Software Verification and Validation

Prof. Lionel Briand
Ph.D., IEEE Fellow

Lionel's background

- Worked in industry, academia, and industry-oriented research institutions
- France, USA, Germany, Canada, Norway
- Scientist in a NASA GSFC lab, Maryland, USA
- Worked with ESA on Ariane launcher SW testing processes
- Head of the SW quality engineering department at a Fraunhofer Institute in Germany (Siemens, Bosch, Daimler)
- Professor and Canada Research Chair in software quality engineering (collaborated with IBM, Siemens, Nortel)
- Moved to Simula Research Laboratory in July 2007 to build a new area of expertise: Software Testing, Verification, and Validation
- Founded the first IEEE conference on software verification and validation (ICST)
- On the editorial board of Software Testing, Verification, and Reliability (STVR) journal, Wiley
- Leading a new research center on software V&V at Simula

V&V Definitions

- **Verification and Validation** (**V&V**) is the process of checking that a software system meets specifications and that it fulfils its intended purpose.
- **Verification** is a Quality control process that is used to evaluate whether or not a product, service, or system complies with regulations, specifications, or conditions imposed at the start of a development phase. This is often an internal process.
- **Validation** is a Quality assurance process of establishing evidence that provides a high degree of assurance that a product, service, or system accomplishes its intended requirements. This often involves acceptance of fitness for purpose with end users and other product stakeholders.
- "Are you building the right thing?" versus "Are you building it right?"

Course Introduction

Course Objectives

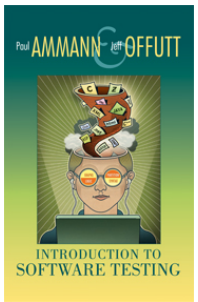
- Foundations of software verification and validation (V&V)
- Mostly a practical perspective
- Emphasis on testing (many kinds)
- Most important V&V technique in practice
- But also safety analysis, fault tolerance, model checking, requirements quality assurance, etc
- More basic coverage of other aspects
- No emphasis on tools as this will change (quickly) over time
- But, in the future, you'll be able to assess how useful a tool is …

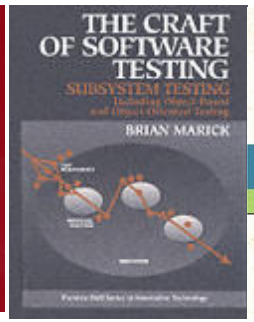
Main Sources

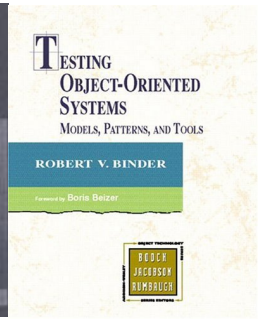
- A. Mathur, Foundations of Software Testing, Pearson Education, 2008
- M. Pezze and M. Young, Software Analysis and Software Testing, Wiley, 2007
- P. Ammann and J Offutt, Introduction to Software Testing, Cambridge Press, 2007

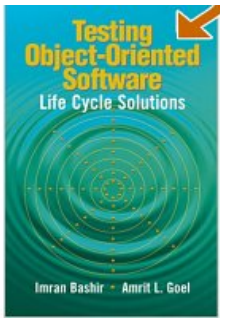
Other Sources

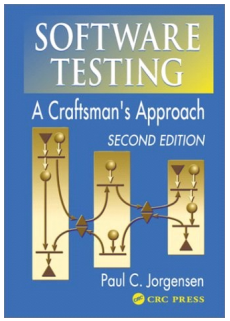
1. B. Beizer, "Software Testing Techniques", Van Nostrand Reinhold, New York, 2nd Ed., 1990
2. B. Marick, "The Craft of Software Testing", Prentice Hall, 1995
3. M. Roper, "Software Testing", McGraw-Hill, 1995
4. Bashir and Goel. "Testing Object-oriented Software", Springer, 2000
5. Jorgensen, "Software Testing", A Craftman's Approach", CRC Press, 1995
6. R. V. Binder, "Testing Object-Oriented Systems - Models, Patterns, and Tools", Addison-Wesley, 1999





