Software Quality

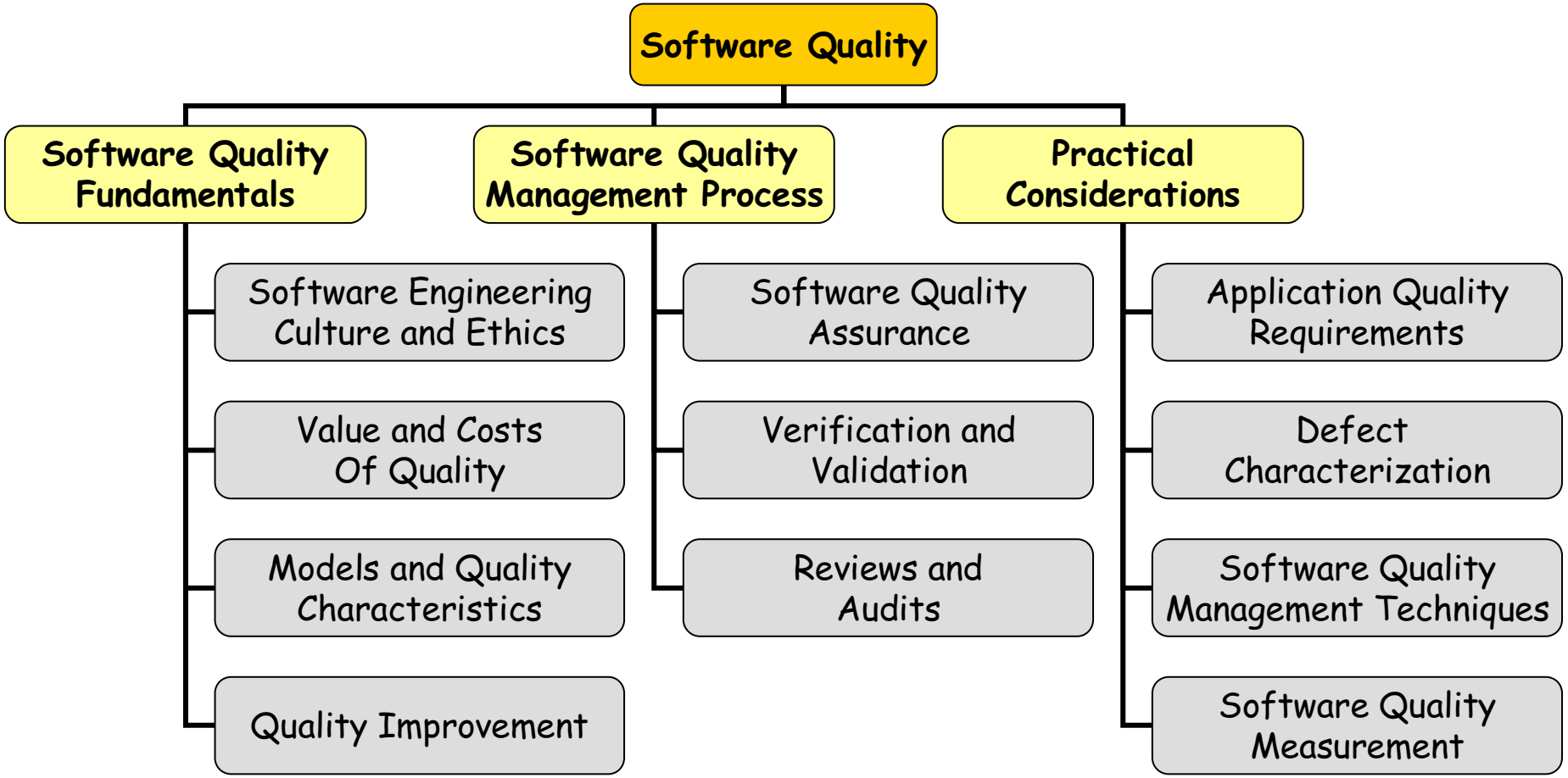
Massimo Felici

Room 1402, JCMB, KB

0131 650 5899

mfelici@inf.ed.ac.uk

Software Quality at a Glance



Software Quality Fundamentals

- **Commitment** to **Software Quality** as part of **organisation culture**
- The notion of "**quality**" is difficult
  - Identification of quality **characteristics**, **attributes** and **targets**
- **Quality cost**
- **Modelling Software Quality**
  - **Process** and **Product quality**
  - It is difficult (may be, not entirely possible) to distinguish process quality from product quality

