Challenges to Make Software Good, Safe, and Reliable

September 6, 2022

Byung-Gon Chun

(Slide credits: George Candea, EPFL)

Reminder

- **Watch** the github https://github.com/swsnu/swppfall2022

- HW1 out: 9/1 (Thur); due 9/9 (Fri) 6pm
- HW2 out: 9/7 (Wed); due 9/19 (Mon) 6pm

- Project team formation due 9/13 (Tue) 6pm
- Project proposal due 9/28 (Wed) 6pm

Project Proposal

- Write a 2-page project proposal on a web service software system you would like to build this semester.
- Think ambitiously. Imagine that you create a new start-up. You can propose a new web service that innovates e-commerce, social media, education, multi-player gaming, hyperlocal service, etc.
- In the proposal, you need to justify why the service is interesting to build and what new feature(s) it brings to market.

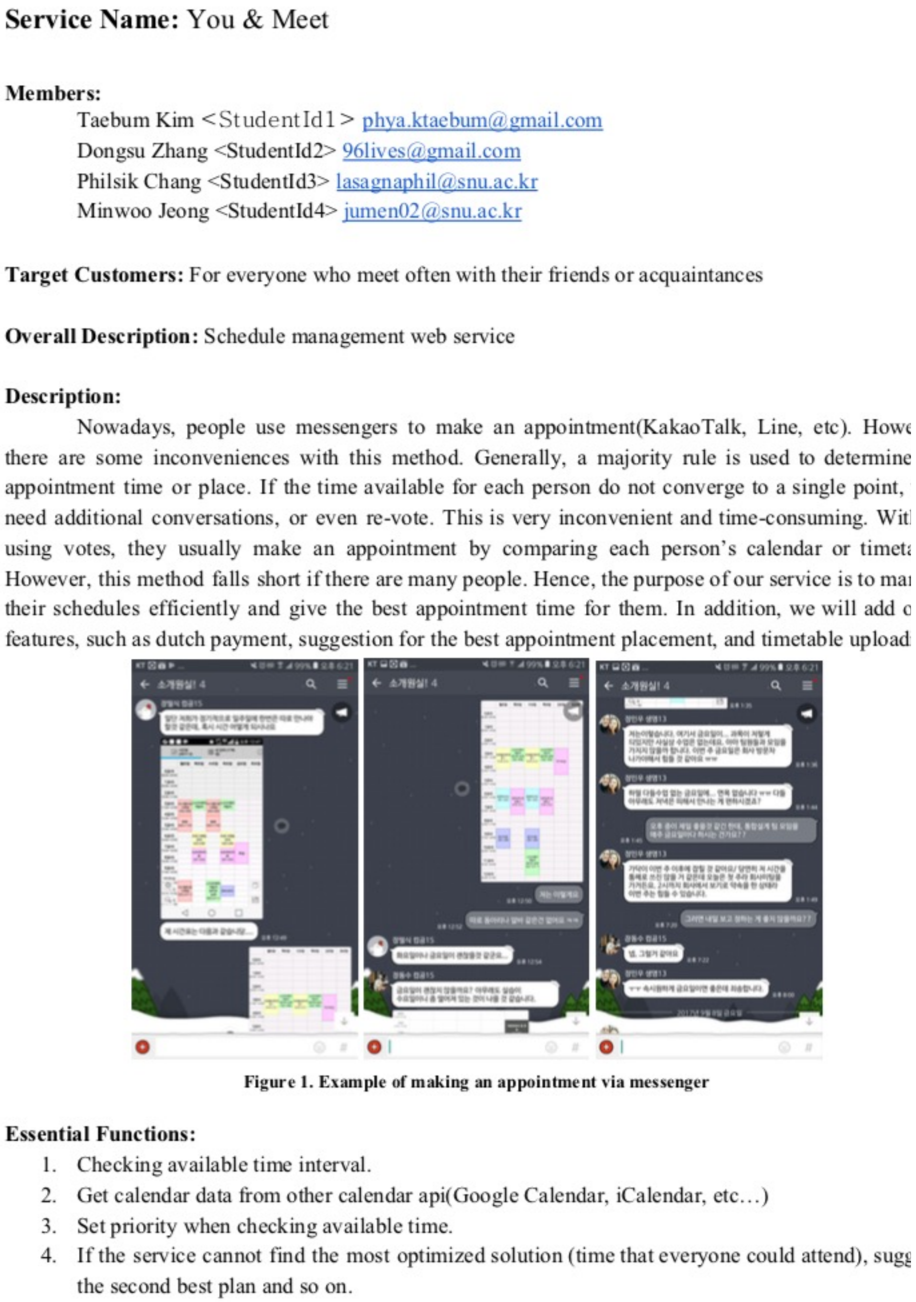
Project Proposal

- In particular, your proposal should include:
  - team information
  - a title (it can be changed later)
  - your target customers
  - what feature the system would add or what problem the system would solve and why the system is important and exciting (including ML features if you have)
  - what the system would do
  - how you will test and demo the system
- Send your proposal in pdf or doc to swpp-staff@spl.snu.ac.kr
- The proposal is due 6:00PM, 9/22 (Tue).
- Please write your proposal in *English*!
  All project docs should be written in *English*!

Come up with Catchy Projects!

- LinkedIn: Connect, share ideas, and discover opportunities
- Uber: Moving people
- Facebook: Connect with friends and the world around you on Facebook
- Github: Build software better, together
- …

Proposal Examples



Service Name: You & Meet

Members:

Taebum Kim <StudentId1> phya.ktaebum@gmail.com
Dongsu Zhang <StudentId2> 96lives@gmail.com
Philsik Chang <StudentId3> lasagnaphil@snu.ac.kr
Minwoo Jeong <StudentId4> jumen02@snu.ac.kr

Target Customers: For everyone who meet often with their friends or acquaintances

Overall Description: Schedule management web service

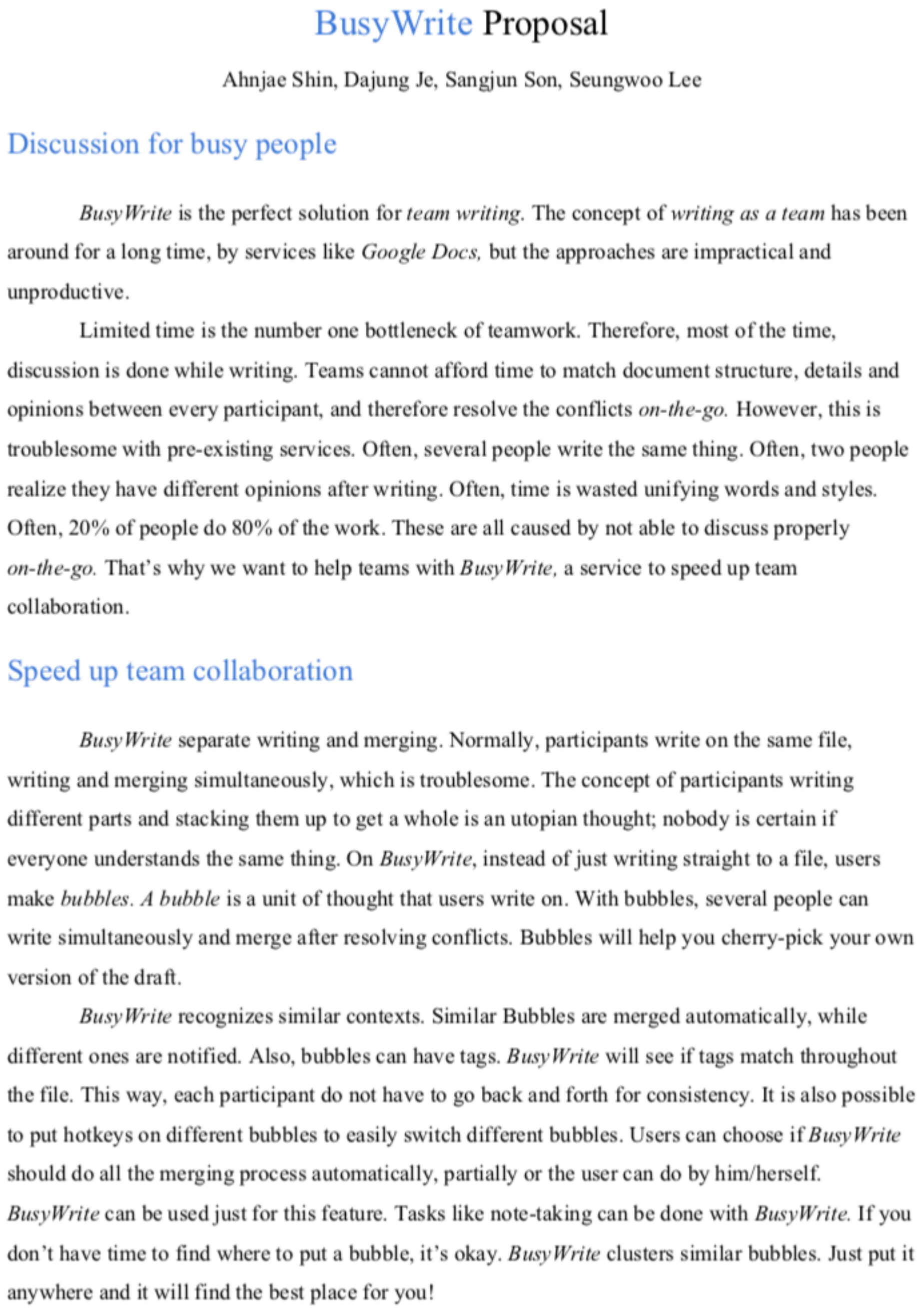
Description:

Nowadays, people use messengers to make an appointment(KakaoTalk, Line, etc). Howe there are some inconveniences with this method. Generally, a majority rule is used to determine appointment time or place. If the time available for each person do not converge to a single point, need additional conversations, or even re-vote. This is very inconvenient and time-consuming. Wit using votes, they usually make an appointment by comparing each person's calendar or timeta However, this method falls short if there are many people. Hence, the purpose of our service is to mai their schedules efficiently and give the best appointment time for them. In addition, we will add o features, such as dutch payment, suggestion for the best appointment placement, and timetable upload

Figure 1. Example of making an appointment via messenger

Essential Functions:

1. Checking available time interval.
2. Get calendar data from other calendar api(Google Calendar, iCalendar, etc…)
3. Set priority when checking available time.
4. If the service cannot find the most optimized solution (time that everyone could attend), sugg the second best plan and so on.



BusyWrite Proposal

Ahnjae Shin, Dajung Je, Sangjun Son, Seungwoo Lee

Discussion for busy people

BusyWrite is the perfect solution for *team writing*. The concept of *writing as a team* has been around for a long time, by services like *Google Docs*, but the approaches are impractical and unproductive.

Limited time is the number one bottleneck of teamwork. Therefore, most of the time, discussion is done while writing. Teams cannot afford time to match document structure, details and opinions between every participant, and therefore resolve the conflicts *on-the-go*. However, this is troublesome with pre-existing services. Often, several people write the same thing. Often, two people realize they have different opinions after writing. Often, time is wasted unifying words and styles. Often, 20% of people do 80% of the work. These are all caused by not able to discuss properly *on-the-go*. That's why we want to help teams with *BusyWrite*, a service to speed up team collaboration.

