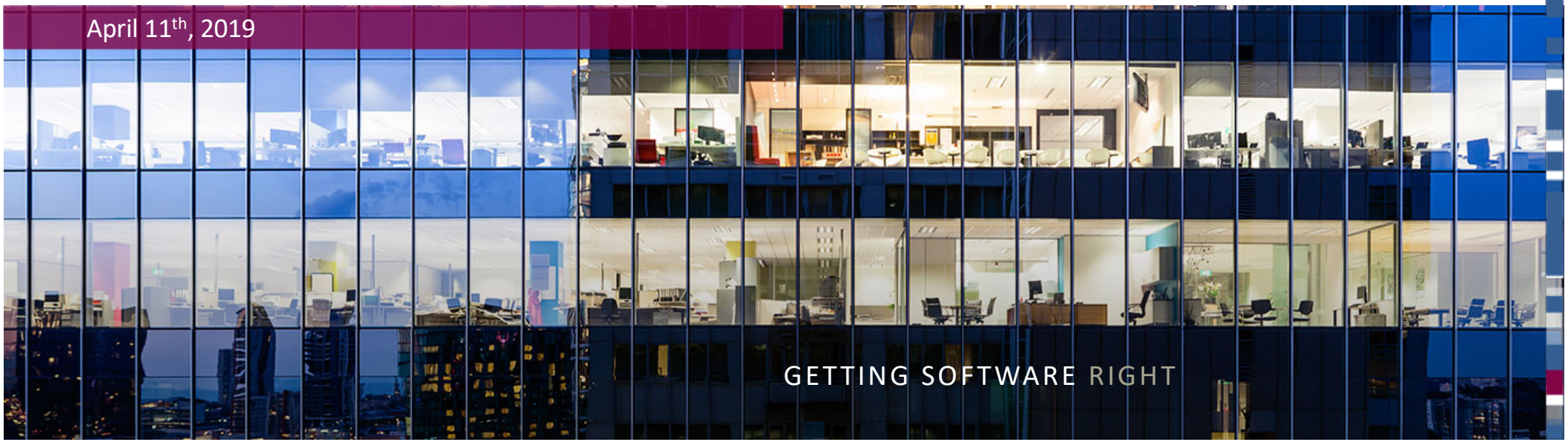
Software Improvement Group

Preparing for the Judges: How mature is my Hackathon prototype?

Great software doesn't just happen

April 11th, 2019



2018 vs 2019

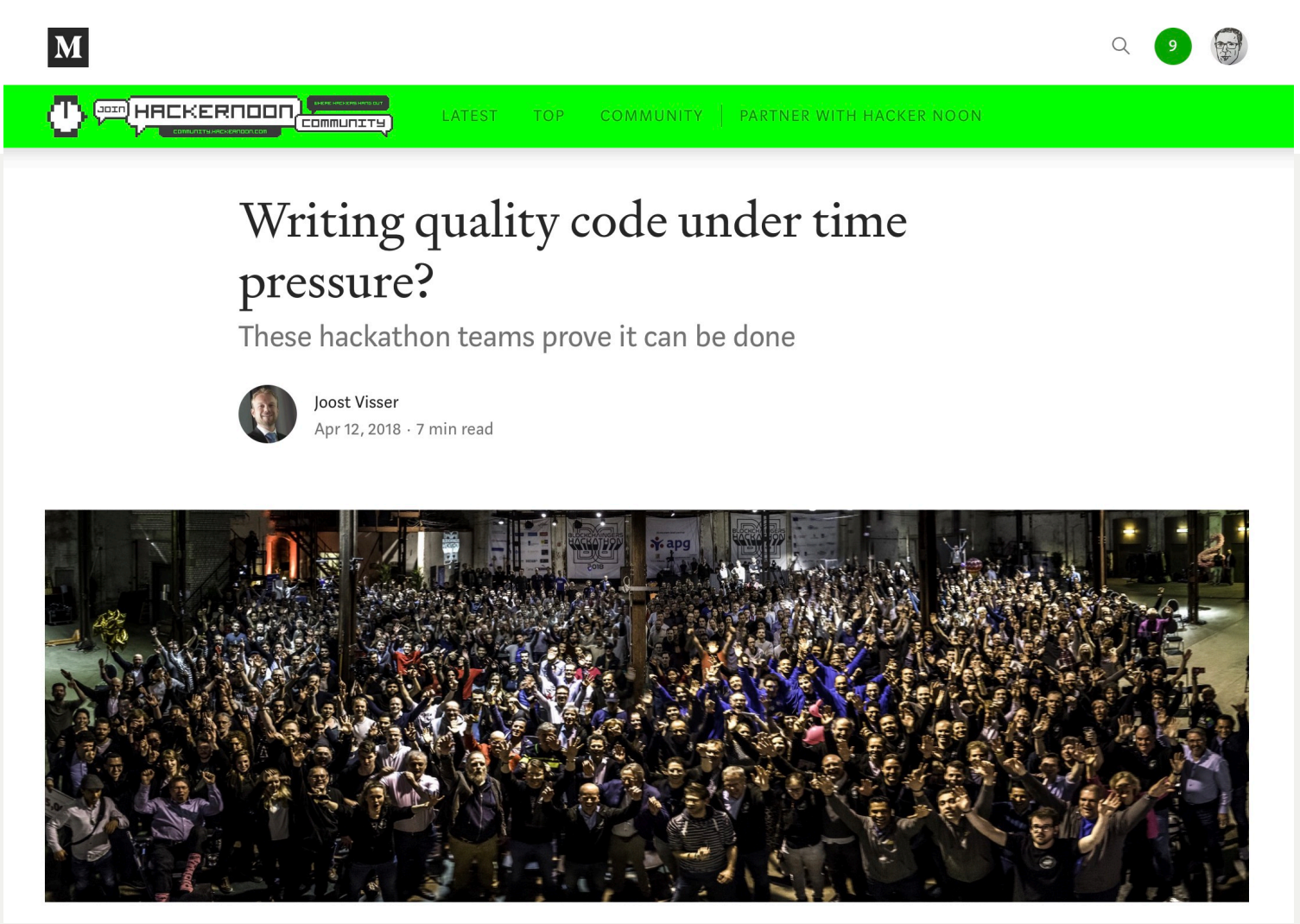
Who was here last year?

→ Experiences & Remarks

What is different this year?

→ Grill sessions by JEDI

→ ‘The Mature Prototype’



Introductions

Luc Brandts

CTO

@lucbrandts

Michiel Cuijpers

Manager Academy

Community manager BetterCodeHub

@michielcuijpers

Software Improvement Group

IT Management Advisory and Measurement Lab

- ISO 25010 software product inspections
- ISO 17025 lab analyses 25 million lines of code each week
- Translate technical findings into actionable recommendations

Agenda today:

1 Introduction Software Improvement Group

2 What is Maintainable code any way? And why should we care?

- → https://odysseyhack.github.io/code-quality-matters
- → The 10 guidelines

3 How to get a mature prototype certificate

- → https://odysseyhack.github.io
- → The Mature Prototype Grill Masters

4 Questions?

About – Software Improvement Group

GETTING SOFTWARE RIGHT

> Factual and actionable management advice through source code analysis and structured methods

> Assessment of a variety of technologies by using technology-independent methods

Impartial, objective mediator between IT suppliers and clients

Active contributor to scientific research in software engineering

Providing management with insight into IT projects and systems by means of facts and measurements

Eligible for TÜViT certification of software systems

SIG/TÜViT Maintainability Model

19 years of software quality measurement

a.k.a. The SIGRID platform

200+
technologies
from mainframe to mobile

2,500
industry systems evaluated

225+
clients

10×10^6
lines of code analyzed each week

22,000+
inspections

8.6×10^9
lines of code in analysis warehouse

Some clients of SIG

Some services of SIG

Assessments
Support risk-based decision making with in-depth analysis of your software

Inspections
Profit from regular checks – security, reliability, portability, performance, usability

Monitoring
Have a grip on quality, architecture, and productivity, at all times

IT Due Diligence
Secure your investment by upfront identification of technology risks

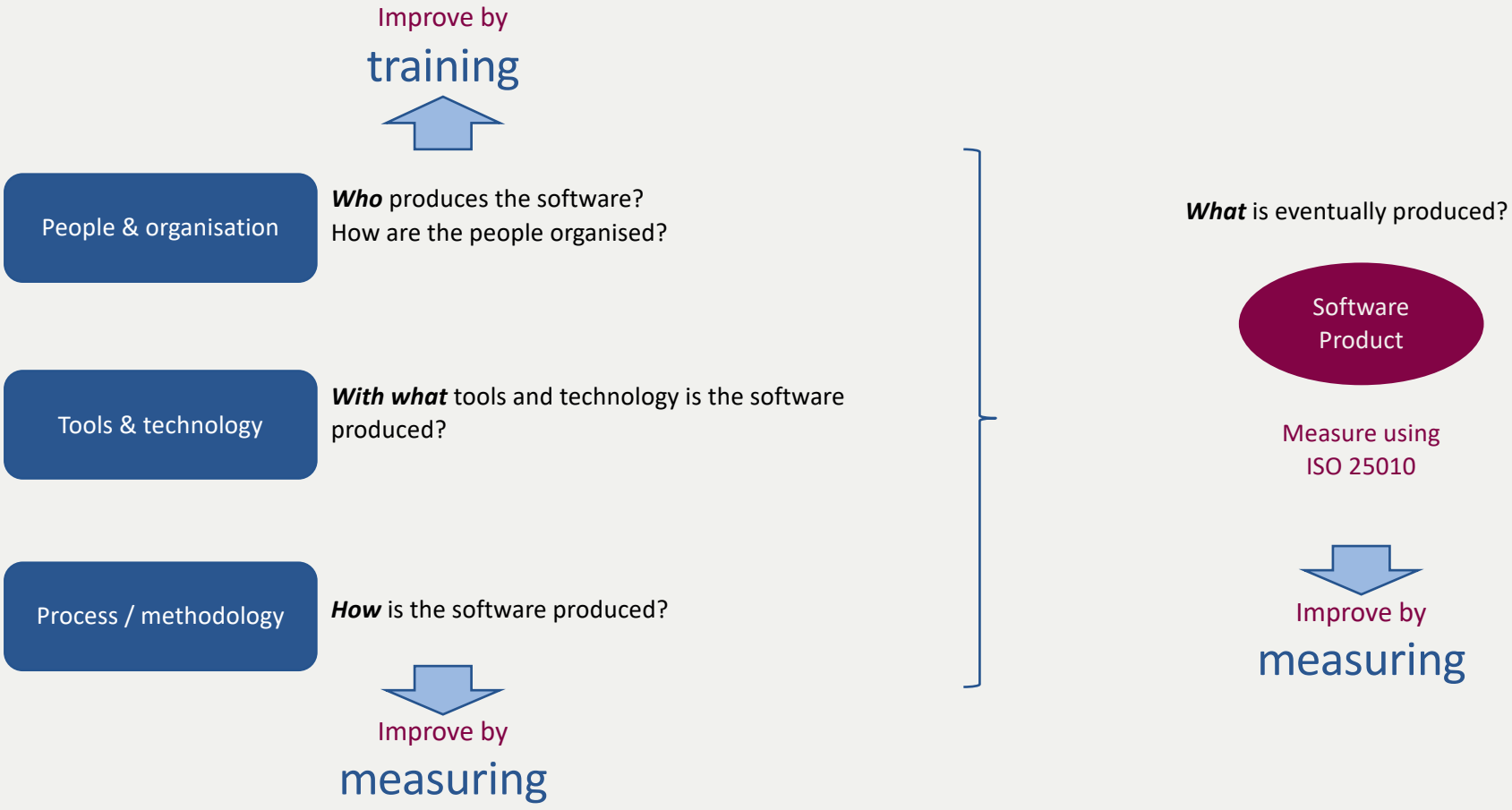
Software Delivery Assurance
Ensure a straight path from solution shaping to product delivery

Training and Certification
Facilitate developers to deliver top achievements

Relation of code quality – investor value

Great software doesn't just happen

Train and Measure

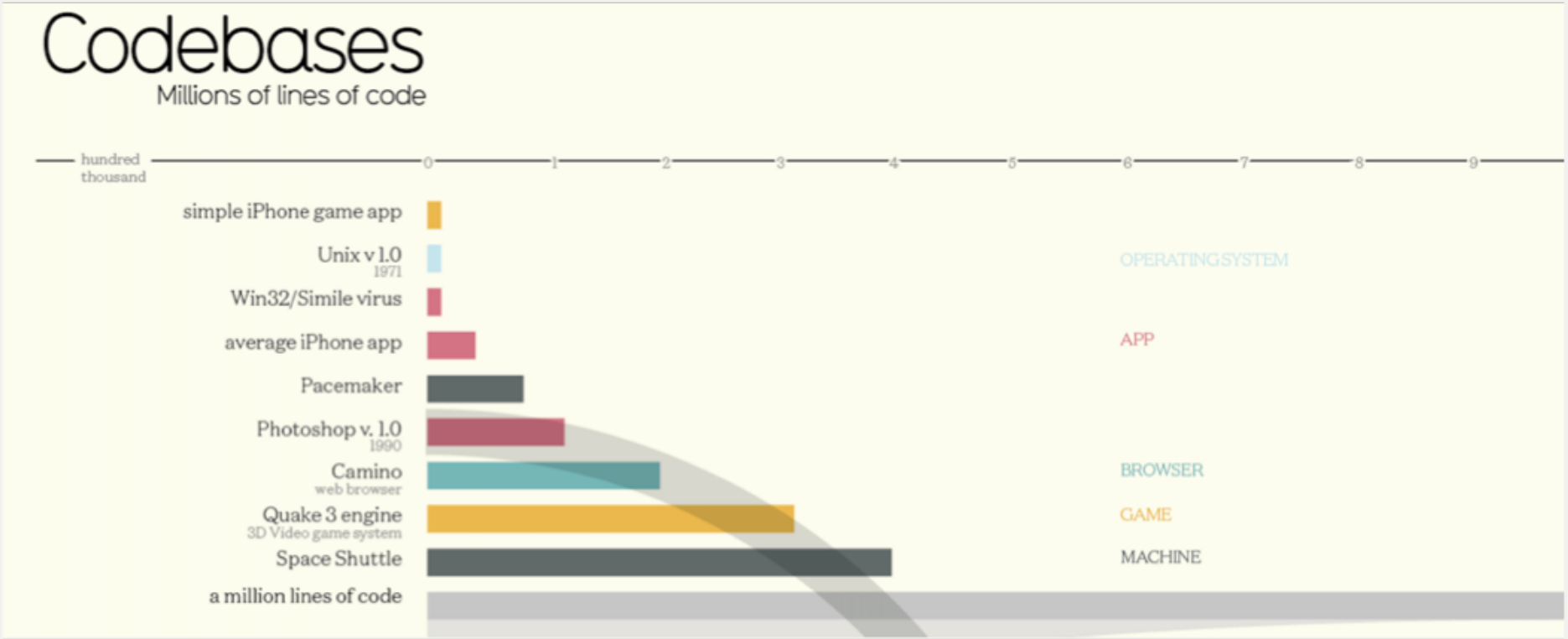


Question:

