Foundations of Software Engineering

Jean-Rémy Falleri

Foundations of Software Engineering

- **Prerequisite**
  - Object oriented programming
  - Programmation mobile
  - Project management
- **We learn**
  - Main concepts and activities involved in software development
  - Some good practices
- **We do not learn**
  - A magical formula to create perfect software systems
  - Advanced tools for requirements, architecture or design
- **Notation**
  - Practical / Project
- **Organisation**
  - 9 lectures of 2h40 and 1 of 1h20

Introduction

Prelude: once in a lab

Jean-Rémy Falleri

Associate Professor at ENSEIRB-Matmeca

Head of the Group

Thomas Degueule

Full time CNRS researcher

Xavier Blanc

Full Professor at Université de Bordeaux

Head of Progress Group

Laurent Réveillère

Full Professor at Université de Bordeaux

Floréal Morandat

Associate Professor at ENSEIRB-Matmeca

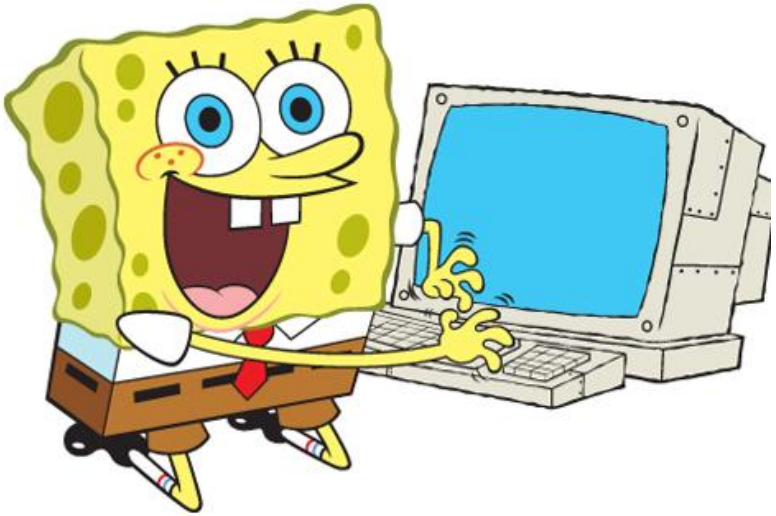
Joachim Bruneau-Queyreix

Associate Professor at ENSEIRB-Matmeca

Assisting developers to craft complex software systems

What is a software project

A vague idea in someone's mind



A functioning software system

The many stakeholders of a software project

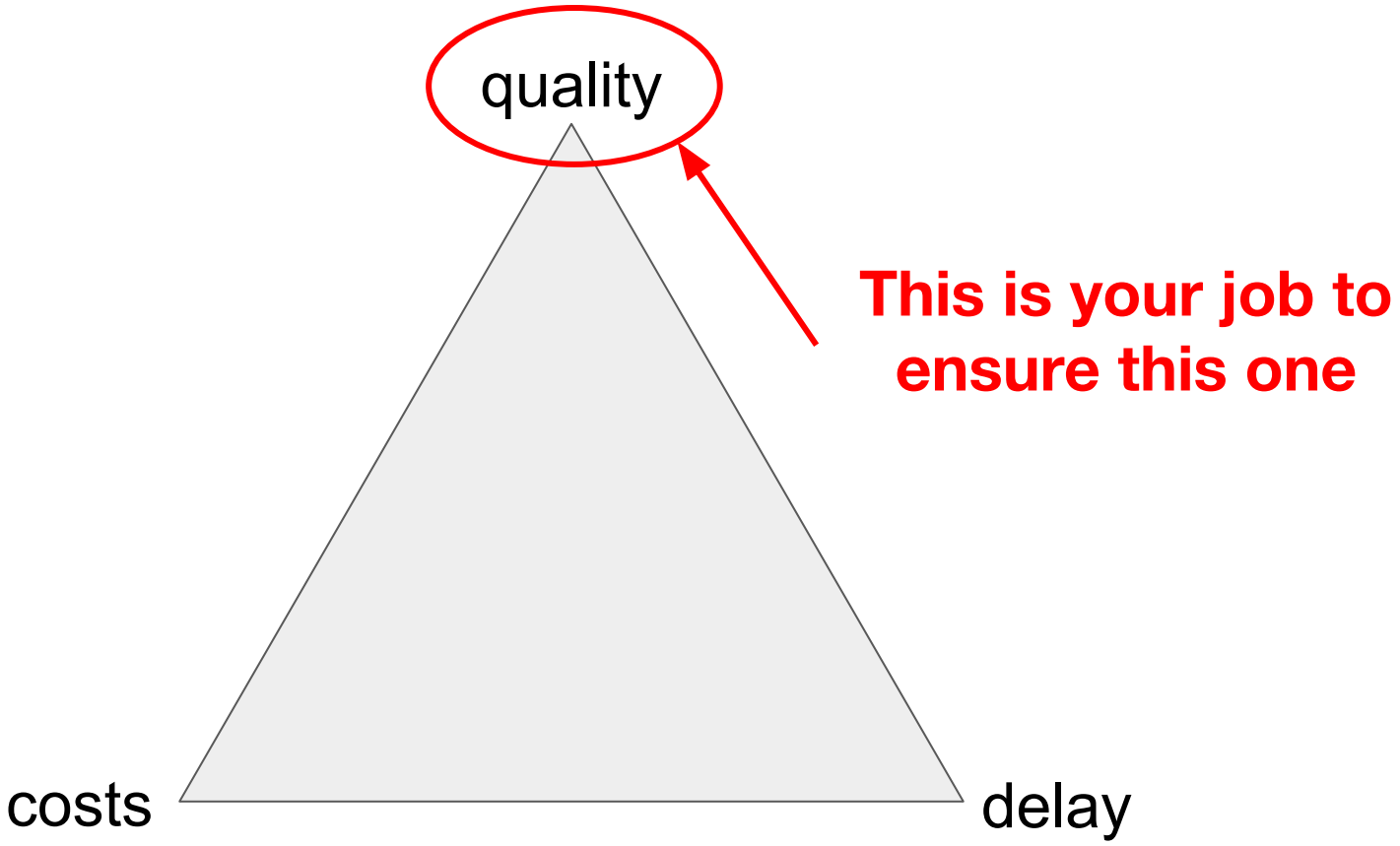
Users

Contractor

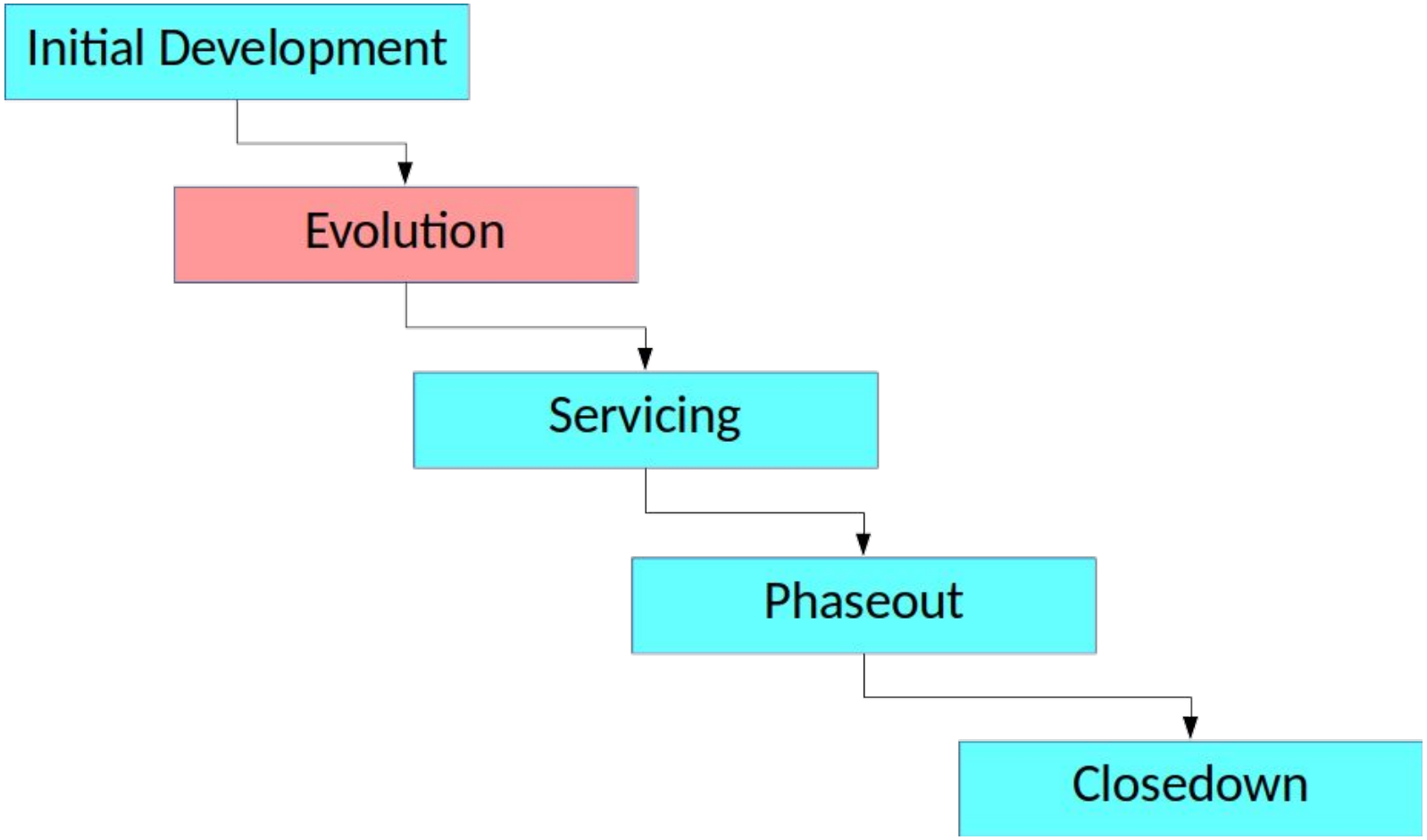
Client

You work here

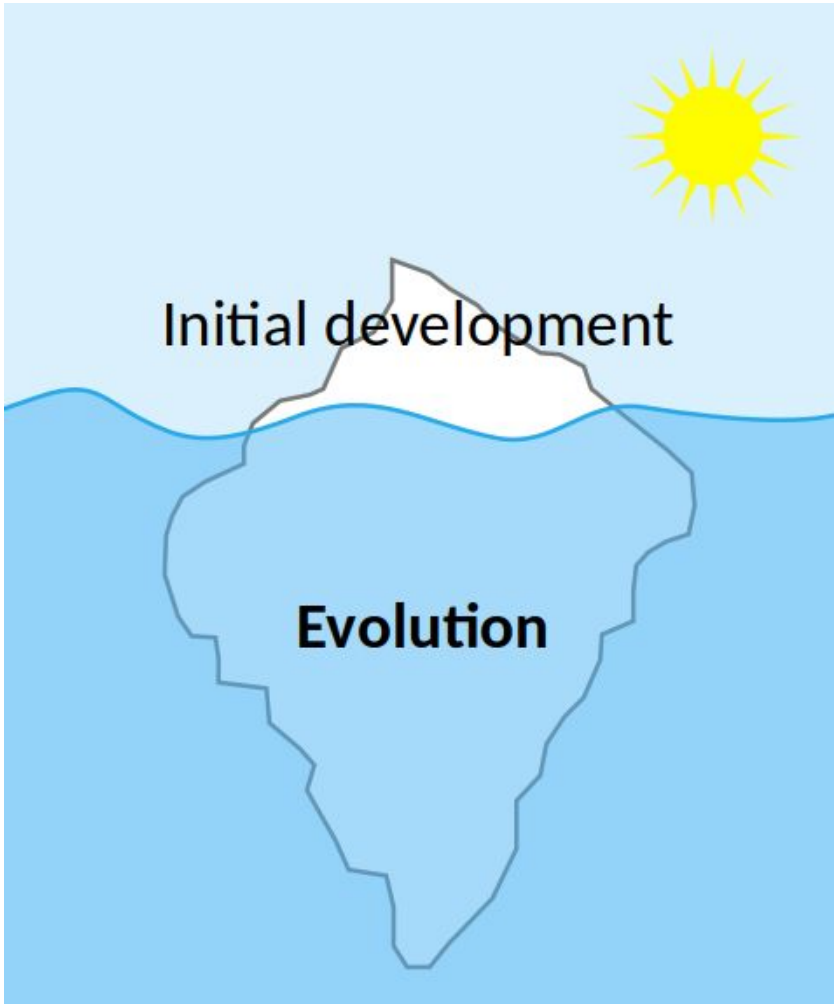
The main constraints of a software project



Life and death of a software project



The main phases' expenses



Success and failures in the software industry

Paradigm	Successful	Challenged	Failed
Lean	72%	21%	7%
Agile	64%	30%	6%
Iterative	65%	28%	7%
Ad hoc	50%	35%	15%
Traditional	49%	32%	18%

Dr Dobbs : The Non-Existent Software Crisis: Debunking the Chaos Report

Bad software can lead to dire consequences

Therac-25

The six documented accidents occurred when the high-current electron beam generated in X-ray mode was delivered directly to patients. Two software faults were to blame.[3] One, when the operator incorrectly selected X-ray mode before quickly changing to electron mode, which allowed the electron beam to be set for X-ray mode without the X-ray target being in place. A second fault allowed the electron beam to activate during field-light mode, during which no beam scanner was active or target was in place.

Previous models had hardware interlocks to prevent such faults, but the Therac-25 had removed them, depending instead on software checks for safety.

The high-current electron beam struck the patients with approximately 100 times the intended dose of radiation, and over a narrower area, delivering a potentially lethal dose of beta radiation. The feeling was described by patient Ray Cox as "an intense electric shock", causing him to scream and run out of the treatment room.[4] Several days later, radiation burns appeared, and the patients showed the symptoms of radiation poisoning; in three cases, the injured patients later died as a result of the overdose.[5]

Ariane 5

Ariane 5's first test flight (Ariane 5 Flight 501) on 4 June 1996 failed, with the rocket self-destructing 37 seconds after launch because of a malfunction in the control software.[35] A data conversion from 64-bit floating point value to 16-bit signed integer value to be stored in a variable representing horizontal bias caused a processor trap (operand error)[36] because the floating point value was too large to be represented by a 16-bit signed integer. The software was originally written for the Ariane 4 where efficiency considerations (the computer running the software had an 80% maximum workload requirement[36]) led to four variables being protected with a handler while three others, including the horizontal bias variable, were left unprotected because it was thought that they were "physically limited or that there was a large margin of safety".[36] The software, written in Ada, was included in the Ariane 5 through the reuse of an entire Ariane 4 subsystem despite the fact that the particular software containing the bug, which was just a part of the subsystem, was not required by the Ariane 5 because it has a different preparation sequence[36] than the Ariane 4.

Boeing 737-MAX

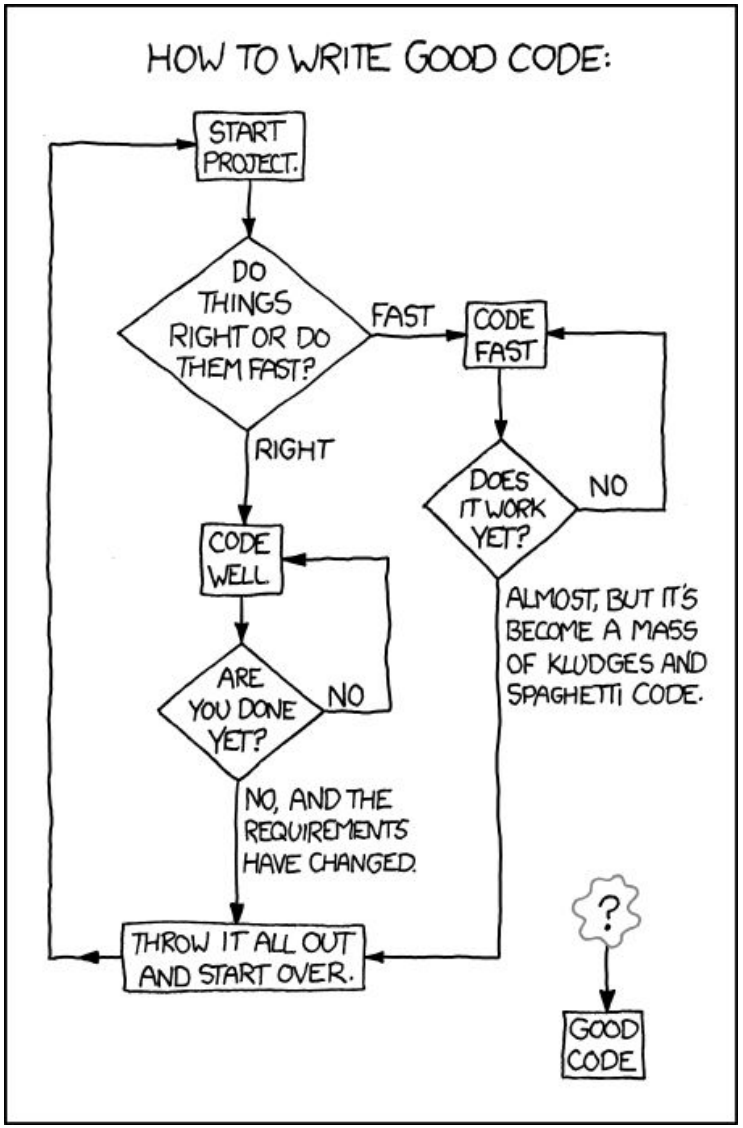
MCAS on the 737 MAX was designed to mimic the pitching behavior of the previous generation of the series, the Boeing 737 NG, by pushing down the aircraft nose from an elevated angle of attack (AoA) by automatically adjusting the horizontal stabilizer and trim tab. The system was intended to protect pilots from inadvertently flying at too steep an angle, which could result in a stall. Boeing, however, asserted that MCAS was not an anti-stall system, as the media widely reported it to be. Pilot movement of the control column on the MAX did not disable MCAS, unlike an earlier implementation of MCAS on the U.S. Air Force Boeing 767 Tanker. During certification of the MAX, Boeing requested and received permission from the FAA to remove a description of MCAS from the aircraft manual, leaving pilots unaware of the system when the airplane entered service in 2017.[102][103] Boeing had also knowingly withheld knowledge, for at least a year before the Lion Air crash, that a system to warn of a possible AoA malfunction did not work as advertised.[104]

On November 6, 2018, Boeing published a supplementary service bulletin prompted by the first crash. The bulletin describes warnings triggered by erroneous AoA data could cause the pitch trim system to repeatedly push down the nose of the airplane and referred pilots to a "non-normal runaway trim" procedure as resolution, specifying a narrow window of a few seconds before the system would reactivate and pitch the nose down again.[105] The FAA issued an emergency airworthiness directive, 2018-23-51, on November 7, 2018, requiring the bulletin's inclusion in the flight manuals, and that pilots immediately review the new information provided.[106][107] Pilots wanted to know more about the issue, and Boeing responded by publicly naming MCAS for the first time in another message to airlines, noting that MCAS operates "without pilot input."[108][109]

In December 2018, the FAA had privately predicted that 15 MCAS-related accidents could result if the system was not redesigned. Boeing said it would revise MCAS software by April 2019 to correct any problems. The study was only revealed a year later at the December 2019 House of Representatives hearing. Stephen Dickson, who became FAA administrator during the accident investigations, testified at the hearing about his agency's response after the Lion Air accident, saying "the result was not satisfactory".[110]

Take home message

It's your job to **make sure the software system is built right** and **you don't want to improvise**

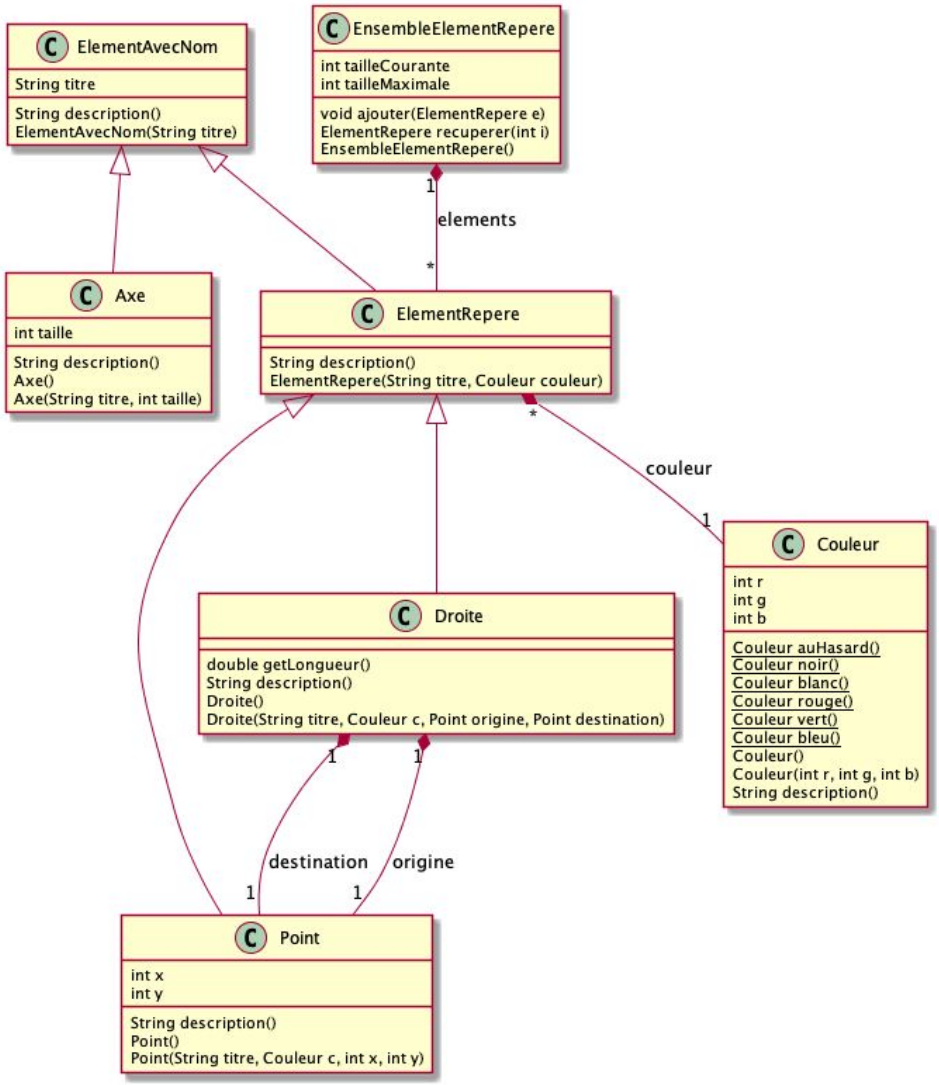


Main activities' overview

Requirements

Title:	Customer Inter Account Transfer
Value Statement:	As a bank customer, I want to transfer funds between my linked accounts, So that I can fund my credit card.
Acceptance Criteria:	*Acceptance Criterion 1:* Given that the account is has sufficient funds When the customer requests an inter account transfer Then ensure the source account is debited AND the target account is credited. *Acceptance Criterion 2:* Given that the account is overdrawn, When the customer requests an inter account transfer Then ensure the rejection message is displayed And ensure the money is not transferred.
Definition of Done:	• Unit Tests Passed • Acceptance Criteria Met • Code Reviewed • Functional Tests Passed • Non-Functional Requirements Met • Product Owner Accepts User Story
Owner:	MR I Owner
Iteration:	Unscheduled
Estimate:	5 Points

Architecture and design



Implementation

```
     * This is useful if kernel is booting in an unreliable e
147  * For ex. kdump situaiton where previous kernel has cras
148  * skipped and devices will be in unknown state.
149  */
150 unsigned int reset_devices;
151 EXPORT_SYMBOL(reset_devices);
152
153 static int __init set_reset_devices(char *str)
154 {
155     reset_devices = 1;
156     return 1;
157 }
158
159 __setup("reset_devices", set_reset_devices);
160
161 static const char * argv_init[MAX_INIT_ARGS+2] = { "init"
162 const char * envp_init[MAX_INIT_ENVS+2] = { "HOME=/", "TE
163 static const char *panic_later, *panic_param;
164
165 extern const struct obs_kernel_param __setup_start[], __s
166
167 static int __init obsolete_checksetup(char *line)
168 {
169     const struct obs_kernel_param *p;
        int had_early_param = 0;
171
172     p = __setup_start;
173     do {
174         int n = strlen(p->str);
175         if (parameqn(line, p->str, n)) {
```

Documentation

SENDING MESSAGES

Sending an SMS or MMS is one of the most common tasks performed on the Twilio Platform. Sending a message is as simple as POSTing to the Messages resource. We'll outline required and optional parameters, messaging services, alphanumeric sender ID, rate limiting, and handling message replies below.