Speed up team collaboration

BusyWrite separate writing and merging. Normally, participants write on the same file, writing and merging simultaneously, which is troublesome. The concept of participants writing different parts and stacking them up to get a whole is an utopian thought; nobody is certain if everyone understands the same thing. On *BusyWrite*, instead of just writing straight to a file, users make *bubbles. A bubble* is a unit of thought that users write on. With bubbles, several people can write simultaneously and merge after resolving conflicts. Bubbles will help you cherry-pick your own version of the draft.

BusyWrite recognizes similar contexts. Similar Bubbles are merged automatically, while different ones are notified. Also, bubbles can have tags. *BusyWrite* will see if tags match throughout the file. This way, each participant do not have to go back and forth for consistency. It is also possible to put hotkeys on different bubbles to easily switch different bubbles. Users can choose if *BusyWrite* should do all the merging process automatically, partially or the user can do by him/herself. *BusyWrite* can be used just for this feature. Tasks like note-taking can be done with *BusyWrite*. If you don't have time to find where to put a bubble, it's okay. *BusyWrite* clusters similar bubbles. Just put it anywhere and it will find the best place for you!

Why Study Software Engineering?

- How to write software that is
  - Safe
  - Secure
  - Reliable
  - High-performance
  - Manageable
- Improve developers’ productivity
- Stimulate programmers’ creativity
- Make it cheaper to build good software

What makes it challenging to write good, safe, and reliable software?

Challenges to Make Software Good, Safe, and Reliable

1. Complexity = The #1 Challenge

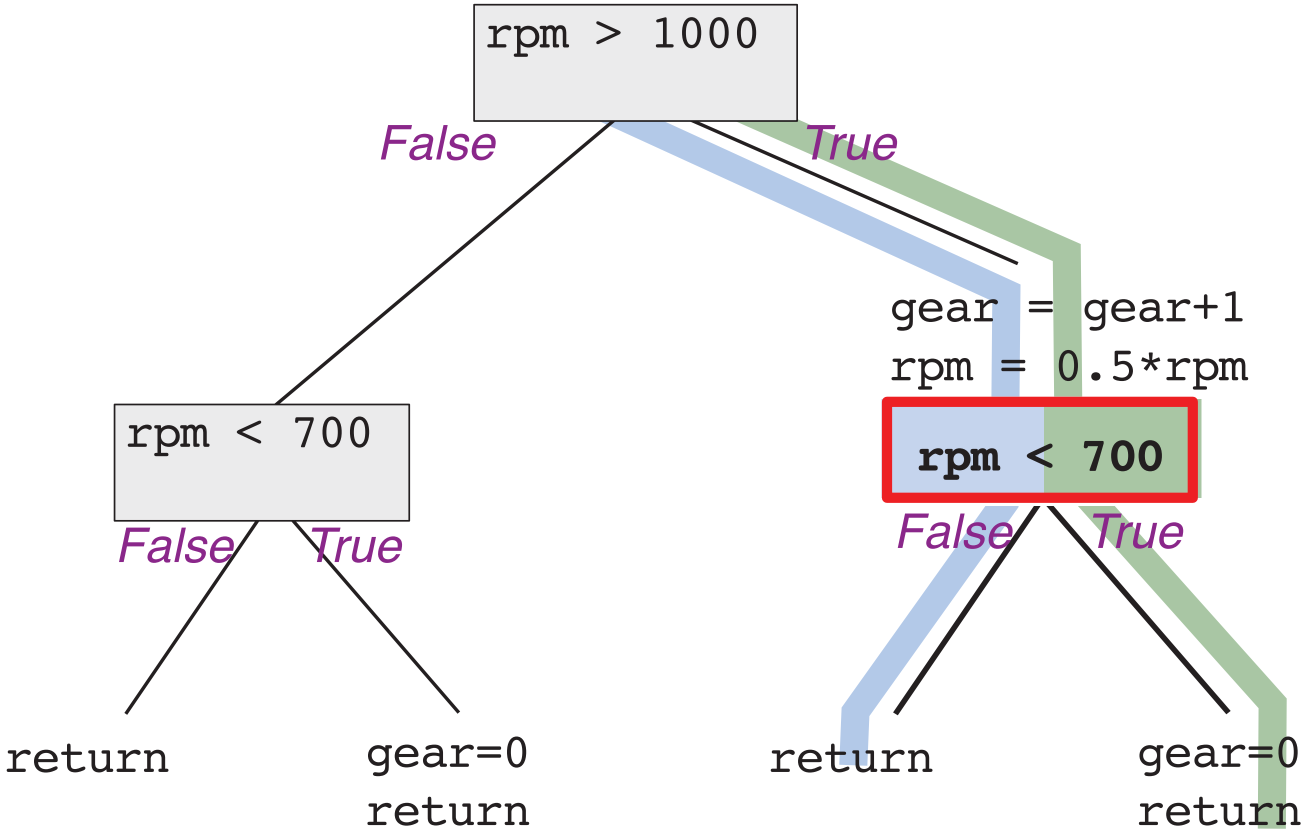
- How to handle it?
  Taming Complexity
  (including Preventing and Finding Errors Early)

2. Management Challenge

- How to handle it?
  Managing Your Software Project

Complexity = The #1 Challenge

```
autoShift (int rpm)
  if (rpm > 1000)
    gear = gear+1
    rpm = 0.5*rpm
  if (rpm < 700)
    gear=0
  return
```

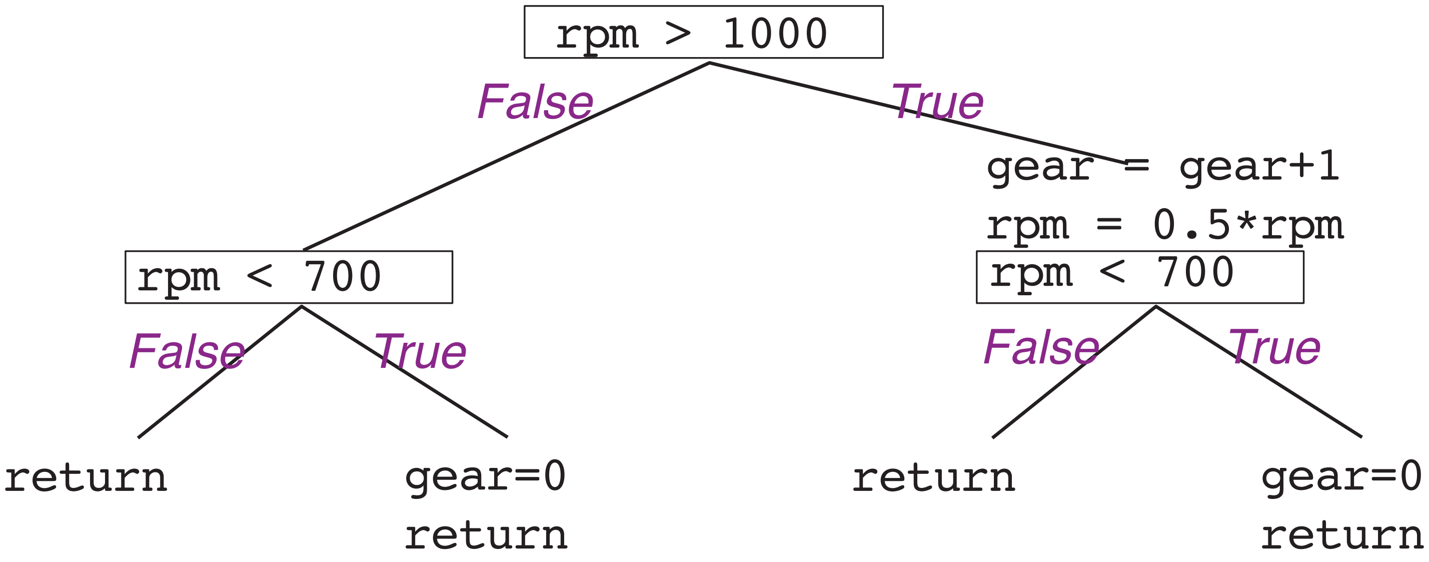


How do you ensure that this code works correctly?

Microsoft Google

rpm E {0,350,944,1200,1800}

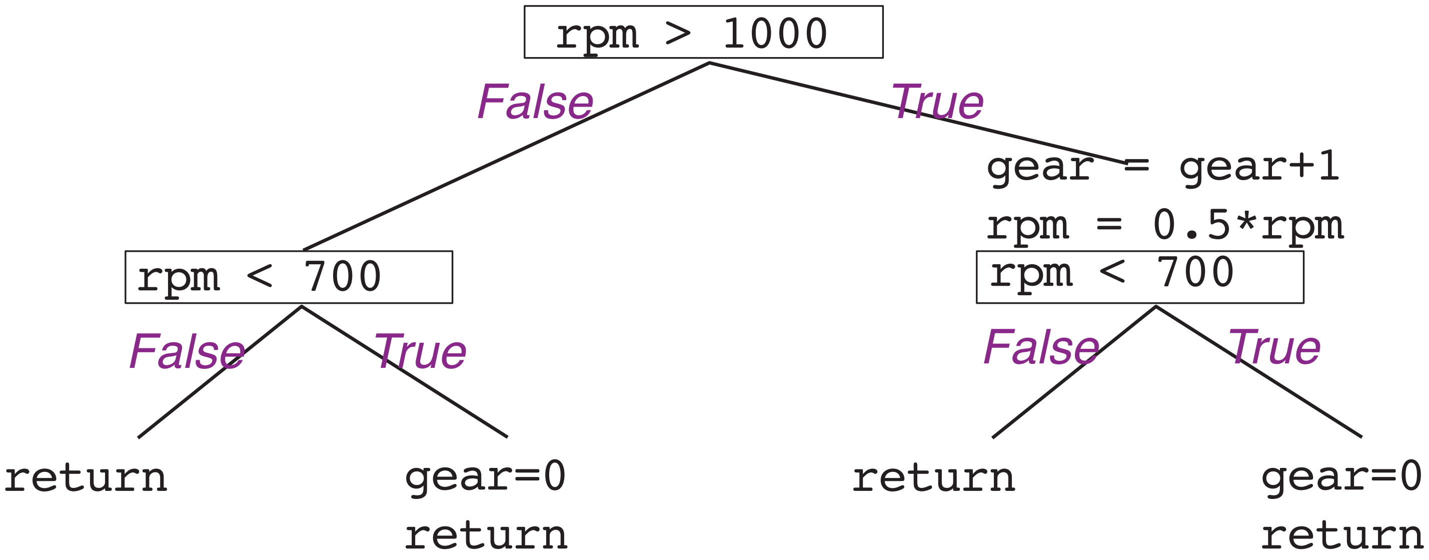
```
autoShift (int rpm)
  if (rpm > 1000)
    gear = gear+1
    rpm = 0.5*rpm
  if (rpm < 700)
    gear=0
  return
```



How many execution paths?

$$\text{paths} = 2^{\text{program size}}$$

```
autoShift (int rpm)
  if (rpm > 1000)
    gear = gear+1
    rpm = 0.5*rpm
  if (rpm < 700)
    gear=0
  return
```



rpm > 1000

False

True

gear = gear+1

rpm = 0.5*rpm

rpm < 700

False

True

return

gear=0

return

rpm < 700

False

True

return

gear=0

return

$$\texttt{paths} = 2^{\texttt{program size}}$$

>5,000,000 lines of code[1] (LOC) ⇒ ~$2^{500,000}$ paths

Can we test $2^{500,000}$ paths?

