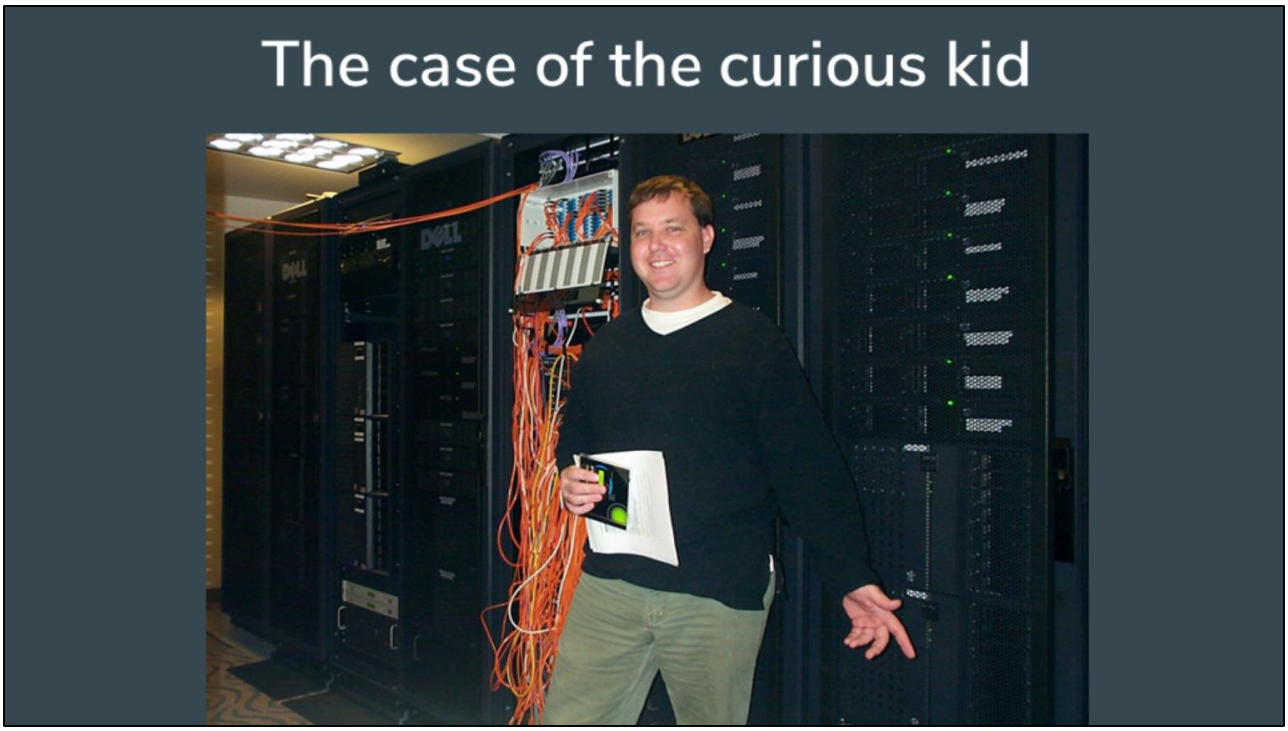
How robust apps are made

UC SANTA BARBARA

How robust apps are made!

- Not fragile, handle problems gracefully
- Consistent user experience, intuitive, easy to use
- Maintainable, easy to change



