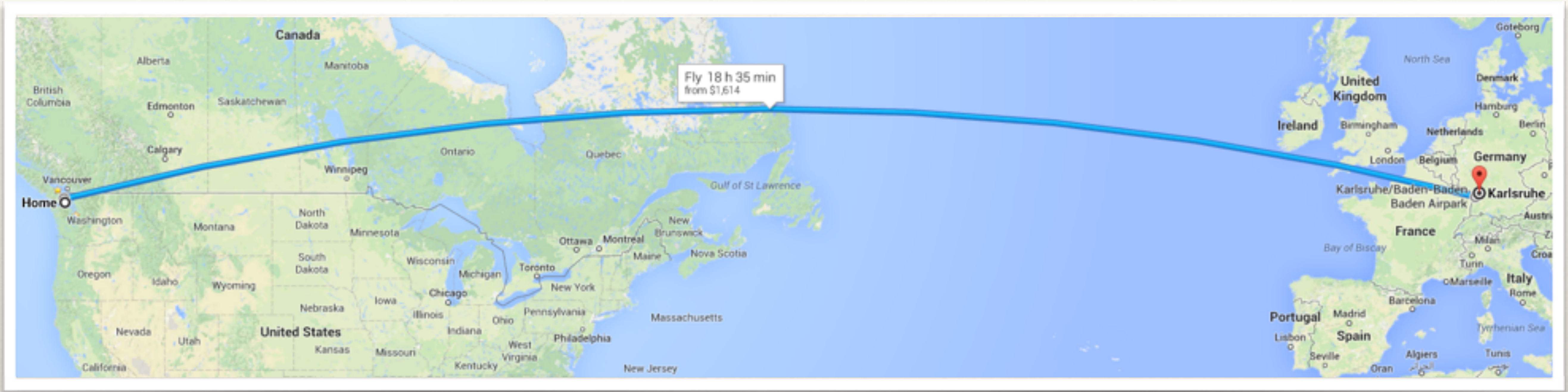
Run Your Project Like It's An

Open Source Project

Dr. R. Ian Bull
EclipseSource

@irbull
ianbull.com



Who am I?

R. Ian Bull

- Completed PhD 2008
- Remote working for 5 years

Greenland
Iceland
Finland
Russia
Canada
United States
Mexico
North Atlantic Ocean
Kazakhstan
Mongolia
China
South Korea
Japan
Italy
Algeria
Libya
Egypt
Iraq
Iran
Afghanistan
Pakistan
Saudi Arabia
Mali
Niger
Sudan
Chad
Nigeria
Ethiopia
Thailand
Venezuela
Colombia
Kenya
DR Congo
Tanzania
Indonesia
Papua New Guinea
Brazil
Peru
Bolivia
Angola
Namibia
Botswana
Madagascar
Indian Ocean
South Atlantic Ocean
Chile
South Africa
Australia
Argentina
New Zealand
Southern