HTTP POST to Messages

To send a new outgoing message, make an HTTP POST to your Messages list resource URI:

```
/2010-04-01/Accounts/{AccountSid}/Messages
```

POST Parameters

Required Parameters

The `To` parameter is **required** in your POST to send the message:

PARAMETER	DESCRIPTION
To	The destination phone number. Format with a '+' and country code e.g., +16175551212 (E.164 format).

Send a message with an image URL

NODE.JS ⌄ CODE OUTPUT

SDK Version: 2.x 3.x

```
1  // Download the Node helper library from twilio.com/docs/node/install
2  // These vars are your accountSid and authToken from twilio.com/user/
3  var accountSid = 'ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX';
4  var authToken = "your_auth_token";
5  var client = require('twilio')(accountSid, authToken);
6
7  client.messages.create({
8      body: "Jenny please?! I love you <3",
9      to: "+15558675309",
10     from: "+14158141829",
11     mediaUrl: "http://www.example.com/hearts.png"
12 }, function(err, message) {
13     process.stdout.write(message.sid);
14 });
```

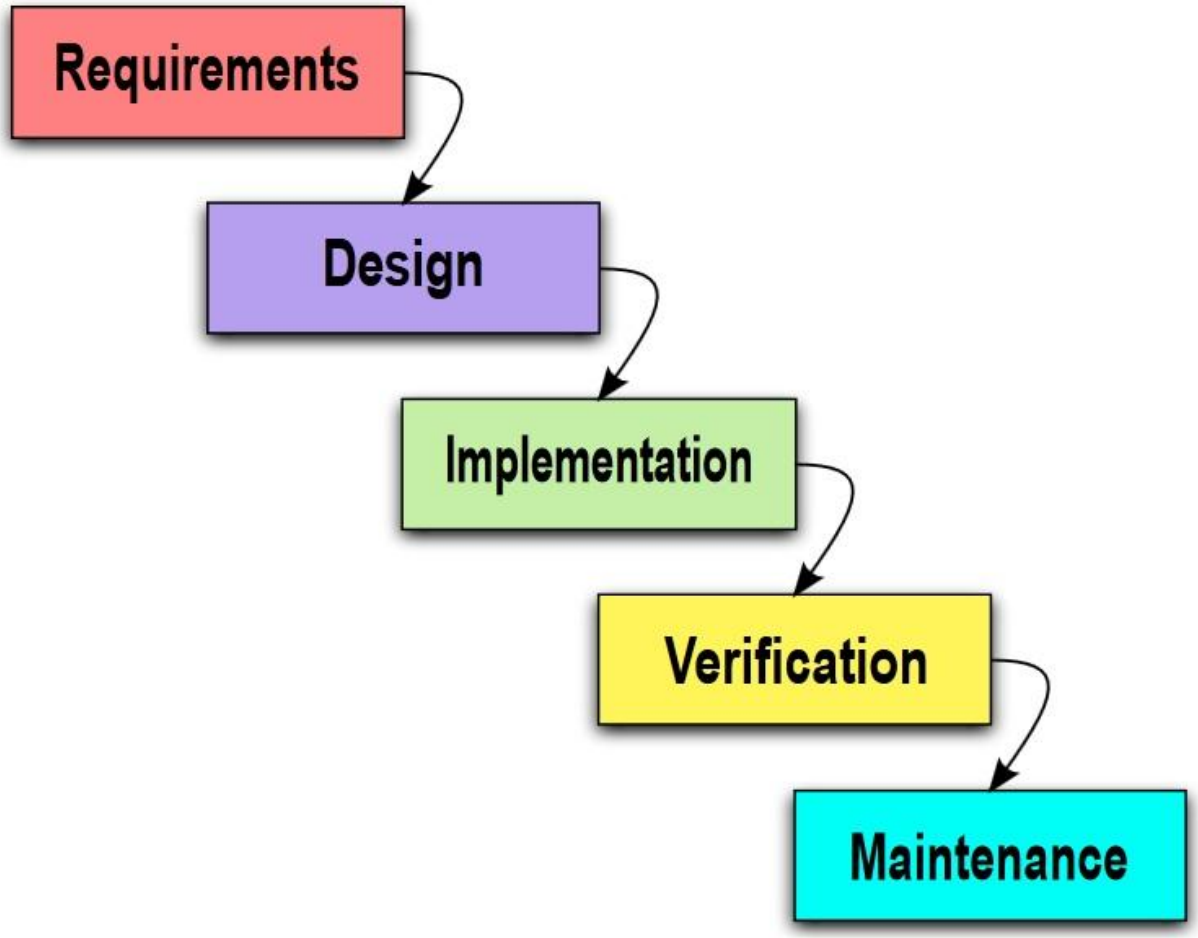
Validation

```
test "error access is indifferent" do
  errors = ActiveModel::Errors.new(self)
  errors[:foo] << "omg"

  assert_equal ["omg"], errors["foo"]
end

test "values returns an array of messages" do
  errors = ActiveModel::Errors.new(self)
  errors.messages[:foo] = "omg"
  errors.messages[:baz] = "zomg"

  assert_equal ["omg", "zomg"], errors.values
end

test "values returns an empty array after try to get a message only" do
  errors = ActiveModel::Errors.new(self)
  errors.messages[:foo]
  errors.messages[:baz]

  assert_equal [], errors.values
end

test "keys returns the error keys" do
  errors = ActiveModel::Errors.new(self)
  errors.messages[:foo] << "omg"
  errors.messages[:baz] << "zomg"

  assert_equal [:foo, :baz], errors.keys
end
```

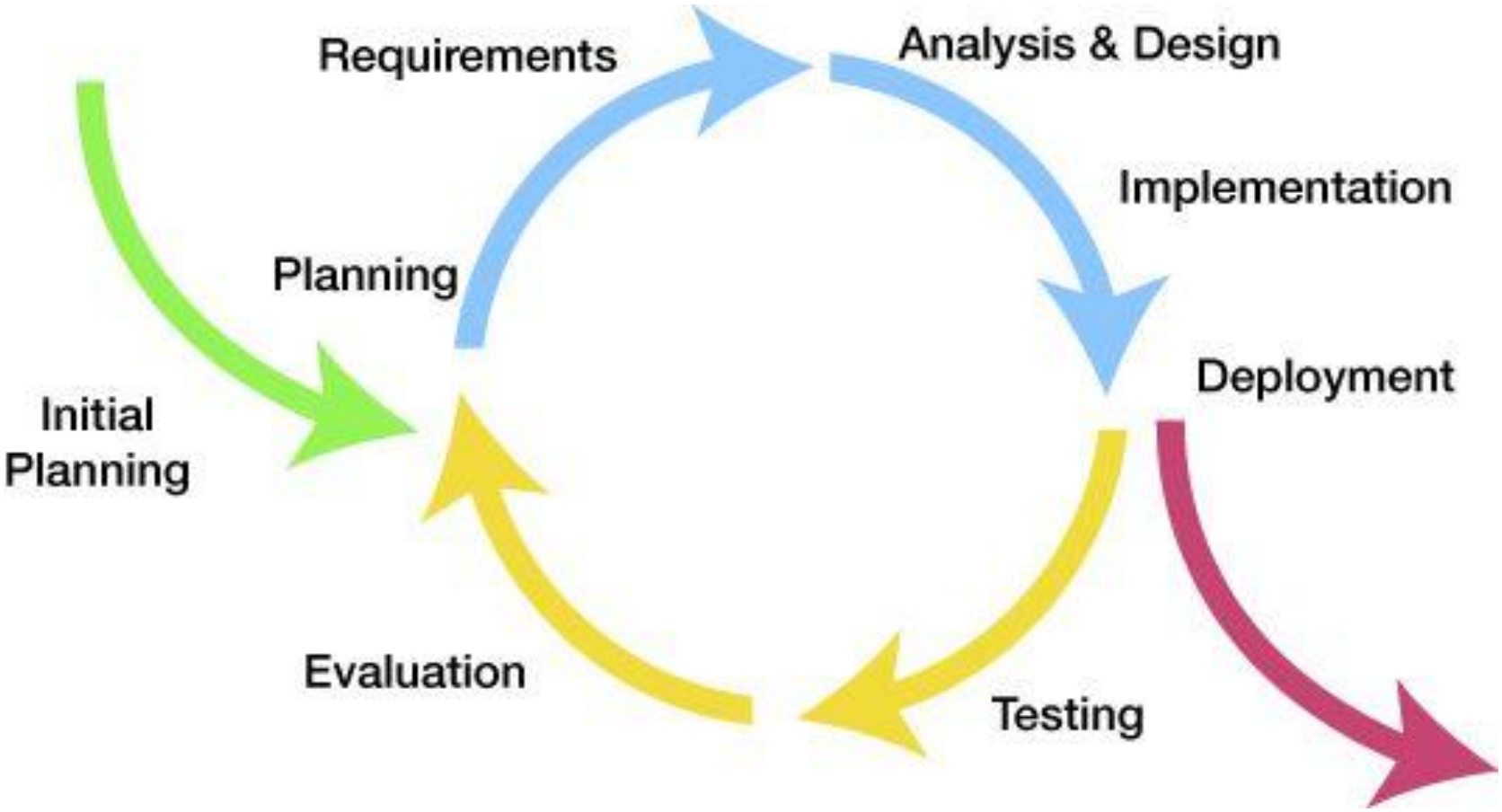
Deployment


THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."
HEY! GET BACK TO WORK!
COMPILING!
OH. CARRY ON.

Processes : big design up-front

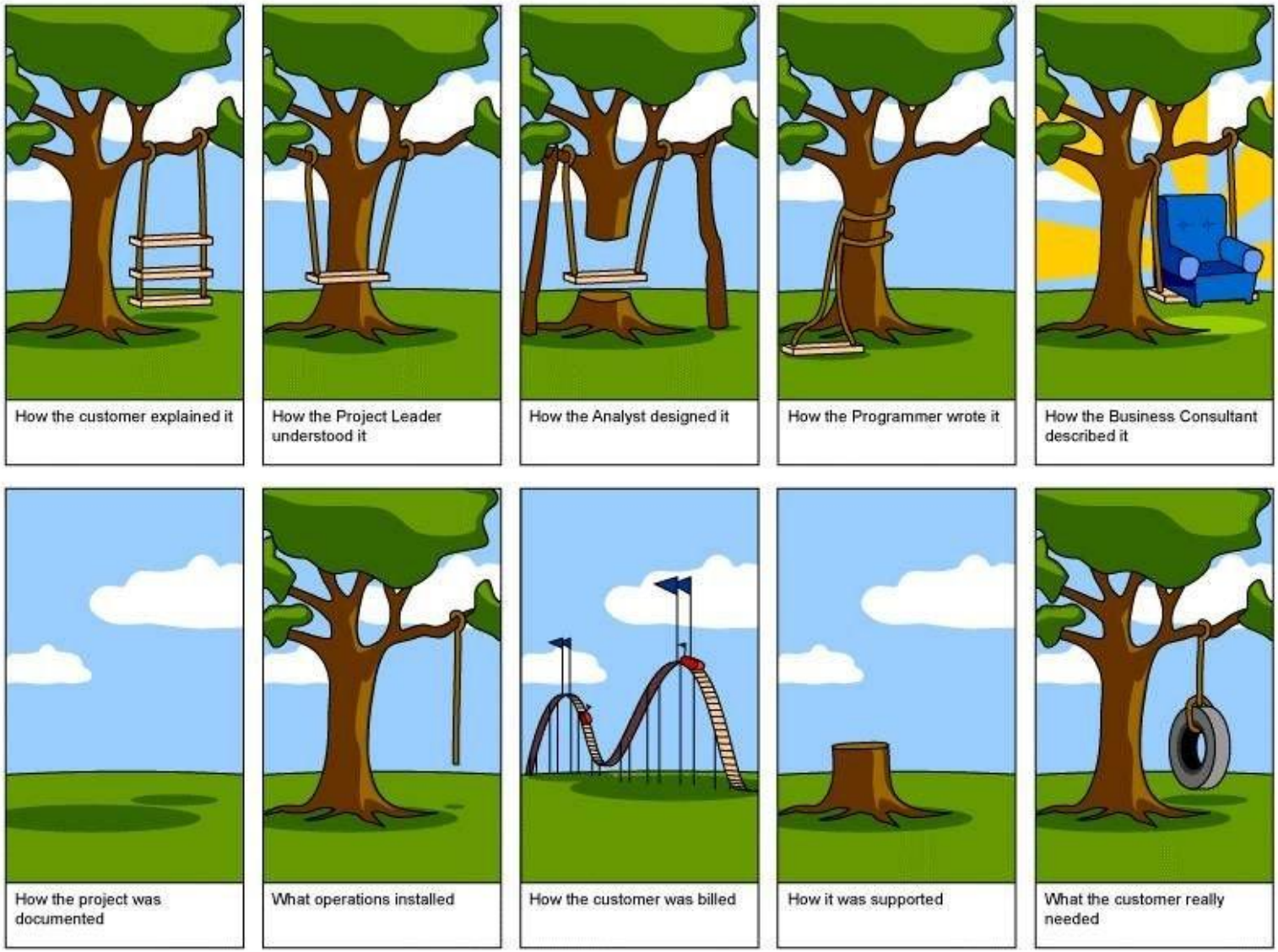


Processes: iterative style



Requirements, in a nutshell

Requirements importance in one comic strip

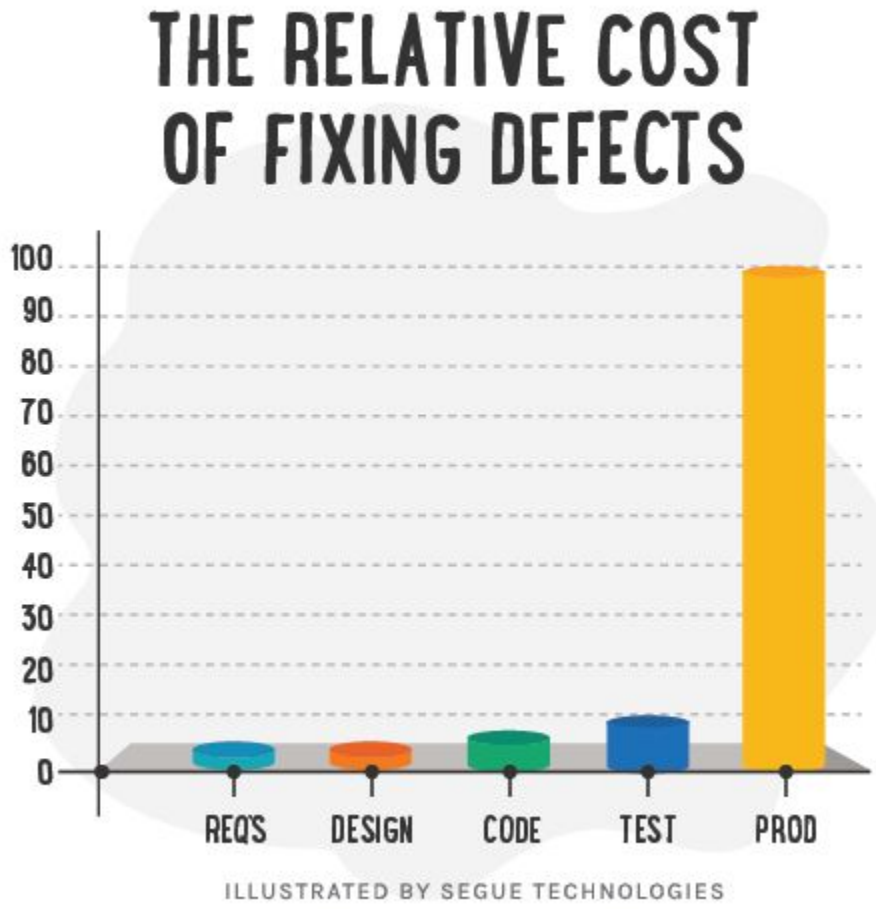


Requirements importance, revisited

Frederick Brooks "No Silver Bullet: Essence and Accidents of Software Engineering" (1987)

The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.

The mythical cost of defect by phases



Recommended book

"Hello, Phil? This is Maria in Human Resources. We're having a problem with the personnel system you programmed for us. An employee just changed her name to Sparkle Starlight, and we can't get the system to accept the name change. Can you help?"

"She married some guy named Starlight?"

"No, she didn't get married, just changed her name," Maria replied. "That's the problem. It looks like we can change a name only if someone's marital status changes."

"Well, yeah, I never thought someone might just change her name. I don't remember you telling me about this possibility when we talked about the system," Phil said.

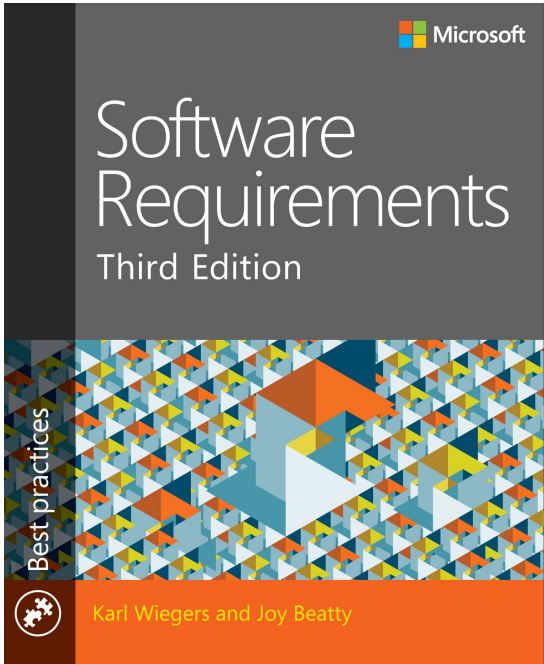
"I assumed you knew that people could legally change their name anytime they like," responded Maria. "We have to straighten this out by Friday or Sparkle won't be able to cash her paycheck. Can you fix the bug by then?"

"It's not a bug!" Phil retorted. "I never knew you needed this capability. I'm busy on the new performance evaluation system. I can probably fix it by the end of the month, but not by Friday. Sorry about that. Next time, tell me these things earlier and please write them down."

"What am I supposed to tell Sparkle?" demanded Maria. "She'll be upset if she can't cash her check."

"Hey, Maria, it's not my fault," Phil protested. "If you'd told me in the first place that you had to be able to change someone's name at any time, this wouldn't have happened. You can't blame me for not reading your mind."

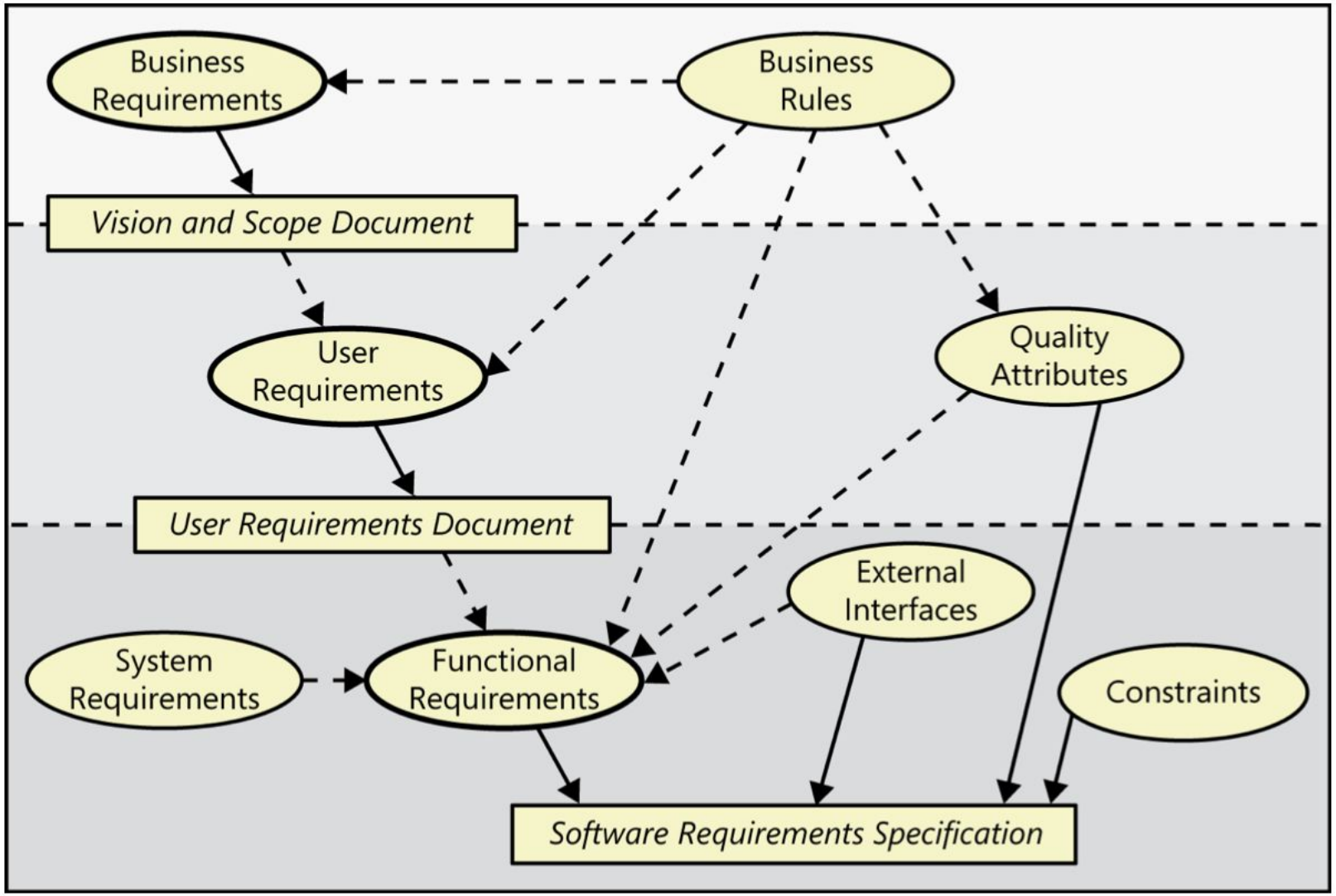
Angry and resigned, Maria snapped, "Yeah, well, this is the kind of thing that makes me hate computers. Call me as soon as you get it fixed, will you?"



What is a requirement?

Ian Sommerville and Pete Sawyer (1997)
Requirements are a specification of what should be implemented. They are descriptions of how the system should behave, or of a system property or attribute. They may be a constraint on the development process of the system.

Broad kind of requirements



Gathering requirements

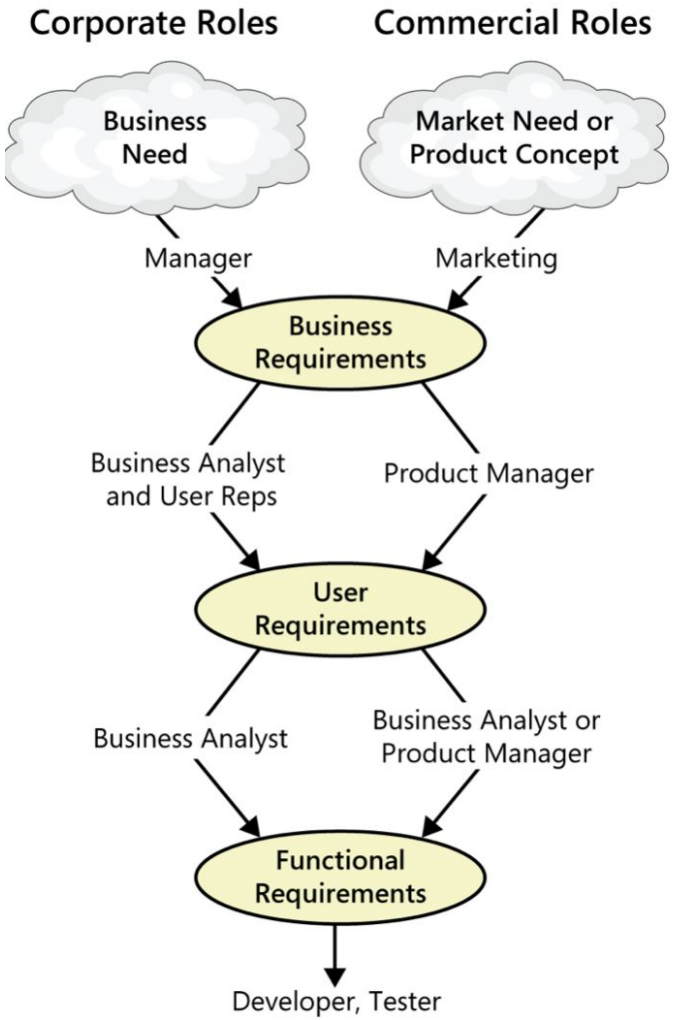


FIGURE 1-3 An example of how different stakeholders participate in requirements development.