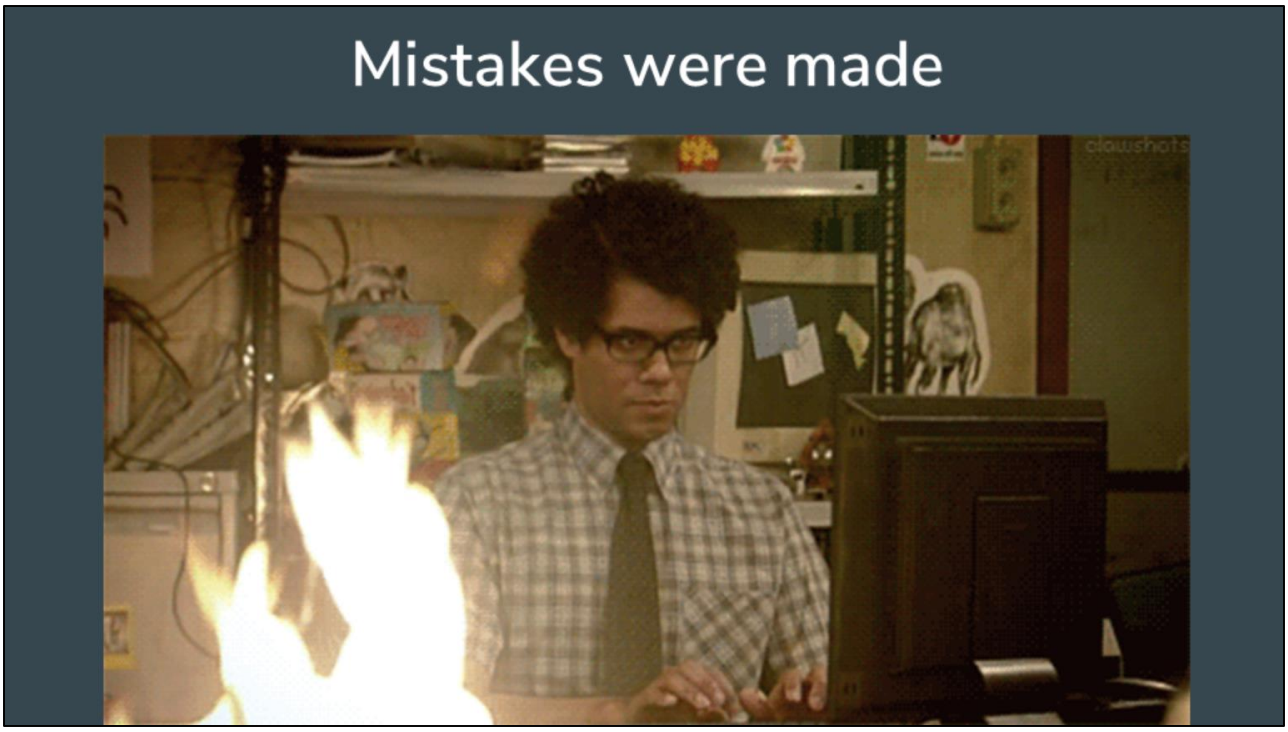
Let me tell you a story about this curious kid.

He found computers fascinating. He loved to write some special words and the computer would do things. He wanted to know how that worked so he investigated everything around it. Programming, networking, hardware, web servers, security, you name it, this kid looked into it. This curiosity landed him a job. “Wow I get paid to do my hobby, how cool is that”, he would say.

Soon he was doing more complex tasks. He took pride in making a difference in people’s work and lives. But then things would break or fail in unexpected ways. The calls would come in, late at night, or on the weekend, or during a really busy time for business.

Soon this curious kid was just fighting fires and not making new or better things.

Yes, this curious kid was me!



I want to share with you some of the hard lessons I've learned. I want my pain and suffering to mean something, to help someone avoid the same pitfalls.

I geared this discussion not just for software devs, but towards the people that do many things, the jack of all trades out there. I know you want your apps to just work so you can focus on those many other things.

This discussion is also useful for people who don't write code, but you have to serve it, or secure it, or move it around the network. This should give you some insight into some of the things code monkeys do (or should do) to keep things running smoothly.

Even you you are using the cloud, you can still run fragile code there, and you must design and test them properly. But if you got four 9s and want to go to five 9s, this is not for you.

Let me introduce myself.

I've been involved with web apps in one way or another for over 20 years now. Ya since the very beginning.

I've used over 2 dozen programming languages and environments.

I gather requirements, I grok business's needs, I architect and build systems.

I struggle with CSS and public speaking.

I've done information security and intrusion detection at a bank.

I've written learning management systems from scratch using text files to store data.

I've racked servers and I've accidentally run "rm -rf /" as root (it deletes everything, I mean EVERYTHING)!

I love UX, process improvement, and cats. I hate fighting fires.

My adventure buddy and I have visited over 400 of the 1000 California Historical Landmarks. We saw 32 on the way up here. 804 is just west of here, the The Wolfskill Grant, 100 acres given to the UC for an experimental farm in 1937.

Here are the topics. Got two out of the way.

I want this to be an interactive discussion. We have X more minutes together. Thank you for coming to see me ramble.

I will pause after each and ask for your insight!

There are many more topics we could talk about, but I only have so much time before they kick me out.

Some one liners: Don't use spaces. Make everything lowercase! Practice Read-only Fridays.

Environments

Isolated, but configured the same

- Development
- Test
- Staging
- Production
- Others

This is one I see a lot. Have at least two, Production and test. Even for vendor apps. You need to be able to test upgrades and reconfigurations.

“Read/Write” systems

- Use separate web, database, and mail servers.

- Anything writable. Where you send data to an external system. Yes, even email. I found a great tool called MailDev that accept any incoming email and gives you a web-interface to read it.

“Read Only” systems

- Test systems can use read only productions systems like AD, LDAP, and other lookup or dictionary data sources, API, NTP.

