CSE1005: Software Engineering

Module 6:
Software Quality

Dr. Mehfooza M
SCOPE, VIT-AP University

Recap

- Software and Software Engineering
- The Software Process
- Agile Process
- Principles that guide practice
- Requirement Engineering – Understanding Requirements
- Requirements Modeling - Scenarios, Information, and Analysis Classes
- Requirements Modeling: Flow, Behavior, Patterns, and WebApps
- Software Design
- Software Testing – issues and strategies
- Software Product and Process Metrics
- Software Project Management

Introduction

VIT-AP UNIVERSITY

- By the 1990s, major corporations recognized that billions of dollars each year were being wasted on software that didn't deliver the features and functionality that were promised
- In 2005, ComputerWorld 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
- 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.

  Who's right? Both—and that's the problem!

Introduction (cont…)

VIT-AP UNIVERSITY

- What is quality?

  “quality is a complex and multifaceted concept” that can be described from five different points of view -- David Garvin [Harvard Business School]

  - The **transcendental view** argues that quality is something that you immediately recognize, but cannot explicitly define
  - The **user view** sees quality in terms of an end user’s specific goals. If a product meets those goals, it exhibits quality
  - The **manufacturer’s view** defines quality in terms of the original specification of the product. If the product conforms to the spec, it exhibits quality
  - The **product view** suggests that quality can be tied to inherent characteristics (e.g., functions and features) of a product
  - The value-based view measures quality based on how much a customer is willing to pay for a product

  In reality, quality encompasses all of these views and more.

Introduction (cont…)

- What is software quality?
  - 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.

  - Garvin's Quality Dimensions – multidimensional viewpoint
    - **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?
    - **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?

Introduction (cont…)

- What is software quality?
  - 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.

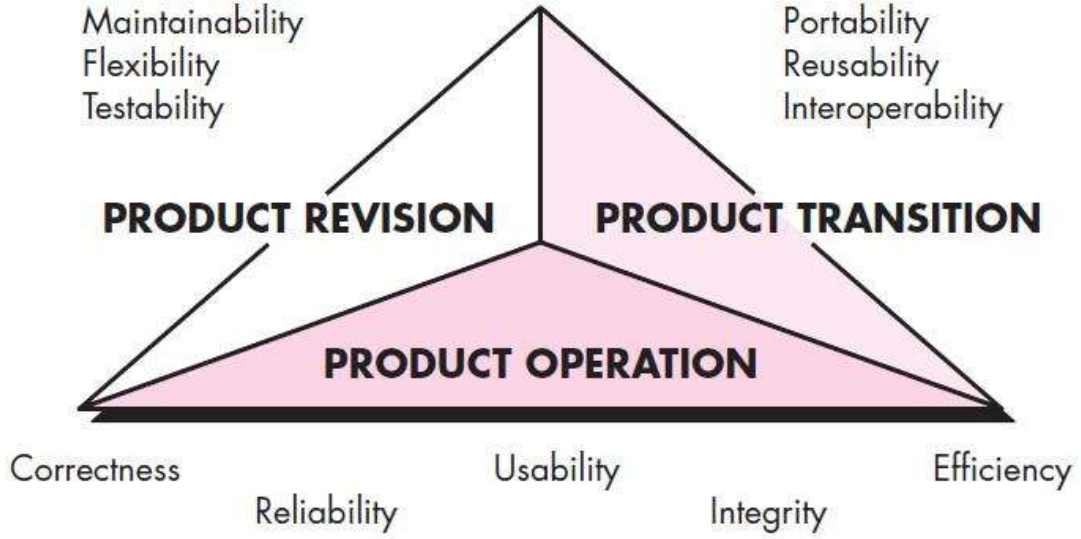
  - Garvin's Quality Dimensions – multidimensional viewpoint (cont…)
    - **Serviceability.** Can the software be maintained (changed) or corrected (debugged) in an acceptably short time period? Can support staff acquire all information they need to make changes or correct defects? Douglas Adams [Ada93] makes a wry comment that seems appropriate here: "The difference between something that can go wrong and something that can't possibly go wrong is that when something that can't possibly go wrong goes wrong it usually turns out to be impossible to get at or repair".
    - **Aesthetics.** There's no question that each of us has a different and very subjective vision of what is aesthetic. And yet, 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 are evident nonetheless. Aesthetic software has these characteristics.
    - **Perception.** In some situations, you have a set of prejudices that will influence your perception of quality. For example, if you are introduced to a software product that was built by a vendor who has produced poor quality in the past, your guard will be raised and your perception of the current software product quality might be influenced negatively. Similarly, if a vendor has an excellent reputation, you may perceive quality, even when it does not really exist

