Software Quality Assurance and Testing (SQAT)

Chapter 1: Introduction to Software QA and Testing

dr. Joost Schalken-Pinkster
Windesheim University of Applied Science
The Netherlands

joost@schalken.me

The contents of these course slides is (in great part) based on:
Neil Taylor (2019) *Course presentations for Software Quality Assurance and Testing (SQAT)*, Aberystwyth University.

Learning Objectives

- Discuss the main purpose of quality assurance and testing
- Discuss the terms testing and debugging
- Talk about some different problems you have found with software
- Describe the role of testing at Microsoft

The need for testing

My first ever program…

```
10 Print "HELLO"
20 GOTO 10
RUN
```

What does the program do?
How could we test the program?

How do we test that a camera captures the correct image?

What is a software bug?

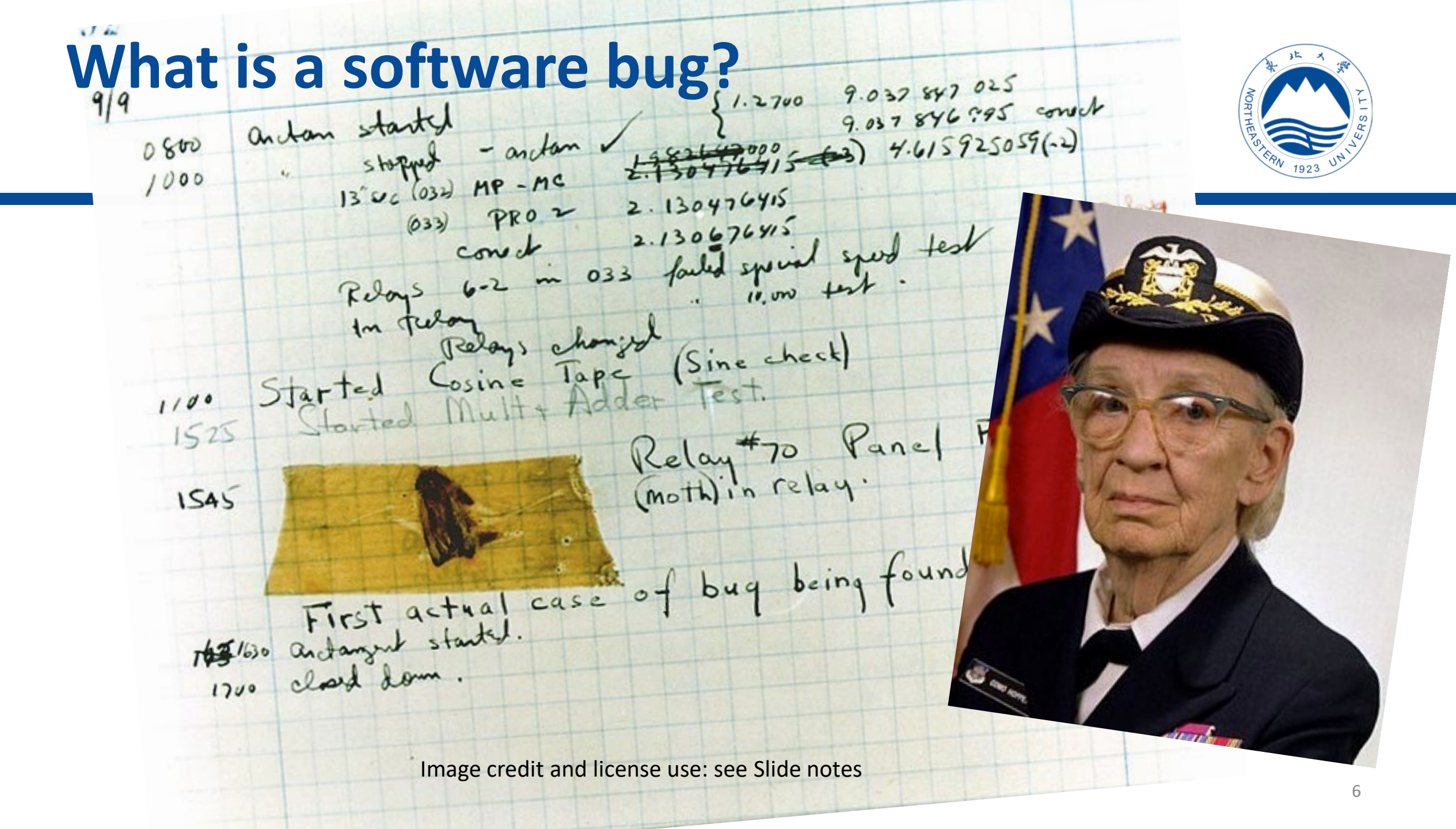


Image credit and license use: see Slide notes

We can all create ‘bugs’

“If debugging is the process of removing bugs, then programming must be the process of putting them in.”
E. J. Dijkstra*

Image credit and license use: see Slide notes

* The quote is sometimes accredited to Dijkstra, but I haven’t found the original source. Whether or not he said it, it has an element of truth.