So you can FULLY exercise an application without fear of messing things up! I mean FULLY. You need to be able to hit every corner of an app with abandon!

Modern tools and techniques

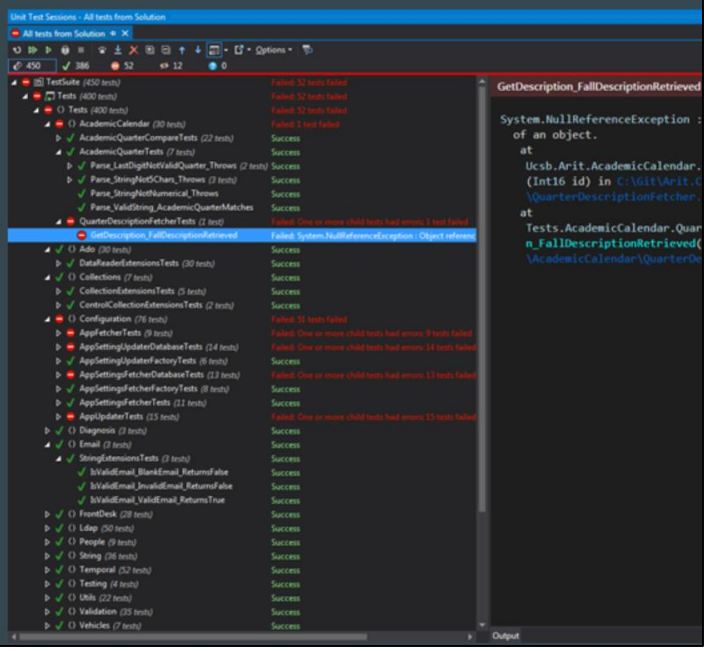
Vendor/Community Supported
Lots of common knowledge
Lots of online help
Learned from past mistakes
Save time and money
Security is baked-in

Code separation

Do one thing well!
Separate and isolate code as much as possible
Allows for testability
Allows for reuse
Removes spaghetti

Testing

```
// Arrange

let x = 2; let y = 5;

// Act

let z = Add(x, y);

// Assert

7.IsEqualTo(z);
```



Unit Testing
Integration Testing
Regression Testing
Load Testing
User Acceptance Testing
UI Testing, manual and/or automated