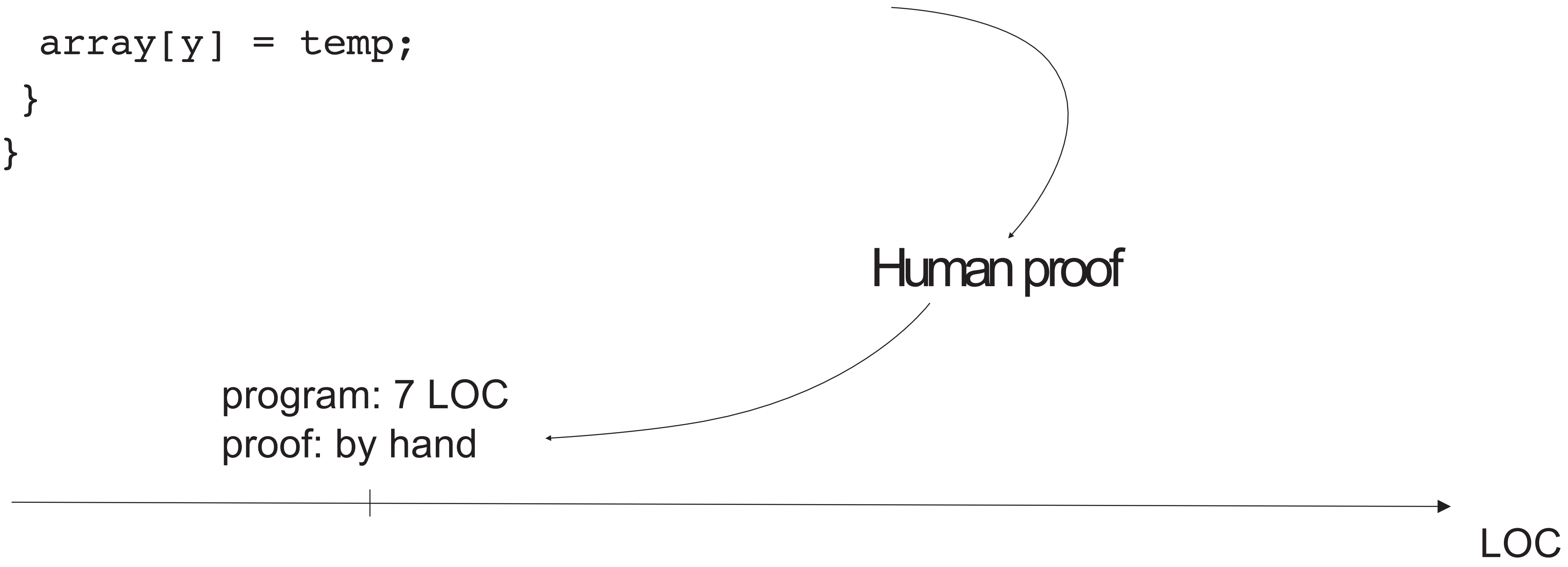
- 30 picoseconds/test ⇒ $2^{499,968}$ years to finish
  - planet Earth is ~2^{32} years old
- 1 bit/test ⇒ $2^{499,957}$ Terabytes to store the answer
  - Universe contains ~2^{266} atoms in total

[1] Black Duck Software, Inc. *Mozilla Firefox Code Analysis*, http://www.ohloh.net/p/firefox/analyses/latest (May 20, 2012)

Software Verification

- Testing
  - – Most basic form of software verification
  - – Naïve approach does not offer a lot of confidence
- Human proofs
  - – Mathematical view of software engineering
  - – Write code and prove it correct right away
- Machine proofs
  - – Do the same as human proofs, but using a computer
  - – Key element is to automatically prove correctness

```
void bubbleSort( int array[], int length )
{
 for (int x=0; x<n; x++) {
  for (int y=0; y<n-1; y++) {
   if (array[y]>array[y+1]) {
    int temp = array[y+1];
    array[y+1] = array[y];
    array[y] = temp;
   }
  }
 }
}
```



Human proof

program: 7 LOC
proof: by hand

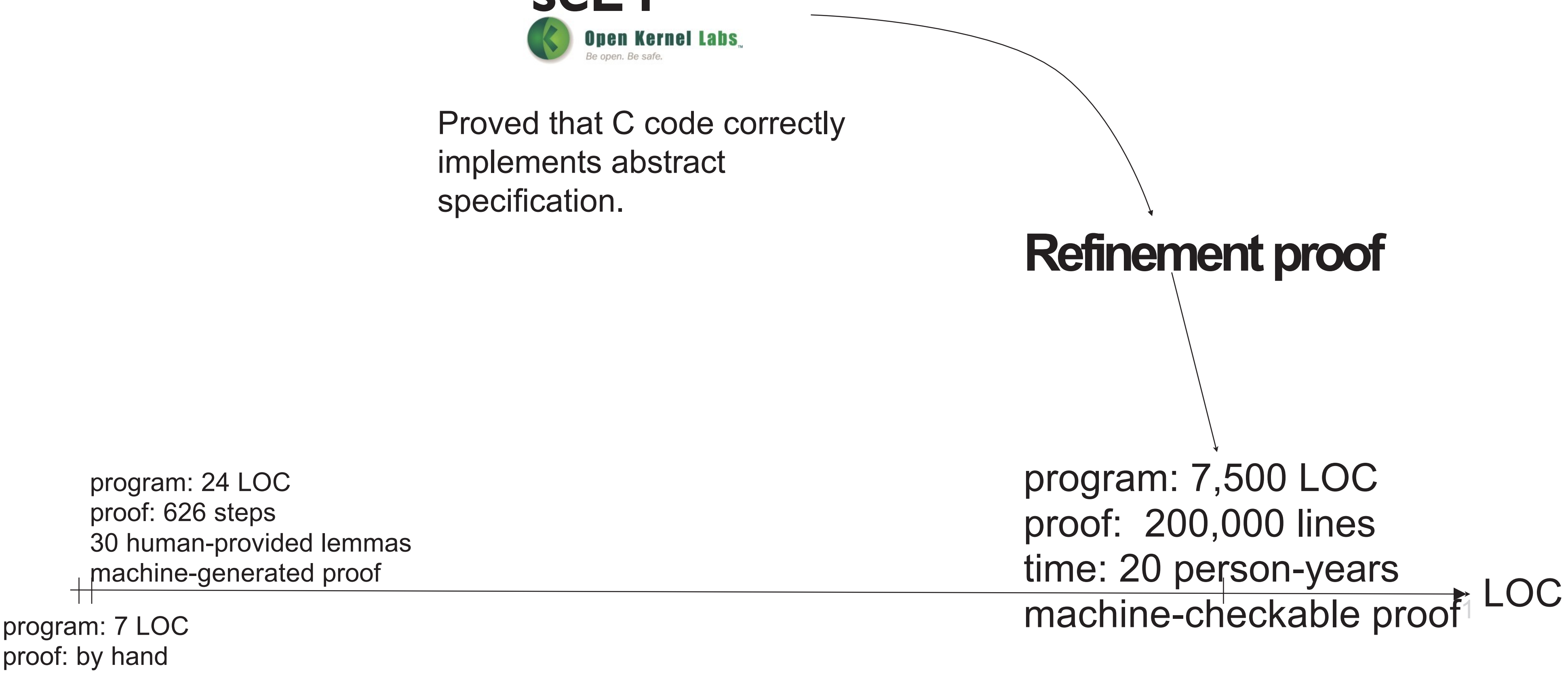
```
void heapSort( int numbers[], int arraySize ) {
   int i, temp;
   for (i = (arraySize / 2); i >= 0; i--) {
    siftDown(numbers, i, array_size - 1);
   }
   for (i = arraySize-1; i >= 1; i--) {
     temp = numbers[0];
     numbers[0] = numbers[i];
     numbers[i] = temp;
     siftDown(numbers, 0, i-1);
   }
}
void siftDown( int numbers[], int root, int bottom ) {
  int maxChild = root * 2 + 1;
  if (maxChild < bottom) {
     int otherChild = maxChild + 1;
     if (numbers[otherChild] > numbers[maxChild]) maxChild = otherChild;
  } else {
     if (maxChild > bottom) return;
  }
  if (numbers[root] >= numbers[maxChild]) return;
  int temp = numbers[root];
  numbers[root] = numbers[maxChild];
  numbers[maxChild] = temp;
  siftDown(numbers, maxChild, bottom);
}
```

Theorem prover

program: 24 LOC
proof: 626 steps
30 human-provided lemmas
machine-generated proof[1]

LOC

[1] J-C Filliatre and N. Magaud, *Certification of Sorting Algorithms in the Coq System*, in “Theorem Proving in Higher Order Logics: Emerging Trends” (1999)



seL4

Open Kernel Labs

Be open. Be safe.

Proved that C code correctly implements abstract specification.

Refinement proof

program: 7,500 LOC
proof: 200,000 lines
time: 20 person-years
machine-checkable proof[1]

program: 24 LOC
proof: 626 steps
30 human-provided lemmas
machine-generated proof

program: 7 LOC
proof: by hand

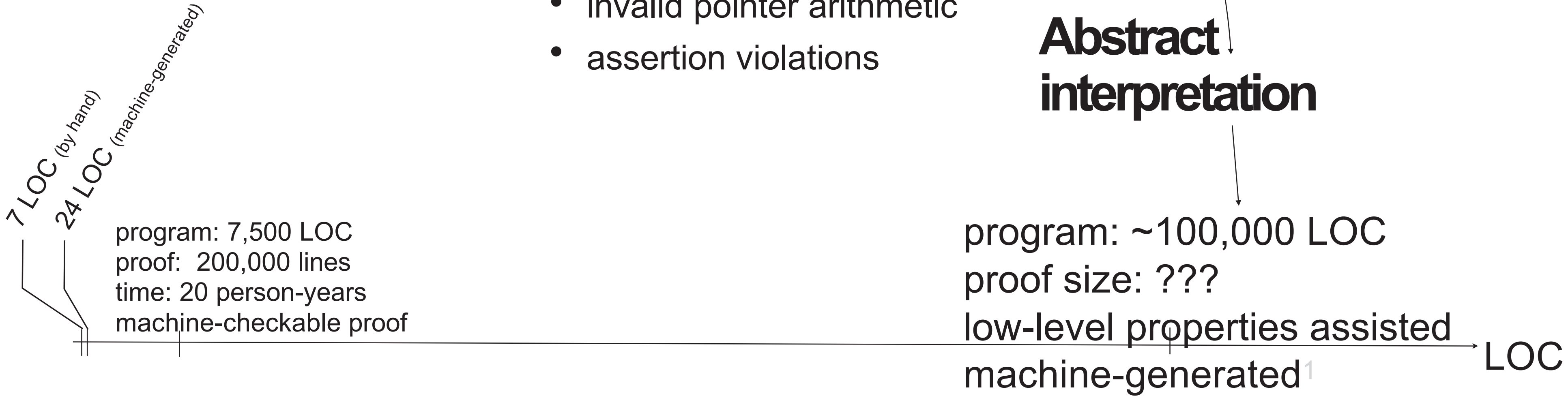
[1] G. Klein et al., *seL4: formal verification of an OS kernel*, in Proc. Symp. on Operating Sys. Princ., (2009)

AIRBUS
AN EADS COMPANY

Parts of code free of...

