INTRODUCTION
PART 1

What is Software Engineering?

Software Engineering (SE)

One cannot teach it

One can only preach it

- SE is concerned with theories, methods, techniques, and tools for professional, cost effective, software development and management

Software Engineering Definition – IEEE 1993

- The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software;
  that is, the application of engineering to software.

What is Software?

- "Computer **programs**, **procedures**, and possibly associated **documentation** and **data** pertaining to the operation of a computer system."
  - [IEEE Standard Glossary of Software Engineering Terminology]
- Software consists
  - of **informal information** (documents)
  - and **formal information** (code, data)

Nature of Software

- Software is more than coding
  - software: programs, documents, data
  - software is developed/engineered and not manufactured
  - does deteriorate; does not wear out
  - custom built; not enough reuse
- For many: software is art, never become craft

Nature of Software

- Development of a software system is a part of finding a solution to a larger problem which may entail the development of an overall system involving software, hardware, procedures, and organizations

System

A set of inter-related components working together towards some common objective. The system may include software, mechanical, electrical and electronic hardware and be operated by people. [Sommerville]

System

- The proportion of software in systems is increasing
- The properties and behavior of system components are inextricably intermingled
- Systems must be designed to last many years in a changing environment

Hardware History

- Till the early 70th: large, expensive computers (mainframes); small programs; batch oriented; money, speed, i/o constraints
- The 70th: mini computers; real time; databases; software teams

Hardware History

- The 80th: PC's; large programs (software systems); distributed systems; communication
- The 90th: client-server; internet; distributed systems; communication

Software History: Languages

- First generation
  - assembly languages
  - no abstractions used
- Second generation
  - mathematical expressions-formulas (Fortran I)
  - abstraction used: mathematics
  - global data (common)

Software History: Languages

- Third generation
  - subroutines, data handling, blocks, typing (Pascal)
  - abstraction used: function and procedure (algorithm)
  - prominence to behavioral characterization
    - complex behavior is repeatedly decomposed into subcomponents until plausibly implementable units are obtained
- global and internal (subprograms) data

Software History: Languages

- Fourth generation
  - 4GL
- ? Generation
  - classes (C++, Java, Eiffel)
  - abstraction used: data and data models of real-world entities
  - ADT
  - data local to objects
  - focus is on programming in the large

Evolution of Software

- 1st era: 1950's - early 1960's
  - batch orientation, limited distribution, custom software
- 2nd era: mid 1960's - early 1970's
  - multi-user, real-time, database, product software

Evolution of Software

- 3rd era: mid 1970's - mid 1980's
  - distributed systems, embedded intelligence, low-cost hardware, consumer impact
- 4th era: mid 1980's - early 1990's
  - powerful desktop systems, OO technology, parallel computing, expert systems, AI (neural network)

Evolution of Software

- What next?
  - component-based software engineering
  - mega-programming
  - frameworks
  - design patterns
  - reuse on higher-level
  - middleware, client/server
  - parallel computing for general business applications

Software History: Approaches

- Assembly languages
- Structured programming
- Modular programming (not monolithic)
- Structured design (improves program modularization)
- Structured analysis (improves identification and grouping of modules; rooted in third generation languages)

Software History: Approaches

- OO (Programming+Design+Analysis)
- Next???
- Note:

  newer technologies support the older

Software Crisis

Software Crisis

- Example: OS/360 (Brooks)
  - symptoms
  - reasons
- 1 million lines of code
- Error fixed caused new errors
- Adding people was wrong
- Result: OS was completely rewritten

Software Crisis: Symptoms

- Late delivery, over budget
- Product does not meet specified requirements
- Inadequate documentation

return

Software Crisis: Observations

- A malady that has carried on this long must be called normal [Booch]
- Software system requirements are moving targets
- There may not be enough good developers around to create all the new software that users need

Software Crisis: Observations

- A significant portion of developers time must often be dedicated to the maintenance or preservation of geriatric software
- As programs grow
  - they are more difficult to implement
  - subtle usage bugs get introduced
  - reliability decreases

Software Crisis: Observations

- Usually, no reuse of past experience to predict schedule and costs (the experienced left: became managers, other projects, etc.)
- Customers needs only vaguely understood (hence, never satisfied with the delivered software ...)