Topics

Introduction

Story time

Environments

Modern tools and techniques

Code separation

Testing

Source control

Version numbers and releases

Automated build and deploy

Persistence

Logging and alerting

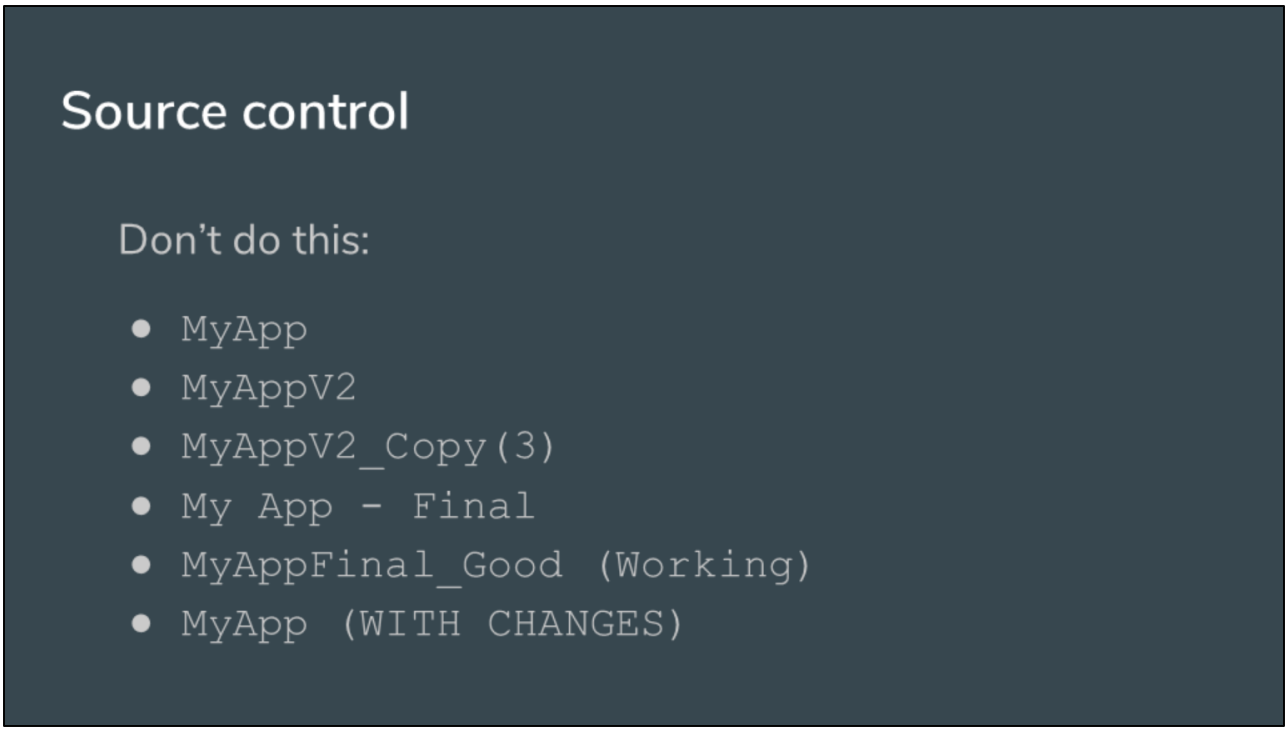
Planning and assessment

Half way done!

Source control

Source Control

It goes by many names but it is basically a system to track files and changes to those files.



This is what I did just a few years ago. Don’t do this.

Which one is the right one?

Which one is in production?

Did we fix things in production and not copy them back into our “good” folder?

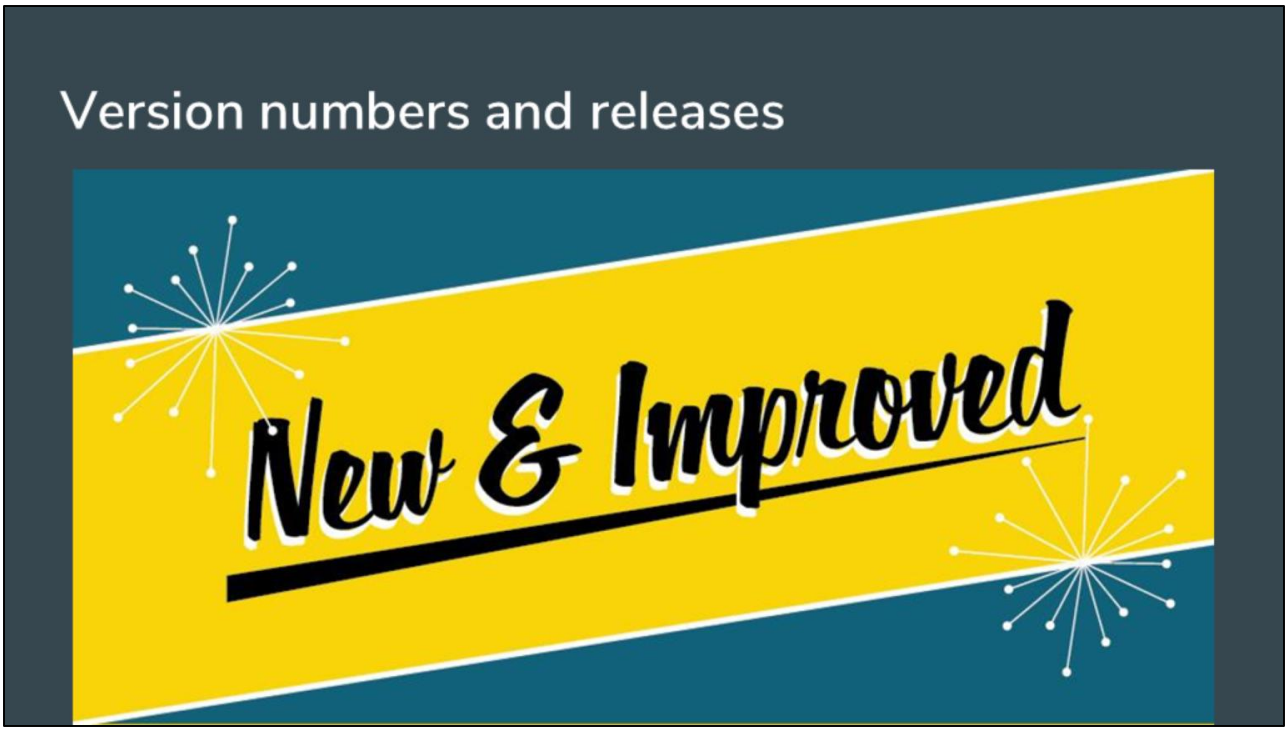
These are folders with many files? Do you trust the state of those files? Did someone else mess something up, did you mess something up and now it is going to break in your next production release?

Use Git!