More Accurate

- Open Source 10+ years

(Distributed) Software Development

3 Challenges

1. Communication
2. Requirements / Control
3. Time sinks

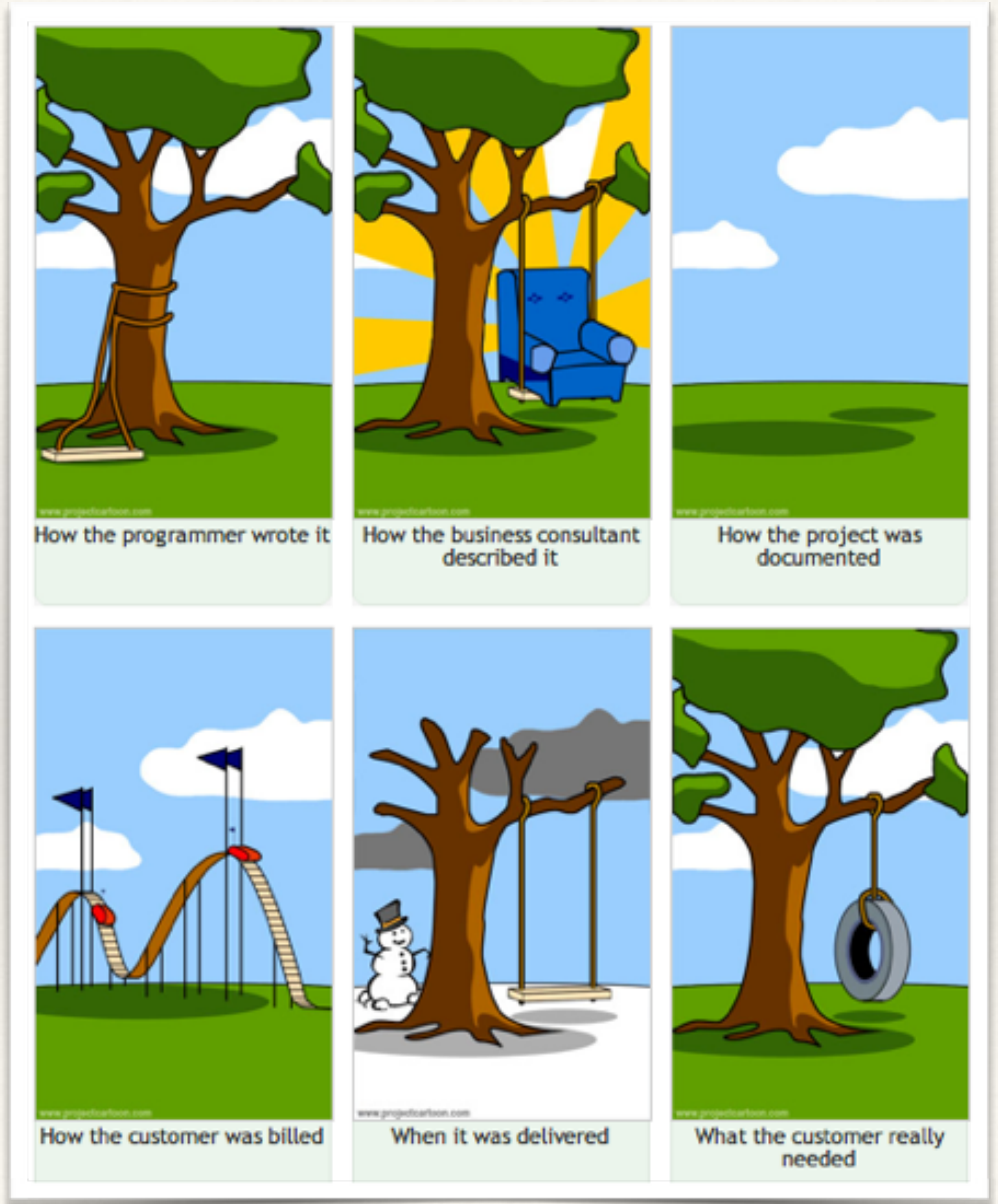
Communication

*If a manager wants to know the **truth** about their project's state,*
*they should **follow** the development team on twitter*

- **Who** is doing what?
- **Who** needs help?
- **What** is the current state of the system?

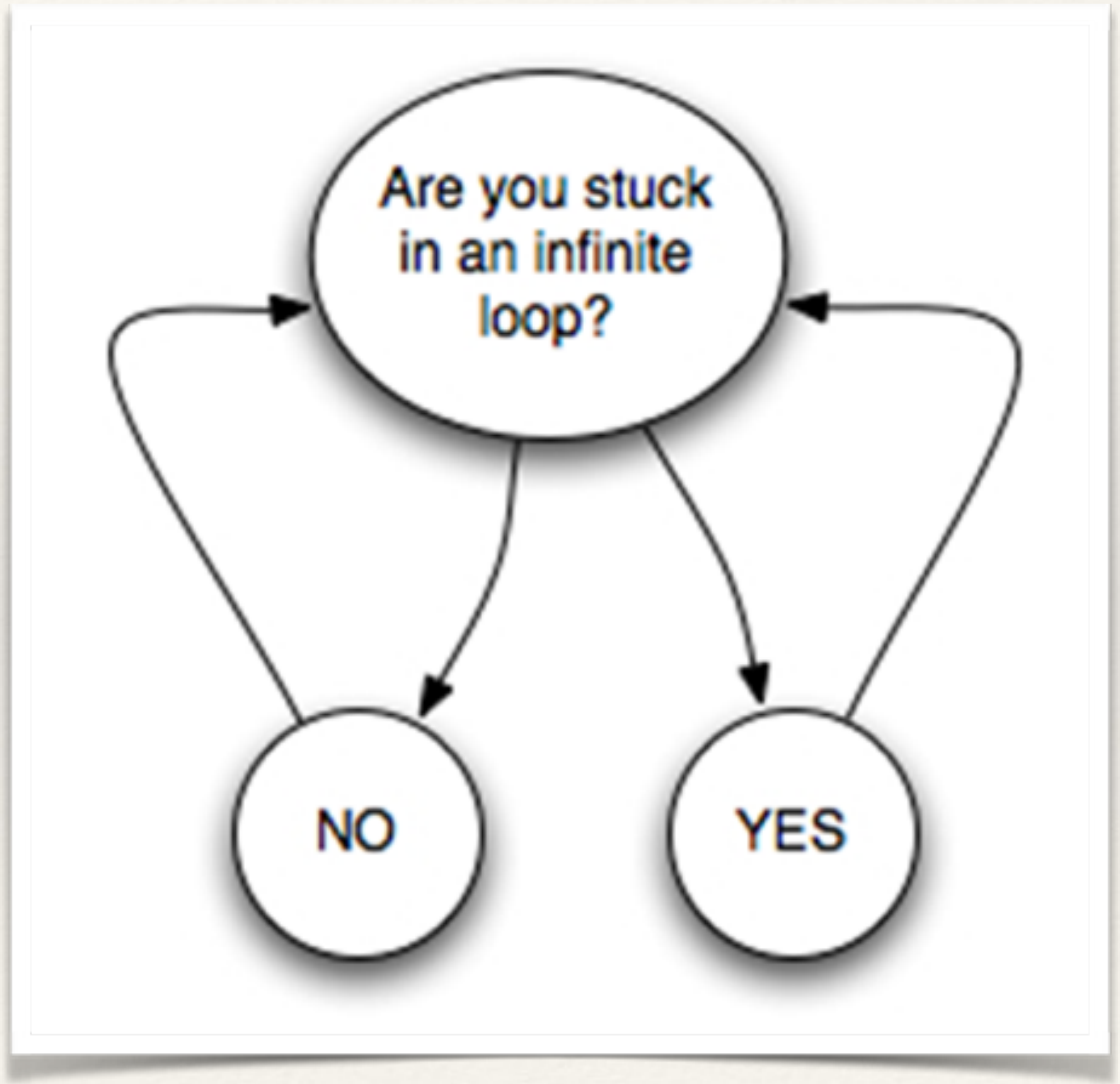
Requirements / Control

- **What** should I be doing next?
- **Who** do I need to talk?
- **Who** is running the show?