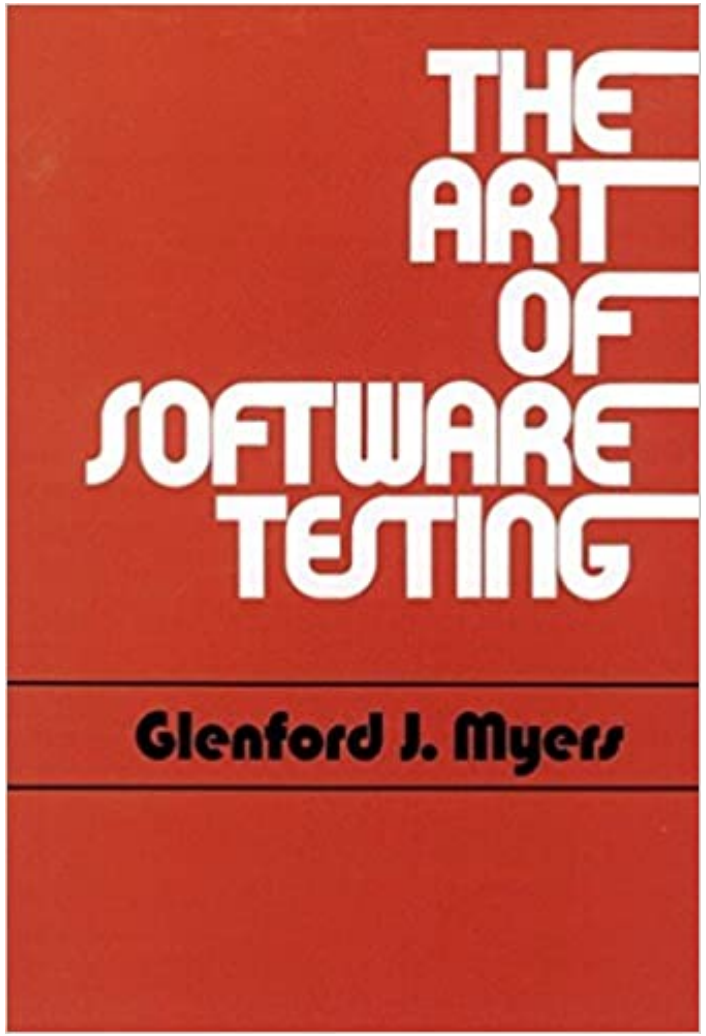
What is the purpose for testing?

A. There is no difference between testing and debugging
B. The purpose of testing is to show that the software works
C. The purpose of testing is to show that the software doesn't work
D. The purpose of testing is not to prove anything specific, but to reduce the risk of using the software
E. Testing is a mental discipline that helps all IT professionals develop higher quality software

The purpose (goal) of testing

"If our goal is to demonstrate that a program has no errors, then we will be steered subconsciously toward this goal; that is, we tend to select test data that have a low probability of causing the program to fail. On the other hand, if our goal is to demonstrate that a program has errors, our test data will have a higher probability of finding errors. The latter approach will add more value to the program than the former."

Myers, Badgett, Sandler, "The Art of Software Testing", 2012, John Wiley & Sons, Inc., New Jersey, USA.



Your Experience

What is the biggest piece of software you've ever written?

- How did you test it?
- Did it still have bugs?

What kind of bugs in software annoy you the most?

What is the worst bug you have ever found?

Exercise

On your own:

think of some different examples of the use of software and the possible risks if the software fails

With the class:

we will then discuss your examples

Microsoft’s Experience

Microsoft Office Windows Surface Xbox Deals Support All Microsoft Sign in

Software	PCs & Devices	Entertainment	Business	Developer & IT	Other
Windows apps	PCs & tablets	Xbox games	Microsoft Azure	.NET	Microsoft Store
OneDrive	Accessories	PC games	Microsoft Dynamics 365	Visual Studio	Microsoft Rewards
Outlook	VR & mixed reality	Windows digital games	Microsoft 365	Windows Dev Center	Free downloads & security
Skype	Microsoft HoloLens	Movies & TV	Windows Server	Docs	Education
OneNote			Enterprise solutions		Gift cards
			Data platform		

Microsoft's Experience

100 major product families (possibly more)

80 languages and dialects (possibly more)

4 Engineering Groups

- Cloud and AI
- Experiences and Devices
- AI and Research
- Core Services and Engineering

In 2008, there were a reported 35,000 software engineers (how many do you think are test engineers?)

- More recent company information suggests 67,000 engineers, but no information on how many might be involved in testing

Note – Looking at Facts About Microsoft, we might estimate that the number of engineers at approximately 67,000. See https://news.microsoft.com/facts-about-microsoft/#EmploymentInfo

Microsoft's Experience

From previous slide:

- In 2008, there were a reported 35,000 engineers
- Of these, 9,000 test engineers - about 25% of the total number of engineers
- 100,000 computers running tests

Evolving Test Practices at Microsoft

11/14/2017 • 9 minutes to read •

By Munil Shah

Testing at Microsoft circa 1990s

For a long period of time Microsoft shipped software with a basic setup for the engineering team. Every product team had three distinct disciplines. *PMs* owned understanding customer requirements and writing feature specs, *Devs* owned designing and coding the features. *Testers* owned testing the features. Teams had a ratio of 1 dev to 1 test to 0.5 PM, with some variance among teams.

The test discipline had two separate roles: Software Design Engineer Test (SDETs) who developed the automation, the test infrastructure etc., and then Software Test Engineer (STEs) who ran the automation or who ran manual tests. We hired SDETs from the same places we hired SDEs. They had very similar qualifications and skills when they got hired. They went to the same colleges. If we hired from industry, we'd pretty much hire developers and then convert them into SDETs.

How Did It Work in Practice?