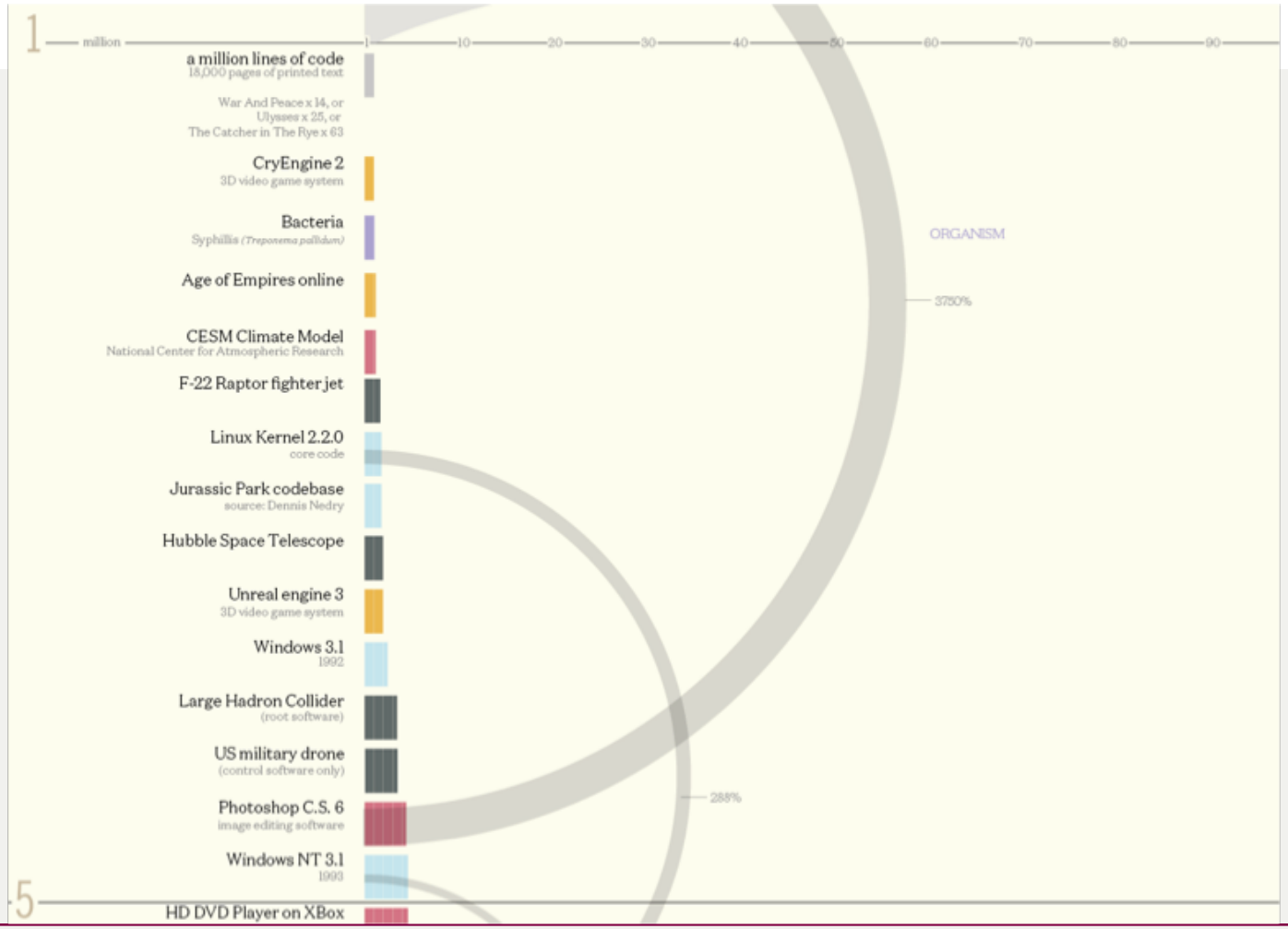
Any idea about size?

About Real software

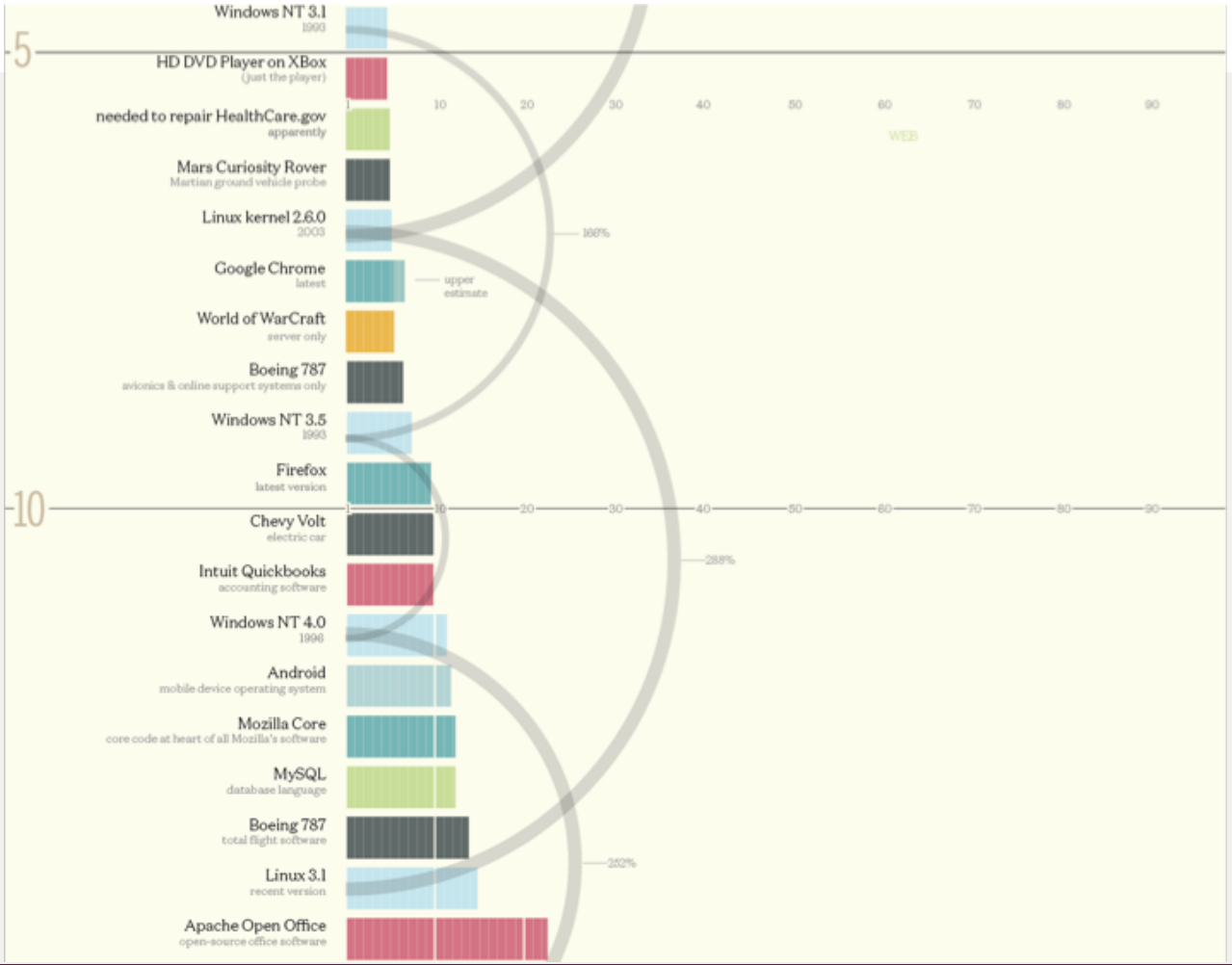
Just to get an idea how big software gets over time



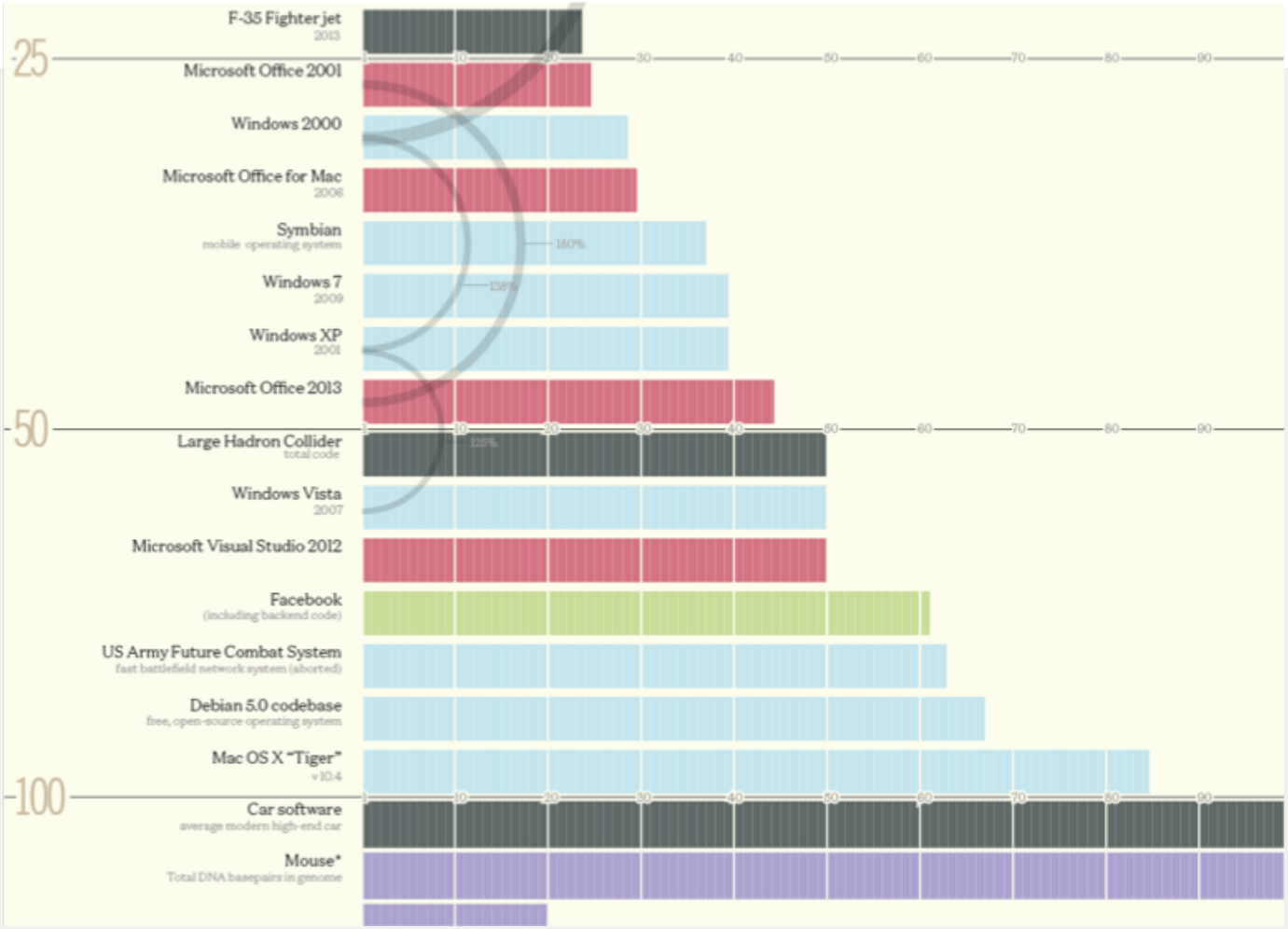
Real software is very large…



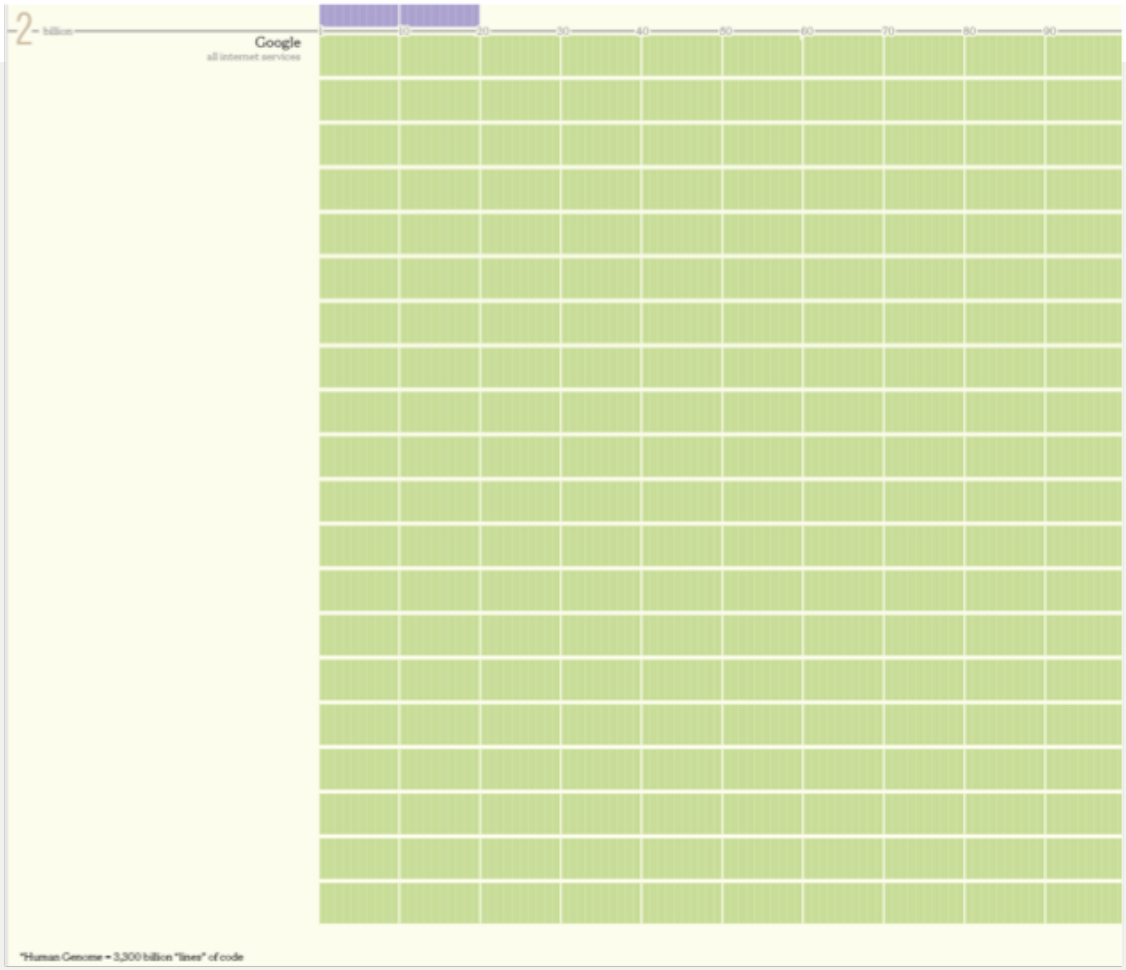
Real software is huge…



Real software is absurdly huge…



Software is taking over the world…



Some interesting statistics

Job security ..

- Average programmer productivity:
  - Roughly 45 lines of working Java code per working day
  - Let's say 10.000 LOC/year
- Operational software ***yearly*** change rate:
  - Roughly 5 to 15% of the code base
- Average ***yearly*** maintenance effort a high-end consumer car:
  - 100 MLOC x 5 to 15% = 5 to 15 MLOC
  - 15 MLOC / 10.000 LOC / year = 500 to 1.500 fte worth of maintenance programmers!

Some more interesting statistics

> In 2020 only 30% software projects are “new”, the rest is “maintenance.” (source: Software Productivity Research)

> 0.1 - 10 bugs per 1,000 lines of code. (sources: SEI, Coverity)

Relevance of code quality – speed

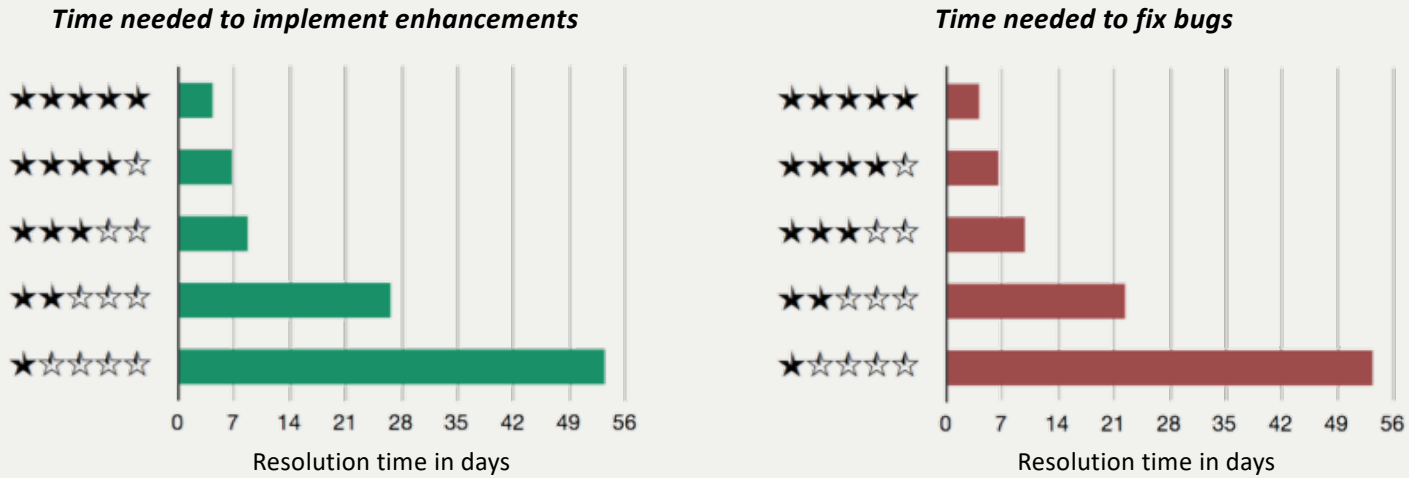
> Who recognizes this and dares to share?