Software Crisis: Observations

- No solid software quality metric (software quality is a human problem)
- Difficult to maintain software

NATO conference (Garmisch-Partenkirchen 1968):

shouldn't it be possible to build software in the way one builds bridges and houses?

Software Engineering

- At the NATO conference F.L. Bauer said:
  - The whole trouble comes from the fact that there is so much tinkering with software
  - It is not made in a clean fabrication process, which it should be
  - What we need, is Software Engineering
- Note
  - The word was invented in a provocative sense
- It took some five or eight years before the term was generally accepted
  - 1977, the ICSE was started, and the IEEE Transactions on Software Engineering were launched

Software Engineering

- Software engineers should
  - adopt a **systematic** and organized approach to their work and
  - use **appropriate tools** and **techniques** depending on
    - The **problem** to be solved
    - The development **constraints**
    - The **resources** available

The Goal of Software Engineering

- Our ultimate goal is to develop software which (as a product or part of a product or an internal aid) helps us to be **successful** in the market

The Goal of Software Engineering

- We can break down that goal to three sub-goals: we want to solve the problems
  - Effectively — effective
  - Within short time — fast
  - At little effort (cost) — cheap
- Furthermore, we want to know early that we will be successful, i.e.
  - We want to plan the date of delivery, the cost, and the quality of the product

The Goal of Software Engineering

- A company will hopefully have more contracts after a project
- Therefore, the project should improve the position in the market by
  - Establishing a **positive image**
  - Building up **know how**
  - Developing or improving **reusable** software

Software Caused Disasters

- US study (1995) 81 milliard US$ spend per year for failing software development projects
- European Space Agency Ariane 5 (1996)
  - track control system failure results in self destruction
  - http://www.esrin.esa.it/htdocs/tidc/Press/Press96/ariane5rep.html

Software Caused Disasters

- NASA Mars Climate Orbiter (1999)
  - incorrect conversion from imperial to metric leads to loss of Mars satellite
- The Therac-25
  - a radiation therapy machine that massively overdosed five people over a period of two and a half years

Software Caused Disasters

- Denver Airport
  - telecars were misrouted and crashed
  - baggage was lost and damaged
  - without baggage handling, the airport couldn’t open, costing $1.1M per day
  - late delivery of software for the baggage system delays the opening of the airport by 16 months

Software Caused Disasters

- Core System 90 (CS90) for Westpac Banking Corporation (Australia)
  - going live with an essentially untested system
  - Waste: $250 Million
- See:

  http://sunnyday.mit.edu/accidents/

  http://catless.ncl.ac.uk/Risks/

Reasons for Late Delivery and High Costs

- Unrealistic deadlines forced from the outside
- Changed requirements not reflected in schedule
- Underestimation of effort
- Risks not considered
- Unforeseen technical and human difficulties
- Miscommunication among project staff
- Management fails to see that the project falls behind schedule

SE Characteristics

Essential Characteristics of Software Engineering

- Construction of large programs
- DeRemer:
  - programming-in-the-small: 1 person, short period, tools exist
  - programming-in-the-large: multi person, more than 1/2 year
  - productivity of large programs falls sharply: difficulty to find and fix problems increases exponentially

Essential Characteristics of Software Engineering

- Mastering complexity
- Team work
- Software evolves
- Software development efficiency is of crucial importance
- Software has to effectively support users

Software vs Hardware

- Both start with requirements and end with a product

but

- Software constructing cost is incurred during development and not during production
  - copying software is almost free
- Software is logical in nature rather than physical

Software vs Hardware

- Software maintenance:

  error corrections, new requirements, etc.
- Software reliability is determined by the manifestation of errors already present

What are Software Engineering Methods?

