Introduction: Software Failures and Crisis

ECSC409: Software Engineering Principles

1

2

Tutorials

- Check the group you have been allocated to
- Check the time table and find out your tutorial slot
- Please stick with your tutorial slot as you will be doing an in-class project
- Lack of attendance in tutorials will severely obstruct the progress with your in-class project and, therefore, will affect your mark in this assessment as well as those marks for the two course works
- Course works will rely on your in-class projects

3

Module Information

- Outline available on Blackboard
- Assessment pattern (available on Blackboard)

  30 / 30, 35 / 30, 35 / 30 for the in-class project and the two course works, respectively

- Roughly, *we will be learning SE principles and how you can apply them for designing and evaluating flexible, scalable and effective, large scale software systems by harnessing complexity*

- *The two in-class projects and course works will address DESIGN and EVALUATION, respectively*

4

Software Failures and Crisis

ECSC409: Software Engineering Principles

5

Computing in crisis?

- Long standing debate whether Computing is engineering or science
- Engineering view dominated for four decades due to building reliable computers, networks, complex software
- Three waves of attempts to come up with a unifying view (see next slide)
- In the mid-1980s, the science view gained momentum with the computational science movement (computation a new sub-paradigm of science) – *Concerns raised about lack of common definition in the field*

6

Computing in crisis?

- Three waves of attempts along the way for a unique and unifying view:
  - First wave by Alan Perlis, Allen Newell, Herb Simon: Computing unique among all sciences and engineering in its study of information processes (science of artificial, *Herb Simon*)
  - Second by Edsger Dijkstra and Donald Knuth in the late 1960's: Focus on programming (a coder) as art of designing information processes and unifying theme
  - Third wave led by Bruce Arden in the mid-1970's: Computing as automation

7

Computing out of the crisis?

- Mid-1980's: Computing is a unique combination of the traditional paradigms of math, science and engineering (see also table on the next slide) – *Report by the ACM Education Board*
- All three made substantial contributions but none tells the whole story
- Programming as a practice crossing all three paradigms was essential but did not fully portray the depth and richness of the field

8

Sub-paradigms embedded in Computing

	Math	Science	Engineering
Initiation	Characterize objects of study (definition)	Observe a possible recurrence of pattern of phenomena (hypothesis)	Create statements about desired system actions and responses (requirements)
Conceptualisation	Hypothesise possible relationships among objects (theorem)	Construct a model that explains the observation and enables predictions (model)	Create formal statements of system functions and interactions (specifications)
Realisation	Deduce which relationships are true (proof)	Perform experiments and collect data validate)	Design and implement prototypes (design)
Evaluation	Interpret results	Interpret results	Test the prototypes
Action	Act on results (apply)	Act on results (predict)	Act on results (build)

9

Computing out of the crisis?

- Around 1997: IT (Information Technology) could reconcile the three parts under a single umbrella unique to computing
- Time has proved us wrong…
- IT connotes technological infrastructure but not the core technical aspects of computing

10

Computing out of the crisis?

Recent thinking: Computing is not a blend of three sub-paradigms but a computing paradigm on its own…

- Shifting attention from computing machines to information processes, including natural information processes such as DNA transcription
- Interpretation of computing through the seven dimensions of computation, communication, coordination, recollection, automation, evaluation, design
- See also http://www.greatprinciples.org
- Strong argument for computing as the fourth great domain of science alongside physical, life and social science

11

The Computing paradigm

	Computing
Initiation	Determine if the system to be built (or observed) can be represented by information processes, either finite (terminating) or infinite (continuing interactive)
Conceptualisation	Design (or discover) a computational model (algorithms or set of computational agents) that generates the system's behaviour
Realisation	Implement design processes in a medium capable of executing its instructions. Design simulations and models of discovered processes. Observe behaviours of information processes.
Evaluation	Test the implementation for logical correctness, consistency with hypotheses performance constraints and meeting original goals. Evolve the realisation as needed.
Action	Put the results to action in the world. Monitor for continued evaluation.

12

A New Computing Paradigm?

- No longer just about algorithms, data structures, numerical methods, programming languages, operating systems, networks
- Includes exciting new subjects such as Web Science, Mobile Computing, Cyberspace protection, Information Visualisation, et al
- *Discovery* and search tools is as important as *construction* or *design*
- *Discovery and design are closely linked*
- *Behaviour of many large designed systems (e.g., Web) is discovered by observation*

13

Central focus of the new computing paradigm

- *Information processes – natural or constructed processes that transform information*
- *These can be discrete or continuous*

14

Software Engineering in Crisis?