How the programmer wrote it
How the business consultant described it
How the project was documented
How the customer was billed
When it was delivered
What the customer really needed

Time Sinks

- Where do I get help?
- Who do I offer help to?
- Am I spinning my wheels?



open source

Are these problems not magnified for OpenSource projects?

What Would OpenSource Do?

- Awareness
- Autonomy
- Automation

Awareness

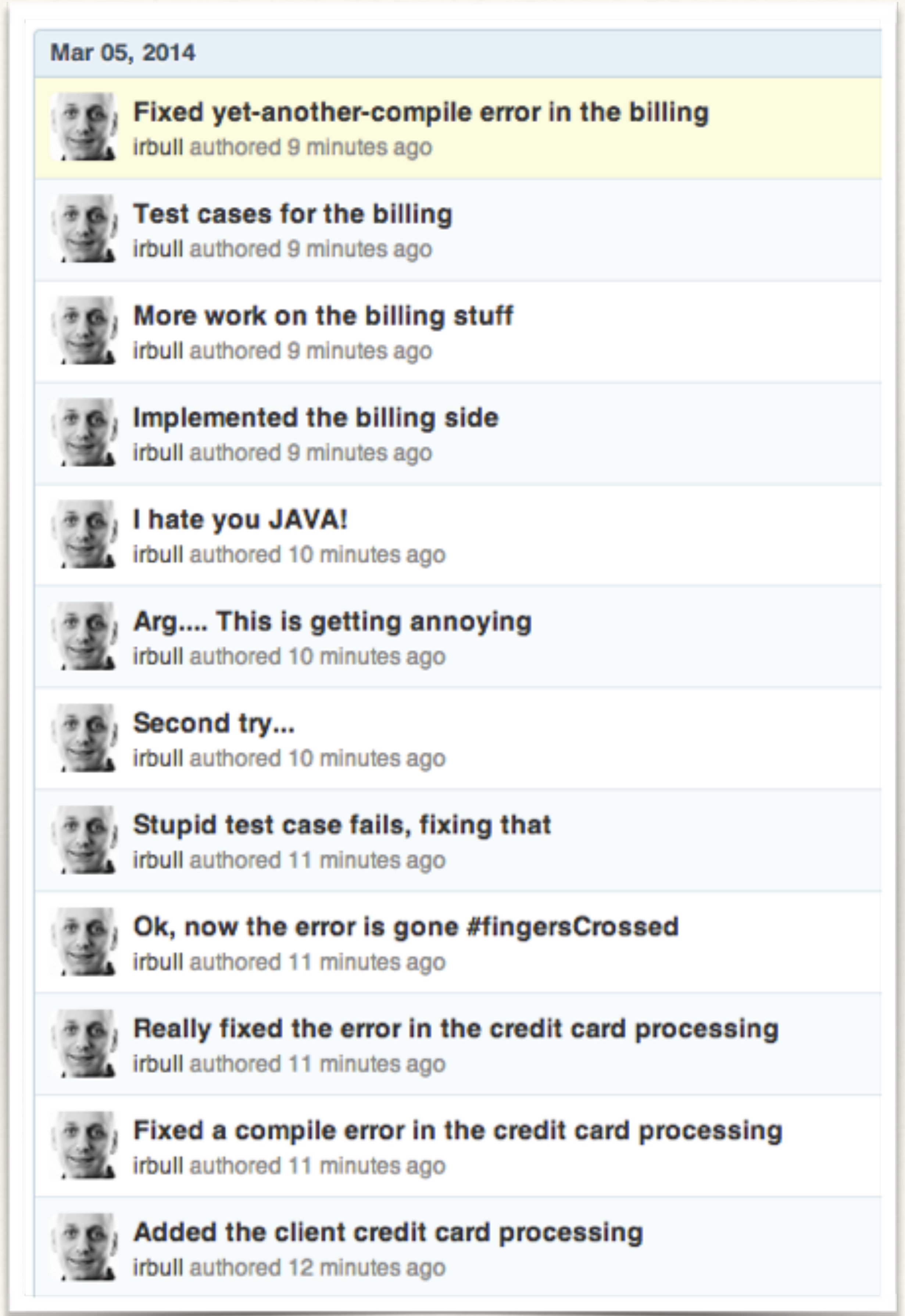
*Team members should **never** send a private email about a project*

- All communication should be **open**
- Have a limited number of communication channels
- Commit history is the **#1 channel**

