CS 563:
Software Maintenance and Evolution

Software Quality

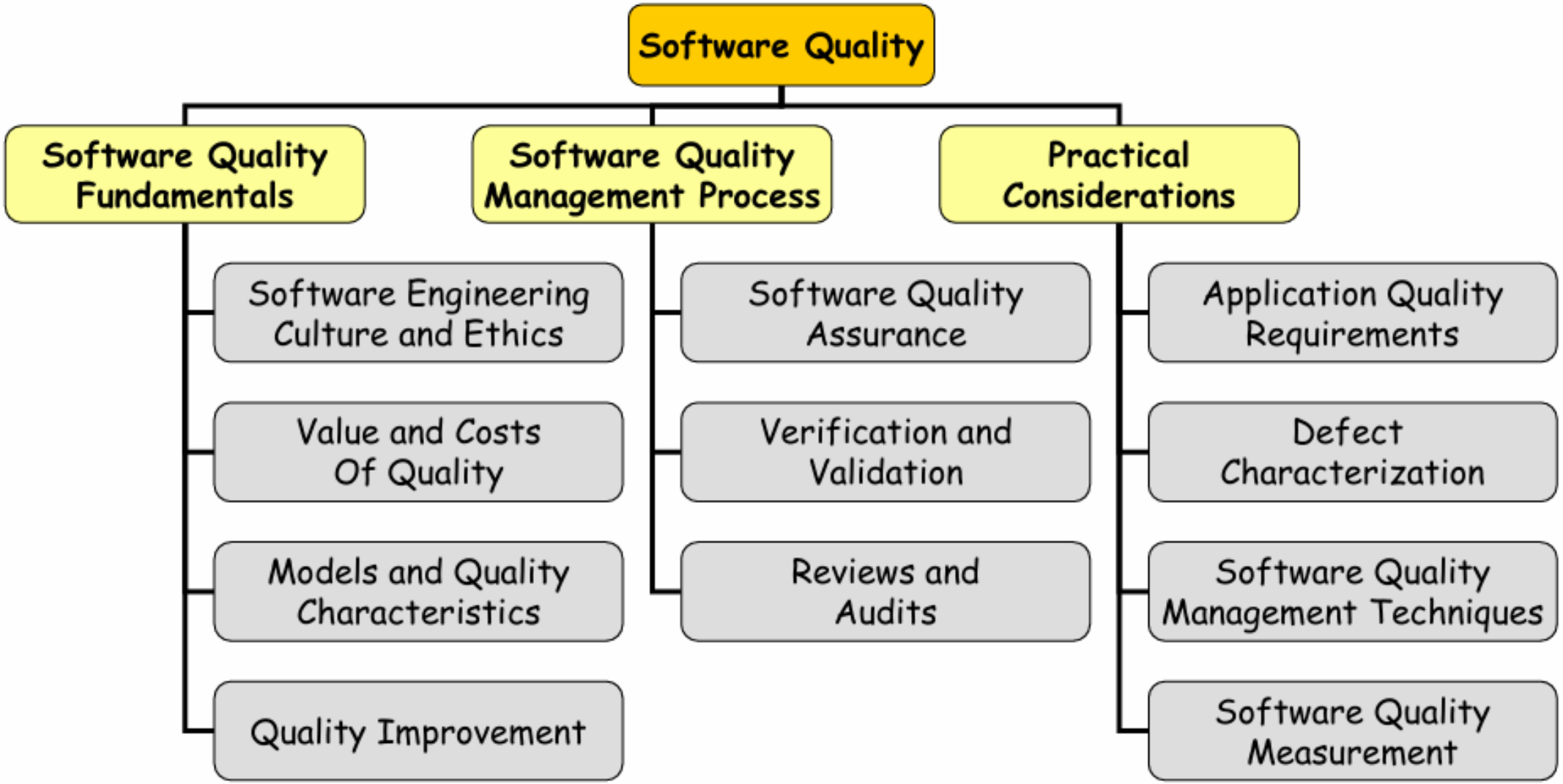
Oregon State University, Spring 2024

Today's Plan

Learn about:

- Software Quality
- "how to present a paper" and "how to write a paper review" (helpful for paper presentation related assignments due by the end of this week and next week)

Software Quality At A Glance



Software Quality Fundamentals

- **Commitment** to **Software Quality** as part of **organization culture**
- The notion of "quality" is difficult to define
  - 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
- The link between software process quality and software product quality is complex

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
- 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**
  - Planning the processes to achieve the required product
- 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*

Software Quality Management 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 through and Audits

Quality Planning

- **Quality planning:** a 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

Software Quality Assurance

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

Software Quality Assurance Approaches

- Software quality defect management approach
  - Focuses on counting and managing defects
  - Use tools like defect leakage matrices and control charts to measure and enhance SDLC
- Software quality attributes approach
  - Focuses on directing the engineer's attention to several quality factors:
    - Reliability - system's ability to continue operating over time under different working environments and conditions
    - Usability - how easy it is to learn and navigate the application.
    - Efficiency - how well the system uses all the available resources, shown by the amount of time the system needs to finish any task
    - Maintainability - how easy it is to maintain different system versions and support changes and upgrades cost-effectively
    - Portability - system's ability to run effectively on various platforms — for example, data portability, viewing, hosting, and more