- All along the Computing crisis (1989: Are we all scientists, mathematicians or engineers?)
- Software Engineering as Engineering remains disputed
- Software Engineering crisis due to lack of a rigorous engineering process for the design and implementation of software (according to the founders of the software engineering field, legendary 1968 NATO conference)

15

Definition of the Engineering Process

The dictionary defines engineering as the application of scientific and mathematical principles to achieve the design, manufacture, and operation of efficient and economical structures, machines, processes, and systems.

So what is SE? Late us take the view of a leading expert…

- IBM Fellow, Grady Booch
- http://www.youtube.com/watch?v=doHVjoTD_ro

Software Engineering in a Nutshell

- the "engineering process" consists of a repeated cycle through requirements, specifications, prototypes, and testing
- the process models have evolved into several forms that range from highly structured preplanning (waterfalls, spirals) to relatively unstructured agile (XP, SCRUM, Crystal, and evolutionary).
- No one process is best for every problem.

However, ….

- Despite long experience with these processes, none consistently delivers reliable, dependable, and affordable software systems.
- Approximately one third of software projects fail to deliver anything, and another third deliver something workable but not satisfactory.
- Often, even successful projects took longer than expected and had significant cost overruns.
- Large systems, which rely on careful preplanning, are routinely obsolescent by the time of delivery years after the design started.
- Faithful following of a process, by itself, is not enough to achieve the results sought by engineering.

19

Scale of the problem

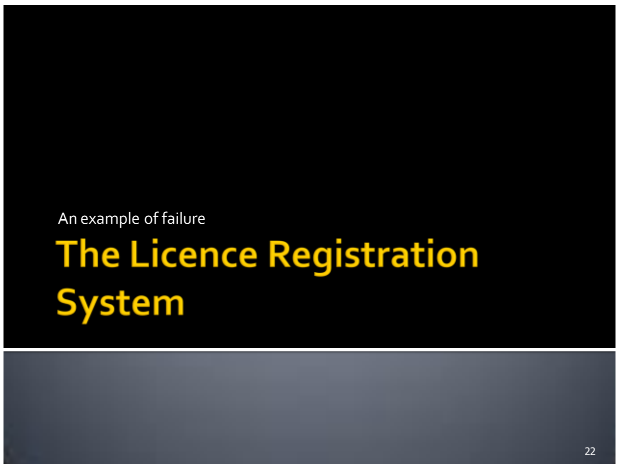
- Quoted from CIO Magazine Dec 1998:
  - *Recent Standish Group survey indicates "46% of IT projects were over budget and overdue and 28 % failed altogether"*
  - *"only 24 % of IT projects undertaken by Fortune 500 companies in 1998 will be completed successfully"*
- From 1994 Standish report:
  - *91% of projects at large companies failed, 30% of projects at large companies were eventually cancelled*

20

Scale of the problem (2)

- A source in the US military claims:
  - There is one Army project that is 1 billion dollars over budget, and still has no working system.
  - 100% of all DoD projects over 1 million lines of code are delayed.
  - One third of all projects are cancelled before completion.
  - One half of all projects cost twice as much as originally estimated.
  - Documentation of a 350 KLOC DoD project costs about four hundred dollars per page.