- Structured approaches to software development which include system models, notations, rules, design advice and process guidance **[Sommerville]**

What are Software Engineering Methods?

- Model descriptions
  - Descriptions of (graphical) models which should be produced
- Rules
  - Constraints applied to system models;
- Recommendations
  - Advice on good practice
- Process guidance
  - What activities to follow

The Essence of SE

- Architecture
  - A foundation of concepts and techniques, selected from the universe of approaches
  - Set of modeling techniques (one for each model)
  - The denotation of the set of modeling techniques
- Method
  - Comprise the techniques used to perform the various phases of the software development
  - A method is a planned procedure by which a specified goal is approached step by step
    - Methodology is the science of methids
  - Includes: models, techniques, tools

The Essence of SE

- A method consists of:
  - Notation with associated semantics
    - E.g circle defines a process
  - Procedure/pseudo-algorithm/recipe for applying the notation
  - Criterion for measuring progress and deciding to terminate
- Methods are often unique to the organization

Methods

- **Ad hoc**
- **Functional** Decomposition and Structured Methods
  - Methods which are based on functional decomposition and stepwise refinement
  - Breaking down of complex systems into single-function tasks and subtasks

Methods

- Data-Structured Methods
  - Methods which are based on decomposition of complex data
  - Used for systems that tend to be mostly databases and the modification of how that data was presented to the user
- Formal Methods
- OO methods

The Essence of SE

- Models
  - DFD, ERD, UML…
- Techniques
  - How to create models
  - Techniques examples:
    - SA, SD, SP, strategic planning, project managing, user interviewing, data modeling

The Essence of SE

- Automated tools can support methods
- Common phenomenon
  - First, tools are selected, then try to apply them to method or adapt methods to tools
- Procedures/processes
  - Formal, best if documented, activities performed at the various phases
  - Best if automated

The Essence of SE

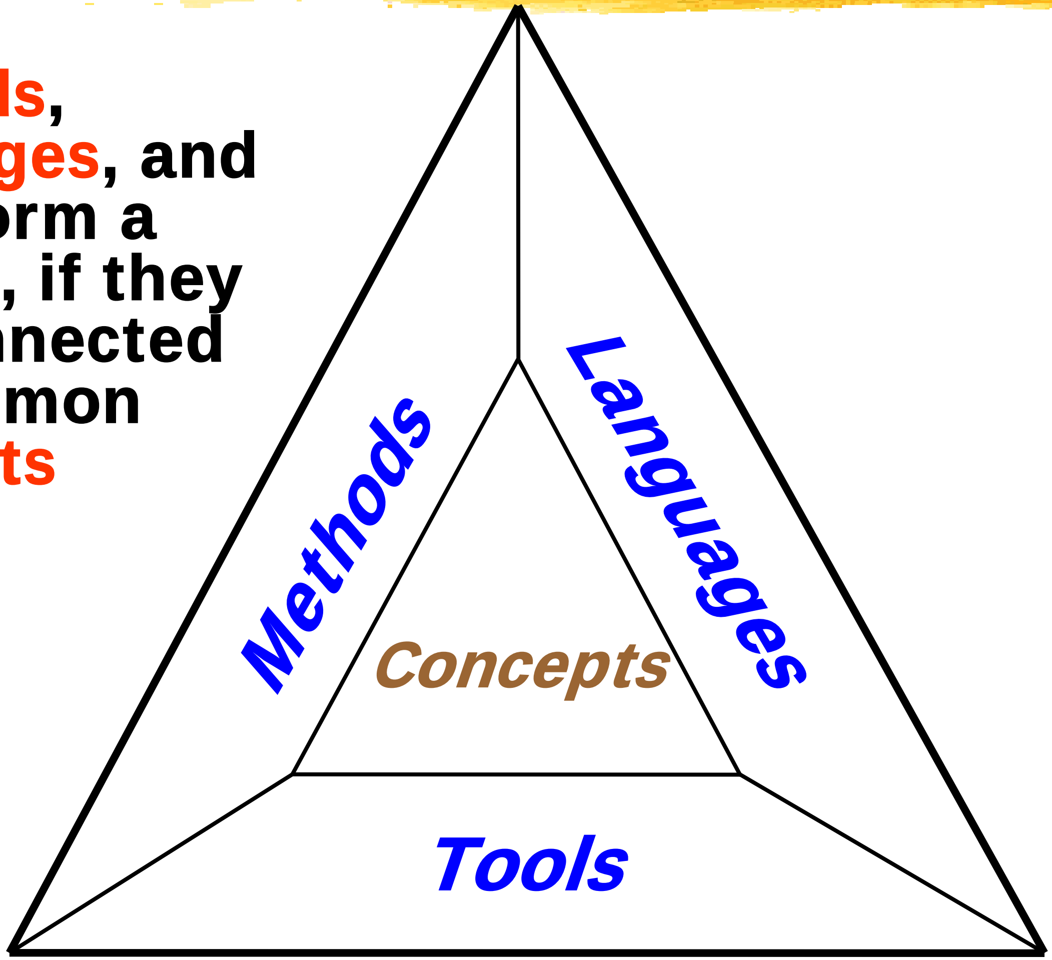
- Software process
  - refers to the defined processes for management and development of software as standard software engineering practice
- Management
- Quality assurance
- Configuration management