It’s a quick install on any computer. Command line or GUIs. No complicated client/server needed. Things like GitHub, Bitbucket and Gitlab are really just offsite storage that you sync your code too.

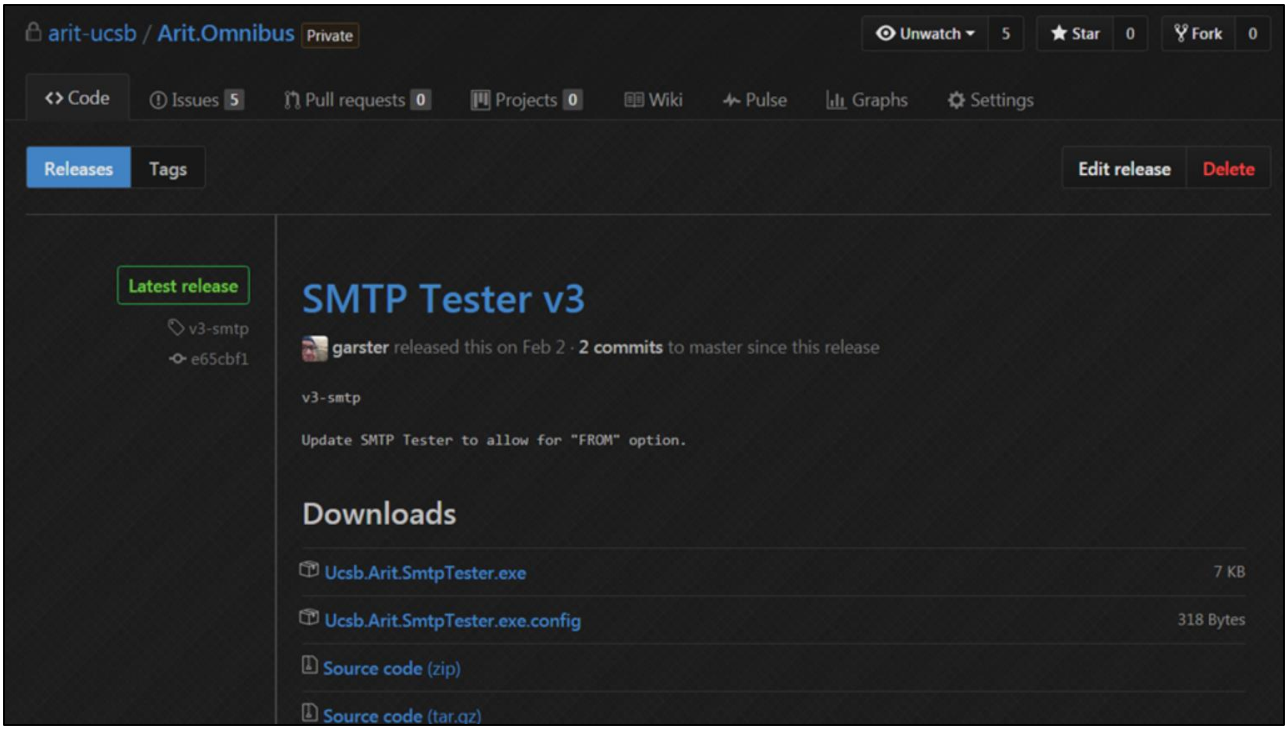
The basics are easy, you can do 90% of what you need with 10% of the features.

My favorite part is if any files is changed unexpectedly. When I open up git it will tell me right away the file has changes and I can simply “discard changes” to get back the last good version.



Version Numbers
Use them! Make them up.
Use the date (18.8) or 1, 2, 3, 4, 5
or SemVer: X.X.X.X (4.5.2.56).
They will help you when things break. You will be able to tell the difference among codebases.
“That was fixed in version 2.1”, “The test server has version 3.5, but Prod has 3.6.”.

Releases
Make a deployable package.
Don’t just update a few files to Prod. How can you tell what is there?
Think of how some of the apps you use are packaged and released.
GitHub has a great “Releases” feature.



This is a simple utility I made. I “Released” it. Compiled code, link to the version in source control. It was actually really easy.

Aka Robots working for you

Continuous Integration, Continuous Deployment, Build Servers.
Setup build profiles that are run automatically so manual steps are not missed.
Like setting up an assembly line. Time consuming to set up but is awesome ones it gets going.

I currently use JetBrains TeamCity. It cost $1000 a year and pays for itself over and over
Watches Github for changes
Compiles code
Runs automated tests
Builds and deploys documentation
Builds release package
Deploy to webserver or NuGet server
Alert if break builds.

Save tons of error prone manual steps.

