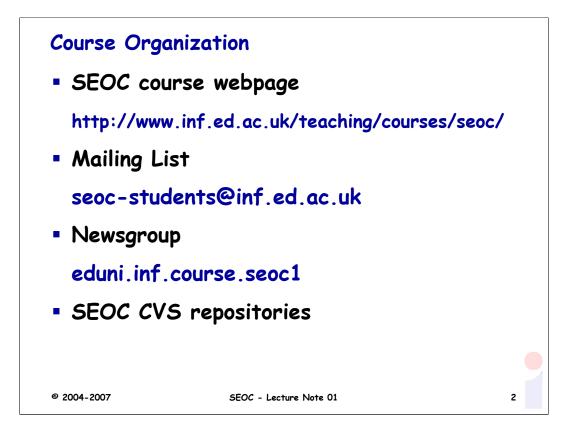
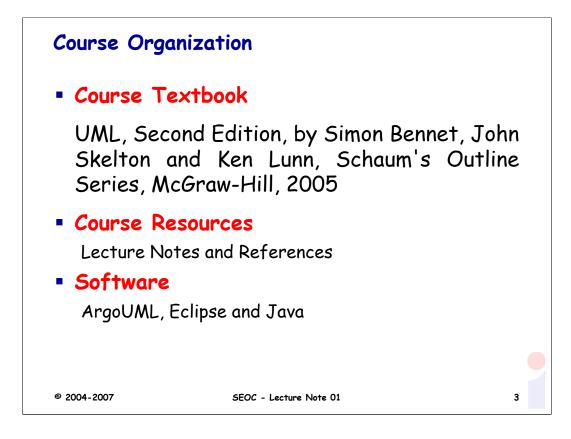
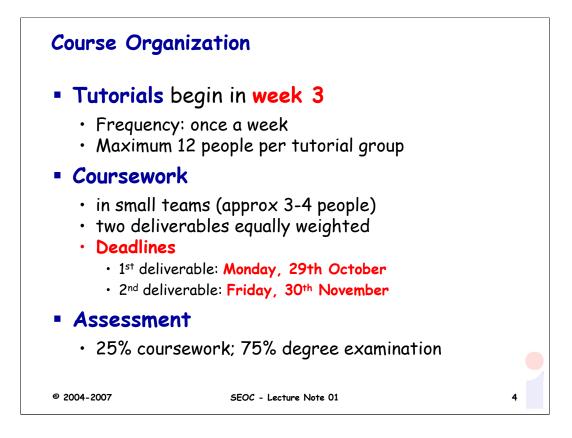
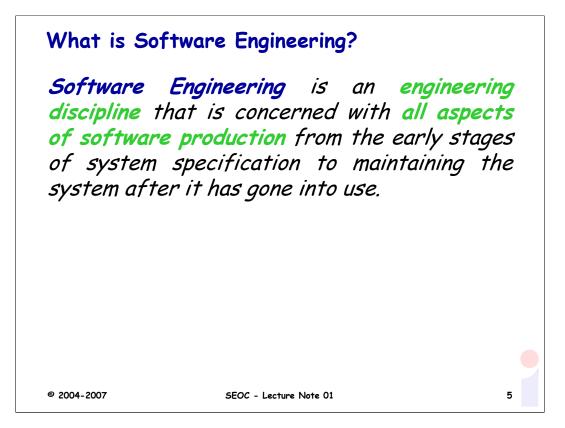
Software Engineering with Objects and Components

Massimo Felici JCMB-1402 0131 650 5899 1BP-G04 0131 650 4408 mfelici@inf.ed.ac.uk









This lecture provides a very brief introduction to Software Engineering. The SEOC course focuses on engineering software systems using Objects and Components. The main learning objectives of the course involve the acquisition of software engineering knowledge and ability to design, assess and implement object-oriented systems. The course uses UML as modelling language. The course organization embeds some general software engineering principles and practices.

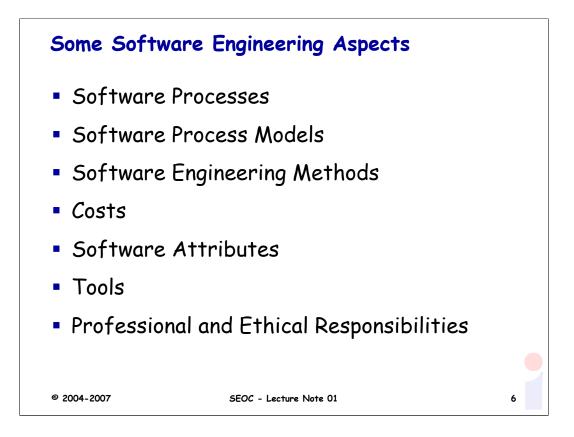
Readings

• B. Meyer. Software Engineering in the Academy. IEEE Computer, May 2001, pp. 28-35. It provides a discussion on software engineering education.

