Change the implementation of `setName()` to avoid memory allocation?

Your code provides a Service

Would your threading code keep working if the JVM providers were to:

- Change the name of the `Thread` class? → No
- Remove a (public) method? → No
- Add a method → Yes
- Change the return type of a method → No (or to something more specific)
- Change a method's parameter name → Yes (but not in all languages)
- Change the parameter type → No (or to something more general)
- Change the name of the main thread? → It should (but does it?)
- Change the implementation of `setName()` to avoid memory allocation? → Yes

```java
/**
 * Causes the currently executing thread to sleep (temporarily cease
 * execution) for the specified number of milliseconds plus the specified
 * number of nanoseconds, subject to the precision and accuracy of system
 * timers and schedulers. The thread does not lose ownership of any
 * monitors.
 *
 * @param  millis
 *         the length of time to sleep in milliseconds
 * @param  nanos
 *         {@code 0-999999} additional nanoseconds to sleep
 * @throws  IllegalArgumentException
 *          if the value of {@code millis} is negative, or the value of
 *          {@code nanos} is not in the range {@code 0-999999}
 * ...
 */
public static void sleep(long millis, int nanos) throws InterruptedException {

}
```

Setting boundaries

Everything that is visible (i.e. public):

- will be relied upon by your users
- cannot be changed without breaking someone else's code

Everything that is hidden:

- can be freely changed in the future
- only your code is affected

Setting boundaries

Everything that is visible (i.e. public):

- will be relied upon by your users
- cannot be changed without breaking someone else's code

Everything that is hidden:

- can be freely changed in the future
- only your code is affected

Some things are visible but not contractual:

- e.g. java's default thread is called `main`
- should **not** be relied upon

Setting boundaries: Java visibility modifiers

Java visibility modifiers:

- **private**: inside the class
- **default**: inside the package
- **protected**: inside the package and subclasses
- **public**: everywhere (your contract)

Boundaries should be set for each class and package

Section 2

Reliable Code: Error Handling

Reliable Code: Error Handling

```
Path filePath = Path.of("/tmp/secret");
String content = Files.readString(filePath);
System.out.println(content);
```

Reliable Code: Error Handling

```
Path filePath = Path.of("/tmp/secret");
String content = Files.readString(filePath);
System.out.println(content);
```

```
Path filePath = Path.of( first: "/tmp/secret");
String content = Files.readString(filePath);
System.out.println(content);
```

Unhandled exception: java.io.IOException

Student' solution to error handling

```java
Path filePath = Path.of("/tmp/secret");
String content = null;
try {
    content = Files.readString(filePath);
} catch (IOException e) {
}
System.out.println(content);
```

AKA: "Make the Compiler SHUT UP Approach"

The Cases of Error Handling

1. We can recover from the error locally (local handling)
2. The error may be handled by someone else (the caller) with more context information (error propagation)
3. There is no possibility of recovering from the error (crash)

Error Handling: Error propagation

```java
public static String readFile(String filename)
    throws UnreadableFile
{
    Path filePath = Path.of(filename);
    try {
        return Files.readString(filePath);
    } catch (IOException e) {
        // propagate error upwards
        throw new UnreadableFile(filename, e);
    }
}
```

Error Handling: Error propagation

```java
public static String readFile(String filename)
    throws UnreadableFile
{
    Path filePath = Path.of(filename);
    try {
        return Files.readString(filePath);
    } catch (IOException e) {
        // propagate error upwards
        throw new UnreadableFile(filename, e);
    }
}
```

```java
// Error raised when the system is unable
// to read a file
class UnreadableFile extends Exception {
    // File that couldn't be access
    private final String filename;
    // Error thrown by the JVM when attempt
    private final IOException cause;

    UnreadableFile(String filename,
                   IOException cause) {
        this.filename = filename;
        this.cause = cause;
    }

    ...
```

Error Handling: Local Recovery

```
Config loadConfig() {
    String config = readFile("~/.conf");
    return new Config(config);
}
```

Error Handling: Local Recovery

```
static String DEFAULT_CONFIG = "address=localhost; port=80";

static Config loadConfig() {
    String config = null;
    try {
        config = readFile("~/.conf");
    } catch (UnreadableFile e) {
        //
        System.out.println("Could not read config file "+e.filename +
                "\n  Caused by "+ e.cause);
        System.out.println("Loading default configuration");
        config = DEFAULT_CONFIG;
    }
    return new Config(config);
}
```

Error Handling: Unrecoverable Error

```
public static void main(String[] args) {
  Config config = loadConfig();
  startHttpServer(config);
  ...
}
```

Error Handling: Unrecoverable Error

```java
public static void main(String[] args) {
  Config config = loadConfig();
  try {
      startHttpServer(config);
  } catch (Exception e) {
      // log error
      e.printStackTrace();
      System.err.println("Could not start HTTP Server");

      // Terminate program with error code
      System.exit(1);
  }
}
```

Error Handling: Guidelines

How should I handle the error?

1. Recover locally (retry, default, ignore, ...)
2. Delegate responsibility to caller (rethrow)
3. Take the responsibility of terminating (`System.exit()`, `throw new RuntimeException()`)

Who should I communicate the error to?

1. Developer / Technical User (logging, stack traces)
2. End User (standard output, dialog)

Section 3

Reliable Code: Tests

Reliable Code: Tests

Tests ensure that:

- the code is correct
- fullfills its contract

Main approach in students population:

- try the most obvious case in the main function

Limitations:

- not exhaustive
- not durable

Software Testing

Software testing is the process of building a **test suite** that:

- ensures that the code fulfills its contract
- can be automatically run
- will stay throughout the code's lifecycle

Relies on software testing libraries

- $>$ JUnit in Java (most popular option)

The Bare Minimum: Gradual Testing

You already write tests, don't throw them away!

```java
// PROHIBITED: make your test
// automatable and durable
public static void main(String[] args) {
    int result = add(4, 5);
    System.out.println(result);
}
```

```java
@Test
public void testAdd() {
    int result = add(4, 5);
    assert result == 9;
}
```

Rule of thumb:

- always write testing code in a dedicated method that you will keep
- change your manual inspections (prints) with automatic ones (asserts)

What is Test Driven Development (TDD) ?

Test-Driven Development is a software development process that emphasizes writing tests before writing the actual code.

TDD Workflow Steps

1. Write a **failing** test for a small unit of functionality.
2. Write the **minimum** amount of code to make the test **pass**.
3. **Refactor** the code for simplicity and maintainability.
4. Repeat the process for the next unit of functionality.

Benefits of Tests / TDD

- **Improved Code Quality:** helps catch and fix defects early in the development process.
- **Sustainable Development:** It promotes smaller, manageable iterations (TDD).
- **Documentation:** Tests serve as living documentation for your code.
- **Confidence:** Developers have confidence in the code's correctness in the long run.

Conclusion

Reliable Software...

- clearly states its promises (public API / Contract)
- has slack to evolve (hidden implementation details)
- correctly handles corner cases (error handling)
- can always be shown to fulfill its contract through tests