Outline

- Software Quality Factors
- Software Quality Assurance
- The Capability Maturity Model

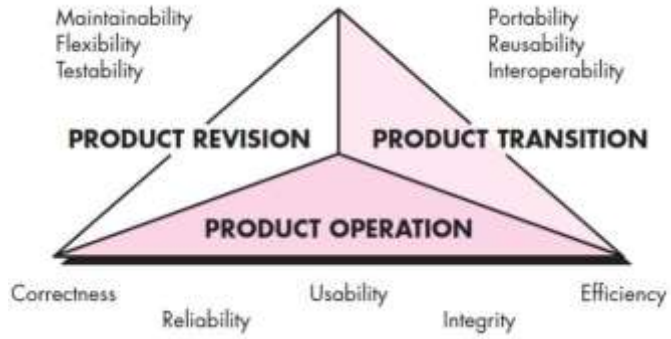
Software Quality Factors

- McCall's Quality Factors
  - McCall, Richards, and Walter propose a useful categorization of factors that affect software quality focusing on three important aspects of a software product: its operational characteristics, its ability to undergo change, and its adaptability to new environments



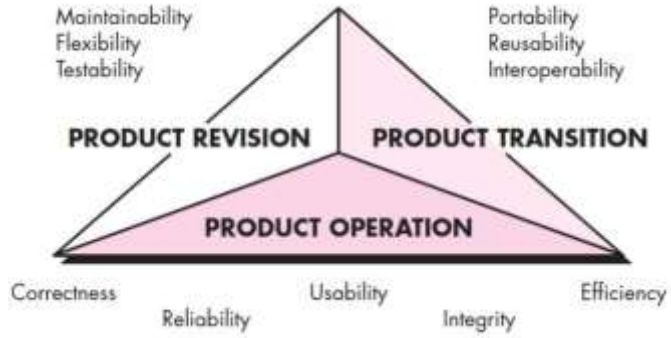
Software Quality Factors

- McCall's Quality Factors - Description
  - **Correctness.**
    - The extent to which a program satisfies its specification and fulfills the customer's mission objectives



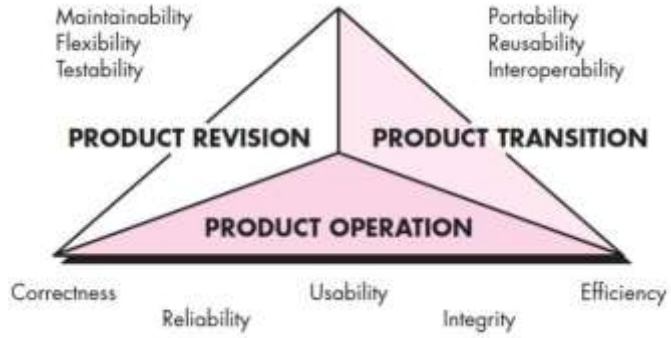
Software Quality Factors

- McCall's Quality Factors - Description
  - **Reliability.**
    - The extent to which a program can be expected to perform its intended function with required precision.



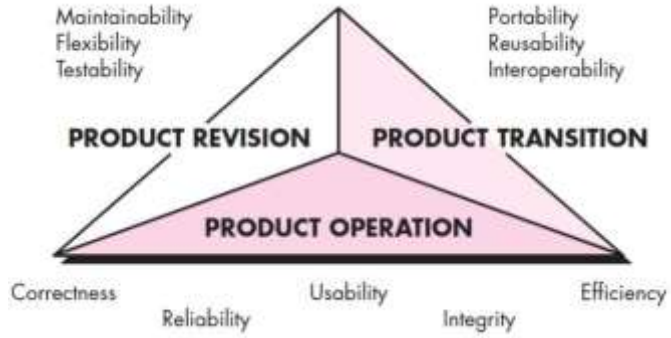
Software Quality Factors

- McCall's Quality Factors - Description
  - **Efficiency.**
    - The amount of computing resources and code required by a program to perform its function.