It worked reasonably well back in the days. We achieved commercial success with big products like Windows and Office. The test discipline provided product signoff based on very formal quality measurements. That gave us good confidence in declaring a product

In this article

Specialization

A core principle of the Combined Engineering is elimination of tasks passing from one team to another. Handoff introduce delays and dilutes accountability. There are no specialized central teams that do certain tasks. Each feature team has end to end accountability of delivering features.

At the same time, we understood the importance of specialized skills. Performance testing can be a specialized skill. Security testing can be a specialized skill. How do we accomplish such specialized tasks in the Combined Engineering model? We do this by forming virtual teams (v-teams) with specific engineers from each feature team asked to take on special roles in addition to the normal feature development role. In other words, we kept the specialization needed for some tasks but distributed that work into the feature teams from the central teams.

We created an Test Arch v-team and put some of our most senior engineers on it. They were responsible for building the new test frameworks and championing the changes through the org. We created another v-team with subject matter experts in quality tenets like Security and Accessibility. We created Performance v-team which identified common performance bottlenecks in the product and drove changes. Another v-team was responsible for monitoring the health of the CI pipeline and driving quick actions. Engineers in these special v-teams share expertise but their accountability is aligned to the feature team they work in.

Higher Velocity

On the surface it would appear that every engineer is now doing twice or more amount of work than before and therefore feature velocity would slow down. But total capacity in feature teams hasn't changed because of merging of dev and test teams. The move to combined engineering definitely required increased investment in training and development of new skills for each engineer, but it was more than offset by the efficiency gain from the reduced handoff and new testing practices. We have seen continuous and fairly noticeable gain in feature velocity since implementing this change.

As Partner Director of Engineering in Microsoft's Cloud and Enterprise division, Munil Shah leads engineering for Azure DevOps and TFS products. Munil has over 20 years of experience building large scale software and distributed services. Prior to his current role, he held various engineering leadership positions Bing Advertising and Windows groups at Microsoft. He is passionate about leading engineering teams through significant transformation to deliver successful solutions to customers.

Feedback

Submit and view feedback for

This product | This page

Is this page helpful?

Yes No

In this article

The challenge of scale

Real projects might be 50,000 lines of code or 5 million lines of code

They are linked to libraries which change and might cause failures in this code

Complex systems can have unforeseen interactions between different parts of the system

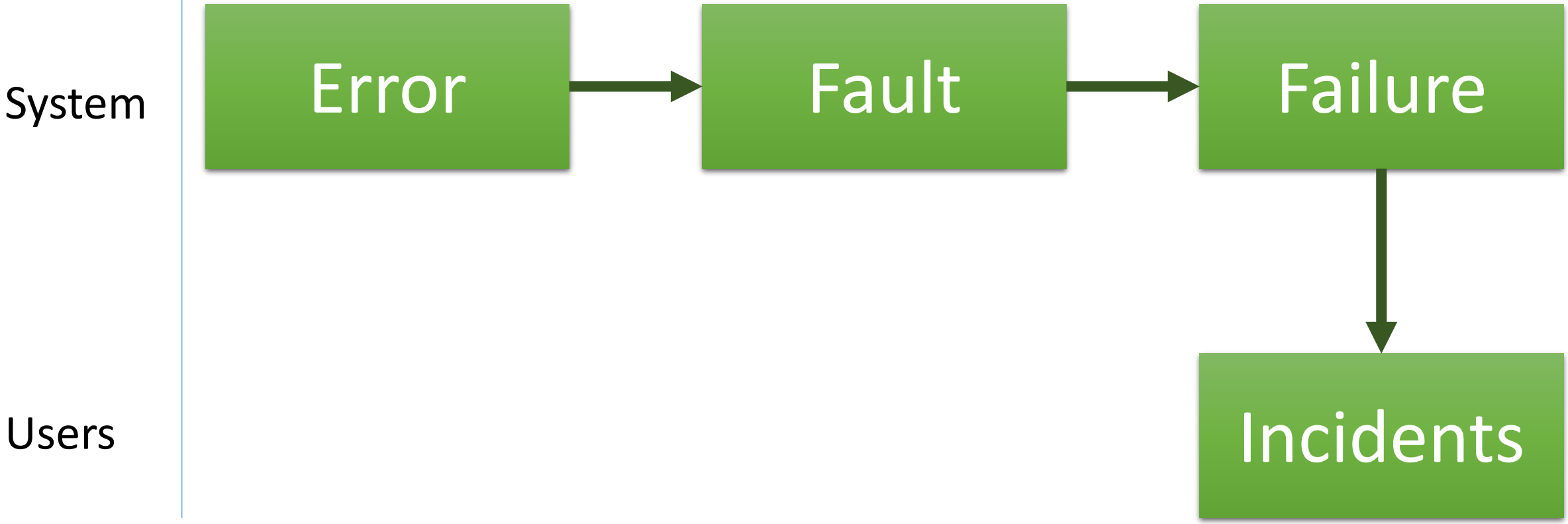
In this course, we will think about how we manage real systems such as those that Microsoft make

Any Questions?

Basic Terminology

Errors, Faults and Failures

There are different aspects to the problems we find in software.



Errors, Faults and Failures

Errors (or mistakes) – A problem that is introduced into a system during development.

Faults (defects) - A result of an error.

- If a designer makes an error that adds something that we don't want, the fault is incorrect execution.
- If a designer makes an error that misses out something, then the fault is something that is not in the software.

Failures – The result when a fault executes.

Incidents – The symptoms that indicates that there is a fault.