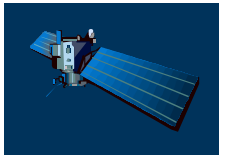






Software has become prevalent in all aspects of our lives









Nature of Software Development

- Development, not production
- Human intensive
- Engineering, but also social process
- Increasingly complex software systems
- Pervasive in an increasing number of industries

Errors

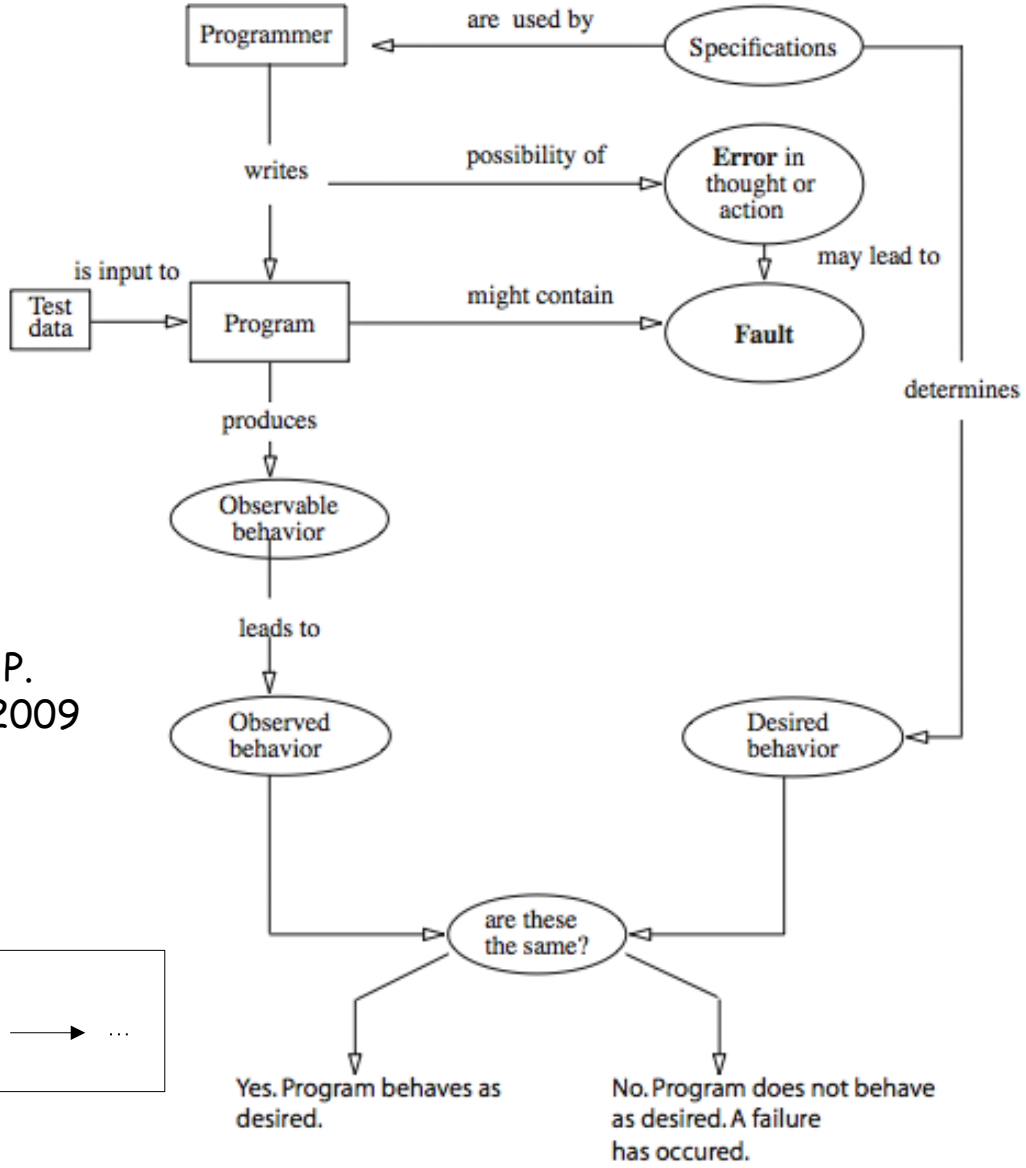
Errors are a part of our daily life.