Those who don't study history are doomed to repeat it

Study History

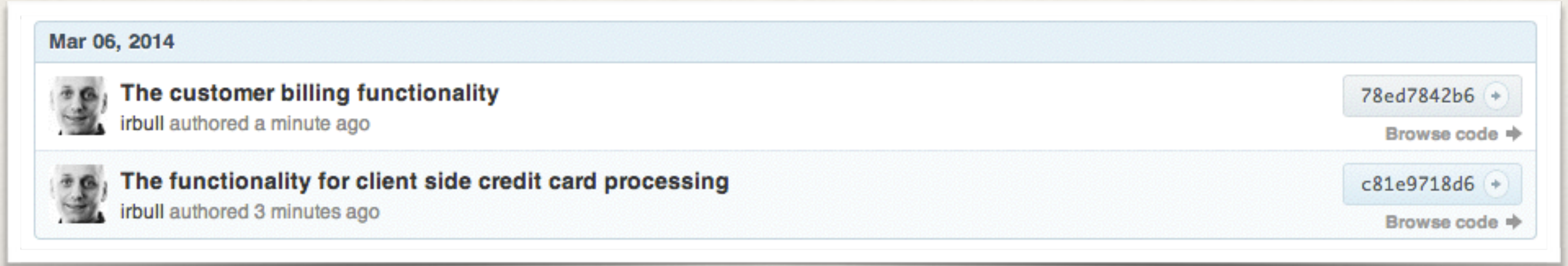
- Read the commit logs each day
- History is (almost) **immutable**
- Make each commit **perfect**



```
Mar 05, 2014
Fixed yet-another-compile error in the billing
irbull authored 9 minutes ago
Test cases for the billing
irbull authored 9 minutes ago
More work on the billing stuff
irbull authored 9 minutes ago
Implemented the billing side
irbull authored 9 minutes ago
I hate you JAVA!
irbull authored 10 minutes ago
Arg.... This is getting annoying
irbull authored 10 minutes ago
Second try...
irbull authored 10 minutes ago
Stupid test case fails, fixing that
irbull authored 11 minutes ago
Ok, now the error is gone #fingersCrossed
irbull authored 11 minutes ago
Really fixed the error in the credit card processing
irbull authored 11 minutes ago
Fixed a compile error in the credit card processing
irbull authored 11 minutes ago
Added the client credit card processing
irbull authored 12 minutes ago
```

Record History

History is not recorded for today,
it's recorded for tomorrow


Mar 06, 2014

The customer billing functionality
irbull authored a minute ago
78ed7842b6
Browse code

The functionality for client side credit card processing
irbull authored 3 minutes ago
c81e9718d6
Browse code

- The most **daft** developer you will meet is **you**, 6 months from now
- The worst developer you will ever meet is **you**, 6 months ago

Commit History

If you spend hours, days or weeks implementing a new feature, take 10 minutes to explain it!

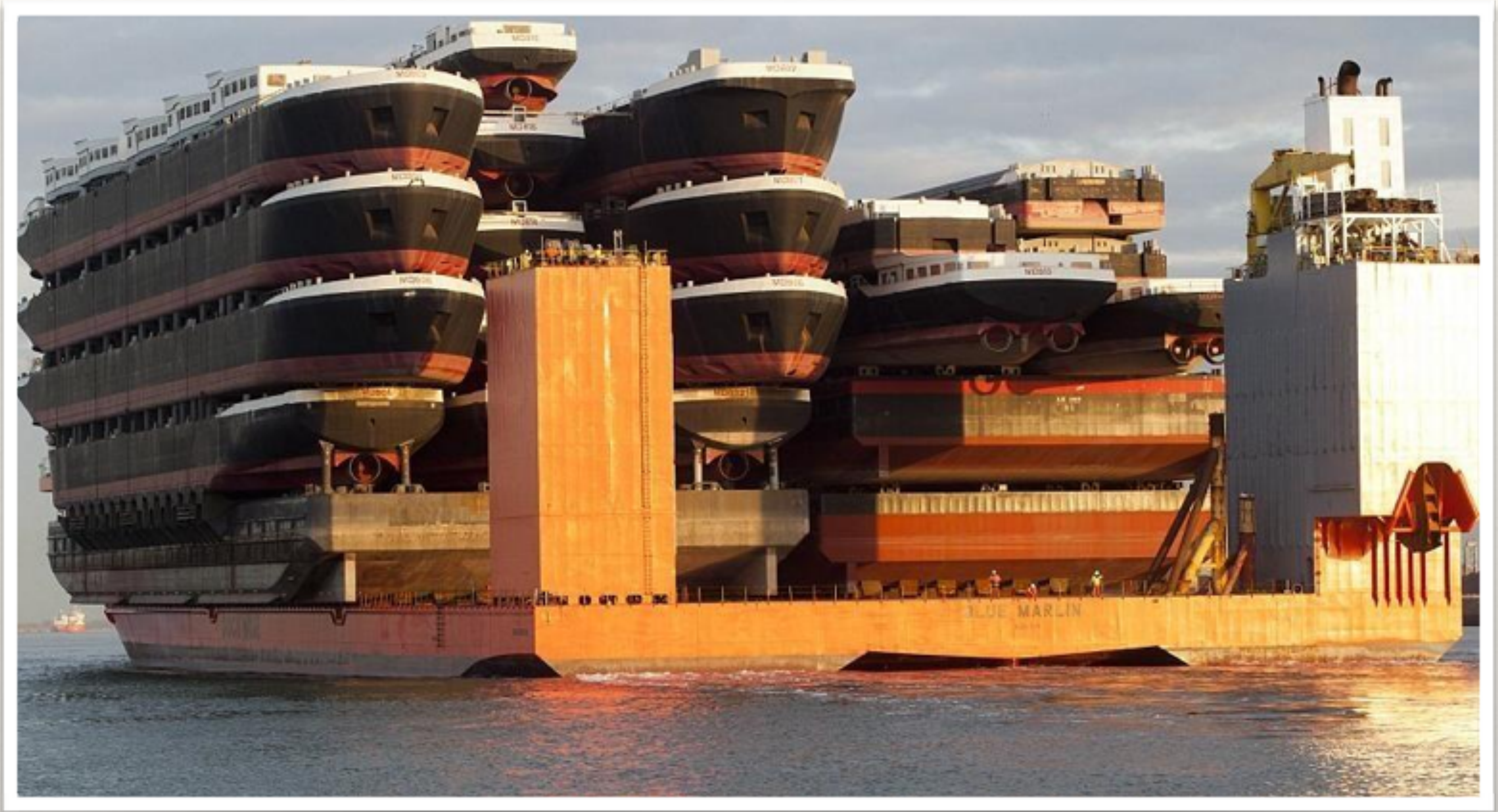
- **Who, Where, When:** Handled for you by the SCM
- **How:** The code itself, should be self describing
- **What:** A short description of the change. Sometimes it's hard to see the forest through the trees
- **Why:** *Most Important!* Why did you make this change, and why did you choose this implementation

Autonomy

Code talks, words walk!