Test Cases

In testing, we develop different test cases

- A set of inputs to a system
- An expected output

Each test case should be identifiable, e.g. assigned a Test ID

We should review the outputs of tests to check if they reveal any problems and areas for further testing

Relationship between Development, Testing and Quality Assurance

Testing and the Lifecycle

How do we make software of the size and complexity of Microsoft and other companies…?

Back to Software Lifecycle module:

- Make sure people build things with right tools and methods
- Make sure the system has a sensible architecture
- Use walkthroughs to make sure we agree on what is being done and how it is being done

TESTING is also a vital part of this process

The Joel Test on how to better code

1. Do you use source control?
2. Can you make a build in one step?
3. Do you make daily builds?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
6. Do you have an up-to-date schedule?
7. Do you have a spec?
8. Do programmers have quiet working conditions?
9. Do you use the best tools money can buy?
10. Do you have testers?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?

http://www.joelonsoftware.com/articles/fog0000000043.html

What is a good score?

"A score of 12 is perfect, 11 is tolerable, but 10 or lower and you've got serious problems.

The truth is that most software organizations are running with a score of 2 or 3, and they need serious help, because companies like Microsoft run at 12 full-time.”

Joel Spolsky, joelonsoftware.com

Where testing fits

- Testing checks your daily building (2,3)
- Testing shows that bugs have been fixed (4,5)
- Testing indicates how you are doing against the schedule (6)
- Testing validates development against the specification (7,10)
- Usability testing checks you are building what users want (12)

The Joel Test on how to better code

http://www.joelonsoftware.com/articles/fog0000000043.html

1. Do you use source control?
2. Can you make a build in one step?
3. Do you make daily builds?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
6. Do you have an up-to-date schedule?
7. Do you have a specification?
8. Do programmers have quiet working conditions?
9. Do you use the best tools money can buy?
10. Do you have testers?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?

The Joel Test on how to better code

1. Do you use source control?
2. Can you make a build in one step?
3. Do you make daily builds?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
6. Do you have an up-to-date schedule?
7. Do you have a spec?
8. Do programmers have quiet working conditions?
9. Do you use the best tools money can buy?
10. Do you have testers?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?

http://www.joelonsoftware.com/articles/fog0000000043.html

Software Quality

Software Quality Management is concerned with ensuring that developed software systems are 'fit for purpose.'

From: Sommerville, "Software Engineering, Global Edition'

- We need processes and standards that should help a team to develop high-quality products
- We need to check that the processes and standards have been followed
- We will be thinking further about Software Quality and how it can be managed

Test everything?

Test Everything?

Is it possible to test all possible aspects of a program?
Why do you think that?

What is considered ‘good enough’? Some topics:

- Test early and test often
- Integrate development and testing lifecycles
- Formalize the testing methodology
- Develop a comprehensive test plan
- Use static and dynamic testing

