Lecture 9

- Software Quality Assurance

“What is quality software and how to achieve it?”

Topics

- Software Quality
- Software Quality Dimensions
- Software Quality Assurance
- Software Quality Standards

1. Software Quality

Software Quality

- In 2005, *ComputerWorld* [Hil05] lamented that
  - “bad software plagues nearly every organization that uses computers, causing lost work hours during computer downtime, lost or corrupted data, missed sales opportunities, high IT support and maintenance costs, and low customer satisfaction.
- A year later, *InfoWorld* [Fos06] wrote about the
  - “the sorry state of software quality” reporting that the quality problem had not gotten any better.
- Today, software quality remains an issue, but who is to blame?
  - Customers blame developers, arguing that sloppy practices lead to low-quality software.
  - Developers blame customers (and other stakeholders), arguing that irrational delivery dates and a continuing stream of changes force them to deliver software before it has been fully validated.

Quality

- The *American Heritage Dictionary* defines *quality* as
  - “a characteristic or attribute of something.”
- For software, two kinds of quality may be encountered:
  - Quality of design encompasses requirements, specifications, and the design of the system.
  - Quality of conformance is an issue focused primarily on implementation.
  - User satisfaction = compliant product + good quality + delivery within budget and schedule

Software Quality

- Software quality can be defined as:
  - *An effective software process applied in a manner that creates a useful product that provides measurable value for those who produce it and those who use it.*
- This definition has been adapted from [Bes04] and replaces a more manufacturing-oriented view presented in earlier editions of this book.

Effective Software Process

- An *effective software process* establishes the infrastructure that supports any effort at building a high quality software product.
- The management aspects of process create the checks and balances that help avoid project chaos—a key contributor to poor quality.
- Software engineering practices allow the developer to analyze the problem and design a solid solution—both critical to building high quality software.
- Finally, umbrella activities such as change management and technical reviews have as much to do with quality as any other part of software engineering practice.

Useful Product

- A *useful product* delivers the content, functions, and features that the end-user desires
- But as important, it delivers these assets in a reliable, error free way.
- A useful product always satisfies those requirements that have been explicitly stated by stakeholders.
- In addition, it satisfies a set of implicit requirements (e.g., ease of use) that are expected of all high quality software.

Adding Value

- By *adding value for both the producer and user* of a software product, high quality software provides benefits for the software organization and the end-user community.
- The software organization gains added value because high quality software requires less maintenance effort, fewer bug fixes, and reduced customer support.
- The user community gains added value because the application provides a useful capability in a way that expedites some business process.
- The end result is:
  - (1) greater software product revenue,
  - (2) better profitability when an application supports a business process, and/or
  - (3) improved availability of information that is crucial for the business.

