COMP 587 Lecture 1

Kyle Dewey

About Me

- My research: automated test case generation
  - Testing things that are **hard** to test
- This is my third year at CSUN
- Second time teaching this course

About this Class

- First time this version of the class is taught (tweaked from last time)
- See something wrong? Want something improved? Email me about it! (kyle.dewey@csun.edu)
- I generally operate based on feedback

Bad Feedback

- This guy sucks.
- This class is boring.
- This material is useless.

-I can’t do anything in response to this

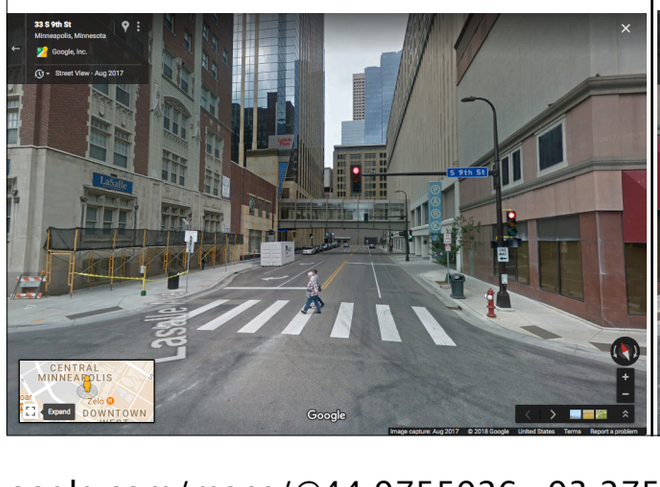
Good Feedback

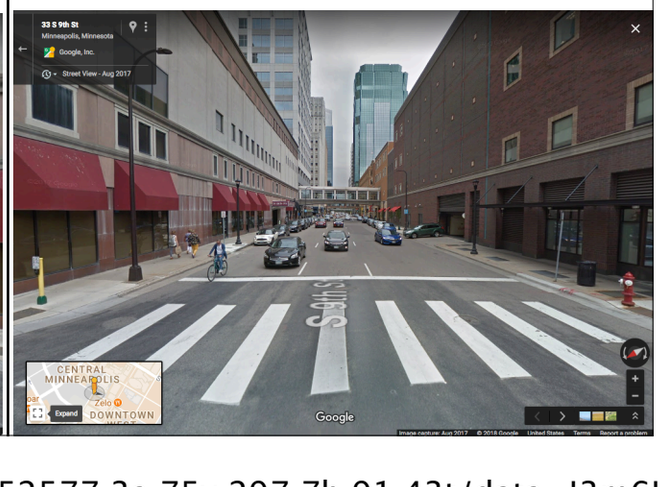
- This guy sucks, *I can't read his writing.*
- This class is boring, *it's way too slow.*
- This material is useless, *I don't see how it relates to anything in reality.*

- I can't fix anything if I don't know what's wrong

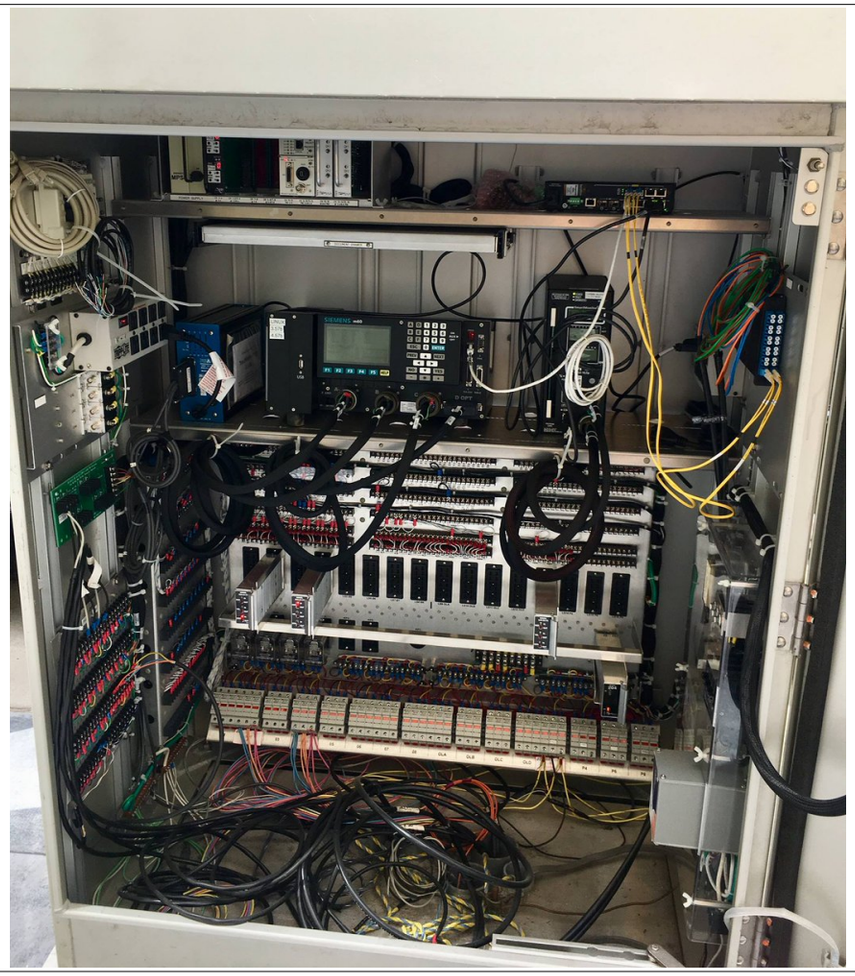
-I can actually do something about this!