> Who has rejected a job because of bad code?

Brian McAllister

Correlation between maintainability rating and issue resolution time

Issues can be resolved a factor 3.5 faster on systems of high technical quality



Source: "Dennis Bijlsma, Faster issue resolution with higher technical quality of software", Software Quality Journal, 2011

What if we could make it easy to write **maintainable** code?

Tushar Sharma

"Improving maintainability does not require magic or rocket science. A combination of relatively simple skills and knowledge, plus the discipline and environment to apply them, leads to the largest improvement in maintainability."

prof Joost Visser

If you should remember just one thing...

The quality of (your) code matters & should be easy to control!

But how...?

Software quality can be measured and controlled using tools.

Measuring software maintainability

Foundational concepts

Simple to understand	Everyone with a software engineering background should be able to understand the measurement results.
Objective	The measurement results should not depend on the person performing the measurements.
Allow root-cause analysis	The measurements should give insight into *why* the system is maintainable.
Allow comparing systems	It should be possible to compare the measurement results for different systems, to determine which areas are more/less maintainble.
Technology-independent	The measurements should be usuable regardless of what technologies were used to implement the system.

How to define Maintainability?

Sustainable business requires maintainable software

Quality

product – process – people

measurement – management – education

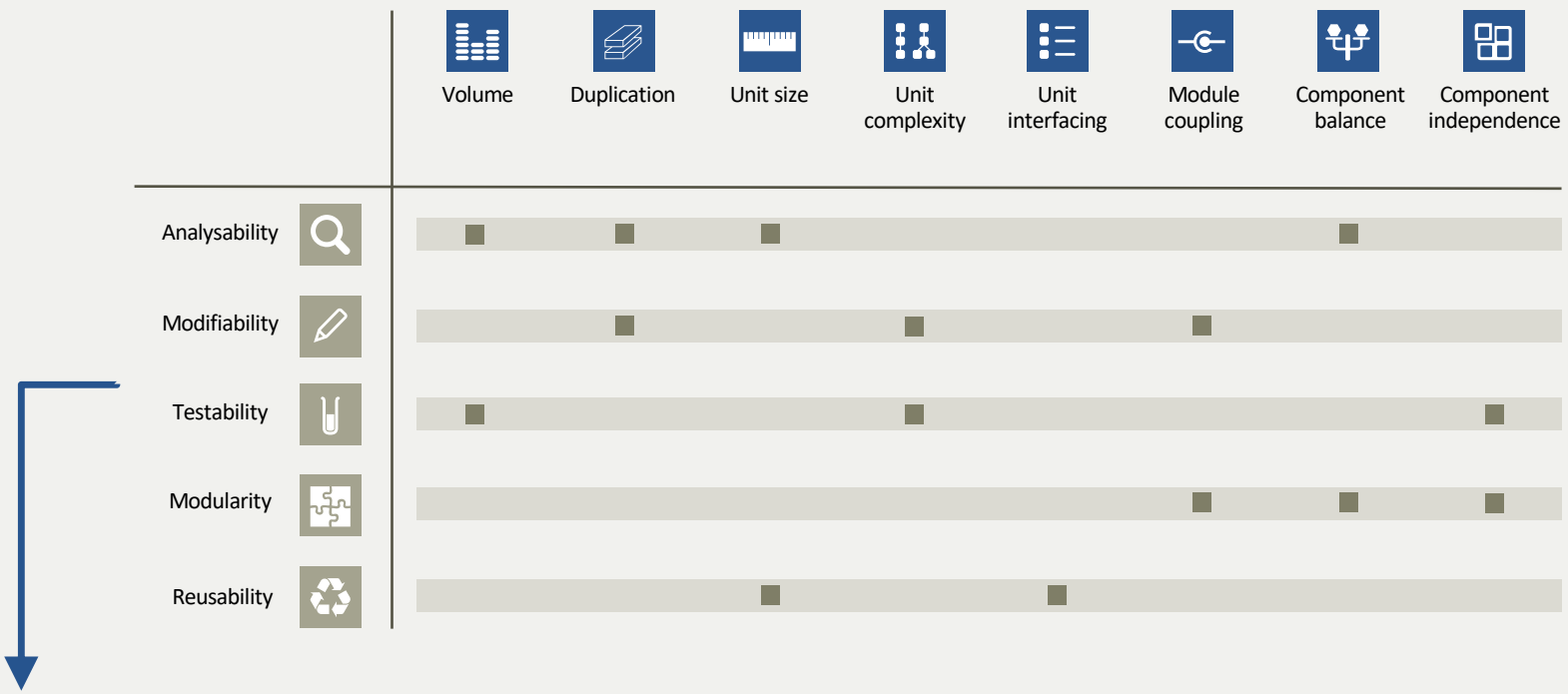
continuous change

SIG Quality Model • Maintainability

Operationalization of ISO 25010 (was: ISO 9126)

First published as “A practical model for measuring maintainability”, Heitlager, Kuipe, Visser, QUATIC 2007.

Most Influential Paper

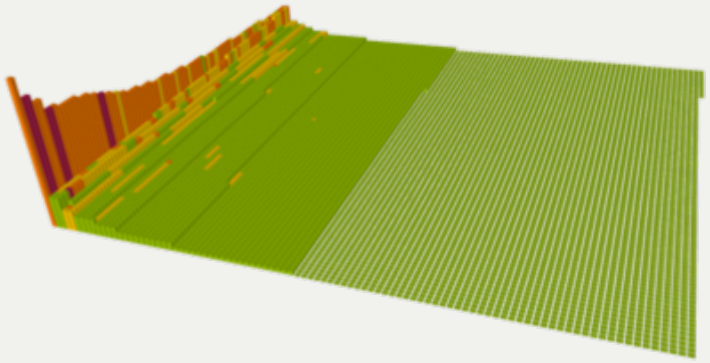


The sub-ratings are aggregated to an overall maintainability rating, where ★★★☆☆ is market average.

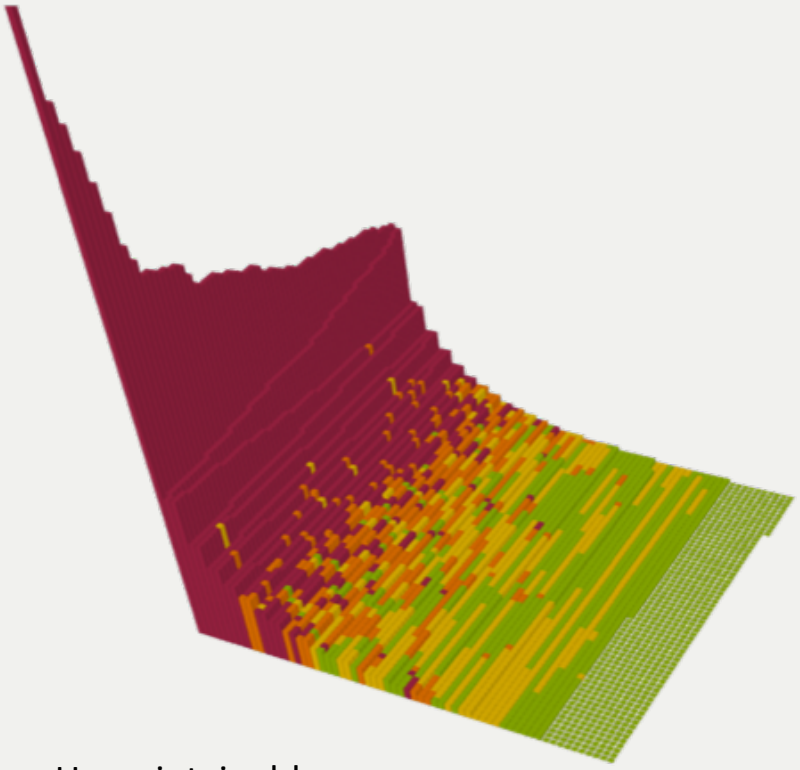
Source code measurements are aggregated in statistically sound “quality profiles” and then rated on a calibrated scale from ★☆☆☆☆ to ★★★★★.

No codebase is perfect!

But some code bases are more ‘perfect’ than others



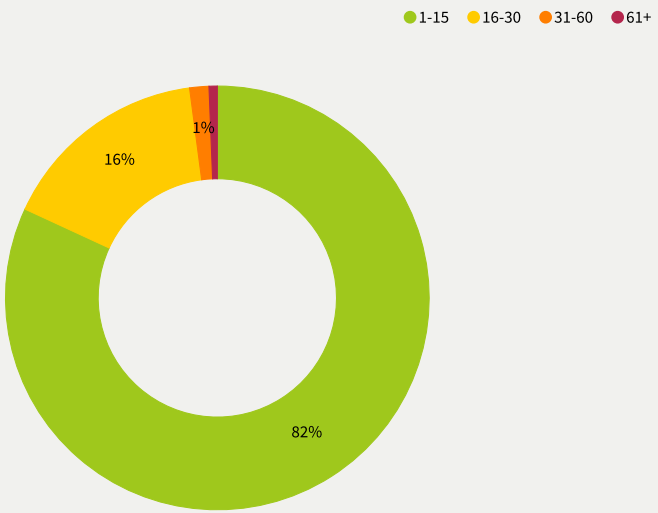
Maintainable



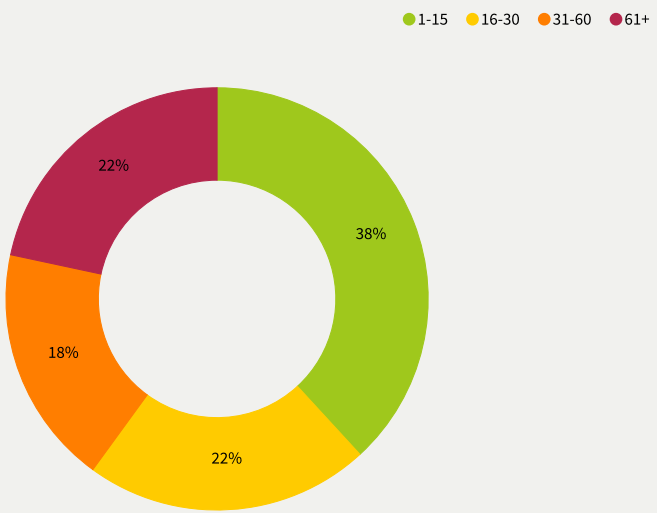
Unmaintainable

Quality profiles: percentage of code in various risk categories

Quality profiles for Unit Size



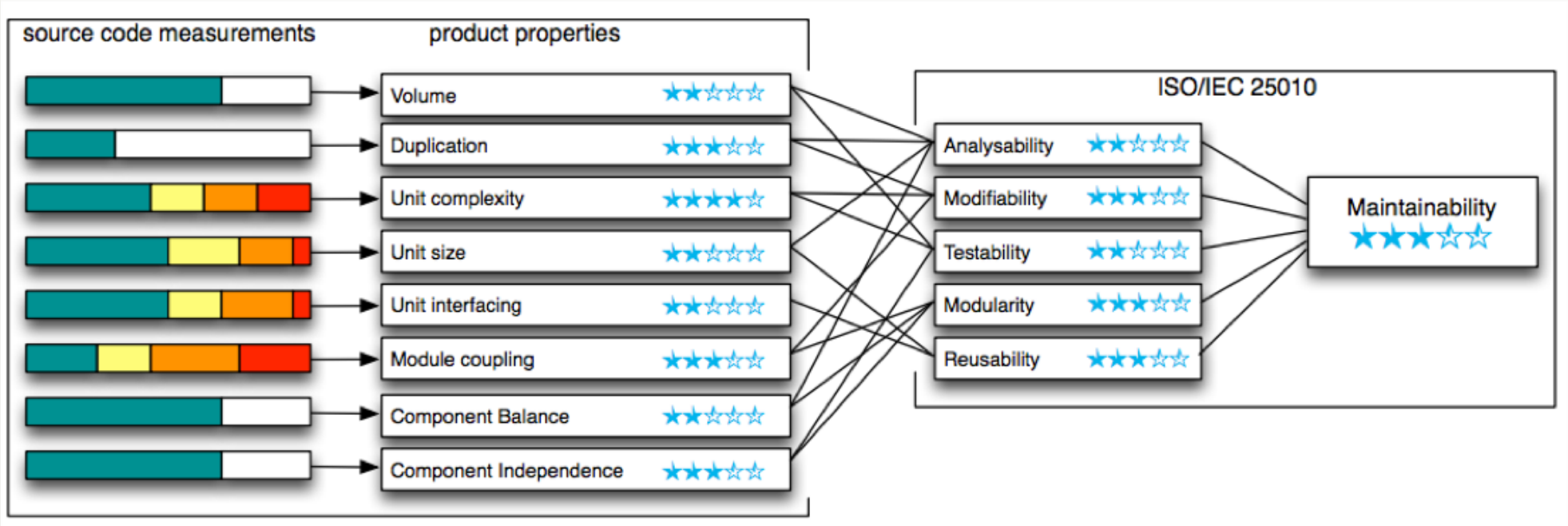
Maintainable



Unmaintainable

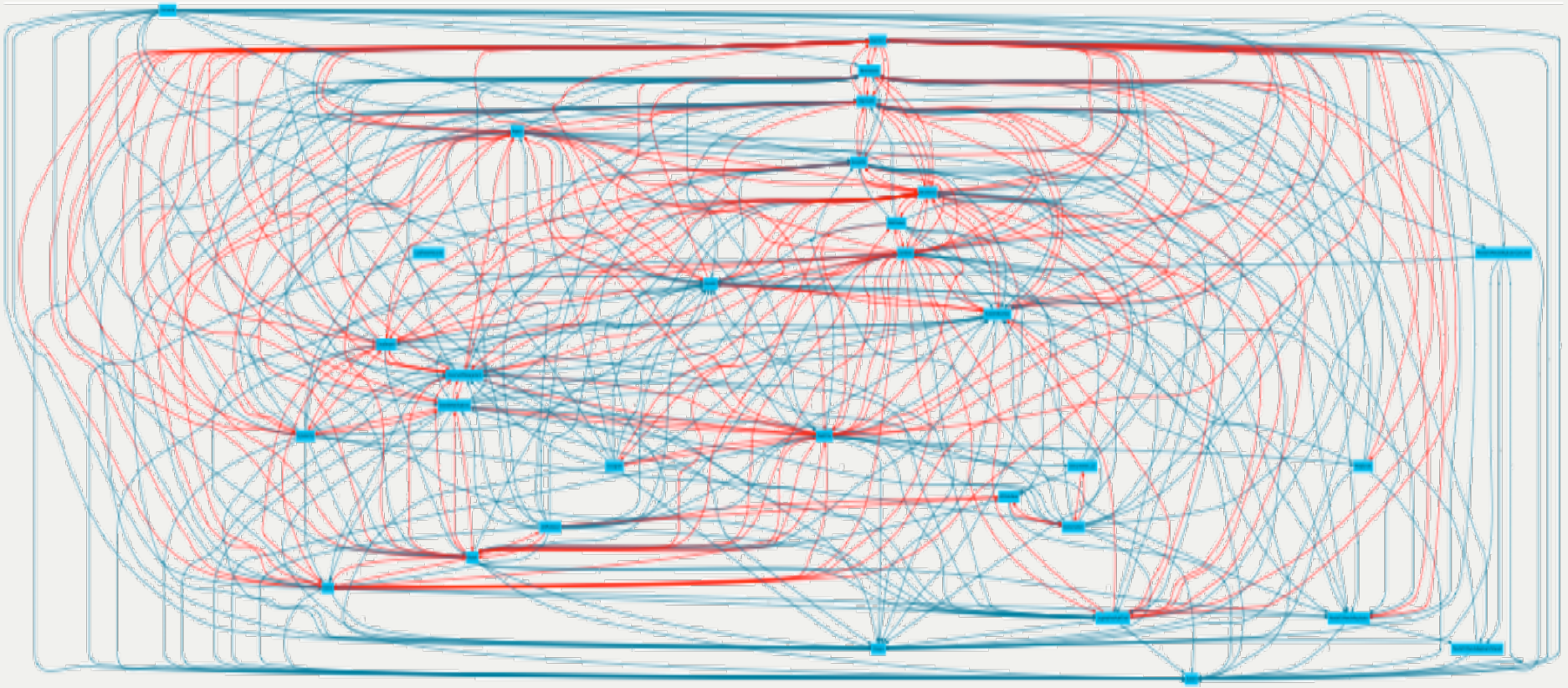
A star rating for software system maintainability

Software Improvement Group SIGRID platform uses star rating



What do we see & How did this happen?