2. Software Quality Dimensions

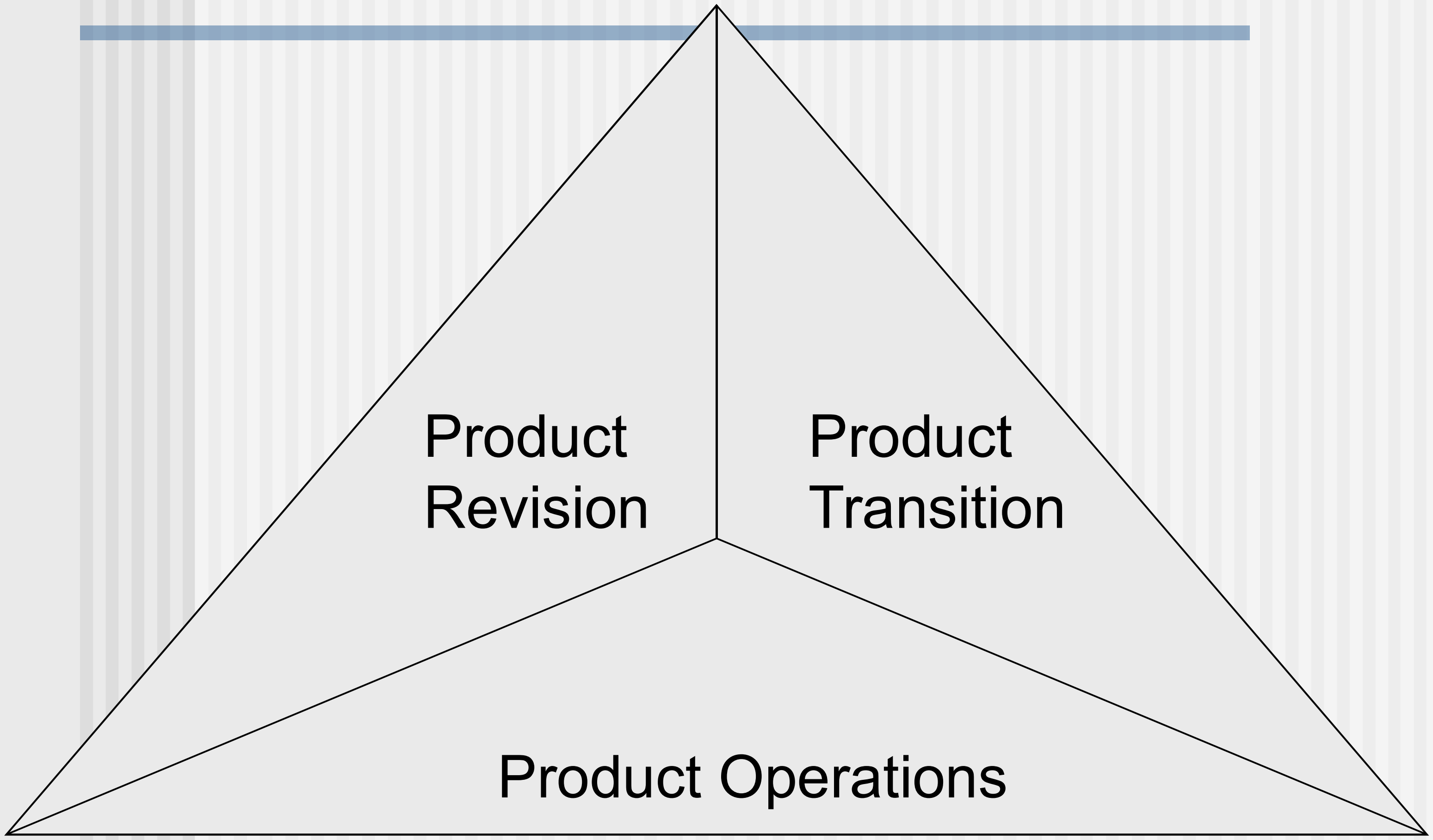
Quality Dimensions

- David Garvin [Gar87]:
  - **Performance Quality.** Does the software deliver all content, functions, and features that are specified as part of the requirements model in a way that provides value to the end-user?
  - **Feature quality.** Does the software provide features that surprise and delight first-time end-users?
  - **Reliability.** Does the software deliver all features and capability without failure? Is it available when it is needed? Does it deliver functionality that is error free?
  - **Conformance.** Does the software conform to local and external software standards that are relevant to the application? Does it conform to de facto design and coding conventions? For example, does the user interface conform to accepted design rules for menu selection or data input?

Quality Dimensions

- **Durability.** Can the software be maintained (changed) or corrected (debugged) without the inadvertent generation of unintended side effects? Will changes cause the error rate or reliability to degrade with time?
- **Serviceability.** Can the software be maintained (changed) or corrected (debugged) in an acceptably short time period. Can support staff acquire all information they need to make changes or correct defects?
- **Aesthetics.** Most of us would agree that an aesthetic entity has a certain elegance, a unique flow, and an obvious "presence" that are hard to quantify but evident nonetheless.
- **Perception.** In some situations, you have a set of prejudices that will influence your perception of quality.

McCall's Software Quality Factors



McCall’s Software Quality Factors

- Product Operations
  - Operational characteristics
- Product Revision
  - Ability to undergo changes
- Product Transition
  - Adaptability to new environments

Product Operations Factors

- Correctness (Does it do what I want?)
- Reliability (Does it do it accurately all of the time?)
- Efficiency (Will it run on my hardware as well as it can?)
- Integrity (Is it secure?)
- Usability (Is it designed for the user?)

Product Revision Factors

- Maintainability (Can I fix it?)
- Flexibility (Can I change it?)
- Testability (Can I test it?)

Product Transition Factors

- Portability (Will I be able to use it on another machine?)
- Reusability (Will I be able to reuse some of the software?)
- Interoperability (Will I be able to interface with another system?)