Motivation Part #1: (Software) systems are complex.

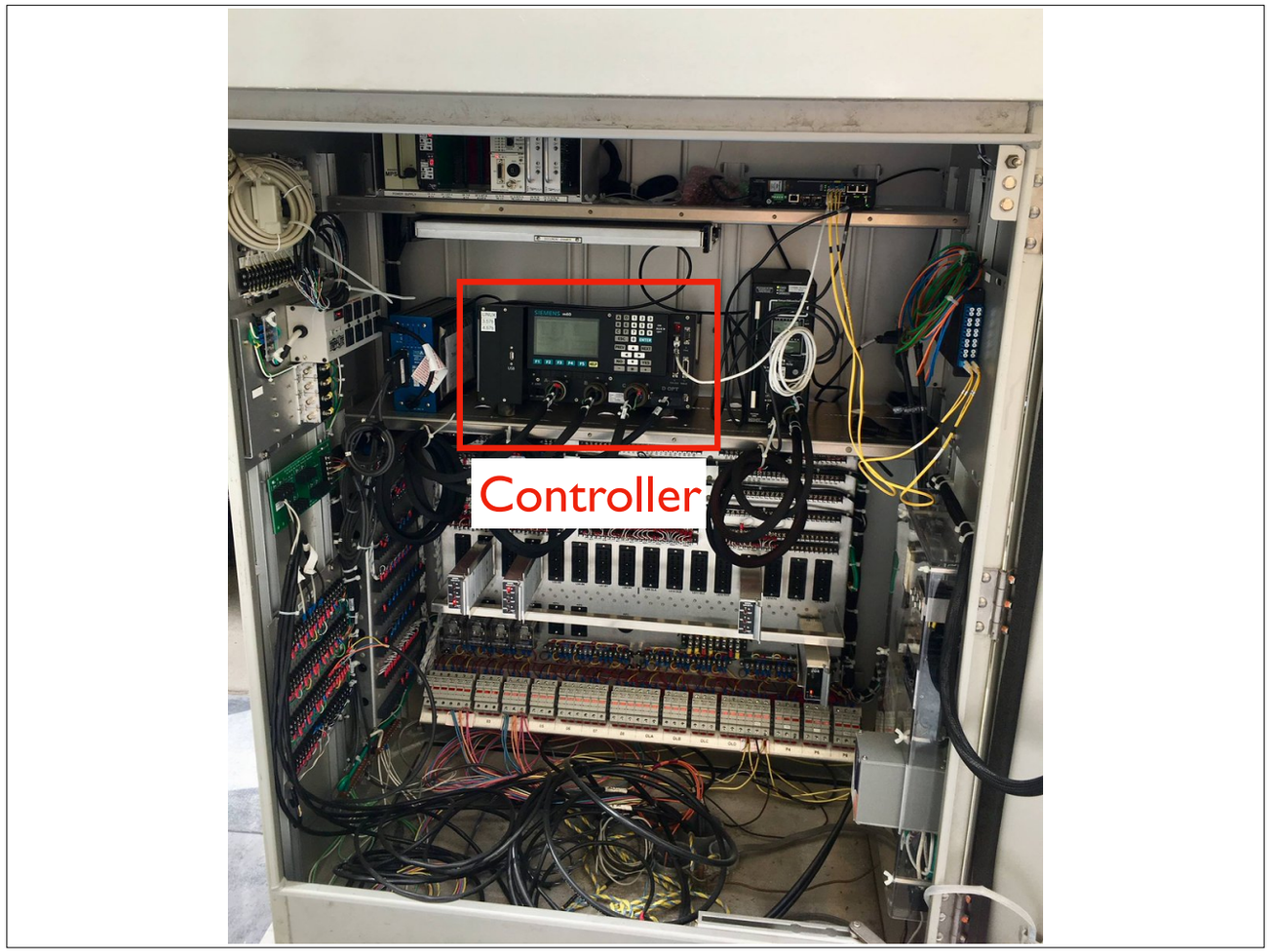




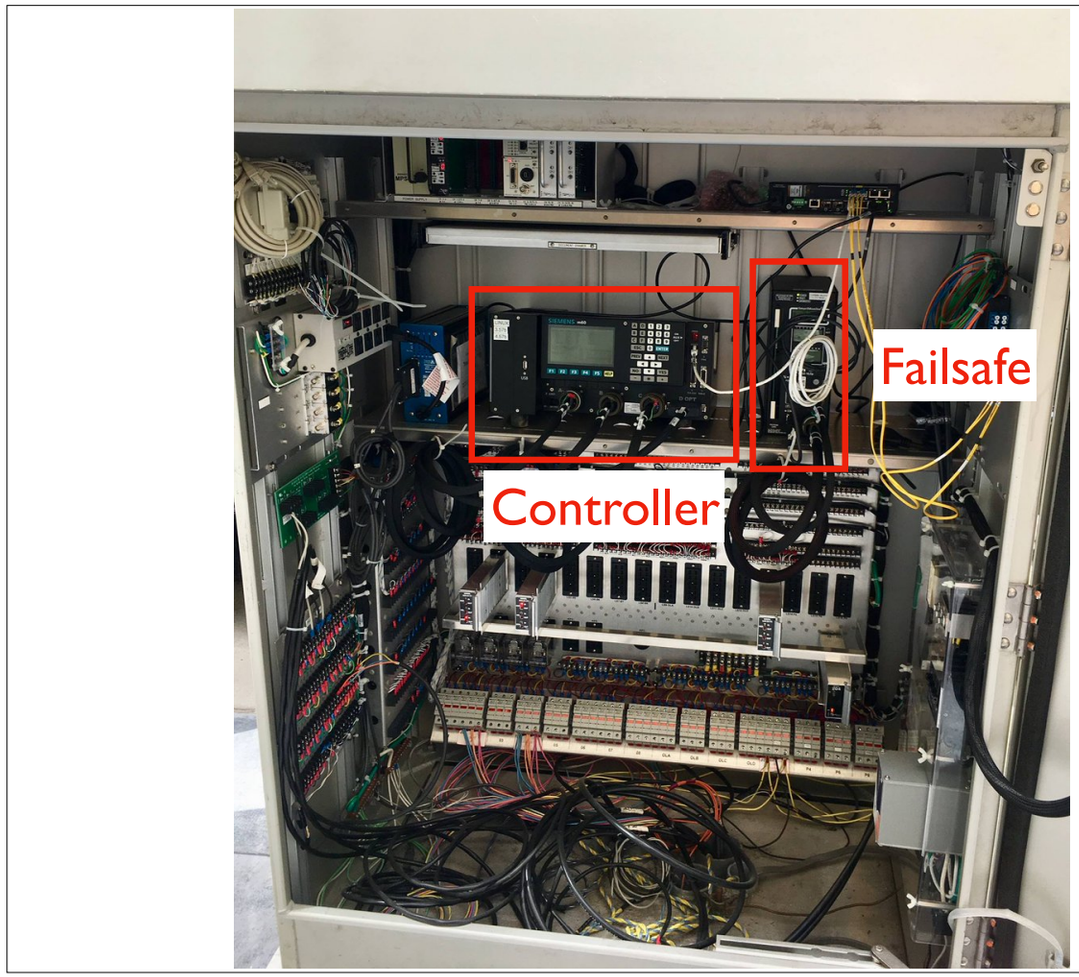
-See this intersection: https://www.google.com/maps/@44.9755926,-93.2752577,3a,75y,297.7h,91.43t/data=!3m6!1e1!3m4!1sqI6wFLM28xt77ncG4har1g!2e0!7i13312!8i6656



-This is the traffic light controller for this intersection
-First impression: wow there are a lot of things in here



-This is the traffic light controller for this intersection
-Controller handles normal operations



-This is the traffic light controller for this intersection