- Developers should always *earn* their stripes
  - **Learn** the system, traps, pits, design and more
  - Earn by doing, not by title (**Meritocracy**)
  - Developers should **use the system**, influence its design

*No developer should be given access to the repository until they've **earned** it,*
*all developers should make a **change** to system on day one*



Continually ship, continually deploy

Ship it!

- Each change results in a **useable** system
- Each change should be **used**

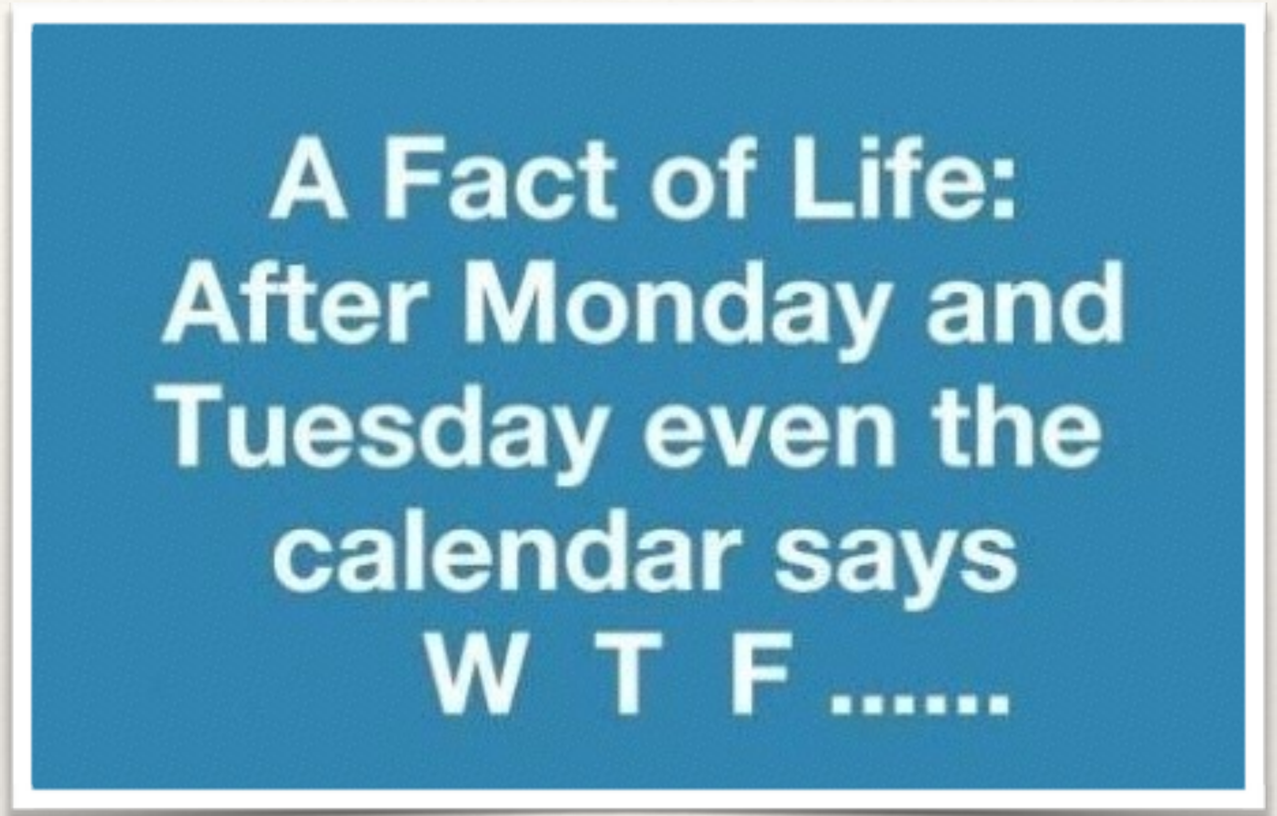
Automation

- Development environment setup should be automated
- Developers should be able to build and test the entire code-base, locally
    - Code **must** compile
    - All regression tests **must** pass
    - Code **must** conform to conventions
    - **Zero** effort should be spent verifying things

This is ***crazy*** *talk*

What?

- **Zero** effort to validate
- Contribute on **day-one** without access
- **Perfect** commits
- **Continuous** use
- Constant information flow