Topics

Introduction

Story time

Environments

Modern tools and techniques

Code separation

Testing

Source control

Version numbers and releases

Automated build and deploy

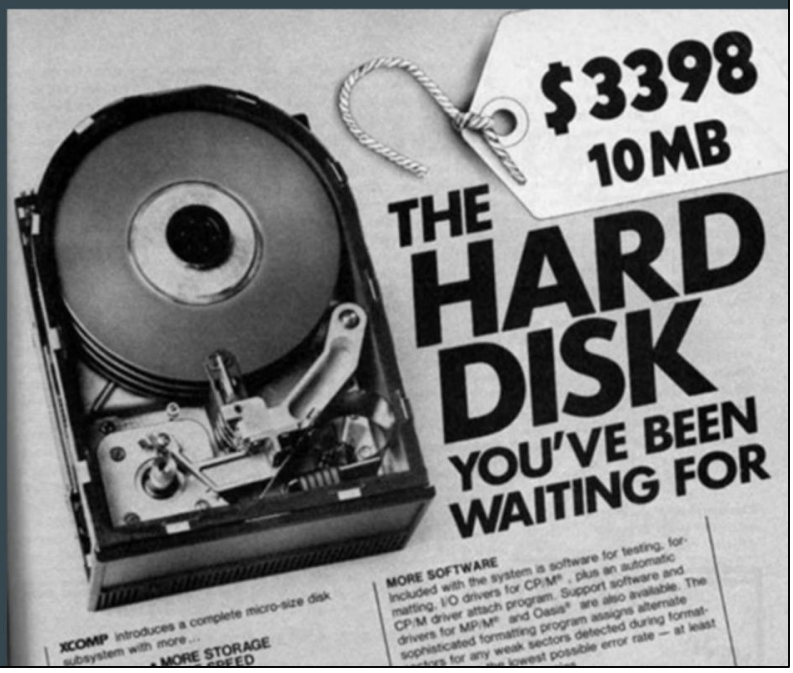
Persistence

Logging and alerting

Planning and assessment

Half way done!

Persistence



Apps use objects in volatile RAM.

We may need to save that info to long-term storage.

Older methods were to use low level database access methods and lots of boilerplate code.

Better way is to use an ORM (Object-relational mapping) tool for Relational databases.

SQL injection nearly possible. That is where a bad actor can run their own sql code.

Or checkout NoSQL. Very simple storage of structured data. MongoDB, AWS Dynamo DB. Couch DB.

There is a lot that could go wrong here so you need your code to check and handle those problems.

Logging and alerting

Capture events and errors as they happen
Know of a problem before it affects too many people.
Better than the “customer early alert system”.
Levels like Debug, Info, Warn, Error, Fatal

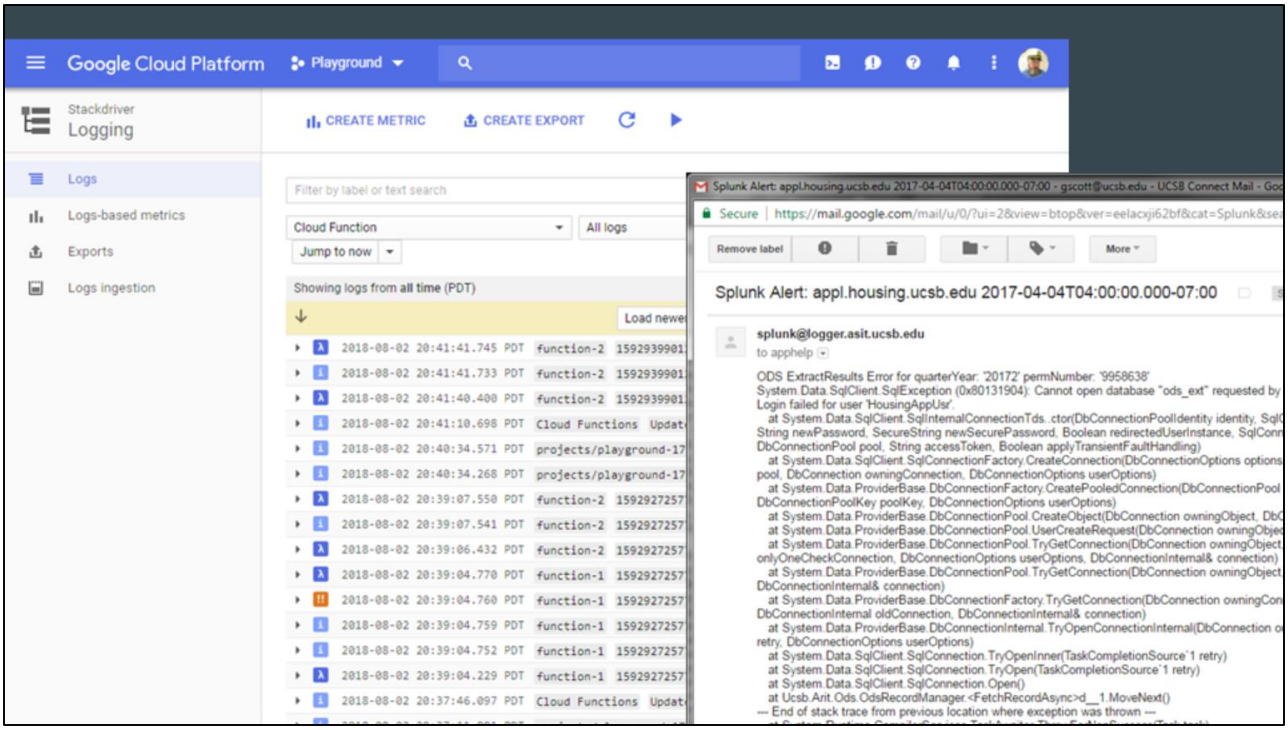
This allows you to set those logging events in code, but only turn up the know to capture them when needed.