-Failsafe looks at the controller output, and checks if there are conflicting greens
-This is a much simpler job
-If conflicting greens are detected, the failsafe can override the controller (e.g., flashing reds)

Motivation Part #2: (Software) systems usually suck.

1982: Therac-25

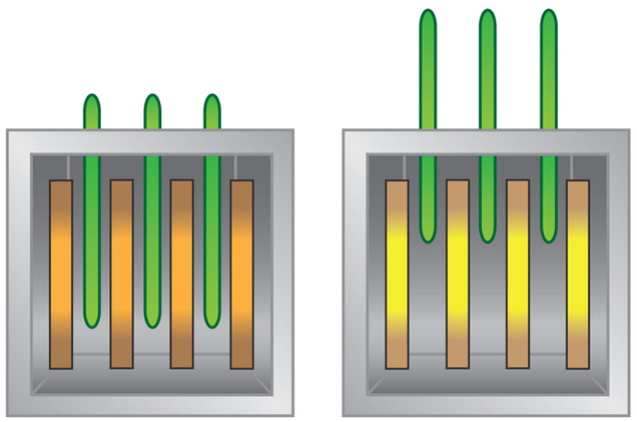
-Race condition meant that if an operator input two commands too quickly, it would give patient ~100X the radiation dose
-Three died from radiation burns
-Poor development strategies were blamed instead of simple bugs

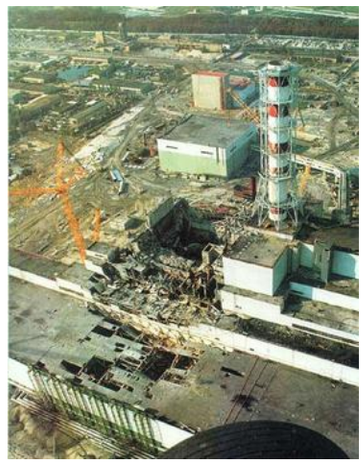
1982: Therac-25

Race condition kills three people.