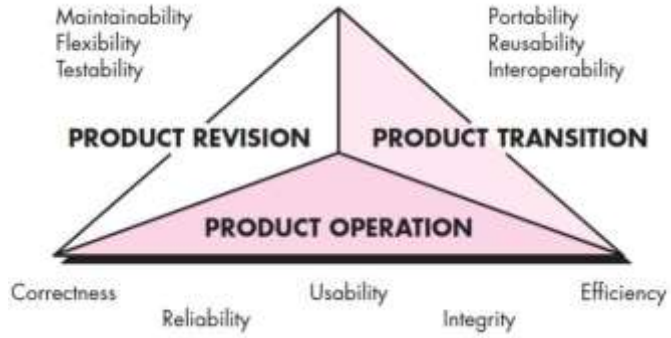
Software Quality Factors

- McCall's Quality Factors - Description
  - **Integrity.**
    - Extent to which access to software or data by unauthorized persons can be controlled.



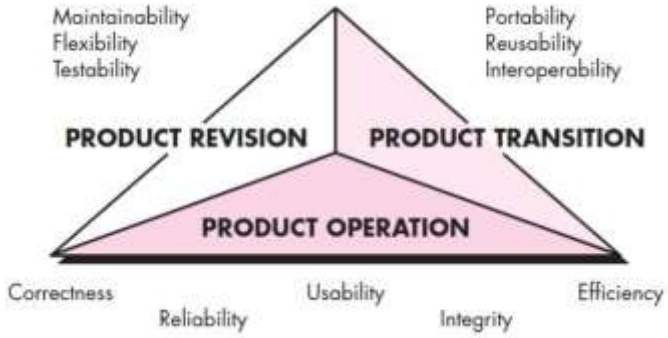
Software Quality Factors

- McCall's Quality Factors - Description
  - **Usability.**
    - Effort required to learn, operate, prepare input for, and interpret output of a program.



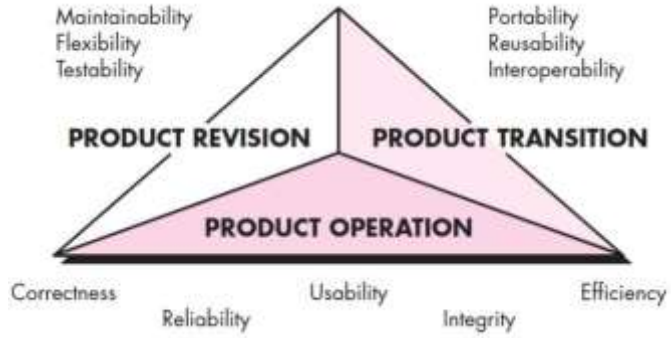
Software Quality Factors

- McCall's Quality Factors - Description
  - **Maintainability.**
    - Effort required to locate and fix an error in a program.



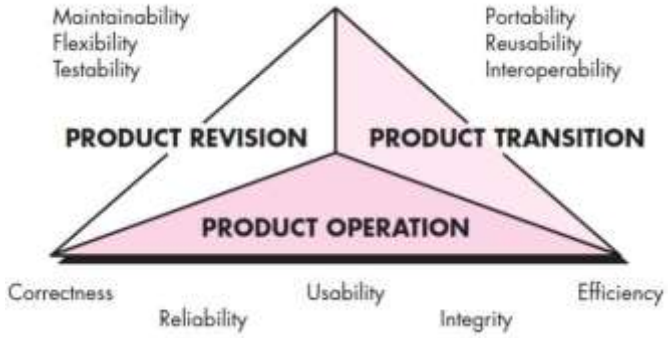
Software Quality Factors

- McCall's Quality Factors - Description
  - **Flexibility.**
    - Effort required to modify an operational program.



