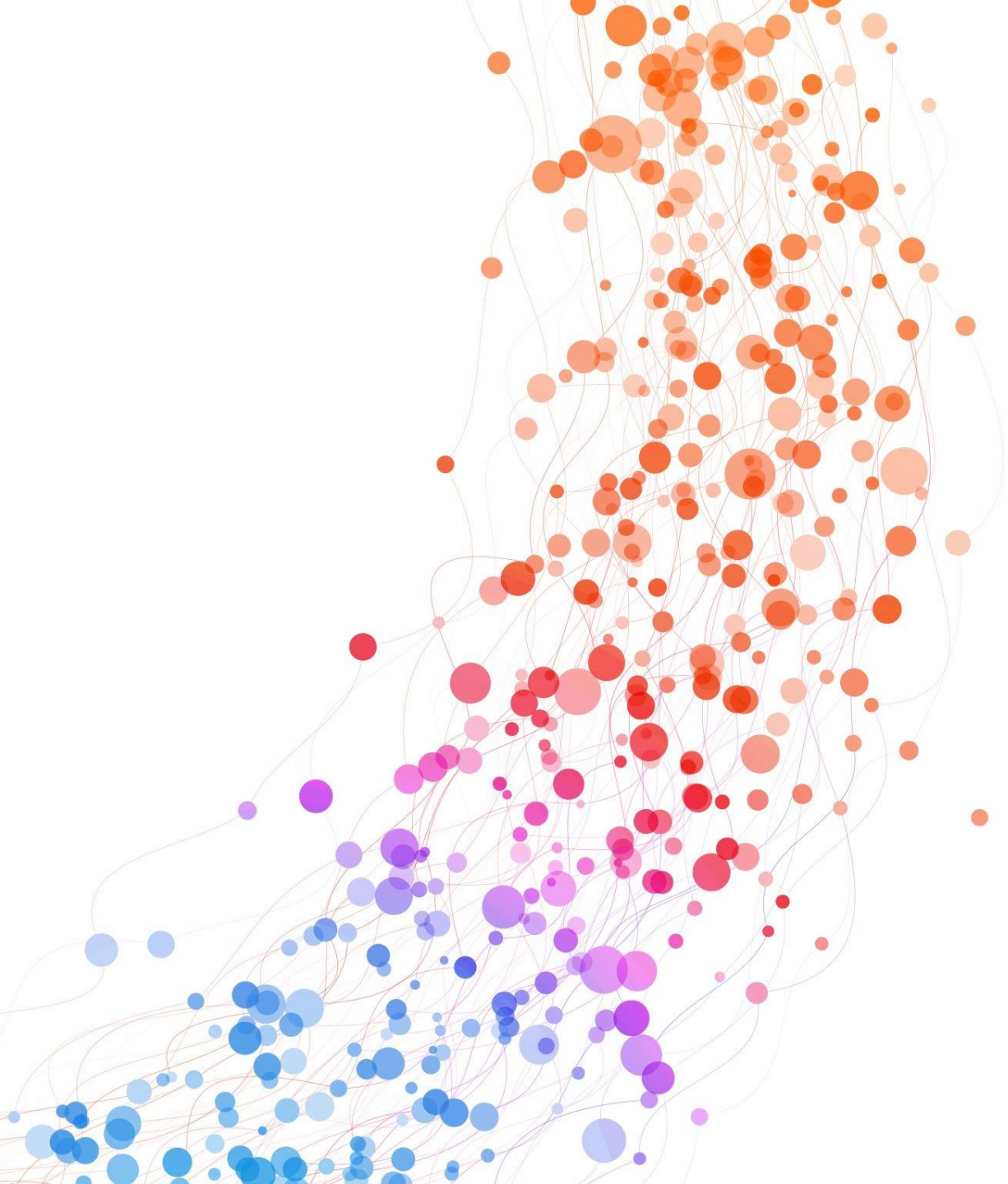
Software Engineering COMP1035

Lecture 10

Implementation & Debug

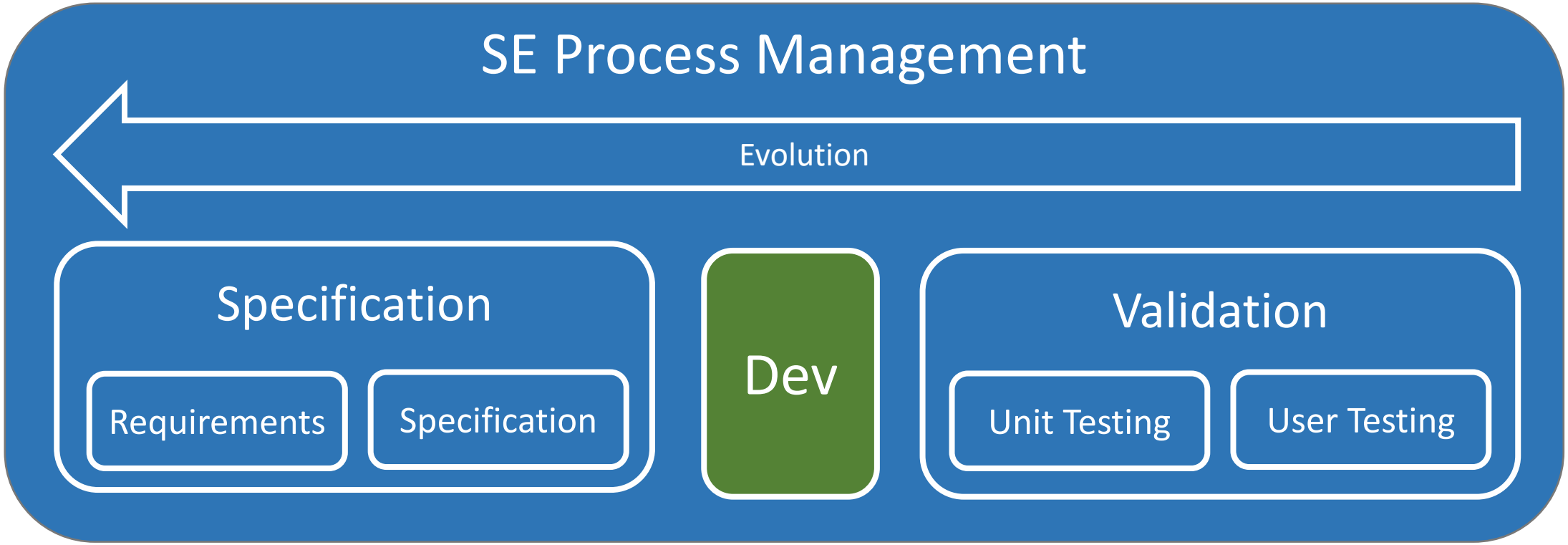
Today's Objectives:

- Software Engineering Dev is a social concern.
  - Coding for, alongside, and even with other people.
- Well Implemented means "easy to maintain".
  - Coding to certain standards.
  - Coding according to certain strategies .
- Debugging is a social, multi-faceted skill too.