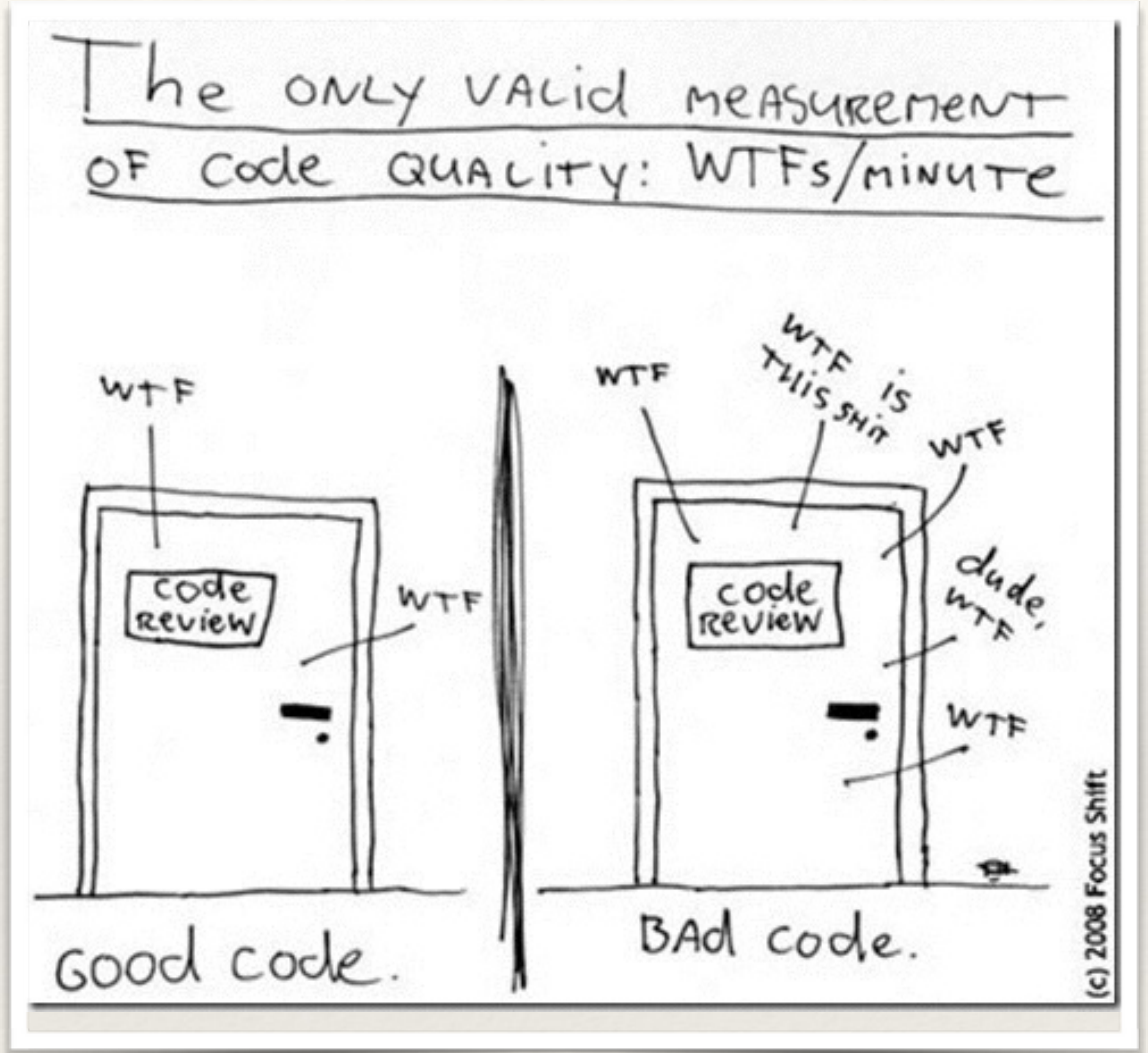
Code Reviews

We sometimes perform code reviews so senior engineers can review the work of the junior developers

- WRONG, **WRONG**, WRONG!
  - Code reviews are for **everyone** on the team
  - **New team members** should review code to better understand the system
  - **Senior developers** should review code to share wisdom
  - A **Smart Ignoramus** should be able to review anyone's code

Code Reviews

- **Anyone** can contribute a change
- Each **accepted** change produces a working system
- Automatically catch careless mistakes
- Use a **distributed** code review system



Summary

- **Anybody** in you organization can contribute to the project
- All **communication** is open
- **Setup** & **Build** should be exactly 2 steps
- Each commit is **perfect**
- The master branch is **always** useable
- **Commit history** tells the story

I'm an egotistical bastard, and I name all my projects after myself. First Linux, now git.
- Linus Torvalds (creator of git)