- arithmetic overflows
- out-of-bounds array accesses
- invalid pointer arithmetic
- assertion violations

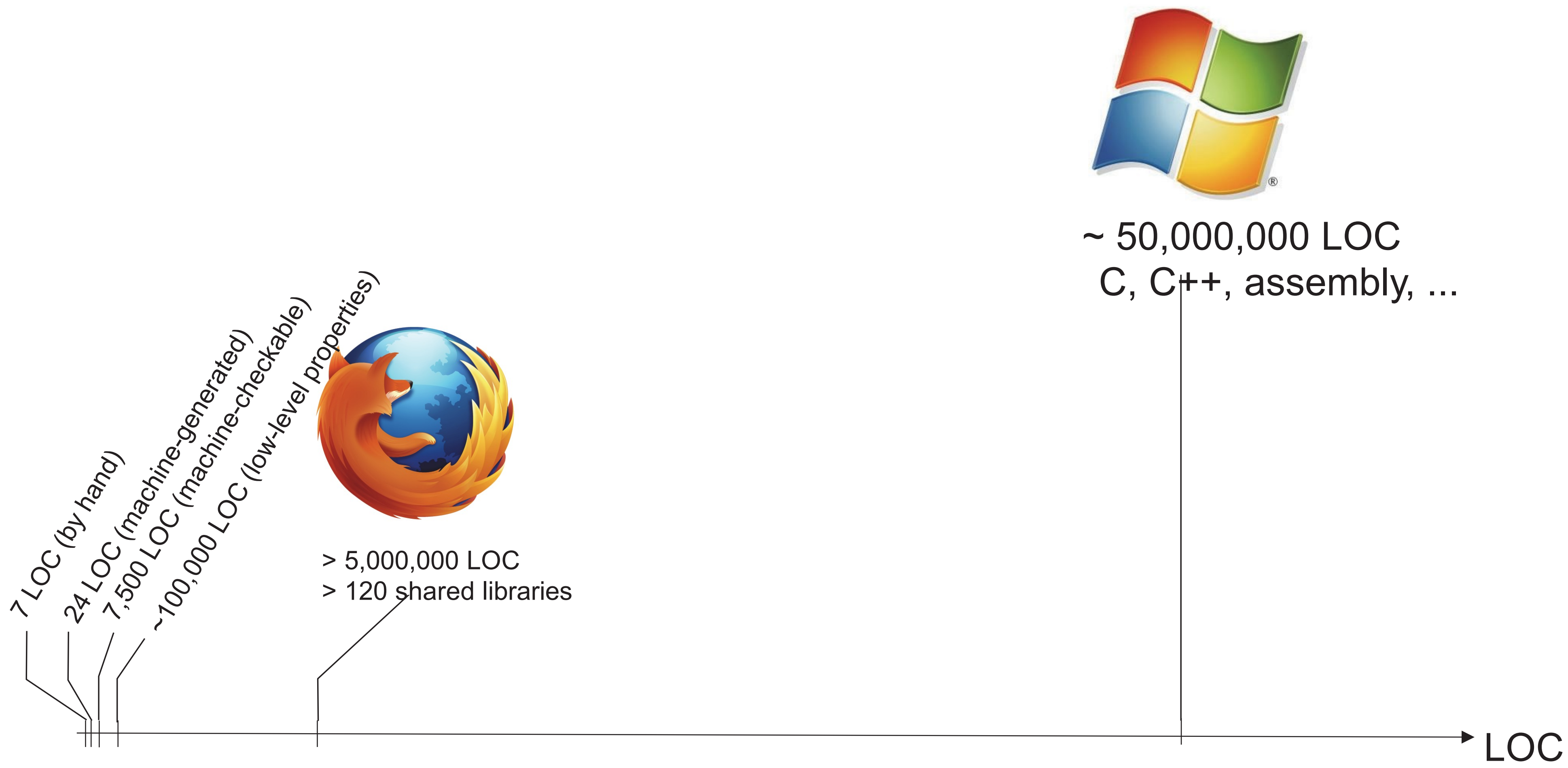
Abstract interpretation



program: 7,500 LOC
proof: 200,000 lines
time: 20 person-years
machine-checkable proof

program: ~100,000 LOC
proof size: ???
low-level properties assisted
machine-generated[1]

[1] J, Souyris and D. Delmas, *Experimental Assessment of Astrée on Safety-Critical Avionics Software*. Proc. Int. Conf. on Computer Safety, Reliability, and Security (2007)



7 LOC (by hand)

24 LOC (machine-generated)

7,500 LOC (machine-checkable)

~100,000 LOC (low-level properties)

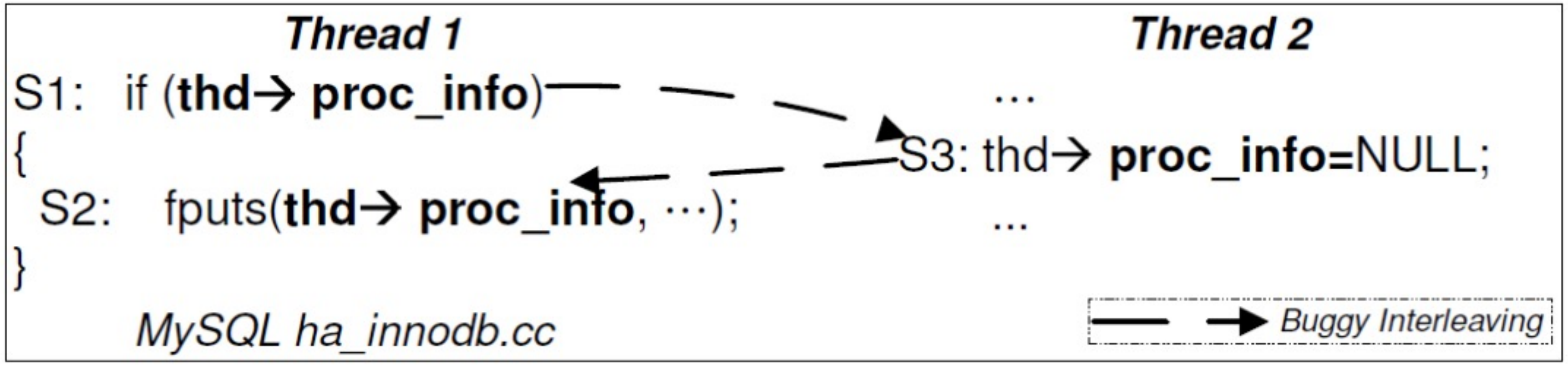
> 5,000,000 LOC
> 120 shared libraries

~ 50,000,000 LOC
C, C++, assembly, ...

LOC

Complexity Challenges

- Concurrency
  - Multiple threads accessing shared data
  - Atomicity bugs, ordering bugs, data race, deadlock bugs



```
Thread 1                                   Thread 2
S1:  if (thd→ proc_info)                   ...
{                                          S3: thd→ proc_info=NULL;
  S2:   fputs(thd→ proc_info, ···);        ...
}
```

MySQL ha_innodb.cc

Figure 1. An atomicity violation bug from MySQL.

The Challenges of Building Software

- #1 Challenge in Software Engineering: Complexity

- Software complexity grows fast
  - At least exponential in the number of branches
- Cannot rely only on testing and proofs
  - We need principles, discipline, and processes

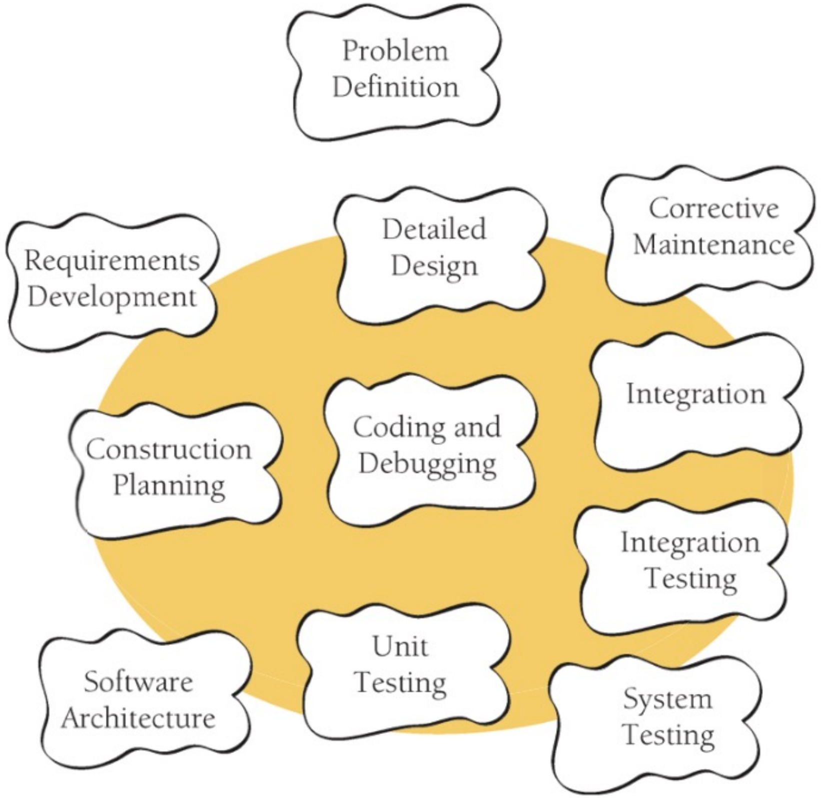
- Key techniques
  - Modularity, object-oriented programming, etc.

