CITS5501 Software Testing and Quality Assurance
Risk, standards and metrics

Unit coordinator: Arran Stewart

Risk

Motivation

Why do we look at risk?

- Because every project, even the simplest, involves risks

Motivation

Why do we look at risk?

- Because every project, even the simplest, involves risks
- Because people are usually not good at allowing for them

Motivation

Why do we look at risk?

- Because every project, even the simplest, involves risks
- Because people are usually not good at allowing for them
- Because risk assessment determines what other testing and quality assurance activities you undertake

Example

- What sort of requirements documentation should we require for, say, a website?
- Is there complicated or confusing terminology involved? Is there a risk of miscommunication or misunderstanding between us and our client? (e.g. perhaps for a website to be used internally by a financial services enterprise)
  We might try to reduce the risk of miscommunication by ensuring we have *glossaries* of terms, and that client representatives and developers agree on their understanding of these.

Example

- Is the domain, or the technologies we're likely to have to use, complicated? Is there a risk we have modelled it incorrectly? (Example: software to manage telephone exchange systems; systems which are inherently concurrent, as they are notoriously difficult to reason about.)
  We might reduce risk by putting additional effort into modelling the domain and/or the system – perhaps through a notation such as UML, perhaps even through formal (i.e. mathematical) specifications of the domain and/or system.

Example

- On the other hand, if this is a website for a local kids' football club, then many of these risks are unlikely to be relevant, or to cause significant problems if they do – so effort on those activities would be misplaced.

What is risk?

- Something that *might* happen, which would cause loss
  - If it will *definitely* happen, that's not a risk – it's usually called a "constraint"
  - If it doesn't result in some kind of unwanted consequence (i.e., a loss), it's also not a risk

What are some sorts of risk?

One categorization (Pressman):

- Project risks
  - Things that could affect the project plan or schedule
- Technical risks
  - Risks resulting from the problem being *harder to solve* than we thought
- Business risks
  - Things that threaten the viability, as a product, of the software to be built
  - (Could be commercial viability, but also applies to in-house or even open source software)

Sorts of risk – project risks

Project risks:

- Things that could affect the *project plan and/or schedule*, causing it to slip and costs to increase
- Examples: personnel move or resign, resources turn out to be more expensive/take longer to acquire than estimated, exchange rates shift

Sorts of risk – technical risks

Technical risks:

- Risks resulting from the problem being *harder to solve* than we thought
- They threaten the quality and timeliness of the software to be produced
- If they eventuate, implementation may become difficult or impossible
- Examples: design turns out to be infeasible, dependencies have bugs, language/framework turns out to be difficult to maintain

Sorts of risk – business risks

Business risks - Things that threaten the viability, as a product, of the software to be built (even if there are no technical or project barriers to it being implemented on time and within budget) - Examples: over-estimating the market for a product; being beaten to release by a competitor

What are some sorts of risk? (2)

Another (due to Robert N. Charette, 1989):

- Known risks
  - Those risks that can be uncovered after careful evaluation of the project plan, the business and technical environment in which the project is being developed, and other reliable information sources (e.g., unrealistic delivery date)
- Predictable risks
  - Those risks that are extrapolated from past project experience (e.g., past turnover)
- Unpredictable risks
  - Those risks that can and do occur, but are extremely difficult to identify in advance

Reactive vs. proactive risk strategies

- *Reactive* risk strategies
  - "Don't worry, I'll think of something"
  - The majority of software teams and managers rely on this approach
  - Nothing is done about risks until something goes wrong
    - The team then flies into action in an attempt to correct the problem rapidly ("fire fighting")
- *Proactive* risk strategies
  - Steps for risk management are followed
  - Primary objective is to *reduce* risk and to have a contingency plan in place to handle unavoidable risks in a controlled and effective manner

Steps for risk management

1. Identify possible risks; recognize what can go wrong
2. Analyze each risk to estimate the probability that it will occur and the impact (i.e., damage) that it will do if it does occur
3. Rank the risks by probability and impact
   - Impact may be negligible, marginal, critical, and catastrophic
4. Develop a plan to manage (some of) those risks