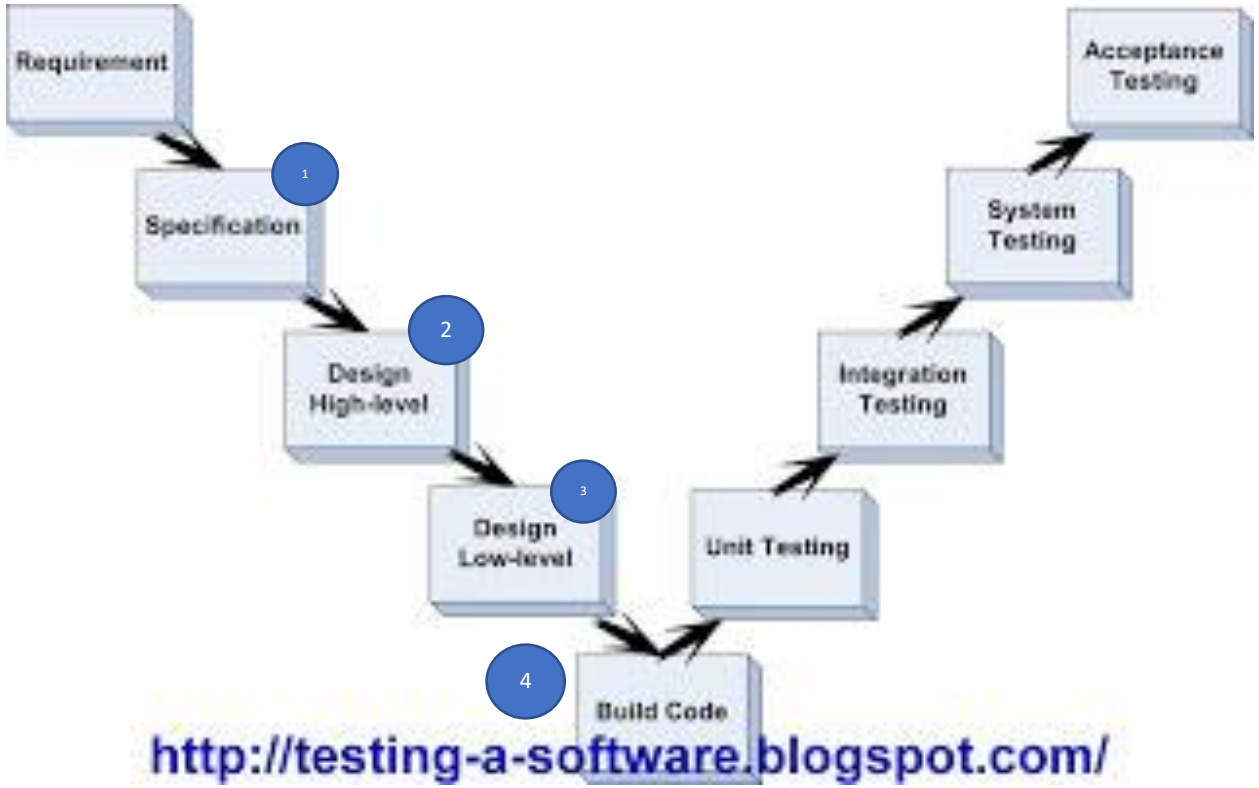


Where We Are In the Process

Keeping Track of SE Module

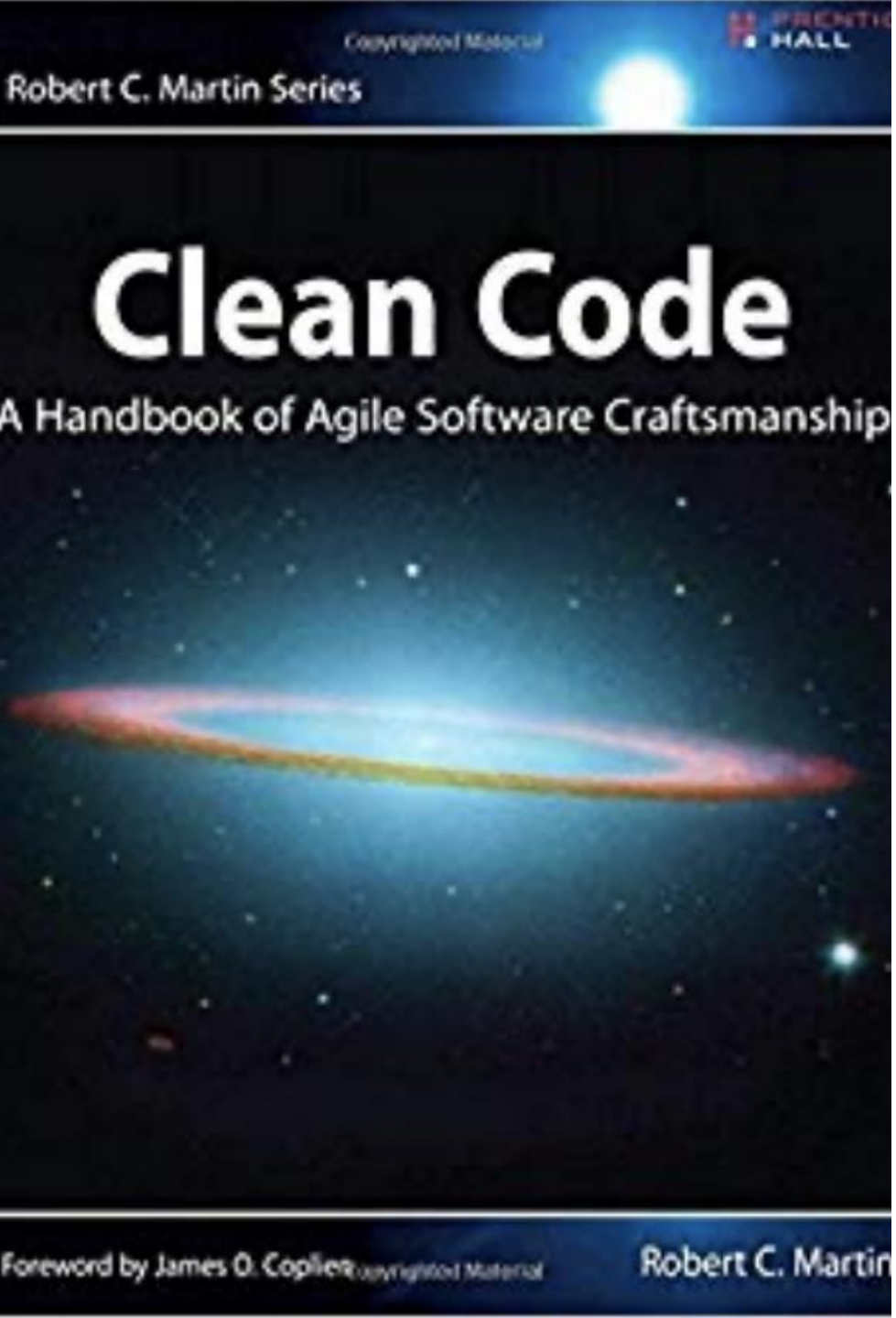


Keeping Track of SE Module



What are SE concerns for Dev?

