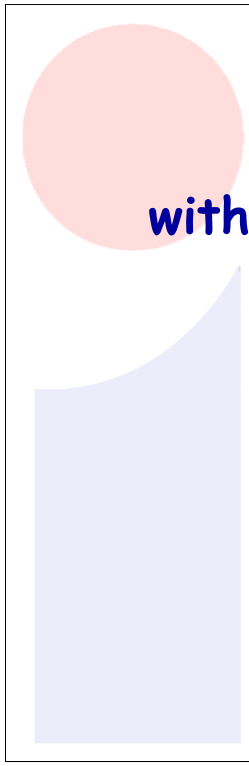

# Software Engineering with Objects and Components

Massimo Felici

IF-3.46 0131 650 5899

mfelici@inf.ed.ac.uk

## Course Organization

- SEOC course webpage

  http://www.inf.ed.ac.uk/teaching/courses/seoc/

- Mailing List

  seoc-students@inf.ed.ac.uk

- Newsgroup

  eduni.inf.course.seoc1

- SEOC CVS repositories

## Course Organization

- **Course Textbook**

  UML, Second Edition, by Simon Bennet, John Skelton and Ken Lunn, Schaum's Outline Series, McGraw-Hill, 2005

- **Course Resources**

  Lecture Notes and References

- **Software**

  Eclipse + UML plug-ins and Java

## Course Organization

- **Tutorials** begin in **week 3**
  - Frequency: once a week
  - Maximum **12 people** per tutorial group
- **Coursework**
  - in small teams (**approx 3-4 people**)
  - two deliverables equally weighted
  - **Deadlines**
    - **1st deliverable: Friday, 31st October**
    - **2nd deliverable: Friday, 28th November**
- **Assessment**
  - 25% coursework; 75% degree examination

## What is Software Engineering?

***Software Engineering*** *is an* ***engineering discipline*** *that is concerned with* ***all aspects of software production*** *from the early stages of system specification to maintaining the system after it has gone into use.*





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.

## Some Software Engineering Aspects

- Software Processes
- Software Process Models
- Software Engineering Methods
- Costs
- Software Attributes
- Tools
- Professional and Ethical Responsibilities



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.

## Why Software Fails?

- Complex causes (interactions) trigger software failures
- Software fails in context
- Some issues related to software engineering
  - Misunderstood requirements
  - Design issues
  - Mistakes in specification, design or implementation
  - Operational issues
- Faults, Errors and Failures





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.

## Faults, Errors and Failures

- Some **definitions**:
  - **Fault** – The adjudged or hypothesized cause of a an error is called a fault. A fault is **active** when it causes an error, otherwise it is **dormant**.
  - **Error** – The deviation from a **correct** service state is called an error. An error is the part of the total state of the system that may lead to its subsequent service failure.
  - **Failure** – A failure is an **event** that occurs when the delivered service deviates from correct service.
- **Warnings: different understandings of faults, errors and failures.**



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.

## Some "Famous" Software Failures

- Patriot Missile failure
  - Inaccurate calculation of the time since boot due to computer arithmetic errors.
  - Coding errors may effect overall software system behaviour.
- The Ariane 5 Launcher failure
  - the complete loss of guidance and altitude information 37 seconds after start of the main engine ignition sequence.
  - The loss of information was due to specification and design errors in the software of the inertial reference system.
  - The software that failed was reused from the Ariane 4 launch vehicle. The computation that resulted in overflow was not used by Ariane 5.
- The London Ambulance fiasco
- Therac 25 and other medical device failures
  - (Software) Reliability is different than (System) Safety



**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.

## An Example: The Patriot Missile Failure

**Accident Scenario**: On February 25, 1991, during the Gulf War, an American Patriot Missile battery in Dharan, Saudi Arabia, failed to track and intercept an incoming Iraqi Scud missile. The Scud struck an American Army barracks, killing 28 soldiers and injuring around 100 other people.





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.

## The Patriot Missile continued…

- **Fault** – Inaccurate calculation of the time since boot due to computer arithmetic errors.
- **Error** – The small chopping error, when multiplied by the large number giving the time in tenths of a second, lead to a significant error of 0.34 seconds.
- **Failure** – A Scud travels at about 1,676 meters per second, and so travels more than 500 meters in this time. This was far enough that the incoming Scud was outside the range gate that the Patriot tracked.





**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

0.00000000000000000000011001100... binary, or about 0.000000095 decimal. Multiplying by the number of tenths of a second in 100 hours gives 0.000000095×100×60×60×10=0.34.

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.

## The Patriot Missile …conclusions

- Identifying coding errors is very hard
  - seemingly insignificant errors result in major changes in behaviour
- Original fix suggested a change in procedures
  - reboot every 30 hours – impractical in operation
- Patriot is atypical
  - coding bugs rarely cause accidents alone
- Maintenance failure
  - failure of coding standards and traceability

## Supporting Software Engineering Practices

- UML provides a range of **graphical notations** that capture various aspects of the engineering process
- Provides a common notation for various different facets of systems
- Provides the basis for a range of consistency checks, validation and verification procedures
- Provides a common set of languages and notations that are the basis for creating tools



**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.

## Some UML diagrams

- Use Case Diagrams
- Class Diagrams
- Interaction Diagrams
  - Sequence and Communication Diagrams
- Activity Diagrams
- State Machines



**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**

# Readings

- **UML course textbook**
  - Chapter 1 on the Introduction to the Case Studies.
  - Chapter 2 on the Background to UML
- B. Meyer. Software Engineering in the Academy. IEEE Computer, May 2001, pp. 28-35.
- R.N. Charette. Why Software Fails. IEEE Spectrum, pp. 42-49, September 2005.
- 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.
- Rational Unified Process: Best Practice for Software Development Teams: Rational Software White Paper, TP026, Rev 11/01.

# Suggested Readings

- I. Sommerville. Software Engineering, Eighth Edition, Addison-Wesley 2007.
  - Chapter 1 for a general account of Software Engineering
  - Chapter 3 on Critical Systems
  - Chapter 4 on Software Processes
- SWEBOK - Guide to the Software Engineering Body of Knowledge. 2004 Version, IEEE.
- 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.
- 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.
- G. Cernosek, E. Naiburg. The Value of Modeling. Rational Software, Copyright IBM Corporation 2004.

## 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**