Risk identification

- A systematic attempt to identify risks to the project
- Because if they aren't identified, how can they be planned for?
- Two main sorts of risk:
  - Generic risks
    - Risks that are common to every software project
  - Product-specific risks
    - Risks that can be identified only by those a with a clear understanding of the technology, the people, and environment specific to the software that is to be built

Risk identification (2)

- One way of identifying risks – use a *risk checklist*
- Focuses on known and predictable risks in specific subcategories

Risk checklists

Some typical categories of items on risk checklists:

- Product size – risks associated with overall size of the software to be built
- Business impact – risks associated with constraints imposed by management or the marketplace
- Customer characteristics – risks associated with sophistication of the customer and the developer's ability to communicate with the customer in a timely manner
- Process definition – risks associated with the degree to which the software process has been defined and is followed
- Development environment – risks associated with availability and quality of the tools to be used to build the project
- Technology to be built – risks associated with complexity of the system to be built and the "newness" of the technology in the system
- Staff size and experience – risks associated with overall technical and project experience of the software engineers who will do the work

A project risk questionnaire

1. Have top software and customer managers formally committed to support the project?
2. Are end-users enthusiastically committed to the project and the system/product to be built?
3. Are requirements fully understood by the software engineering team and its customers?
4. Have customers been involved fully in the definition of requirements?
5. Do end-users have realistic expectations?
6. Is the project scope stable?
7. Does the software engineering team have the right mix of skills?
8. Are project requirements stable?
9. Does the project team have experience with the technology to be implemented?
10. Is the number of people on the project team adequate to do the job?
11. Do all customer/user constituencies agree on the importance of the project and on the requirements for the system/product to be built?

Risk estimation

- Risk estimation attempts to rate, for each risk:
  - The probability that it eventuates
  - The magnitude of loss associated with the risk, should it eventuate
- The project planner, managers, and technical staff perform risk estimation so that risks can be prioritized, and limited resources be allocated where they will have the most impact

Risk tables

- A *risk table* provides a project manager with a simple technique for risk projection
- It consists of five columns
  - Risk summary – short description of the risk
  - Risk category – (see the slides on risk categorization)
  - Probability – estimation of risk occurrence based on group input
  - Categorization of impact – e.g. (1) catastrophic (2) critical (3) marginal (4) negligible
  - RMMM – pointer to a paragraph in the Risk Mitigation, Monitoring, and Management Plan

Developing a Risk Table

- List all risks in the first column (by way of the help of the risk item checklists)
- Mark the category of each risk
- Estimate the probability of each risk occurring
- Assess the impact of each risk based on an averaging of the four risk components to determine an overall impact value
- Sort the rows by probability and impact in descending order
- Draw a horizontal cutoff line in the table that indicates the risks that will be given further attention

Assessing risk impact

- Three factors affect the consequences that are likely if a risk does occur
  - Its nature – This indicates the problems that are likely if the risk occurs
  - Its scope – This combines the severity of the risk (how serious was it) with its overall distribution (how much was affected)
  - Its timing – This considers when and for how long the impact will be felt
- The overall risk exposure formula is RE $=$ P $\times$ C, where
  - P $=$ the probability of occurrence for a risk
  - C $=$ the cost to the project should the risk actually occur
- Example
  - P $=$ 80% probability that 18 of 60 software components will have to be developed
  - C $=$ Total cost of developing 18 components is \$25,000
  - RE $= .80 \times$ \$25,000 $=$ \$20,000

Risk mitigation, monitoring, and management

An effective risk strategy for dealing with risk must consider three issues

- (Note: these are not mutually exclusive)
  - Risk mitigation (i.e., avoidance)
  - Risk monitoring
  - Risk management and contingency planning
- Risk mitigation (avoidance) is the primary strategy and is achieved through a plan

Example

Example: Risk of high staff turnover

- Meet with current staff to determine causes for turnover (e.g., poor working conditions, low pay, competitive job market)
- Mitigate those causes that are under our control before the project starts
- Once the project commences, assume turnover will occur and develop techniques to ensure continuity when people leave
- Organize project teams so that information about each development activity is widely dispersed
- Define documentation standards and establish mechanisms to ensure that documents are developed in a timely manner
- Conduct peer reviews of all work (so that more than one person is "up to speed")
- Assign a backup staff member for every critical technologist

Standards and quality control systems

Overview

Sources: based on material from:

- Stephen Dannelly, Winthrop University
- Pressman, R., *Software Engineering: A Practitioner's Approach*, McGraw-Hill, 2005

Overview:

- If I am working in an organization that adheres to the ISO 9001 standard, what is involved in that?
- What is the Capability Maturity Model?
- When should we adopt one of these quality models?