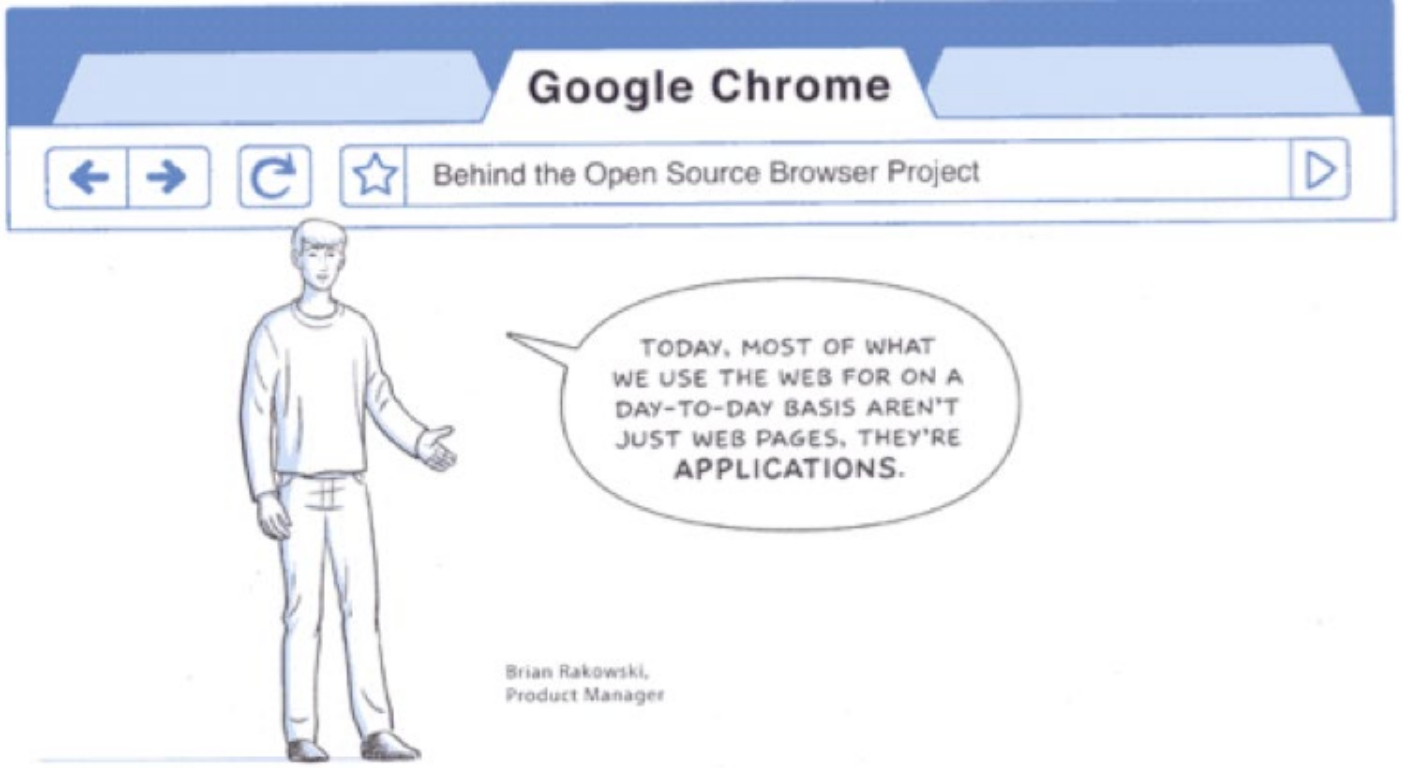
Balancing time, money and quality

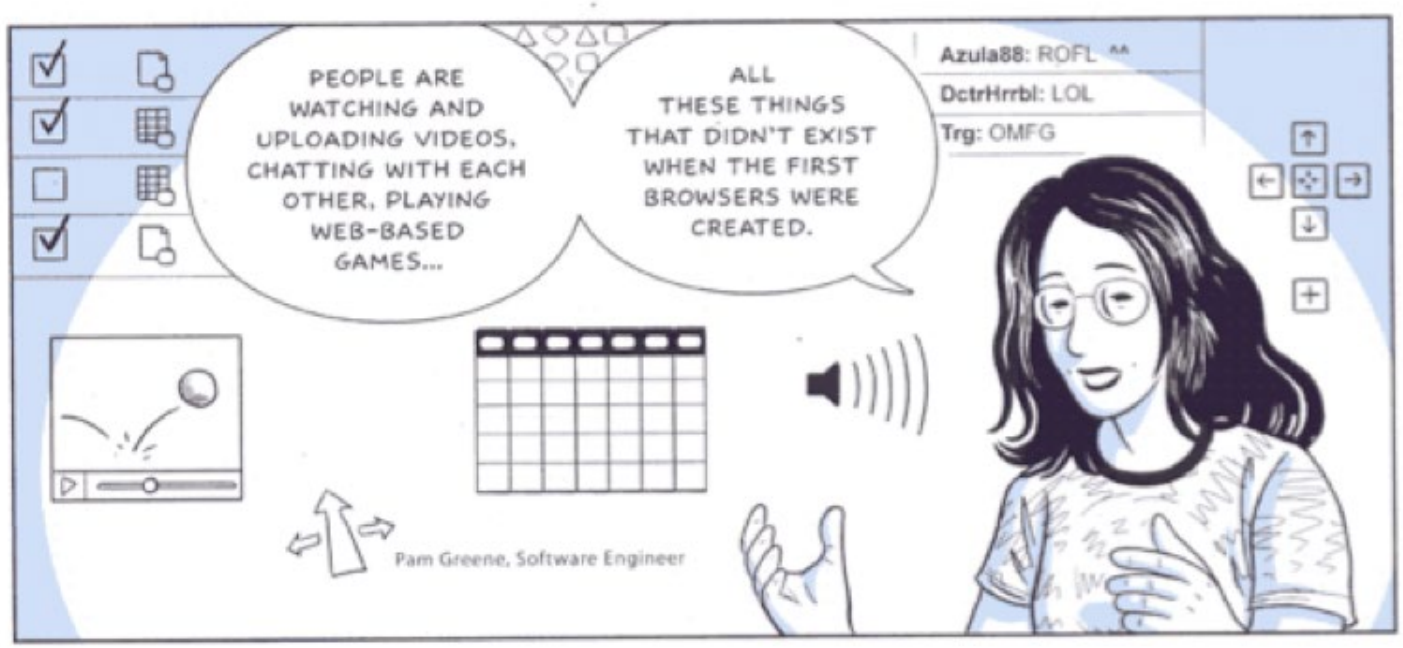
One view of Google's Experience

http://www.google.com/googlebooks/chrome/

The document is covered by the license

- http://creativecommons.org/licenses/by-nc-nd/2.5/legalcode

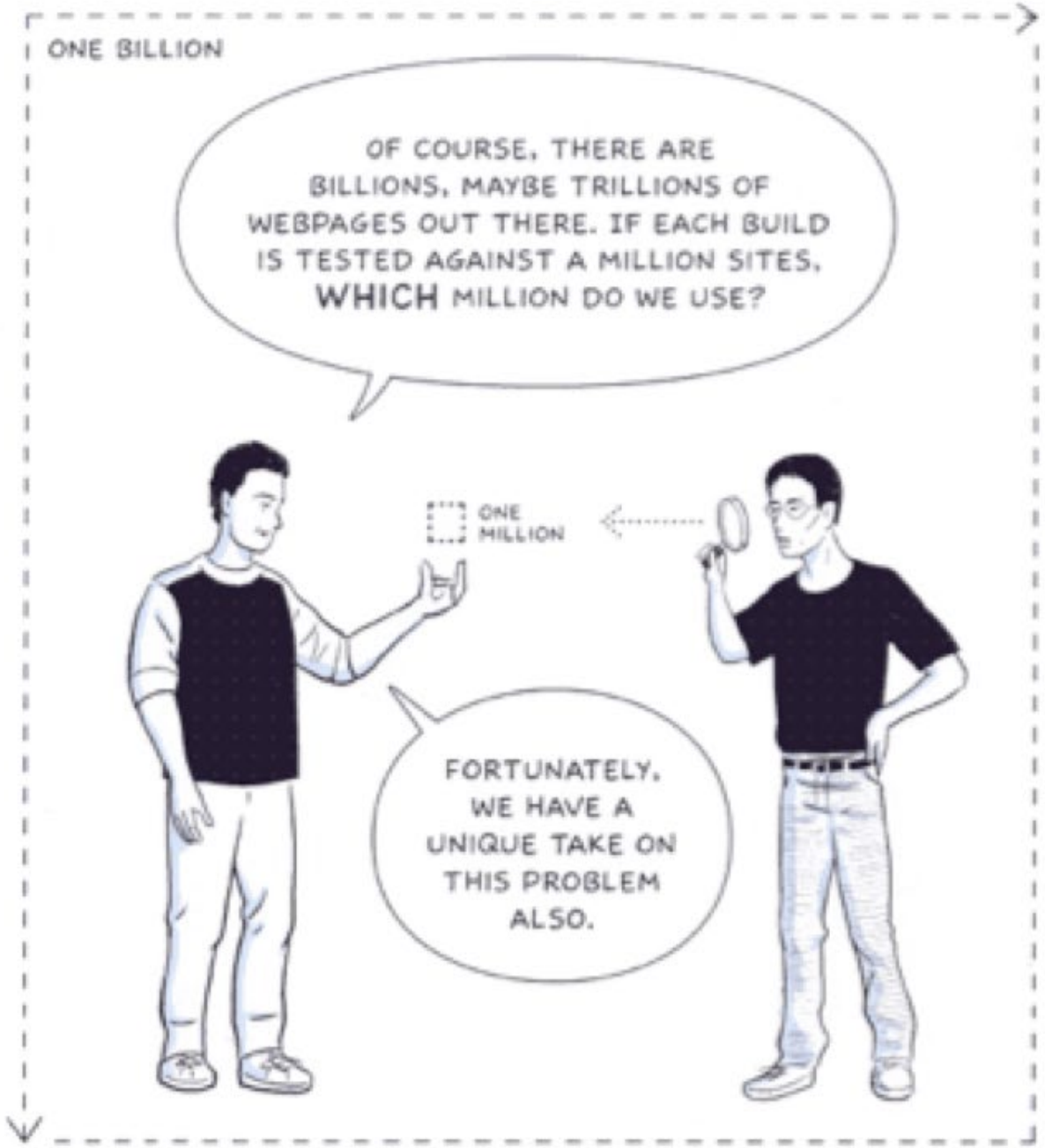

Google Chrome
Behind the Open Source Browser Project