Suggested Readings

For an introduction to various aspects of Software Engineering refer to

- I. Sommerville. Software Engineering, Eighth Edition, Addison-Wesley 2007. In particular, Chapter 1 for a general account of Software Engineering.
- SWEBOK Guide to the Software Engineering Body of Knowledge. 2004 Version, IEEE.



Software Engineering is concerned with all aspects of software production. The main objective is to support software production in order to deliver software that is "fit for purpose", e.g., good enough (functionally, non-functionally), meets constraints (e.g., time and financial) of the environment, law, ethics and work practices. For instance, some software engineering aspects are:

Software Process: the set of activities and associated results (e.g., software specification, software development, software validation and software evolution) that produce a software product. Software essential activities are:

- Software Requirements: gaining an accurate idea of what the users of the system want it to do.
- Software Design: the design of a system to meet the requirements.
- Software Construction: the realisation of the design as a program.
- Software Testing: the process of checking the code meets the design.
- Software Configuration, Operation and Maintenance: major cost in the lifetime of systems.

Software Process Model: An overview of the software activities and results' organization.

Software Engineering processes (e.g., waterfall, spiral, etc.) arrange (deploy effort) these activities differently. The SEOC organization, to a certain extent, embeds some basic principles underlying different software engineering processes.

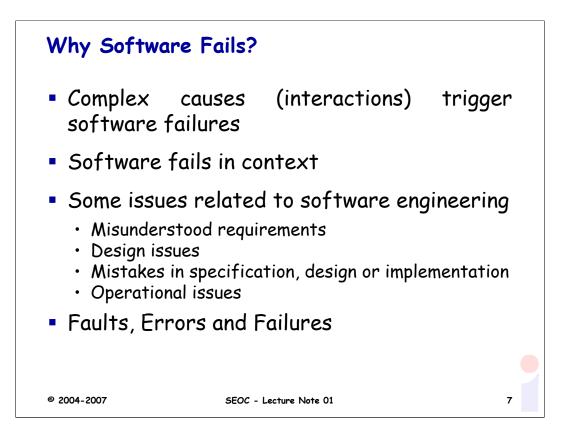
Readings

Among the various process, the Rational Unified Process (RUP) is the most relevant one. It "provides a guide for how to effectively use the Unified Modeling Language (UML)".

• Rational Unified Process: Best Practice for Software Development Teams: Rational Software White Paper, TP026, Rev 11/01.

Suggested Readings

• Chapter 4 on Software Processes in Summerville's book.



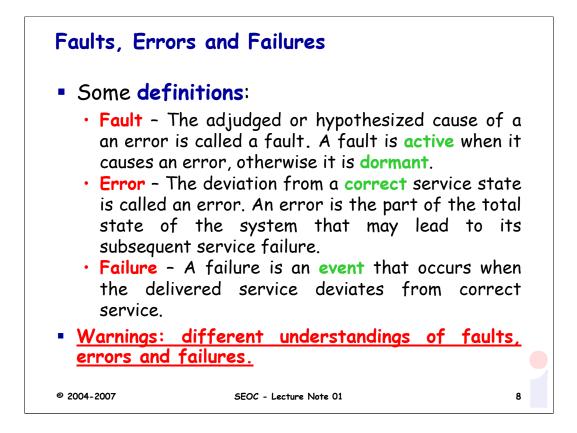
Unfortunately, software still *fails* too often. Software fails in complex manners. Although the course stresses the importance of software designs and models, it is often difficult to understand how software engineering aspects (e.g., design, implementation, etc.) relate to or address software failures. Software failures may have dependability (e.g., safety, reliability, etc.) as well as financial implications.

Readings

• R.N. Charette. Why Software Fails. IEEE Spectrum, pp. 42-49, September 2005.

Suggested Readings

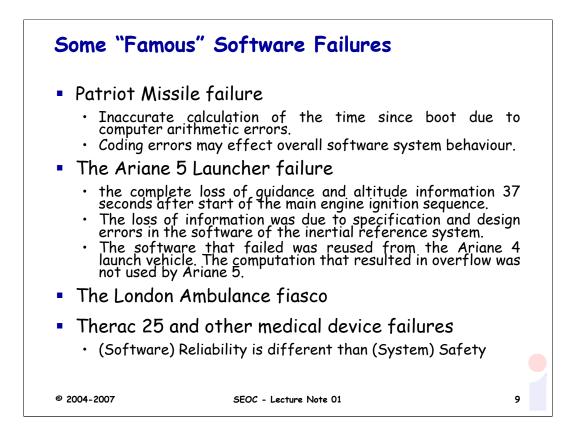
• Chapter 3 on Critical Systems in Sommerville's book.