Humans make errors in their thoughts, actions, and in the products that might result from their actions.

Errors occur wherever humans are involved in taking actions and making decisions.

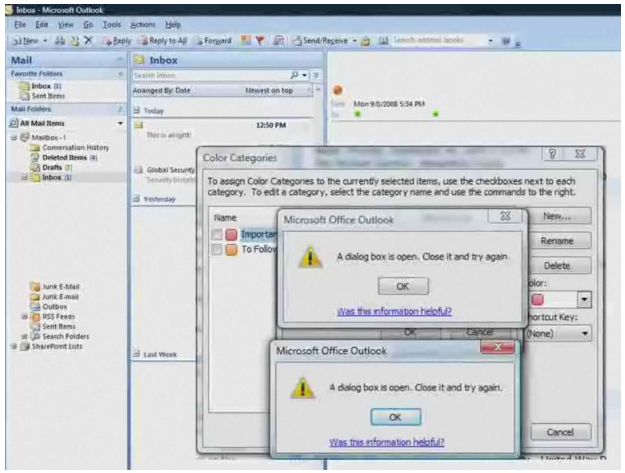
These fundamental facts of human existence make testing an essential activity.

Software Error, faults, failures

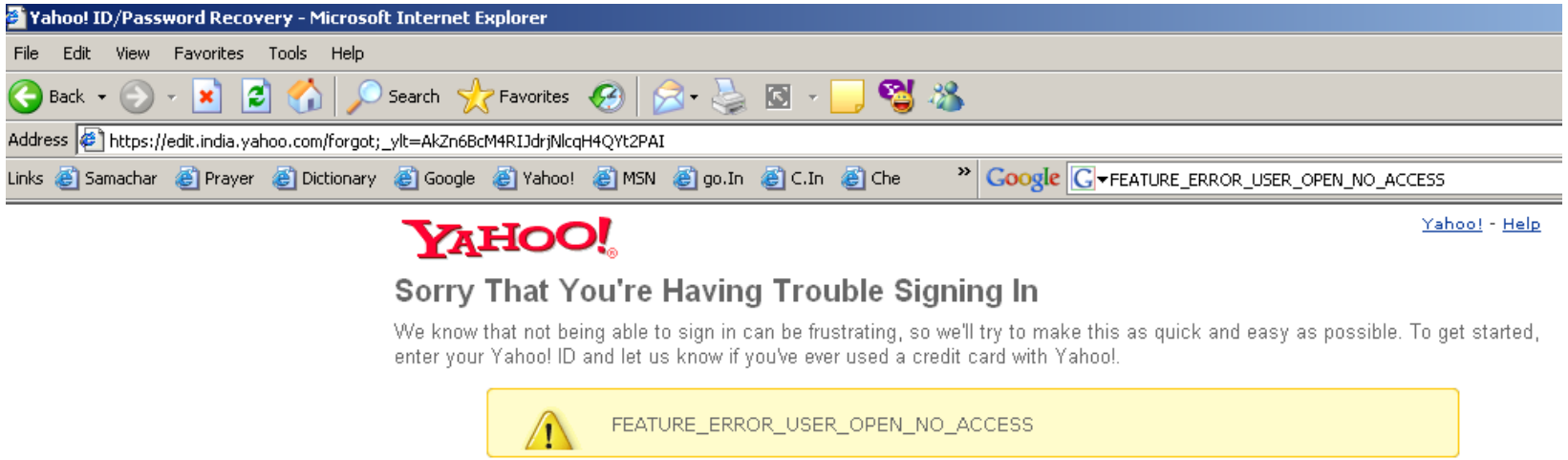