What ISO 9000 mandates

ISO:

Organizations vary greatly – in revenue, number of emnployees, industry, degree of regulation etc.

How does ISO 9001:2008 allow for the diversity of say, on the one hand, a "Mon and Pop" enterprise, and on the other, to a multinational manufacturing company with service components, or a public utility, or a government administration?

The answer is that ISO 9001:2008 lays down **what** requirements your quality system must meet, but does **not** dictate **how** they should be met in any particular organization. This leaves great scope and flexibility for implementation in different business sectors and business cultures, as well as in different national cultures.

Insuring compliance

ISO:

1. The standard requires the organization itself to audit its quality system to verify that it is managing its processes effectively - or, to put it another way, to check that it is fully in control of its activities.
2. In addition, the organization may invite its clients to audit the quality system in order to give them confidence that the organization is capable of delivering products or services that will meet their requirements.
3. Lastly, the organization may engage the services of an independent quality system certification body to obtain an ISO 9001:2008 Certificate of Conformity. This last option has proved extremely popular in the market-place because of the perceived credibility of an independent assessment.

ISO 9001 Contents

- Section 4 — General Requirements
- Section 5 — Management Responsibility
- Section 6 — Resource Management
- Section 7 — Product Realization
- Section 8 — Measurement, Analysis and Improvement

ISO 9003

- ISO/IEC 90003 – "Software engineering – Guidelines for the application of ISO 9001:2008 to computer software"
- Generally, mirrors the structure of ISO 9001

ISO 90003

Section 7 - Product Realization

```
7.1 Product Realization Planning
7.2 Customer Processes
  7.2.2 Review of Software Product Requirements
     7.2.2.1 Review Product Requirements related to
             Customer Contract
7.3 Software Design and Development
7.4 Purchasing Parts and Components
7.5 Product and Service Provisions
    * tracking builds, deliveries, releases
7.6 Monitoring and Measuring
```

see http://www.praxiom.com/iso-90003.htm

ISO 90003

Section 8 - Measurement, Analysis, and Improvement

8.1 Carry out remedial processes
  * Plan how monitoring, measuring, and analytical processes will be used to demonstrate conformity.
  * Use monitoring, measuring, and analytical processes to demonstrate conformance.

8.2 Monitor and measure quality
  8.2.1 Monitor and measure customer satisfaction.
  8.2.2 Plan and perform regular internal audits.
  8.2.3 Monitor and measure quality processes.
  8.2.4 Monitor and measure product characteristics.

8.3 Control your nonconforming software products
  * Prevent the delivery or use of nonconforming software products.

8.4 Analyze quality information

8.5 Take required remedial actions

see: http://www.praxiom.com/iso-90003.htm

9001 Required Documents

1. Quality Policy
2. Control of Documents
3. Control of Records
4. Internal Audits
5. Control of Nonconforming Product / Service
6. Corrective Action
7. Preventive Action

These may go in a single “Quality Manual”.

see http://en.wikipedia.org/wiki/ISO_9000

Documentation

- **Quality policy**
  - intended for all levels of employees
  - linked to business plan, marketing plan, customer needs
  - measurable objectives
- **Records**
  - allows problems to be traced back to causes
  - includes
    - test results, customer comments, etc.
    - actions taken to improve
- **Internal Audits**
  - is the system working?
  - what improvements can be made?

Is it worth it?

- Good business judgment is needed to determine ISO9001's proper role for a company.
- Is certification important to the marketing plans of the company? If not, do not rush to certification.
- Even without certification, companies should utilize the ISO 9001 model as a benchmark to assess the adequacy of its quality programs.