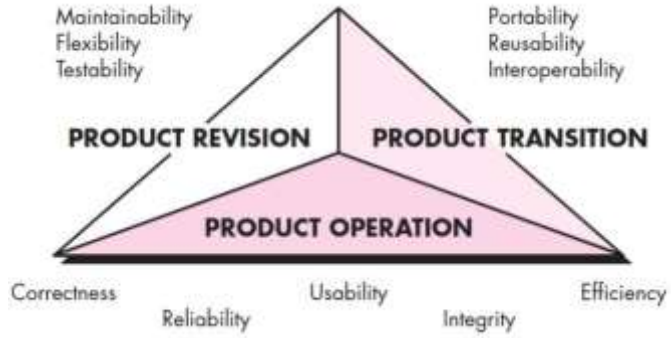
Software Quality Factors

- McCall's Quality Factors - Description
  - **Testability.**
    - Effort required to test a program to ensure that it performs its intended function.



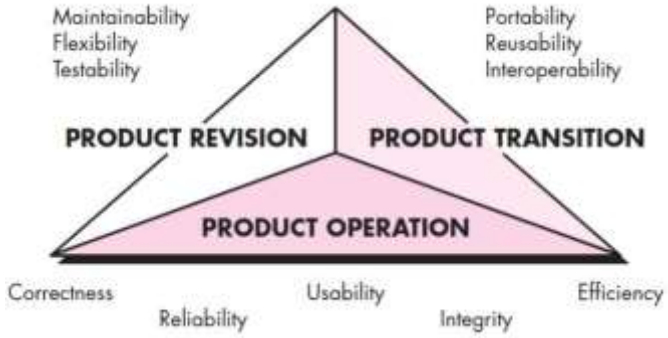
Software Quality Factors

- McCall's Quality Factors - Description
  - **Portability.**
    - Effort required to transfer the program from one hardware and/or software system environment to another.



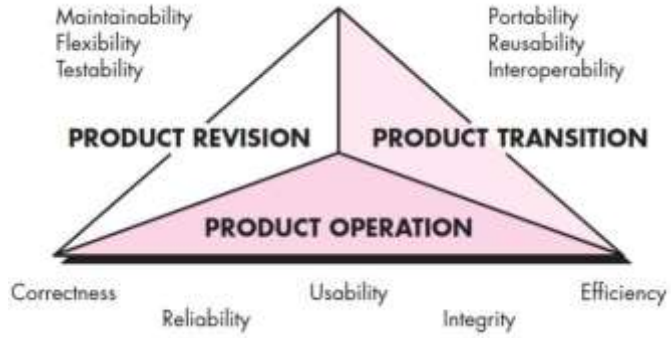
Software Quality Factors

- McCall's Quality Factors - Description
  - **Reusability.**
    - Extent to which a program [or parts of a program] can be reused in other applications—related to the packaging and scope of the functions that the program performs.



Software Quality Factors

- McCall's Quality Factors - Description
  - **Interoperability.**
    - Effort required to couple one system to another.



Software Quality Factors

- ISO 9126 Quality Factors
  - The ISO 9126 standard was developed in an attempt to identify the key quality attributes for computer software. The standard identifies six key quality attributes:
    1. **Functionality.**
       - The degree to which the software satisfies stated needs as indicated by the following subattributes: suitability, accuracy, interoperability, compliance, and security.

Software Quality Factors

- ISO 9126 Quality Factors
  - The ISO 9126 standard was developed in an attempt to identify the key quality attributes for computer software. The standard identifies six key quality attributes:

    2. Reliability.

    - The amount of time that the software is available for use as indicated by the following subattributes: maturity, fault tolerance, recoverability.

Software Quality Factors

- ISO 9126 Quality Factors
  - The ISO 9126 standard was developed in an attempt to identify the key quality attributes for computer software. The standard identifies six key quality attributes:

    3. Usability.

    - The degree to which the software is easy to use as indicated by the following subattributes: understandability, learnability, operability.