Error —*propagation*→ Fault —*causation*→ Failure —*results in*→ Incident → …

Everywhere …



Example 1



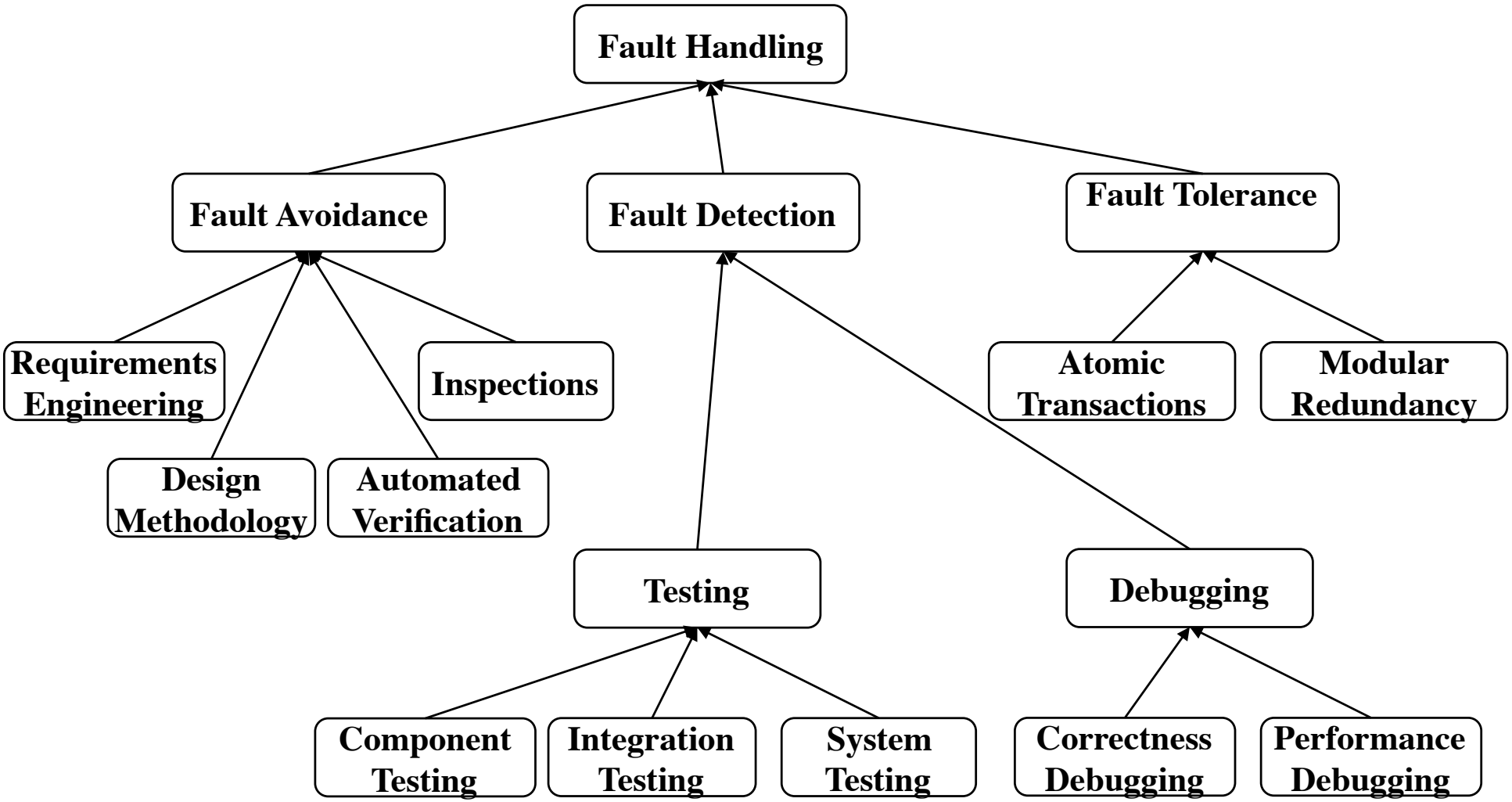
- Incident: Yahoo! mail doesn't let me log in
- Failure: The user account cannot be accessed in the user database.
- Fault: The user database can not be reached.
- Error: There was no backup user database in the system.