Introduction to Git

- Commit early
- Commit often

What is Git

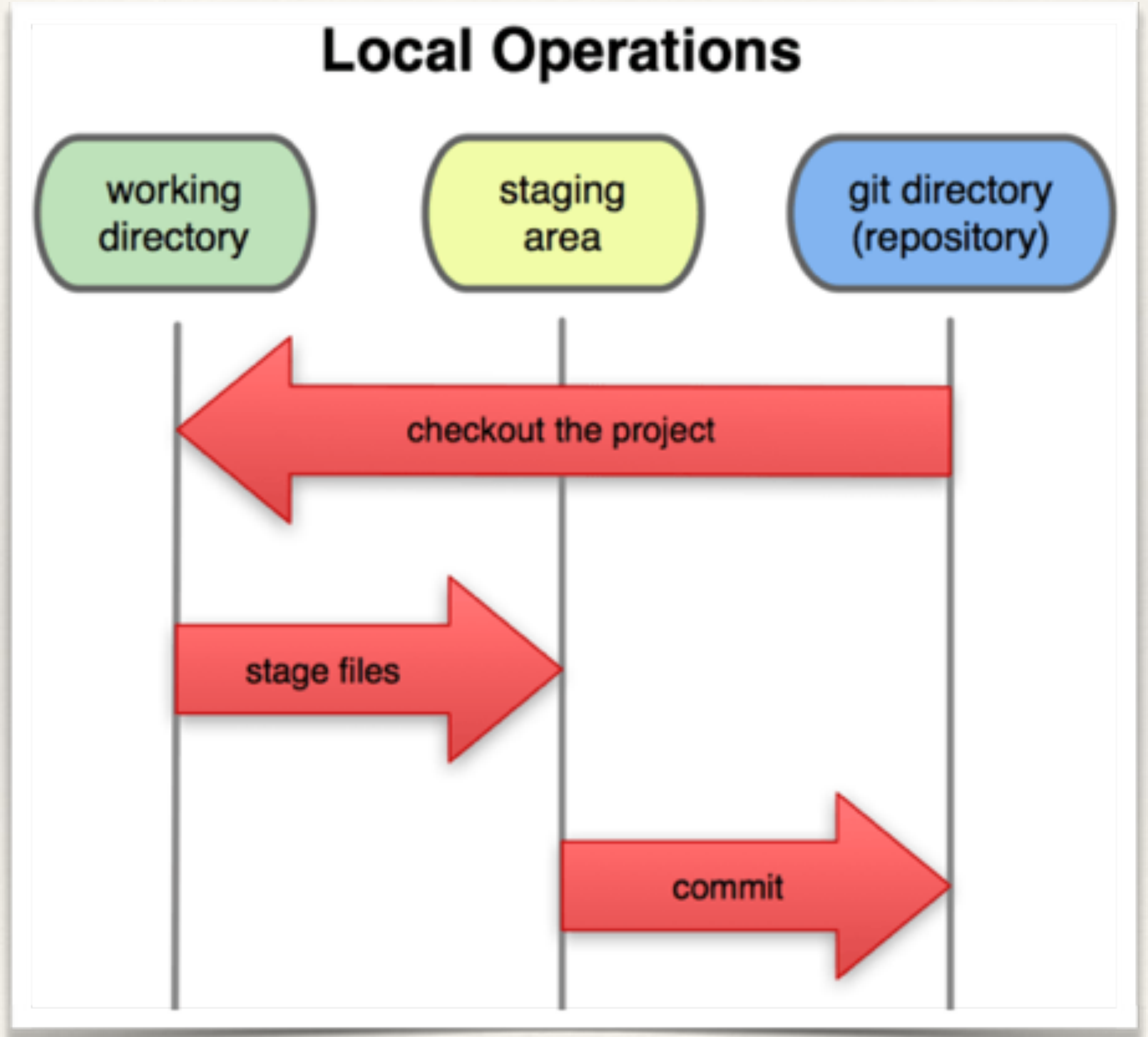
- Distributed version control system
- Your working directory is a full-fledged repository with complete history and full version-tracking capabilities
  - A local copy of the full history is available on your machine

Git Basics

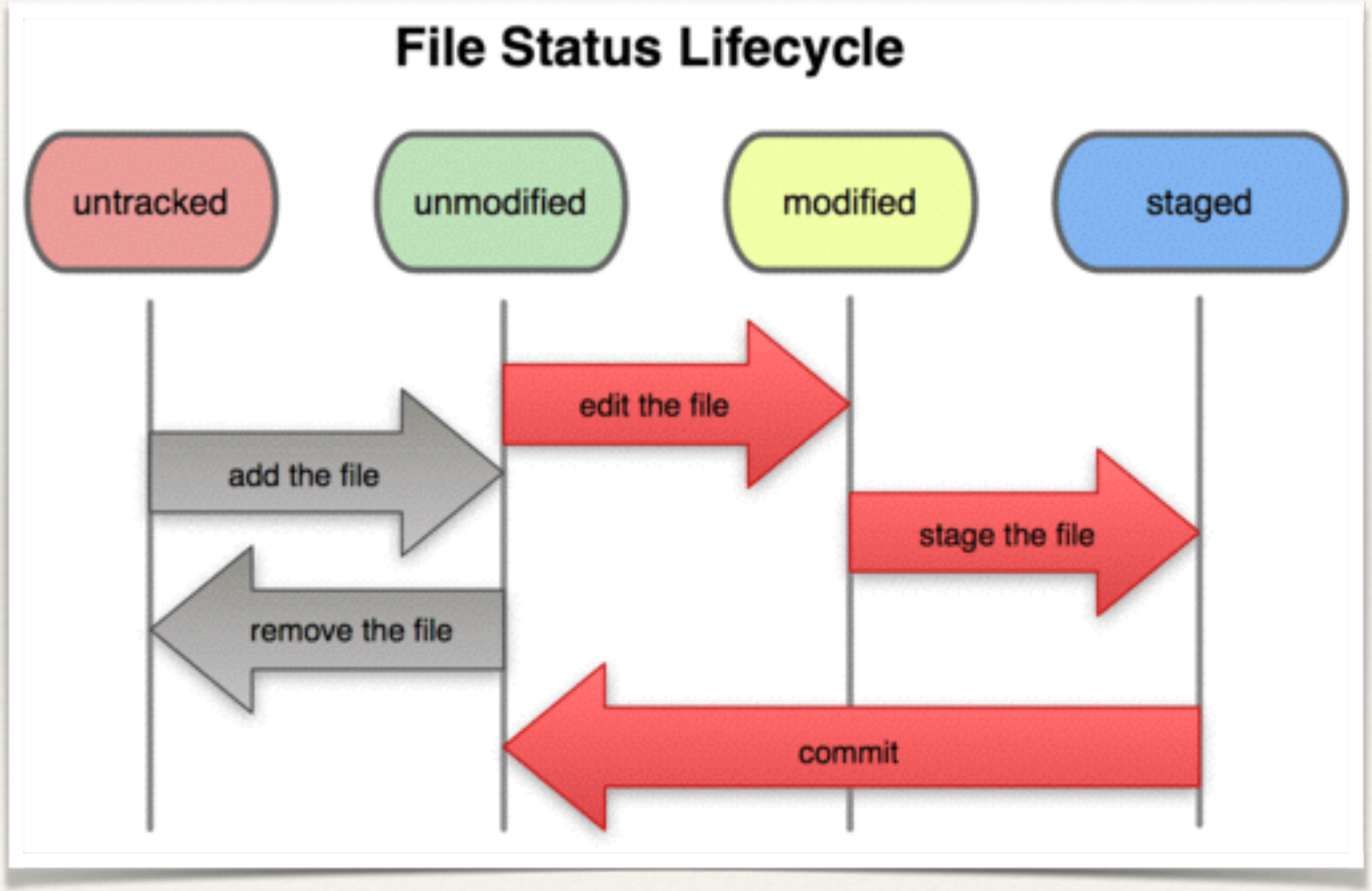
- Nearly every operation is **local**
- Git has **integrity** (everything is **check-summed**)
- Git commands generally only **adds data**
  - **git init** create an empty git repository
  - **git clone** get a copy of an existing repository