Dealing with Complexity

- Preventing and Finding Errors Early

SW Engineering Teaches You How To…

- Ensure requirements are well specified
- Design and write classes and routines
- Create and name variables appropriately
- Select and organize control structures
- Unit test, integration test, and debug
- Review designs and code
- Polish code through format and comments
- Integrate third-party components
- Make code run faster and use fewer resources

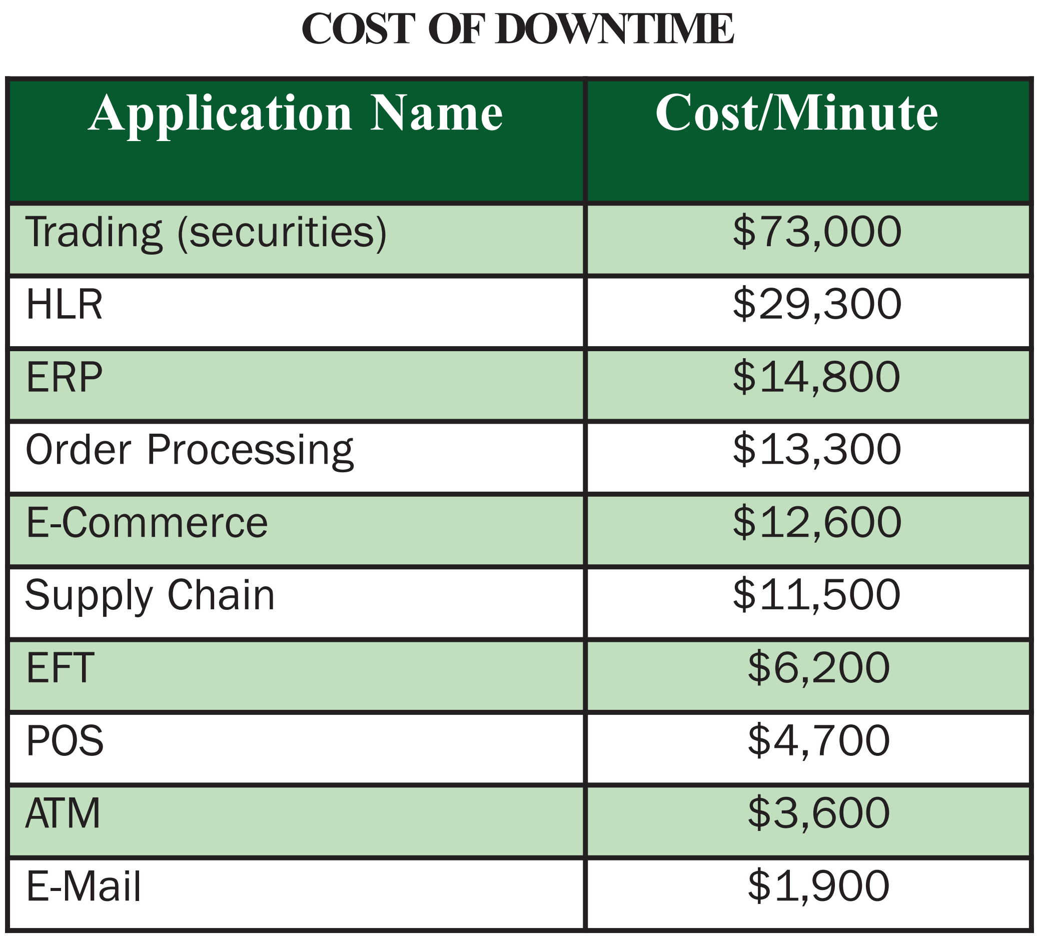


Average vs. Best Programmer

- Initial coding time
  - 20x ratio
- Debugging time
  - 20x ratio
- Program execution speed
  - 10x ratio
- 80% of contribution from 20% of programmers
  - At the end of this course, I hope you will be in the top-20% ☺

Find Errors Early

- Debugging and associated rework
  - 50% of the average development cycle
- Long-lived errors are expensive
  - Finding errors early reduces time and cost



COST OF DOWNTIME

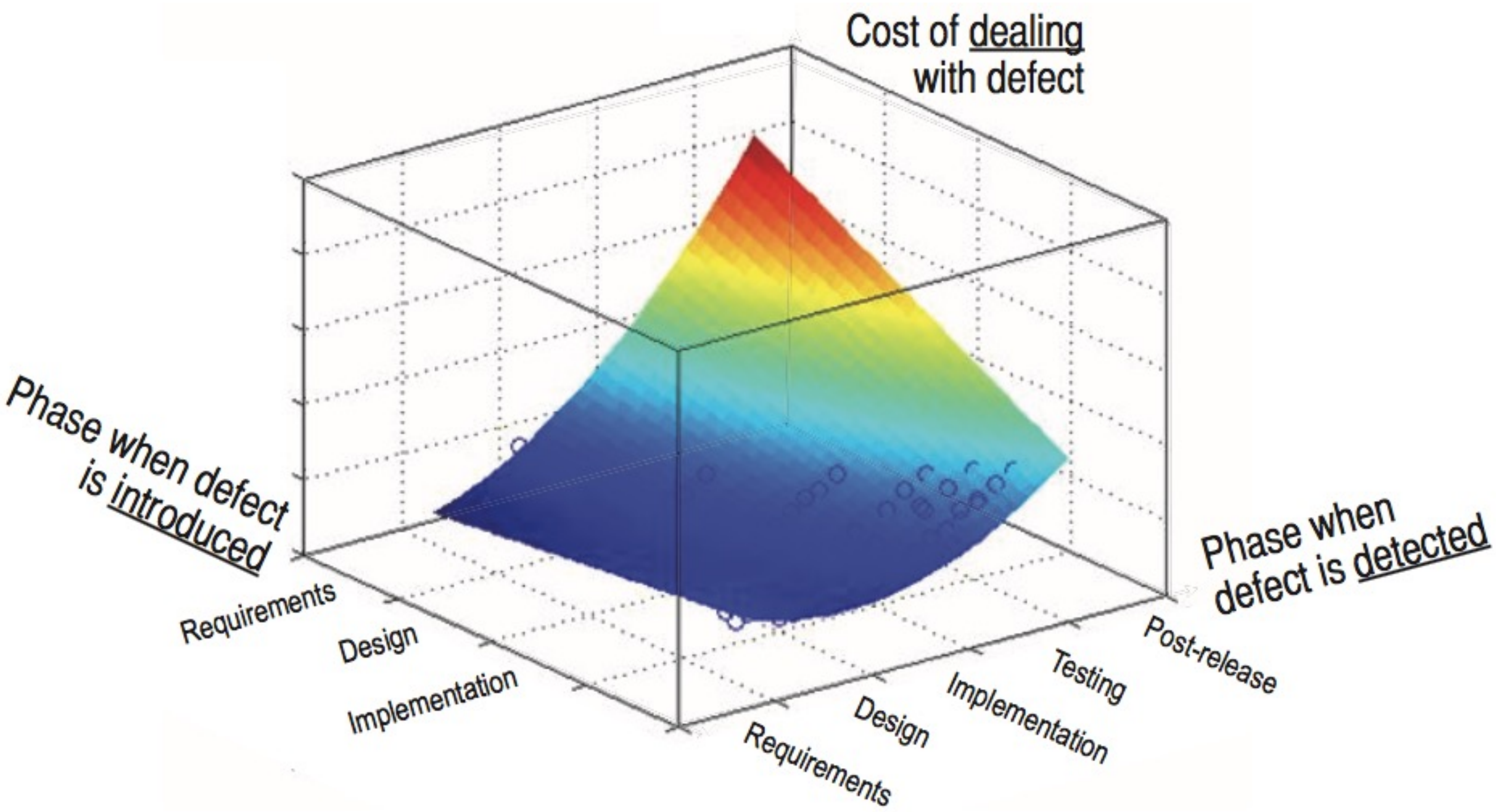
Application Name	Cost/Minute
Trading (securities)	$73,000
HLR	$29,300
ERP	$14,800
Order Processing	$13,300
E-Commerce	$12,600
Supply Chain	$11,500
EFT	$6,200
POS	$4,700
ATM	$3,600
E-Mail	$1,900

From Trends in IT Value, The Standish Group International, 2008

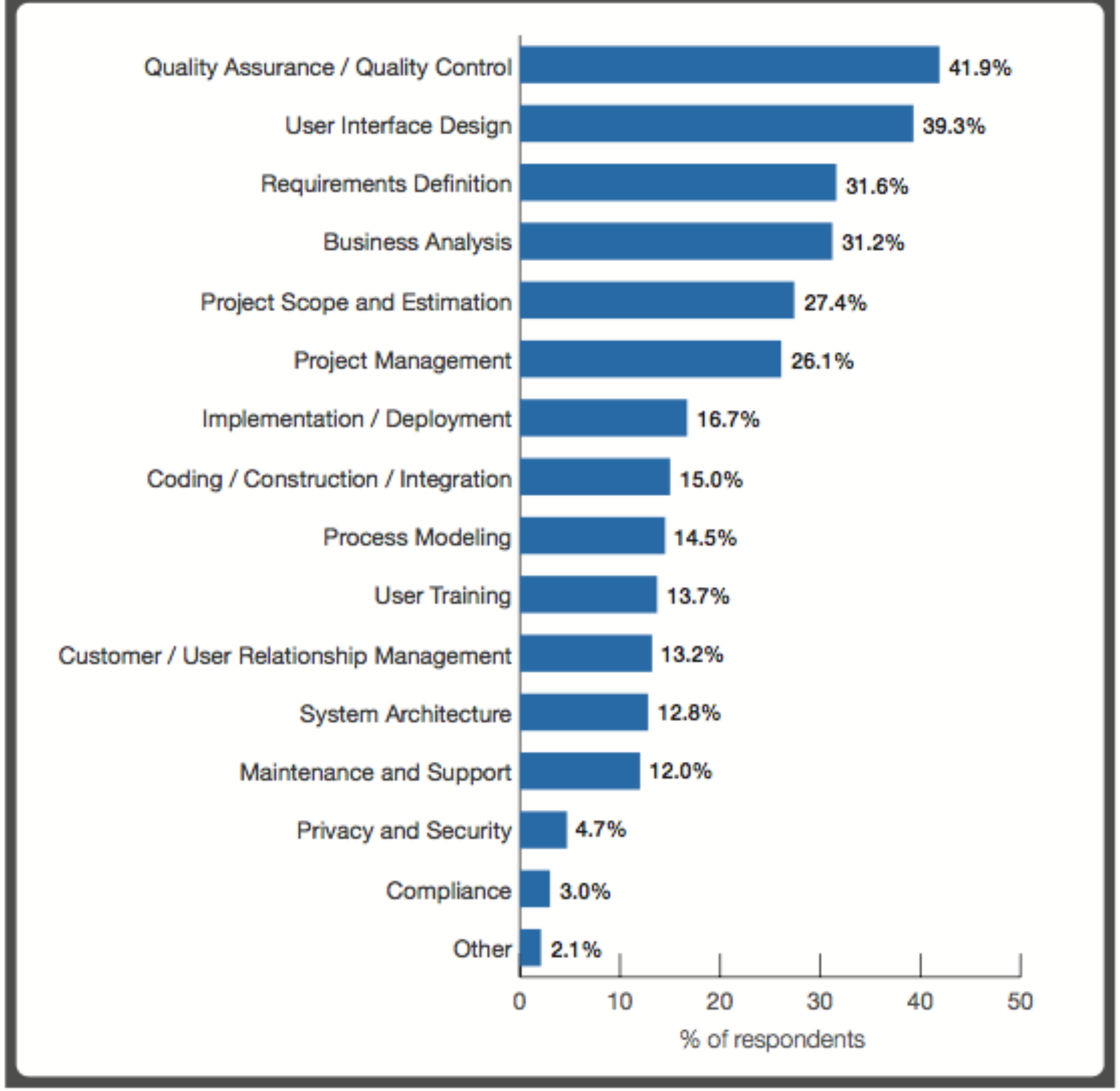
Amazon.com suffered a glitch today leaving its website inaccessible for approximately 13 minutes.

Amazon reported revenues of $107 billion in 2015, which comes out to $203,577 every minute in today's numbers, or a **$2,646,501** price tag for the 13 minute episode of downtime.