- You are being taught to write code in many other modules.
- Section II and III of SE book are different types of Dev concerns.
  - Secure, reliable, dependable, distributed, embedded, real time, etc .
  - These also come out of different modules.
  - We are not going to study these chapters in detail (no need for you to read them all).
- We need to talk about how a team does SE together.

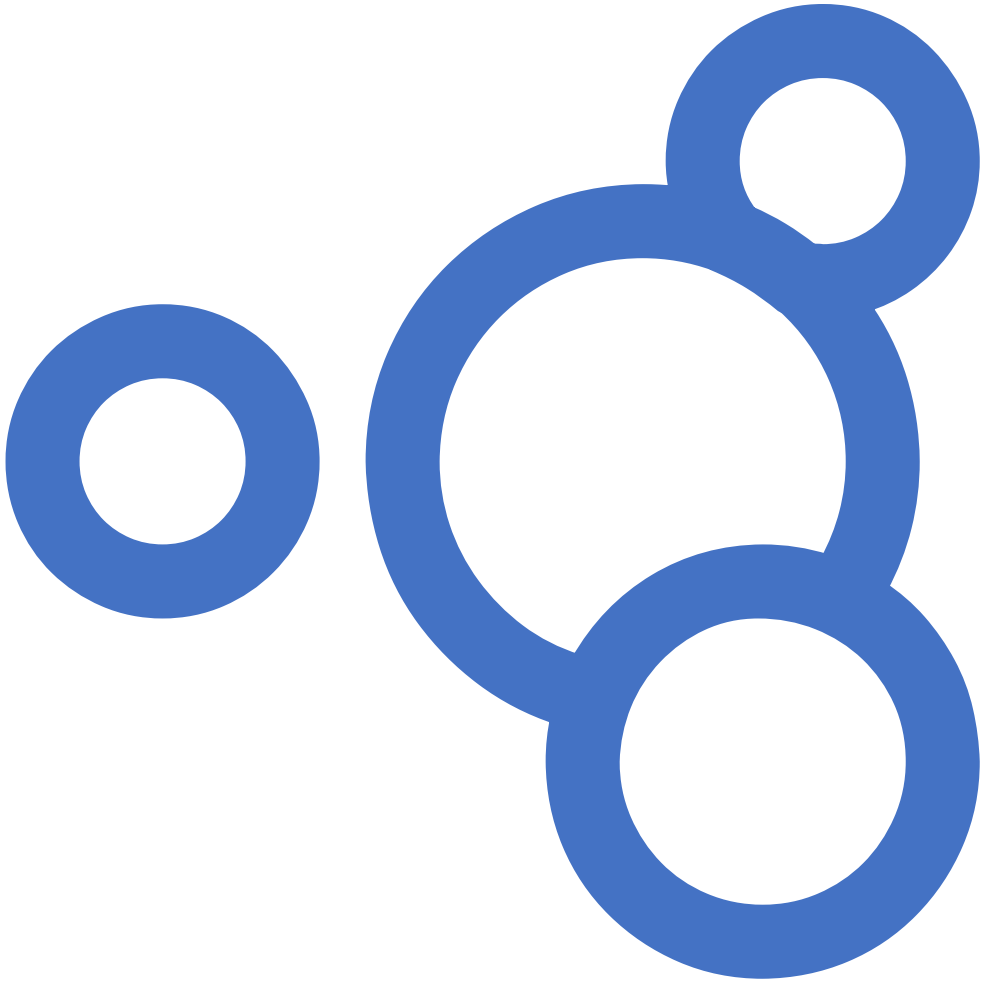


"Writing clean code is what you must do to call yourself a Professional"

A great book for developers

- Section 1 is really inspiring and relevant.
- Section 2-7 relevant to you now.
- Naming, comments, functions, formatting, error handling, etc.
- Sections 8 & 10 relate to PGP : Java
- Section 9 relates to TDD (next lecture)
- Section 15 is JUnit - kind of hard to understand in my opinion
- Sections 11-16 will become relevant after 2nd year
- Section 17 is list of 'smelly code' issues

Social Dev Perspectives



You Are Not Working Alone

- The incorrect scenarios:
  - I did the requirements.
  - I did the specifications.
  - I did the design.
  - I am now building what I think.
  - I am going to test it.
  - I am going to release it online and maintain it.
  - *This is not the situation we are concerned with*.

Scenario 1 - The Developer

- You've been asked to build a specific class.
  - Mostly easy but with one complex function.
  - Now all you need to know is – how is the class supposed to work?
- If life was going to be easy, you should now have:
  - A spec document, class diagram etc.
  - Perhaps a sequence diagram explaining **how your code is used**.
- Later, you must:
  - Deliver the code to the team for their approval.
  - Prove that your code meets all the specs.

Scenario 2 - The Dev Manager

- You have 20 developers who can build this system.
  - You have all the specs, and requirements etc.
  - You want one solid piece of software that doesn't look like it was built by 20 different people.
- You must **make sure everyone is talking to each other and helping each other**, not themselves.
- Later, you must prove that the whole system is finished.
- You must hand over documentation at the end too.

Scenario 3 - The Integration Team

- You are part of a special team of 4 to bring it all together.
  - You must integrate the classes and build the main system.
  - You must be able to **read other people code**.

- It's a large job, so there are a few of you on the single task.
  - You must **coordinate** between yourselves,
  - Comprehend other people's code,
  - Send code back to people if it is wrong.

Setting the Right Scene

- If you oversee some code, its not a one-time task.
  - You might **get it back after other people 'use it'**.
- Its unlikely that you are the only person who must be able to read and understand your code.
- You might have to **fix other people's code**.
- You need some sort of coordination between people.
- Documenting code is a team effort too.
- "Finishing" is a matter of other people's judgements.
- "Correctness" is a matter of other people's judgements.
- "It does everything the spec I received said it should" is probably not good enough.