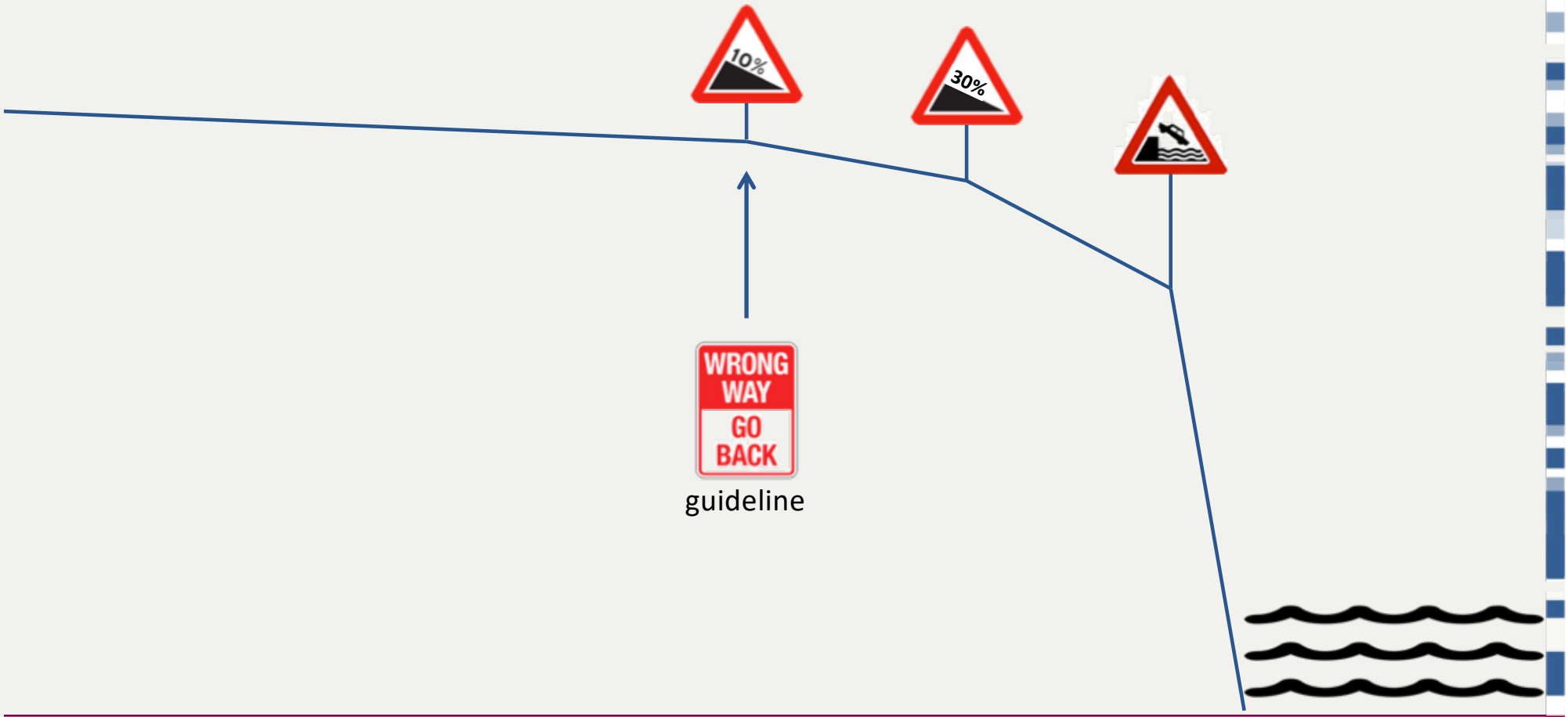
One line of code at a time



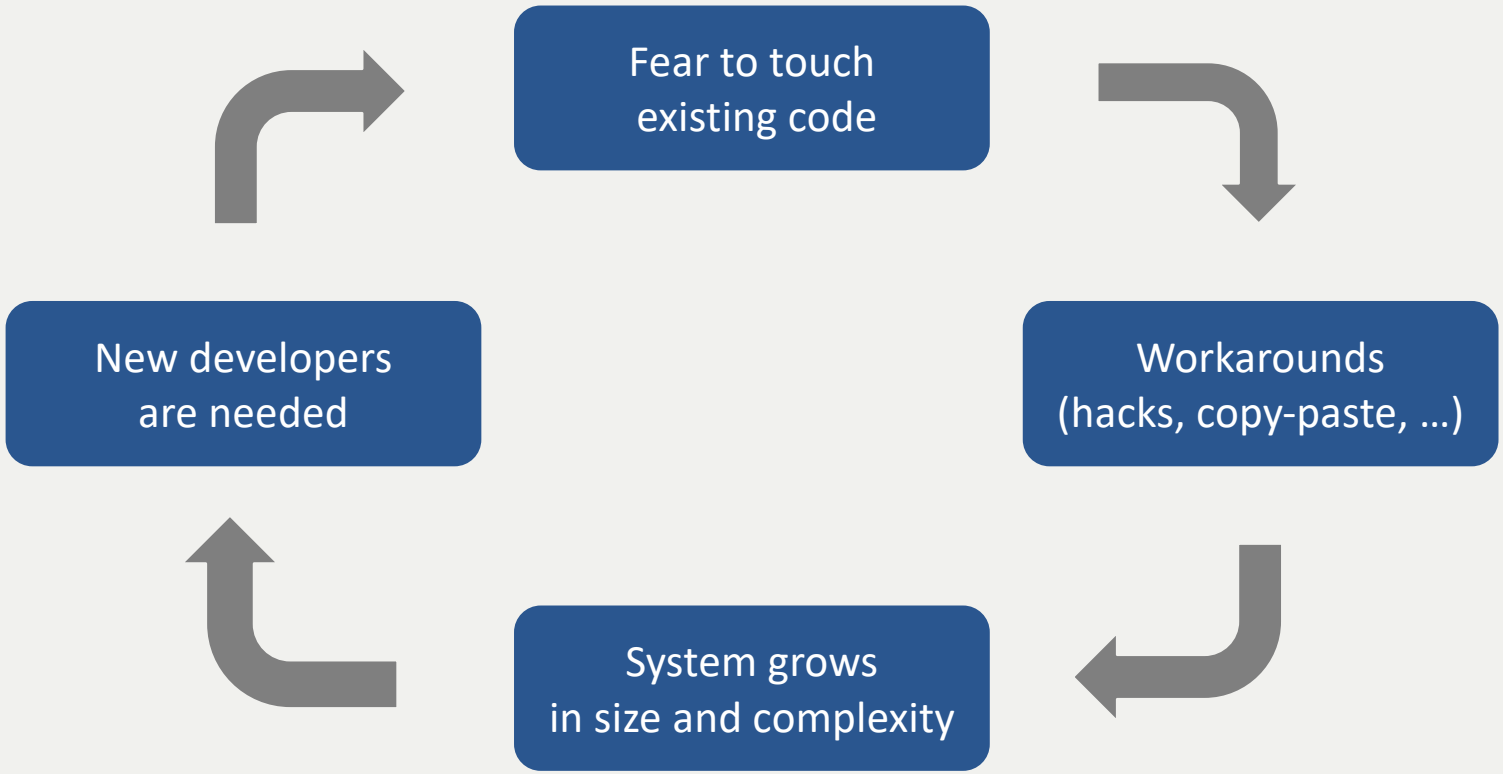
And how did *this* happen?

Also one line of code at a time

Identifying Tipping Points

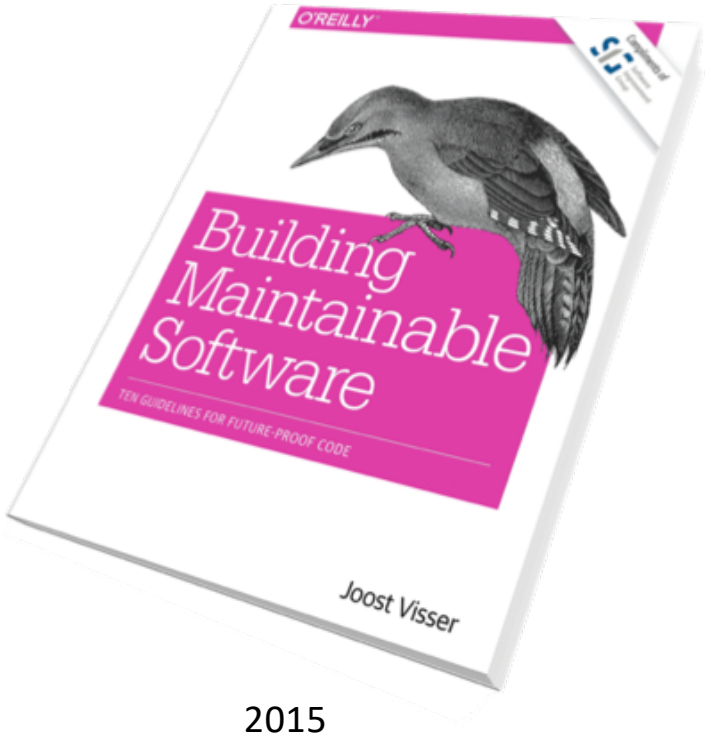


The Vicious Cycle of Unsustainable Software Development



Recent practitioner-oriented books we published

To break the vicious cylcle



2015

2016

The 10 Guidelines for Maintainable Software

Basic

1. Write Short Units of Code
Short units are easier to understand.

2. Write Simple Units of Code
Simple units are easier to test.

3. Write Code Once
Duplicated code means duplicated bugs and duplicating changes.

4. Keep Unit Interfaces Small
Units with small interfaces are easier to reuse.

Advanced

5. Separate Concerns in Modules
Modules with a single responsibility are easier to change.

6. Couple Architecture Components Loosely
Independent components can be maintained in isolation.

7. Keep Architecture Components Balanced
A balanced architecture makes it easier to find your way.

8. Keep Your Codebase Small
A small codebase requires less effort to maintain.

If time allows

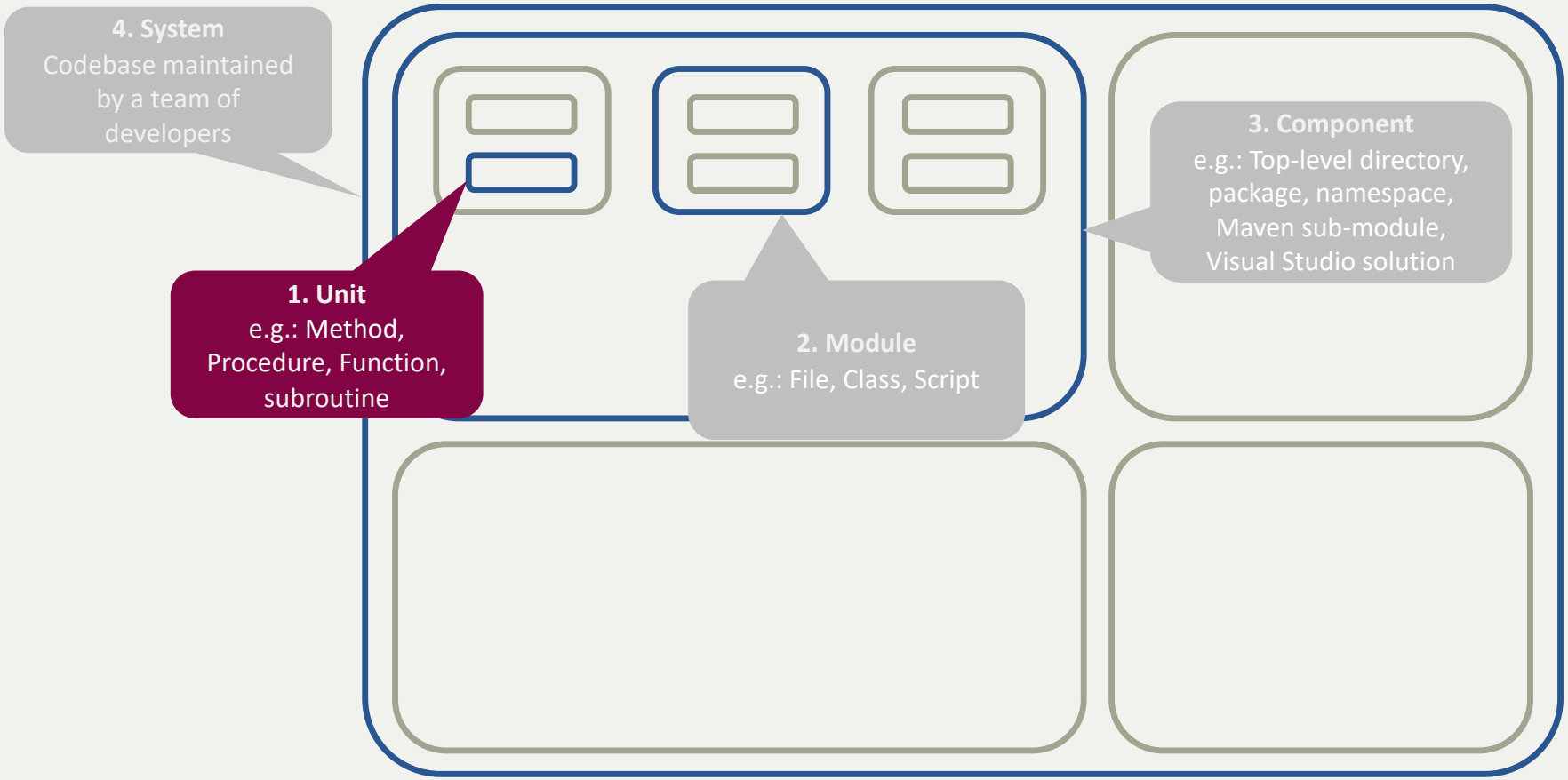
9. Automate Tests
Automated tests are repeatable, and help to prevent bugs.

10. Write Clean Code
"Leave the campground cleaner than you found it."

1. Write Short Units of Code

Short units are easier to understand.

What is a unit?



