

# CS777

---

## Forensics

Dr Olly Gotel

[ogotel@pace.edu](mailto:ogotel@pace.edu)

<http://csis.pace.edu/~ogotel/cs777>



# Today's Agenda

---

- Why does software fail?
- Classic examples from the software engineering hall of shame
- Over to you ... case studies in engineering failure ... & the lessons we learn from them
- Movie

# What Have all These Got in Common?



# All Areas of Engineering Failure

---

- Engineering is all about creating cost-effective solutions to practical problems, by applying scientific knowledge to building **things** in the service of mankind

**Software things**

- Good engineering practices are designed to prevent accidents – especially those that cause loss of life!
- We improve our practices by studying & learning from:
  - good examples
  - things that go wrong!



# Normal Accidents

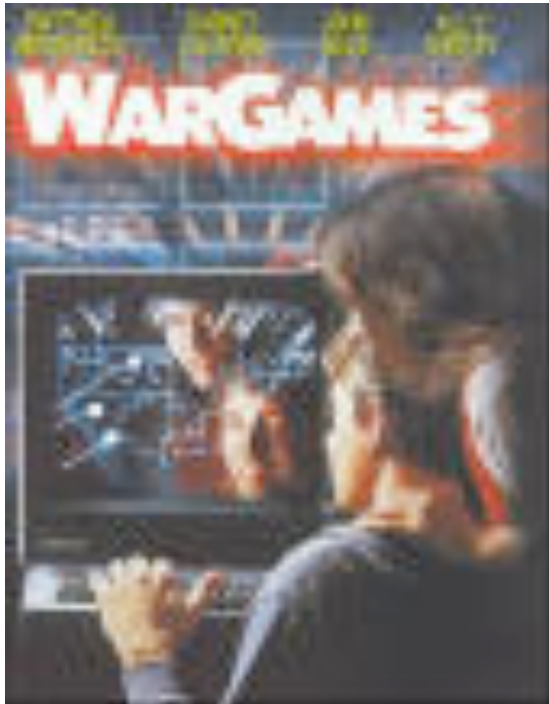
---

- Perrow's "Normal Accident Theory":
  - accidents can be normal, but lead to catastrophes
  - particularly true in systems with high interactive complexity
  - systems fail *systemically*
  - the answers to such failures are also systemic!





“The quality of the systems we develop increasingly determines the quality of our existence” [Van Vliet]



The world is in danger!



YOU are in danger!

## From a Dutch Newspaper (1980)

---

- “For a short period last Tuesday the United States brought their atomic bombers & nuclear missiles to an increased state of alarm when, because of a computer error, a false alarm indicated that the Soviet Union had started a missile attack.”
- “For the second time within a few days, a deranged computer reported that the Soviet Union had started a nuclear attack against the United States. Last Saturday, the DoD affirmed the false message, which resulted in the engines of the planes of the strategic air force being started up.”

[Reported in Van Vliet]

## From a Computer Magazine (1983)

---

- “The court in Dusseldorf has discharged a woman (54), who was on trial for murdering her daughter. An erroneous message from a computerized system made the insurance company inform her that she was seriously ill. She was said to suffer an incurable form of syphilis. Moreover, she was said to have infected both her children. In panic, she strangled her 15 year old daughter and tried to kill her 13 year old son and herself. The boy escaped, and with some help he enlisted prevented the woman from dying of an overdose. The judge blamed the computer error and considered the woman not responsible for her actions.”

[Reported in Van Vliet]



## More Light-Hearted (or not?)

---

- “A 51-year-old woman was subjected to a harrowing two-hour ordeal [on 16 Apr 2001] when she was imprisoned in a hi-tech public convenience. Maureen Shotton, from Whitley Bay, was captured by the maverick cyberloo during a shopping trip to Newcastle-upon-Tyne. The toilet, which boasts state-of-the-art electronic auto-flush and door sensors, steadfastly refused to release Maureen, and further resisted attempts by passers-by to force the door. Maureen was finally liberated when the fire brigade ripped the roof off the cantankerous crapper. Maureen's terrifying experience confirms that it is a short step from belligerent bogs to Terminator-style cyborgs hunting down and exterminating mankind.”