Working with Git

- Your files are in 1 of 3 **states**
  - **Committed**: Stored in the repository
  - **Modified**: Changed, but not committed
  - **Staged**: Marked the current version as modified to go into the next commit



Lifecycle


File Status Lifecycle
untracked
unmodified
modified
staged
edit the file
add the file
stage the file
remove the file
commit

Working with Git

- **git add** stage a modified file
- **git status** check the status of your files
- **git commit** commit your files to the repository
- **git log** view the commit history

Collaboration

- **git remote** configure / view the list of remote repositories
- **git fetch** fetch the latest changes from a remote
- **git merge** merge existing changes into your branch
- **git push** push your latest changes to a remote

It is a poor workman who blames his tools

Tools

- Eclipse plug-ins (EGit)
- GitHub for Mac / Windows
- SourceTree, Tower, GitBox
- **Learn command line git!!!**

Resources

- http://git-scm.com/
- https://try.github.io/levels/1/challenges/1
- **http://git-scm.com/book**
- http://git-scm.com/downloads/guis
- **http://eagain.net/articles/git-for-computer-scientists/**

Software Components

- Provide **boundaries** between different parts of the system
- A way to distribute work
  - OS System Calls or **Services**
  - Built in Software **Libraries**
  - Third-party **Libraries**
  - **Web-Services**

PLEASE BE SAFE.
Do not stand, sit, climb or lean on fences.
If you fall, animals could eat you and that might make them sick.
Thank you.

Good fences make good neighbours!

Organizations** which design systems … are constrained to produce designs which are copies of the **communication structures** of these **organizations

- Conway's law

*APIs specify **proper** interaction with a component*

Application Programmer Interface

- Specified in terms of:
  - Operations
  - Inputs
  - Outputs
  - Types

Software Libraries

- A **giant** collection of libraries are freely available
- Bootstrap, Jodatime, Flight.js, Apache Commons, jquery, etc…
- If it's not part of what makes you **unique**, look for **existing solutions**
- Review the **license**, make sure you **understand** it!
- If it's key to your business, make sure you are **involved**

Software Services

- **Machine-to-machine** interaction over a network
  - Twitter, Google Maps, Facebook, GitHub, **ICNDB**
- Web services were originally designed around **XML, WSDL** and **SOAP**
- Most common services simply use **REST**
- Some provide an **SDK**, or library to help you **integrate**

Terms of Service

- Make sure you read and understand the **Terms of Service**
- Your entire business may be at the mercy of those providing the API

twitpic.com

TNSTAAFL / TANSTAAFL / TINSTAAFL

Representational State Transfer

- Constraints — **Client-server**, **Stateless**, **Cacheable**
- **HTTP** Methods:
  - **GET**: Retrieve a representation of the member (*no side effects*)
  - **PUT**: Replace the member with new information (***idempotent,*** *can be called multiple times*)
  - **POST**: Create a new member
  - **DELETE**: Delete the member (***idempotent,*** *can be called multiple times*)

*No **official standard** for RESTful APIs. **SOAP** is a standard, **REST** is an **architecture style***

Data Formats

- REST can be used with standard such as XML, URI, HTTP
- Often used with JSON (JavaScript Object Notation)
    - A collection of name / value pairs (Objects)
    - An ordered list of values (Array)
    - Names can be a **String**
    - Values can be a **String, Number, Object, Array, Boolean**

API Example

- Let's build an application:

  http://example.ianbull.com/chuck

- http://api.icndb.com/jokes/random?limitTo=nerdy

```
{ "type": "success",
  "value": {
     "id": …,
      "joke": …
   }
}
```

More Complex Example

- Embed **Google Maps** into your Application
  - Interactive events (click, drag, mouse over, etc…)
  - Controls (zoom, pan, map type, rotate, etc…)
  - Overlays (markers, lines, info window, etc…)
  - Layers (heatmap, traffic, transit, weather, etc…)
- Google Maps (~~requires~~) an **API key**

  https://code.google.com/apis/console
- https://developers.google.com/maps/

Architecting Your System

- Design your **own API** between Client and Server
- Write your **tests** against your API
- Integrate **early**, integrate **often**
- Use components as a opportunity to **split work**
- But, *ensure everyone has a working knowledge of the entire system*

Summary

- Look for and reuse **existing components**
- Look for **existing services**
- Read the API!
- Build your own **API** and **component boundaries**
- **License** and **Terms of Services** are important