TODAY, MOST OF WHAT WE USE THE WEB FOR ON A DAY-TO-DAY BASIS AREN'T JUST WEB PAGES, THEY'RE APPLICATIONS.
Brian Rakowski, Product Manager

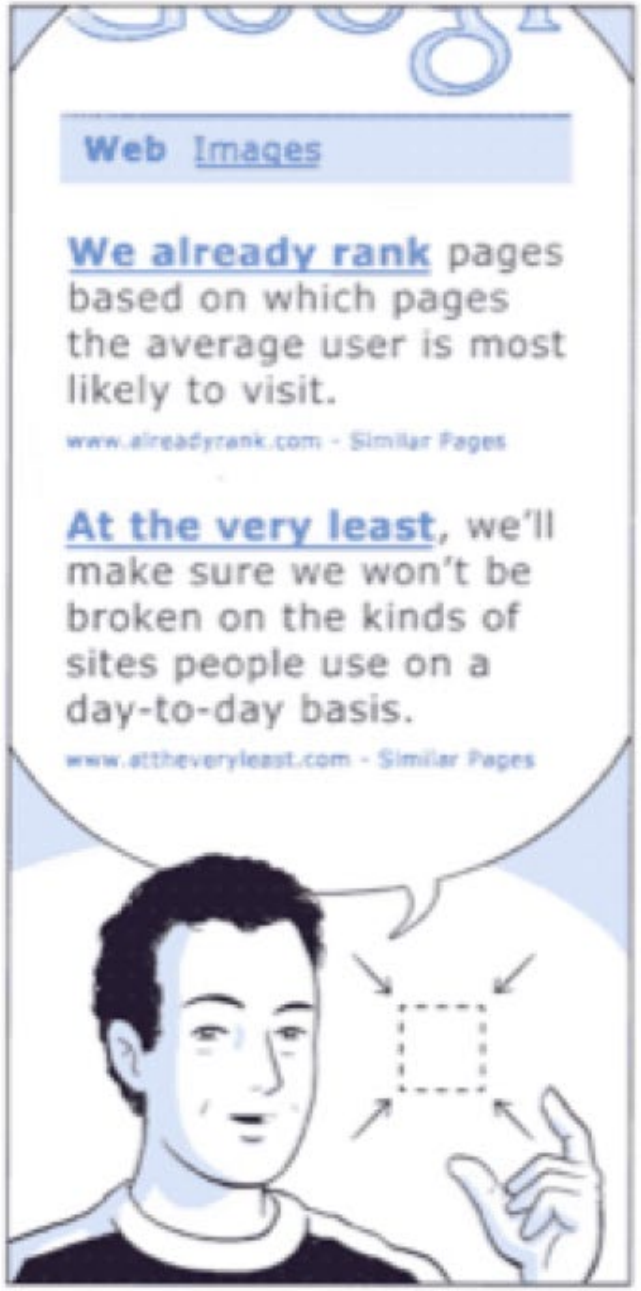

PEOPLE ARE WATCHING AND UPLOADING VIDEOS, CHATTING WITH EACH OTHER, PLAYING WEB-BASED GAMES...
ALL THESE THINGS THAT DIDN'T EXIST WHEN THE FIRST BROWSERS WERE CREATED.
Azula88: ROFL
DctrHrrbl: LOL
Trg: OMFG
Pam Greene, Software Engineer