An important aspect is to understand how faults, errors and failures relate each other. Research and practice in engineering safety-critical systems emphasize the underlying mechanisms of software failures. Note that understanding these concepts (i.e., faults, errors and failures) in practice often requires expertise within specific application domains, which might have different interpretations of them.

Suggested Readings

• A. Avizienis, J.-C. Laprie, B. Randell and C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing 1(1):11-33, January-March 2004.

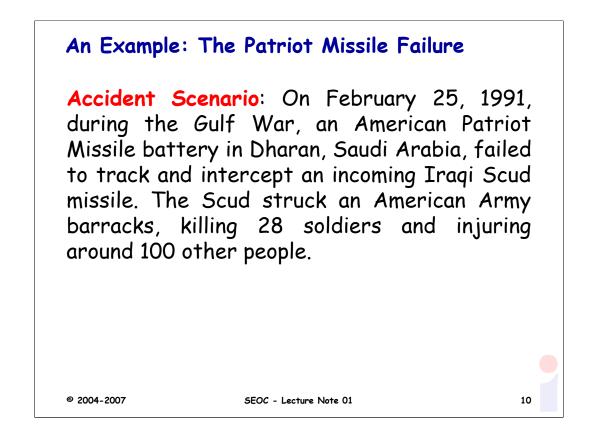


Readings

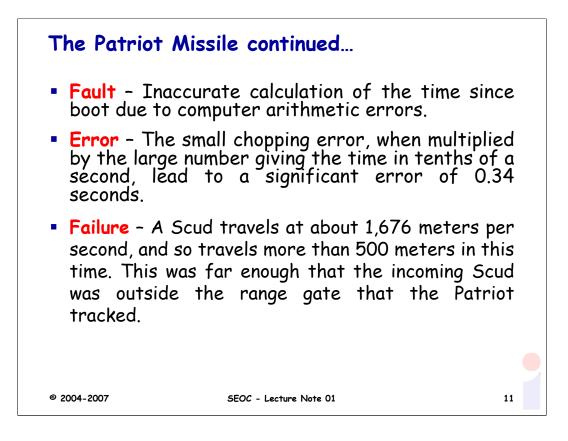
- B. Nuseibeh. Ariane 5: Who Dunnit? IEEE Software, pp. 15-16, May/June 1997.
- J.-M. Jézéquel, B. Meyer. Design by Contract: The Lessons of Ariane. IEEE Computer, pp. 129-130, January 1997.
- M. Grottke, K.S. Trivedi. Fighting Bugs: Remove, Retry, Replicate, and Rejuvenate. IEEE Computer, pp. 107-109, February 2007.

Suggested Readings

- N.G. Leveson, C.S. Turner. An investigation of the Therac-25 accidents. IEEE Computer 26(7): 18-41, Jul 1993.
- D.R. Wallace, D.R. Kuhn. Lessons from 342 Medical Device Failures. In Proceedings of HASE 1999, pp. 123-131.



A report of the General Accounting office, GAO/IMTEC-92-26, entitled *Patriot Missile Defense:* Software Problem Led to System Failure *at Dhahran, Saudi Arabia,* reported on the cause of the failure.



Fault – The time in tenths of second as measured by the system's internal clock was multiplied by 1/10 to produce the time in seconds. This calculation was performed using a 24 bit fixed point register. In particular, the value 1/10, which has a non-terminating binary expansion, was chopped at 24 bits after the radix point.

Error – Indeed, the Patriot battery had been up around 100 hours, and an easy calculation shows that the resulting time error due to the magnified chopping error was about 0.34 seconds. The binary expansion of 1/10 is

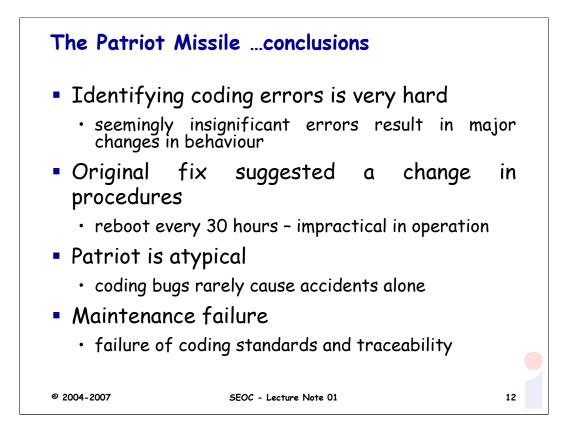
0.0001100110011001100110011001100...