Long-Lived Errors Are Expensive



In what Software Developmental Area(s) does your organization need the greatest improvement?



Mitigate Risk

- Lots of testing
  – Not enough by itself
- Verification
  – Are you building it right?
- Validation
  – Are you building the right thing?

How Does Software Engineering Do This?

- Test-driven development (TDD)
  - Perform verification before coding
- Behavior-driven development (BDD)
  - Perform validation before coding
- Agile development processes
  - Shorten the time between coding and error discovery
- Powerful tools
- Discipline and rigor

Example: Design Patterns

- Design patterns: toolbox of reusable solutions on well-known software problems
  - Later in the semester we will talk about 23 design patterns

- Decorator
  - Problem: augment functionality without changing the code that uses the class and without using inheritance
  - Solution template: keep interface the same, dynamically add/override behaviors underneath same interface

```
start_time = int(round(time.time() * 100s0))
employees = Employee.get_all_employee_details()
time_diff = current_milli_time() — start_time
print_time({'FETCH_TIME': time_diff})
```

```
def timeit(method):
    def timed(*args, **kw):
        ts = time.time()
        result = method(*args, **kw)
        te = time.time()

        print '%r %2.2f ms' % \
                (method.__name__, (te - ts) * 1000)
        return result

    return timed
```

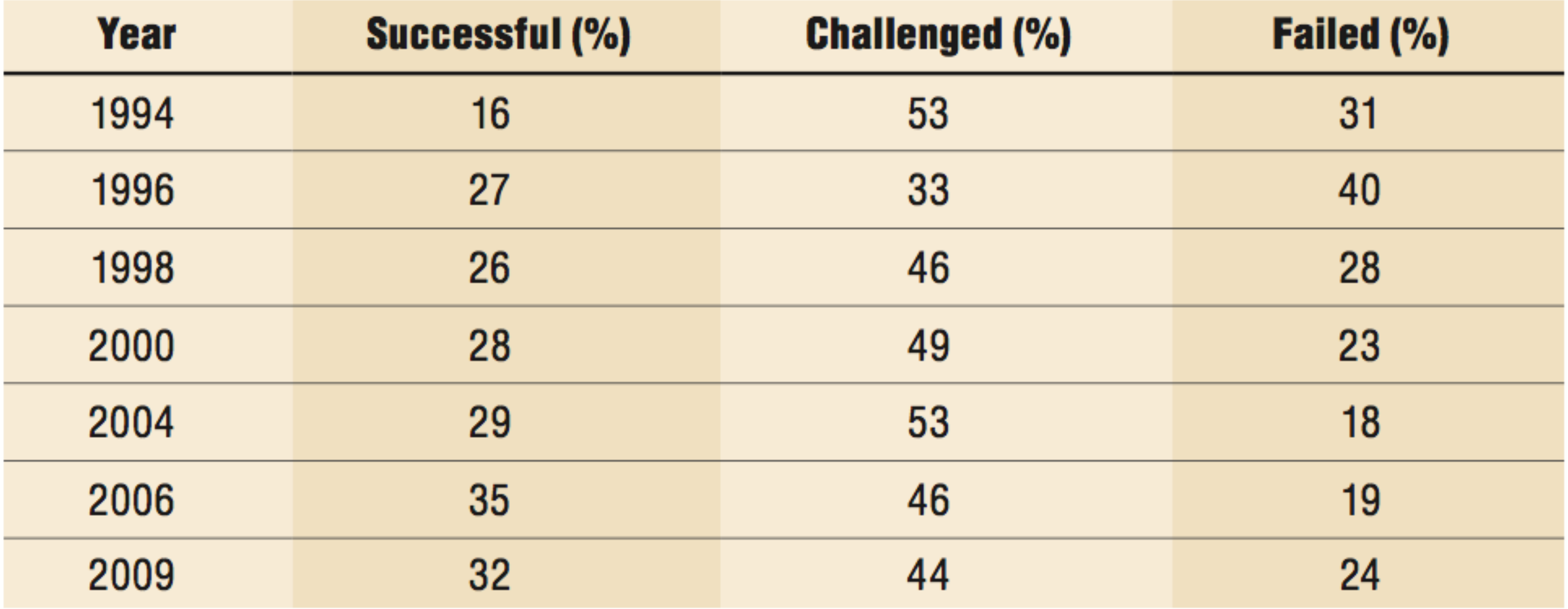
```
@timeit
def get_all_employee_details(**kwargs):
    print 'employee details'


employees = Employee.get_all_employee_details()
```

The Management Challenge

Do We Need Management?

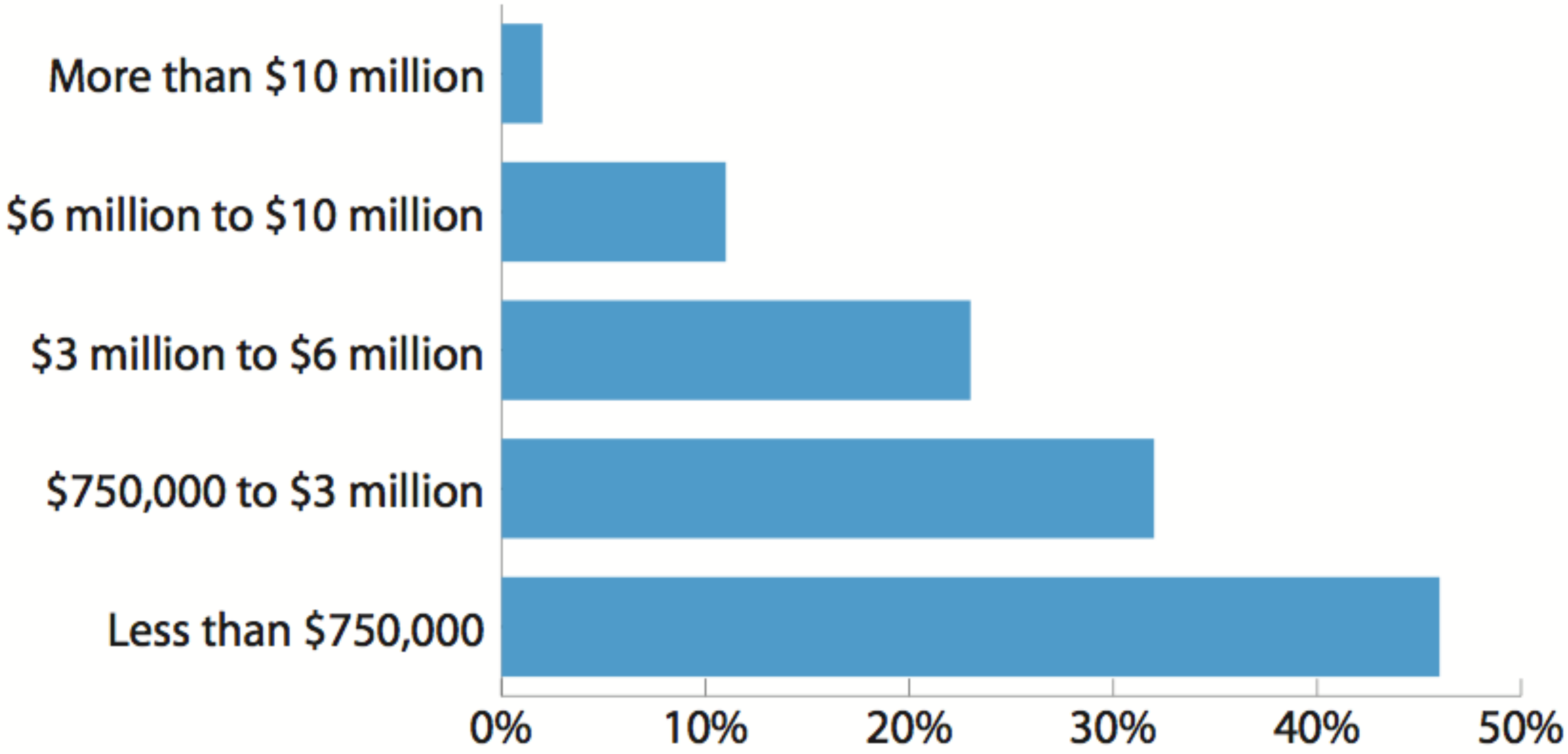
- Large teams need managers
  - *A manager coordinates the energies of the team*
- Managing a software project is difficult
  - *Partly because software engineering is a young field*
- Large software projects
  - *Are 1 year late (on average)*
  - *Are 100% over budget (on average)*

Success Rate of Software Projects



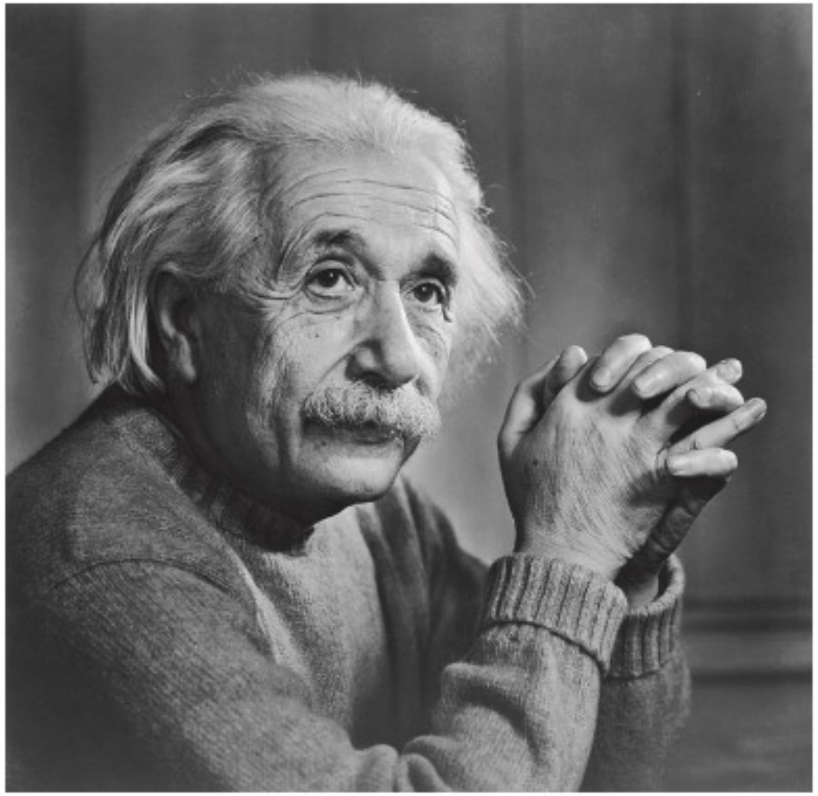
Year	Successful (%)	Challenged (%)	Failed (%)
1994	16	53	31
1996	27	33	40
1998	26	46	28
2000	28	49	23
2004	29	53	18
2006	35	46	19
2009	32	44	24

Project Success Rate As A Function of Budget



Dealing with Management Challenges
- Managing Your Software Project

Understand The Problem



> "If I had one hour to save the world, I would spend 55 minutes defining the problem and 5 minutes finding the solution."
>
> — Albert Einstein