Example 2

- Fatal Therac-25 X-ray Radiation
- In 1986, a man in Texas received between 16,500-25,000 radiations in less than 10 sec, over an area of about 1 cm.
- He passed away 5 months later.
- The root cause of the incident was a SW failure ☹

- Incident: A patient passed away
- Failure: The device applied higher frequency of radiations than what was safe. Safety range: [1…10,000 Hz].
- Fault: The software controller of the device did not have a conditional block (if …. else statements) to perform range checking on the frequency of the radiation to be applied.
- (2) Errors:
  1. The SW developer of the device controller system had forgotten to include a range checking conditional block on the frequency of the radiation to be applied.
  2. The device operator was NOT supposed to enter anything outside [1…10,000 Hz] range.

Dealing with SW Faults



Software Quality

Software quality attributes

Static quality attributes: structured, maintainable, testable code as well as the availability of correct and complete documentation.

Dynamic quality attributes: software reliability, correctness, completeness, consistency, usability, and performance

Software quality (contd.)

Completeness refers to the availability of all features listed in the requirements, or in the user manual. An incomplete software is one that does not fully implement all features required.

Consistency refers to adherence to a common set of conventions and assumptions. For example, all buttons in the user interface might follow a common color coding convention. An other example of inconsistency regarding data confidentiality would be when a database application displays the date of birth of a person in the database.

Software quality attributes (contd.)

Usability refers to the ease with which an application can be used. This is an area in itself and there exist techniques for usability testing. Psychology plays an important role in the design of techniques for usability testing.

Performance refers to the time the application takes to perform a requested task. It is considered as a *non-functional requirement*. It is specified in terms such as ``This task must be performed at the rate of X units of activity in one second on a machine running at speed Y, having Z gigabytes of memory."

ISO 9126: Evaluation of Software Quality

- **Functionality** - *A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.*
  - Suitability
  - Accuracy
  - Interoperability
  - Compliance
  - Security
- **Reliability** - *A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.*
  - Maturity
  - Recoverability
  - Fault Tolerance
- **Usability** - *A set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users.*
  - Learnability
  - Understandability
  - Operability
- **Efficiency** - *A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions.*
  - Time Behaviour
  - Resource Behaviour
- **Maintainability** - *A set of attributes that bear on the effort needed to make specified modifications.* Stability
  - Analyzability
  - Changeability
  - Testability
- **Portability** - *A set of attributes that bear on the ability of software to be transferred from one environment to another.*
  - Installability
  - Replaceability
  - Adaptability
  - Conformance (similar to compliance, above, but here related specifically to portability, e.g. conformance to a particular database standard)

Pervasive Problems

- Software is commonly delivered late, way over budget, and of unsatisfactory quality
- Software validation and verification are rarely systematic and are usually not based on sound, well-defined techniques
- Software development processes are commonly unstable and uncontrolled
- Software quality is poorly measured, monitored, and controlled.
- Software failure examples: http://www.cse.lehigh.edu/~gtan/bug/softwarebug.html

Consequences of Poor Quality

- Standish Group surveyed 350 companies, over 8000 projects, in 1995
- 31% cancelled before completed, 9-16% were delivered within cost and budget
- US study (1995): 81 billion US$ spent per year for failing software development projects
- http://www.it-cortex.com/Stat_Failure_Rate.htm
- NIST study (2002): bugs cost $ 59.5 billion a year. Earlier detection could save $22 billion.

Definitions: Software Engineering

- Software engineering (SE) is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software.
- The discipline of software engineering encompasses knowledge, tools, and methods for defining software requirements, and performing software design, software construction, software testing, and software maintenance tasks.
- Software engineering draws on knowledge from fields such as computer engineering, computer science, management, mathematics, project management, quality management, software ergonomics, and systems engineering.
- The term *software engineering* was popularized during the 1968 NATO Software Engineering Conference (held in Garmisch, Germany).