The Essence of SE

- Measurement
- Innovative technology insertion
- CASE tools
  - can be valuable aid
  - introduce a high degree of risk if organization is immature in its methods
- ...

The System Triangle

- **Methods**, **languages**, **and tools form a system, if they are connected by common concepts**



Why is Software Complex?

- Usually problem domain is complex, as well as managing the development process
- Too much flexibility
- Discrete systems (changes are not continuous; state explosion)
- Large and complex systems exhibit a rich set of behaviors

Why is Software Complex?

- Complex systems need team work
- Not thrown away, hence patched, out-of-date environments and documentation
  - systems are expected to live long and used by many people
- Complex systems are implemented in an hierarchical structure
  - the determination of the hierarchy is relatively arbitrary

Why is Software Complex?

- Working complex system have invariably evolved from working simpler systems
- Software systems must fulfill the requirements of a client

Software Quality

Quality Factors

correctness
robustness
extendibility
reusability
compatibility

efficiency
portability
ease of use
functionality
timeliness

Modularity is a key technique to achieve quality

Software Quality Factors

- Correctness
  - The ability to perform the exact tasks as defined by requirements and specification
    - the prime quality
- Robustness
  - The ability to react appropriately even in abnormal conditions
- Dependability
  - No physical/economical damage when a failure occurs

Software Quality Factors

- Reliability
  - (IEEE) the probability that software will not cause the failure of a system for a specified time under specified conditions (measures with mean time to failure; availability)

correctness + robustness —> reliability

Software Quality Factors

- Compatibility
  - The ease of combining software elements with other products
    - Software products need to interact with each other, but they too often have trouble interacting because each product makes conflicting assumptions about the rest of the world
  - Keys to improve compatibility:
    - Design homogeneity
    - Standardized conventions (e.g. file formats, data structures, interfaces)

Software Quality Factors

- Maintainability
  - The effort required to locate and fix an error
    - Software should evolve to meet changing requirements
- Testability
  - The effort required to test a program/system

Software Quality Factors

- Extendibility
  - The ease of adaptation to changes in specifications (also called flexibility)
    - Programming-in-the-large phenomenon: as programs grow bigger, they become harder and harder to adapt
  - Keys (principles) to improve it:
    - Design simplicity
      - A simple architecture will always be easier to adapt to changes than a complex one
    - Decentralization
      - the more autonomous the modules in a software architecture, the higher the likelihood that a simple change will affect just a small number of modules

Software Quality Factors

- Efficiency
  - In use of resources (place few demands on resources)
  - Synonymous with *performance*
  - But: make it right before you make it fast
  - Efficiency must be balanced with goals like extendibility and reusability

