/5

CMPSCI 630 Systems **Fall 2019**

Lecture 12/5

Lecturer: Emery Berger *Scribe: Arta Razavi*

12.1 Testing

In the past software developers would develop code and have testers test it to find bugs
Today testing has been pushed out to the users
”you are the tester” = the users are the testers

12.2 Correctness is an Economic Issue

correctness is more of an economic question
In 1968 there was a ”software crisis” and that somehow lead to the start of software engineering
It used to cost lots of money and take a long time to develop useful software
The big question became how can we make programs more correct?
There needed to be software architecture improvements made in order to make software development profitable. This lead to the development of:

- waterfall
- agile
- scrum
- TDD (test driven development)

Because of these software architecture improvements today software developers can create code that is ”good enough”. This Means they are able to ship software quickly and efficiently to the hands of costumers and make profit from it.

12.3 Solve Economic Issues

Users are more forgiving about failures today. For example we do not care if our cellphone drops a call every now and then.
Users favor features so they don’t mind trading off correctness for an increase in feature releases
”first mover advantage” means first to the market will get user base and crowd everyone else out. A good example of this is social media sites because it is hard to convince everyone to migrate to a new service if there is no one there.
network effect means that products and services gain additional value as they gain more users

”second system advantage” is when a company learns from the mistakes of companies that came before it. Facebook did this when they came after Myspace.
This reinforces the notion that it is more profitable to ship products that are kind of crappy but be able to push them out quickly versus spending a really long time developing perfect products
how do we know when the software is ready / good enough?
We study the relationship between product value and the time it took to ship it to land at a certain threshold for correctness that the costumer will put up with.

12.4 Bug Reports

user reports and bug feedback are space and not perfect
In order to reproduce the error we need all the steps that lead to it
bug reports often fail to tell you the steps you need to take to reproduce the problem because ordinary people do not know this is necessary.
Microsoft can bypass this issue and get quality bug reports automatically without any user intervention
Watson (Windows error reporting) is the tool developed by Microsoft to track bugs
When a program fails it asks you ”do you want to send a error report?”
why people say yes:

- give up privacy in exchange for fixing errors important to you
- want to help make application better
- usually pretty easy to just say ”yes” on bug popup

why people say no:

- invasion of privacy
- there is a high volume of error so there is a chance yours will get ignored
- too much work its not just ”yes”/”no” sometimes there are follow up questions that annoy users

12.5 What Happens to Crash Reports?

a mini dump is created containing metadata from the crash
this metadata contains:

- software versions
- operating system version
- some meaningful state information. This is usually the stack and not the heap because the heap is too large.
- PID (personally identifiable information)

When personally identifiable information is shared then we start running into user privacy issues
GDPR: Europe has privacy protection laws for its users that require websites to disclose when they collect personally identifiable information like cookies and be transparent about their use for that information. This means the website must give the user a clear warning when their data is being collected and stored.
lots of companies decided it was just easier to become GDPR compliant all across the board instead of only being GDPR compliant for their Europe based users to make everything easier
GDPR also states that user data cannot be retained for longer than 120 days which makes it a time sensitive issue for bug fixes because you can only keep the bug reports for 120 days
differential privacy: we want to be able to release user data but make it hard for people to gain access to personally identifying information. This is done through randomization and some stats magic.
this is a hard issue to solve because less info is better for privacy but also horrible for bug reports
for debugging the best case is getting entire user session and perform root cause analysis on it
min dumps: are stack traces that contain function calls. They bucket errors into groups by hashing their metadata and when that bucket is full they notify a developer