-Race condition meant that if an operator input two commands too quickly, it would give patient ~100X the radiation dose
-Three died from radiation burns
-Poor development strategies were blamed instead of simple bugs

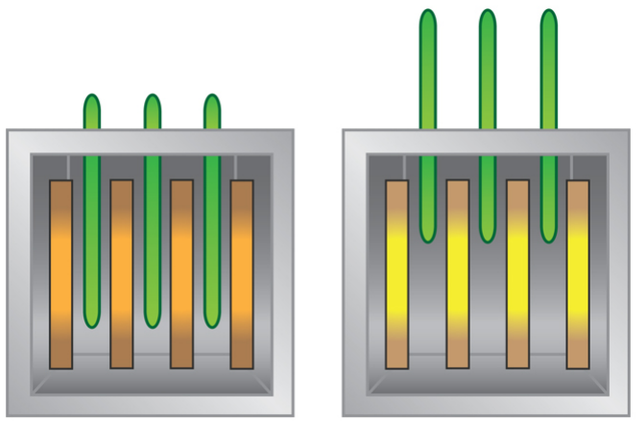
1986: Chernobyl

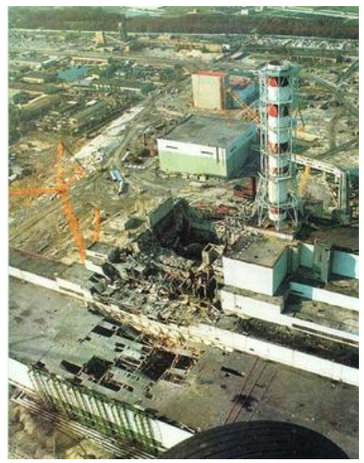




-A systems problem, but not a software system problem
-A test was to be carried out which ultimately require the reactor to be powered-down to a certain level
-Powering-down procedures started, but were unexpectedly delayed when another power plant failed and Chernobyl needed to pick up the slack.
-For nuclear engineering reasons, this delay ultimately made it so the reactor powered-down too quickly later, lower than what the experiment allowed.
-Engineers chose to power the reactor back up (EXTREMELY DANGEROUS)
-Engineers overrode just about every control system
-Final straw was a tiny design flaw that may have made the reactor resemble a nuclear bomb

1986: Chernobyl

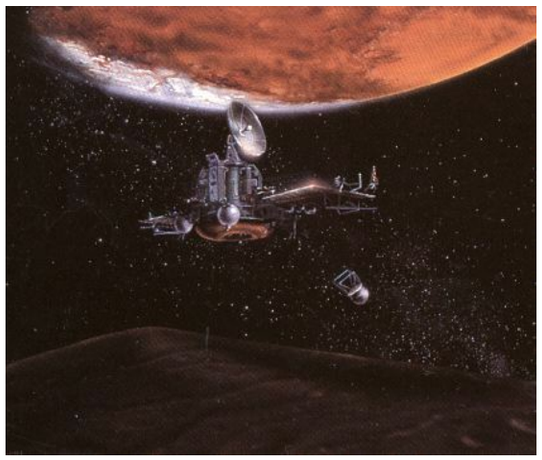




Human factors lead to worst unintentional nuclear disaster in history, hundreds killed and thousands irradiated.

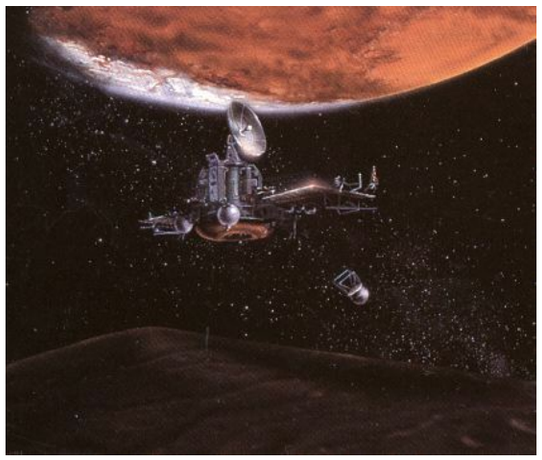
-A systems problem, but not a software system problem
-A test was to be carried out which ultimately require the reactor to be powered-down to a certain level
-Powering-down procedures started, but were unexpectedly delayed when another power plant failed and Chernobyl needed to pick up the slack.
-For nuclear engineering reasons, this delay ultimately made it so the reactor powered-down too quickly later, lower than what the experiment allowed.
-Engineers chose to power the reactor back up (EXTREMELY DANGEROUS)
-Engineers overrode just about every control system
-Final straw was a tiny design flaw that may have made the reactor resemble a nuclear bomb
-Human factors are a V&V concern. Overall system is as strong as its weakest link, and humans can be the weak link.