Software Quality Factors

- Usability
  - Appropriate UI and documentation
    - Ease of use
    - Training time and number of help frames
- Portability
  - The ease of transferring software products to various environments
    - Measured with percentage of target dependent statements

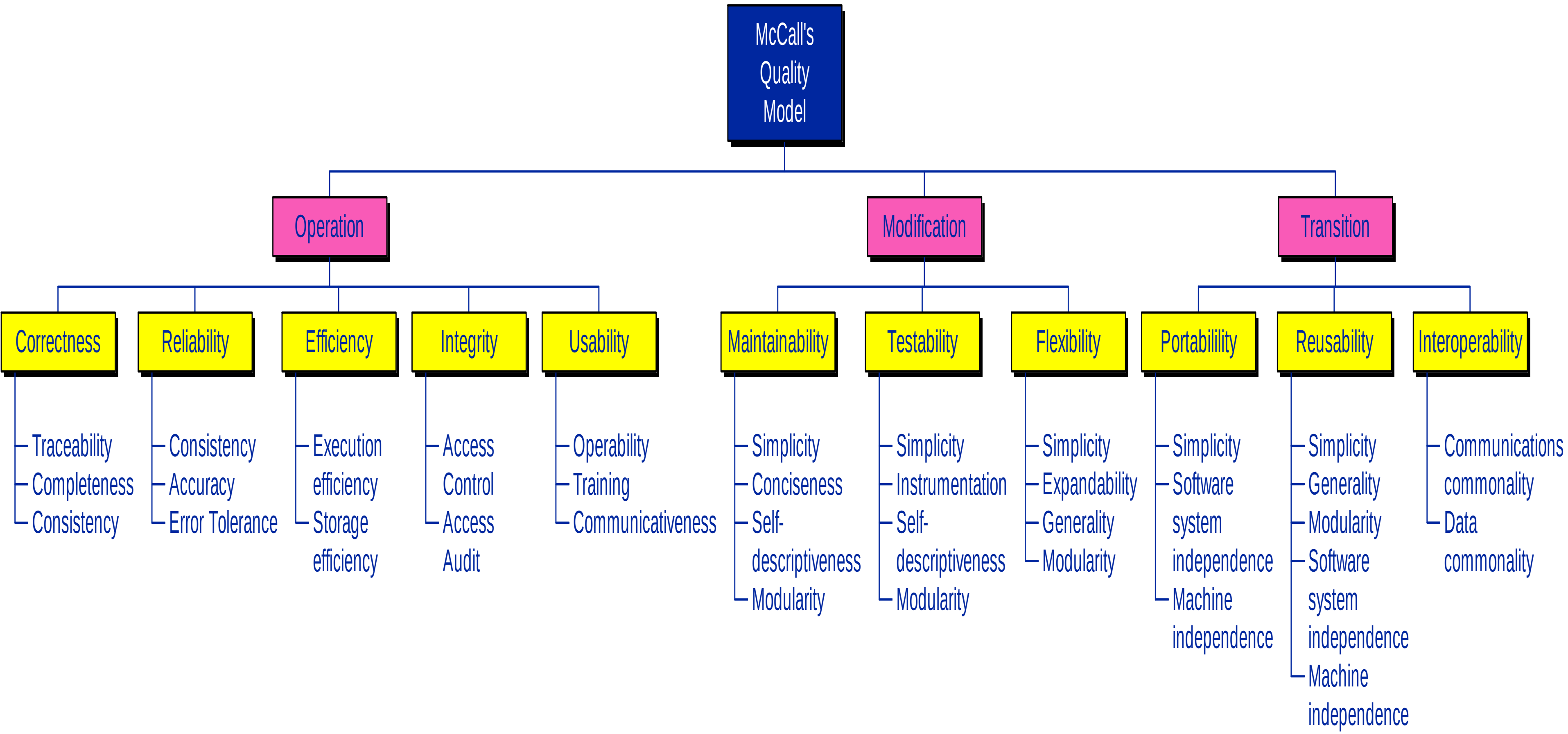
Software Quality Factors

- Interoperability
  - The effort required to couple one system with another
- Reusability
  - The ability to be reused (in whole or in part) in new applications
- Functionality
  - The extent of possibilities (*features*) provided by the system