<http://catless.ncl.ac.uk/Risks/>

# Systems Failure

---

- System components fail for many reasons:
  - parts wear out
  - screws come loose
  - circuits get fried
  - components used to do things they were not designed to do

Point failure

...

- These rarely lead to catastrophe:
  - back-ups & redundancy
  - fault tolerance
  - certification
  - checks and balances

Engineering practice

...

# Reasons For Failure

---

- Can *generally* trace failures to a single root cause, but many systemic reasons for failure
- Humans make mistakes, but good engineering practices (especially a system of thorough testing & validation), are designed to catch these mistakes
- Failure is generally a result of failure in engineering & in its management – so this is what YOU have to learn to avoid!

False safety in numbers

False confidence

“Its worked up to now...”

# Software Failure

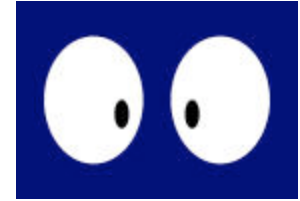
---

- Software systems have components that:
  - are invisible, intangible & abstract
  - make little sense in isolation
  - have no limits on complexity & not continuous
  - have no laws governing behaviour
  - don't wear out
  - replicate perfectly
- Software reliability is determined by manifestation of errors already present, not wear & tear
- Needs its own system of engineering practices

# Errors in Software Development

---

- Programming errors – programs don't meet specifications)
- Specification errors – specifications simply plain wrong
- Requirements errors – solving the wrong problems
- Requirements change – solving yesterday's problems



# How to Look at Failure?

---

- This is NOT a negative orientation
- Our aim in engineering software should not be the absence of bugs (nice but impractical?), it should be the demonstrable presence of **quality** (a quality product engineered using a quality process)
- What is quality? Some say conformance to user requirements is the key issue
- Positive orientation - look for the presence of **QUALITY** in those systems and projects that do work

Start building YOUR checklist of things to look for!



# Classic Case Studies

---

- **Self-destruct of Ariane-5, Flight 501 in 1996**
- Space Shuttle Challenger Accident (Flight STS51-L) in 1986
- Loss of NASA's Mars Polar Lander and Deep Space 2 Mission in 1999
- Loss of NASA's Mars Climate Orbiter in 1999
- Denver International Airport Baggage Handling System Fiasco (1989-1995)
- Therac-25 Accidents (1985-1987)
- AT&T Network Crash of 1990
- Abandonment of the London Stock Exchange Taurus Trading System in 1993
- Patriot Missile Failure of 1991
- Failure of the Aegis System on U.S.S. Vincennes in 1988
- Failure of the Titan IV B-32 Mission in 1999
- Various Airbus Disasters (over the years)

## Remember Your Task?

---

- To do some software forensics and find out about the failure/incident you were allocated
- Discuss:
  - Summary of the incident – what happened?
  - Cause(s) of failure & events leading up to failure?  
In what way was the system unreliable? In what way does this example show good or poor quality practices?
  - What would be an acceptable failure rate for such a system and how would you determine and specify this?
  - Broader lesson(s) from the failure?

# Student Work

---

- See the zip folder for slides on some of the studies done by the students...

# 1<sup>st</sup> Set of Incidents - Synopsis

Factor	STS 51L	Ariane 501	Path- finder	MCO	MPL	DS-2
Didn't test to spec		●		●	●	?
Insufficient test data	●	●			●	●
Tested "wrong" system					●	
No regression test					●	
Lack of integration testing		●		●		●
Lack of expertise at inspections		●		●	●	
System changed after testing					●	?
Requirement not implemented		?		●	●	
Lack of diagnostic data during operation			●	●	●	●
S/W used before ready				?	?	●
Different team maintains software				●	●	
Didn't use problem reporting system	●		●	●	●	?
Didn't track problems properly	●	●	●	●	●	?
Didn't investigate anomalies	●		●	●		
Poor communication between teams	●	●	●	●	●	?
Insufficient staffing	●			●	●	●
Failure to adjust budget and schedule	●			●	●	●
Inexperienced managers	?			●	●	●
Commercial pressures took priority	●	●		●	●	●
Reused code w/o checking assumptions		●				
'Redundant' design not really redundant	●	●				

# Software Runaways - Lessons

---

- Generally huge
- Generally multiple causes
- Generally lauded as “breakthroughs” in early days
- Technology just as often a cause of failure as management:
  - use of new technology
  - use of latest software engineering concepts
  - complexity got out of hand
  - integrating too many new technologies
  - performance issues (i.e. too slow to be useful)

The findings from [Glass 1998]



# Cobb's Paradox

---

- “We know why projects fail, we know how to prevent their failure -- so why do they still fail?”