FIRST, BROWSERS NEED TO BE MORE STABLE. WHEN YOU'RE WRITING AN IMPORTANT EMAIL OR EDITING A DOCUMENT, A BROWSER CRASH IS A BIG DEAL.
Darin Fisher, Software Engineer
BROWSERS ALSO NEED TO BE FASTER. THEY NEED TO START FASTER, LOAD PAGES FASTER --
-- AND FOR WEB APPS, JAVASCRIPT ITSELF CAN BE A LOT FASTER.
Lars Bak, Software Engineer
Kasper Lund, Software Engineer
THEY NEED TO BE MORE SECURE. GIVEN WHAT'S KNOWN ABOUT MASS BROWSER EXPLOITS, BROWSERS NEED ARCHITECTURAL CHANGES TO DISADVANTAGE MALWARE.
Ian Fette, Product Manager


GOOGLE CHROME IS A MASSIVE, COMPLICATED PRODUCT THAT WILL NEED TO LOAD BILLIONS OF DIFFERENT WEB PAGES, SO TESTING IS CRITICAL.
Huan Ren, Software Engineer
WebMD
FORTUNATELY, HERE AT GOOGLE, WE HAVE AN EQUALLY MASSIVE INFRASTRUCTURE FOR CRAWLING WEB PAGES.
WITHIN 20-30 MINUTES OF EACH NEW BROWSER BUILD, WE CAN TEST IT ON TENS OF THOUSANDS OF DIFFERENT WEB PAGES.
EACH WEEK, "CHROME BOT" TESTS MILLIONS OF PAGES, GIVING OUR DEVELOPERS EARLY RESULTS THEY'D OTHERWISE HAVE TO WAIT UNTIL EXTERNAL BETA FOR.


THE KEY IS CATCHING PROBLEMS AS EARLY AS POSSIBLE. IT IS LESS EXPENSIVE AND EASIER TO FIX THEM RIGHT AWAY. AFTER A FEW DAYS IT IS HARDER TO TRACK THEM DOWN.
SPLAT!
BZZZZ
AND CATCHING THEM EARLY HELPS ENGINEERS WRITE BETTER CODE. THEY SAY, "OH, THIS MISTAKE IS PART OF A PATTERN" AND THE NEXT TIME, THEY'RE LESS LIKELY TO MAKE IT.
9
Erik Kay, Software Engineer

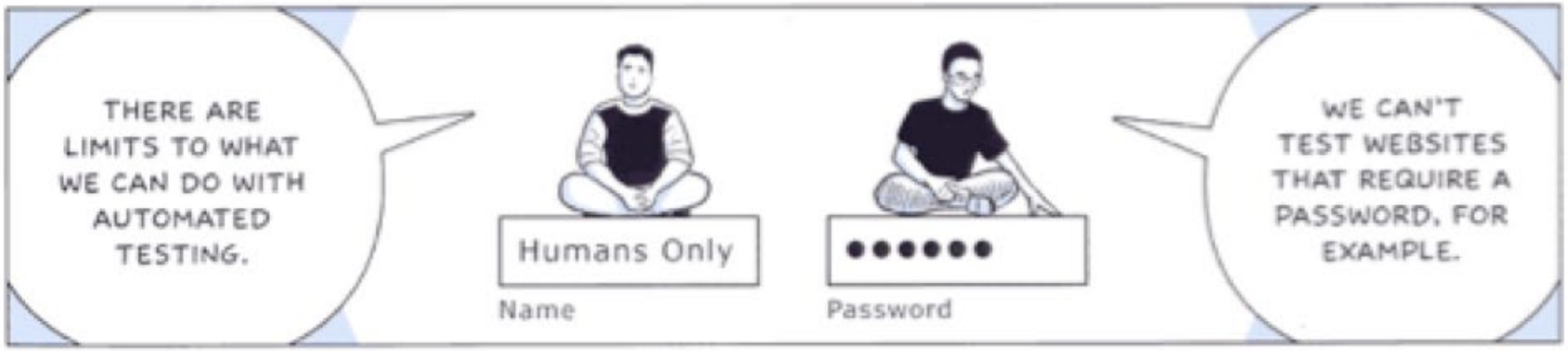

ONE BILLION
OF COURSE, THERE ARE BILLIONS, MAYBE TRILLIONS OF WEBPAGES OUT THERE. IF EACH BUILD IS TESTED AGAINST A MILLION SITES, WHICH MILLION DO WE USE?
ONE MILLION
FORTUNATELY, WE HAVE A UNIQUE TAKE ON THIS PROBLEM ALSO.