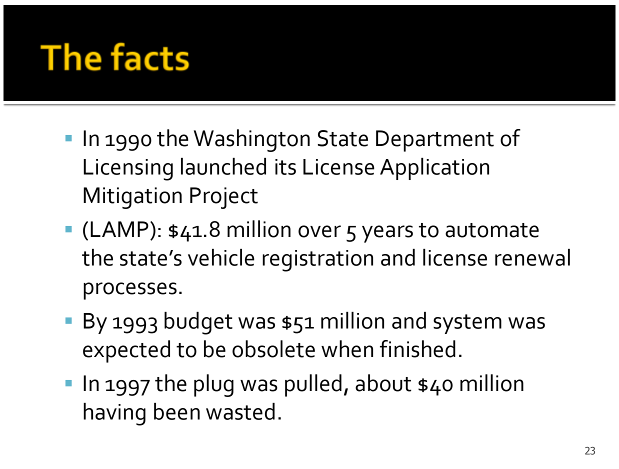
21



An example of failure

The Licence Registration System

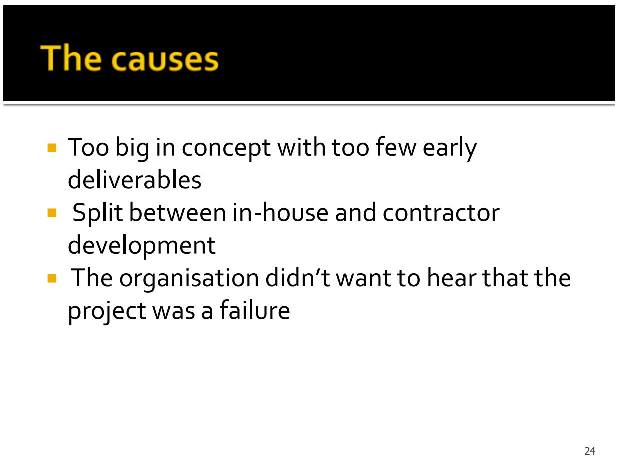
22



The facts

- In 1990 the Washington State Department of Licensing launched its License Application Mitigation Project
- (LAMP): $41.8 million over 5 years to automate the state's vehicle registration and license renewal processes.
- By 1993 budget was $51 million and system was expected to be obsolete when finished.
- In 1997 the plug was pulled, about $40 million having been wasted.

23



The causes

- Too big in concept with too few early deliverables
- Split between in-house and contractor development
- The organisation didn't want to hear that the project was a failure

24

An example of failure

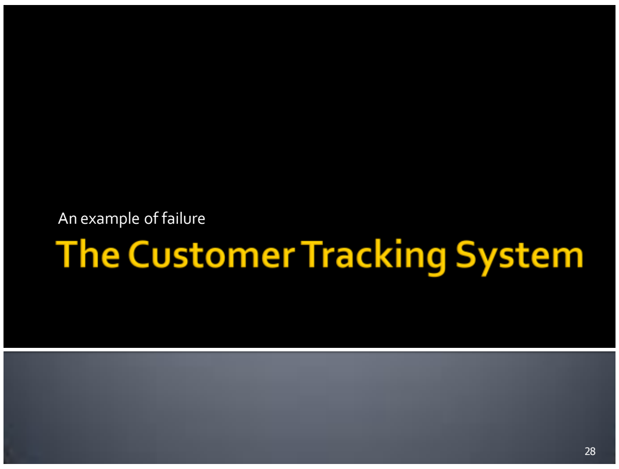
The Customer Database System

The facts

- In 1996 a US consumer group embarked on an 18-month, $1 million project to replace its customer database.
- The new system was delivered on time but didn't work as promised, handling routine transactions smoothly but tripping over more complex ones.
- Within three weeks the database was shut down, transactions were processed by hand and a new team was brought in to rebuild the system.

The causes

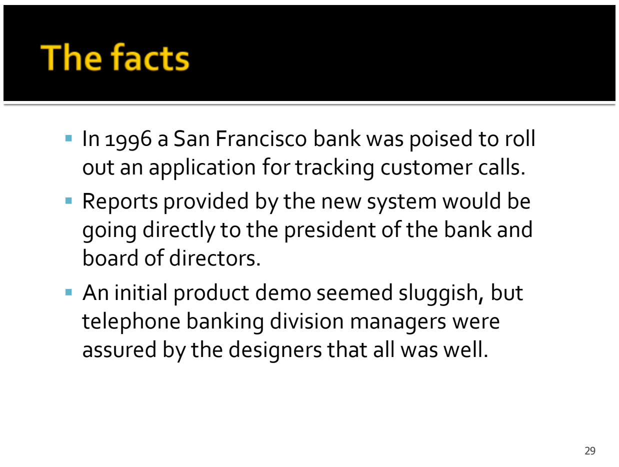
- The design team was over-optimistic in agreeing to requirements
- Developers became fixated on deadlines, ignoring errors