Measuring Quality

- General quality dimensions and factors are not adequate for assessing the quality of an application in concrete terms
- Project teams need to develop a set of targeted questions to assess the degree to which each application quality factor has been satisfied
- Subjective measures of software quality may be viewed as little more than personal opinion
- Software metrics represent indirect measures of some manifestation of quality and attempt to quantify the assessment of software quality

The Software Quality Dilemma

- If you produce a software system that has terrible quality, you lose because no one will want to buy it.
- If on the other hand you spend infinite time, extremely large effort, and huge sums of money to build the absolutely perfect piece of software, then it's going to take so long to complete and it will be so expensive to produce that you'll be out of business anyway.
- Either you missed the market window, or you simply exhausted all your resources.
- So people in industry try to get to that magical middle ground where the product is good enough not to be rejected right away, such as during evaluation, but also not the object of so much perfectionism and so much work that it would take too long or cost too much to complete. [Ven03]

“Good Enough” Software

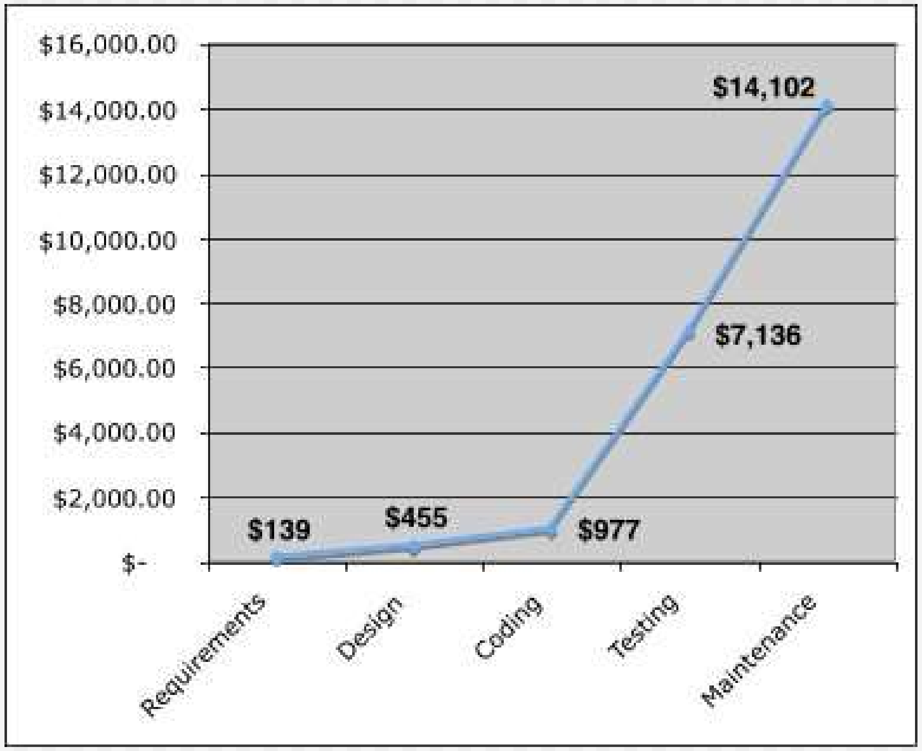
- Good enough software delivers high quality functions and features that end-users desire, but at the same time it delivers other more obscure or specialized functions and features that contain known bugs.
- Arguments *against* “good enough.”
  - It is true that “good enough” may work in some application domains and for a few major software companies. After all, if a company has a large marketing budget and can convince enough people to buy version 1.0, it has succeeded in locking them in.
  - If you work for a small company be wary of this philosophy. If you deliver a “good enough” (buggy) product, you risk permanent damage to your company’s reputation.
  - You may never get a chance to deliver version 2.0 because bad buzz may cause your sales to plummet and your company to fold.
  - If you work in certain application domains (e.g., real time embedded software, application software that is integrated with hardware can be negligent and open your company to expensive litigation.

The Cost of Quality

- **Prevention costs** include:
  - Quality planning
  - Formal technical reviews
  - Test equipment
  - Training
- **Internal failure costs** include:
  - Rework
  - Repair
  - Failure mode analysis
- **External failure costs** are:
  - Complaint resolution
  - Product return and replacement
  - Help line support
  - Warranty work

Cost

- The relative costs to find and repair an error or defect increase dramatically as we go from prevention to detection to internal failure to external failure costs.