Write short units of code

The guideline

Do

> Limit the length of code units to 15 lines of code

By

> Not writing units that are longer than 15 lines of code

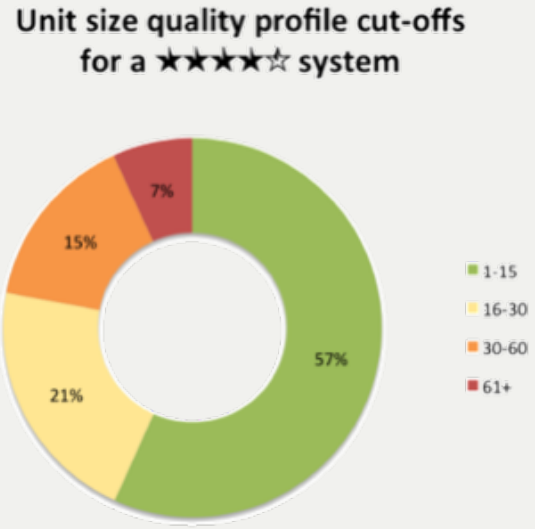
> Splitting long units into multiple smaller units

Because small units are

> Easy to understand

> Easy to test

> Easy to reuse



Positive effects on other guidelines

> Unit Complexity

> Size of codebase

Write short units of code

How you can measure it

Count lines of code

> Every line in the unit that is non-empty and does not contain only comments is a line of code.

```java
/**
 * Replaces all occurrences of a word in a given string with blanks.
 */
public static String clearAllMatches(String input, String word) {
   String result = "";
   int startIndex = 0;
   int matchIndex = input.indexOf(word);

   while (matchIndex >= 0) {
      result += input.substring(startIndex, matchIndex);
      // Insert spaces as filler
      for (int i = 0; i < word.length(); i++) {
         result += " ";
      }
      startIndex = matchIndex + word.length();
      matchIndex = input.indexOf(word, startIndex);
   }

   result += input.substring(startIndex);
   return result;
}
```

15 lines of code

Write short units of code

An example of non-compliant code

```
public void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
    resp.setContentType("application/json");
    try {
        Connection conn = DriverManager.
            getConnection(this.conf.getProperty("handler.jdbcurl"));
        ResultSet results =
            conn.createStatement()
                .executeQuery(
                    "SELECT account, balance FROM ACCTS WHERE id="
                        + req.getParameter(conf.
                            getProperty("request.parametername")));
        float totalBalance = 0;
        resp.getWriter().print("{\"balances\":[");
        while (results.next()) {
            // Assuming result is 9-digit bank account number,
            // validate with 11-test:
            int sum = 0;
            for (int i = 0; i < results.getString("account")
                .length(); i++) {
                sum = sum + (9 - i)
                        * Character.getNumericValue(results.getString(
                        "account").charAt(i));
            }
            if (sum % 11 == 0) {
                totalBalance += results.getFloat("balance");
                resp.getWriter().print(
                    "{\"" + results.getString("account") + "\":"
                        + results.getFloat("balance") + "}");
            }
            if (results.isLast()) {
                resp.getWriter().println("],");
            } else {
                resp.getWriter().print(",");
            }
        }
        resp.getWriter().println("\"total\":" + totalBalance + "}");
    } catch (SQLException e) {
        System.out.println("SQL exception: " + e.getMessage());
    }
}
```

This unit has 38 lines of code

2. Write Simple Units of Code

Simple units are easier to test.

Write simple units of code

The guideline

Do

> Limit the number of branch points to 4

By

> Avoiding complex units

> Splitting complex units into simpler ones

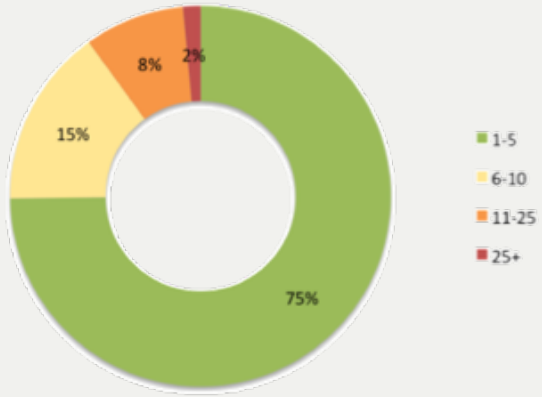
Because simple units are

> Easy to test

> Easy to understand

> Easy to modify

Unit complexity quality profile
cut-offs for a ★★★★☆ system



Positive effects on other guidelines

> Unit Size

> Unit Interfacing

Write simple units of code

How you can measure it

Count the cyclomatic complexity

> Every branch point (if, case, for, &&, ||) is counted, and we add 1 to the total

```java
/**
 * Replaces all occurrences of a word in a given string with blanks.
 */
public static String clearAllMatches(String input, String word) {
   String result = "";
   int startIndex = 0;
   int matchIndex = input.indexOf(word);

   while (matchIndex >= 0) {
      result += input.substring(startIndex, matchIndex);
      // Insert spaces as filler
      for (int i = 0; i < word.length(); i++) {
         result += " ";
      }
      startIndex = matchIndex + word.length();
      matchIndex = input.indexOf(word, startIndex);
   }

   result += input.substring(startIndex);
   return result;
}
```

2 branch points + 1
= cyclomatic
complexity of 3

3. Write Code Once

Duplicated code means duplicated bugs and duplicating changes.

Write code once

The guideline

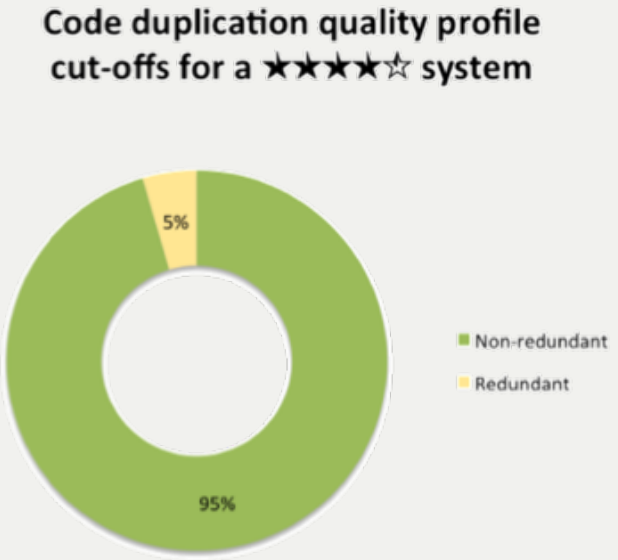
> Not copy code
> (no ⌘C/⌘V, no CTRL+C/CTRL+V)

By

> Instead writing reusable, generic code and/or calling existing methods

Because

> When code is copied, bugs need to be fixed in multiple places, this is inefficient and error-prone



Positive effects on other guidelines

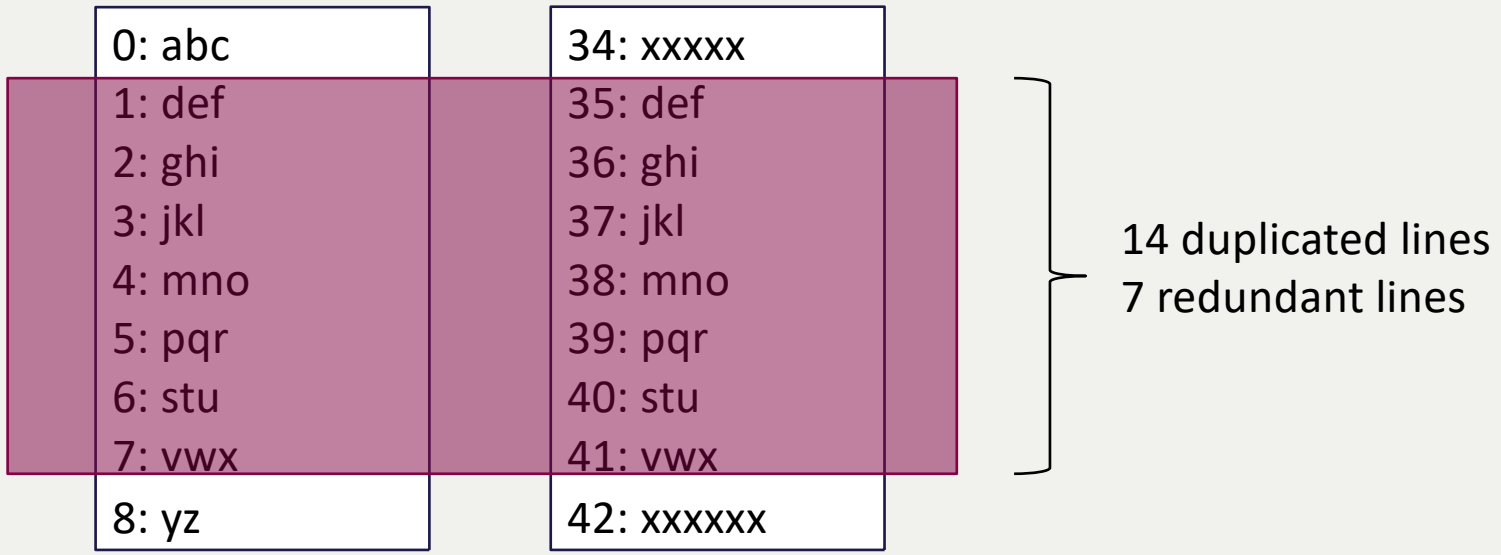
- Keep your codebase small

Write code once

How you can measure it

Count the percentage of duplication

> Every piece of code of 6+ lines of code that occurs more than once in the codebase is a duplicate



Write code once

Impact of duplicated code on future changes

Low duplication

- One defect means one fix

High duplication

- One defect means several fixes ...
- ... and even more might be missed

4. Keep Unit Interfaces Small

Units with small interfaces are easier to reuse.

Keep unit interfaces small

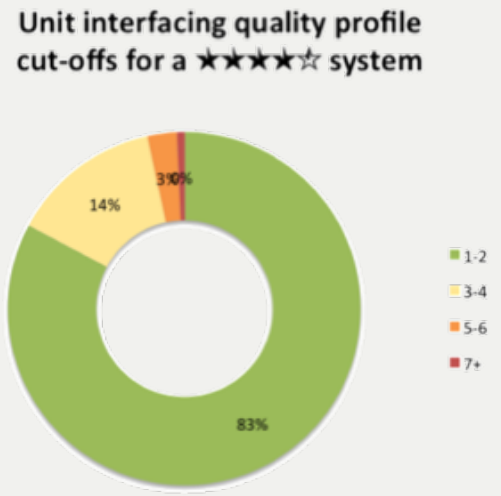
The guideline

Do

> Limit the number of parameters per unit to 4

By

> Extracting parameters into objects



Because units with few parameters are

> Easy to understand

> Easy to test

> Easy to reuse

Positive effects on other guidelines

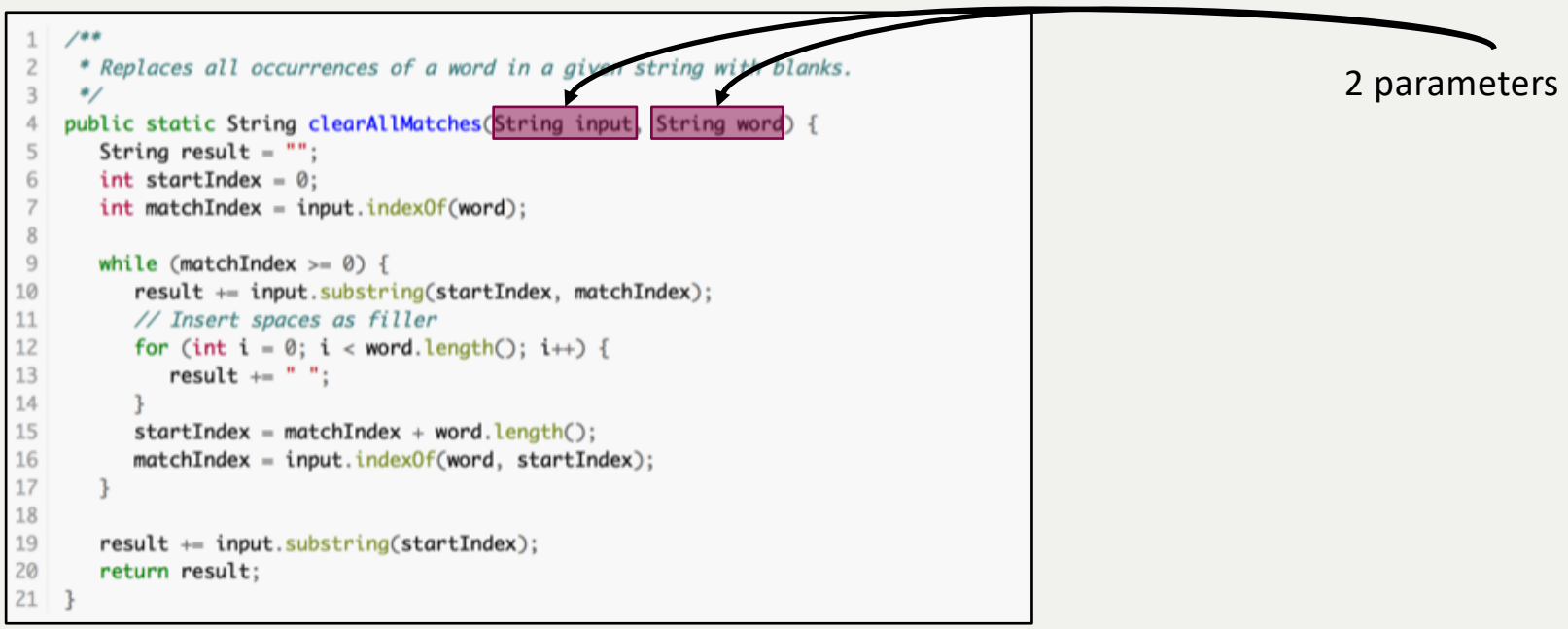
- Simple Units
- Short Units

Keep unit interfaces small

How you can measure it

Count the number of parameters

> Every parameter in the unit header




```java
/**
 * Replaces all occurrences of a word in a given string with blanks.
 */
public static String clearAllMatches(String input, String word) {
   String result = "";
   int startIndex = 0;
   int matchIndex = input.indexOf(word);

   while (matchIndex >= 0) {
      result += input.substring(startIndex, matchIndex);
      // Insert spaces as filler
      for (int i = 0; i < word.length(); i++) {
         result += " ";
      }
      startIndex = matchIndex + word.length();
      matchIndex = input.indexOf(word, startIndex);
   }

   result += input.substring(startIndex);
   return result;
}
```