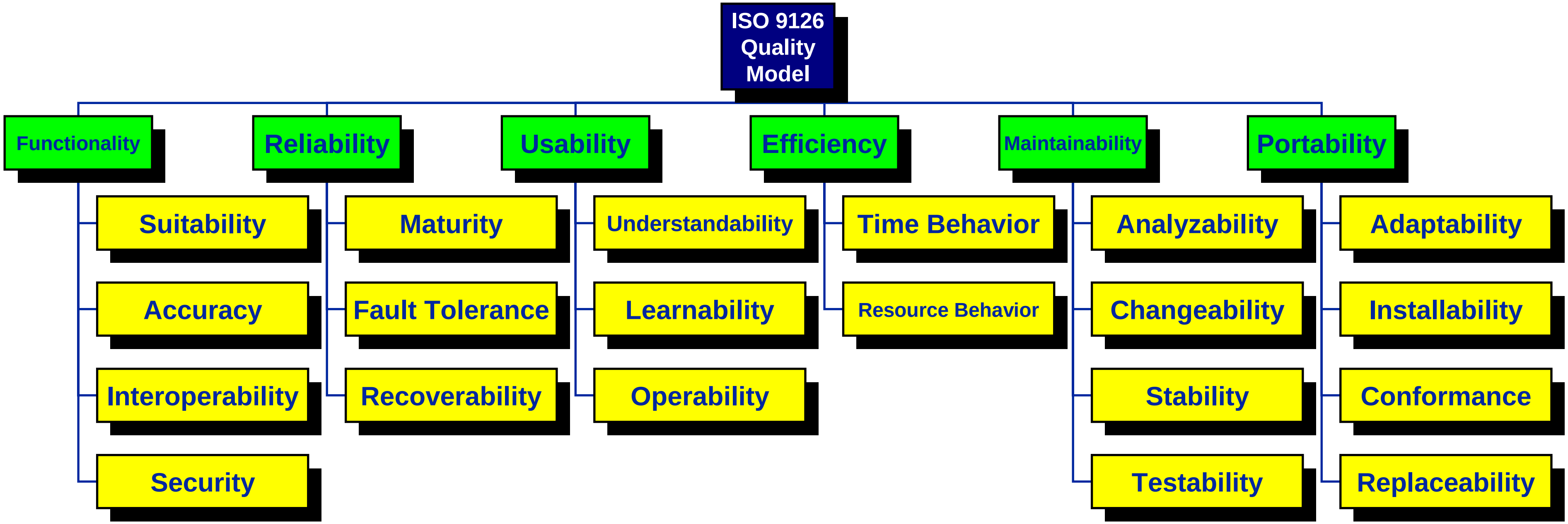
Software Quality Factors

- Verifiability
  - The ease of preparing acceptance procedures
- Integrity
  - The ability to protect components against unauthorized access