Recover gracefully if you can.

Log and alert based on the severity of the problem.

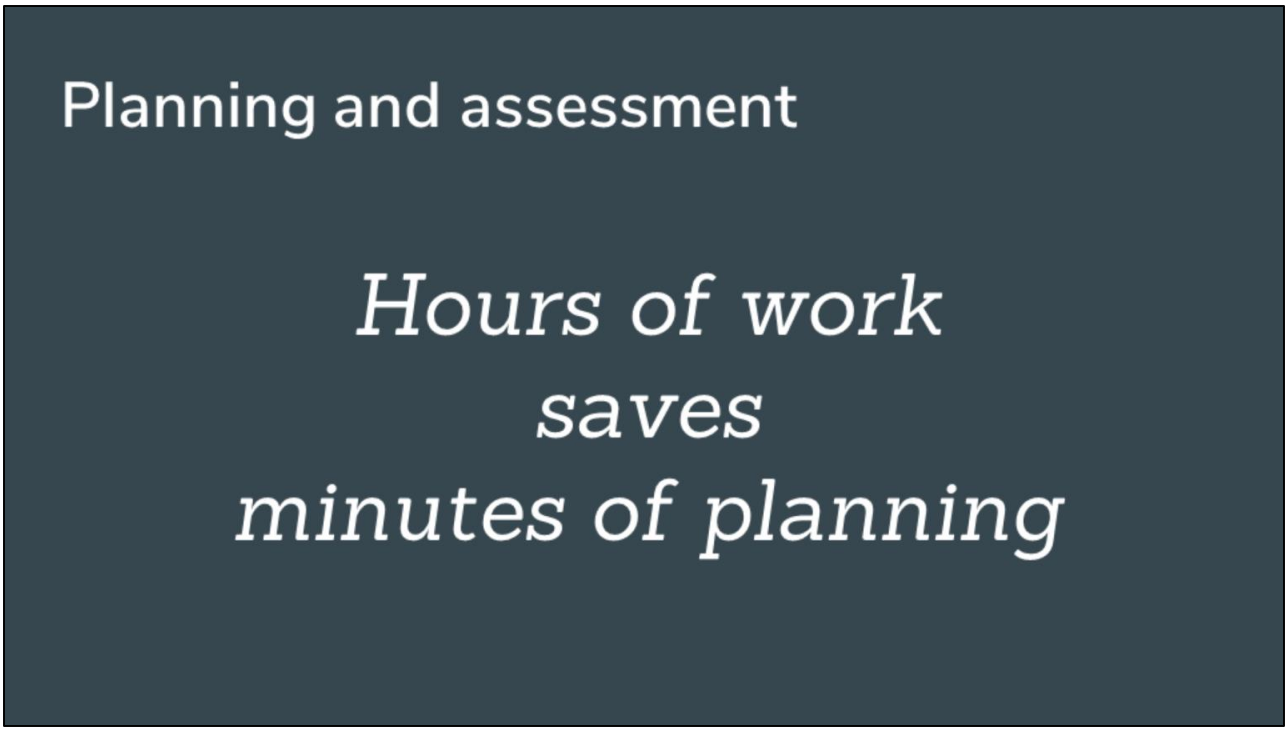
Also if you log to text files, like in a webserver. Those can fill up harddrives.