Keep unit interfaces small

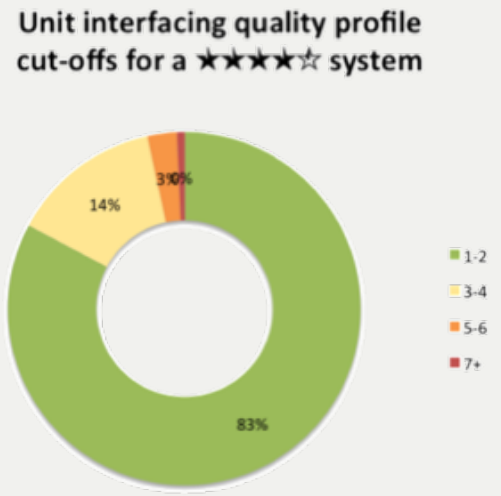
The guideline

Do

> Limit the number of parameters per unit to 4

By

> Extracting parameters into objects



Because units with few parameters are

> Easy to understand

> Easy to test

> Easy to reuse

Positive effects on other guidelines

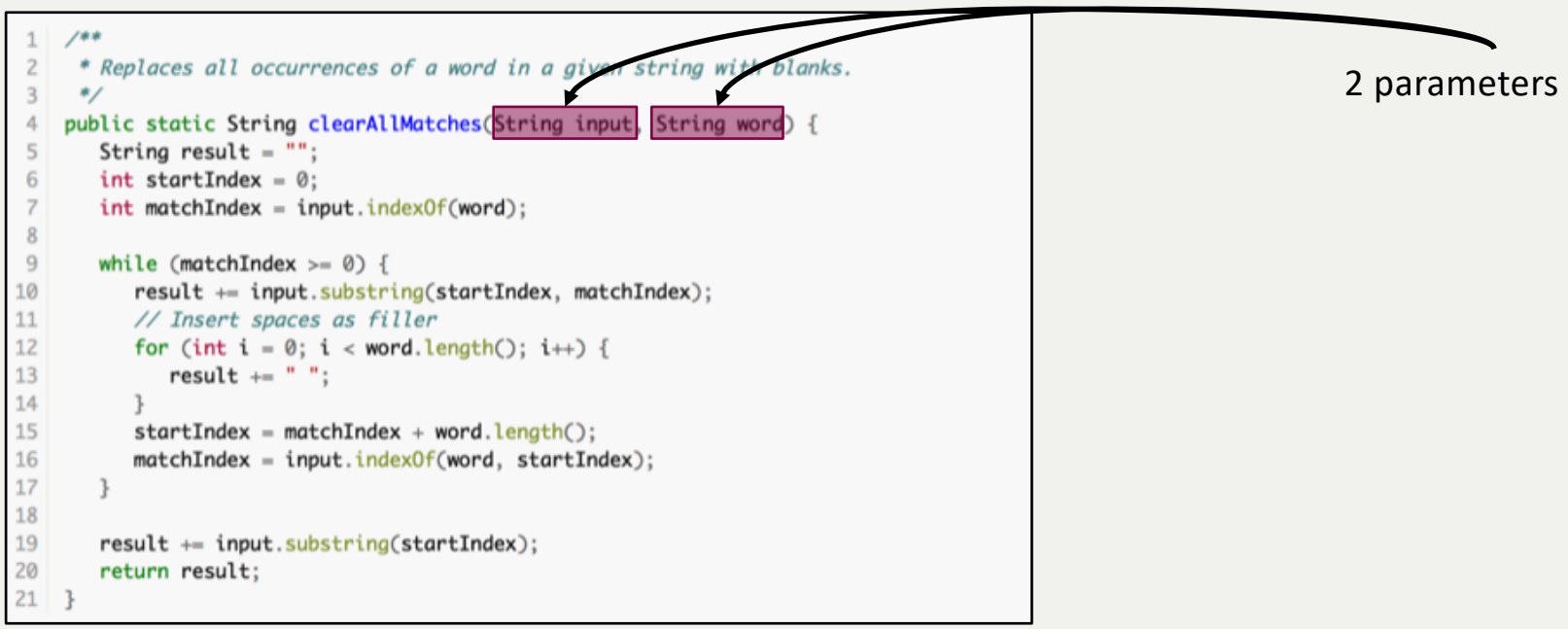
- Simple Units
- Short Units

Keep unit interfaces small

How you can measure it

Count the number of parameters

> Every parameter in the unit header



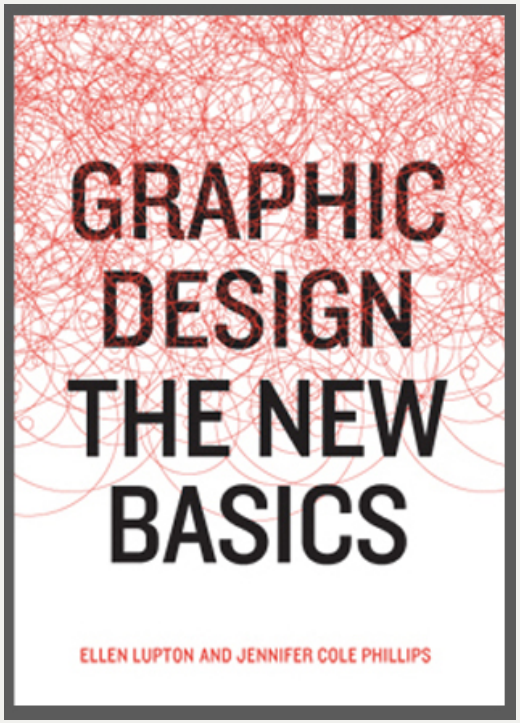

```java
/**
 * Replaces all occurrences of a word in a given string with blanks.
 */
public static String clearAllMatches(String input, String word) {
   String result = "";
   int startIndex = 0;
   int matchIndex = input.indexOf(word);

   while (matchIndex >= 0) {
      result += input.substring(startIndex, matchIndex);
      // Insert spaces as filler
      for (int i = 0; i < word.length(); i++) {
         result += " ";
      }
      startIndex = matchIndex + word.length();
      matchIndex = input.indexOf(word, startIndex);
   }

   result += input.substring(startIndex);
   return result;
}
```

Software design & architecture

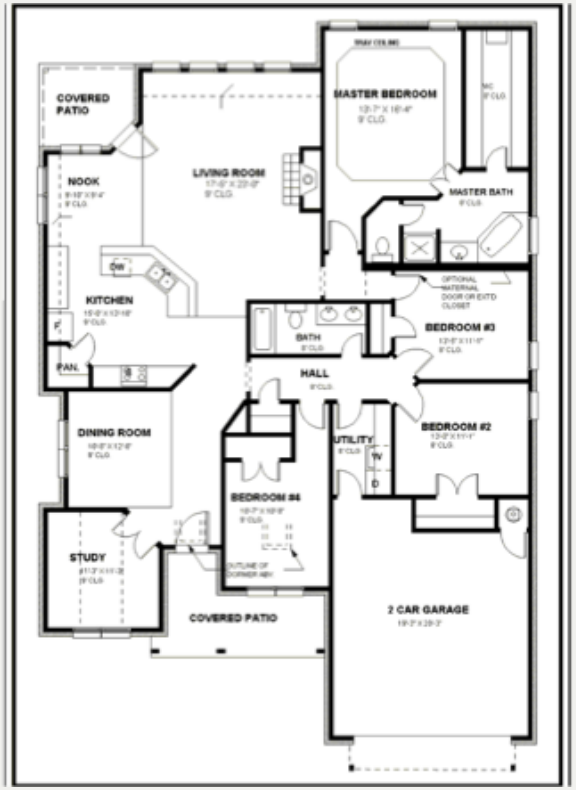
What do we mean by software design?

No, not that kind of design

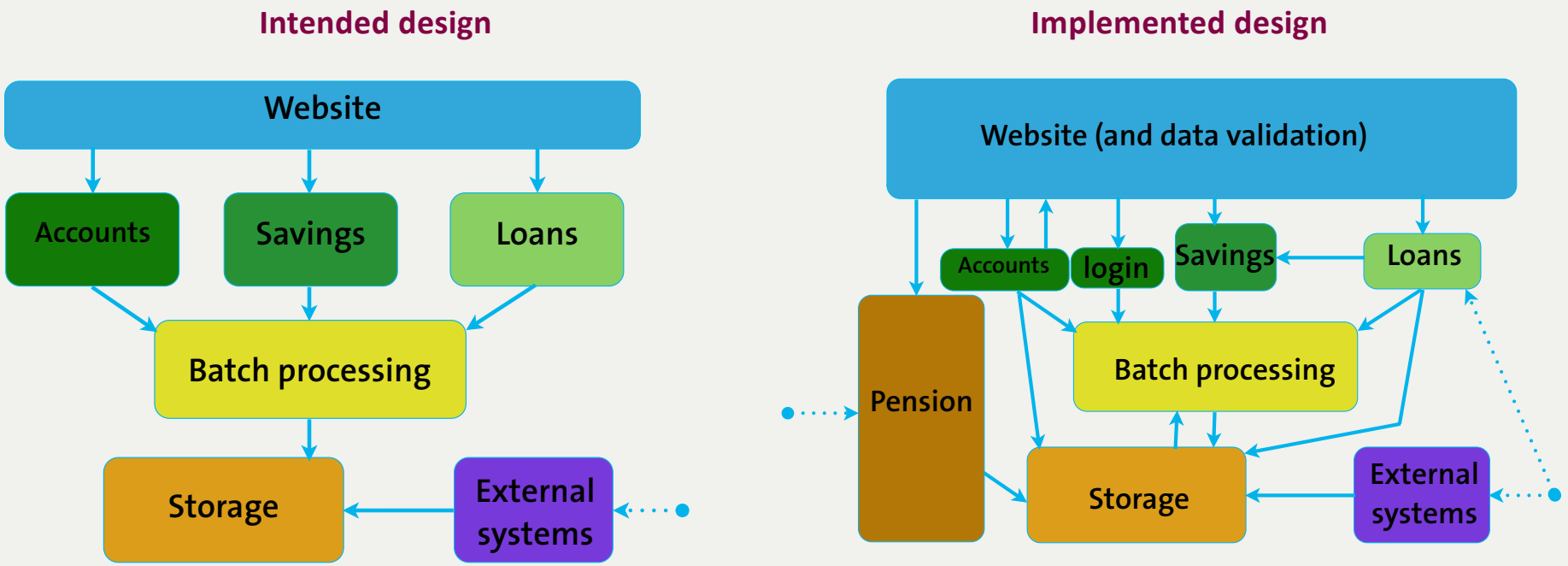
(though it's also important)



We mean "design" as in a high-level plan

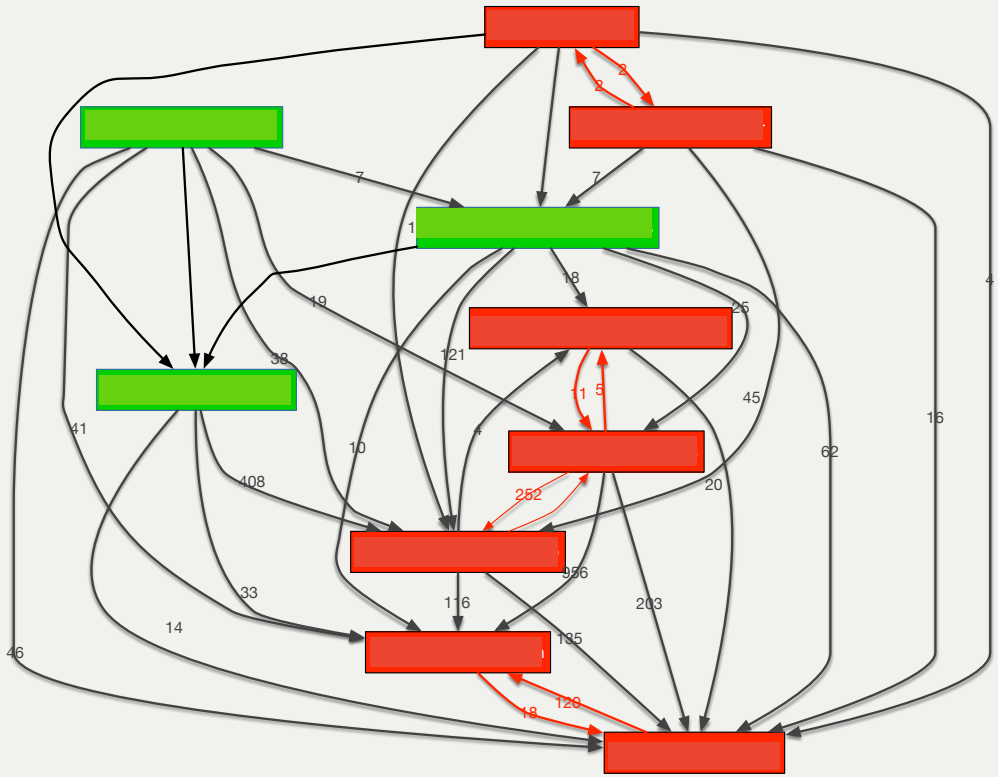


Intended design vs. implemented design



How did this happen?

This is what happens when architecture is not governed



- Over time, unintended dependencies are introduced
- Responsibilities between components are fading over time
- Circular dependencies are introduced

9. Automate Tests

Automated tests are repeatable, and help to prevent bugs.

Automate Tests

The guideline

Do:

> Automate tests for your codebase

By:

> Writing automated tests using a test framework
> Executing these tests as often as feasible

Because automated tests:

> Are repeatable
> Are more efficient
> Make your code predictable

10. Write Clean Code

“Leave the campground cleaner than you found it.”

Write Clean Code

The guideline

Do:

> Write clean code

By:

> Avoiding code smells

> Not leaving smelly code behind

Because clean code:

> Is more maintainable

> Is less confusing than unhygienic code

Write Clean Code

Seven ‘Boy Scout rules’

1. Leave no **unit level code smells** behind
2. Leave no **bad comments** behind
3. Leave no **code in comments** behind
4. Leave no **dead code** behind
5. Leave no **long identifiers** behind
6. Leave no **magic constants** behind
7. Leave no **badly handled exceptions** behind

Better Code Hub

Improve what matters • online • integration with GitHub

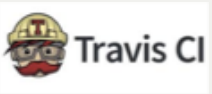
Software Improvement Group supports Odyssey.org in getting code quality right!