Web Images
We already rank pages based on which pages the average user is most likely to visit.
www.alreadyrank.com - Similar Pages
At the very least, we'll make sure we won't be broken on the kinds of sites people use on a day-to-day basis.
www.attheveryleast.com - Similar Pages

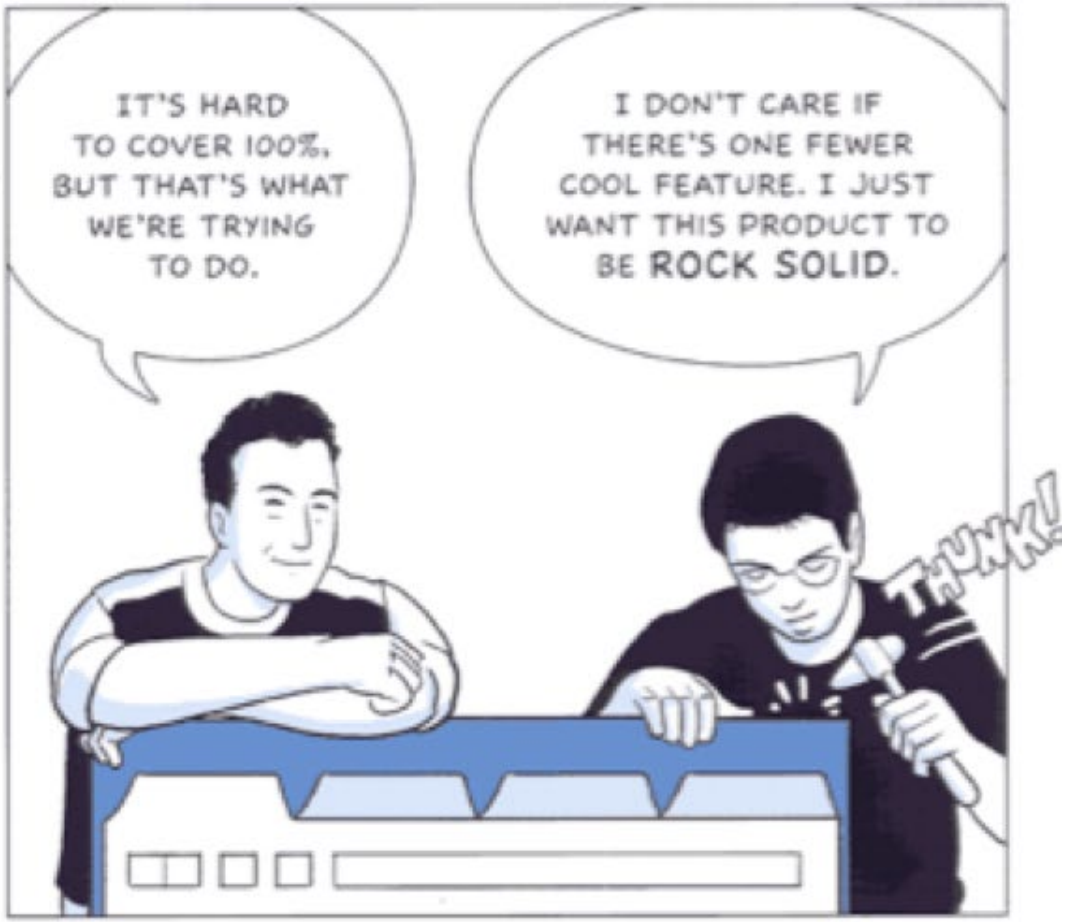

THERE ARE SEVERAL WAYS WE TEST EACH CHECK-IN. FROM UNIT TESTS OF INDIVIDUAL PIECES OF CODE --
-- TO AUTOMATED UI TESTING OF SCRIPTED USER ACTIONS LIKE "CLICKED BACK BUTTON... WENT TO PAGE..." --
-- TO FUZZ TESTING: SENDING YOUR APPLICATION RANDOM INPUT.
Pam Greene, Software Engineer
ACK! TAP! OOK! TAP! TAP!
TAP! TAP! TAP! TAP!


IN LAYOUT TESTING, WEBKIT FOUND THAT PRODUCING A SCHEMATIC OF WHAT THE BROWSER THINKS IT'S DISPLAYING IS A MORE PRECISE WAY TO COMPARE LAYOUTS THAN TAKING SCREENSHOTS AND CREATING A CRYPTOGRAPHIC HASH.


WHEN WE STARTED WE WERE PASSING 23% OF WEBKIT'S LAYOUT TESTS. MOVING FROM THERE TO 99% HAS BEEN A FUN CHALLENGE AND AN INTERESTING EXAMPLE OF TEST-DRIVEN DESIGN.


THERE ARE LIMITS TO WHAT WE CAN DO WITH AUTOMATED TESTING.
Humans Only
Name
Password
WE CAN'T TEST WEBSITES THAT REQUIRE A PASSWORD, FOR EXAMPLE.


AND IT'S NOT THE SAME AS A HUMAN BEING WALKING AROUND AND MISUSING THINGS. WE ARE USING THE BROWSER IN THE WAY WE'VE DESIGNED IT TO BE USED.


IT'S HARD TO COVER 100%, BUT THAT'S WHAT WE'RE TRYING TO DO.
I DON'T CARE IF THERE'S ONE FEWER COOL FEATURE. I JUST WANT THIS PRODUCT TO BE ROCK SOLID.
THUNK!

Summary

Terms:

- Error, Fault, Failure, Incident, Test Case

Relationship between testing and development

- Joel's example list of points for good development practice

The need for automation at different levels of testing

- Example of testing at Google

Any Questions?