Software Quality Factors

- ISO 9126 Quality Factors
  - The ISO 9126 standard was developed in an attempt to identify the key quality attributes for computer software. The standard identifies six key quality attributes:
    4. Efficiency.
      - The degree to which the software makes optimal use of system resources as indicated by the following subattributes: time behavior, resource behavior.

Software Quality Factors

- ISO 9126 Quality Factors
  - The ISO 9126 standard was developed in an attempt to identify the key quality attributes for computer software. The standard identifies six key quality attributes:

    5. Maintainability.

    - The ease with which repair may be made to the software as indicated by the following subattributes: analyzability, changeability, stability, testability.

Software Quality Factors

- ISO 9126 Quality Factors
  - The ISO 9126 standard was developed in an attempt to identify the key quality attributes for computer software. The standard identifies six key quality attributes:

    6. Portability.

    - The ease with which the software can be transposed from one environment to another as indicated by the following subattributes: adaptability, installability, conformance, replaceability.

Software Quality Factors

VIT-AP UNIVERSITY

- Targeted Quality Factors
  - To conduct assessment, you'll need to address specific, measurable (or at least, recognizable) attributes of the interface
  - Example 1:

    Intuitiveness. The degree to which the interface follows expected usage patterns so that even a novice can use it without significant training
    - Is the interface layout conducive to easy understanding?
    - Are interface operations easy to locate and initiate?
    - Does the interface use a recognizable metaphor?
    - Is input specified to economize key strokes or mouse clicks?
    - Do aesthetics aid in understanding and usage?

Software Quality Factors

VIT-AP UNIVERSITY

- Targeted Quality Factors
  - To conduct assessment, you'll need to address specific, measurable (or at least, recognizable) attributes of the interface
  - Example 2:

    Efficiency. The degree to which operations and information can be located or initiated.

    - Does the interface layout and style allow a user to locate operations and information efficiently?
    - Can a sequence of operations (or data input) be performed with an economy of motion?
    - Are output data or content presented so that it is understood immediately?
    - Have hierarchical operations been organized in a way that minimizes the depth to which a user must navigate to get something done?

Software Quality Assurance

- SQC vs SQA (Software Quality Control vs Software Quality Assurance)
  - Quality control has now become an activity performed by people other than the ones who built the products
  - Quality assurance is focused more
    - The SQA group serves as the customer's in-house representative. That is, the people who perform SQA must look at the software from the customer's point of view.

Software Quality Assurance

VIT-AP UNIVERSITY

- Elements of SQA
  - Standards (e.g., IEEE, ISO)
  - Reviews and audits
  - Testing
  - Error/defect collection and analysis
  - Change management
  - Education
  - Vendor management
  - Security management
  - Safety
  - Risk management

Software Quality Assurance

- SQA Tasks, Goals, and Metrics
  - SQA Tasks
    - Prepares an SQA plan for a project
    - Participates in the development of the project's software process description
    - Reviews software engineering activities to verify compliance with the defined software process
    - Audits designated software work products to verify compliance with those defined as part of the software process
    - 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

Software Quality Assurance

- SQA Tasks, Goals, and Metrics
  - Goals, Attributes, and Metrics
    - The SQA actions described in the previous slide are performed to achieve a set of pragmatic goals:
      - **Requirements quality** – correctness, completeness, and consistency of the requirements model
      - **Design quality** – model should be assessed by the software team
      - **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** – team should apply limited resources in a way that has the highest likelihood of achieving a high-quality result

Software Quality Assurance

- SQA Tasks, Goals, and Metrics
  - Goals, Attributes, and Metrics

Goal	Attribute	Metric
Requirement quality	Ambiguity	Number of ambiguous modifiers (e.g., many, large, human-friendly)
	Completeness	Number of TBA, TBD
	Understandability	Number of sections/subsections
	Volatility	Number of changes per requirement
		Time (by activity) when change is requested
	Traceability	Number of requirements not traceable to design/code
	Model clarity	Number of UML models
		Number of descriptive pages per model
		Number of UML errors