— Frank Barnes

Capability Maturity Model

- **1986** - Effort started by SEI and MITRE Corporation
  - assess capability of DoD contractors
- First version published in 1991
- closely related to **TQM** ("Total quality management")
  - goal is customer satisfaction
    - not required that customer be "delighted"
- Process improvement is based on small steps, rather than revolutionary innovation.
- CMM is not exhaustive or dictatorial.
- CMM focuses on processes that are of value across the organization.

Levels

1. Initial
2. Repeatable
3. Defined
4. Managed
5. Optimizing

Level 1: The Initial Level

- ad hoc, sometimes chaotic
- overcommitment leads to a series of crises
- during a crisis, projects abandon plans
- capability is characteristic of individuals, not the organization
  - when a good manager leaves, the success leaves with them

Level 2: The Repeatable Level

- Planning is based on experience with similar projects
  - past successes can be repeated
- Policies for managing and implementation
  - installed basic management controls
  - track costs and schedules
  - notice and deal with problems as they arise

Level 3: The Defined Level

- Standard processes defined across the organization and used by all projects
  - standard set of roles, activities, quality tracking, etc
  - each project uses a tailored version of this standard process
- Training program is in place to ensure everyone has the skills required for their assigned role

Level 4: The Managed Level

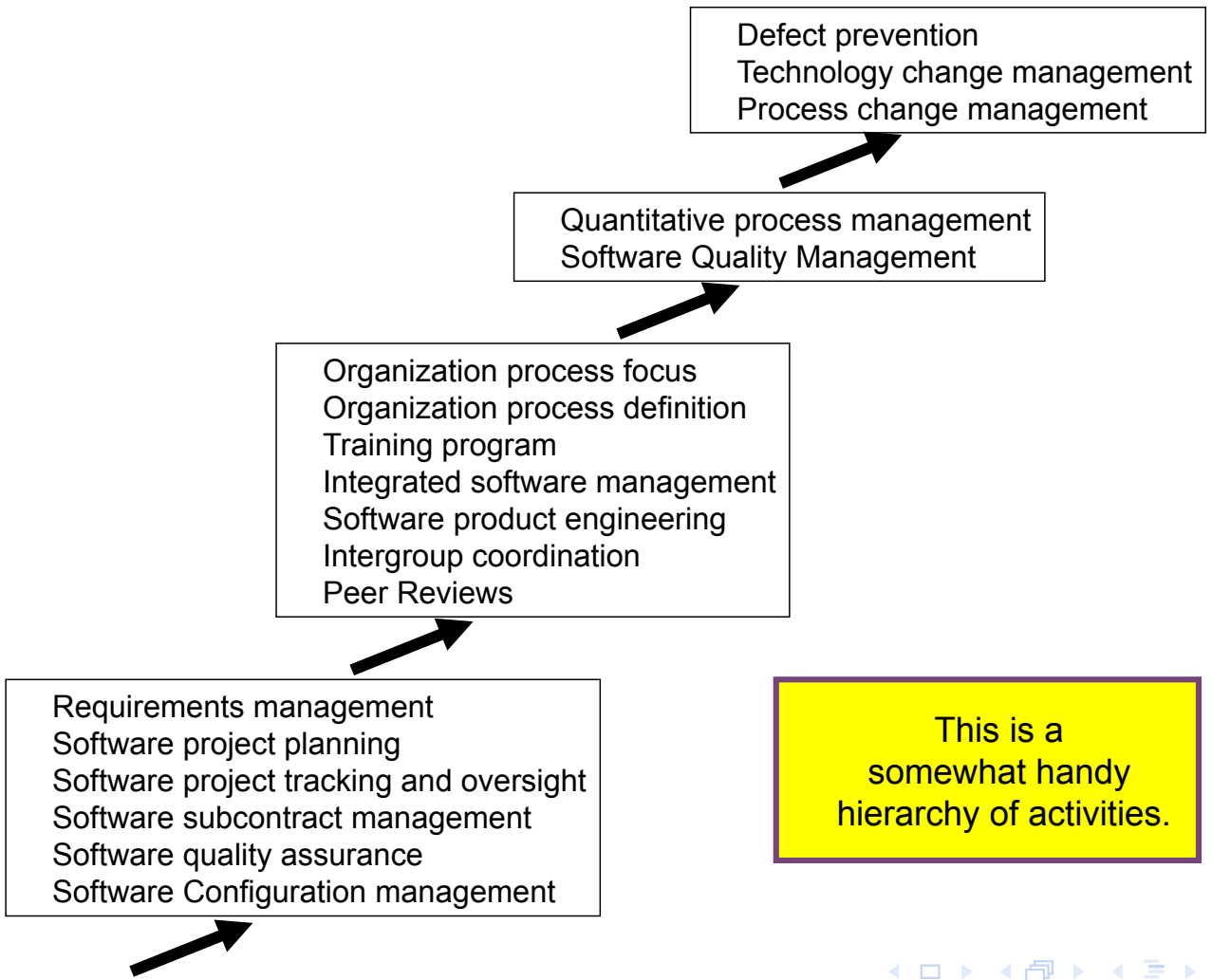
- Quantitative quality goals
  - for both Products and Processes
- Organization-wide process database
  - meaningful variations in process performance can be distinguished from random noise
  - actions are then taken to correct the situation
- Products are of predictably high quality