1988: Phobos 1



-Probe intended to explore Mars
-A command was sent to it with a missing hyphen (yes, a typo)
-Commands were supposed to be proofread by a computer before being sent, but the computer was malfunctioning so someone overrode it.
-Somehow, this invalid command was interpreted as end-of-mission: power down all systems, including radios

1988: Phobos 1



Typo in a command and ignored protocol leads to $300 million loss.

-Probe intended to explore Mars
-A command was sent to it with a missing hyphen (yes, a typo)
-Commands were supposed to be proofread by a computer before being sent, but the computer was malfunctioning so someone overrode it.
-Somehow, this invalid command was interpreted as end-of-mission: power down all systems, including radios

1996: Ariane 5 Rocket

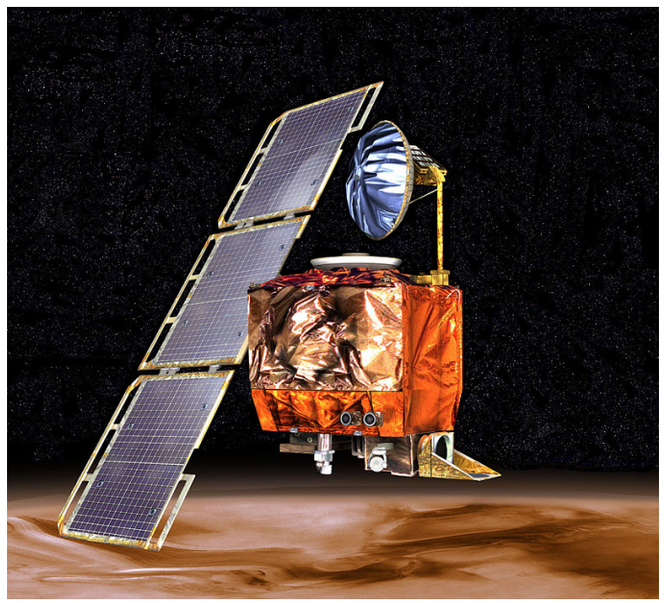
-Cast of a 64-bit integer into a 16 bit integer triggers overflow
-Overflow causes guidance system to think the rocket is 90 degrees off expected trajectory - commands gimbals to make the hardest “correction” physically possible
-Rocket immediately loses control and tears to bits under the force

1996: Ariane 5 Rocket

Integer overflow costs $7 billion dollars.

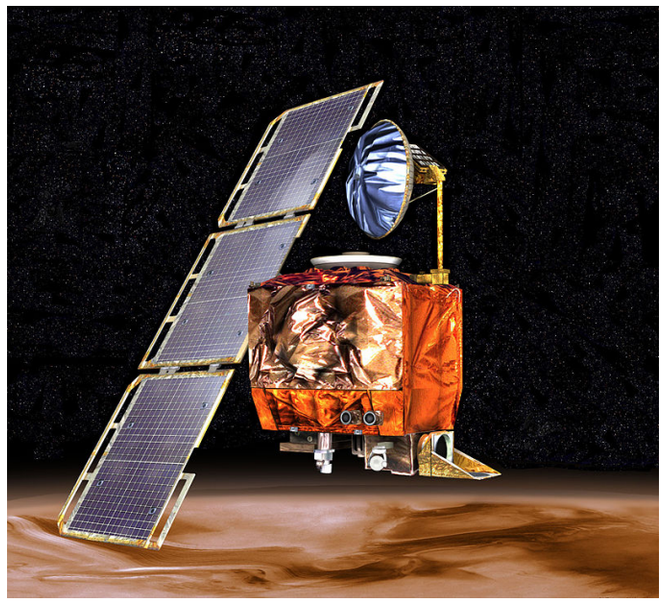
-Cast of a 64-bit integer into a 16 bit integer triggers overflow
-Overflow causes guidance system to think the rocket is 90 degrees off expected trajectory - commands gimbals to make the hardest “correction” physically possible
-Rocket immediately loses control and tears to bits under the force

1999: Mars Climate Orbiter



-Two systems communicate with each other
-One speaks in Newtons (metric), another in pounds-force (imperial)
-“Speak” by passing integers
-Causes it to approach Mars too closely, and is destroyed by atmospheric stresses