Process and Product Quality

- A fundamental assumption of quality management is that the quality of the development process directly affects the quality of the delivered products
  - Origin in manufacturing
- The link between software process quality and software product quality is complex
- **Process quality management** involves
  - **Defining process standards** such as how and when reviews should be conducted
  - **Monitoring** the development process to ensure that the standards are being followed
  - **Reporting** the software process to project management and to the buyer of the software

Software Quality Management Processes

- **Planning** software quality involves
  - Defining the required product in terms of its **quality (internal and external) characteristics**
    - Characteristics that govern how the product works in its environment are called **external**, which include, for example, usability and reliability
    - Characteristic related to how the product was developed are called **internal** quality characteristics, and include, for example, structural complexity, size, test coverage, and fault rates
  - Planning the processes to achieve the required product
- Some **Software Quality Management** (SQM) processes
  - **Quality assurance process**: ensuring that the software products and processes in the project life cycle conform to their specified requirements by planning, enacting, and performing a set of activities to provide adequate confidence that the quality is being built into the software
  - **Verification and validation processes**: assessing software products throughout the product life-cycle
  - **Review and audit processes**: involving Management reviews, Technical reviews, Inspections, Walk-throughs, and Audits

Quality Planning

- **Quality planning** is the process of developing a quality plan for the project
- A quality plan involves
  - **Product introduction**: A description of the product, its intended market and the quality expectation for the product
  - **Product plans**: The critical release dates and responsibilities for the product along with plans for distribution and product servicing
  - **Process descriptions**: the development and service processes that should be used for product development and management
  - **Quality Goals**: The quality goals and plans for the product including an identification and justification of critical product quality attributes
  - **Risks and risk management**: The key risks that might affect product quality and the actions to address these risks

Quality Assurance and Standards

- **Quality assurance** is the process of defining how software quality can be achieved and how the development organization knows that the software has the required level of quality
- **Quality assurance standards**
  - **Product standards**
  - **Product standards**
- **Software standards**
  - Are based on knowledge ("best-practice" or "state-of-the-art")
  - Provide a framework for implementing a quality assurance process
  - Assist/support continuity in practice within an organization
- **Issues**: Software engineers sometimes dislike standards
  - Need to involve software engineering in the selection of standards
  - Review and modify standards regularly to reflect changing technologies
  - Provide software tools to support standards where possible

Software Metrics

Measurement

- **Measurement** is the process by which numbers or symbols are assigned to **attributes** of **entities** in the real world in such a way as to describe them according to clearly defined rules
  - Measurement is a **direct** quantification
  - Calculation (or indirect measurement) is **indirect**
- **Issues**. Unfortunately, most software development processes
  - Fail to set measurable targets for software products
  - Fail to understand and quantify the component costs of software projects
  - Fail to quantify and predict the quality of the produced product
  - Allow anecdotal evidence to convince us to try yet another revolutionary new development technology, without doing pilot projects to assess whether the technology is efficient and effective
- **Tom Gilb's Principle of Fuzzy Targets**: projects without clear goals will not achieve their goals clearly
- **Tom DeMarco's Principle**: You cannot control what you cannot measure

The Scope of Software Metrics

- Cost and effort estimation
- Productivity
- Data collection
- Quality models and measures
- Reliability models
- Performance evaluation and models
- Structural and complexity metrics
- Capability-maturity assessment
- Management by metrics
- Evaluation of methods and tools

The Basics of Measurement

- **Measurement**: a mapping from the empirical world to the formal, relational world
- A **measure** is the number or symbol assigned to an entity by this mapping in order to characterise an attribute
  - **Direct** and **Indirect measurement**
  - **Direct measurement** of an attribute of an entity involves no other attribute or entity
- **Measurement for prediction**. A prediction system consists of a mathematical model together with a set of prediction procedures for determining unknown parameters and interpreting results (Littlewood, 1988)
- Measurement scales (mappings between measurement and empirical and numerical relation systems) and scale types (e.g., nominal, ordinal, interval, ratio, absolute, etc.)