12.6 Ways to Hunt Errors

Microsoft uses Watson to triage errors
Companies can perform field experiments because they believe the bugs they should be focusing on are the ones that the users are experiencing. This is a form of uncontrolled testing that is different from traditional lab controlled testing.
companies also deploy software as an experiment to part of the user base. A segment of users will see new versions / features. This testing is randomized and evenly distributed over the user base. Software is tested this way and then gradually rolled out to more users.
this gradual deployment model makes it to test software without effecting the whole population and Facebook used to do this all the time.
also can be refereed to as a ”gradual roll out”
There are some downsides to the gradual roll out approach:
people do not want to be exposed to experiences that makes their software crash. Breaking a few things sometimes is fine but if you do it all the time you will annoy your users. There is some measurable threshold for this.
Facebook had to return to traditional testing approaches because users were getting annoyed of their constant tests

12.7 Logging

logs are very long and hard to process
logging often will slow the program down
logging not enough times cannot reconstruct meaningful errors
sampling at random will not find rare errors it is only able to find those that happen often

12.8 Kinds of Testing

we now have access to:

- program

- inputs
- you control everything
- deterministic program
- observable results

If the end results do not match expectation you must then debug the program for errors
if any of the things from the above list are missing what do you do? think harder? be smarter? you cannot anticipate the un-anticipated and write tests for it.
end to end test: this is the worst kind of testing it can only tell you if there was an issue in the entire program and not specifically where it was. This is especially problematic in very large programs.
Unit tests: every time the program is run I can run tests on specific parts of it and see if there are any issues.
The issue with unit tests is that you cannot test everything as there are infinite possibilities. Also if underlying code changes many tests might have to be updated because they rely on internal state.
unit tests are restricted to particular class / module and cannot test module level dependencies.
Unit tests can be wrong because they are software too
if the developer writes the tests they introduce bias and a skewed sense of correctness
often times today unit tests are written after the software has been written
integration tests modules are combined and tested as a group to define cross module dependencies. Most software undergoes continuous integration testing.

12.9 When are Tests Written?

a priori is the "best practice"
TDD: (test driven development) write all tests first don't write any code and when you are done write software to fit the tests. This was somehow supposed to remove bias but that didn't really happen. This kind of testing was morale crushing and no one really wants to test all day they want to write code.
XP:(extreme programming) an agile software development framework
pair programming: when two or more software developers work on the same piece of code together using the same computer
companies were somehow aiming to amortise reduce the cost of hiring software developers by making them work harder and more efficiently
generally tests are written after or during software development. Usually you run across a bug and write a test case to test for it. This is known as **regression testing**

12.10 Test Coverage

Cannot know how bug free the program is without some sort of metric
line coverage: write a test for every line of code
line coverage does not imply good tests were written
line coverage is a poor approximation for path coverage
the number of possible paths grows exponentially while the number of lines grows linearly. You cannot use a linear means to cover an exponential cases exhaustively.
flakey tests: when a test fails it could be non deterministic meaning its failure does not necessarily imply the program is incorrect. Multi threaded programs are non deterministic and run into this issue.

12.11 Fuzzing

hugely important for security
This is done automatically you just push a button and get a bunch of tests
these tests are unbiased and randomly generated
Fuzzing makes it more likely to explore cases you have not considered
Fuzzing automatically exposes bugs
Fuzzing is able to explore the state space in an unbiased manner exploring unanticipated corner and edge cases
Fussing has its own issues: complex inputs are unlikely to be randomly generated correctly due to infinite input space leading to the state **space explosion**.

12.12 AFL American Fuzzy Lop

try all mutations of a running program path
coverage driven (branch coverage)
take inputs and mutate them see what happens
randomized process

12.13 DART (direct automatic random testing)

while running program if we come across a branch that is not taken we take note. We then send all program constraints to a SAT solver to find a set of constraints that would set the program down the branch not taken.
this works best for systems that do not interact with the heap much this is because adding constraints for all program variables will overwhelm the SAT solver. The SAT solver will run out of time or be unable to solve problem in state space.
Windows device drivers are an excellent candidate for DART because they do not use the heap
Microsoft use to blue screen all the time in the past due to faulty device drivers
Microsoft now ships DART with the DDK (device driver development kit) and device drivers must pass DART tests in order to be varified