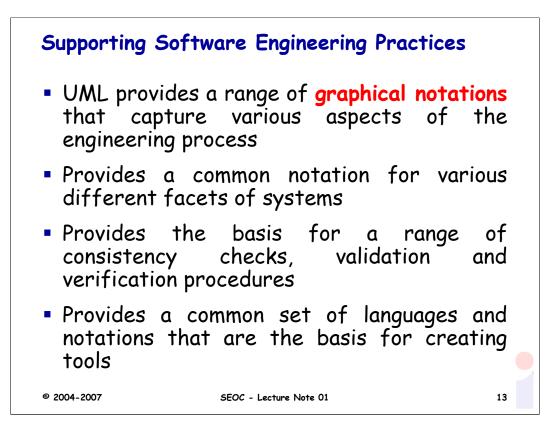
The 24 bit register in the Patriot stored instead

0.00011001100110011001100

introducing an error of

Ironically, the fact that the bad time calculation had been improved in some parts of the code, but not all, contributed to the problem, since it meant that the inaccuracies did not cancel.





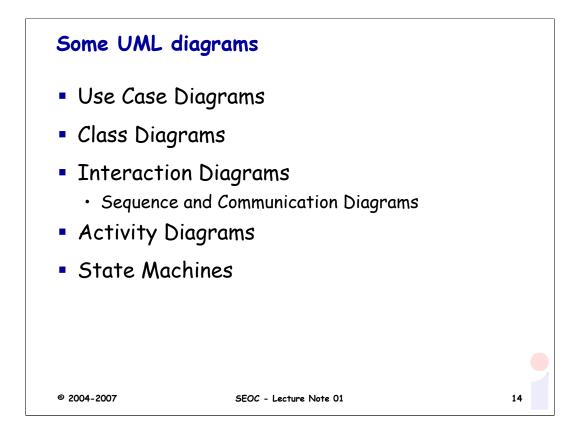
Readings

• UML course textbook

- Chapter 1 on the Introduction to the Case Studies.
- Chapter 2 on the Background to UML.

Suggested Readings

• G. Cernosek, E. Naiburg. The Value of Modeling. Rational Software, Copyright IBM Corporation 2004. This paper provides a brief technical discussion on software modeling.



Use Case Diagrams

• Used to support requirements capture and analysis; show the actors' involvement in system activities

Class Diagrams

• Capture the static structure of systems; associations between classes

Interaction Diagrams

• Capture how objects interact to achieve a goal

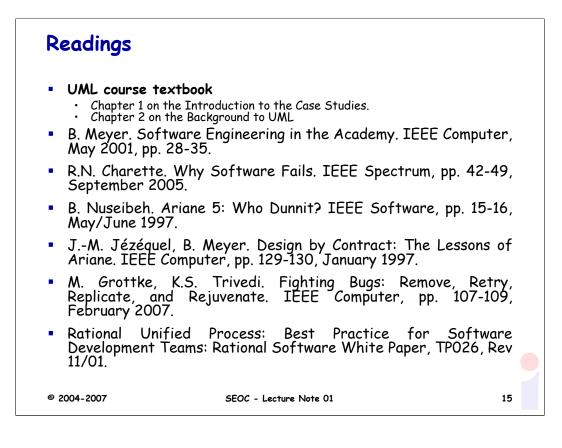
Activity Diagrams

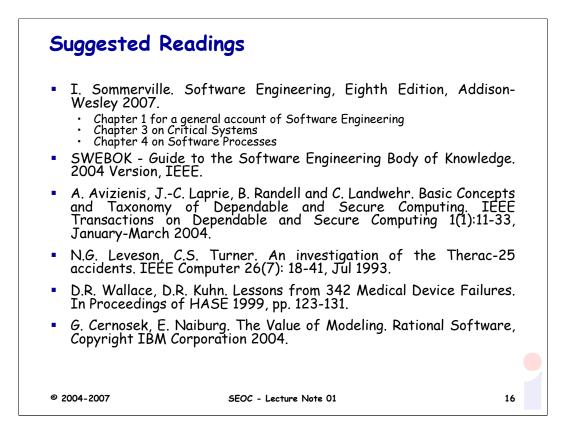
• Capture the workflow in a situation

State Machines:

• Capture state change in objects of the system

Other Diagrams: Component and Deployment Diagrams





Summary

- SEOC organization
- An introduction to Software Engineering
- Why Software Fails
- Faults, Errors and Failures
- Examples of Software Failures
- An Outline of some UML diagrams
- Readings and Suggested Readings

© 2004-2007

SEOC - Lecture Note 01

17