Classifying Software Measures

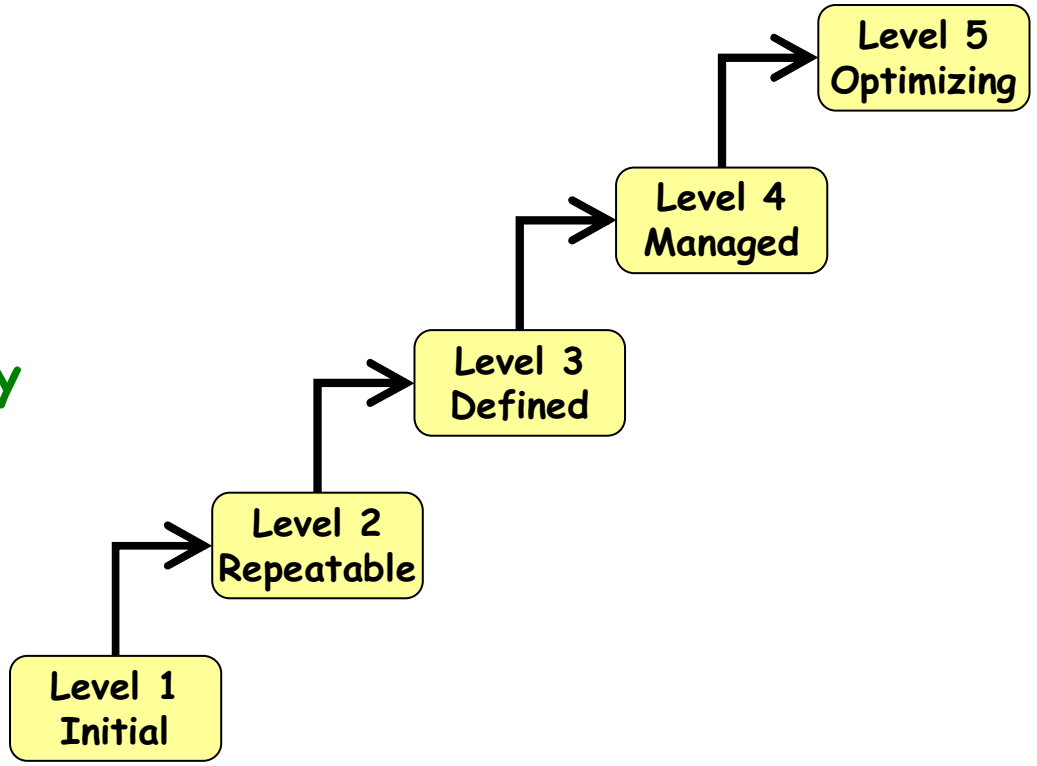
- Relevant **software entities**
  - **Processes** are collection of software related activities
  - **Products** are any artifacts, deliverables or documents that result from a process activity
  - **Resources** are entities required by a process activity
- **Internal attributes** of a product, process or resource are those that can be measured purely in terms of the product, process or resource itself. In other worlds, an internal attribute can be measured by examining the product, process or resource on its own, separate from its behaviour
- **External attributes** of a product, process or resource are those that can be measured only with respect to how the product, process or resource relates to its **environment**. Here, the **behaviour** of the process, product or resource is important, rather than the entity itself.

Determining What to Measure: GQM

- **Goal-Question-Metric (GQM)** is an approach to selecting and implementing metrics
- The **GQM** approach provides a framework involving three steps
  1. List the major **goals** of the development or maintenance project
  2. Derive from each goal the **questions** that must be answered to determine if the goals are being met
  3. Decide what **metrics** must be collected in order to answer the questions adequately

Measurement and Process Improvements

- Measurement enhances visibility into the ways in which processes, products, resources, methods, and technologies of software development relate to one another
- The Software Engineering Institute (SEI)'s **Capability Maturity Model** (**CMM**) consists of five maturity levels
  - **Level 1: Initial**
  - **Level 2: Repeatable**
  - **Level 3: Defined**
  - **Level 4: Managed**
  - **Level 5: Optimizing**
- Other models: ISO 9000, SPICE, etc.



Software Measurement Validation

- **Validating a prediction system** in a given **environment** is the process of establishing the accuracy of the prediction system by empirical means. That is, by comparing model performance with known data in the given environment.
- **Validating a software measure** is the process of ensuring that the measure is a proper numerical characterization of the claimed attribute by showing that the representation condition is satisfied.