SQA Approaches in Industry

- Traditional
  - SQA is performed at the end of each phase in Waterfall SDLC model
- Agile
  - Focuses on self-organizing teams, continuous integration and testing, continuous delivery, and continuous feedback to ensure a high-quality software product.
- DevOps
  - Combination of development and IT operations
  - Emphasizes collaboration, automation, and continuous delivery to deliver the software product quickly and efficiently.
- Six Sigma
  - Data-driven approach that focuses on reducing defects and errors
  - Uses statistical tools and techniques to measure and improve the software quality

SQA Approaches in Industry (Contd.)

- Lean
  - Focuses on efficiency and waste reduction in the software development process.
  - Emphasizes continuous improvement and the elimination of non-value-added activities.
- Continuous integration and continuous deployment (CI/CD)
  - Focuses on continuous integration and deployment of software products.
  - Emphasizes automation, continuous testing, and continuous delivery of software products.
- Test-driven development (TDD)
  - Involves writing tests before writing the code to ensure that the code meets the requirements and specifications of the software product
- Risk-based approach
  - Involves risk assessment, risk mitigation, and risk monitoring to ensure that the software product meets established standards.

QUIZ: Code Reviews

What changes (if any) would you suggest in the following code?

```
public class Account {
  double principal,rate; int daysActive,accountType;
  public static final int STANDARD = 0, BUDGET=1,
      PREMIUM=2, PREMIUM_PLUS = 3;
  } ...

  public static double calculateFee(Account[] accounts)
  {
      double totalFee = 0.0;
      Account account;
      for (int i=0;i<accounts.length;i++) {
          account=accounts[i];
          if ( account.accountType == Account.PREMIUM ||
          account.accountType == Account.PREMIUM_PLUS )
            totalFee += .0125 * (   // 1.25% broker's fee
            account.principal * Math.pow(account.rate,
            (account.daysActive/365.25))
            - account.principal);   // interest-principal
      }
      return totalFee;
   }
}
```

QUIZ: Code Reviews

What changes (if any) would you suggest in the following code?

```
public class Account {
  double principal,rate; int daysActive,accountType;
  public static final int STANDARD = 0, BUDGET=1,
      PREMIUM=2, PREMIUM_PLUS = 3;
  } ...

  public static double calculateFee(Account[] accounts)
  {
      double totalFee = 0.0;
      Account account;
      for (int i=0;i<accounts.length;i++) {
          account=accounts[i];
          if ( account.accountType == Account.PREMIUM ||
          account.accountType == Account.PREMIUM_PLUS )
            totalFee += .0125 * (   // 1.25% broker's fee
            account.principal * Math.pow(account.rate,
            (account.daysActive/365.25))
            - account.principal);   // interest-principal
      }
      return totalFee;
   }
}
```

- Add comments
- Make fields private.
- Replace magic values (e.g., 365.25) with constants.
- Use an ***enum*** for account types.
- Use consistent whitespace, line breaks, etc.

QUIZ: Code Reviews

Improved Code

```
/** An individual account. Also see CorporateAccount. */
public class Account {
    /** The varieties of account our bank offers. */
    public enum Type {STANDARD, BUDGET, PREMIUM, PREMIUM_PLUS}

    /** The portion of the interest that goes to the broker. */
    public static final double BROKER_FEE_PERCENT = 0.0125;

    private Type type;
    private double principal;

    /** The yearly, compounded rate (at 365.25 days per year). */
    private double rate;

    /** Days since last interest payout. */
    private int daysActive;

    /** Compute interest on this account. */
    public double interest() {
        double years = daysActive / 365.25;
        double compoundInterest = principal * Math.pow(rate, years);
        return compoundInterest - principal;
    }
    ...
```

- Add comments
- Make fields private.
- Replace magic values (e.g., 365.25) with constants.
- Use an ***enum*** for account types.
- Use consistent whitespace, line breaks, etc.

QUIZ: Code Reviews

Improved Code

```
    ...
    /** Return true if this is a premium account. */
    public boolean isPremium() {
        return accountType == Type.PREMIUM ||
               accountType == Type.PREMIUM_PLUS;
    }

    /** Return the sum of broker fees for all given accounts. */
    public static double calculateFee(Account[] accounts) {
        double totalFee = 0.0;
        for (Account account : accounts) {
            if (account.isPremium()) {
                totalFee += BROKER_FEE_PERCENT * account.interest();
            }
        }
        return totalFee;
    }
}
```

- Add comments
- Make fields private.
- Replace magic values (e.g., 365.25) with constants.
- Use an ***enum*** for account types.
- Use consistent whitespace, line breaks, etc.

Software Metrics

What is **Measurement**?

Software Metrics

What is **Measurement**?

It 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**

Software Metrics

- What is **Measurement**?
  It 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 (recall the cobra effect!)