3. Software Quality Assurance

Achieving Software Quality 1

- Software quality is the result of good project management and solid engineering practice
- To build high quality software you must understand the problem to be solved and be capable of creating a quality design the conforms to the problem requirements
- Eliminating architectural flaws during design can improve quality

Achieving Software Quality 2

- Project management – project plan includes explicit techniques for quality and change management
- Quality control - series of inspections, reviews, and tests used to ensure conformance of a work product to its specifications
- Quality assurance - consists of the auditing and reporting procedures used to provide management with data needed to make proactive decisions

Elements of SQA

- **Standards**
- **Reviews and Audits**
- **Testing**
- **Error/defect collection and analysis**
- **Change management**
- **Education**
- **Vendor management**
- **Security management**
- **Safety**
- **Risk management**

Role of the SQA Group

- **Prepares an SQA plan for a project.** The plan identifies:
  - Evaluations to be performed.
  - Audits and reviews to be performed.
  - Standards that are applicable to the project.
  - Procedures for error reporting and tracking.
  - Documents to be produced by the SQA group.
  - Amount of feedback provided to the software project team.
- **Participates in the development of the project's software process description.**
  - The SQA group reviews the process description for compliance with organizational policy, internal software standards, externally imposed standards (e.g., ISO-9001), and other parts of the software project plan.

Role of the SQA Group

- **Reviews software engineering activities to verify compliance with the defined software process.**
  - Identifies, documents, and tracks deviations from the process and verifies that corrections have been made.
- **Audits designated software work products to verify compliance with those defined as part of the software process.**
  - Reviews selected work products; identifies, documents, and tracks deviations; verifies that corrections have been made
  - Periodically reports the results of its work to the project manager.

Role of the SQA Group

- **Ensures that deviations in software work and work products are documented and handled according to a documented procedure.**
- **Records any noncompliance and reports to senior management.**
  - Noncompliance items are tracked until they are resolved.

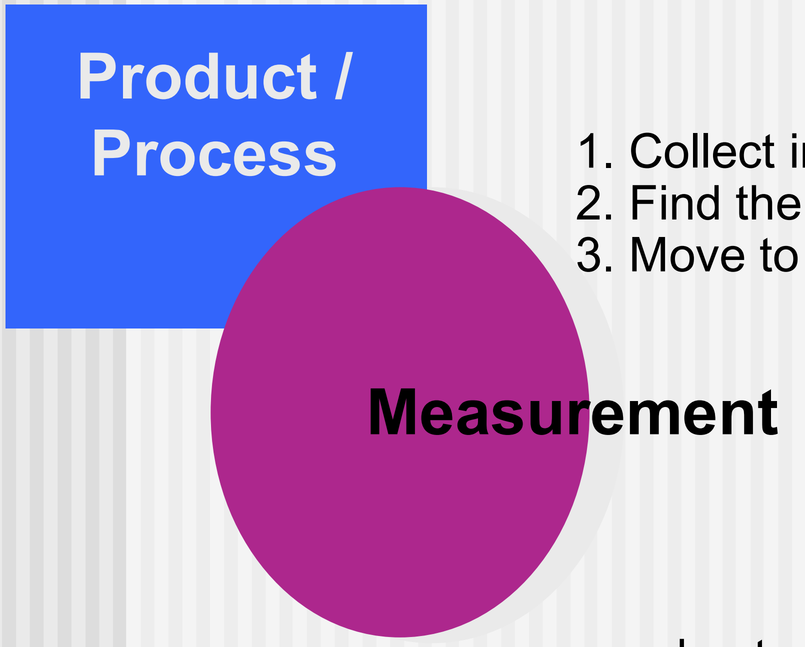
SQA Goals

- **Requirements quality.** The correctness, completeness, and consistency of the requirements model will have a strong influence on the quality of all work products that follow.
- **Design quality.** Every element of the design model should be assessed by the software team to ensure that it exhibits high quality and that the design itself conforms to requirements.
- **Code quality.** Source code and related work products (e.g., other descriptive information) must conform to local coding standards and exhibit characteristics that will facilitate maintainability.
- **Quality control effectiveness.** A software team should apply limited resources in a way that has the highest likelihood of achieving a high quality result.

Steps of Conducting a Statistical SQA

- **Information about software errors and defects** is collected and categorized.
- An attempt is made to **trace each error and defect to its underlying cause** (e.g., non-conformance to specifications, design error, violation of standards, poor communication with the customer).
- Using the **Pareto principle** (80 percent of the defects can be traced to 20 percent of all possible causes), isolate the 20 percent (the vital few).
- Once the vital few causes have been identified, move to **correct the problems that have caused the errors and defects**.