- Read carefully the statement
  - *Validation: make sure you solve the right problem*
- Do this early, to have time to "digest"
  - *If in doubt, ask teammates, discussion board, or staff*
- Break the problem down
  - *Smaller sub-problems are less intimidating*
  - *It makes it easier to get one step closer to being done*
- Separate thinking from execution
  - *This helps you think better and execute faster*

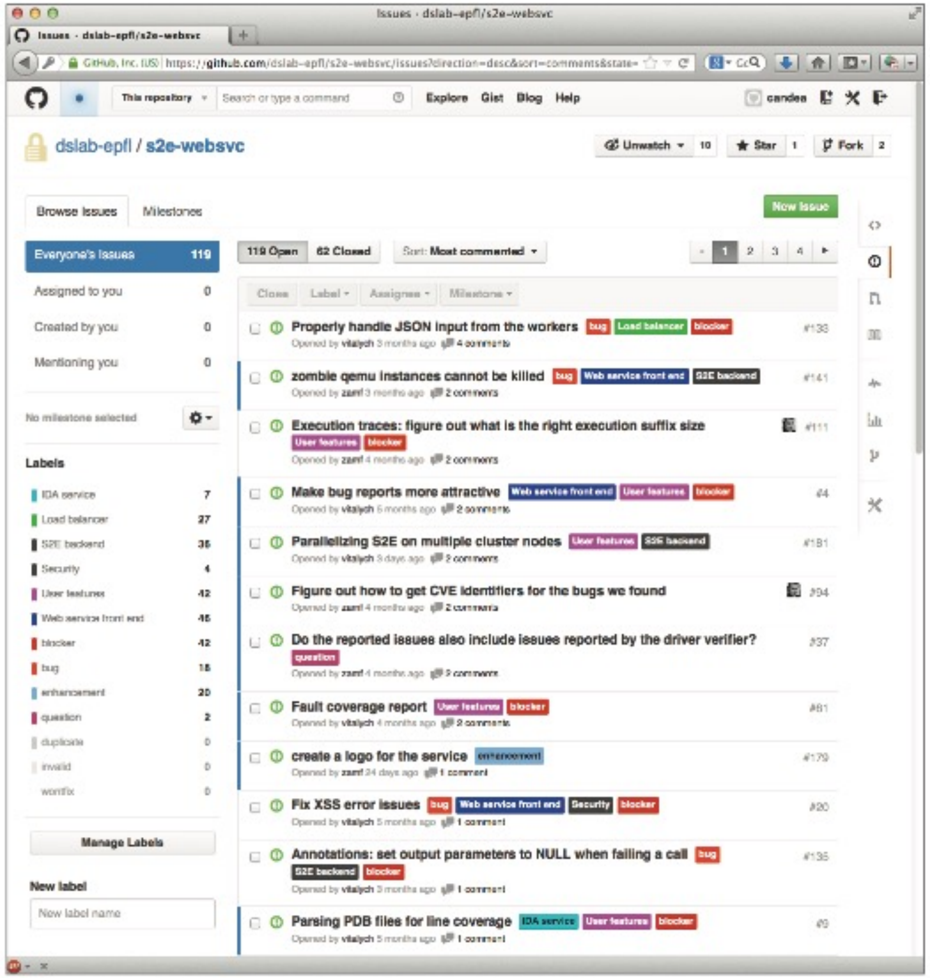
Time Management: The Key to Success

A dream is ... just a dream.
A goal is a dream with a plan and a deadline.

— paraphrased from Antoine de Saint-Exupéry

- Make a plan with deadlines
  - *Match deadlines to sub-problems*
  - *Space them out, don't let them bunch up at the end*
    - Code written at the last minute has higher chances to be buggy
  - *Make sure your entire team buys the plan*
- Expect the unexpected → margin of error
  - *Budget time for the "things that should not happen"*
  - *Expect nasty bugs, team miscommunication, etc.*
- Track time-to-deliver, learn from mistakes
  - *After every deliverable, do a "post-mortem analysis"*
  - *Deadlines + post-mortem analysis forces team to get to know itself and learn how to become better*

Team Coordination



Everyone's Issues	119
Assigned to you	0
Created by you	0
Mentioning you	0

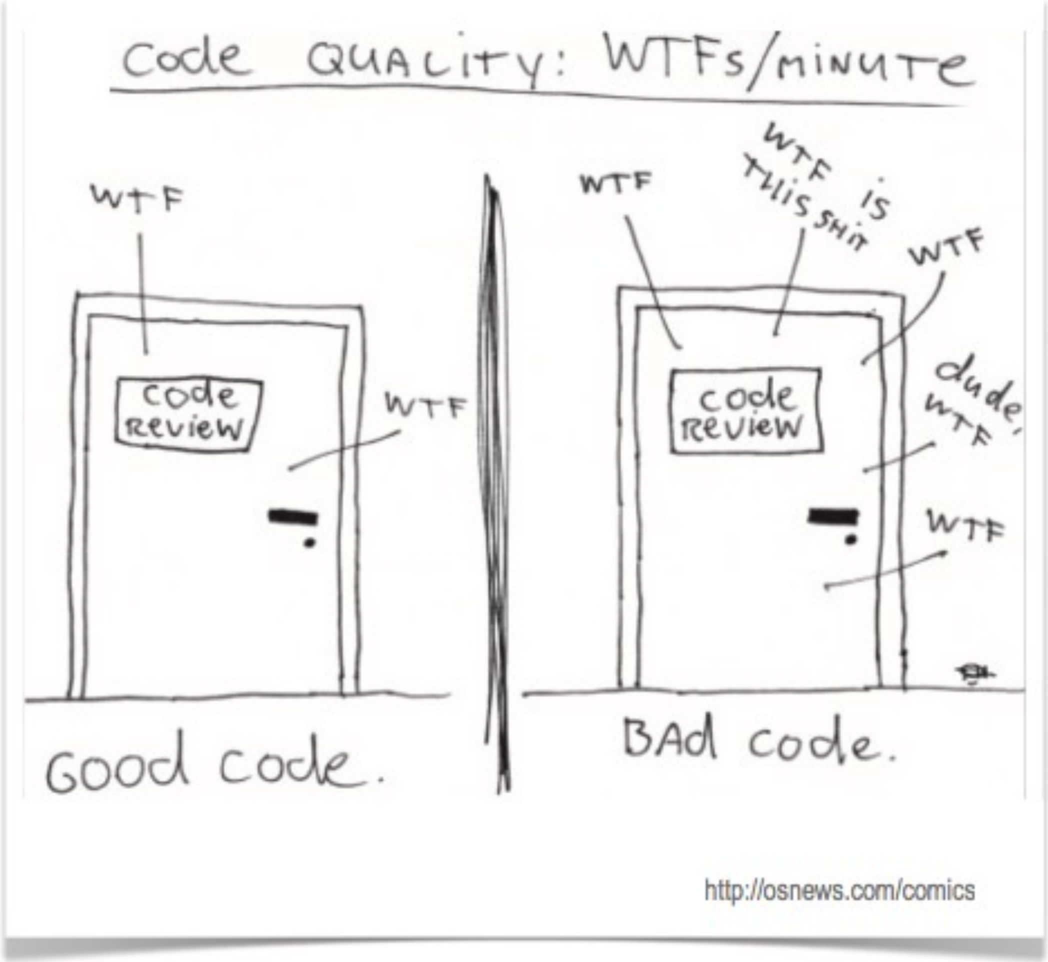
Labels

IDA service	7
Load balancer	27
S2E backend	35
Security	4
User features	42
Web service front end	45
blocker	42
bug	18
enhancement	20
question	2
duplicate	0
invalid	0
wontfix	0

- Properly handle JSON input from the workers `bug` `Load balancer` `blocker` #133
  Opened by vitalych 3 months ago 4 comments
- zombie qemu instances cannot be killed `bug` `Web service front end` `S2E backend` #141
  Opened by zamf 3 months ago 2 comments
- Execution traces: figure out what is the right execution suffix size `User features` `blocker` #111
  Opened by zamf 4 months ago 2 comments
- Make bug reports more attractive `Web service front end` `User features` `blocker` #4
  Opened by vitalych 6 months ago 2 comments
- Parallelizing S2E on multiple cluster nodes `User features` `S2E backend` #181
  Opened by vitalych 3 days ago 2 comments
- Figure out how to get CVE identifiers for the bugs we found #94
  Opened by zamf 4 months ago 2 comments
- Do the reported issues also include issues reported by the driver verifier? `question` #37
  Opened by zamf 4 months ago 2 comments
- Fault coverage report `User features` `blocker` #81
  Opened by vitalych 4 months ago 2 comments
- create a logo for the service `enhancement` #179
  Opened by zamf 24 days ago 1 comment
- Fix XSS error issues `bug` `Web service front end` `Security` `blocker` #20
  Opened by vitalych 5 months ago 1 comment
- Annotations: set output parameters to NULL when failing a call `bug` `S2E backend` `blocker` #135
  Opened by vitalych 3 months ago 1 comment
- Parsing PDB files for line coverage `IDA service` `User features` `blocker` #9
  Opened by vitalych 5 months ago 1 comment

- Designate a (part-time) manager
  - *Rotate management role among team members*
- Use Github’s issue tracking system
  - *Split problem into individual / group tasks*
  - *Assign tasks to people or groups of people*
    - Make sure everyone agrees with the assignments
- How to deal with slackers?
  - *Set per-task deadlines, react early when they’re missed*
  - *keep emotion out of it, present concrete evidence*
  - *Assign tasks/deadlines according to abilities and goals*
- You are an inter-dependent group
  - *Be proactive and solve team problems promptly*