Working with requirements

- Developing requirements
  - Elicitation (discovering requirements)
  - Analysis (cleaning and ranking requirements)
  - Specification (organizing the requirements)
  - Validation (ensuring the requirements are correct)
- Maintaining requirements
  - Managing change and tracing of requirements in the system

User stories' simple template

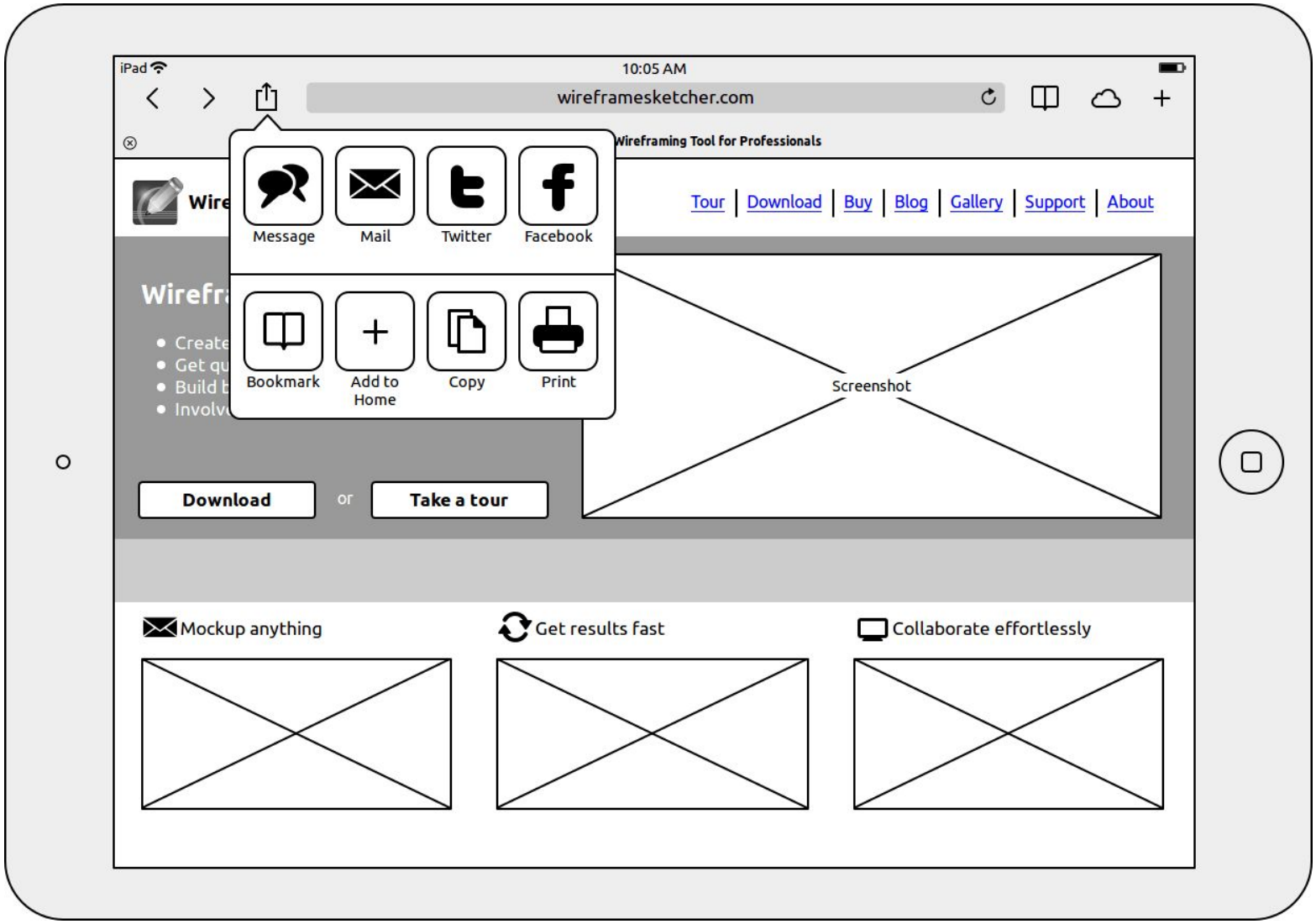
ID	DESCRIPTION	VALUE	COST	ACCEPTANCE
US1	***As a*** *registered* user ***I want to*** log in using a mail and a password ***so that I can*** become an authenticated user	HIGH	1 day	**Given** a registered user **When** I supply my email/password in the login form **Then** I become authenticated
US2	***As a*** *registered* user ***I want to*** be able to recover my password by providing my e-mail ***so that I can*** recover access to the application	MED	1 day	**Given** a registered user **When** I supply my email **Then** I receive my password

Good practices

Follow the INVEST guidelines for good user stories!

one 80

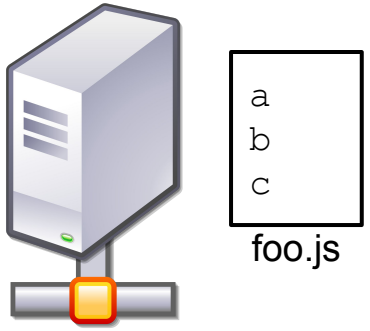
Mockups


iPad
10:05 AM
wireframesketcher.com
Wireframing Tool for Professionals
Message
Mail
Twitter
Facebook
Bookmark
Add to Home
Copy
Print
Tour | Download | Buy | Blog | Gallery | Support | About
Screenshot
Download
or
Take a tour
Mockup anything
Get results fast
Collaborate effortlessly

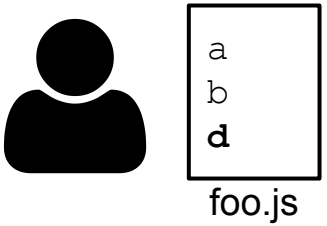
Implementation

Code sharing

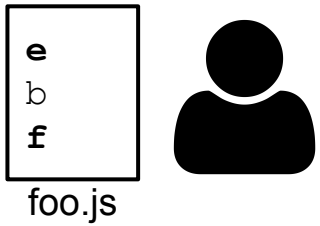
Code sharing



① **Bob** opens the file at 11h50
and saves it at 11h55



② **Alice** opens the file at 11h52
and saves it at 12h00



Bob loses its work!

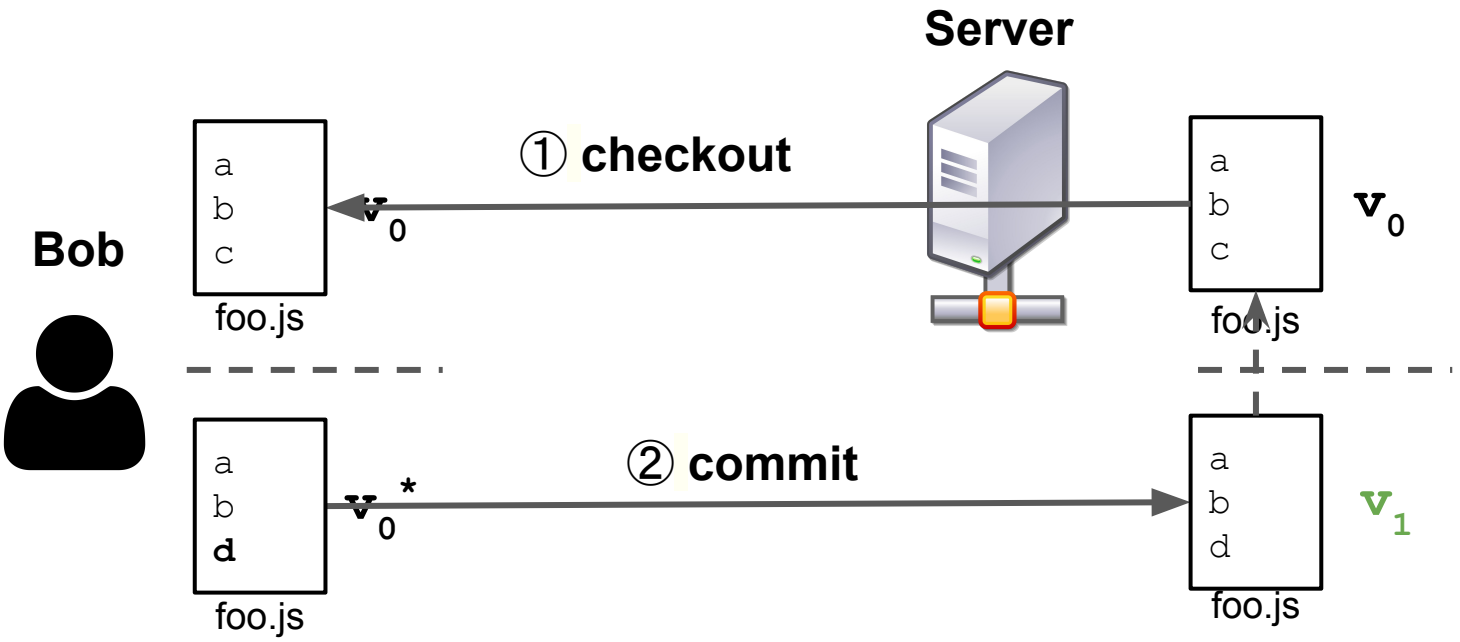
Working on shared resources

Two conceptual models

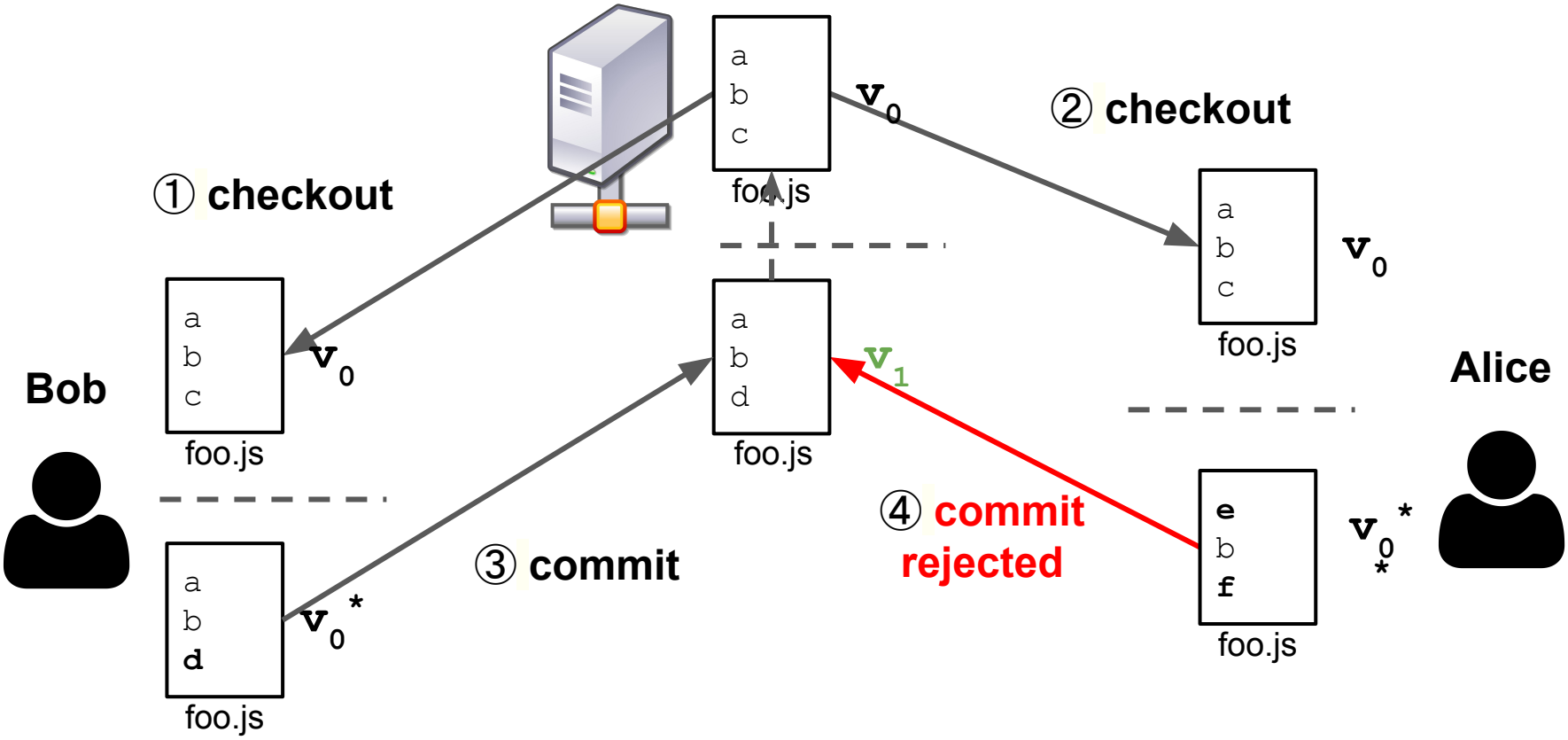
- **Lock / Edit / Unlock**
  - 😃 When locked, you can save at will
  - 😢 Not possible to concurrently edit a resource
  - 😢 You have to unlock
- **Copy / Edit / Merge**
  - 😃 Possible to concurrently edit a resource
  - 😢 Saving your work can incur a merge