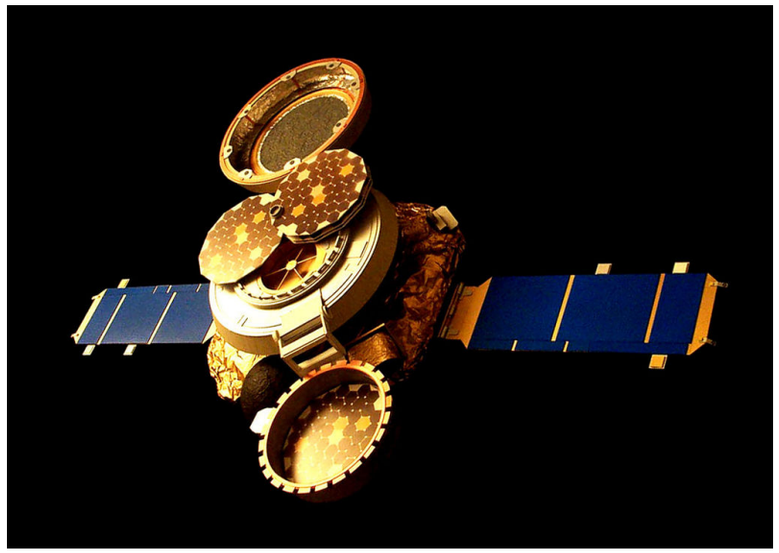
1999: Mars Climate Orbiter

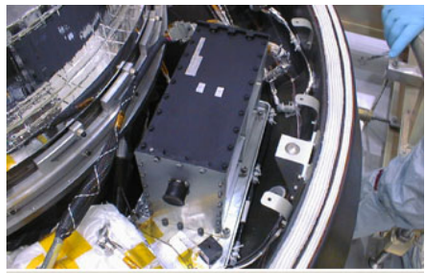


Incompatible unit measurements cause ~$328 million loss.

-Two systems communicate with each other
-One speaks in Newtons (metric), another in pounds-force (imperial)
-“Speak” by passing integers
-Causes it to approach Mars too closely, and is destroyed by atmospheric stresses

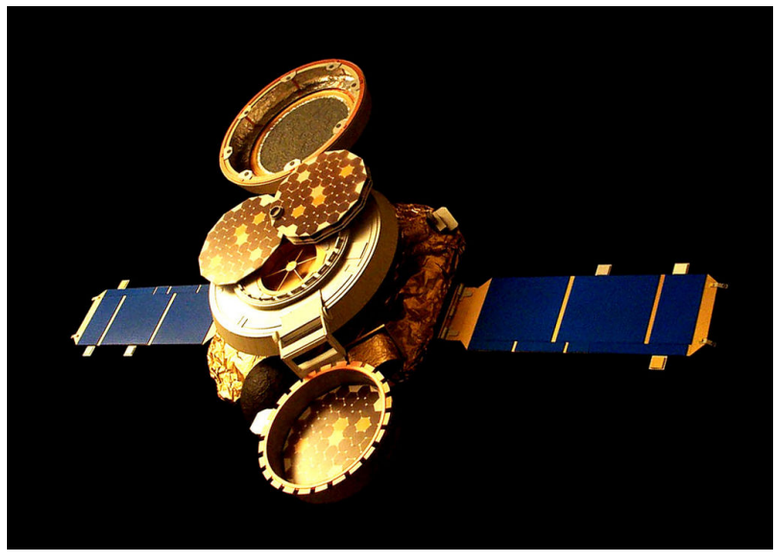
2004: Genesis Probe

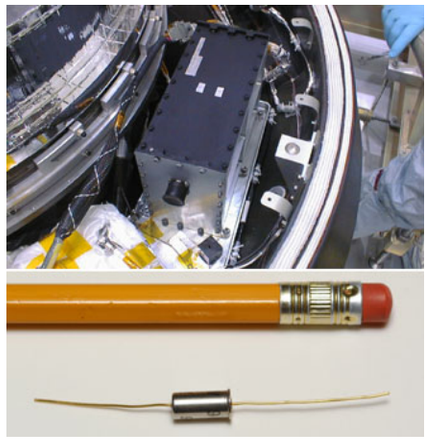




-Intended to collect solar wind particles in space, and send them safely back to Earth
-Launched in 2001, returned 2004
-Parachutes failed to deploy on Earth atmosphere reentry, leading to the partial destruction of some samples
-Root cause: accelerometer installed backwards

2004: Genesis Probe





Accelerometer installed backwards leads to three year loss.
Test which would have caught it was skipped.

-Intended to collect solar wind particles in space, and send them safely back to Earth
-Launched in 2001, returned 2004
-Parachutes failed to deploy on Earth atmosphere reentry, leading to the partial destruction of some samples
-Root cause: accelerometer installed backwards
-The test that would have caught it was skipped because it was too time-consuming (because three year losses aren't long).