An example of failure

The Customer Tracking System

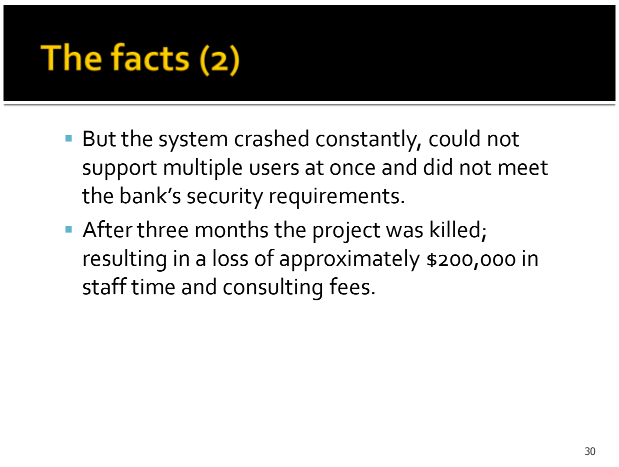
28



The facts

- In 1996 a San Francisco bank was poised to roll out an application for tracking customer calls.
- Reports provided by the new system would be going directly to the president of the bank and board of directors.
- An initial product demo seemed sluggish, but telephone banking division managers were assured by the designers that all was well.

29



The facts (2)

- But the system crashed constantly, could not support multiple users at once and did not meet the bank's security requirements.
- After three months the project was killed; resulting in a loss of approximately $200,000 in staff time and consulting fees.

30

The causes

- The bank failed to check the quality of its contractors
- Complicated reporting structure with no clear chain of command
- Nobody "owned" the software

31

An example of failure

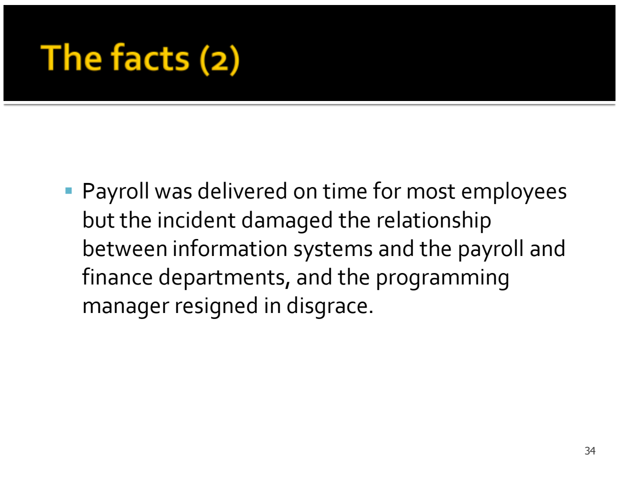
The Payroll System

32

The facts

- The night before the launch of a new payroll system in a major US health-care organization, project managers hit problems.
- During a sample run, the off-the-shelf package began producing cheques for negative amounts for sums larger than the top executive's annual take-home pay, etc.

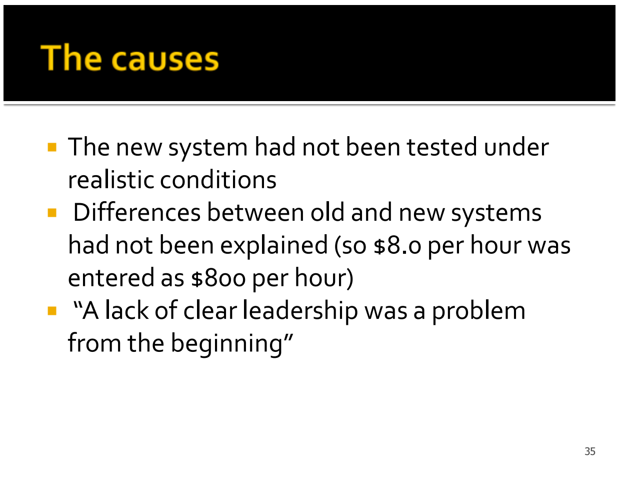
33



The facts (2)

- Payroll was delivered on time for most employees but the incident damaged the relationship between information systems and the payroll and finance departments, and the programming manager resigned in disgrace.