Key Concerns for SE Dev

- Expected Reading: How Google does these things (and more)
  - Conventions for ‘Good Code’.
  - Strategy for approaching your own code.
  - Techniques for ‘Coding in a team’.
    - Alongside with others,
    - Including techniques for Debugging.
    - Techniques for Coding with others

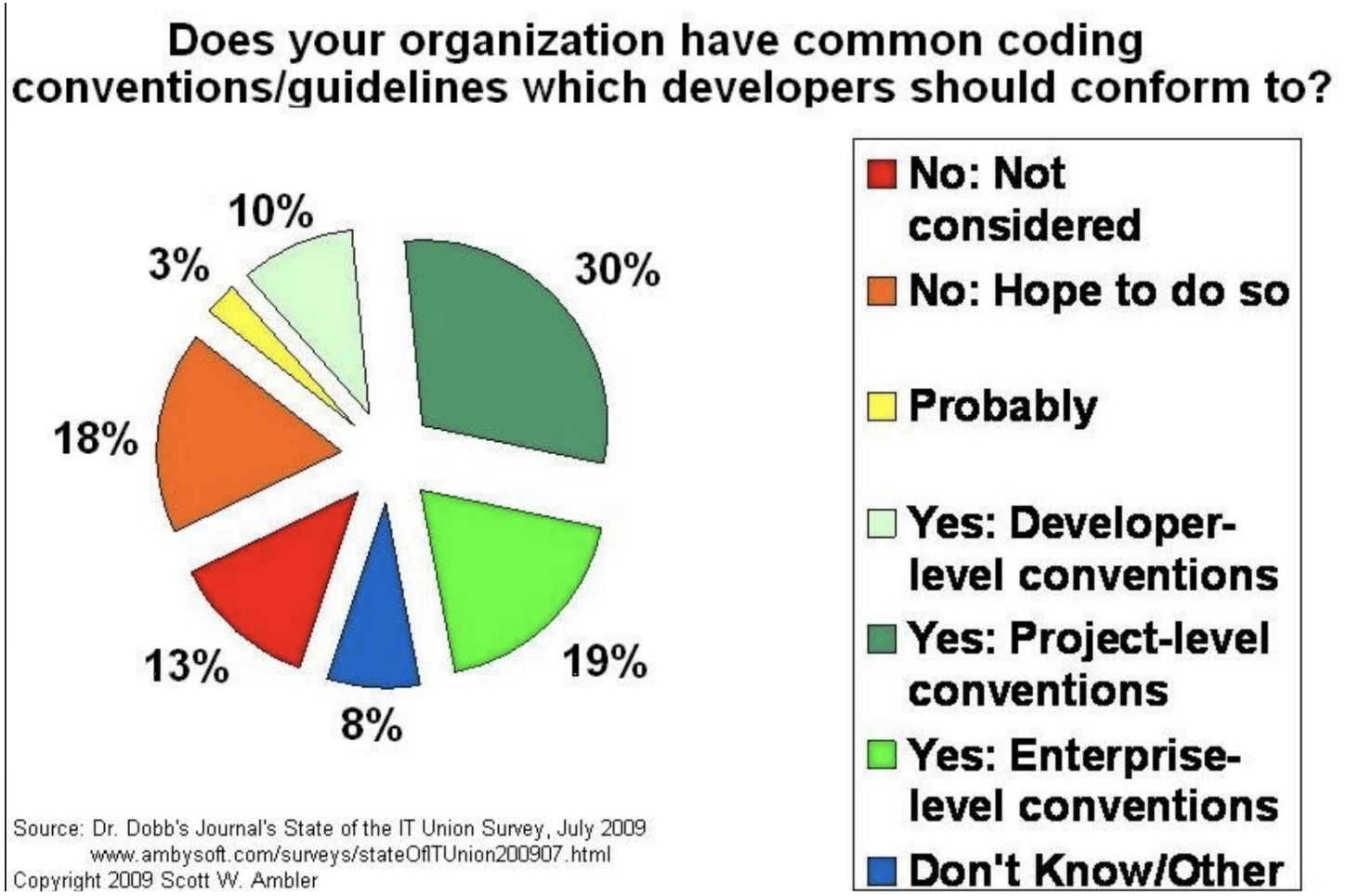
Coding Conventions

Coding Conventions

- Aims:
  - Making code easier for everyone to understand.
  - Producing software that’s built by 1 team - not 20 individuals.
  - Setting quality standards.

- Methods (Consistency):
  - We will all name variables like this.
  - We will all write comments like this.
  - We will organize the contents of a class like this.
  - We will name files like this etc.

Coding Conventions



Coding Conventions

Programming Wisdom
@CodeWisdom

"Any fool can write code that a computer can understand. Good programmers write code that humans can understand." - Martin Fowler