2008: S3 Outage

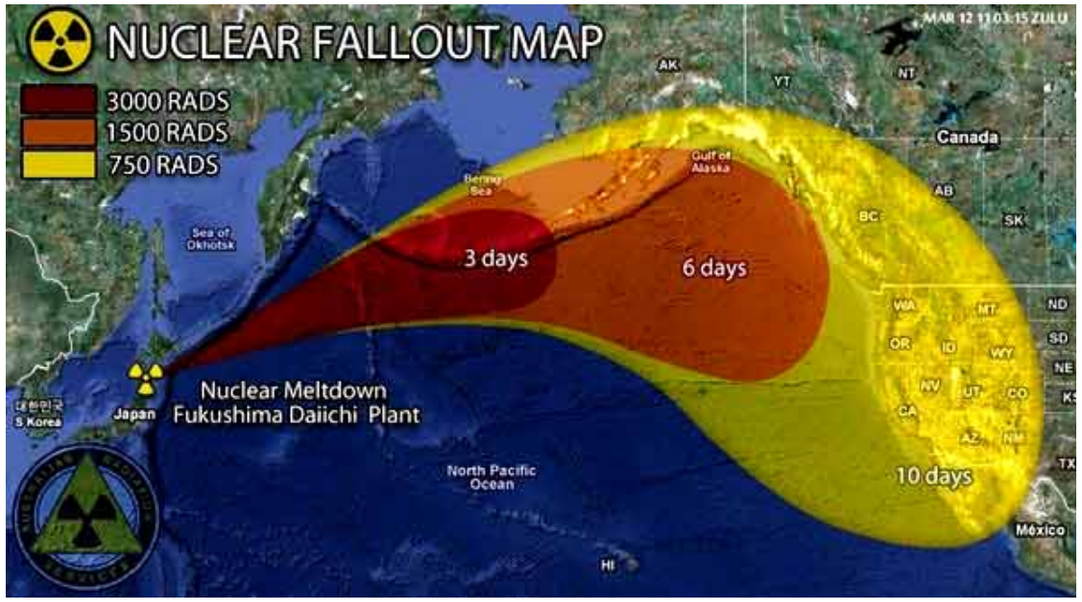
-Message with a corrupted bit sent
-Checksum collision makes message appear valid to fast checks
-Systems which receive the message hang badly as they try to interpret it, and re-send the message in the process
-Effectively required turning off all of S3 and turning it back on.
-Six hours, likely tens of millions lost.

2008: S3 Outage

Unanticipated ultra-low probability event results in 6 hour outage and likely tens of millions of dollars lost.

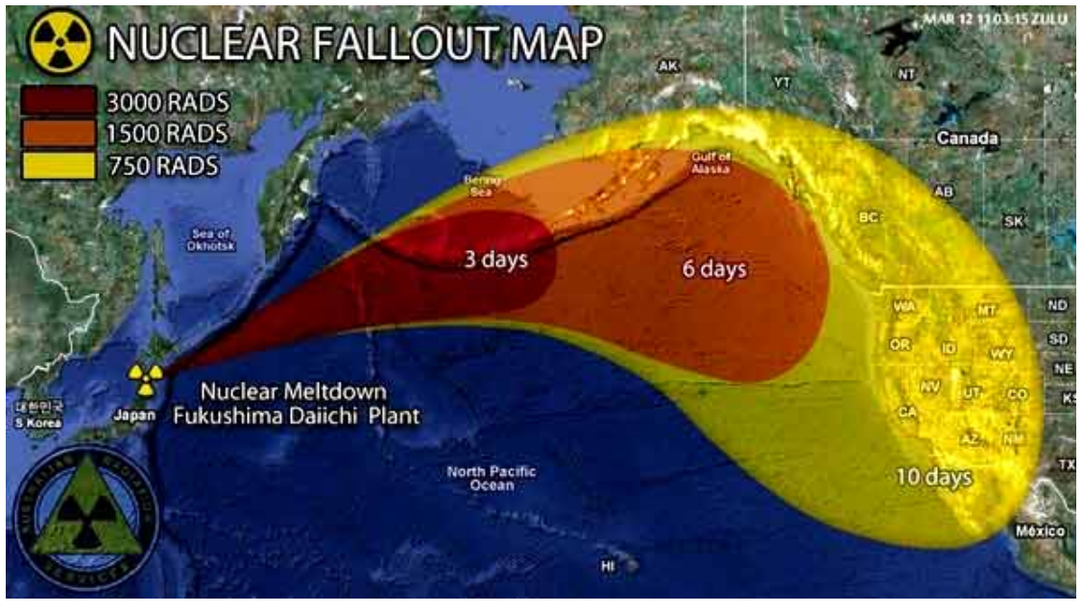
-Message with a corrupted bit sent
-Checksum collision makes message appear valid to fast checks
-Systems which receive the message hang badly as they try to interpret it, and re-send the message in the process
-Effectively required turning off all of S3 and turning it back on.
-Six hours, likely tens of millions lost.

2011: Fukushima



-Tsunami knocks out primary power to reactor cooling system
-Reactor powered down, but reactor design requires steady coolant flow for multiple days afterward before it is safe.
-Backup diesel generators providing emergency power for this purpose were located underground, and flooded when the tsunami hit
-This design flaw was known and left uncorrected

2011: Fukushima



Unexpected scenario + design flaw = second worst accidental nuclear incident in history.

-Tsunami knocks out primary power to reactor cooling system
-Reactor powered down, but reactor design requires steady coolant flow for multiple days afterward before it is safe.
-Backup diesel generators providing emergency power for this purpose were located underground, and flooded when the tsunami hit
-This design flaw was known and left uncorrected

2012: Knight Capital Group

-Has eight systems running custom software doing stock trading.
-Software update occurs, introducing new input. One system accidentally isn't updated.
-The new input just happened to be valid for code that hadn't been operational for 8 years, but was still part of the codebase.
-Stock trading begins. New inputs are received. System which failed to be updated interprets new input as "Buy as fast as possible."
-460 million is lost in 45 minutes, and the company goes bankrupt.

2012: Knight Capital Group

Dead code and lack of update procedures lead to $460 million losses and bankruptcy.