Software Metrics

- What is **Measurement**?
  It 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 (recall the cobra effect!)
- **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 Basics of Measurement

- What is **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* to characterize its **attribute**
  - **Direct** or **Indirect** measurement
  - NOTE: 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
- **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 behavior
- **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 behavior 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 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.
- **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**.

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, then 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 these correct?
  - **Accuracy:** are they accurate?
  - **Precision:** are they appropriately precise?
  - **Consistency**: Are they consistent?
  - Are they replicable?
  - Are they associated with a particular activity or time period?

Software Metrics Data Collection

- **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 behavior.

    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**: strength of beam to 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

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 disjoint sets of local methods

Measuring External Product Attributes

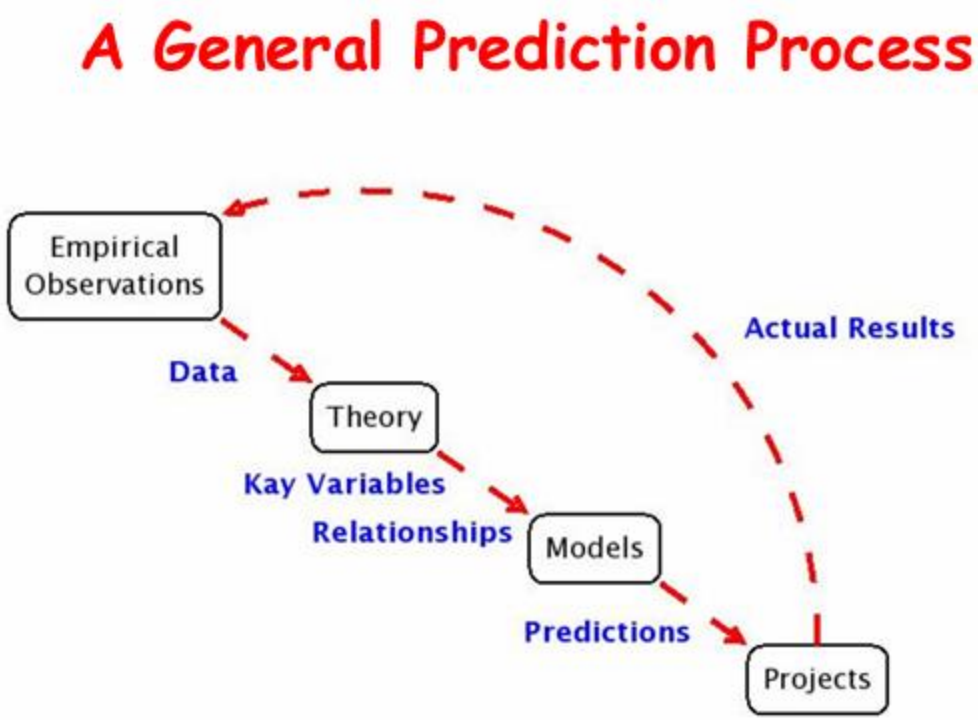
- Modelling Software Quality
- The ISO 9126 standard quality model: **functionality, reliability, efficiency, usability, maintainability** and **portability**
- **Note**: Safety is a system property, not a software quality characteristic
- An example: **Usability** of a software product is the extent 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

- 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 consider 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



Announcements

- **Paper Review** assignment is released today and is due on **Sunday, May 19, 11:59 PM**
- **Final Project** assignment is released today and is due on **Wednesday, June 5, 11:59 AM**
- Next class, we will have project stand-up meetings.
- For students presenting papers on Monday, try to incorporate the best practices while creating your presentation and if possible, create a draft of your presentation to get early feedback in the next class.

In-Class Exercise: How to review a research paper?

Issues

- Will this advance the state of the art?
- Did you learn anything new?
- Does it provide evidence which supports/contradicts hypotheses?
- Experimental validation?
- Will the paper generate discussion at the conference?
- How readable is the paper? (The draft can be modified, and if the ideas are very important, you may accept it anyway.)
- Is the paper relevant to a broader community?

Structure

- 1/2 to 1 page of text (2 - 4 paragraphs)
  Longer reviews are generally given for better papers, shorter reviews for bad papers
- 1 paragraph executive summary
  - what is the paper trying to do?
  - what is potential contribution of paper?
  - short summary of strengths and weaknesses (sentence or 2)
  - accept/reject (hard, because you don't necessarily see the entire sample)
- several paragraphs of details (listed in order of importance)
  - technical flaws?
  - structure of paper?
  - are key ideas brought out?
  - don't want to just describe system, also need motivation and justification of approach
  - presentation? (ex: undefined terms, confusing wording, unclear sections...)
  - justification -- can they say why ideas are important?
  - comparison with other systems? For bigger conferences (SOSP, ISCA, ASPLOS) need quantitative evidence of ideas
  - grammar? (usually only point out consistent errors)

Source: https://people.eecs.berkeley.edu/~fox/paper_writing.html#rev

In-Class Exercise: How to present a research paper?