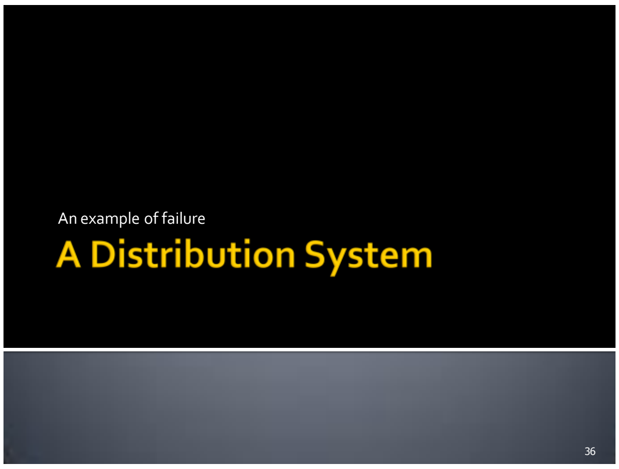
34



The causes

- The new system had not been tested under realistic conditions
- Differences between old and new systems had not been explained (so $8.0 per hour was entered as $800 per hour)
- "A lack of clear leadership was a problem from the beginning"

35



An example of failure

A Distribution System

36

The facts

- Anticipating growth, a $100 million division of a $740 million manufacturing business earmarks $5 million for a new distribution and customer service system to replace its old one.
- The project is to take a year and a half to complete. Two years later, the CIO is sacked and a new executive brought in to save the project. Three months later, the system breaks down altogether.

37

The facts (2)

Nine months later, the CIO approached his boss, the CEO to tell him the project is a failure. "It was kind of like telling him a relative had died," he recalls. "First he denied it, then he went through a grieving process, then he accepted it. It was just so much money for a division that size to wave in the wind."

38

The causes

- Wrong direction from the start
- Inadequate software plan
- Nobody "owned" the software

39

From the Software Hall of Shame

R. N. Charette (September 2005). Why Software Fails. IEEE Spectrum. [DOI link]

40

Critical Failure Factors

- Warning signs of a project doomed to failure, or even disaster:
  - Organization: hostile culture, poor reporting structures
  - Management: over-commitment, political pressures
  - Conduct of the project:

    Initiation phase: technology focused, lure of leading edge, complexity underestimated

41

Critical Failure Factors (2)

- Conduct of the project (continued):
  - Analysis and design phase: poor consultation, design by committee, technical fix for management problem, poor procurement
  - Development phase: Staff turnover, poor competency, poor communication (e.g. split sites)
  - Implementation phase: receding deadlines,
  - inadequate testing, inadequate user training

42

Reasons for software failures attributed to six practices software engineers do not do well….

1. Predictable outcome (*principle of least surprise*)
2. Design metrics, including design to tolerances
3. Failure tolerance (*calculating the risks of all failures…*)
4. Separation of design from implementation
5. Reconciliation of conflicting sources and constraints
6. Adapting to changing environments

Reference: Peter J. Denning, Richard D. Riehle, The profession of IT: Is Software Engineering Engineering? Communications of ACM, April 2009 [DOI Link]

43

Solutions suggested

- Broaden the narrow scope of the "system" incorporating environment and hardware
- Broaden the engineering team according to the roles
  - software architect
  - software engineer
  - programmer
  - project manager
  - systems engineer

44

Recommended Further Essential Reading Available on-line via UoW Library Services

- Software engineering, Ian Sommerville 1951-, 8th ed. Harlow, England ; New York : Addison-Wesley 2007 **Chapter 1: Introduction**
- Software engineering for students, Doug Bell 1944-, 4th ed. Harlow, England ; New York : Addison-Wesley 2005 **PART A: Preliminaries**
- Software Engineering Principles and Practice, Hans van Vliet, John Wiley & Sons 2008 **Chapter 1: Introduction**

45

Suggested articles

- Peter J. Denning, Richard D. Riehle, The profession of IT: Is Software Engineering Engineering? Communications of ACM, April 2009 [DOI Link]
- Peter J. Denning, Richard D. Riehle, The profession of IT: Computing's Paradigm, Communications of ACM, December 2009 [DOI Link]
- R. N. Charette (September 2005). Why Software Fails. IEEE Spectrum. [DOI link]
- R. Bennett (3 May 2007). Chaos as register offices are told to abandon £6m computer system. [Times Online]
- R. Guth (September 2005). Battling Google, Microsoft Changes How It Builds Software. Wall Street Journal

46

Suggested books

- S. Flowers (1996). Software Failure: Management Failure: Amazing Stories and Cautionary Tales. Addison-Wesley.
- R. L. Glass (1998). Software Runaways: Lessons Learned from Massive Software Project Failures. Prentice Hall.

47

NEXT

Architectural Design and Patterns

48