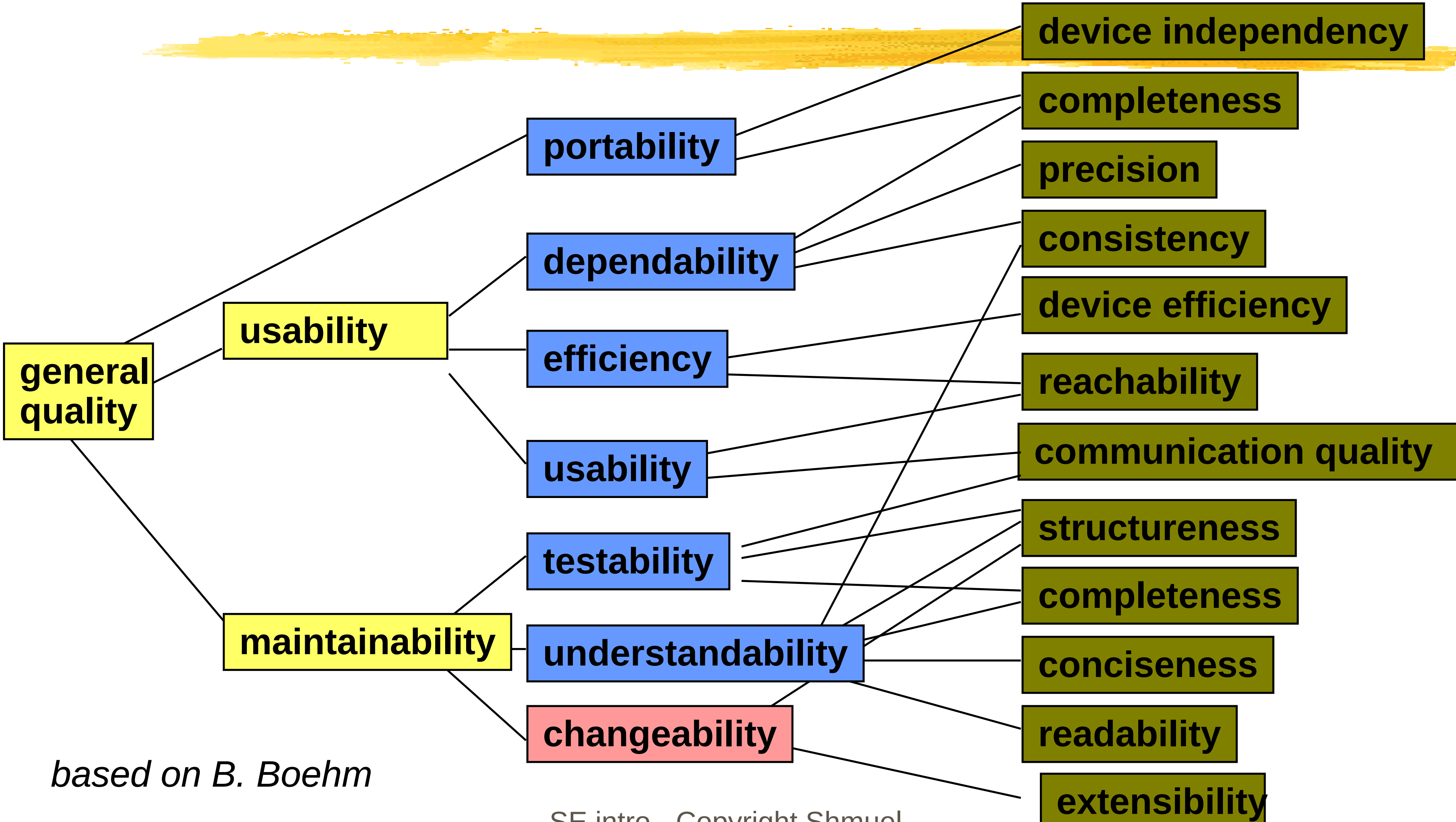
McCall's Quality Model



ISO 9126 Quality Model



Boehm's Quality Model



based on B. Boehm

The Problem in SE

The pursuit of quality is the central problem in Software Engineering

- External quality factors

  factors observable by the user

  - correctness
  - robustness
  - ....

The Problem in SE

- Internal quality factors

  factors observable by the SW engineer

  - readability
  - extendibility
  - reusability
  - type-safety
  - ...

Tradeoffs Between Quality Factors

- Examples
  - integrity vs ease of use
  - efficiency vs portability
  - efficiency vs reusability
- The relative importance of quality characteristics depends on the product and the environment (e.g. RTS)
- Costs may rise exponentially if very high levels of any one attribute are required

Documentation

- The need for documentation is a consequence of the other quality factors
  - external documentation: users can understand the system (ease of use)
  - internal documentation: developers understand the structure and implementation (extendibility)
  - module interface documentation: developers understand the (black-box) functionality (reusability)