Level 5: The Optimizing Level

- Organization has the means to identify weaknesses and strengthen the process proactively
- teams analyze defects to determine their cause, and disseminate lessons learned throughout the organization
- major focus on eliminating waste
  - e.g. reduce amount of rework

Process areas

Key process areas by maturity level:



Level Comparison - Risk

- Level 1
  - Just do it
- Level 2
  - problems are recognized and corrected as they occur
- Level 3
  - problems are anticipated and prevented, or impacts minimized
- Levels 4 and 5
  - sources of problems are understood and eliminated

Level Comparison - People

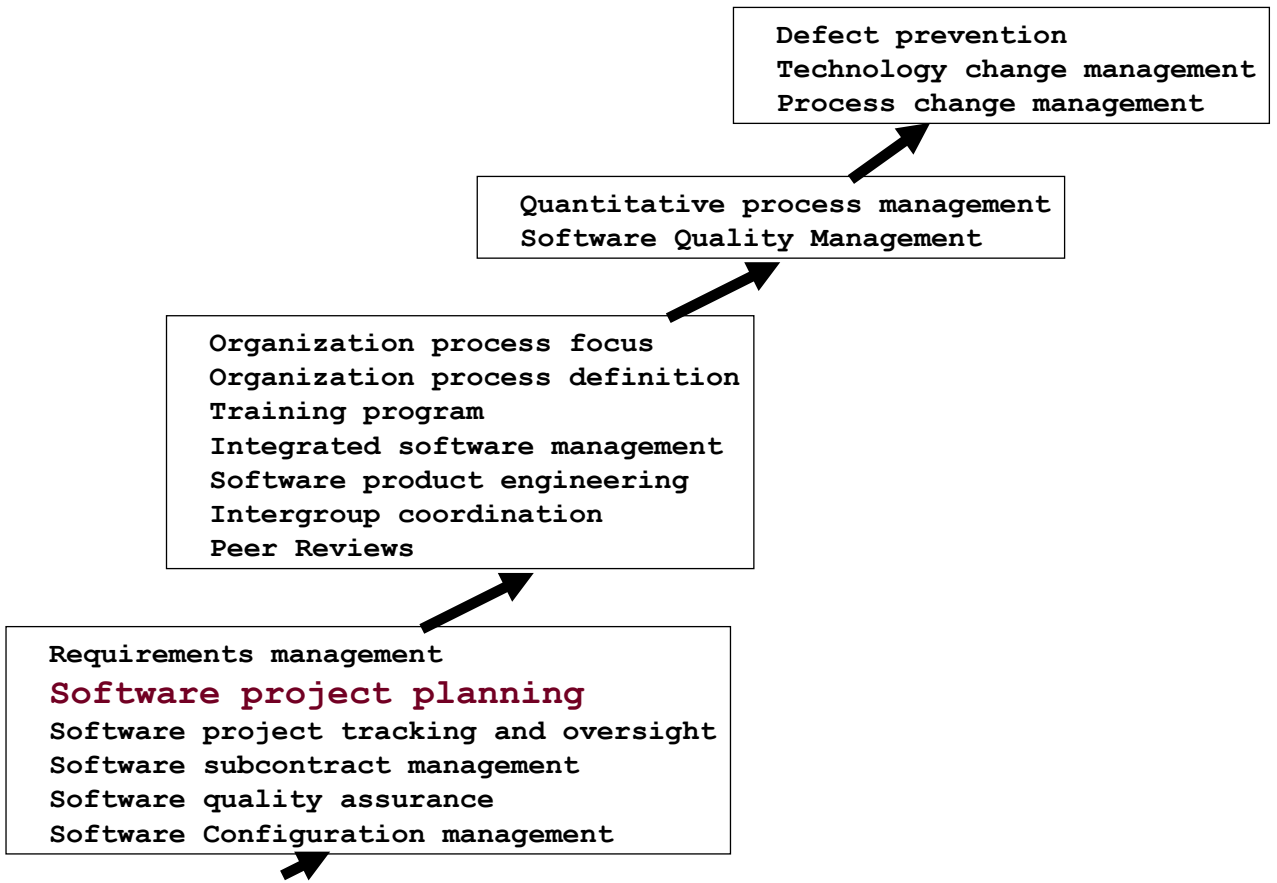
- Level 1
  - success depends on individual heroics
  - fire fighting is the way of life
- Level 2
  - success depends on individuals
  - efforts are supported by management
- Level 3
  - people are trained for their role(s)
  - groups work together
- Levels 4
  - strong sense of teamwork in every project
- Level 5
  - strong sense of teamwork across the organization
  - everyone does process improvement