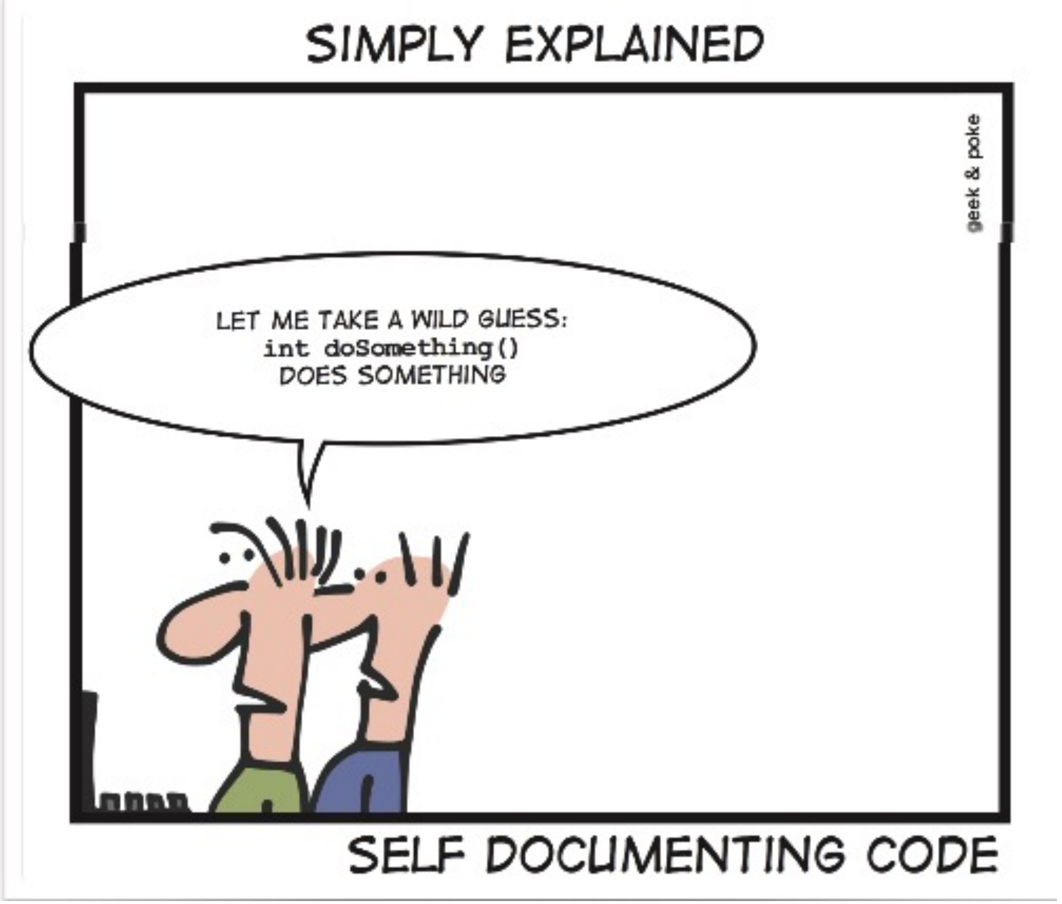
Software = Team Effort



http://osnews.com/comics

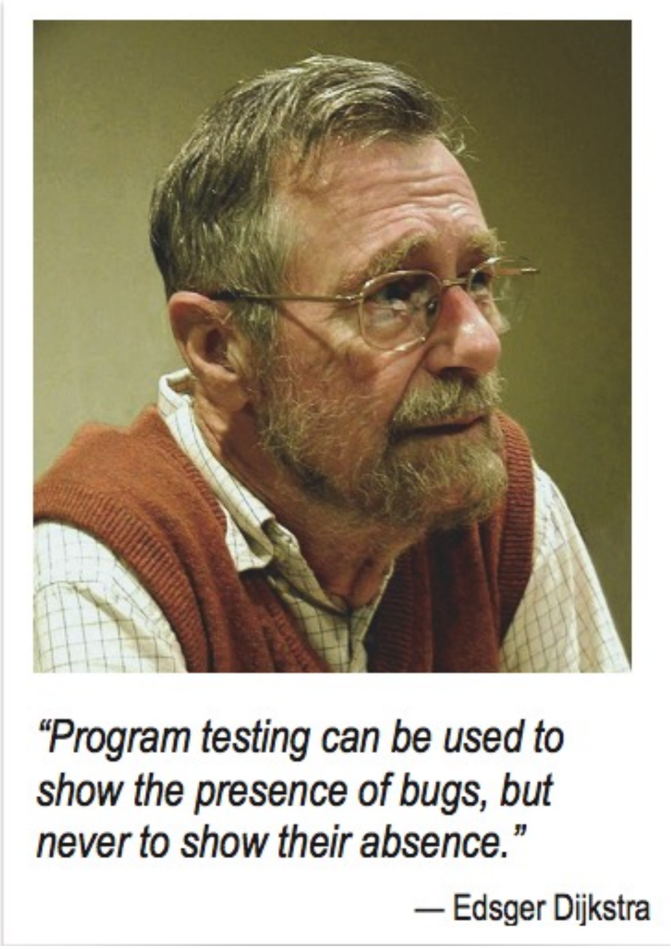
- Code reviewed by peers
  - *>= 2 people on each piece of code*
  - *Safety net, in case team member quits, disappears, etc.*
  - *Improves code quality: peer pressure + verification*
- Put code in the internally "public" eye
  - *Make all test runs and test results public*
  - *Use good code as an example for others*
- Get involved
  - *Read others' code; if you don't understand it, ask !*
  - *Write tests that break others' code*
  - *In our course, <u>entire</u> team must know all code inside-out*

Think of Code As Write-Once / Read-Many



- Focus on code readability
  - *This means clear code, not many comments (distracting)*
- Adopt a specific coding style for the team
  - *Enable checkstyle; heed all warnings*
  - *Retrofitting code to another coding style is very hard*
- Self-documenting code repository
  - *Strive to make each code **push** a meaningful unit*
  - *Use informative commit messages*
    - Bad: “code works now”
    - Good: “Fixed crash at login (closes issue #39)”
    - See https://github.com/WebKit/webkit/commits/master for a good example
- Code in repository should always compile

Write Tests

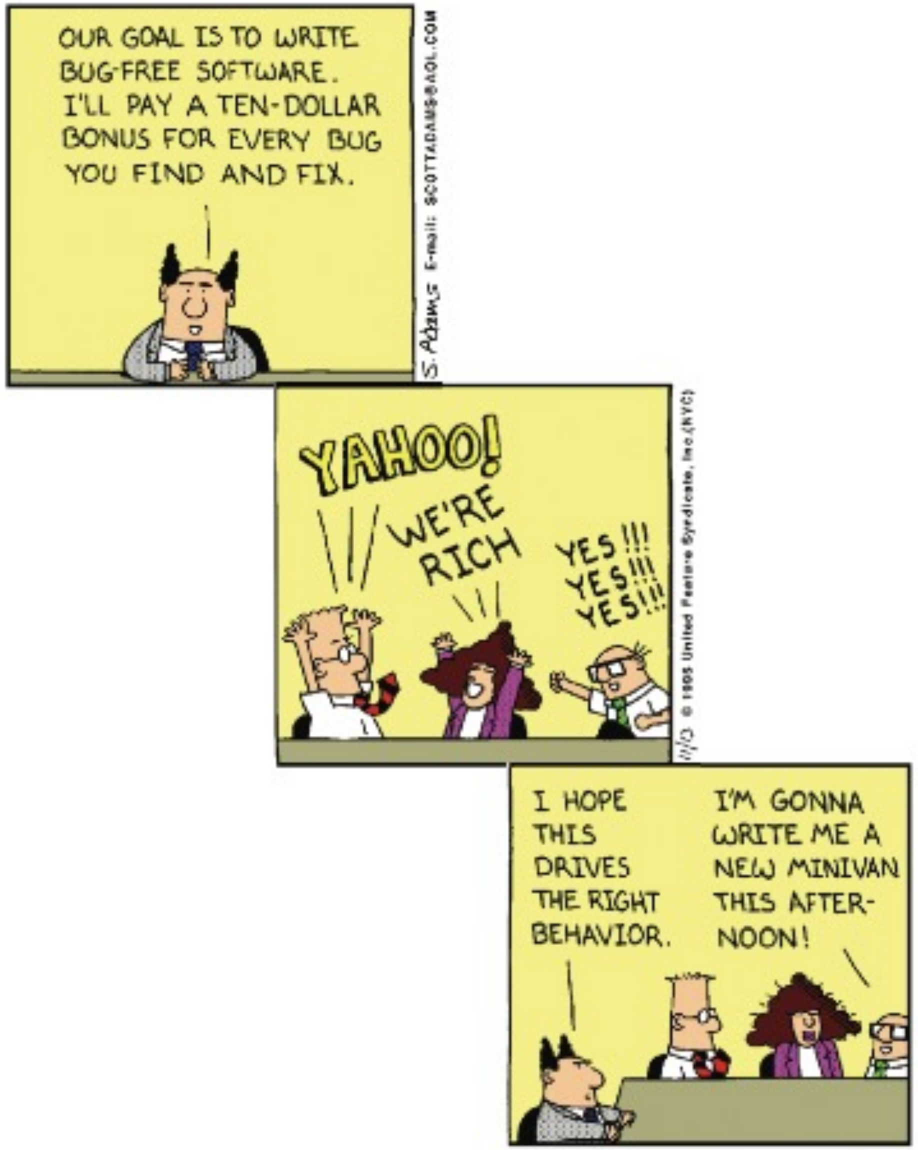


"Program testing can be used to show the presence of bugs, but never to show their absence."

— Edsger Dijkstra

- Goal of testing is to find bugs in the code
  - *Write tests before or during code writing*
  - *For each bug you fix, write a corresponding test case*
  - *Try to catch bugs that may be introduced in the future*
- Tests are part of your software
  - *Version-control them in the same repository*
  - *Be as disciplined about tests as you are about code*
  - *Code in repository should always pass all tests*
- Tests only verify your code
  - *They don't make your code correct*
  - *cannot test all execution paths*

Use Quantitative Metrics


OUR GOAL IS TO WRITE BUG-FREE SOFTWARE. I'LL PAY A TEN-DOLLAR BONUS FOR EVERY BUG YOU FIND AND FIX.
S. Adams E-mail: SCOTTADAMS@AOL.COM
YAHOO!
WE'RE RICH
YES!!! YES!!! YES!!!
© 1995 United Feature Syndicate, Inc.(NYC)
I HOPE THIS DRIVES THE RIGHT BEHAVIOR.
I'M GONNA WRITE ME A NEW MINIVAN THIS AFTERNOON!

- Measurement = #1 rule in engineering
  - *Quantify progress and improvement*
  - *Motivates productivity (but measure the right thing!)*
- Examples of common questions + metrics
  - *Code size*
    - LOCs, number of classes/methods, data declarations
  - *Test effectiveness / efficiency*
    - statement/branch/path coverage, # of defects, defect density
  - *Debugging effectiveness / efficiency*
    - time to find bug, time to fix, # of attempts, # of new bugs
  - *Productivity*
    - person-hours, LOC changed, $/LOC, $/defect, $/project

What If You Screw Up ?

© Tatiana Adamenko/Caters

- Avoid foolish optimism
  - *"requirements took a little longer than expected... but now we're on track... we'll make up time"*
  - *reality: projects fall further behind, rarely make up time*
- Add more people ?
  - *only works when tasks are cleanly partitionable*
- Best choice: reduce scope of project
  - *less functionality → less design/code/debug/test/doc*
  - *partition into must-have/nice-to-have/optional*
  - *move from "chaos" to "project is under control"*
- If you're stuck / stumped...
  - *use web search + brainstorm with teammates, TAs, etc.*

Work Smarter

DONE IS BETTER THAN PERFECT
WHAT WOULD YOU DO IF YOU WEREN'T AFRAID?
MOVE FAST AND BREAK THINGS

(Physical) wall at Facebook headquarters

- Be **agile**
  - *prototype ideas in code, then throw the code away*
    - the experience you gather will help you do it better
  - *fix inefficiencies on the go, don't seek early perfection*
- Be **smart**
  - *factor out things that you repeat >= 3 times*
  - *do not "blindly" copy-paste without understanding*
- **Love what you do**
  - *invest time in learning the* ***full power of your tools***
  - *be passionate about building things, learn new stuff*

Managing Your Software Project

- Understand the problem you are solving
- Make a plan with clear deadlines
  - Use your success rate as feedback for next plan
- Learn and use powerful tools
- Care for your team
  - Team problems are all team members' responsibility
- Write code for others to understand
- Employ quantitative metrics
- Become good at recovering from failures
  - Everybody makes mistakes;
    successful people are the ones who recover fast and move on

Challenges to Make Software Good, Safe, and Reliable