Code sharing tools usually use the Copy/Edit/Merge model

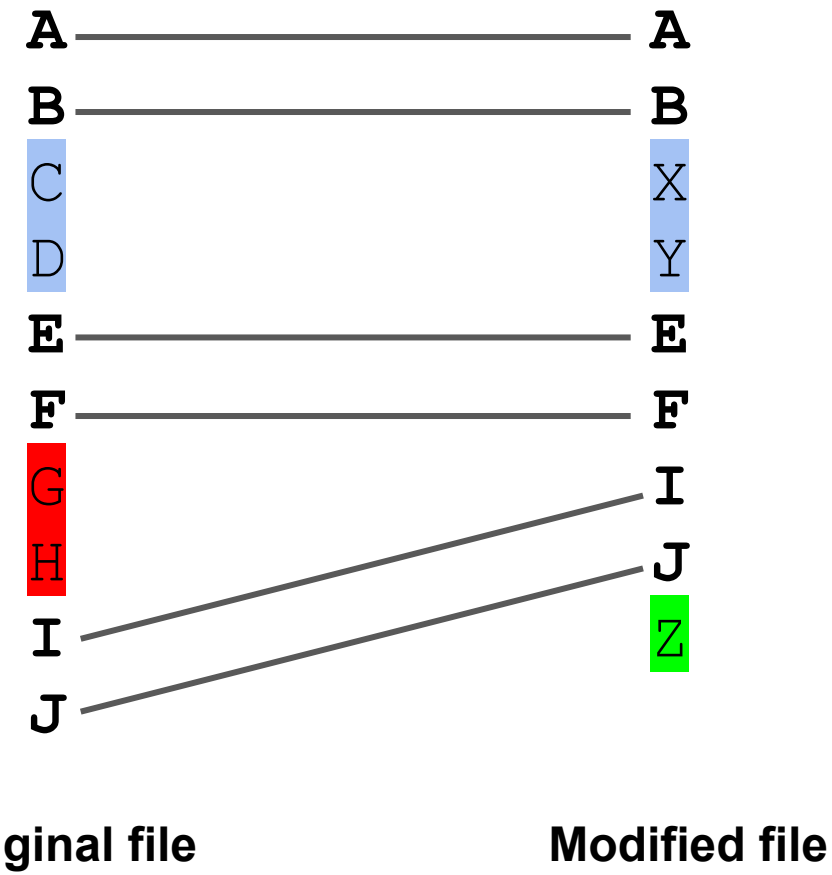
Centralized version control



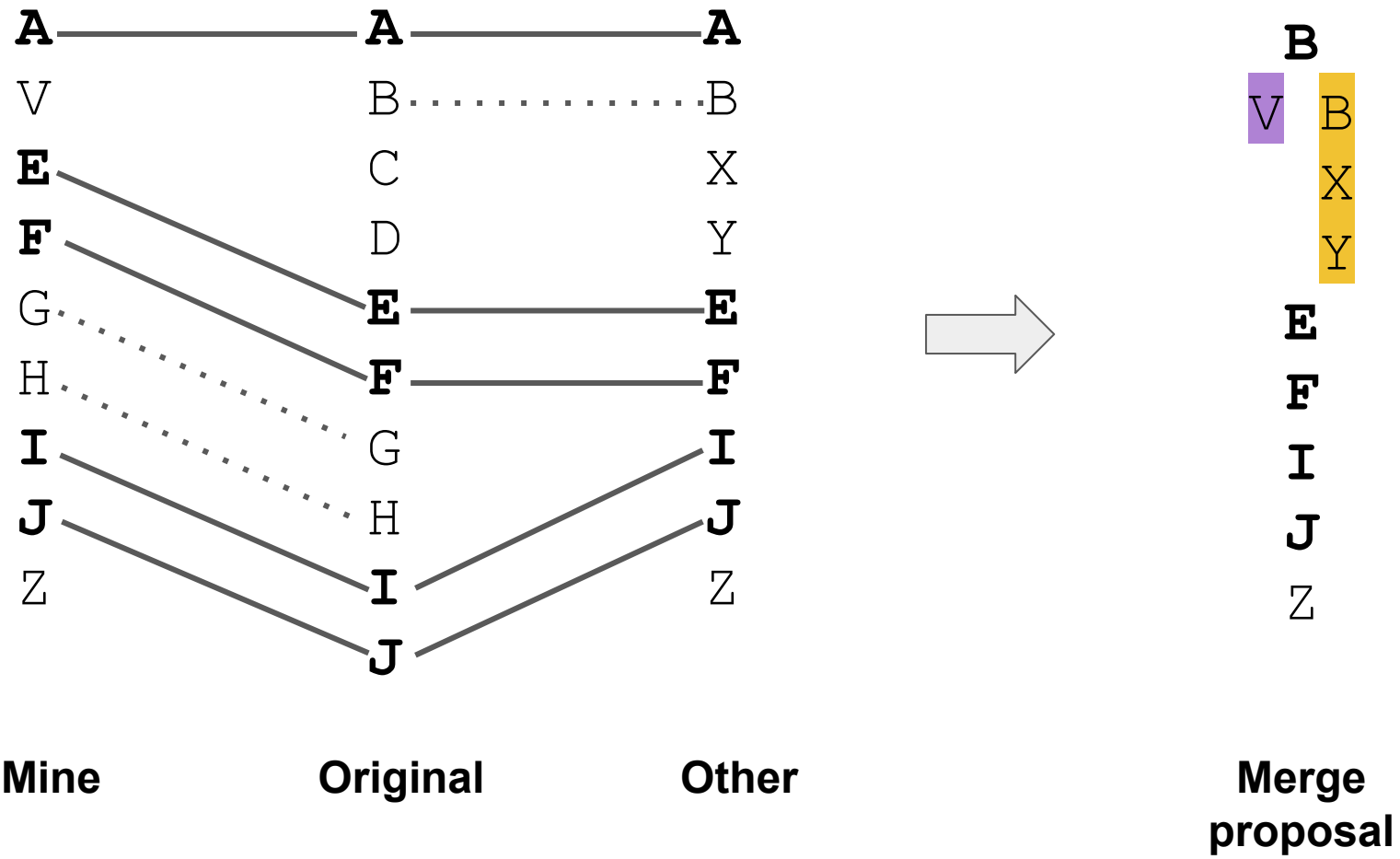
Centralized version control



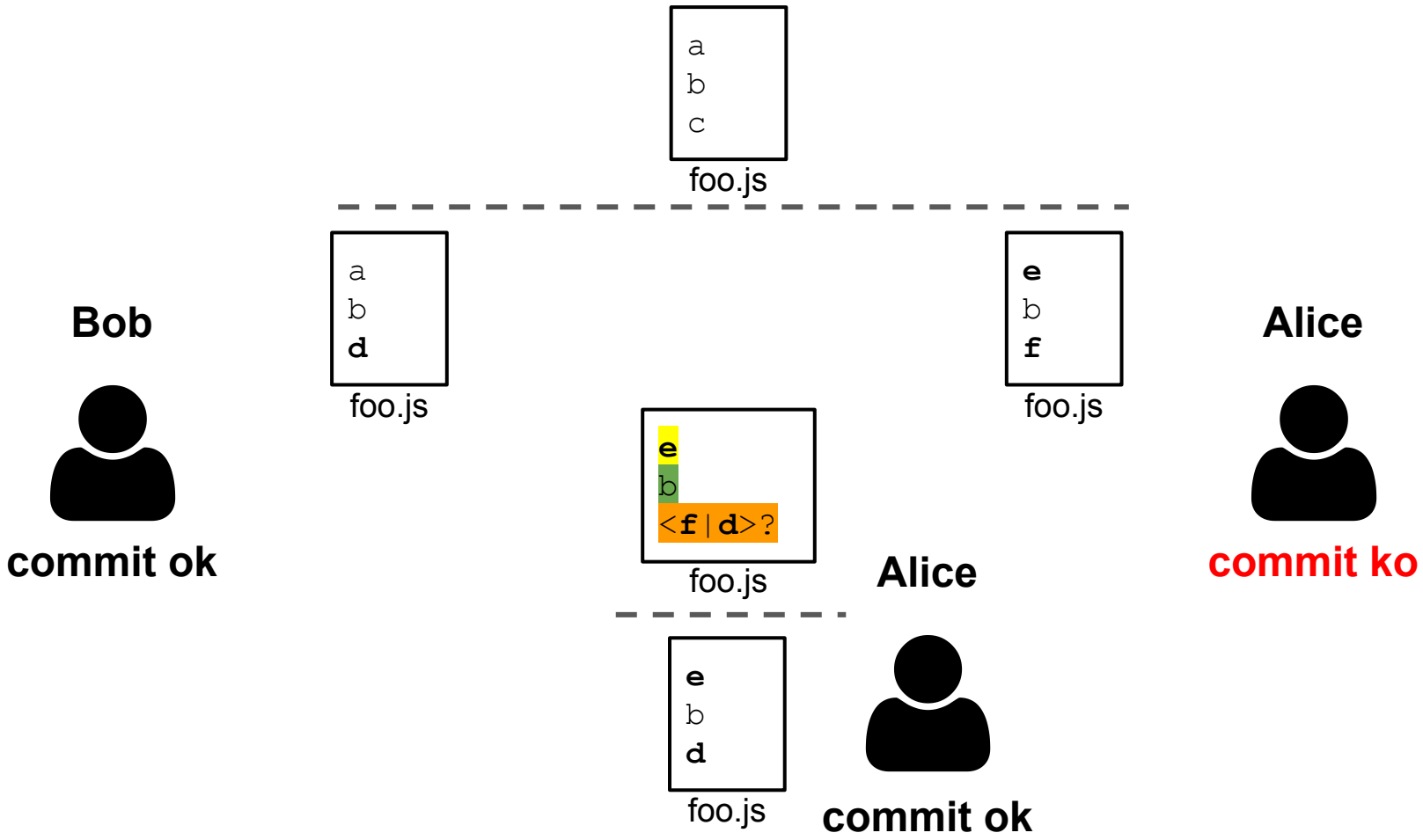
Computing a diff



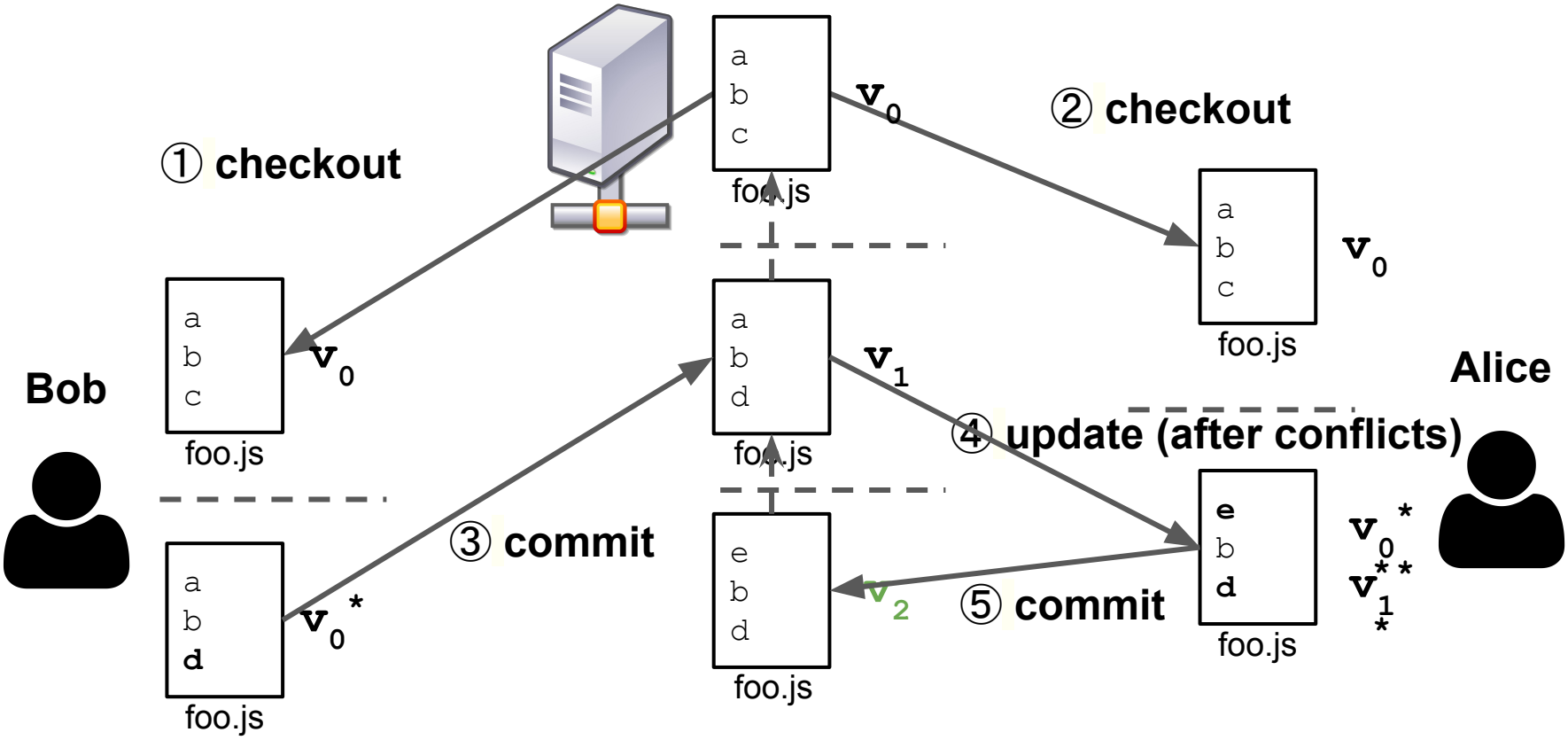
Conflict detection



Conflict handling



Centralized version control

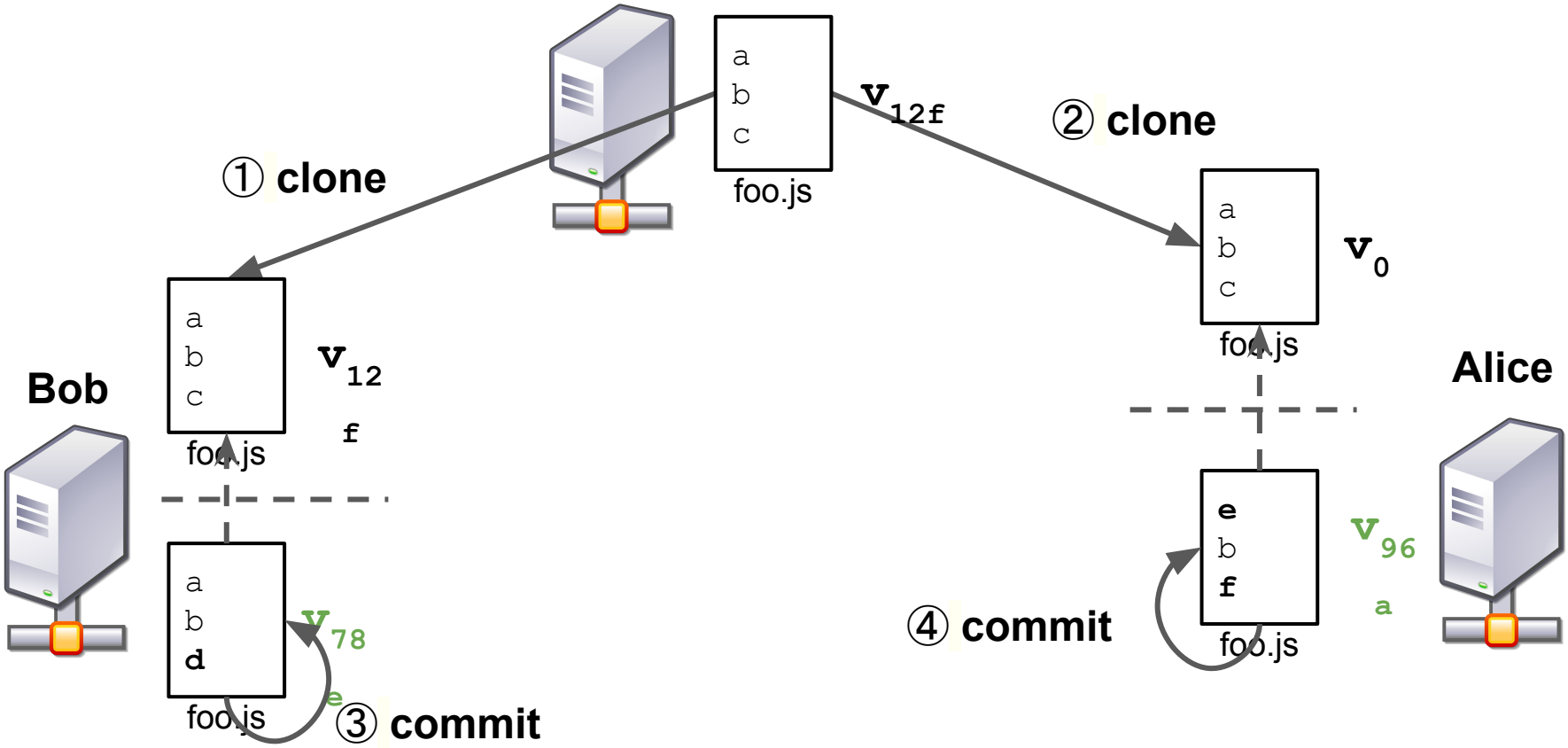


Pray that Bob didn't do anything during the merge 😈

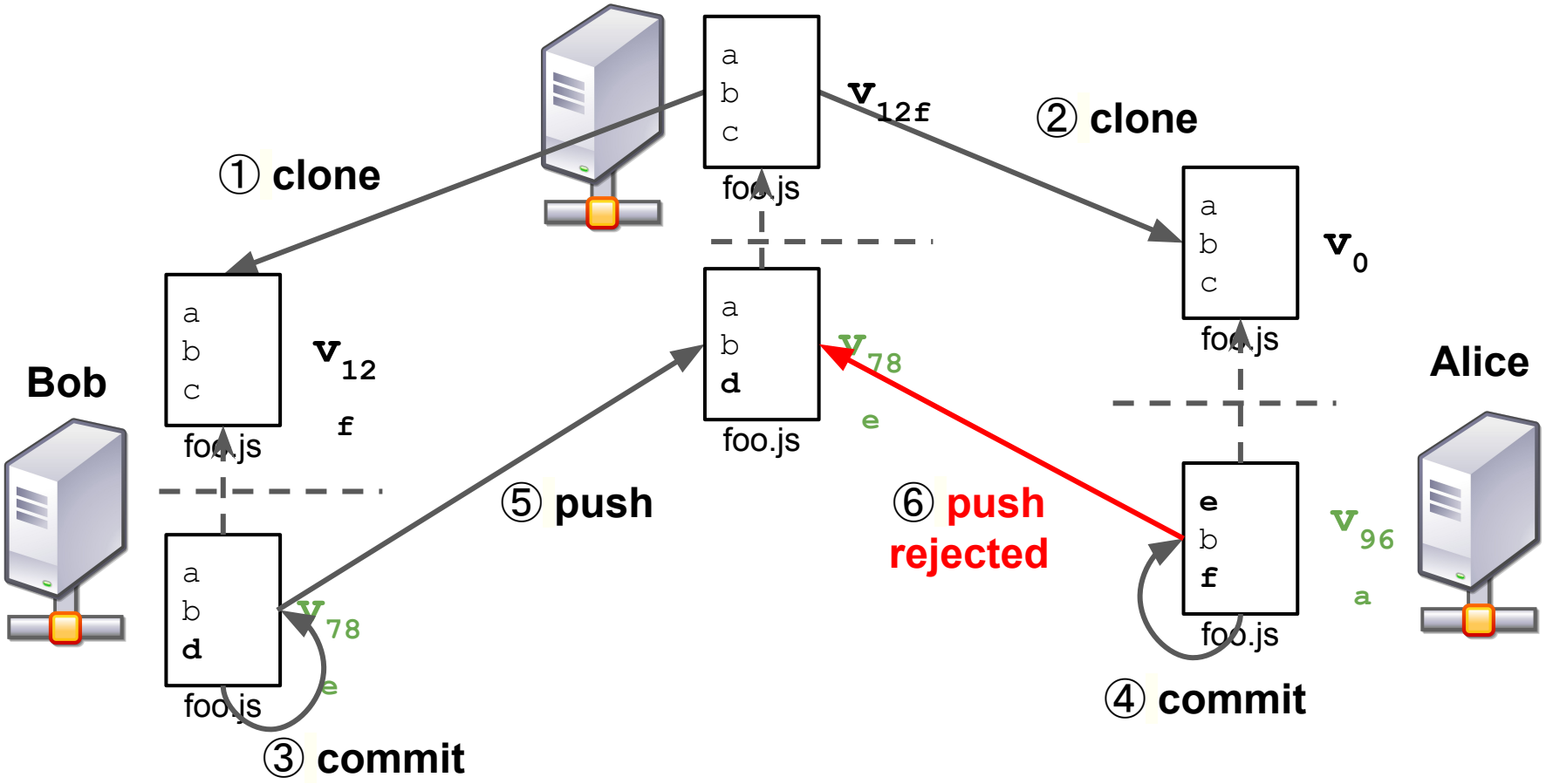
Limitations

- 😢 You can commit only when connected to the server
- 😢 Friday afternoon commit
- 😢 You can have a lot of work when only wanting to submit your changes
- 😢 Opportunistic commit to avoid merges