17:29 · 22/12/2018 · Buffer

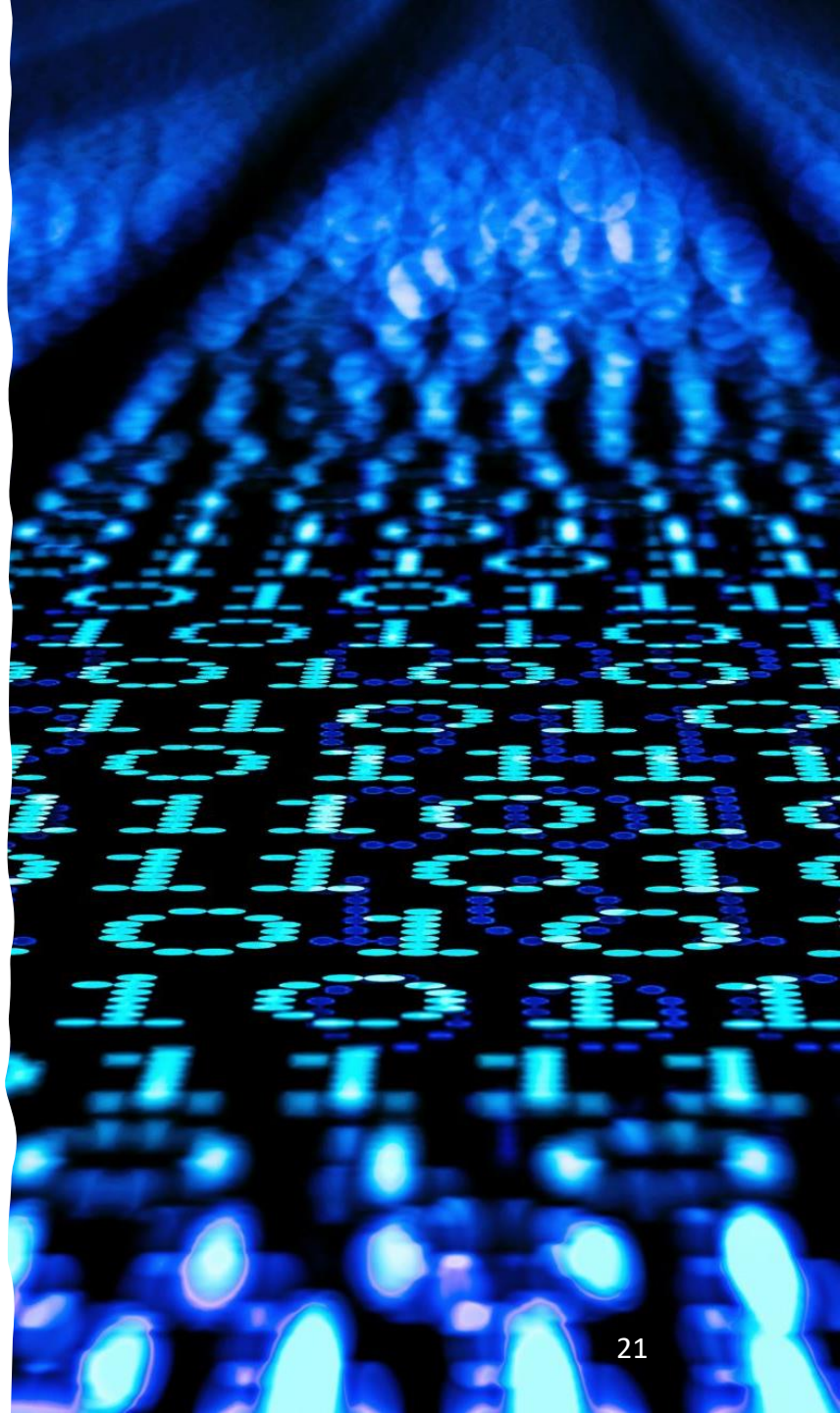
Java Coding Conventions

- From the top link - Why have Coding Conventions?
  - 80% of the lifetime cost of a piece of software goes to **maintenance**.
  - **Hardly any software is maintained for its whole life by the original author**.
  - Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
  - If you ship your source code as a product, you need to **make sure it is as well packaged and clean** as any other product you create.
  - Especially important in open-source situations.
  - When contributions to a project can come from an open community.
  - The community often sets standards and accepts code of an appropriate standard.
  - Google recommends “Appoint at least two team members to ensure the workflow is adhered to” https://opensource.google.com/docs/releasing/contributions/

Google's Internal Code Conventions

There are also Google **style guides** for each language, to ensure that code all across the company is written with similar style, layout, naming conventions, etc. In addition there is a company-wide **readability** training process, whereby experienced engineers who care about code readability train other engineers in how to write readable, idiomatic code in a particular language, by reviewing a substantial change or series of changes until the reviewer is satisfied that the author knows how to write readable code in that language. Each change that adds non-trivial new code in a particular language must be approved by someone who has passed this "readability" training process in that language.

Good Practices for Code



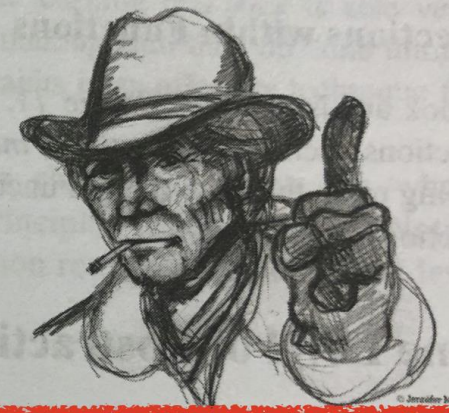
Policy Guidelines

- 8 guidelines for dependable systems:
  1) Control the visibility of information in a program.
  2) Check all inputs for validity.
  3) Provide a handler for all exceptions.
  4) Minimize the use of error-prone constructs.
  5) Provide restart capabilities.
  6) Check array bounds.
  7) Include timeouts when calling external components.
  8) Name all constants that represent real-world values.

Clean Code

Do One Thing

It should be very clear that Listing 3-1 is doing lots more than one thing. It's creating buffers, fetching pages, searching for inherited pages, rendering paths, appending arcane strings, and generating HTML, among other things. Listing 3-1 is very busy doing lots of different things. On the other hand, Listing 3-3 is doing one simple thing. It's including setups and teardowns into test pages.



The following advice has appeared in one form or another for 30 years or more.

FUNCTIONS SHOULD DO ONE THING. THEY SHOULD DO IT WELL. THEY SHOULD DO IT ONLY.

Strategies for Approaching Code

Strategies for Team Code

- These tend to be for setting an approach that breeds code quality.
- Lots of different Higher Level team strategies.
  - Model Driven Development.
  - Feature Driven Development.
- Two very **popular strategies**.
  - Test Driven Development (TDD)
  - Comment Driven Development etc.

Test-Driven Development

1. Before you write code - write the automatic tests that it must pass.
2. Then you write your code.
3. You are done when it passes the tests.