Empirical Investigation

- **Software Engineering Investigations**
  1. **Experiments**: research in the small
  2. **Case Studies**: research in the typical
  3. **Surveys**: research in the large
- State **hypothesis** and determine how much **control** is needed over the **variables** involved
  1. If control is not possible, the a formal experiment is not possible
  2. Then a case study may be a better approach
  3. If the study is retrospective, then a survey may be done
- State Six stage of an experiment: **conception**, **design**, **preparation**, **execution**, **analysis** and **dissemination**

Software-metrics Data Collection

- What is Good **Data**?
  - **Correctness**: Are they correct?
  - **Accuracy**: Are they accurate?
  - **Precision**: Are they appropriately precise?
  - **Consistency**: Are they consistent?
  - Are they associated with a particular activity or time period?
  - Can they be replicated?
- **Software Quality Terminology**
  - A **fault** occurs when a human **error** results in a mistake in some software product
  - A **failure** is the departure of a system from its required behaviour.

    Errors -> Faults -> Failures
  - **NOTE: to many organizations, errors often mean faults**

    Faults -> Errors -> Failures
  - **Anomalies** usually means a class of faults that are likely to cause failures in themselves but may nevertheless eventually cause failures indirectly
  - **Defects** normally refer collectively to faults and failures
  - **Bugs** refer to faults occurring in the code
  - **Crashes** are special type of failure, where the system ceases to function

Problem Record

- **Location**:
  where did the problem occur?
- **Timing**:
  when did it occur?
- **Symptom**:
  what was observed?
- **End result**:
  which consequences resulted?
- **Mechanism**:
  how did it occur?
- **Cause**:
  why did it occur?
- **Severity**:
  how much was the user affected?
- **Cost**:
  how much did it cost?

An example drawn from the Therac 25

- **Location**: East Texas Cancer in Tyler, Texas, USA
- **Timing (1)**: March 21 1986, at whatever the precise time that "Malfucntion 54" appeared on the screen
- **Timing (2)**: total number of treatment hours on all Therac 25 machines up to that particular time
- **Symptom (1)**: "Malfucntion 54" appeared on screen
- **Symptom (2)**: classification of the particular program of treatment being administrated, type of tumor, etc.
- **End result**: strenght of beam tto great by a factor of 100
- **Mechanism**: use of the up-arrow key while setting up the machine led to the corruption of a particular internal variable in the software
- **Cause (1)**: (trigger) unintentional operator action
- **Cause (2)**: (source type) unintentional design fault
- **Severity**: critical, as injury to the patient was fatal
- **Cost**: effort or actual expenditure by accident investigators

Analyzing Software-measurement Data

- Describe a **set of attribute values** using **box plot statistics** (based on median and quartiles) rather than on mean and variance
- Inspect a **scatter plot** visually when investigating the relationship between two variables
- Use **robust correlation coefficients** to confirm whether or not a relationship exists between two attributes
- Use **robust regression** in the presence of atypical values to identify a linear relationship between two attributes, or remove the atypical values before analysis
- Always check the residuals by plotting them against the dependent variable
- Carefully transform non-linear relationships
- Use principal components analysis to investigate the dimensionality of data sets with large numbers of correlated attributes

Measuring Internal Product Attributes

- Examples of Internal Attributes
  - **Size** (e.g., length, functionality, complexity, reuse, etc.) or **Structure**
- Simple measurements of size fail adequately to reflect other attributes, e.g., effort, productivity and cost
- Example of **Length**: Line Of Code (LOC)
- Examples of **Complexity**: problem, algorithmic, structural or cognitive
- Types of **structural measures**: control-flow, data-flow and data
  - The structure of a module is related to the difficulty in **testing** it

Examples of Object-Oriented Metrics

1. **Weighted Methods per Class (WMC)**: is intended to relate to the notion of complexity
2. **Depth of Inheritance Tree (DIT)**: is the length of the maximum path from the node to the root of the inheritance tree
3. **Number of Children (NOC)**: relates to a node (class) of the inheritance tree. It is the number of immediate successors of the class.
4. **Coupling Between Object classes (CBO)**: is the number of other classes to which the class is coupled
5. **Response For Class (RFC)**: is the number of local methods plus the number of methods called by the local methods
6. **Lack of Cohesion Metric (LCOM)**: is defined as the number of disjointsets of local methods

Measuring External Product Attributes

- Modelling Software Quality
- The ISO 9126 standard quality model: functionality, reliability, efficiency, usability, maintainability and portability
- **Note**: Safety is a system property, not a software quality characteristic
- An example: **Usability** of a software product is the extent to which the product is convenient and practical to use
- Another example: **Usability** is the probability that the operator of a system will not experience a user interface problem during a given period of operation under a given **operational profile**