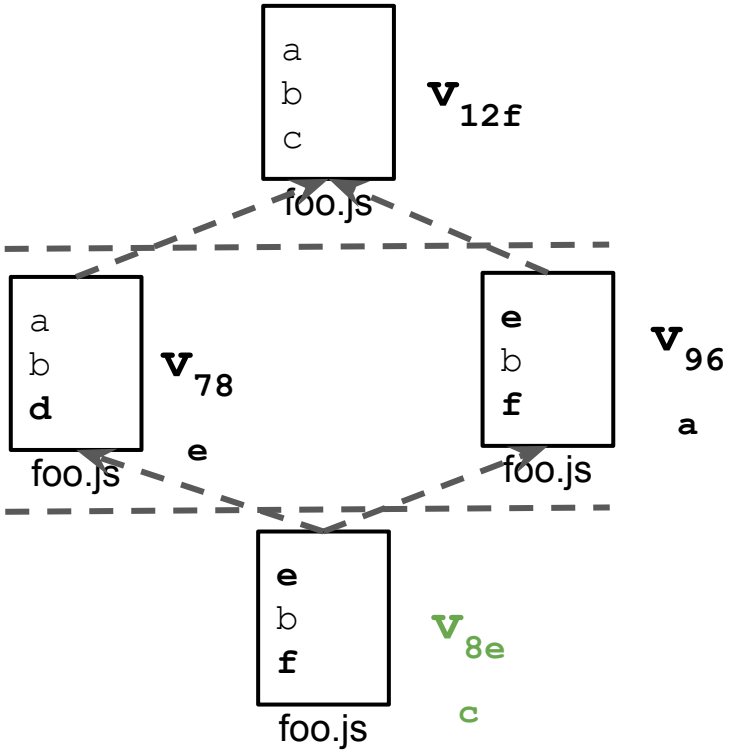
Decentralized version control



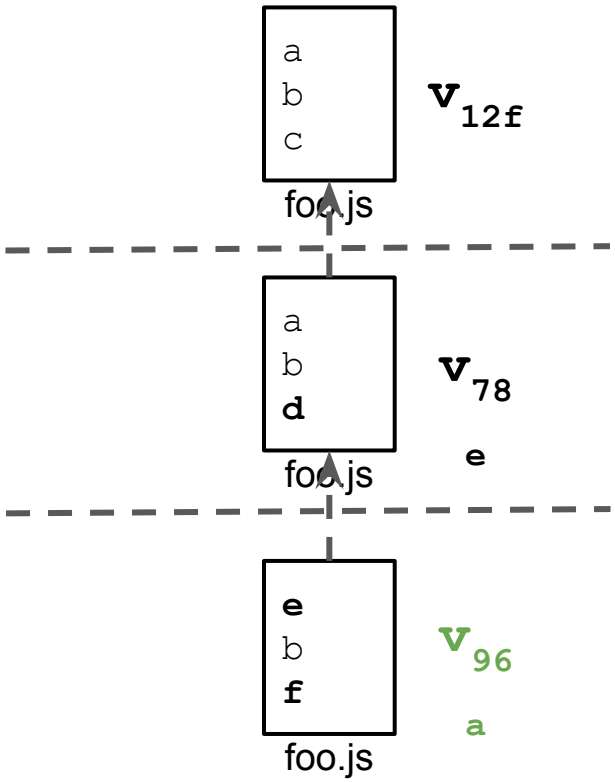
Decentralized version control



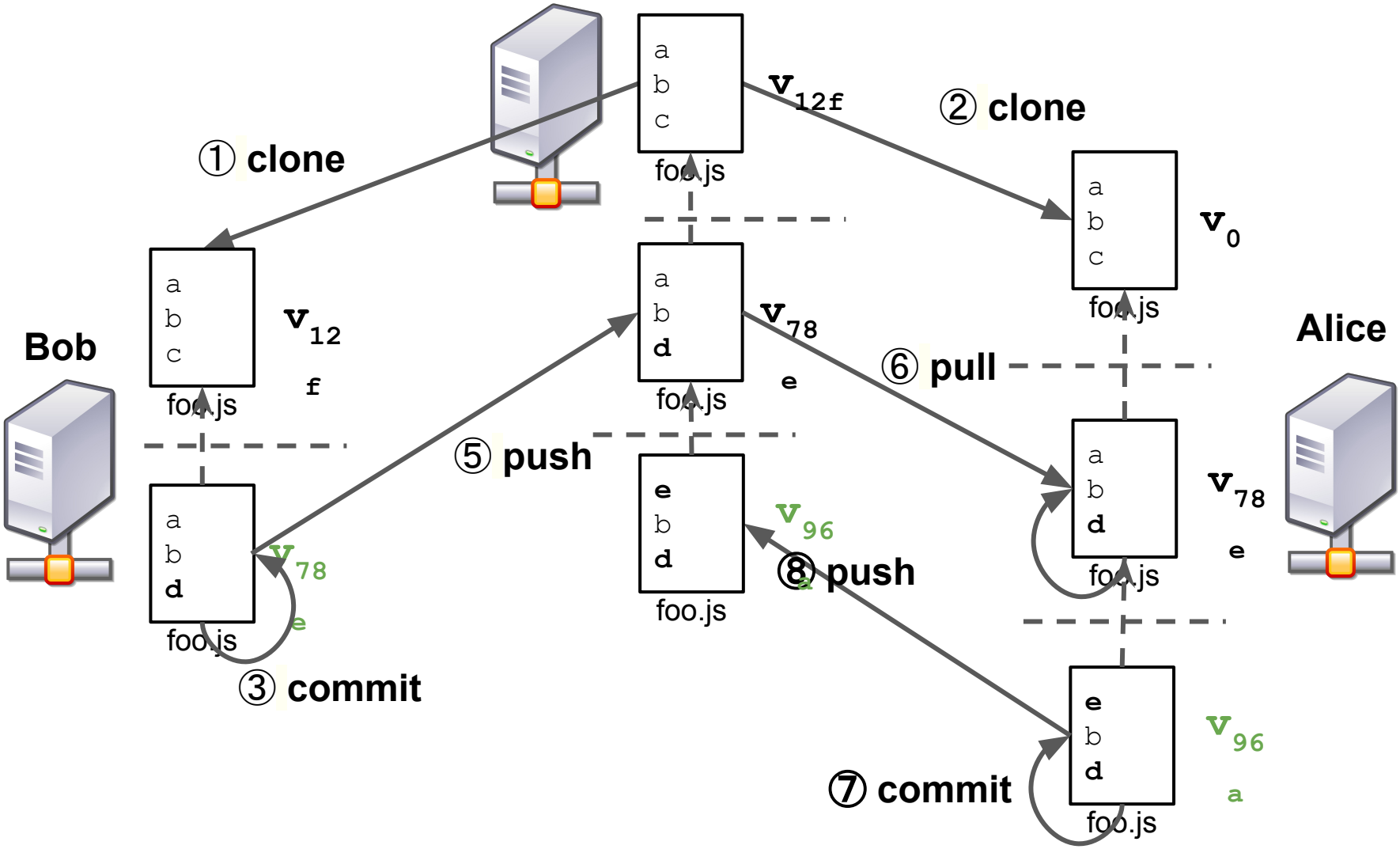
Diamond merge



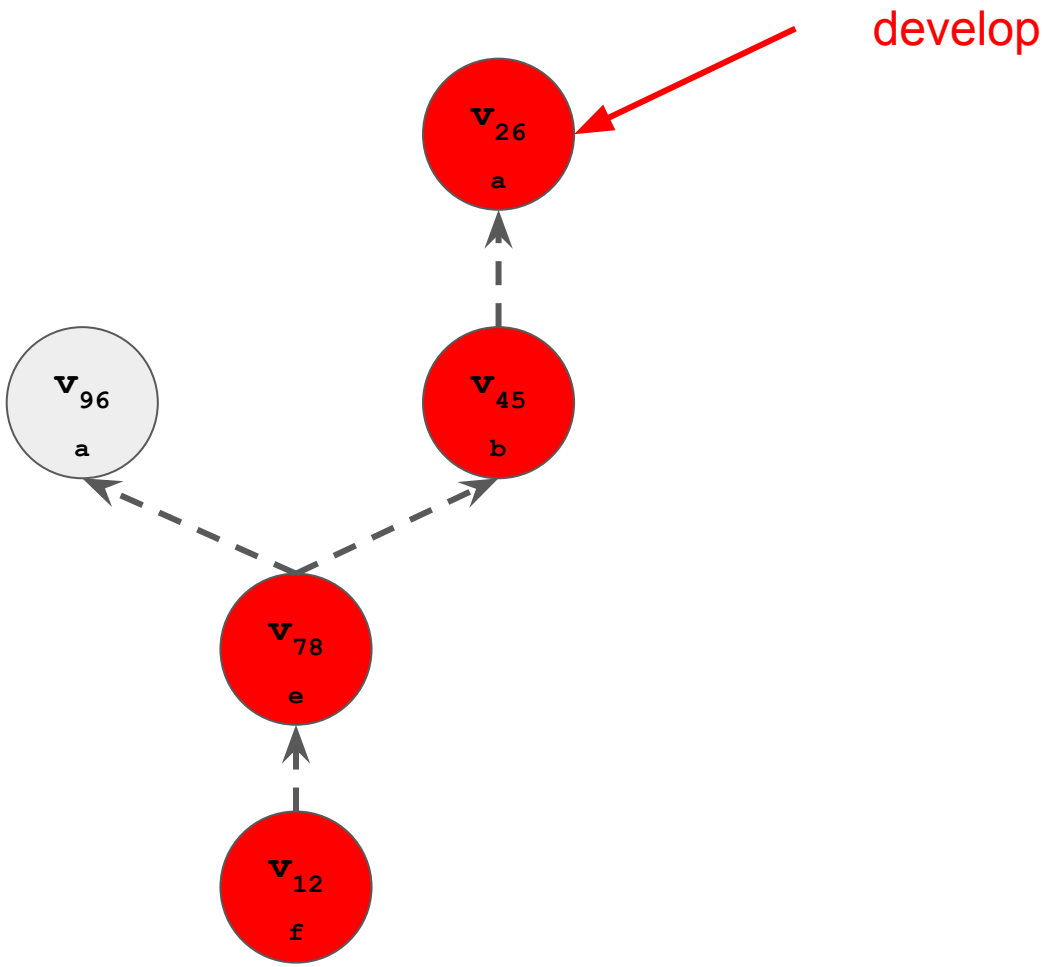
Rebase



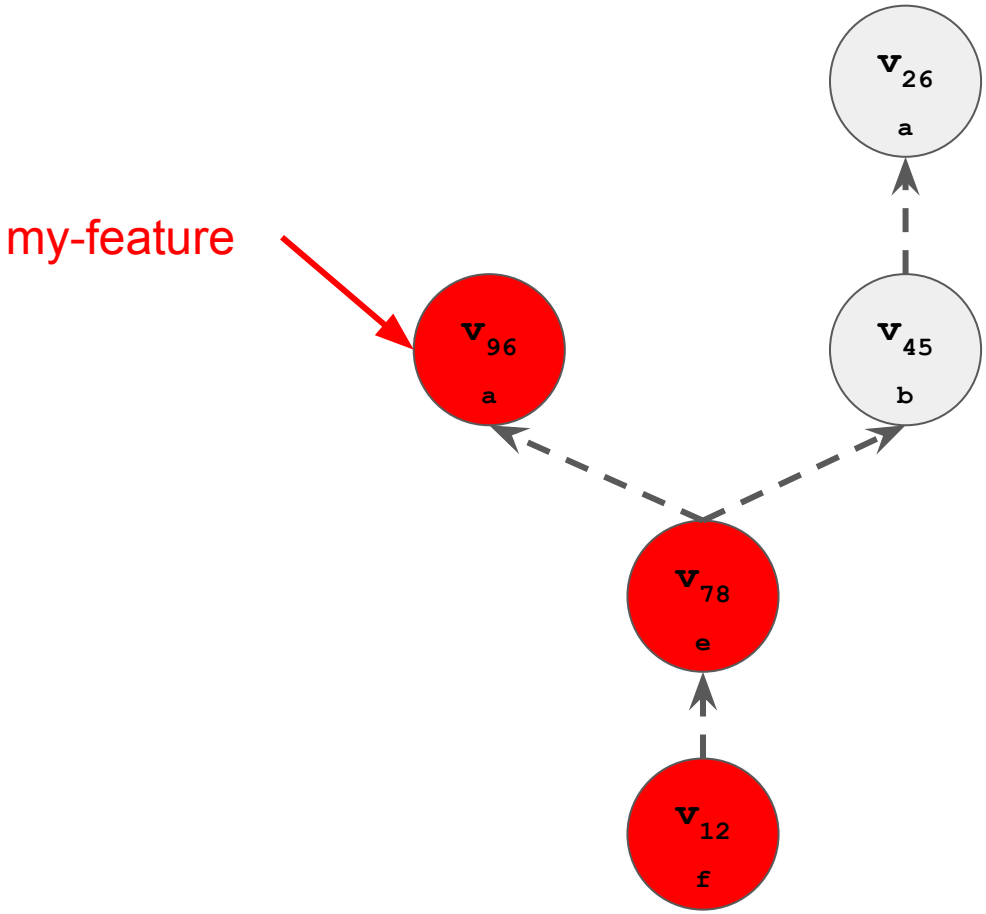
Decentralized version control



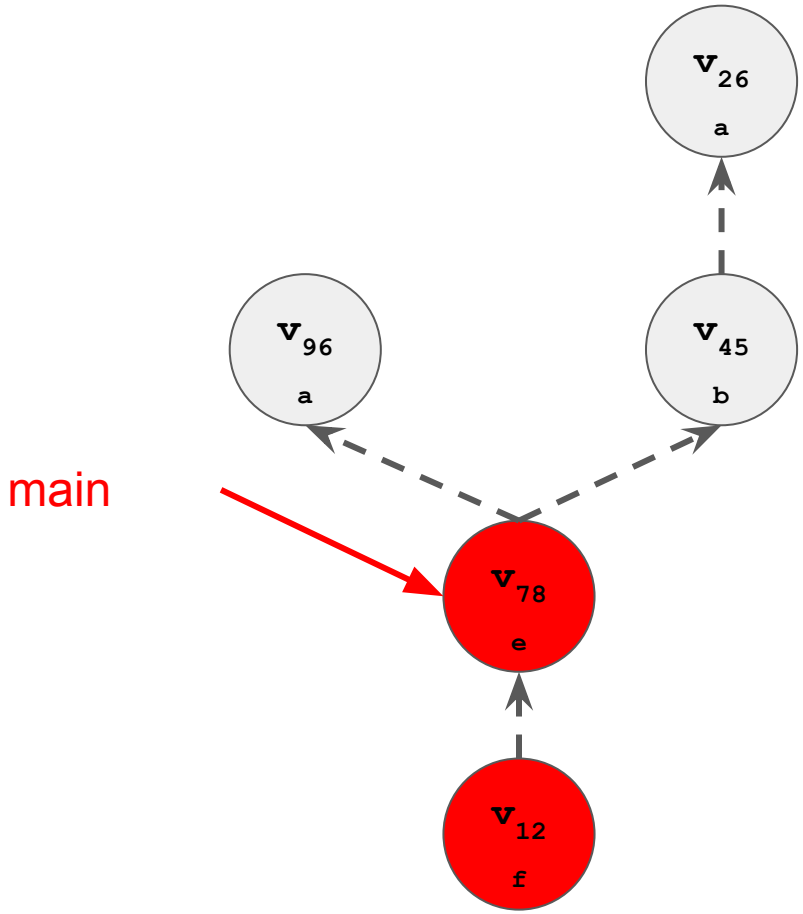
Named branches



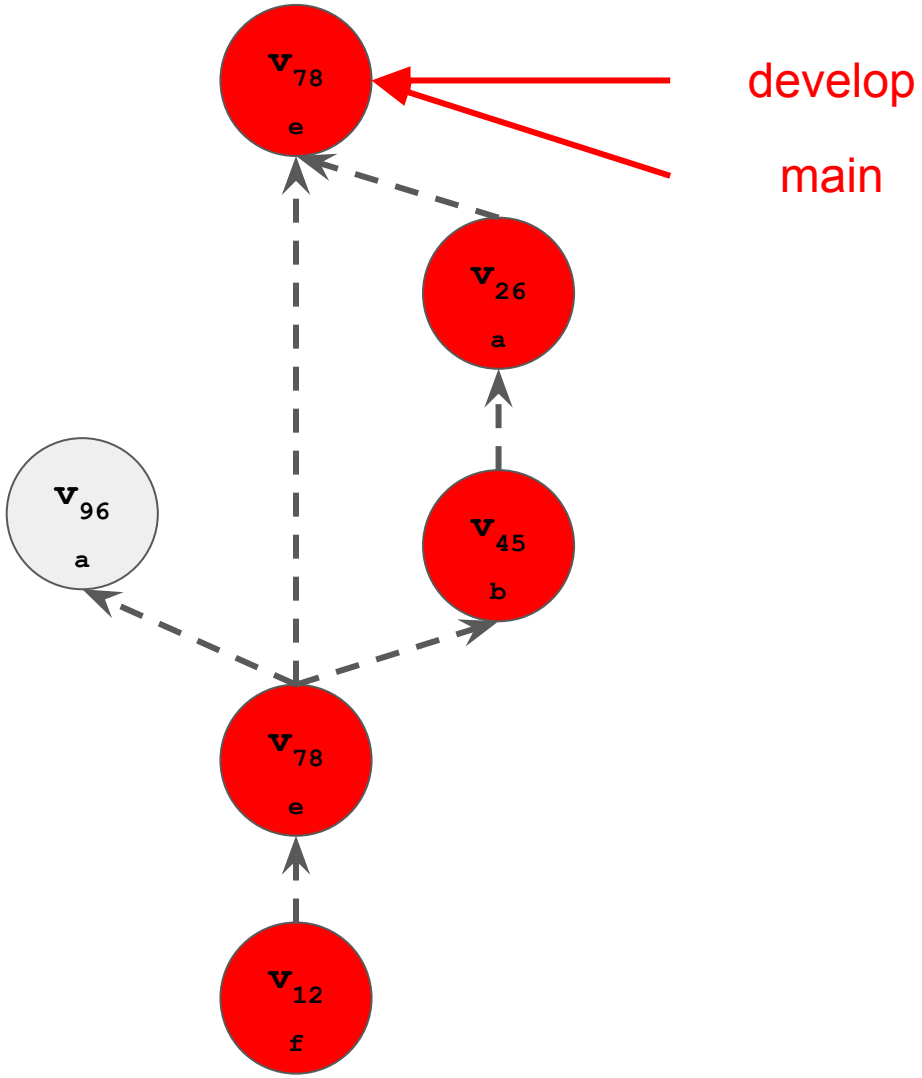
Named branches



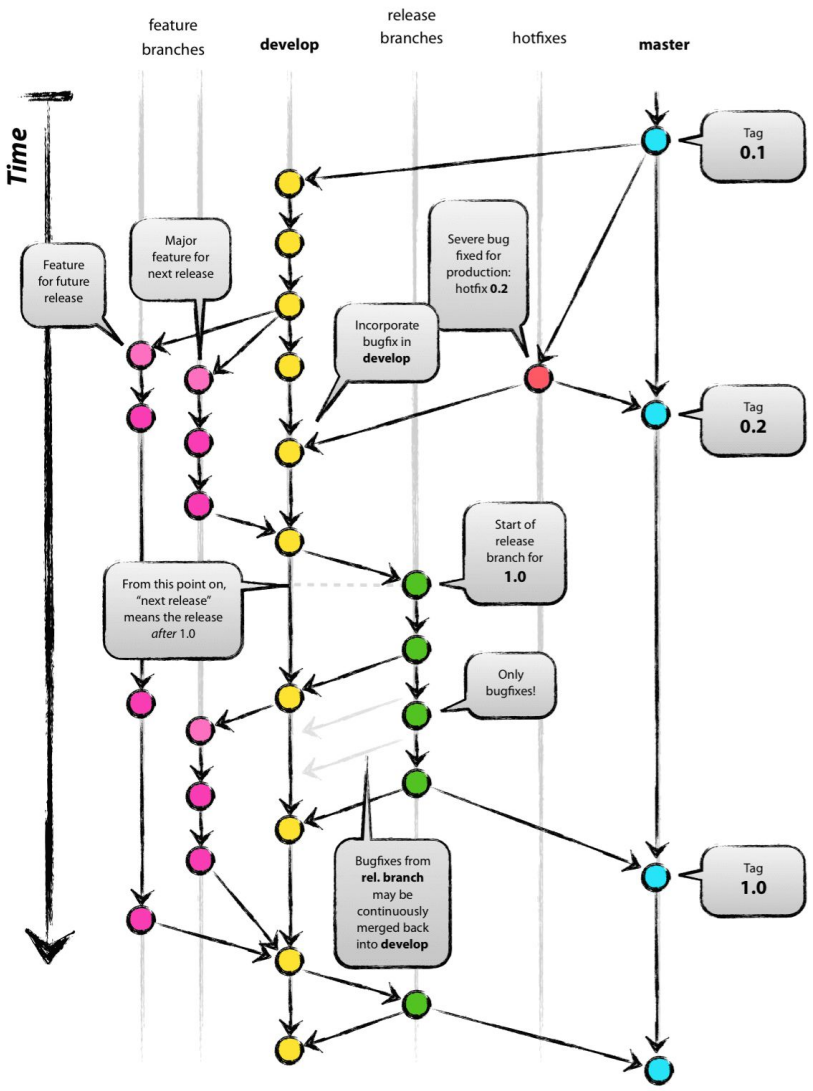
Named branches



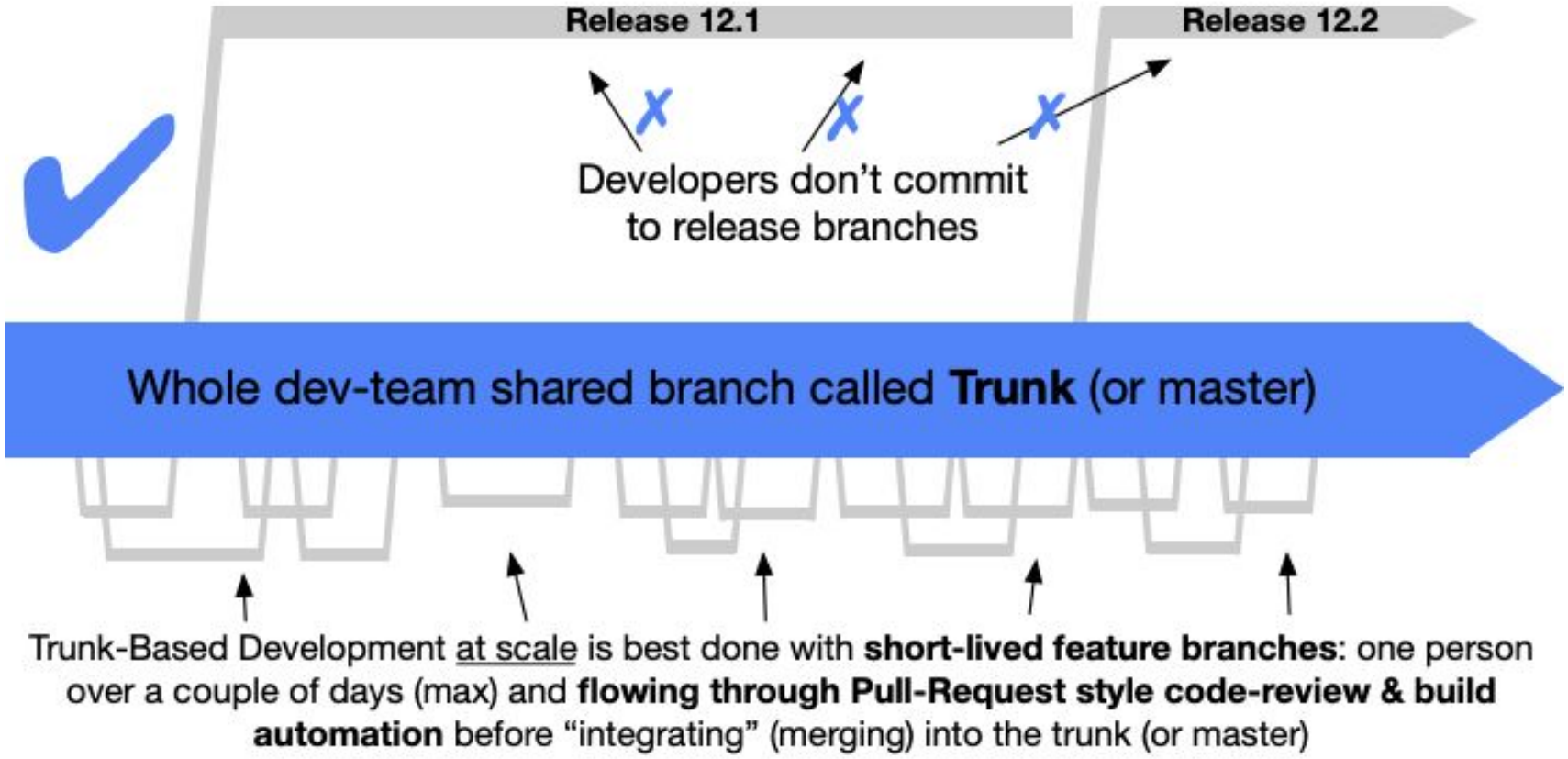
Branch merging



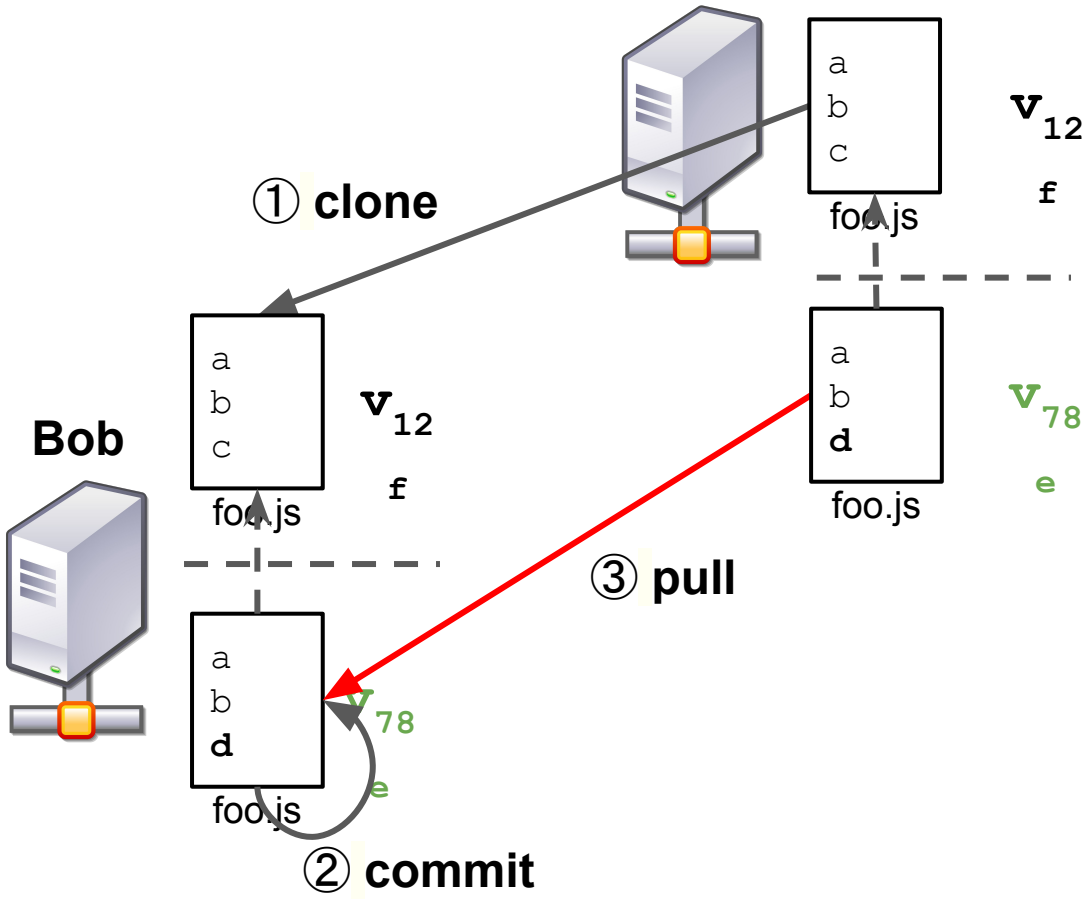
Git flow


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

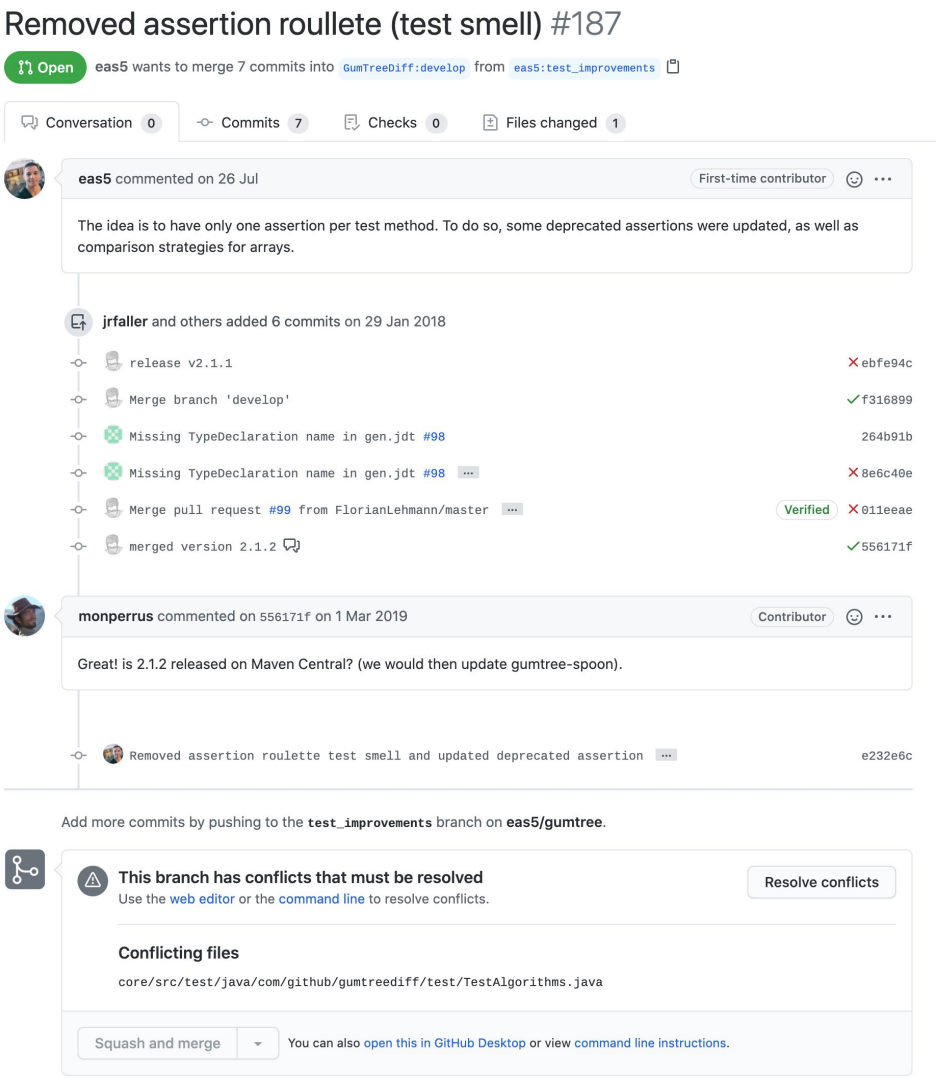
Trunk based development



Pull based development



Pull requests



Code review

chadhietala reviewed 13 days ago

View changes

packages/@ember/-internals/runtime/lib/system/array_proxy.js Outdated

```
14  15    import EmberObject from './object';
15  16    import { isArray, MutableArray } from '../mixins/array';
16  17    import { assert } from '@ember/debug';
17  18
        19  + const CONSUMED = symbol('ARRAY_PROXY_CONSUMED');
```

chadhietala 13 days ago • edited Member

Is there a reason to use this over a `WeakMap` ?

pzuraq 13 days ago Author Contributor

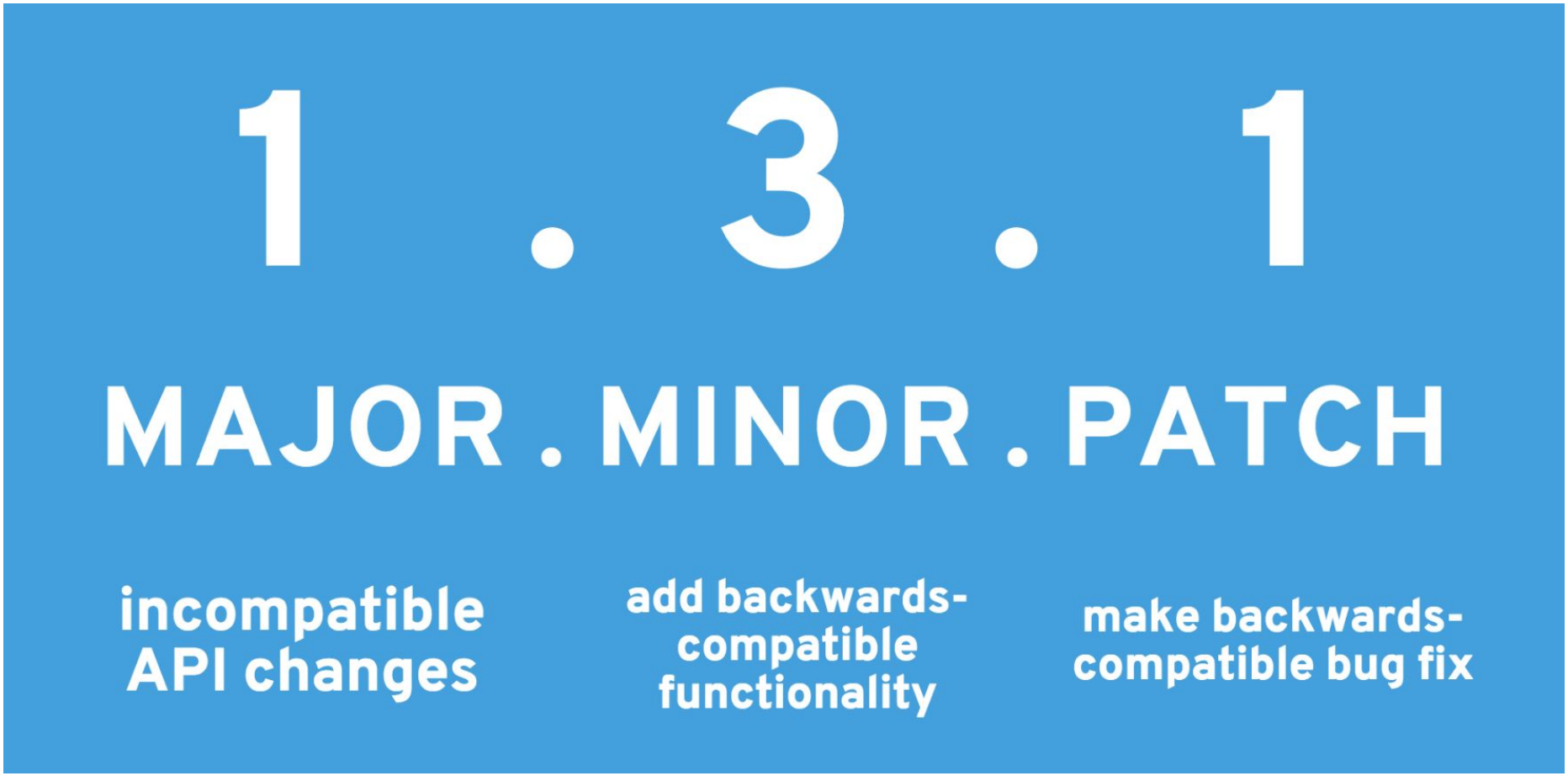
Not really, we could definitely update it to a `WeakMap` or `WeakSet` . I figured there *may* be perf implications, but don't have a strong opinion one way or the other

rwjblue 13 days ago Member

Ya, I generally agree with **@chadhietala** here. I'd prefer to avoid adding arbitrary fields to the underlying objects. I doubt there is much difference between this implementation and a WeakMap/WeakSet based one, but I'm happy to test if you'd like to assuage concerns...