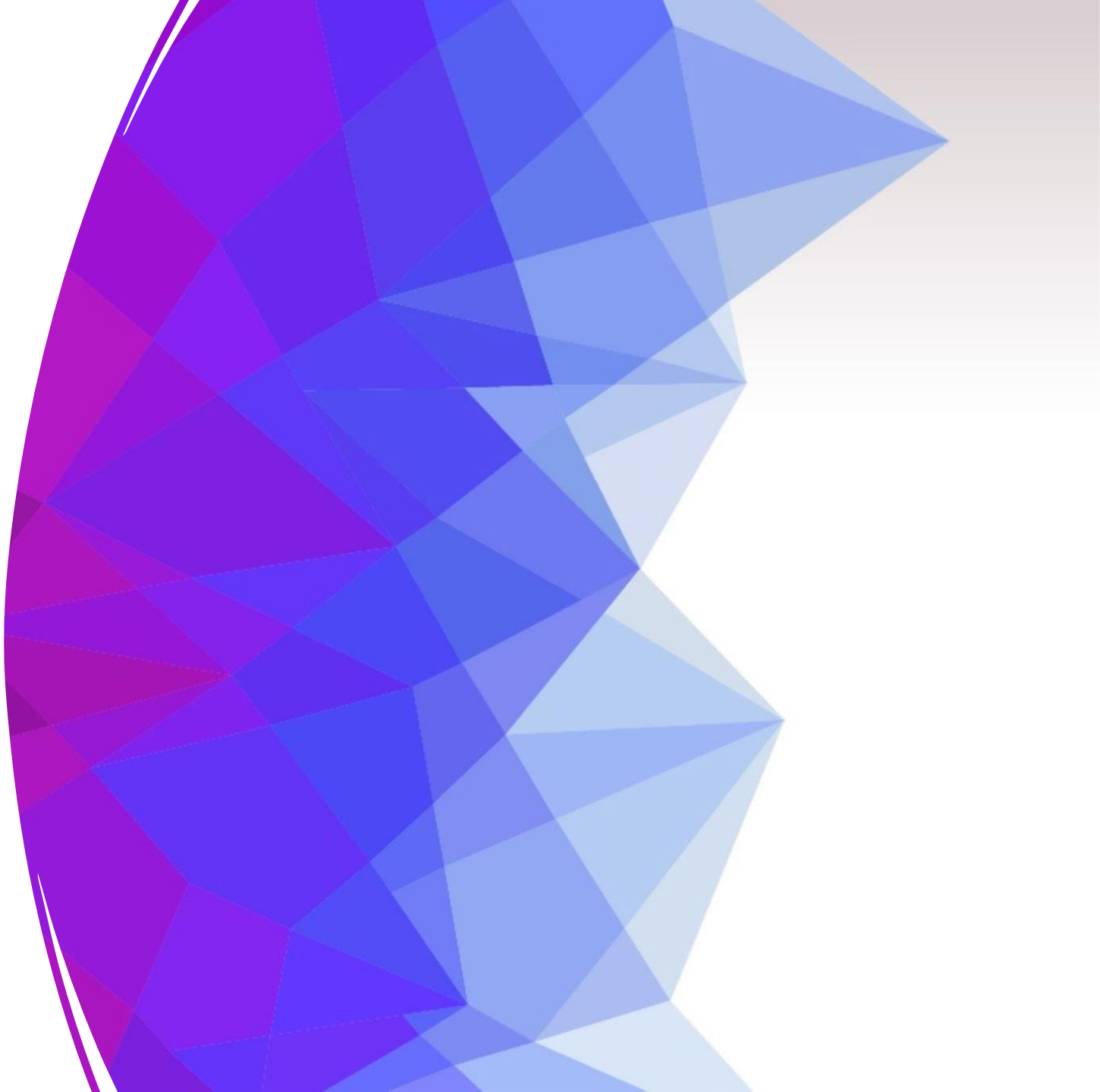
- We will do more of this in subsequent lectures.
  - A team approach might be to write tests for someone else's code,
  - Then swap,
  - This increases accountability within a team,
  - Shares understanding / comprehension of the specs received.

Comment-Driven Development

- Strategy: Write the comments first, then you populate the comments with code.
- Benefits:
  - You must properly comment the code anyway.
  - You can design your code before building it.
  - Figure out the logic before, separately from the code.

- You can do this with the whole codebase.
  - It is simpler to refactor comments than to refactor code.
  - Again - one person can plan the code.
  - Then swap and let other people code.

Coding - Alongside Other People



Bug Reporting/Fixing

- When writing code that uses other code, you might identify new functions it needs, or problems with its output.

- Projects need a mechanism for coders telling other coders about a problem in their code.
  - Keeping track of things that need fixing.
  - Having traceable.
  - Documentation of testing.

- Discovering bugs in code is a multi-coder challenge.
- (The principle is: once you've finished code - you'll get it back)

Bug Reporting

- Someone calls you up, reporting a bug, your first task is to try and reproduce it.
- You may need to ask:
  - What exactly were they doing when it happened.
  - What was the immediate action before the bug was found.
  - What the ‘wrong outcome’ was.
  - The nature of the bug.
- You can not easily judge the type of defect without this information.
- With any luck, a good team member has tried to figure out the problem before they call you.

Bug Trackers

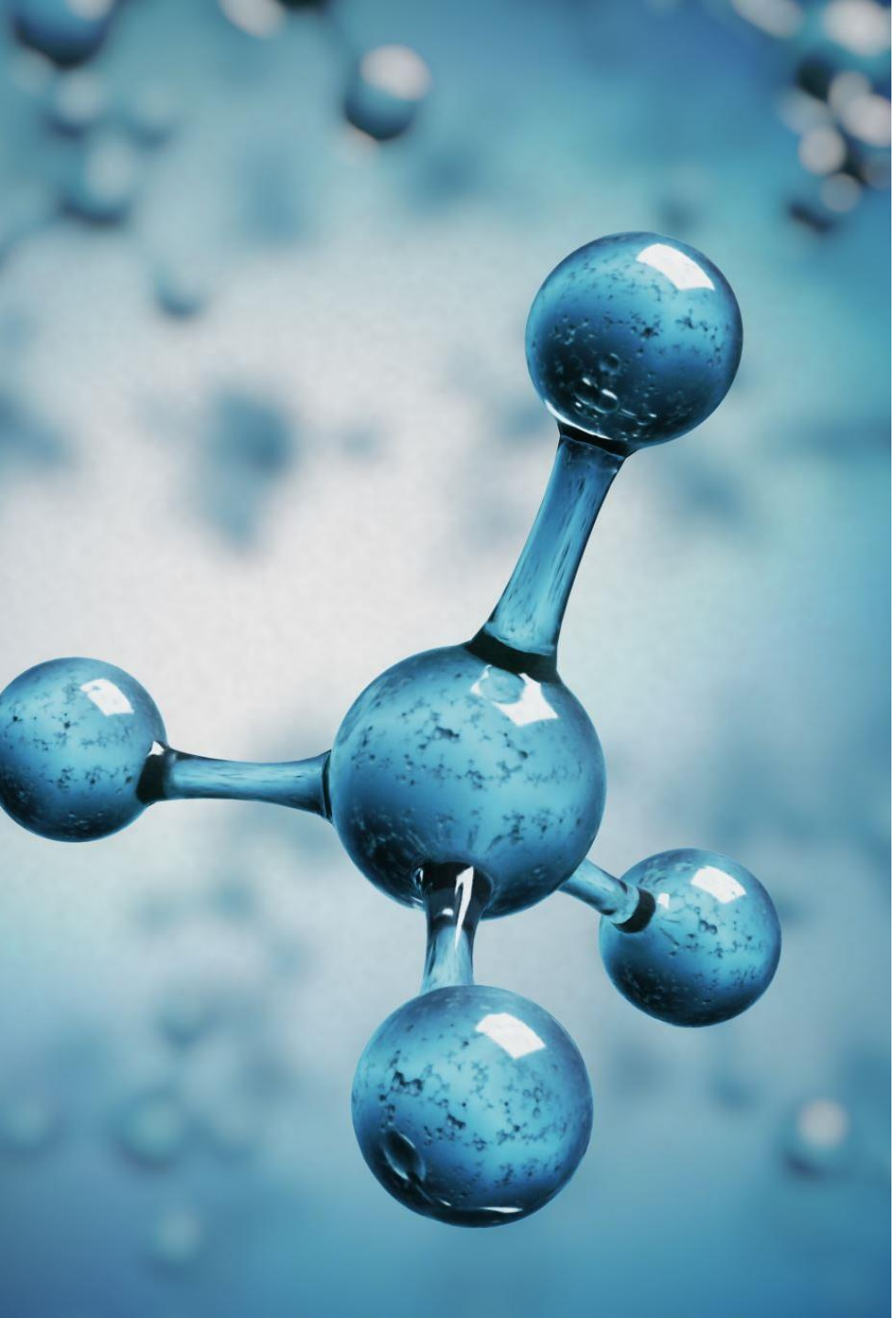
- Many companies use a bug tracker.
  - These are very useful for any project.
  - Especially if the aim is to finish writing code, before fixing bugs that other people have found.