Statistical SQA



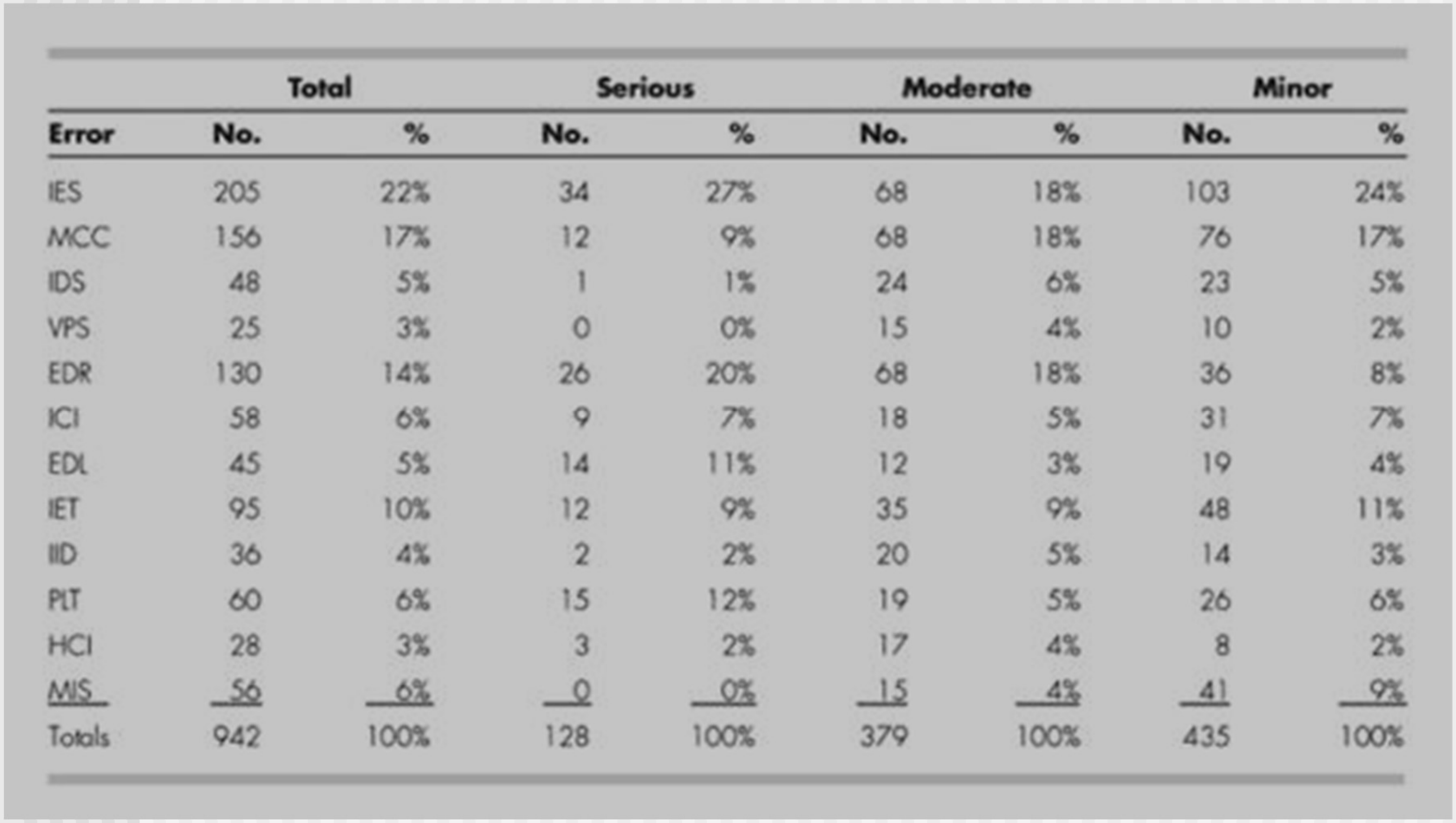
1. Collect information on all defects.
2. Find the causes of the defects.
3. Move to provide fixes for the process.

... an understanding of how to improve quality.