Commit messages

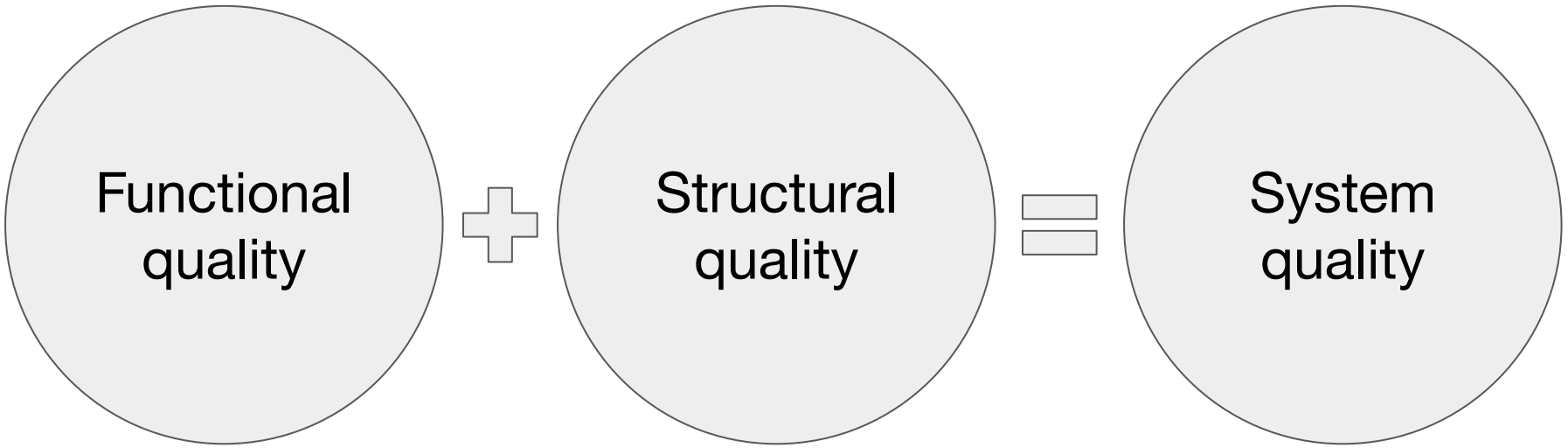
Semantic versioning



https://dev.to/jeiman/how-to-automate-app-releases-using-fastlane-and-semver-for-hybrid-applications-24m0

Code writing

What is software systems' quality?

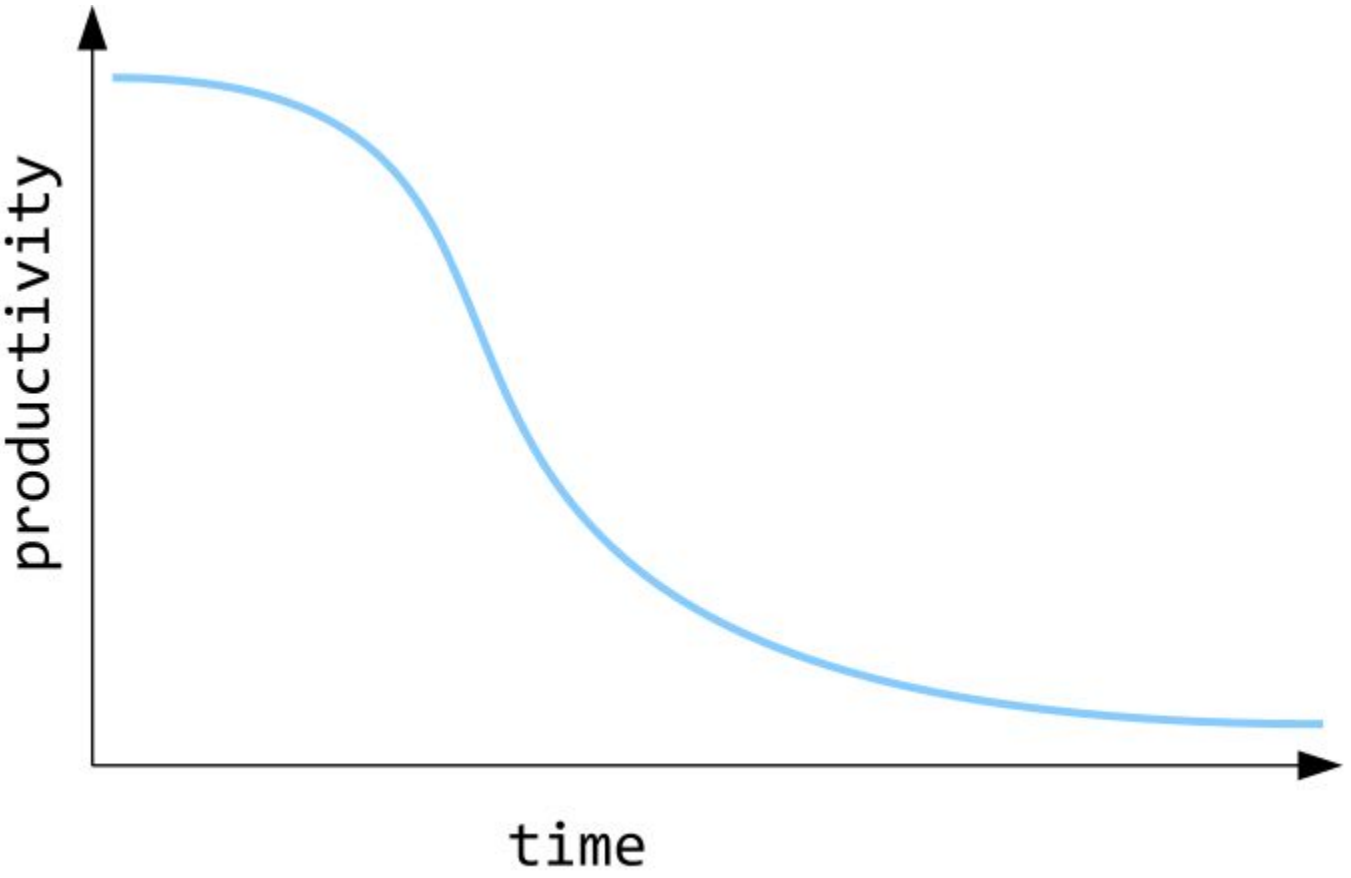


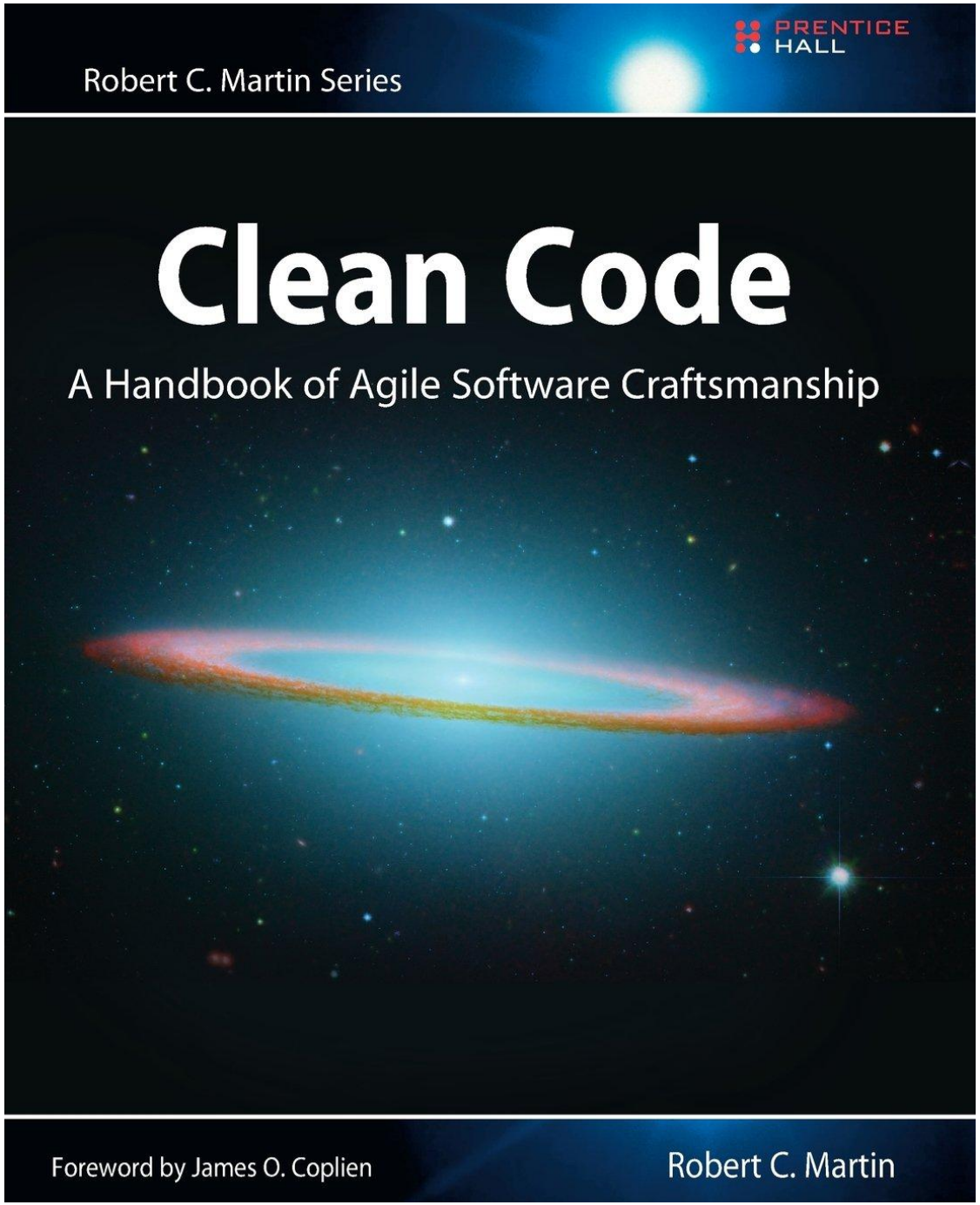
Functional and structural quality

- Qualité fonctionnelle : le logiciel est conforme à ses spécifications fonctionnelles (ce qu'il doit faire)
- Qualité structurelle
  - Le logiciel est conforme à ses spécifications non fonctionnelles (comment il doit faire : est il assez rapide? est il assez sécurisé ?)
  - La structure interne du logicielle est bonne (architecture, code source)

Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.

The mythical productivity curve




Robert C. Martin Series
PRENTICE HALL
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin

Bjarne Strousup (créateur de C++)

I like my code to be elegant and efficient. The logic should be straightforward to make it hard for bugs to hide, the dependencies minimal to ease maintenance, error handling complete according to an articulated strategy, and performance close to optimal so as not to tempt people to make the code messy with unprincipled optimizations. ***Clean code does one thing well****.*

Grady Booch (expert POO)

Clean code is simple and direct. *Clean code reads like well-written prose. Clean code never obscures the designer's intent but rather is full of crisp abstractions and straightforward lines of control*

Dave Thomas (*parrain* d'Eclipse)

Clean code can be read, and enhanced by a developer other than its original author. *It has unit and acceptance tests. It has meaningful names. It provides one way rather than many ways for doing one thing. It has minimal dependencies, which are explicitly defined, and provides a clear and minimal API. Code should be literate since depending on the language, not all necessary information can be expressed clearly in code alone.*

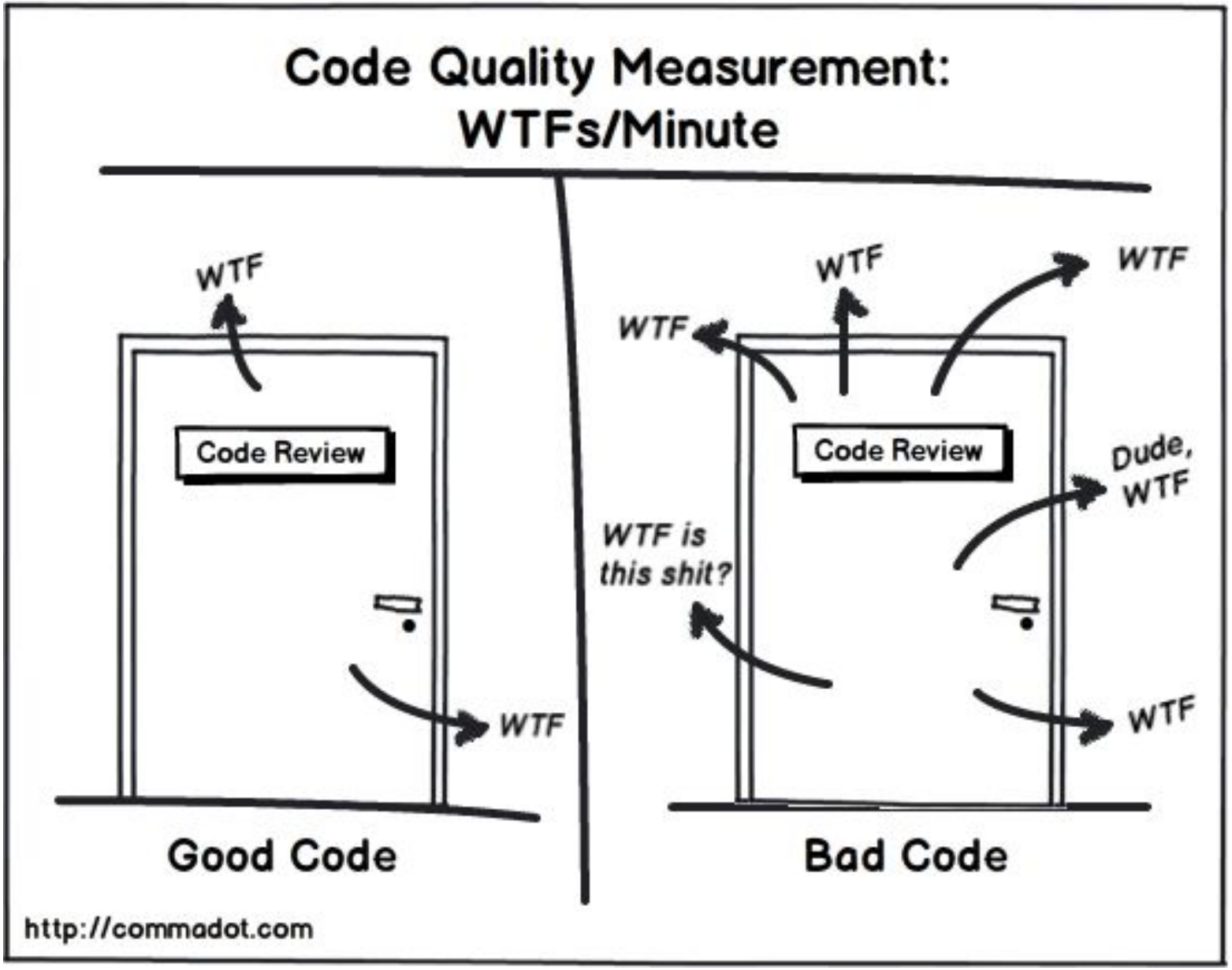
Michael Feathers (expert de code Legacy)

I could list all of the qualities that I notice in clean code, but there is one overarching quality that leads to all of them. ***Clean code always looks like it was written by someone who cares****. There is nothing obvious that you can do to make it better. All of those things were thought about by the code's author, and if you try to imagine improvements, you're led back to where you are, sitting in appreciation of the code someone left for you, code left by someone who cares deeply about the craft.*

What is clean code?

Clean code is

- Easy to understand
- Easy to use
- Easy to reuse
- Easy to evolve


Code Quality Measurement:
WTFs/Minute
WTF
Code Review
WTF
Good Code
WTF
WTF
WTF
WTF
Code Review
Dude,
WTF
WTF is
this shit?
WTF
Bad Code
http://commadot.com

Boy scout rule

Leave the campground cleaner than you found it.

Naming things

Information carrying names

```
public void s(int[] a, int b, int e) {
  if (b < e) {
    int i = p(a, b, e);
    s(a, b, i - 1);
    s(a, i + 1, e);
  }
}
```

```
private int p(int[] a, int b, int e) {
  int p = a[e];
  int i = (b - 1);
  for (int j = b; j < e; j++) {
    if (a[j] ⩽ p) {
      i++;

      int t = a[i];
      a[i]= a[j];
      a[j]= t;
    }
  }

  int t = a[i + 1];
  a[i + 1]= a[e];
  a[e]= t;

  return i+1;
}
```

Information carrying names

```
public void s(int[] values,
    int beginIndex, int endIndex) {
  if (beginIndex < endIndex) {
    int partitionIndex = p(values,
beginIndex, endIndex);
    s(values, beginIndex,
partitionIndex - 1);
    s(values, partitionIndex + 1,
endIndex);
  }
}
```

```
private int p(int[] values,
    int beginIndex, int endIndex) {
  int pivotValue = values[endIndex];
  int i = (beginIndex - 1);
  for (int j = beginIndex;
      j < endIndex; j++) {
    if (values[j] ⩽ pivotValue) {
      i++;

      int valueToSwap = values[i];
      values[i] = values[j];
      values[j] = valueToSwap;
    }
  }

  int valueToSwap = values[i + 1];
  values[i + 1] = values[endIndex];
  vaues[endIndex] = valueToSwap;

  return i + 1;
}
```

Information carrying names

```
public void quickSort(int[] values,
int beginIndex, int endIndex) {

  if (beginIndex < endIndex) {
    int partitionIndex =
partition(values, beginIndex,
endIndex);
    quickSort(values,
      beginIndex, partitionIndex - 1);
    quickSort(values,
      partitionIndex + 1, endIndex);
  }

}
```

```
private int partition(int[] values,
int beginIndex, int endIndex) {
  int pivotValue = values[endIndex];
  int i = (beginIndex - 1);
  for (int j = beginIndex;
      j < endIndex; j++) {
    if (values[j] ⩽ pivotValue) {
      i++;

      int valueToSwap = values[i];
      values[i] = values[j];
      values[j] = valueToSwap;
    }
  }

  int valueToSwap = values[i + 1];
  values[i + 1] = values[endIndex];
  vaues[endIndex] = valueToSwap;

  return i + 1;
}
```

Information carrying names

- The name of a class, method or variable **must** convey its goal
- If you feel like commenting a name, **choose a better name!**
- Don’t hesitate to **spend time** on choosing a good name
- Use business domain names (`Account`)
- Use general developer knowledge names (`LinkedList`)

Avoid disinformation

- Avoid **ambiguous** names
  - `tmp` temperature or temporary variable?
- Avoid **referring to the implementation**, it can change!
  - `accounts` instead of `accountList`
- Avoid **ambiguous** characters
  - `l I 1 O 0`

No encoding

- Modern types systems and IDE make encoding obsolete

😢

```
public class Part {
  private String m_dsc; // The textual description
    void setName(String name) {
      m_dsc = name;
    }
  }
}
```

😀

```
public class Part {
  private String description
    void setDescription(String description) {
      this.description = description;
    }
  }
}
```

Searchable names

- Developer search for identifiers almost every day! Avoid magic number or one letter identifiers

😢

```
for (int j = 0; j < 34; j++) {
  s += (t[j] * 4) / 5;
}
```

😀

```
int realDaysPerIdealDay = 4;
int WORK_DAYS_PER_WEEK = 5;
int sum = 0;
for (int j = 0; j < NUMBER_OF_TASKS; j++) {
  int realTaskDays = taskEstimate[j] * realDaysPerIdealDay;
  int realTaskWeeks = (realdays / WORK_DAYS_PER_WEEK);
  sum += realTaskWeeks;
}
```

One concept, one word

- Avoid using different names for the same concept or developers will be confused

😢

```
void getTimer();
void retrieveTime();
void fetchDate();
```

😀

```
void getTimer();
void getTime();
void getDate();
```

No jokes

- Use only straightforward identifiers

😢

```
void goodbye();
```

😀

```
void exit();
```

Functions

Good practices

- Rule 1 : **write short functions**
  - A few lines only : 5, 10
- Rule 2 : **a function does one thing**
- Rule 3 : **few paramètres**
  - Zero, one or two

Stepdown rule

- One abstraction level per function
  - High-level functions call low level functions that makes the essence of the computation to be done
- The *stepdown rule*
  - Code is read like a story from top to bottom
  - Top-level functions outline the computation, low-level one implement the details

😢

```
private void createGui() {
  add(createDataEntryPanel(),
BorderLayout.NORTH);
  add(createButtonPanel(),
BorderLayout.SOUTH);
  setJMenuBar(createMenuBar());
}
```

😀

```
private void createGui() {
  createDataEntryPanel();
  createButtonPanel();
  createMenuBar();
}

private JPanel createDataEntryPanel() { ... }
private JPanel createButtonPanel() { ... }
private JMenuBar createMenuBar() { ... }
```

Function parameters

- Try to **reduce the number of parameters**
- When more than two parameters
  - Introduce new functions
  - Use the receiver
  - Introduce custom data types

😢

```
void render(boolean isSuite);
render(true); // when called
```

😀

```
void renderForSuite();
void renderForSingleTest();
```

Function parameters

😢

```
public void doSomethingWithEmployee(String name, double pay, Date  hireDate);
```

😀

```
public void doSomethingWithEmployee(Employee employee);
```

😢

```
void appendFooter(Report report); // side-effect on report?
```

😀

```
Report r;
r.appendFooter();
```

Side effects

- Avoid function that **makes both side effects and return values**
- If a function make side-effects it **must be crystal clear in its name**

😢

```
public class UserValidator {
  private Cryptographer cryptographer;

  public boolean checkPassword(String userName, String password) {
    User user =  UserGateway.findByName(userName);
    if (user != User.NULL) {
      String codedPhrase = user.getPhraseEncodedByPassword();
      String phrase = cryptographer.decrypt(codedPhrase, password);
      if ("Valid Password".equals(phrase)) {
        Session.initialize();
        return true;
      }
    }
    return false;
  }
}
```

Handle error with exceptions

😢

```
if (deletePage(page) == E_OK) {
  if (registry.deleteReference(page.name) == E_OK) {
    if (configKeys.deleteKey(page.name.makeKey()) == E_OK){
      logger.log("page deleted");
    } else {
      logger.log("configKey not deleted");
}
} else {
    logger.log("deleteReference from registry failed"); }
} else {
  logger.log("delete failed"); return E_ERROR;
}
```

😀

```
try {
  deletePage(page);
  registry.deleteReference(page.name);
  configKeys.deleteKey(page.name.makeKey());
} catch (Exception e) {
  logger.log(e.getMessage());
}
```

Error handling is one thing

- Avoid mixing classical computation and error handling

😀

```
public void delete(Page page) {
  try {
    deletePageAndAllReferences(page); }
  catch (Exception e) {
    logError(e);
  }
}

private void deletePageAndAllReferences(Page page) throws Exception {
  deletePage(page);
  registry.deleteReference(page.name);
  configKeys.deleteKey(page.name.makeKey());
}

private void logError(Exception e) {
  logger.log(e.getMessage());
}
```

D’ont Repeat Yourself

- Avoid writing twice the same code
  - You’ll have to fix the bugs everywhere!
  - Maybe a missing abstraction?

😢

```
void setR(int r) {
  if (r < 0) this.r = 0;
  else if (r > 255) this.r = 255;
  else this.r = r;
}

void setG(int g) {
  if (g < 0) this.g = 0;
  else if (g > 255) this.g = 255;
  else this.g = g;
}
```

😀

```
void setR(int r) {
  this.r = putIn1To10Range(r);
}

void setG(int g) {
  this.g = putIn1To10Range(g);
}

int putIn0To10Range(int number) {
  if (number < 0) return 0;
  else if (number > 255) return 255;
  else return number;
}
```

Comments

Good practices

- Comments can quickly become a **bad practice**
- They have to evolve along with the code
- Therefore, they can become **outright lies**

Comments don’t make up for bad code. If you feel the need for a comment to explain some code, put effort into improving the code, not authoring comments for it.

Naming and comments

- A well chosen name often avoid the necessity of comments

😢

```
// Check to see if the employee is eligible for full benefits
if ((employee.flags & HOURLY_FLAG) && (employee.age > 65))
```

😀

```
if (employee.isEligibleForFullBenefits())
```

Good comments

- Clarify the intent of the code

😀

```
public int compareTo(Object o) {
  if(o instanceof WikiPagePath) {
    WikiPagePath p = (WikiPagePath) o;
    String compressedName = StringUtil.join(names, "");
    String compressedArgumentName = StringUtil.join(p.names, "");
    return compressedName.compareTo(compressedArgumentName);
  }
  return 1;
  // we are greater because we are the right type.
}
```

Good comments

- Licenses
- TODO

😀

```
// Copyright (C) 2003,2004,2005 by Object Mentor, Inc. All rights reserved.
// Released under the terms of the GNU General Public License version 2 or later.
```

😀

```
// TODO-MdM these are not needed
// We expect this to go away when we do the checkout model
protected VersionInfo makeVersion() throws Exception {
  return null;
}
```

Bad comments

- Paraphrasing the code

😢

```
/**
 * Default constructor.
 */
protected Date() {}

/**
 * Returns the day of the month.
 *
 * @return the day of the month.
 */
public int getDayOfMonth() {
  return dayOfMonth;
}
```

Good comments

- Explain the code further
- Generate a high-quality documentation

😢

```
/**
 * Create a date with the current system's timestamp.
 */
protected Date() {}

/**
 * @return the day of the month of the corresponding date, between 1 and 31.
 */
public int getDayOfMonth() {
  return dayOfMonth;
}
```

Formatting

Code should be written for human beings to understand, and only incidentally for machines to execute.