Software Quality Assurance

VIT-AP UNIVERSITY

- SQA Tasks, Goals, and Metrics
  - Goals, Attributes, and Metrics

Goal	Attribute	Metric
Design quality	Architectural integrity	Existence of architectural model
	Component completeness	Number of components that trace to architectural model
		Complexity of procedural design
	Interface complexity	Average number of pick to get to a typical function or content
		Layout appropriateness
	Patterns	Number of patterns used

Software Quality Assurance

- SQA Tasks, Goals, and Metrics
  - Goals, Attributes, and Metrics

Goal	Attribute	Metric
Code quality	Complexity	Cyclomatic complexity
	Maintainability	Design factors (Chapter 8)
	Understandability	Percent internal comments
		Variable naming conventions
	Reusability	Percent reused components
	Documentation	Readability index

Software Quality Assurance

- SQA Tasks, Goals, and Metrics
  - Goals, Attributes, and Metrics

Goal	Attribute	Metric
QC effectiveness	Resource allocation	Staff hour percentage per activity
	Completion rate	Actual vs. budgeted completion time
	Review effectiveness	See review metrics (Chapter 14)
	Testing effectiveness	Number of errors found and criticality
		Effort required to correct an error
		Origin of error

Software Quality Assurance

- Statistical Software Quality Assurance
  - Reflects a growing trend throughout industry to become more quantitative about quality
  - For software, statistical quality assurance implies the following steps:
    1. Information about software errors and defects is collected and categorized.
    2. An attempt is made to trace each error and defect to its underlying cause (e.g., nonconformance to specifications, design error, violation of standards, poor communication with the customer).
    3. 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).
    4. Once the vital few causes have been identified, move to correct the problems that have caused the errors and defects

Software Quality Assurance

- Statistical Software Quality Assurance
  - Causes for the errors
    - 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 (IET)
    - Inaccurate or incomplete documentation (IID)
    - Error in programming language translation of design (PLT)
    - Ambiguous or inconsistent human/computer interface (HCI)
    - Miscellaneous (MIS)

Software Quality Assurance

- Statistical Software Quality Assurance
  - Causes for the errors

	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%

Statistical quality assurance techniques for software have been shown to provide substantial quality improvement.

In some cases, software organizations have achieved a 50 percent reduction per year in defects after applying these techniques.

The Capability Maturity Model

- What is it?
  - An SPI framework
  - SPI stands for *Software Process Improvement*