*Martin Cobb*

*Treasury Board of Canada Secretariat*

*Ottawa, Canada*

- Reasons:
  - SE is hard
  - SE is multi-disciplinary & requires lots of skills
  - perception that discipline in SE is too costly
  - people still think SE is not that important
  - Quality costs!



**Incidents like this don't happen often in the software field – but when they do, they are very VISIBLE**

## General Lessons

---

- Systems fail *systemically*
- In real applications there is a collision of social, human & technical systems
- Good software engineering is **not** a luxury:
  - if you fail to use known good practice, expect to answer to a Public Inquiry
  - software professionals are likely to require a licence in the future ... get a jump start
  - it is not acceptable for you not to know the quality and reliability of your software systems!!!



## Key Points

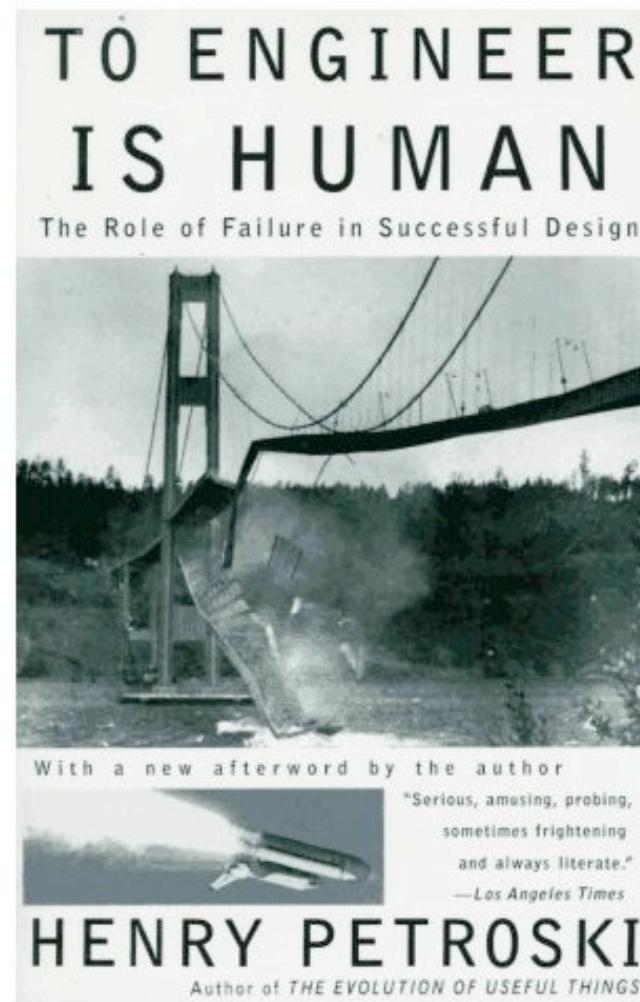
---

- Developing reliable software systems depends, not so much on writing perfect code, but on how good we are at:
  - sharing information between people (communication)
  - identifying risks early & tracking these
  - questioning all assumptions
  - tracking & discharging problem reports
  - managing (resources & risks)
  - following engineering best practice
  - testing everything we do...systematically
  - BUT... building quality in! Not testing it in!

Depends  
on smart  
&  
informed  
people ...  
i.e. YOU!

# Movie Time...

---



## Some Additional Useful Resources

---

- “To Engineer is Human” by Henry Petroski
- “Software Runaways” by Robert Glass
- “Computer-Related Risks” by Peter Neumann
- “Safeware” by Nancy Leveson
- “Normal Accidents” by Charles Perrow