Statistical SQA : Generic Causes of Problems

- Some of the errors are uncovered as software is being developed; others (defects) are encountered after the software has been released to its end users.
- Although hundreds of different problems are uncovered, all can be tracked to one (or more) of the following causes:
  - **Incomplete or Erroneous Specifications** (IES)
  - **Misinterpretation of Customer Communication** (MCC)
  - Intentional Deviation from Specifications (IDS)
  - Violation of Programming Standards (VPS)
  - **Error in Data Representation** (EDR)
  - Inconsistent Component Interface (ICI)
  - Error in Design Logic (EDL)
  - Incomplete or erroneous Testing (IEL)
  - Inaccurate or incomplete Documentation (IID)
  - Error in Programming Language Translation of Design (PLT)
  - Ambiguous or Inconsistent Human / Computer Interface (HCI)
  - Miscellaneous (MIS)

Statistical SQA : Generic Causes of Problems



	Total		Serious		Moderate		Minor	
Error	No.	%	No.	%	No.	%	No.	%
IES	205	22%	34	27%	68	18%	103	24%
MCC	156	17%	12	9%	68	18%	76	17%
IDS	48	5%	1	1%	24	6%	23	5%
VPS	25	3%	0	0%	15	4%	10	2%
EDR	130	14%	26	20%	68	18%	36	8%
ICI	58	6%	9	7%	18	5%	31	7%
EDL	45	5%	14	11%	12	3%	19	4%
IET	95	10%	12	9%	35	9%	48	11%
IID	36	4%	2	2%	20	5%	14	3%
PLT	60	6%	15	12%	19	5%	26	6%
HCI	28	3%	3	2%	17	4%	8	2%
MIS	56	6%	0	0%	15	4%	41	9%
Totals	942	100%	128	100%	379	100%	435	100%

Software Reliability

- Software reliability is defined in statistical terms as the **probability of failure-free operation of a computer program in a specified environment for a specified time**.
- To illustrate,

Program X is estimated to have a reliability of 0.999 over eight elapsed processing hours.

- If the **program x were to be executed 1000 times** and required **a total of eight hours of elapsed processing time** (execution time), **it is likely to operate correctly 999 times** (without failure).

Software Reliability

- A simple measure of reliability is **mean-time-between-failure** (MTBF), where

MTBF = MTTF + MTTR

- The acronyms MTTF and MTTR are **mean-time-to-failure** and **mean-time-to-repair**, respectively.
- Software availability is the **probability that a program is operating according to requirements at a given point in time** and is defined as:

Availability = [MTTF / (MTTF + MTTR)] x 100%

Software Safety

- Software safety is a software quality assurance activity that focuses on the **identification and assessment of potential hazards that may affect software negatively and cause an entire system to fail**.
- If hazards can be identified early in the software process, software design features can be specified that will either eliminate or control potential hazards.

4. Software Quality Standards

Software Quality Standards

- Several national and international standards institutes, professional and industry-oriented organizations have been involved in the development of SQA standards.
- The following institutes and organizations are the main developers of SQA and software engineering standards
  - IEEE (Institute of Electrical and Electronics Engineers) Computer Society
  - ISO (International Organization for Standardization)
  - DOD (US Department of Defense)
  - ANSI (American National Standards Institute)
  - IEC (International Electro Technical Commission)
  - EIA (Electronic Industries Association)

ISO 9001 Standards

- The international standard that specifies requirements for a quality management system (QMS).
- Organizations use the standard to demonstrate the ability to consistently provide products and services that meet customer and regulatory requirements.
- ISO 9001:2015 is **the quality assurance standard that applies to software engineering**.
- The standard contains requirements that must be present for an effective quality assurance system.

ISO 9001 Requirements

- The requirements delineated by ISO 9001:2015 address topics such as:
  - Leadership's Responsibility
  - Quality Policy
  - Planning for the Quality Management System
  - Supporting the Quality Management System
  - Customer Communication
  - Product Identification and Traceability
  - Customer Satisfaction
  - Effective Management Review
  - Internal Audit
  - Continual Improvement
  - Nonconformity and Corrective Action