- Lots of bug trackers exist:
  - bugzilla.org is an open-source web bug tracker from Mozilla.

- These trackers record data to help you judge the defect.

BugZilla

Fig. A web view of BugZilla from real product



Chromium

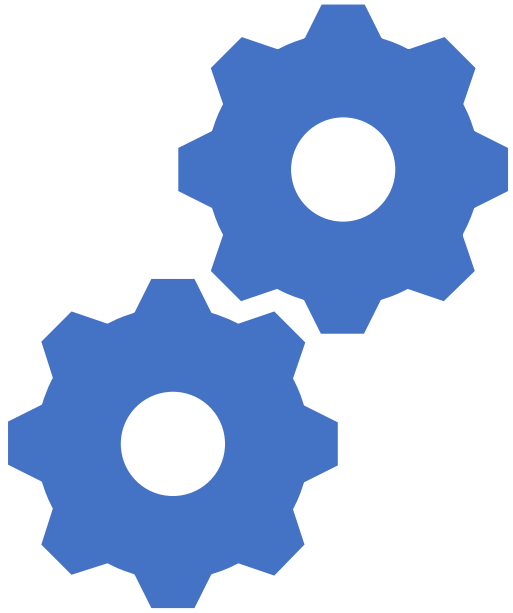
- Its the open-source community for developing Google Chrome.
- Google Chrome = Chromium + proprietary components.

- Report Bugs
  - http://dev.chromium.org/for-testers/bug-reporting-guidelines
- View current open bugs
  - https://bugs.chromium.org/p/chromium/issues/list

Chromium Bug Reporting Guidelines

- Bug reporting guidelines
  - If you're a web developer, see How to file a good bug.
  - Make sure the bug is verified with the latest Chromium (or Chrome canary) build.
  - If it's one of the following bug types, please provide some further information.
    - Web site compatibility problem: Please provide a URL to replicate the issue.
    - Hanging tab: See Reporting hanging tab bugs.
    - Crash: See Reporting crash bugs.
  - Provide a high-level problem description.
  - Mention detailed steps to replicate the issue.
  - Include the expected behavior.
  - Verify the bug in other browsers and provide the information.
  - Include screenshots, if they might help.
  - If a bug can be reduced to a simplified test, then create a simplified test and attach it to the bug.
  - Additional Bug Reporting Guidelines for the Mac & Linux builds.
  - Additional Guidelines for Reporting Security Bugs.

Debugging



Debugging

- Is a social issue too:
  - Communicating with other people about their code.
- "The process of **locating problems with the code and changing the program to fix these problems**".
  - It is an important skill for a professional coder.
  - You want to be capable, not dependent on others.
- Bugs can be hard to find:
  - Especially if its only under certain conditions;
  - Especially as the size of code grows.
- But there are strategies for finding bugs.

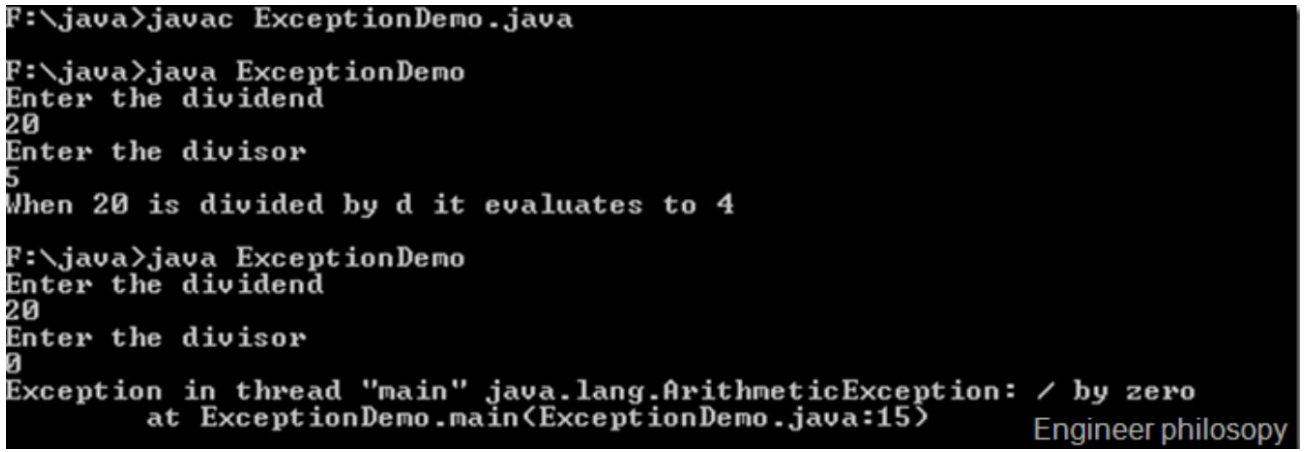
Debugging is Not Testing

- Unit Testing: Aims to establish the presence of bugs.
- Debugging: To locate and remove the cause of bugs.
- “There is no simple method for program debugging”.
- “Skilled debuggers look for patterns in the test output where the defect is exhibited and use their knowledge of the type of defect, the output pattern, the programming language and the programming process to locate the defect.”

Easy Debugging

- You are writing a single script that does not integrate with any other code.

- The call stack tells you exactly where the code broke.



- Fine - if you are getting an exception.
- http://www.cprogramto.com/example-exception-handling-try-catch-java/

Harder (Social) Debugging

- The code runs fine - its just giving you the wrong answer.