-Has eight systems running custom software doing stock trading.
-Software update occurs, introducing new input. One system accidentally isn't updated.
-The new input just happened to be valid for code that hadn't been operational for 8 years, but was still part of the codebase.
-Stock trading begins. New inputs are received. System which failed to be updated interprets new input as "Buy as fast as possible."
-460 million is lost in 45 minutes, and the company goes bankrupt.

2014: Apple's `goto fail`

```
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
  goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
  goto fail;
  goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
  goto fail;
...
err = sslRawVerify(...);

fail:
return err;
```

-What was likely a copy/paste error means that TLS certificates aren't checked
-This enables a "man in the middle" attack, where you can impersonate someone else. The connection is no longer secure, as third parties can read everything sent/received.

2014: Apple's `goto fail`

```
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
  goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
  goto fail;
  goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
  goto fail;
...
err = sslRawVerify(...);

fail:
return err;
```

Code typo + a lack of testability = silent loss of encryption.

-What was likely a copy/paste error means that TLS certificates aren't checked
-This enables a "man in the middle" attack, where you can impersonate someone else. The connection is no longer secure, as third parties can read everything sent/received.
-This was pulled from code which was difficult to test, so the solution was to just not test it.

2017: Another S3 Outage

-Engineer runs command to intentionally bring down a small part of S3
-Typo in the command brings down a significant portion of S3
-Takes three hours to bring it back online

2017: Another S3 Outage

Typo in command results in three hour outage and $150 million lost.

-Engineer runs command to intentionally bring down a small part of S3
-Typo in the command brings down a significant portion of S3
-Takes three hours to bring it back online

2017: Equifax Breach

-Equifax failed to patch its web servers (running Apache) for two months, and an attacker entered a lower-level system through a relevant exploit.
-Now inside, attacker discovered a database secured with "admin/admin". It contained all sorts of financial information, including account information, credit cards, and social security information
-Equifax had the nerve to blame Apache. No one learned anything.

2017: Equifax Breach

Poor system administration practices lead to what may be the biggest financial breach in history.

-Equifax failed to patch its web servers (running Apache) for two months, and an attacker entered a lower-level system through a relevant exploit.
-Now inside, attacker discovered a database secured with "admin/admin". It contained all sorts of financial information, including account information, credit cards, and social security information
-Equifax had the nerve to blame Apache. No one learned anything.

2018: Boeing 737 MAX

-Want to mount an engine that's too big for an airframe
-Making a new airframe is time-consuming (10 years) and expensive (billions)
-Solution: (try to) fix a massive hardware problem in software
-Software gets data from unreliable sensors which aren't very fault-tolerant
-Pilots are not sufficiently informed of this, nor properly trained
-Under the right conditions, the software puts the plane into a nosedive
-This software should NEVER have been written - the software engineers should have refused

2018: Boeing 737 MAX

Safety concerns made secondary to economic concerns; hundreds dead and billions in damages.

-Want to mount an engine that's too big for an airframe
-Making a new airframe is time-consuming (10 years) and expensive (billions)
-Solution: (try to) fix a massive hardware problem in software
-Software gets data from unreliable sensors which aren't very fault-tolerant
-Pilots are not sufficiently informed of this, nor properly trained
-Under the right conditions, the software puts the plane into a nosedive
-This software should NEVER have been written - the software engineers should have refused

2019: Uber Self-Driving SUV

-Hits and kills a pedestrian who jaywalked in front of it
-System incorrectly identified pedestrian as a bicycle, and incorrectly estimated the path the person was walking in
-Direct quote of incident report: "The system design did not include a consideration for jaywalking pedestrians"
-Had a failsafe which would apply brakes in case of imminent collision, but it had been disabled

2019: Uber Self-Driving SUV

Lack of proper testing and disabled failsafe kills pedestrian.

-Hits and kills a pedestrian who jaywalked in front of it
-System incorrectly identified pedestrian as a bicycle, and incorrectly estimated the path the person was walking in
-Direct quote of incident report: "The system design did not include a consideration for jaywalking pedestrians"
-Had a failsafe which would apply brakes in case of imminent collision, but it had been disabled

The Point

- Buggy systems and software are pervasive
- These flaws have real impact

Why?

- Incomplete knowledge
- Cultural problems
- Correctness secondary to features
- Save a life: be a pessimist

-Incomplete knowledge: we might not know all of our requirements, and this may even be the common case. Low-probability events may not even be on our radar.
-Cultural problems: if it's ok (or frikken encouraged) to have incorrect software, then you'll have incorrect software
-Correctness secondary: related to the previous point. Many V&V activities slow things down in the short-term, making them unfit for deadline-driven development.
-Pessimism can be good: thinking in terms of how things can go wrong, and preparing for it. Cynicism, however, is pessimism with no action.

What Can We Do?

- Establish and follow good coding practices
- Test that our systems operate as expected
- *Prove* that our systems operate as expected

~~What Can We Do?~~

What Is this Class?

- Establish and follow good coding practices
- Test that our systems operate as expected
- *Prove* that our systems operate as expected

Class Structure, Project, and Syllabus