Vertical spacing

- Vertical spacing between concepts

😢

```
package fitnesse.wikitext.widgets;
import java.util.regex.*;
public class BoldWidget extends ParentWidget {
  public static final String REGEXP = "'''.+?'''";
  private static final Pattern pattern = Pattern.compile("'''(.+?)'''",
    Pattern.MULTILINE + Pattern.DOTALL
  );
  public BoldWidget(ParentWidget parent, String text) throws Exception
    {
    super(parent);
    Matcher match = pattern.matcher(text);
    match.find();
    addChildWidgets(match.group(1));
} }
```

Vertical spacing

😀

```
package fitnesse.wikitext.widgets;

import java.util.regex.*;

public class BoldWidget extends ParentWidget {
  public static final String REGEXP = "'''.+?'''";
  private static final Pattern pattern = Pattern.compile("'''(.+?)'''",
    Pattern.MULTILINE + Pattern.DOTALL
  );

  public BoldWidget(ParentWidget parent, String text) throws Exception {
    super(parent);
    Matcher match = pattern.matcher(text);
    match.find();
    addChildWidgets(match.group(1));
  }

}
```

Density and vertical spacing

- Density and order express a link between elements
- Attributes are declared on top of the class
- Related attributes must be close
- Callee methods must be close to caller methods
- Variable are declared closed to their usage location

Horizontal spacing

- Horizontal spacing makes it easier to read the code
  - Space around operators
  - Space around method parameters
  - Always indent scopes

😀

```
public void add(int x,int y) {
int z=x+y;
return z;
}
```

😀

```
public void add(int x, int y) {
  int z = x + y;
  return z;
}
```

Formatting as a team rule

- Each team must think about and enforce formatting rules to produce consistent code
- Language traditions, if any, supersede team rules (think about newcomers)
- Your code must look like anybody’s code

Services and data

Services objects

- Hide implementation details and provide services

😀

```
public interface Point {
  double getX();
  double getY();
  void setCartesian(double x, double y);
  double getR();
  double getTheta();
  void setPolar(double r, double theta);
}
```

Data objects

- Structure the data and do not provide any service

😀

```
public class Point {
  public double x;
  public double y;
}
```

Complementarity data and services

- Easy to add new service objects by extending the base class
- Hard to add new services because all current subclasses must be modified

- Data objects makes it easy to add new services because you can always add a function
- Hard to add new attributes into data objects since you have to modify all existing functions

Demeter's law

- Only talk to your friends
- A method `f` of class `C` only talks to
  - `C`
  - Objects created by `f`
  - Objects passed to `f`
  - Attributes of `C`

😢

```
final String outputDir = ctxt.getOptions().getScratchDir().getAbsolutePath();
```

Classes

Classes should be small

😢

```
public class SuperDashboard extends JFrame implements MetaDataUser
   public String getCustomizerLanguagePath()
   public void setSystemConfigPath(String systemConfigPath)
   public String getSystemConfigDocument()
   public void setSystemConfigDocument(String systemConfigDocument)
   public boolean getGuruState()
   public boolean getNoviceState()
   public boolean getOpenSourceState()
   public void showObject(MetaObject object)
   public void showProgress(String s)
   public boolean isMetadataDirty()
   public void setIsMetadataDirty(boolean isMetadataDirty)
   public Component getLastFocusedComponent()
   public void setLastFocused(Component lastFocused)
   public void setMouseSelectState(boolean isMouseSelected)
   public boolean isMouseSelected()
   public LanguageManager getLanguageManager()
   public Project getProject()
   public Project getFirstProject()
   public Project getLastProject()
   public String getNewProjectName()
   public void setComponentSizes(Dimension dim)
   public String getCurrentDir()
   public void setCurrentDir(String newDir)
}
```

Small classes does not always make sense

😢

```
public class SuperDashboard extends JFrame implements MetaDataUser
    public Component getLastFocusedComponent()
    public void setLastFocused(Component lastFocused)
    public int getMajorVersionNumber()
    public int getMinorVersionNumber()
    public int getBuildNumber()
}
```

Single **R**esponsability **P**rinciple

- Classes should have only one reason to change

😢

```
public class SuperDashboard extends JFrame implements MetaDataUser
    public Component getLastFocusedComponent()
    public void setLastFocused(Component lastFocused)
    public int getMajorVersionNumber()
    public int getMinorVersionNumber()
    public int getBuildNumber()
}
```

😀

```
public class Version {
    public int getMajorVersionNumber()
    public int getMinorVersionNumber()
    public int getBuildNumber()
}
```

Cohesion

- Methods of a class should use its attributes

😀

```
public class Stack {
  private int topOfStack = 0;
  List<Integer> elements = new LinkedList<Integer>();

  public int size() {
    return topOfStack;
  }
  public void push(int element) {
    topOfStack++;
    elements.add(element);
  }
  public int pop() throws PoppedWhenEmpty {
    if (topOfStack == 0)
      throw new PoppedWhenEmpty();
    int element = elements.get(--topOfStack);
    elements.remove(topOfStack);
    return element;
  }
}
```

Cohesion and SRP

- Using both == a lot of tiny classes
- A complex system **IS** complex
- Do you three very large drawers
- Or a bunch of tiny drawers?

Tests

Why do we test software

- Testing make you way more confident about the fact that your system behaves in the right manner
  - It implements the requirements
  - It has the required performance
  - ...
- Tests aims at detecting and correcting errors

Test principle

- We perform a sequence of actions on the system
- We compare the **observed state** of the system to the **expected state**
- If both states are equivalent the test is passing
- If not the test is failing

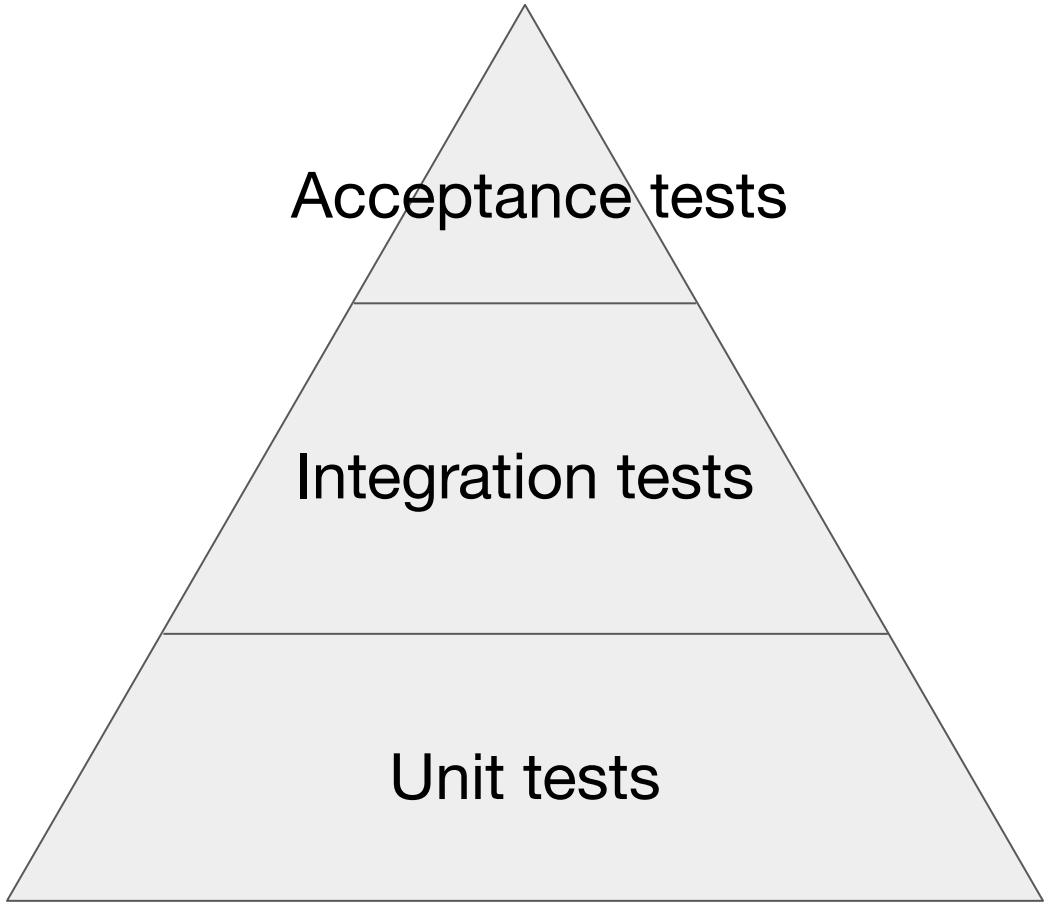
Black box and white box testing

- White box testing assumes that your knows the internal structure of the code to test
  - Example `int z = y / x;`
  - We should test what happens for `x = 0`
  - If you test code you wrote, you are in a white box situation
- Black box testing no nothing about the internal structure, it relies only on the requirements
  - Example : *the system must returns the value of y divided by x*
  - Requires an independent testing team

Tests classification

- **Unit tests** : ensure a software component works well when isolated from the other components
- **Integration tests**: ensures that several dependent components works well together
- **Acceptance test** : ensures that the system works as intended
- **System test** : ensures properties about the behavior of the system (performance, …)

La pyramide des tests



Unit tests

Unit tests

FIRST properties of unit tests

FAST

Many hundreds or thousands per second

Isolates

Failure reasons become obvious

Repeatable

Run repeatedly in any order, any time

Self-validating

No manual evaluation required

Timely

Written before the code

Unit tests principles

- **A**rrange
- **A**ct
- **A**ssert

😀

```java
import org.junit.Test;
import static org.junit.Assert.*;

public class MyTest {

    @Test
    public void testConcatenate() {
        Concatenator c = new Concatenator(); // arrange
        String result = c.concatenate("one", "two"); // act
        assertEquals("onetwo", result); // assert
    }
}
```

Arrange

- Unit tests has to prepare its required data
- Test data has to be independent from the other tests
- Test data has to be independent from the system

Assert

- Check the **observed** (actual) vs **expected** state
- One logical assert per test

😀

```
assertEquals(Object expected, Object actual)
assertSame(Object expected, Object actual)
assertNotSame(Object expected, Object actual)
assertNull(Object object)
assertNotNull(Object object)
assertTrue(boolean condition)
assertFalse(boolean condition)
fail(String)
```

Unit tests isolation

- Fréquemment un module nécessite un autre module pour fonctionner
  - Module `DisplayTemperature` nécessite le module `RetrieveTemperature`
- Ceci dit un test non passant du module `DisplayTemperature` doit **uniquement provenir** d'une erreur interne à ce module
- Que faire en cas d'erreur dans le module `RetrieveTemperature`?

Doublures de test

Qu’est-ce qu’une doublure de test?

- Le principe : **utiliser un module spécialement créé pour le test**
- Les doublures de test servent principalement à assurer l’isolation des tests
  - Au lieu d’utiliser le vrai module `RetrieveTemperature` on utilise un module `RetrieveTemperature` créé spécialement pour le test
- Les doublures de test on beaucoup d’autres bénéfices, elles permettent de tester des modules en présence dépendances
  - non disponible (la classe du développeur lent)
  - non-déterministe (du code qui appelle `Math.random()`)
  - lentes (un algorithme très complexe)
  - difficile à mettre en place (BDD)
  - comportant des situations exceptionnelles (déconnexion réseau)

Un exemple de doublure

😀

```
@Test
public void testDisplayTemperature() {
  DisplayTemperature d = new DisplayTemperature(new ZeroDegreeRetrieveTemperature());
  assertEquals("0 degrees", d.getTemperatureMessage());
}

public class ZeroDegreeRetrieveTemperature extends RetrieveTemperature {
  public double getTemperature() {
    return 0D;
  }
}
```

Les différents types de doublures

- Les fantômes (**dummy**) : valeurs qui se substituent à un objet qui ne sert à rien (exemple: `null` ou `new Object()`)
- Les bouchons (**stubs**) : écrits à la main par le développeur pour un test donné
- Les substituts (**fake**) : écrits à la main par le développeur pour plusieurs tests
- Les simulacres (**mock**) : générés par un framework
- Les espions (**spy**) : simulacres qui retiennent les actions que l'on fait sur eux

Les simulacres

- Sont moins coûteux à mettre en oeuvre que les bouchons
- Les utiliser en priorité

😀

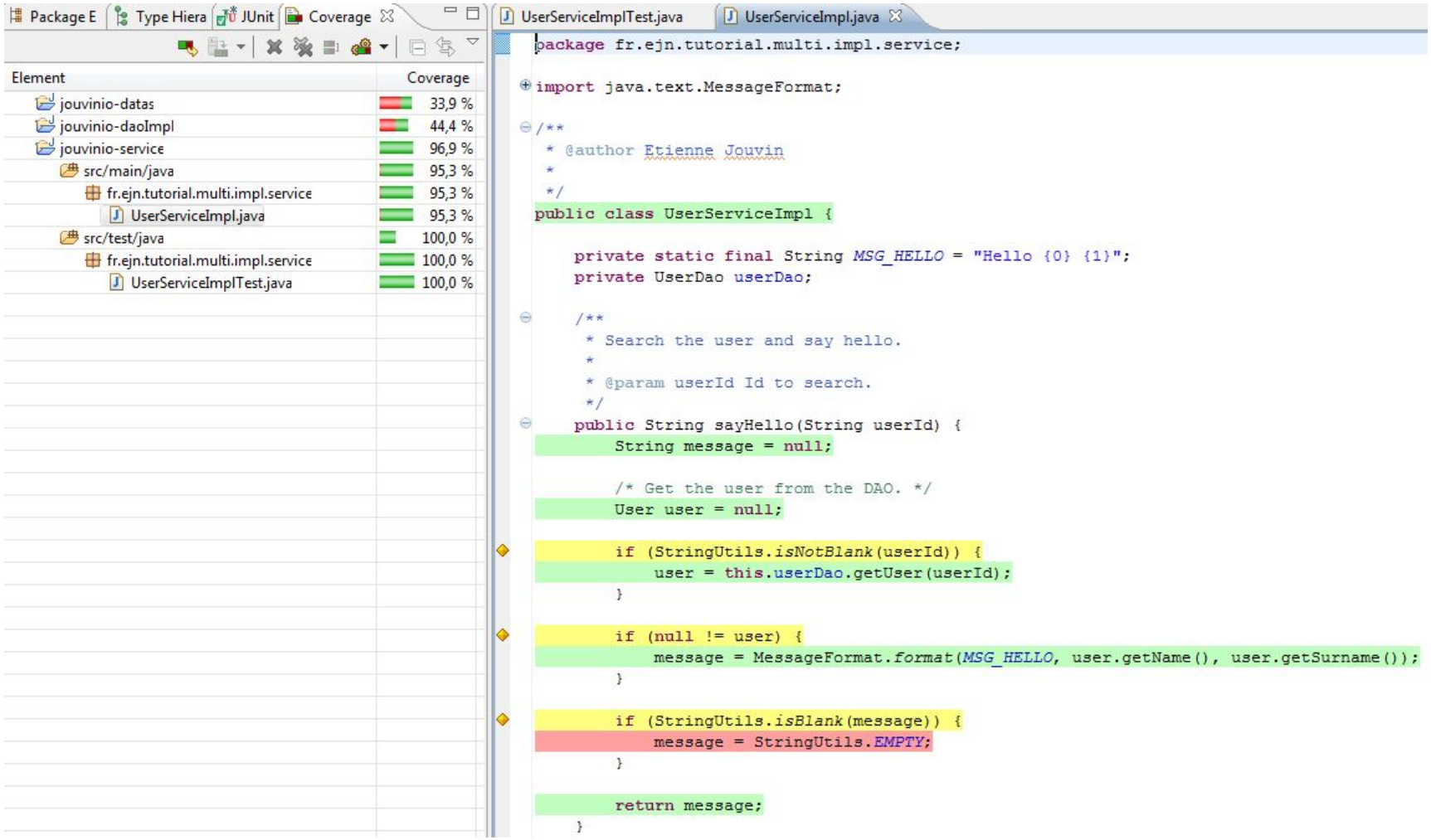
```
@Test
public void testDisplayTemperature() {
  // mocking
  RetrieveTemperature zeroDegree = mock(RetrieveTemperature.class);
  when(zeroDegree.getTemperature()).thenReturn(0);
  // test
  DisplayTemperature d = new DisplayTemperature(zeroDegree);
  assertEquals(d.getTemperatureMessage(), "0 degrees");
}
```

Tests’ quality

Tests’ quality

- Comment peut on s’assurer que les tests sont suffisants?

La couverture de test



La couverture de test

- 😃 Permet de savoir quelles instructions ont été réellement exécutées
- 😃 Permet de donner une mesure quantitative rapide sur l'effort de test
- 😢 100% de couverture ne veux pas dire tout est testé
- 😢 Ne permet pas de savoir si les tests pourront détecter des erreurs si il y en a

L’analyse des tests par mutation

Calculator.java

```
1   package de.triology.blog.pitest;
2
3   class Calculator {
4       static int add(int a, int b) {
5   2       return a + b;
6       }
7
8       static int subtract(int a, int b) {
9   2           return a - b;
10      }
11
12      static int multiply(int a, int b) {
13  2       return a * b;
14      }
15  }
```

Mutations

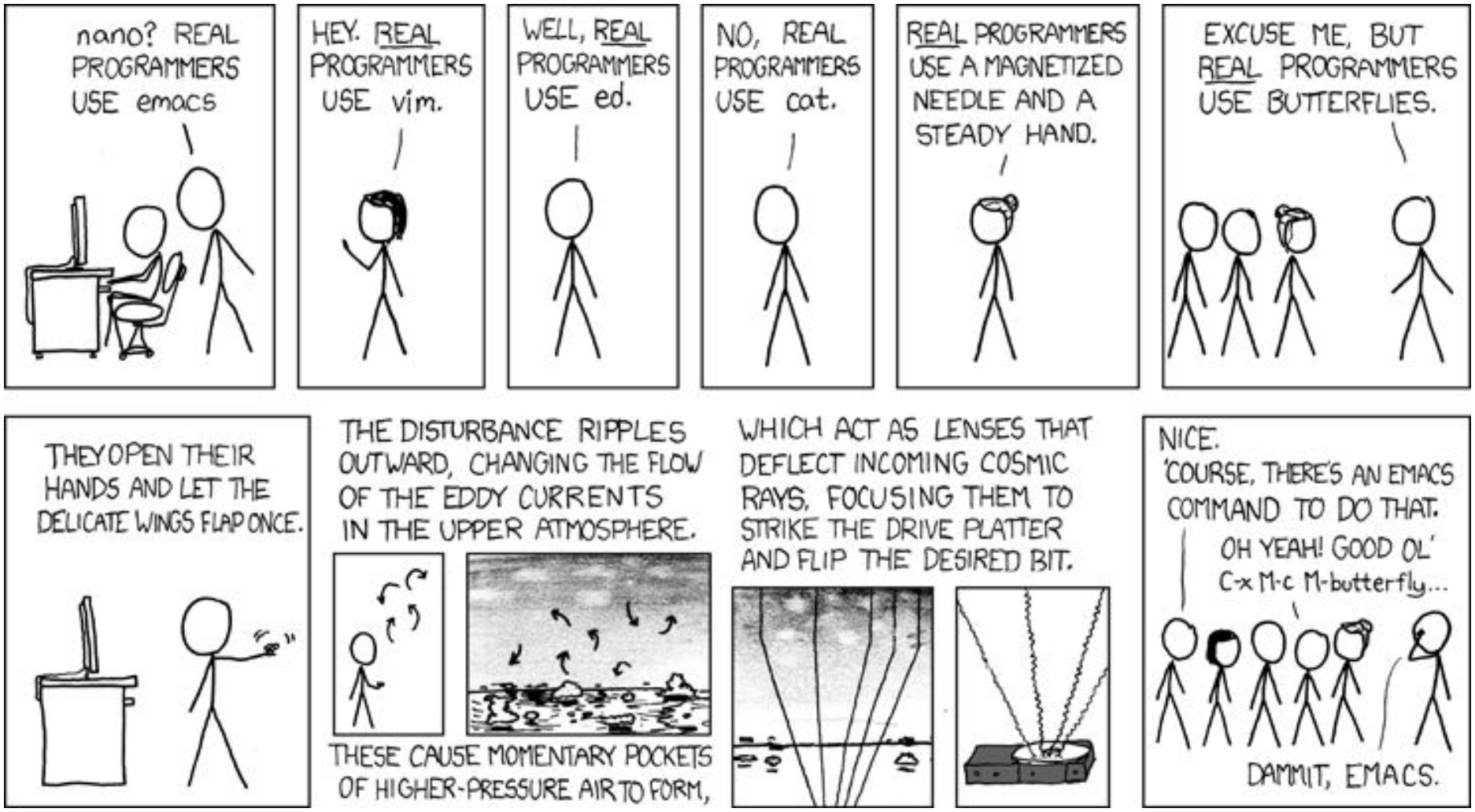
```
5   1. Replaced integer addition with subtraction → KILLED
    2. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED
9   1. Replaced integer subtraction with addition → SURVIVED
    2. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED
13  1. Replaced integer multiplication with division → SURVIVED
```

TRIOLOGY

L’analyse des tests par mutation

- 😀 Permet de vérifier que vos tests permettent **vraiment** de détecter des programmes défaillants
- 😀 Permet de pointer des cas précis ou les tests ne capturent pas une erreur
- 😀 Permet de dériver une métrique quantitative sur la qualité des tests (ratio de mutant tués)
  - mais imprécise car un mutant peut-être équivalent au programme original
- 😢 Ne permet pas de savoir quelles sont les parties testées ou non testée d’un logiciel

Outillage

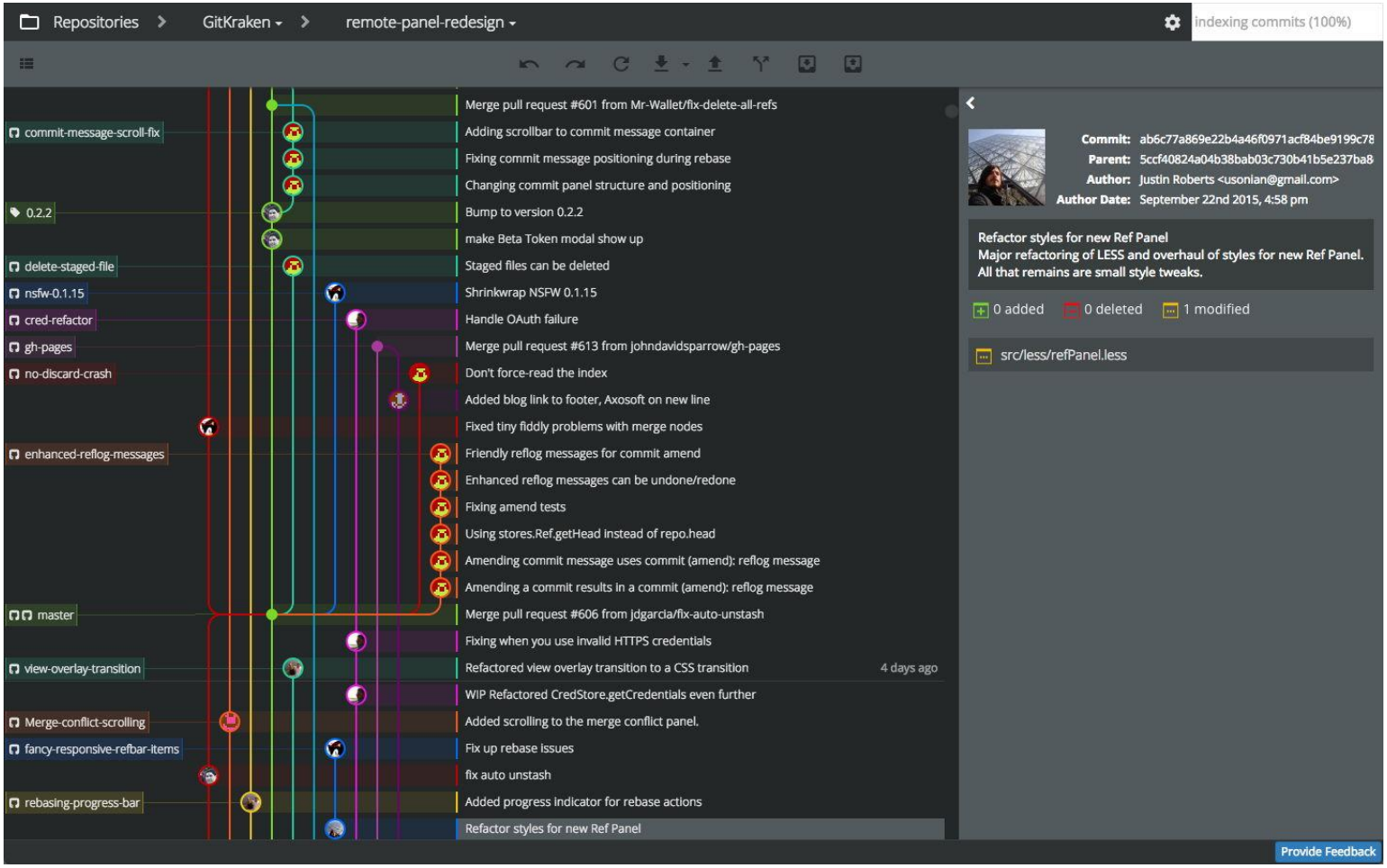

nano? REAL PROGRAMMERS USE emacs
HEY. REAL PROGRAMMERS USE vim.
WELL, REAL PROGRAMMERS USE ed.
NO, REAL PROGRAMMERS USE cat.
REAL PROGRAMMERS USE A MAGNETIZED NEEDLE AND A STEADY HAND.
EXCUSE ME, BUT REAL PROGRAMMERS USE BUTTERFLIES.
THEY OPEN THEIR HANDS AND LET THE DELICATE WINGS FLAP ONCE.
THE DISTURBANCE RIPPLES OUTWARD, CHANGING THE FLOW OF THE EDDY CURRENTS IN THE UPPER ATMOSPHERE.
THESE CAUSE MOMENTARY POCKETS OF HIGHER-PRESSURE AIR TO FORM,
WHICH ACT AS LENSES THAT DEFLECT INCOMING COSMIC RAYS, FOCUSING THEM TO STRIKE THE DRIVE PLATTER AND FLIP THE DESIRED BIT.
NICE.
'COURSE, THERE'S AN EMACS COMMAND TO DO THAT.
OH YEAH! GOOD OL' C-x M-c M-butterfly...
DAMMIT, EMACS.