Backlog & Documentation

Team Communication

Version Control

Product Quality



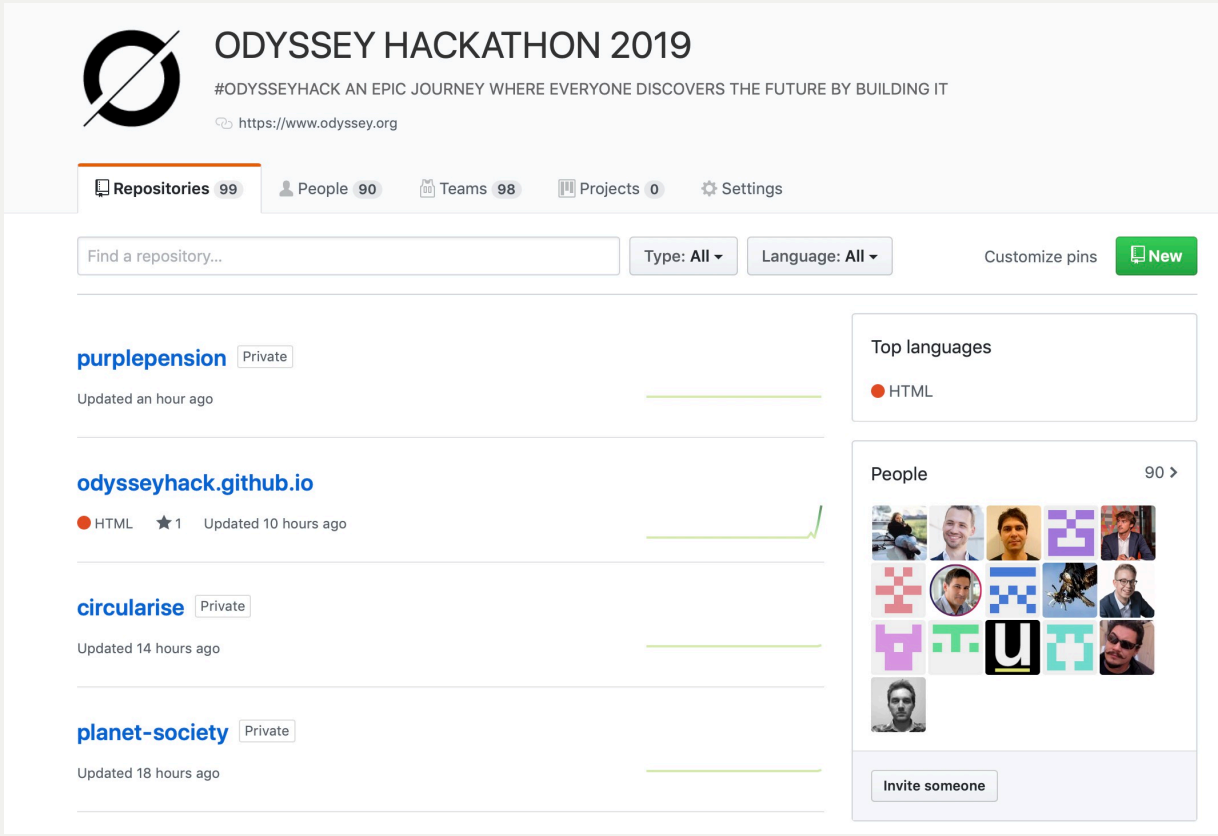
Continuous Integration

Cloud Hosting

Analytics

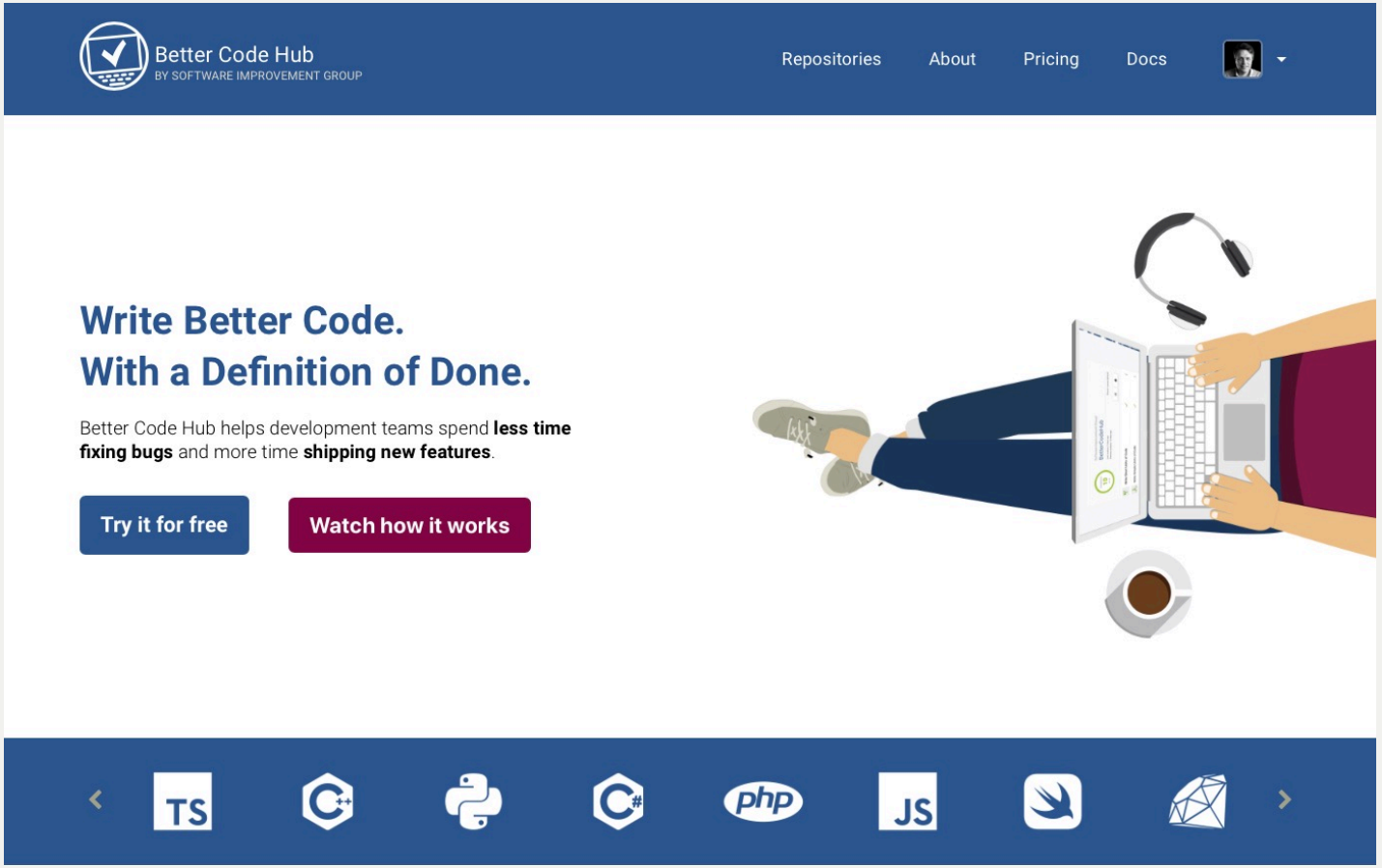
Odyssey GitHub Org

1. Teams
2. Private repo per team
   - ‘Weakest link’
3. Invites have been sent to team leads
4. Send me github handles
5. ***Your IP is protected***

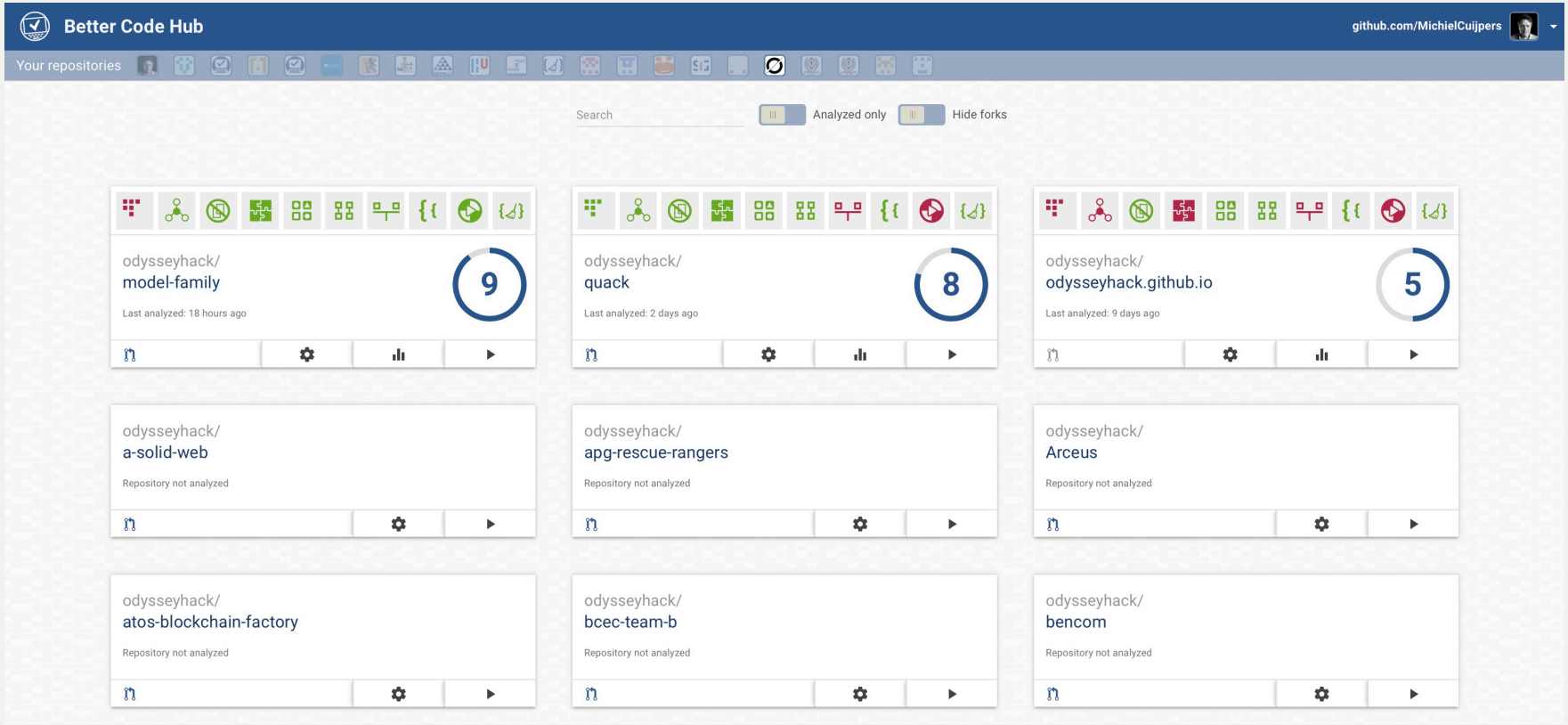


Better Code Hub a tool by Software Improvement Group
https://bettercodehub.com

1. 17 technologies
2. Private and public repos
3. Use Github account
4. Accept scopes
5. View results
6. Select refactor candidates
7. Plan sprint
8. Run again
    1. Repeat step 3
9. Know when done
10. Show score with a badge