- The code you are working on is using other code, written by other developers.

- This is hard because you do not know where the defect is:
  - You must use the symptoms to narrow this down.
  - Use your judgement to look in the right places.
  - Perhaps in your code
  - Perhaps what other code does with what you gave it.

Debugging - IDE Feature

```
/* (non-Javadoc)
 * @see org.eclipse.equinox.app.IApplication#start(org.eclipse.equ
 */
public Object start(IApplicationContext context) {
    Display display = PlatformUI.createDisplay();
    try {
        int returnCode = PlatformUI.createAndRunWorkbench(display,
        if (returnCode == PlatformUI.RETURN_RESTART) {
            return IApplication.EXIT_RESTART;
        }
```

- Breakpoints are points where the software will pause, so that you can check the value of each data point.
- You can debug one step at a time to see what's happening.
- You can step into the code you call (other peoples code), and back again.

Debugging - IDE Feature

- Variable Inspection
  - Hovering over variables when the code is paused (after a break point).
- Trace Tables
  - Ask the system to keep track of certain variables in a table, to watch how they change as you step.
  - You can do this manually on paper too! Its very helpful.
  - Print the code, make a table on the side, follow the code step by step updating the table.
- Exception-based Breakpoints.
  - Pause when you get an error - so you can inspect everything.

Debugging - Strategy Level

- All these prior efforts are fine, but
  - Where do you want to set a breakpoint?
  - What do you trace?
  - What do you printout?

- Strategies help to do this
  - Be systematic.
  - Find ways to exclude possible causes.
  - Keep a log of what you've excluded.

Paired Coding

- In between team-strategy and code-strategy:
  - One person thinks/checks while the other person codes.
  - Take turns to write functions and change pairs for different tasks.

- Benefits - Problem-solving & coding-well can be separated:
  - No single person has ownership, knows how code works.
  - It has an informal review process built in.
  - Good practice is informally spread across the team.

- Has been shown not to take more than two people coding.
  - Especially enter testing/fixing phase.

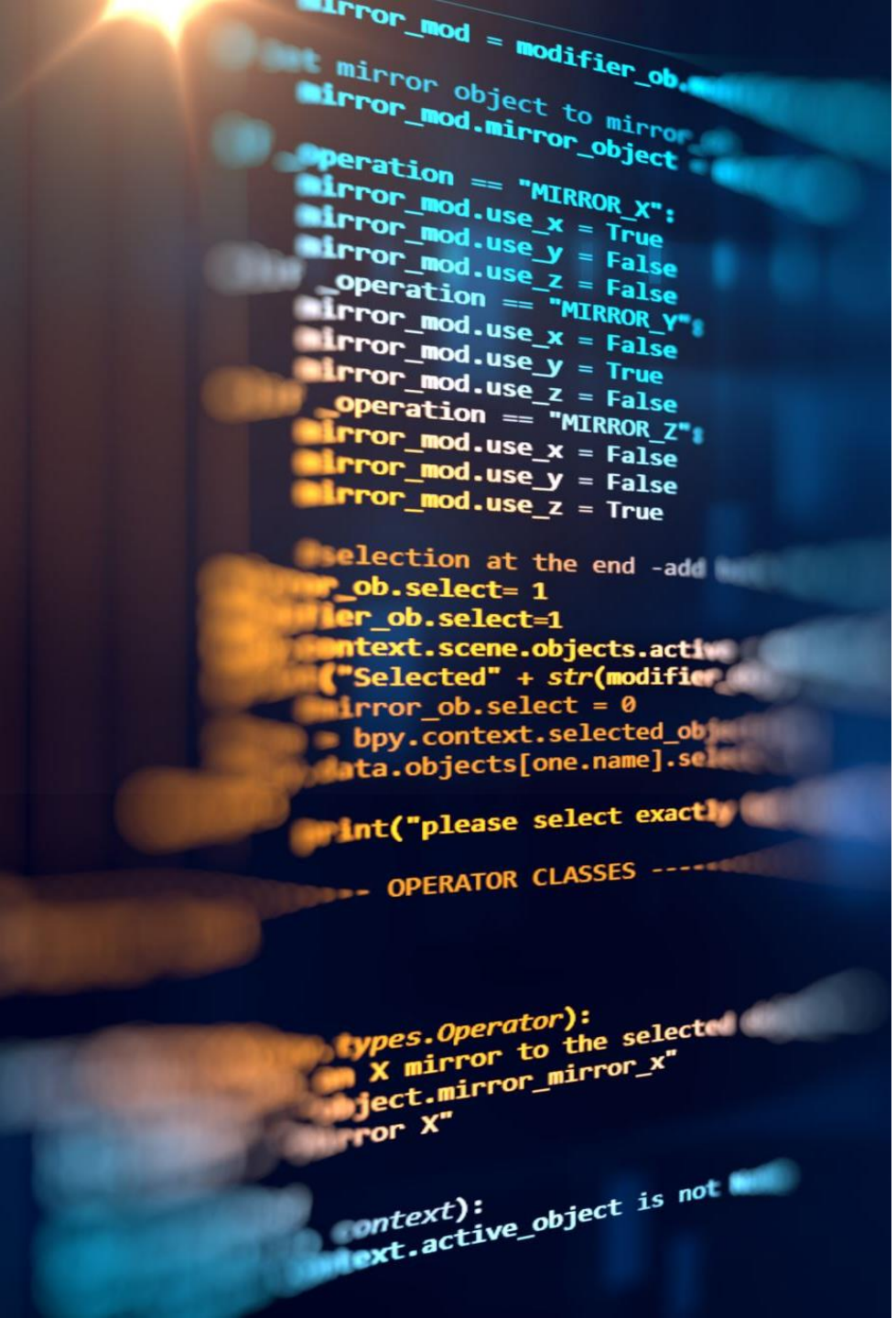
Paired Coding

- It brings together everything I've said today, in one approach
  - inspections as you go
  - its better than talking to a rubber duck ?
  - immediate bug reporting
  - separation of planning and coding
  - learning good coding practice from other people
  - knowing code is readable as you produce it

- Consequently, its an extremely popular SE Team approach

Optional Reading

- https://github.com/Kristories/awesome-guidelines
- http://www.oracle.com/technetwork/java/codeconvtoc-136057.html
- https://google.github.io/styleguide/javaguide.html



Summary

1. Software Engineering Dev is a social concern.
   - Coding for, alongside, and even with other people.
2. Well Implemented means “easy to maintain”.
   - Coding to certain standards.
   - Coding according to certain strategies .
3. Debugging is a social, multi-faceted skill too.

THANK
YOU