V&V Definitions

- SW quality Engineering: The discipline of specifying, assuring, and controlling the quality of software products
- SW management: The discipline of managing projects to achieve quality within time constraints and budget
- SW verification: The goal is to find as many latent defects as possible before delivery
- SW validation: The goal is to gain confidence in the software, shows it meets its specifications

V&V Definitions (contd.)

- SW Testing: Techniques to execute programs with the intent of finding as many defects as possible and/or gaining sufficient confidence in the software system under test.
  - "Program testing can show the presence of bugs, never their absence" (Dijkstra)

- SW Inspections: Techniques aimed at *systematically* verifying non-executable software artifacts with the intent of finding as many defects as possible, as early as possible

Course Project

- *Project*: Apply some of the testing techniques to actual software, quantitatively compare techniques, assess drawbacks and advantages
- Short initial plan + final report
- Guidelines for plan and report

Practical Information

- Students should apply the software testing techniques learned in the course to a case study software system of their choice
- http://sourceforge.net/
- You should use software testing tools:
  - 10 commercial tools can be downloaded with an academic license or a trial version
  - Many open source test tools on: http://www.opensourcetesting.org
  - Study systems can be provided. You can also select your own case study system (but it should be reasonably complex)!
- Note that you must try one or several techniques discussed in the course with the tool(s) of your choice.

Tool
Vendor: Parasoft
Parasoft Jtest
Parasoft C++test
Parasoft .TEST
Vendor: IBM Rational
IBM Rational Functional Tester
IBM Rational Manual Tester
IBM Rational Test RealTime
IBM Rational Performance Tester
IBM Rational TestManager
Others vendors
JUnit
JMeter

Expectations

- Thorough review of literature (journals and conferences)
- Integration and synthesis of state-of-the art material
- Application and comparison of techniques on case studies
- When relevant, report on practical experience (e.g., tools)
- Clear, well-structured report
- Start thinking right away of what you want to do and get out of the course
- Start reading ASAP!

Report Evaluation

- Coverage and depth of material in report
- Thoroughness of gained understanding on selected topic
- Completeness of analysis and results
- Capability to synthesize and structure the discussion
- Quality of presentation

Deadlines

- To be investigated
- Depends on when must grades be provided to the university
- Submissions are made through email to my attention
- Expect acknowledgement of submission

Advice

- Review on a regular basis the course notes, go through the examples
- Start project thinking early on
- Ask questions in class

Additional Books

1. SafeWare: system safety and computers, N. Leveson, Addison-Wesley. 1995
2. Handbook of Software Reliability, McGraw-Hill, M. Lyu, editor. 1995
3. Software fault tolerance, M. Lyu, Chichester, England, Wiley. 1995
4. Metrics and models in software quality engineering, 2nd edition, S. Kan, Addison-Wesley. 2003
5. N. E. Fenton, S. L. Pfleeger; 1996; *Software Metrics: A Rigorous and Practical Approach; 2nd Ed.*; International Thomson Computer Press; U.K.

General Software Engineering

- If needed you have to refresh your SE knowledge (UML, etc.)

1. Object-Oriented Software Engineering, Bruegge and Dutoit, Prentice-Hall 2000
2. Software engineering, I. Sommerville, Addison-Wesley
3. An many others all available at the library …

Journals

- IEEE Transactions on Software Engineering
- ACM Transactions on Software Engineering and Methodology
- Software Testing, Verification, and Reliability (Wiley)
- Journal of Systems and Software (Elsevier)
- Journal of Software Practice and Experience (Wiley)
- Empirical Software Engineering (Springer)

Conferences

- IEEE International Conference on Software Testing, Verification, and Validation (ICST)
- IEEE International Symposium on Software Reliability Engineering (ISSRE)
- ACM International Symposium on Software Testing and Analysis (ISSTA)
- International Conference on Software Engineering (ICSE)