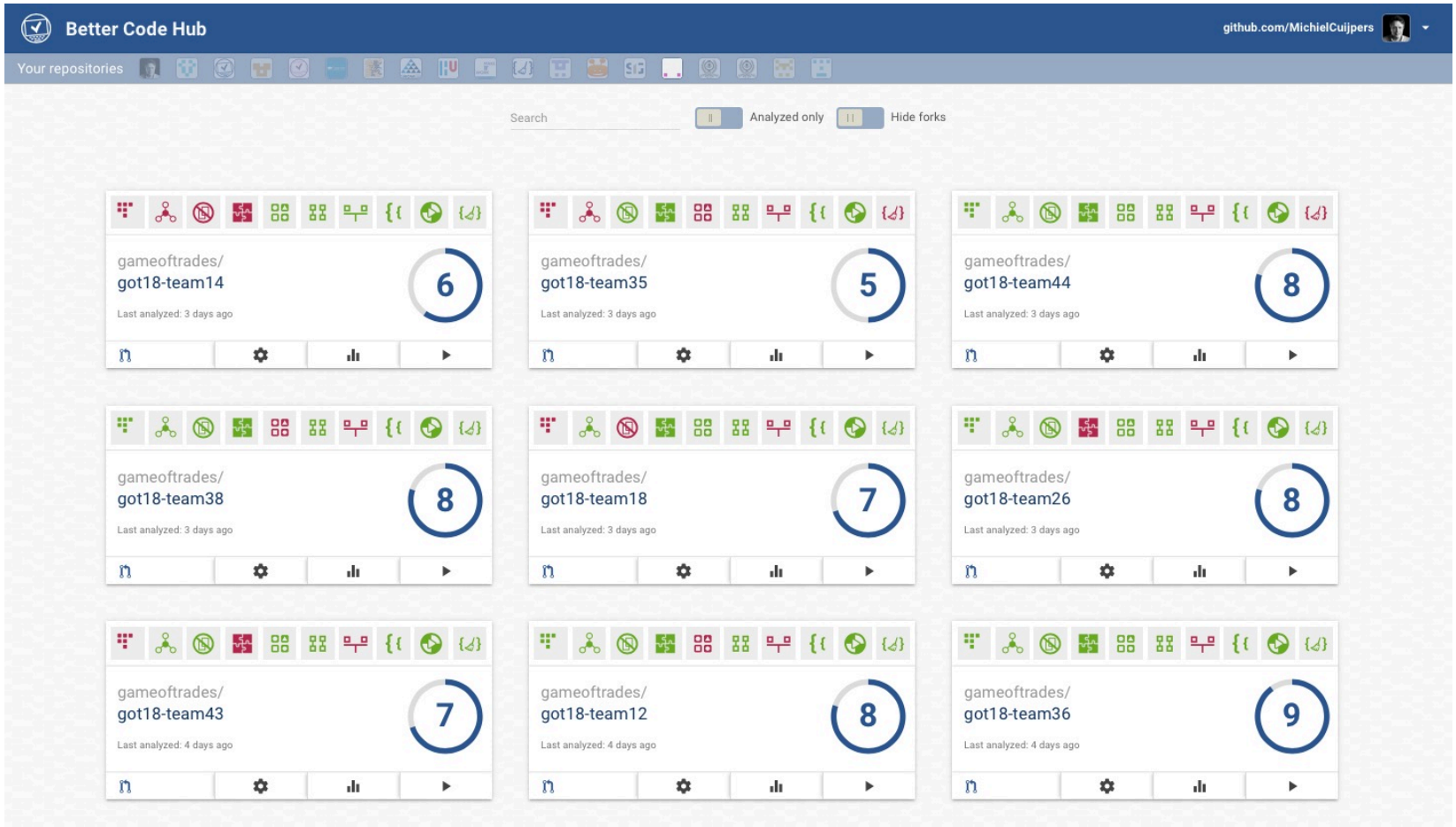


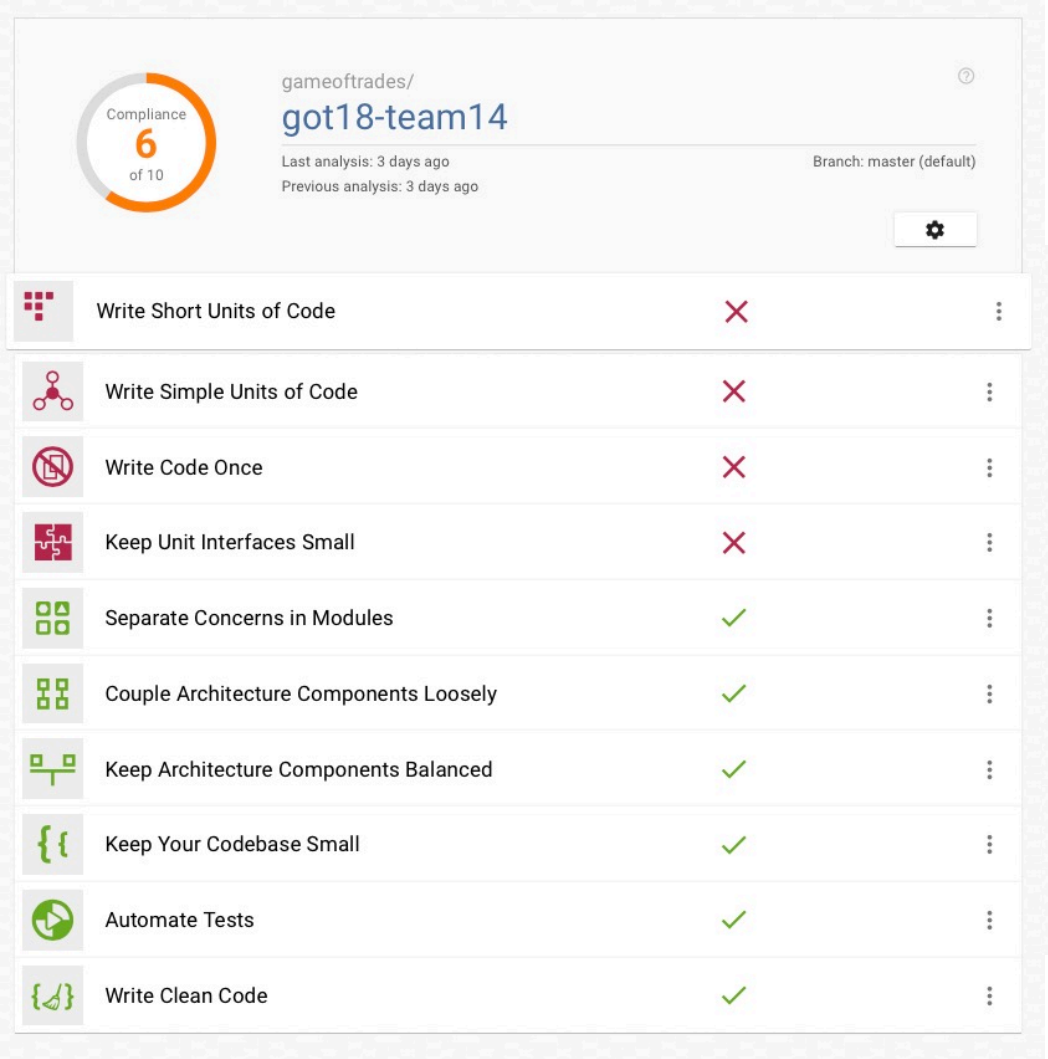
Odysseyhack now

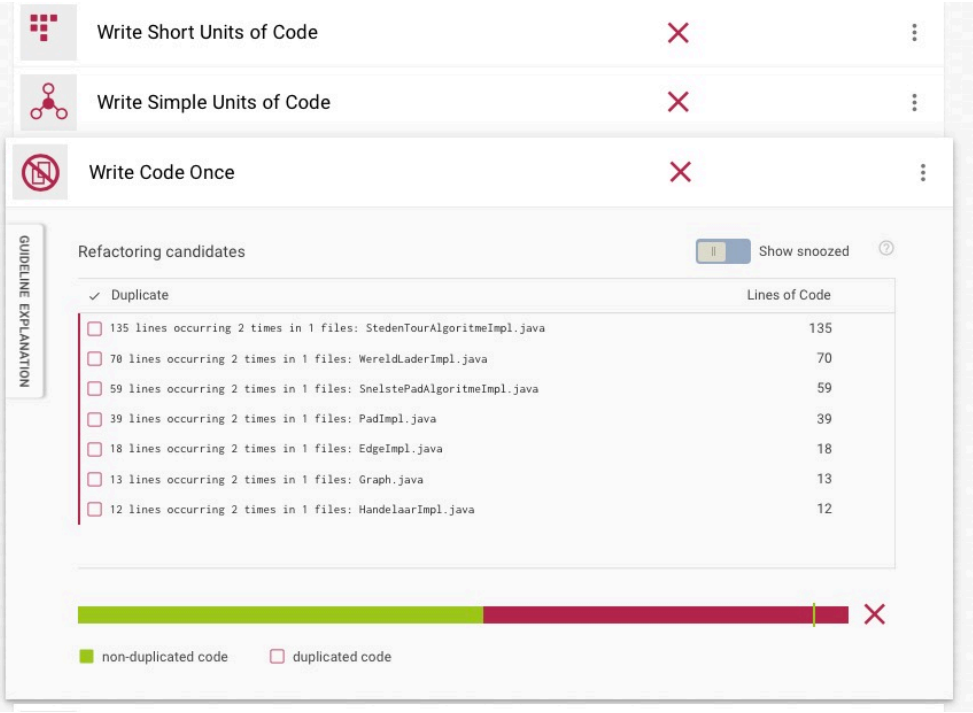


Example overview

Gameoftrades for Haagse Hogeschool

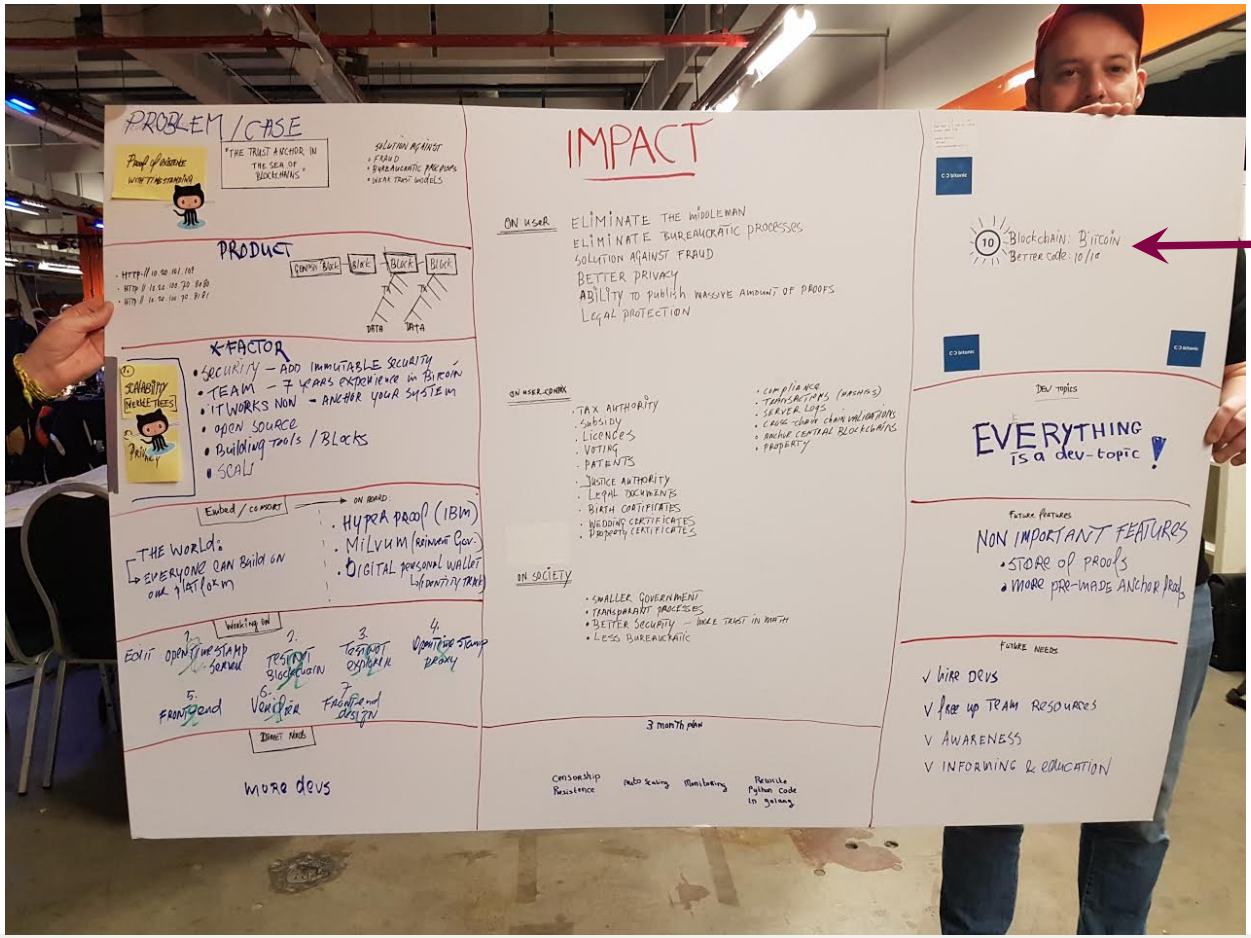



Compliance
6
of 10
gameoftrades/
got18-team14
Last analysis: 3 days ago
Previous analysis: 3 days ago
Branch: master (default)
Write Short Units of Code
Write Simple Units of Code
Write Code Once
Keep Unit Interfaces Small
Separate Concerns in Modules
Couple Architecture Components Loosely
Keep Architecture Components Balanced
Keep Your Codebase Small
Automate Tests
Write Clean Code


Write Short Units of Code
Write Simple Units of Code
Write Code Once
GUIDELINE EXPLANATION
Refactoring candidates
Show snoozed
Duplicate
Lines of Code
135 lines occurring 2 times in 1 files: StedenTourAlgoritmeImpl.java 135
70 lines occurring 2 times in 1 files: WereldLaderImpl.java 70
59 lines occurring 2 times in 1 files: SnelstePadAlgoritmeImpl.java 59
39 lines occurring 2 times in 1 files: PadImpl.java 39
18 lines occurring 2 times in 1 files: EdgeImpl.java 18
13 lines occurring 2 times in 1 files: Graph.java 13
12 lines occurring 2 times in 1 files: HandelaarImpl.java 12
non-duplicated code
duplicated code

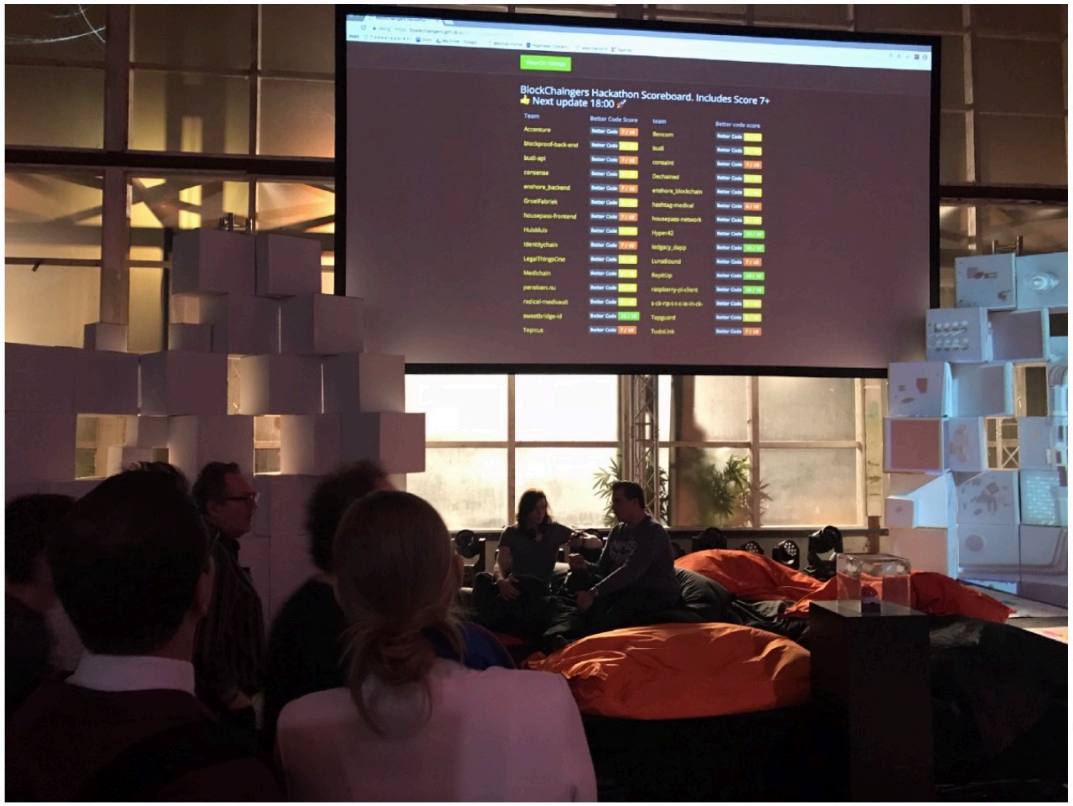
Better Code Hub at Blockchain Hackathons

It started at the Big Building, Groningen



Better Code Hub at Dutch Blockchain Hackathon

Scoreboard



Team	Better Code Hub Score
AC-BC	Better Code 8 / 10
Accenture	Better Code 6 / 10
atma	Better Code 10 / 10
BadgeX	Better Code 8 / 10
Bencom	Better Code 10 / 10
blockproof-back-end	Better Code 9 / 10
blockproof-business-network	Better Code 8 / 10
blockproof-front-end	Better Code 10 / 10
budi	Better Code 9 / 10
budi-api	Better Code 10 / 10
cgi	Better Code 7 / 10
consaint	Better Code 9 / 10
consense	Better Code 10 / 10

The current code quality scores for all teams were displayed as GitHub badges on a central scoreboard for all to see.

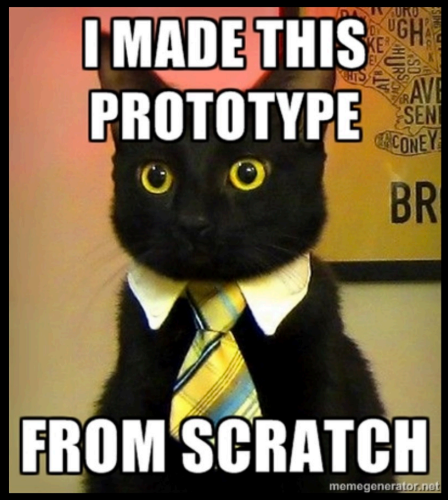
How to get a ‘Mature Prototype’ Certificate?

- If your main repo scores a Better Code Hub 10/10 you qualify to schedule a grilling session.
- Schedule the 10 minute slot for your session at the Jedi Hangout (limited slots available, so be quick!)
- Get Grilled in this interview (slots available Saturday 20.00-22.00 and Sunday 9.00-12.00)

Deliverable 1 of 4 - Working prototype in your Github account

Teams commit code for
- Better Code Hub
- (Continuous) feedback and ranking
- Looking under the hood (does it work?)
- Unlock “mature prototype status”

Audit by Software Improvement Group



Your IP is protected.
Read the General Participation Conditions
https://www.odyssey.org/general-conditions-for-participants-of-the-odyssey-hackathon-2019/

Appreciative Score Model

Used by the jury: maximum of 20 points per team

Quality of Prototype 0-5 points awarded for this quadrant	Impact 0-5 points awarded for this quadrant
21st Century Team & Solution 0-5 points awarded for this quadrant	Incubation plan 0-5 points awarded for this quadrant

Appreciative Model in detail - Quality of Prototype

Solution viability	How does the prototype demonstrate it can solve the described problem?
Technical difficulty	To what level is the technical approach innovative?
Blockchain and/or AI implementation in prototype	How are the benefits of blockchain and/or AI technology leveraged for this solution?
Compatibility	To what level are internet and industry standards used where relevant?
Maturity of Prototype	Has the team earned a star indicating they have a mature prototype?
Clean Code?	What is the Better Code Hub Score?

How to earn this star for Mature Prototype?

Qualify by scoring 10 out of 10 for your code quality?

Schedule your 10 minute Grilling Session at the Jedi Hangout

Earn your ‘Mature Code’ Star to proof to the jury you have a mature prototype

Don’t snooze - only limited spots available
Saturday 20.00-22.00
Sunday 9.00-12.00

What do you check during the Grilling Session?

Odyssey EASSI model	How mature is your prototype?
Existential	Why Blockchain? Why is Blockchain a good fit for your application, as opposed to a more traditional approach?
Architecture	How good is your system architecture? SIG Software engineering system characteristics.
Scalability	Will it scale, what are the bottlenecks? This is what every investor wants to know!
Secuirty	Encryption (private) key storage. How do you encourage/enable your users to safely score their private key?
Inertia	Number of Nodes running the BC. How to make it interesting to join the chain.

Code Coaches JEDI

Jan Laan • 1st

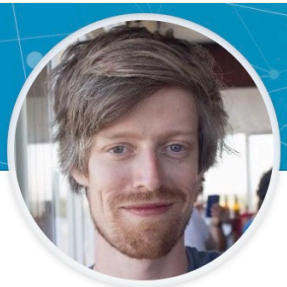
Software Security Consultant at Software Improvement Group (SIG)

Reinier Vis • 1st

Head of Software Consultants at Software Improvement Group (SIG)

Bugra M. Yildiz • 1st

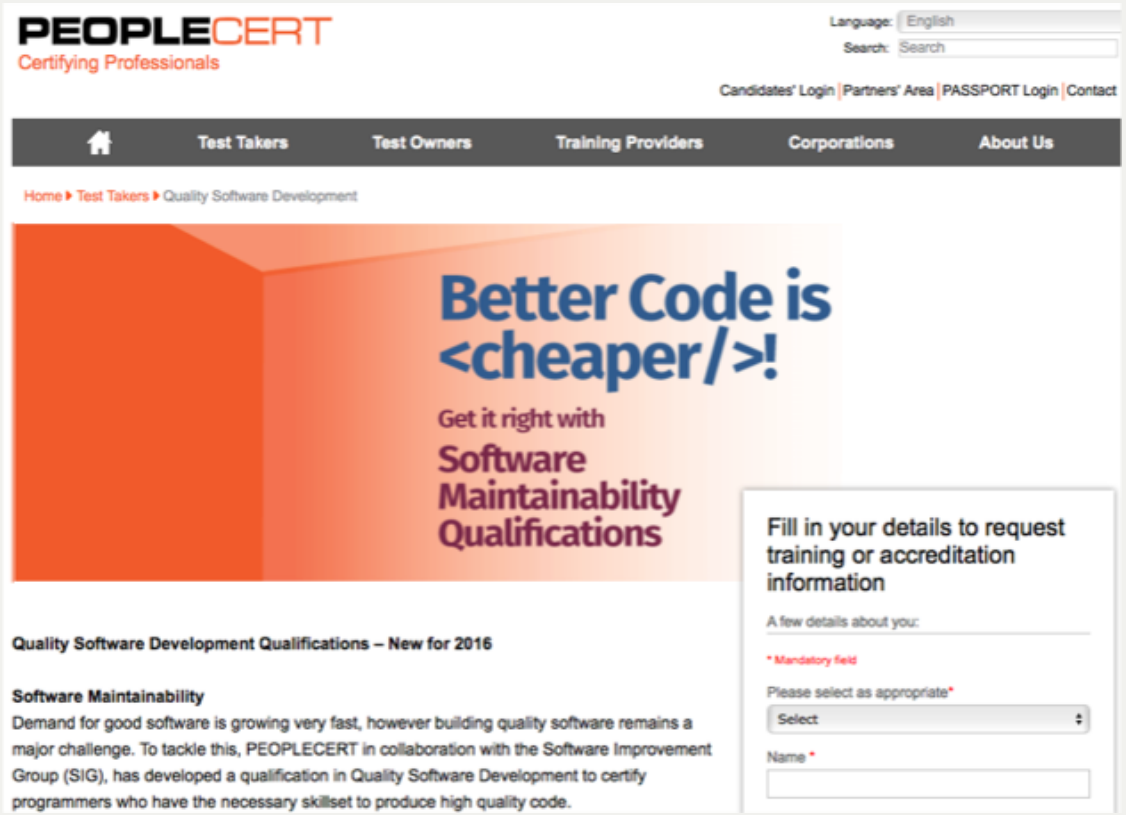
Software Consultant at Software Improvement Group (SIG), Software Engineer



Hugo Schoonewille • 1st

Software Consultant at Software Improvement Group (SIG)

Professional certification for software maintainability



Roles in academic education

Software Engineering

Guest lecturing & courses:

- Open Code Clinic @ TU Delft
- Make IT Work program @ HvA
- Master Software Engineering @ UvA
- Courses at Radboud University Nijmegen
- Guest lectures at Hanze Hogeschool
- ...

Software Improvement Group

Contact

+31 20 314 09 50

m.cuijpers@sig.eu

@sig_eu