  - What is SPI?
    - Described in the next slide

The Capability Maturity Model

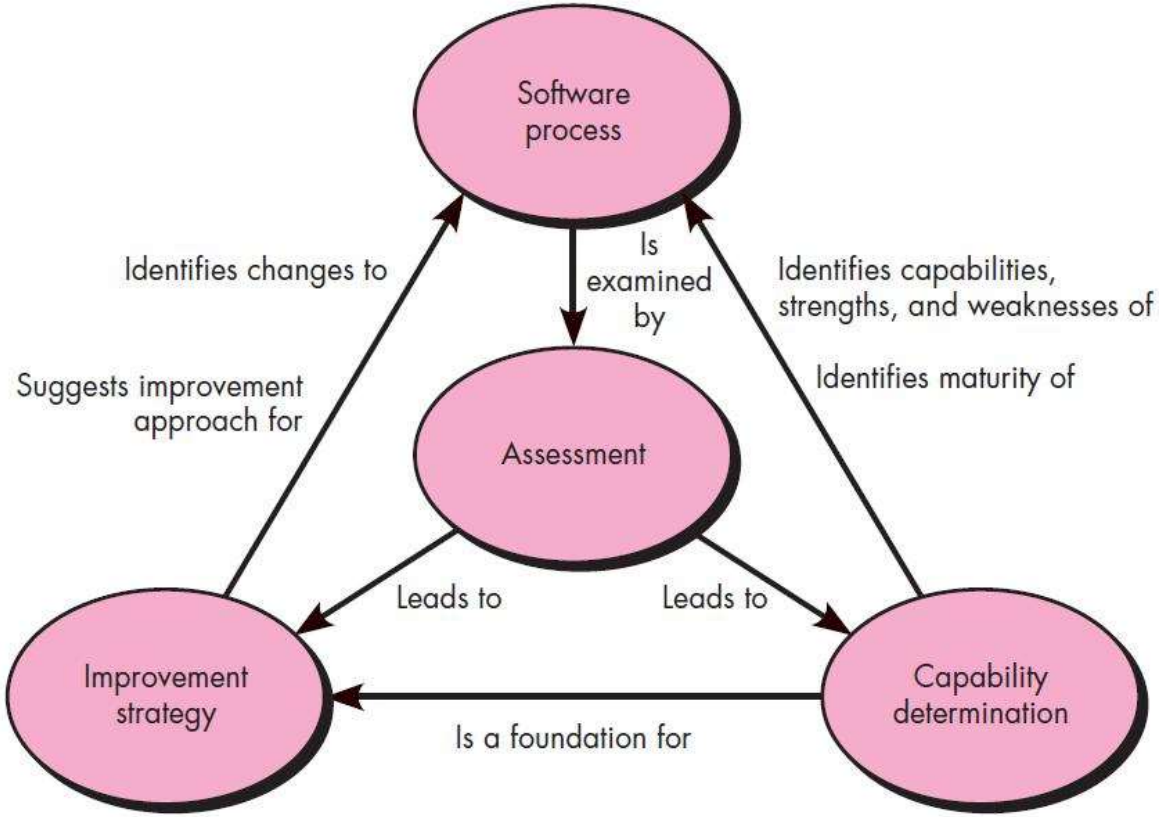
- SPI implies many things
  - The term SPI implies that
    1. elements of an effective software process can be defined in an effective manner
    2. an existing organizational approach to software development can be assessed against those elements
    3. a meaningful strategy for improvement can be defined

The Capability Maturity Model

- Approaches to SPI
  - Although an organization can choose a relatively informal approach to SPI, the vast majority choose one of a number of SPI frameworks
  - An SPI framework defines
    1. a set of characteristics that must be present if an effective software process is to be achieved
    2. a method for assessing whether those characteristics are present
    3. a mechanism for summarizing the results of any assessment
    4. a strategy for assisting a software organization in implementing those process characteristics that have been found to be weak or missing

The Capability Maturity Model

- An overview of a typical SPI framework



The Capability Maturity Model

- Conradi's six different SPI support constituencies
  - Quality certifiers
    - Process improvement efforts championed by this group focus on the following relationship: $Quality(Process) \Rightarrow Quality(Product)$
  - Formalists
    - This group wants to understand (and when possible, optimize) process workflow.
  - Tool advocates
    - This group insists on a tool-assisted approach to SPI that models workflow and other process characteristics in a manner that can be analyzed for improvement
  - Practitioners
    - This constituency uses a pragmatic approach, "emphasizing mainstream project-, quality- and product management, applying project level planning and metrics, but with little formal process modeling or enactment support"
  - Reformers
    - The goal of this group is organizational change that might lead to a better software process (focuses on human issues).
  - Ideologists
    - This group focuses on the suitability of a particular process model for a specific application domain or organizational structure.

The Capability Maturity Model

- Maturity Model
  - A maturity model is applied within the context of an SPI framework
  - The intent of the maturity model is to provide an overall indication of the "process maturity" exhibited by a software organization.

The Capability Maturity Model

- Maturity Model
  - Example:
    - Software Engineering Institute's **Capability Maturity Model** suggests five levels of maturity

    - **Level 5, Optimized**—The organization has quantitative feedback systems in place to identify process weaknesses and strengthen them pro-actively. Project teams analyze defects to determine their causes; software processes are evaluated and updated to prevent known types of defects from recurring.

The Capability Maturity Model

- Maturity Model
  - Example:
    - Software Engineering Institute's **Capability Maturity Model** suggests five levels of maturity

    - **Level 4, Managed**—Detailed software process and product quality metrics establish the quantitative evaluation foundation. Meaningful variations in process performance can be distinguished from random noise, and trends in process and product qualities can be predicted.