La *tool stack*

- Les pratiques seules ne suffisent pas
- Si elles sont uniquement théoriques elles peuvent être oubliées
- Il faut des outils pour systématiser leur application
- Les différents outils utilisés dans un développement logiciel s'appellent la *tool stack*
- La tool stack est vitale pour assurer la qualité logicielle

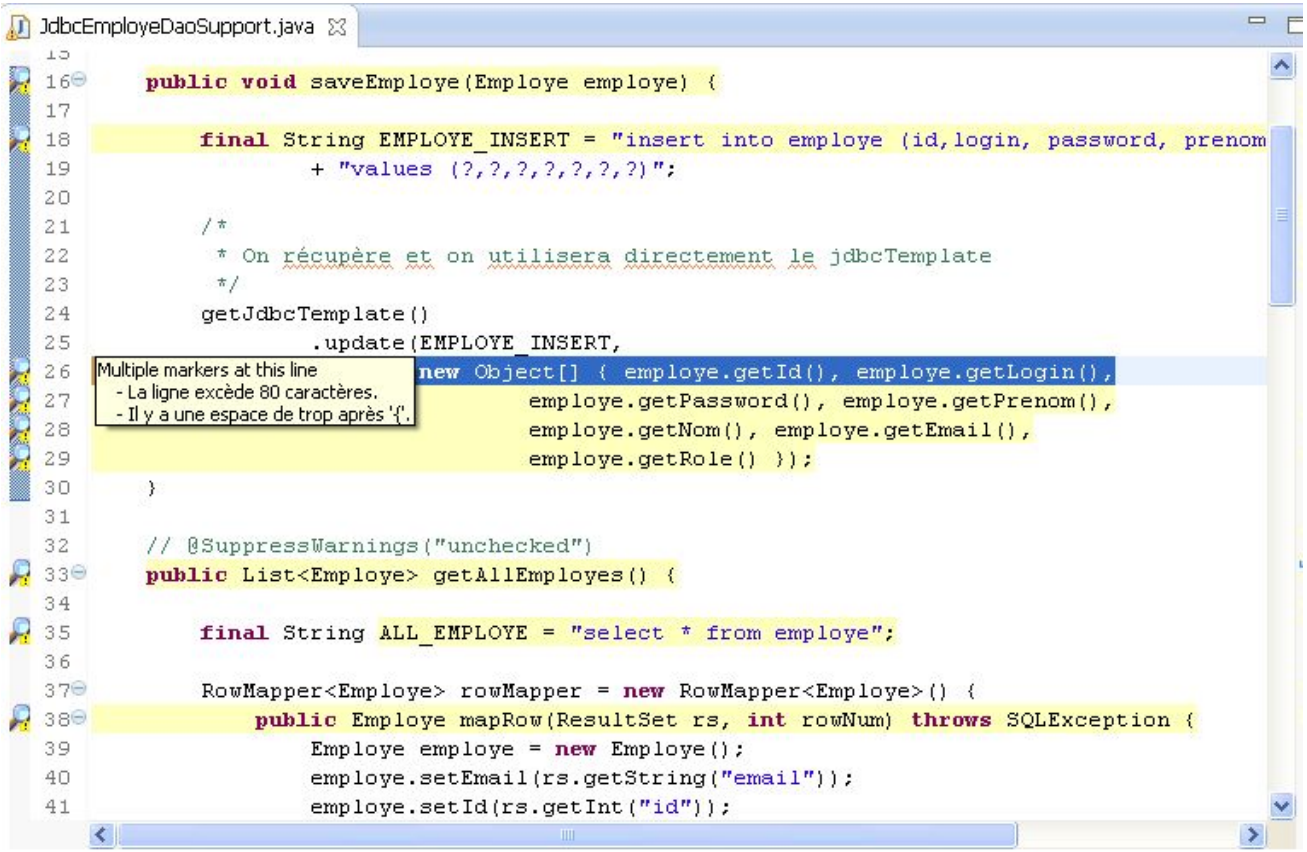
Les VCS

- Permettent de gérer l'historique de développement du code



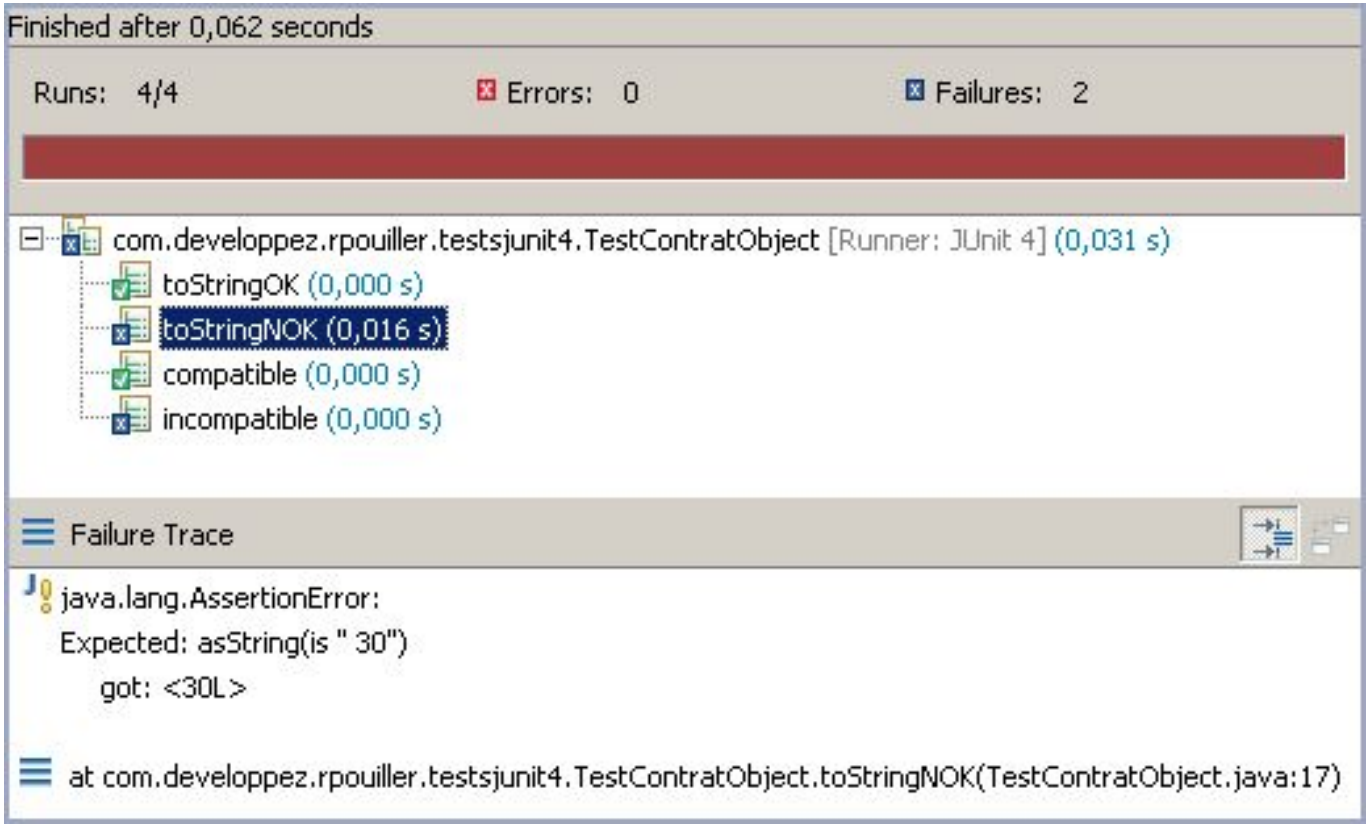
Les linters

- Les linters permettent de vérifier la plupart des principes du clean code (taille des méthodes, formatage, …)

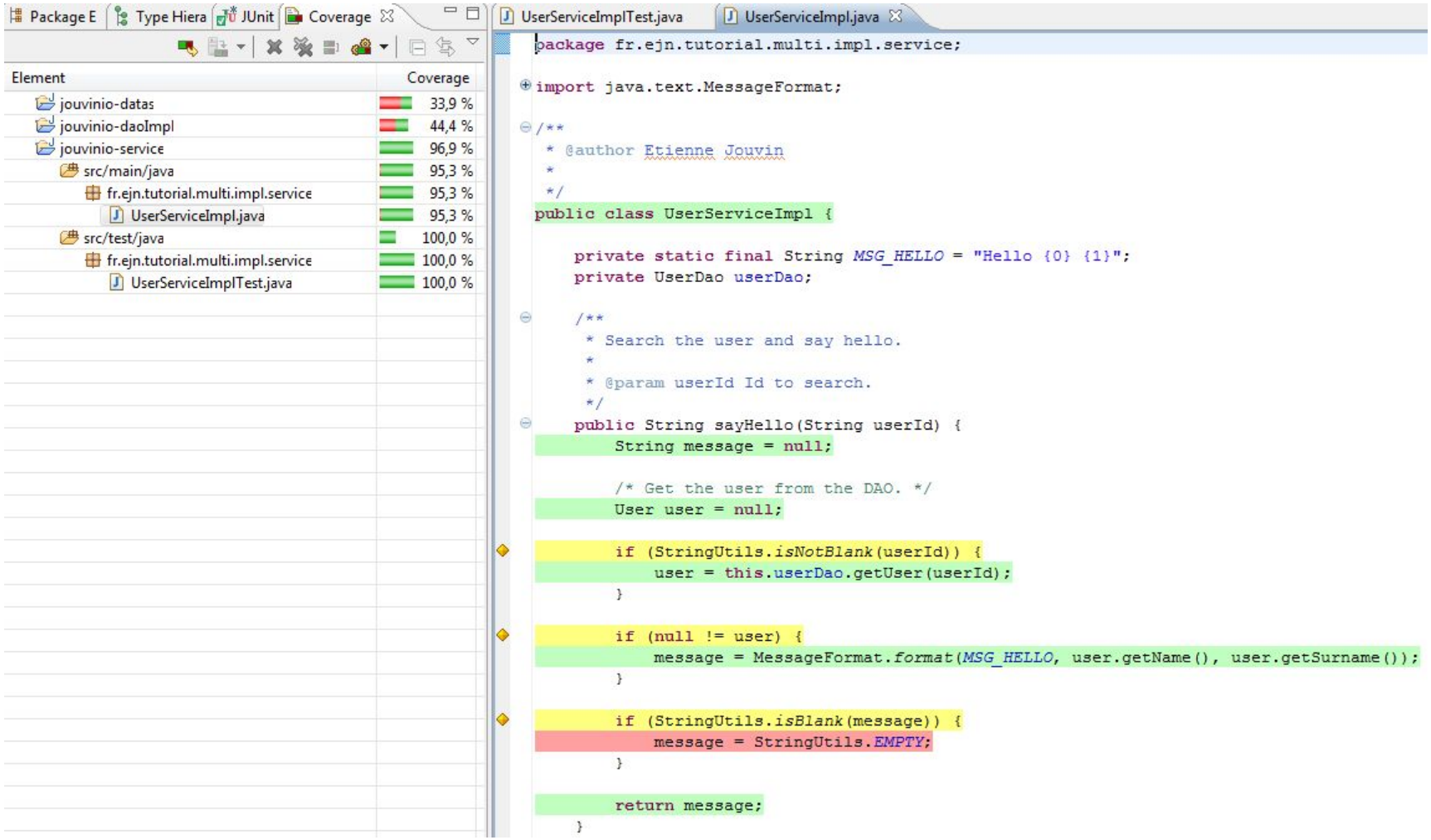


Les runners de test

- Permettent de lancer les tests et de récupérer leurs résultats

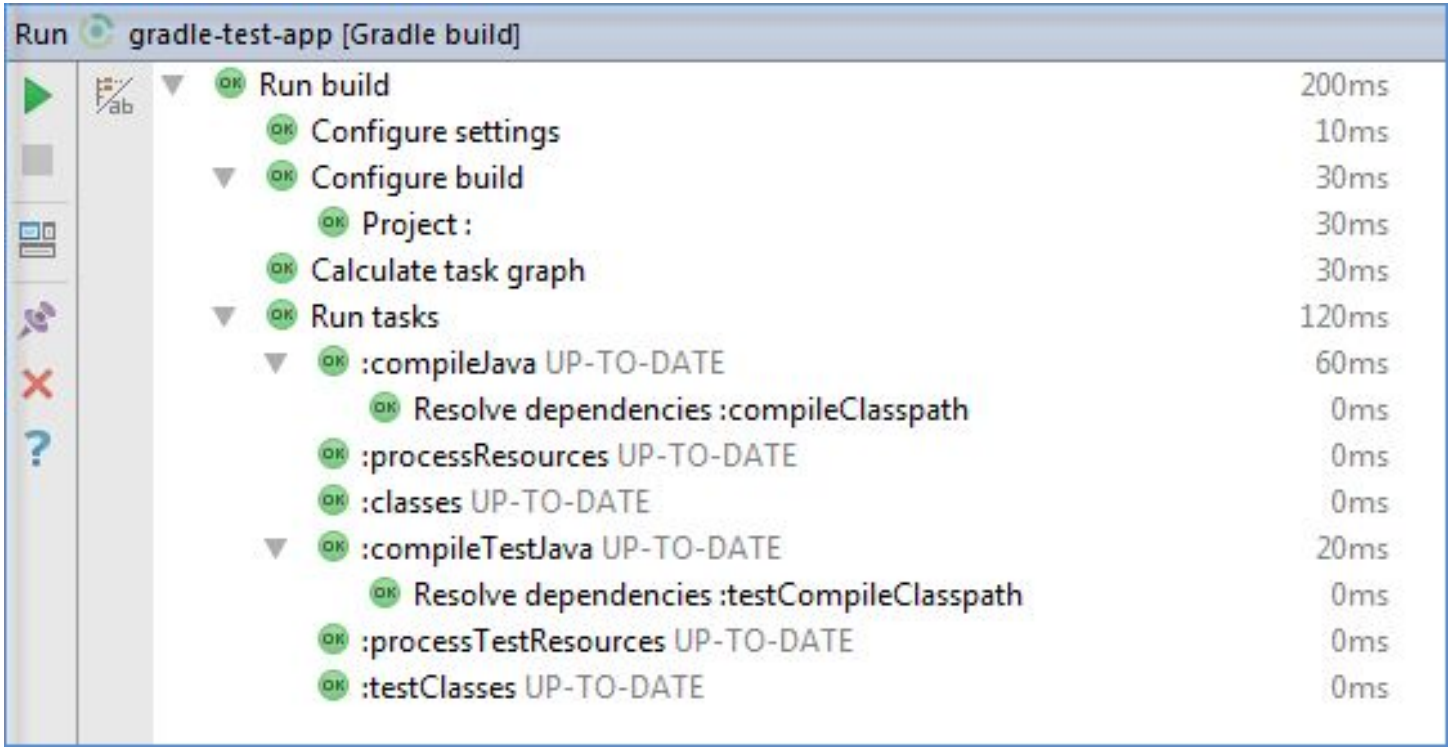


Les outils de couverture



Les *build tools*

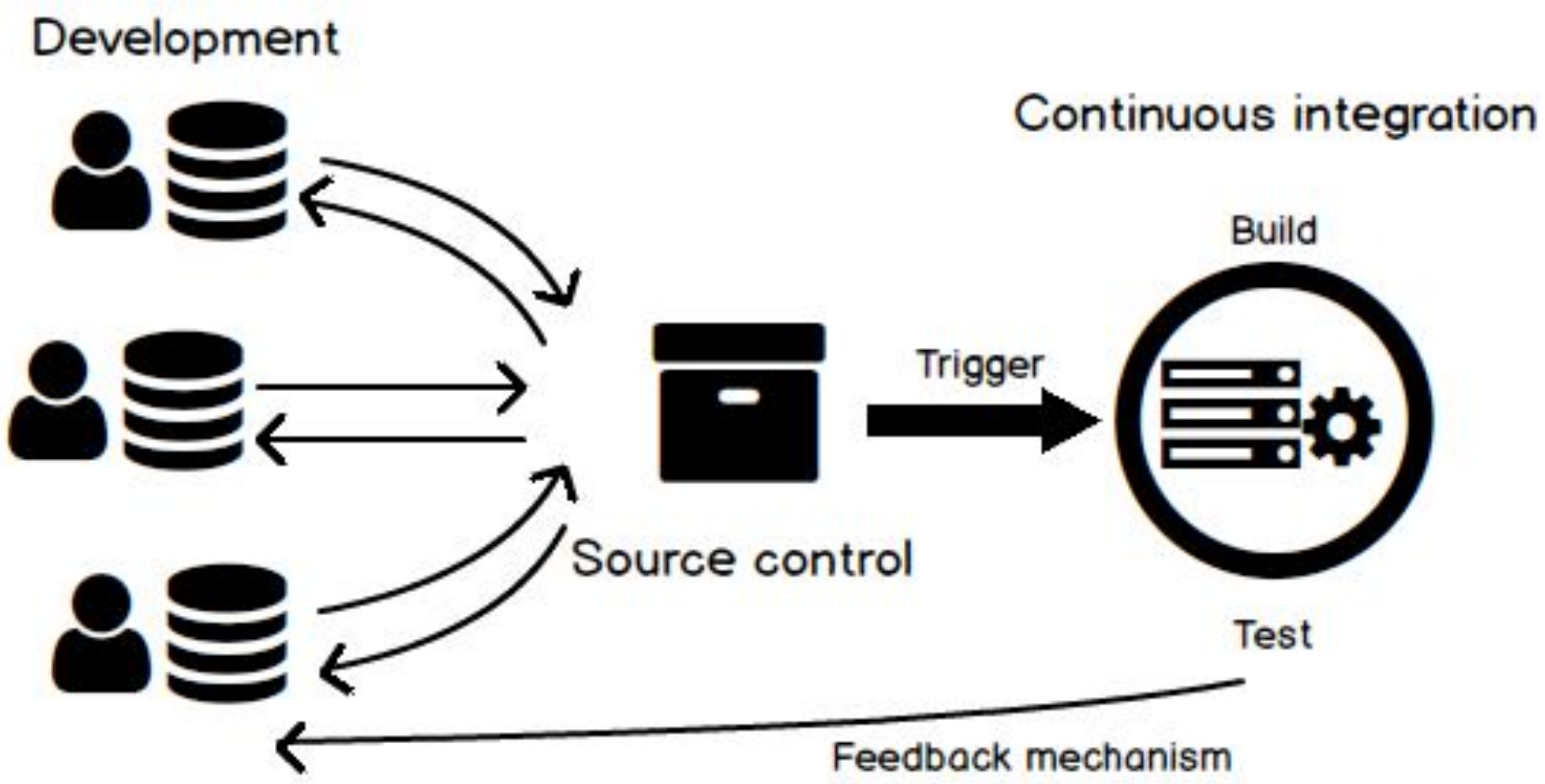
- Permettent de coordonner tous les outils (compilation, linter, test, déploiement, …)



Les différentes façons d’utiliser les outils

- Dans l’IDE
  - Permet un feedback instantané aux développeurs mais peut “polluer“
- En ligne de commande
  - Fournit des résultats à la demande mais il faut penser à le lancer
- Dans les deux cas, la charge est au développeur de penser à utiliser correctement les outils

L’intégration continue



Avantages de l’intégration continue

- Toute la tool stack est appliquée à chaque commit
- Dans un environnement contrôlé
- Quelque soit le développeur

😀

```
script: ./gradlew check
after_success:
  - ./gradlew coveralls uploadArchives
language: java
jdk:
  - oraclejdk8
deploy:
  provider: releases
  skip_cleanup: true
  on:
    branch: master
    tags: true
  file: dist/build/distributions/gumtree.zip
```

GumTreeDiff / gumtree

build passing

Current | Branches | Build History | Pull Requests | More options

✓ develop	Test to remove reflections	#255 passed	7 min 22 sec	
Morandat		1a0655a	2 days ago	
✓ develop	homogeneize property names	#254 passed	8 min 8 sec	
Jean-Rémy Falleri		50e3987	14 days ago	
✓ develop	remove duplicate static property	#253 passed	9 min 24 sec	
Jean-Rémy Falleri		9a1b44d	14 days ago	
✓ develop	downgrade travis cgum dependency	#252 passed	8 min 30 sec	
Jean-Rémy Falleri		194ce24	15 days ago	
✗ develop	update getMappingAsSet method name	#251 failed	7 min 20 sec	
Jean-Rémy Falleri		479ba54	15 days ago	
✗ develop	Merge pull request #66 from FAU-Inf2/optimiz	#250 failed	6 min 55 sec	
Jean-Rémy Falleri		20d3644	15 days ago	

GUMTREEDIFF / GUMTREE / 255

BUILDS	BRANCH	COVERAGE	COMMIT	COMMITTER	TYPE	TIME	VIA
#1	develop	55.61	Test to remove reflections	morandat	push	26 Feb 2018 10:39AM UTC	travis-ci
#254	develop	55.61	homogeneize property names	jrfaller	push	14 Feb 2018 02:21PM UTC	travis-ci
#253	develop	55.57	remove duplicate static property	jrfaller	push	14 Feb 2018 11:19AM UTC	travis-ci
#252	develop	55.58	downgrade travis cgum dependency	jrfaller	push	13 Feb 2018 08:44PM UTC	travis-ci
#251	develop	55.58	update getMappingAsSet method name	jrfaller	push	13 Feb 2018 08:13PM UTC	travis-ci
#249	develop	54.11	fix a bunch of errorprone warnings	jrfaller	push	06 Feb 2018 08:11PM UTC	travis-ci
#248	develop	54.04	remove useless line. fix #78	jrfaller	push	06 Feb 2018 07:08PM UTC	travis-ci
#247	develop	54.03	fix and update dependencies. fix #77	jrfaller	push	05 Feb 2018 07:40PM UTC	travis-ci