Level Comparison - Measurement

- Level 1
  - ad hoc (if any) data collection and analysis
- Level 2
  - individual projects use planning data
- Level 3
  - data collected for all processes
  - data shared across projects
- Levels 4
  - data standardized across the organization
- Level 5
  - data used for process improvement

Key process areas

Key process areas by maturity level:



Change Management

- Also called software configuration management (SCM)
- It is an umbrella activity that is applied throughout the software process
- Its goal is to maximize productivity by minimizing mistakes caused by confusion when coordinating software development
- SCM identifies, organizes, and controls modifications to the software being built by a software development team
- SCM activities are formulated to identify change, control change, ensure that change is being properly implemented, and report changes to others who may have an interest

Change management (2)

- SCM is initiated when the project begins and terminates when the software is taken out of operation
- The Output from the software process makes up the software configuration
  - Computer programs (both source code files and executable files)
  - Work products that describe the computer programs (documents targeted at both technical practitioners and users)
  - Data (contained within the programs themselves or in external files)
- The major danger to a software configuration is change
  - First Law of System Engineering: "No matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle"

Origins of Software Change

- Errors detected in the software need to be corrected
- New business or market conditions dictate changes in product requirements or business rules
- New customer needs demand modifications of data produced by information systems, functionality delivered by products, or services delivered by a computer-based system
- Reorganization or business growth/downsizing causes changes in project priorities or software engineering team structure
- Budgetary or scheduling constraints cause a redefinition of the system or product

Elements of a Configuration Management System

- Configuration elements
  - A set of tools coupled with a file management (e.g., database) system that enables access to and management of each software configuration item
- Process elements
  - A collection of procedures and tasks that define an effective approach to change management for all participants
- Construction elements
  - A set of tools that automate the construction of software by ensuring that the proper set of valid components (i.e., the correct version) is assembled
- Human elements
  - A set of tools and process features used by a software team to implement effective SCM

Software metrics

Source

- Pressman, R. *Software Engineering: A Practitioner's Approach.* Chapter 15. McGraw-Hill, 2005.

Overview

- why look at metrics?