The Capability Maturity Model

- Maturity Model
  - Example:
    - Software Engineering Institute's **Capability Maturity Model** suggests five levels of maturity

    - **Level 3, Defined**—Processes for management and engineering are documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing software.

The Capability Maturity Model

VIT-AP UNIVERSITY

- Maturity Model
  - Example:
    - Software Engineering Institute's **Capability Maturity Model** suggests five levels of maturity
    - **Level 2, Repeatable**—Basic project management processes are established to track cost, schedule, and functionality. Planning and managing new products is based on experience with similar projects.

The Capability Maturity Model

- Maturity Model
  - Example:
    - Software Engineering Institute's **Capability Maturity Model** suggests five levels of maturity
    - **Level 1, Initial—**Few processes are defined, and success depends more on individual heroic efforts than on following a process and using a synergistic team effort.

The CMMI (Capability Maturity Model Integration)

- The original CMM was developed and upgraded by the Software Engineering Institute throughout the 1990s as a complete SPI framework
- Today, it has evolved into the Capability Maturity Model Integration (CMMI)
  - a comprehensive process meta-model that is predicated on a set of system and software engineering capabilities that should be present as organizations reach different levels of process capability and maturity.

The CMMI (Capability Maturity Model Integration)

- Six levels of CMMI
  - **Level 0: Incomplete—**the process area (e.g., requirements management) is either not performed or does not achieve all goals and objectives defined by the CMMI for level 1 capability for the process area.

The CMMI (Capability Maturity Model Integration)

- Six levels of CMMI
  - **Level 1: Performed—**all of the specific goals of the process area (as defined by the CMMI) have been satisfied. Work tasks required to produce defined work products are being conducted

The CMMI (Capability Maturity Model Integration)

- Six levels of CMMI
  - **Level 2: Managed—**all capability level 1 criteria have been satisfied. In addition, all work associated with the process area conforms to an organizationally defined policy; all people doing the work have access to adequate resources to get the job done; stakeholders are actively involved in the process area as required; all work tasks and work products are “monitored, controlled, and reviewed; and are evaluated for adherence to the process description”

The CMMI (Capability Maturity Model Integration)

- Six levels of CMMI
  - **Level 3: Defined—**all capability level 2 criteria have been achieved. In addition, the process is “tailored from the organization’s set of standard processes according to the organization’s tailoring guidelines, and contributes work products, measures, and other process-improvement information to the organizational process assets”

The CMMI (Capability Maturity Model Integration)

- Six levels of CMMI
  - **Level 4: Quantitatively managed—**all capability level 3 criteria have been achieved. In addition, the process area is controlled and improved using measurement and quantitative assessment. “Quantitative objectives for quality and process performance are established and used as criteria in managing the process”

The CMMI (Capability Maturity Model Integration)

- Six levels of CMMI
  - **Level 5: Optimized—**all capability level 4 criteria have been achieved. In addition, the process area is adapted and optimized using quantitative (statistical) means to meet changing customer needs and to continually improve the efficacy of the process area under consideration.

The CMMI (Capability Maturity Model Integration)

VIT-AP UNIVERSITY

Process areas required to achieve a maturity level

Level	Focus	Process Areas
Optimizing	*Continuous process Improvement*	Organizational innovation and deployment Causal analysis and resolution
Quantitatively managed	*Quantitative management*	Organizational process performance Quantitative project management
Defined	*Process standardization*	Requirements development Technical solution Product integration Verification Validation Organizational process focus Organizational process definition Organizational training Integrated project management Integrated supplier management Risk management Decision analysis and resolution Organizational environment for integration Integrated teaming
Managed	*Basic project management*	Requirements management Project planning Project monitoring and control Supplier agreement management Measurement and analysis Process and product quality assurance Configuration management
Performed		

References

- Roger Pressman, “Software Engineering: A Practitioner’s Approach”, McGraw-Hill, 7th Edition, 2016
  - Chapter: 14; Section: 14.1 – 14.2
  - Chapter: 16; Section: 16.1 – 16.3
  - Chapter: 30; Section: 30.1 – 30.3

Next

VIT-AP UNIVERSITY

Software Maintenance

VIT-AP UNIVERSITY

Thank You