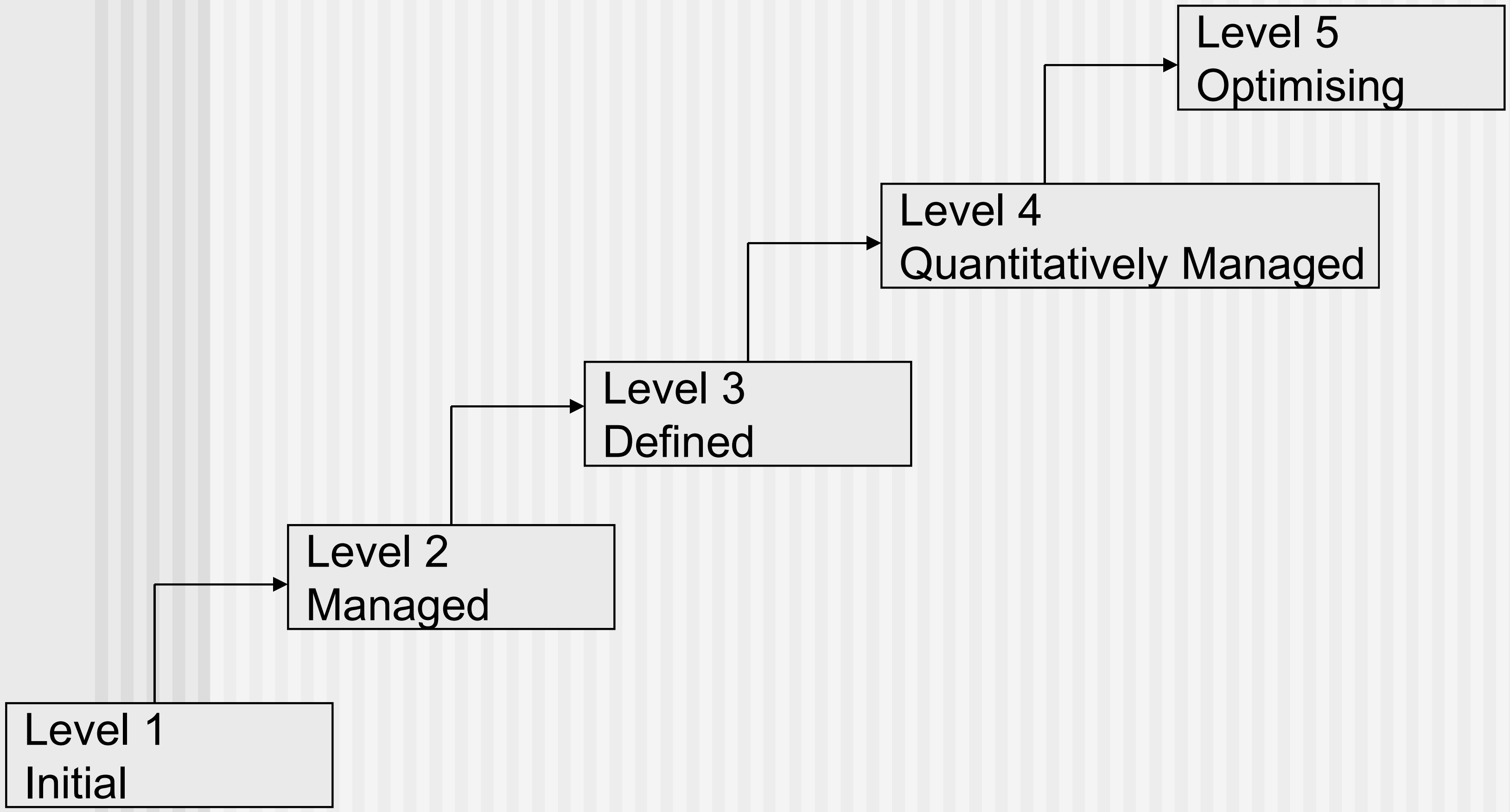
Capability Maturity Model (CMM)

- Developed by Software Engineering Institute (SEI) of Carnegie-Mellon U
- SEI was established to improve the capabilities of US software industry
- Focus on improving the process of software engineering
- Similar to ISO, organisations are assessed and certified to be on Level 1 to Level 5

Capability Maturity Model Integration (CMMI)

- Successor of CMM
- Includes changes to support Agile Software Development
- Improvements to high maturity practices
- Alignment of the representation (staged and continuous)

Process Maturity Model



Six-Sigma for Software Engineering

- The term “six sigma” is derived from six standard deviations—3.4 instances (defects) per million occurrences—implying an extremely high quality standard.
- The Six Sigma methodology defines three core steps:
  - *Define* customer requirements and deliverables and project goals via well-defined methods of customer communication
  - *Measure* the existing process and its output to determine current quality performance (collect defect metrics)
  - *Analyze* defect metrics and determine the vital few causes.
  - *Improve* the process by eliminating the root causes of defects.
  - *Control* the process to ensure that future work does not reintroduce the causes of defects.