Examples of Metrics from Everyday Life

- Working and living
  - Cost of utilities for the month
  - Cost of groceries for the month
  - Amount of monthly rent per month
  - Time spent at work each Saturday for the past month
  - Time spent mowing the lawn for the past two times
- University study
  - Grades received in class last semester
  - Number of classes taken each semester
  - Amount of time spent in class this week
  - Amount of time spent on studying and homework this week
  - Number of hours of sleep last night
- Travel
  - Time to drive from home to the airport
  - Amount of miles traveled today
  - Cost of meals and lodging for yesterday

Why have Software Product Metrics?

- Help software engineers to better understand the attributes of models and assess the quality of the software
- Help software engineers to gain insight into the design and construction of the software
- Focus on specific attributes of software engineering work products resulting from analysis, design, coding, and testing
- Provide a systematic way to assess quality based on a set of clearly defined rules
- Provide an "on-the-spot" rather than "after-the-fact" insight into the software development

Measures, Metrics, and Indicators

- These three terms are often used interchangeably, but they can have subtle differences
- Measure
  - Provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of a product or process
- Measurement
  - The act of determining a measure
- Metric
  - (IEEE) A quantitative measure of the degree to which a system, component, or process possesses a given attribute
- Indicator
  - A metric or combination of metrics that provides insight into the software process, a software project, or the product itself

Activities of a Measurement Process

- Formulation
  - The derivation (i.e., identification) of software measures and metrics appropriate for the representation of the software that is being considered
- Collection
  - The mechanism used to accumulate data required to derive the formulated metrics
- Analysis
  - The computation of metrics and the application of mathematical tools
- Interpretation
  - The evaluation of metrics in an effort to gain insight into the quality of the representation
- Feedback
  - Recommendations derived from the interpretation of product metrics and passed on to the software development team

Purpose of product metrics

- Aid in the evaluation of analysis and design models
- Provide an indication of the complexity of procedural designs and source code
- Facilitate the design of more effective testing techniques
- Assess the stability of a fielded software product

Characterizing and Validating Metrics

- A metric should have desirable mathematical properties
  - It should have a meaningful range (e.g., zero to ten)
  - It should not be set on a rational scale if it is composed of components measured on an ordinal scale
- If a metric represents a software characteristic that increases when positive traits occur or decreases when undesirable traits are encountered, the value of the metric should increase or decrease in the same manner
- Each metric should be validated empirically in a wide variety of contexts before being published or used to make decisions
  - It should measure the factor of interest independently of other factors
  - It should scale up to large systems
  - It should work in a variety of programming languages and system domains

Goal-oriented Software Measurement

- Goal/Question/Metric (GQM) paradigm
- GQM technique identifies meaningful metrics for any part of the software process

GQM

- GQM emphasizes the need to
  - Establish an explicit measurement goal that is specific to the process activity or product characteristic that is to be assessed
  - Define a set of questions that must be answered in order to achieve the goal
  - Identify well-formulated metrics that help to answer these questions
- GQM utilizes a goal definition template to define each measurement goal

GQM example

- Example use of goal definition template
  - Analyze the SafeHome software architecture for the purpose of evaluating architecture components. Do this with respect to the ability to make SafeHome more extensible from the viewpoint of the software engineers, who are performing the work in the context of product enhancement over the next three years.
- Example questions for this goal definition
  - Are architectural components characterized in a manner that compartmentalizes function and related data?
  - Is the complexity of each component within bounds that will facilitate modification and extension?

Attributes of Effective Software Metrics

- Simple and computable
  - It should be relatively easy to learn how to derive the metric, and its computation should not demand inordinate effort or time
- Empirically and intuitively persuasive
  - The metric should satisfy the engineer's intuitive notions about the product attribute under consideration
- Consistent and objective
  - The metric should always yield results that are unambiguous
- Consistent in the use of units and dimensions
  - The mathematical computation of the metric should use measures that do not lead to bizarre combinations of units
- Programming language independent
  - Metrics should be based on the analysis model, the design model, or the structure of the program itself
- An effective mechanism for high-quality feedback
  - The metric should lead to a higher-quality end product

Taxonomy: Metrics for the Analysis Model

- Functionality delivered
  - Provides an indirect measure of the functionality that is packaged within the software
- System size
  - Measures the overall size of the system defined in terms of information available as part of the analysis model
- Specification quality
  - Provides an indication of the specificity and completeness of a requirements specification