Ive also seen one not rote and it ended up being a huge 10GB text file. Both times they crashed the server.



That is Google Cloud Platform Stackdriver Logging on the left.

On the right is Splunk emailing us on a Fatal error is saw.



Another story, quick one this time.

Customer reports a bug
Business flips out: We need to fix this major issue RIGHT NOW.
Me: Well we could do XYZ to solve it
Big boss asks me: how long has that bug been there
Me: Let me look, Hmm that code was last touched about 2 years ago (thank you source control).
Big boss: How many customers are affected.
Business: Well, just this one as far as we can tell
Big boss: I think it can wait for our normal change management process.

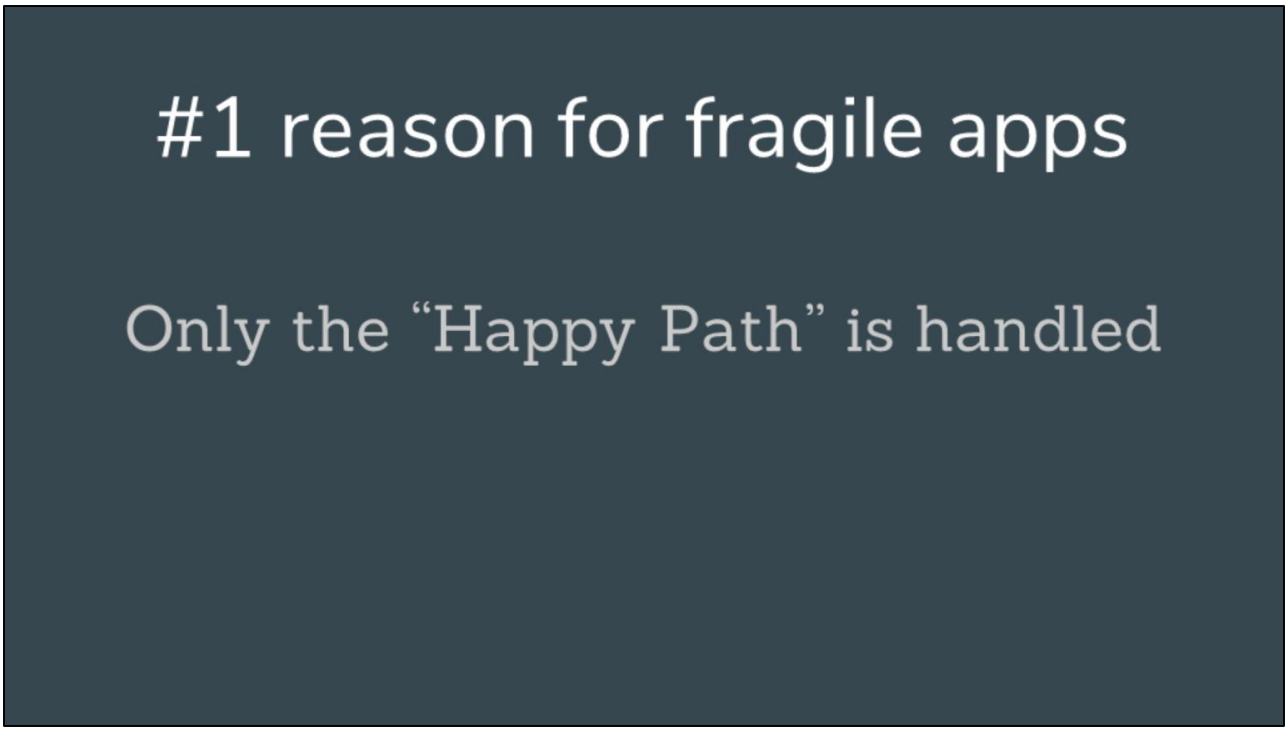
Guess what! My initial solution would have made things worse, but I was given some breathing room to examine the issue and address it properly. I fixed it once, the right way.

Take a minute. Thinks things through. Sleep on it. Talk to others. Get others points of view.

Almost every time I rushed in to do something it has backfired on me. Let the fire burn a bit!


Topics
Introduction
Story time
Environments
Modern tools a
Code separation
Testing
Source control
ion numbers and releases
nated build and deploy
rsistence
Logging and alerting
Planning and assessment
DONE

Done!



Little accommodation for any variations to that flow.

Crash and burns horribly. But our error logging helps dig into this.

Hurts the customer and difficult/time-consuming to diagnose.

Spend all day “firefighting” instead of innovating.

Gary Scott

Administrative & Residential Information Technology

gary.scott@ucsb.edu

garyscott.net

github.com/garster

UC SANTA BARBARA

Thank you. Stay curious!