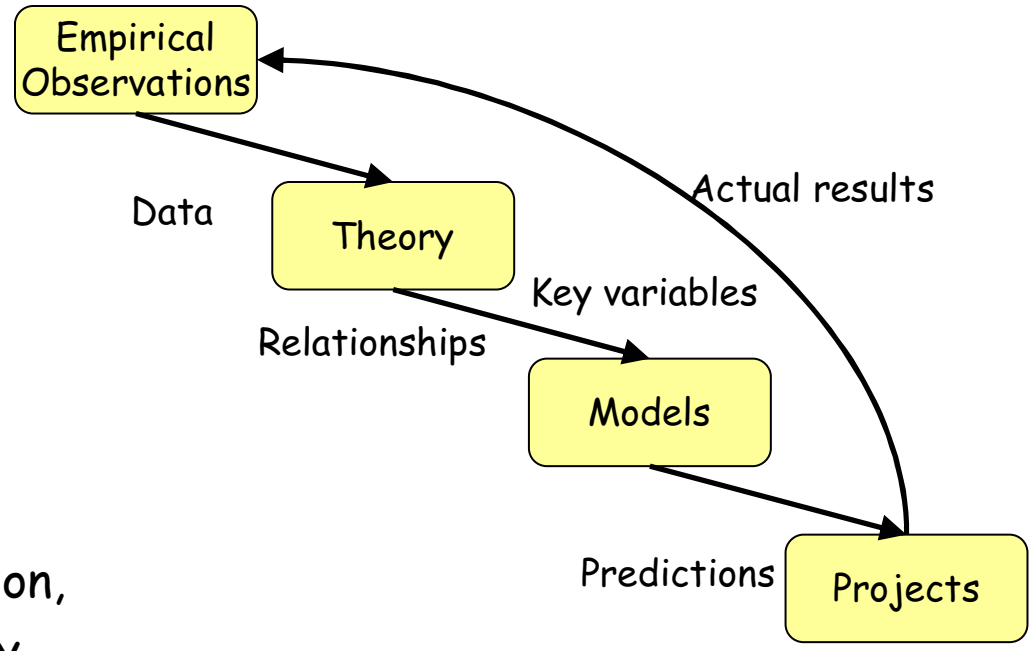
Software Reliability: Measurement and Prediction

- **The software reliability problem**
  - Hardware reliability is concerned with component failures due to physical wear – such failures are probabilistic in nature
  - The key distinction between software reliability and hardware reliability is the difference between intellectual failure (usually due to design faults) and physical failure.
- Reliability is defined in terms of failures, therefore it is impossible to measure before development is complete
  - However, carefully collected data on inter-failure times allow the prediction of software reliability
- **Software Reliability Growth Models** estimate the reliability growth
  - None can guarantee accurate predictions on all data sets in all environments
- **Limitations**: unfortunately, software reliability growth models work effectively only if the software's future **operational environment** is similar to the one in which the data was collected

Beyond Software

- Resource Measurement:
  - Productivity:
    - Distinguish between productivity of a process from the productivity of the resources
    - Should also take into account of the quality of the output
  - Team
    - Team Structure, size, communication density
  - Tools
- Making process prediction
  - Problems of estimations methods: local data definition, calibration, independent estimation group, reduce input subjectivity, preliminary estimates and re-estimation, alternative size measures for cost estimation, locally developed cost models

A General Prediction Process



Reading/Activity

- Please read Chapter 11 on Software Quality of the SWEBOW.
- Please read J. Bøegh, S. De Panfilis, B. Kitchenham and A. Pasquini, A Method for Software Quality Planning, Control, and Evaluation. In IEEE Software, March/April 1999, pp. 69-77.
- Please read A. Avižienis, J.-C. Laprie and B. Randell, Fundamental Concepts of Dependability. UCLA CSD Report no. 010028, LAAS Report no. 01-145, Newcastle University Report no. CS-TR-739.
- N. E. Fenton and S. L. Pfleeger. Software Metrics: A Rigorous and Practical Approach. Second Edition, International Thomson Computer Press, 1996.

Summary

- Software Quality
  - Software fundamentals
  - Process and product quality
  - Software quality management process
  - Quality planning
  - Quality assurance and standards
- Software Metrics