Taxonomy: Metrics for the Design Model

- Architectural metrics
  - Provide an indication of the quality of the architectural design
- Component-level metrics
  - Measure the complexity of software components and other characteristics that have a bearing on quality
- Interface design metrics
  - Focus primarily on usability
- Specialized object-oriented design metrics
  - Measure characteristics of classes and their communication and collaboration characteristics

Taxonomy: Metrics for Source Code

- Complexity metrics
  - Measure the logical complexity of source code (can also be applied to component-level design)
- Length metrics
  - Provide an indication of the size of the software

Taxonomy: Metrics for Testing

- Statement and branch coverage metrics
  - Lead to the design of test cases that provide program coverage
- Defect-related metrics
  - Focus on defects (i.e., bugs) found, rather than on the tests themselves
- Testing effectiveness metrics
  - Provide a real-time indication of the effectiveness of tests that have been conducted
- In-process metrics
  - Process related metrics that can be determined as testing is conducted

Architectural Design Metrics

- These metrics place emphasis on the architectural structure and effectiveness of modules or components within the architecture
- They are “black box” in that they do not require any knowledge of the inner workings of a particular software component

Hierarchical Architecture Metrics

- Fan out: the number of modules immediately subordinate to the module i, that is, the number of modules directly invoked by module i
- Structural complexity
  - e.g. The "fan out" level of some module
- Data complexity
  - e.g. The number of input and output variables that are passed to and from module i

Hierarchical Architecture Metrics, cont'd

- System complexity
  - Structural complexity + data complexity

- As each of these complexity values increases, the overall architectural complexity of the system also increases
- This leads to greater likelihood that the integration and testing effort will also increase

Hierarchical Architecture Metrics, cont'd

- Shape complexity
  - size $= n + a$, where n is the number of nodes and a is the number of arcs
  - Allows different program software architectures to be compared in a straightforward manner
- Connectivity density (i.e., the arc-to-node ratio)
  - $r = a/n$
  - May provide a simple indication of the coupling in the software architecture

Metrics for Object-Oriented Design

- Size
  - Population: a static count of all classes and methods
  - Volume: a dynamic count of all instantiated objects at a given time
  - Length: the depth of an inheritance tree
- Coupling
  - The number of collaborations between classes or the number of methods called between objects
- Cohesion
  - The cohesion of a class is the degree to which its set of properties is part of the problem or design domain
- Primitiveness
  - The degree to which a method in a class is atomic (i.e., the method cannot be constructed out of a sequence of other methods provided by the class)

Specific Class-oriented Metrics

- Weighted methods per class
  - The normalized complexity of the methods in a class, indicates the amount of effort to implement and test a class
- Depth of the inheritance tree
  - The maximum length from the derived class (the node) to the base class (the root), indicates the potential difficulties when attempting to predict the behavior of a class because of the number of inherited methods
- Number of children (i.e., subclasses)
  - As the number of children of a class grows, reuse increases, the abstraction represented by the parent class can be diluted by inappropriate children and the amount of testing required will increase.

Specific Class-oriented Metrics, cont'd

- Coupling between object classes
  - Measures the number of collaborations a class has with any other classes
  - Higher coupling decreases the reusability of a class
  - Higher coupling complicates modifications and testing
- Response for a class
  - This is the set of methods that can potentially be executed in a class in response to a public method call from outside the class, indicates the effort required for testing and the overall design complexity of the class
- Lack of cohesion in methods
  - This measures the number of methods that access one or more of the same instance variables (i.e., attributes) of a class
  - If no methods access the same attribute, then the measure is zero

Metrics for Maintenance

- Software maturity index (SMI)
  - Provides an indication of the stability of a software product based on changes that occur for each release
  - SMI $= \dfrac{MT - (F_a + F_c + F_d)}{MT}$
  - where
    MT = #modules in the current release
    F_a = #modules in the current release that have been added
    F_c = #modules in the current release that have been changed
    F_d = #modules from the preceding release that were deleted in the current release
- As the SMI (i.e., the fraction) approaches 1.0, the software product begins to stabilize
- The average time to produce a release of a software product can be correlated with the SMI