



SE-3002

SOFTWARE QUALITY ENGINEERING

RUBAB JAFFAR

RUBAB.JAFFAR@NU.EDU.PK

Part IV-QUANTIFIABLE QUALITY IMPROVEMENT
Feedback Loop and Activities for Quantifiable Quality
Improvement

Lecture # 37, 38, 39
29,30 Nov, 1 Dec

TODAY'S OUTLINE

- Feedback Loop and Activities for Quantifiable Quality Improvement (Chapter 18)
 - Feedback Loop and Overall Mechanism
 - Monitoring and Measurement
 - Analysis and Feedback
 - Tool and Implementation Support
 - Analysis → Feedback loop → Follow-up
- Quality models and measurements (Chapter 19)
 - Types of Quality Assessment Models & Comparing Quality Assessment Models.
 - Data Requirements and Measurement
 - Measurement and Model Selection.
- Software Configuration Management (SCM)
 - Why SCM? , SCM concepts and Terminology, SCM Activities
 - Change Management, Build and Release Management
 - System Release

QUANTIFIABLE QUALITY IMPROVEMENT BASIC ELEMENTS

- Part IV is quantifiable quality improvement, which includes two basic elements:
- Quantification of quality through quantitative measurements and models so that the quantified quality assessment results can be compared to the pre-set quality goals for quality and process management.
- Quality improvement through analyses and follow-up activities by identifying quality improvement possibilities, providing feedback, and initiating follow-up actions.

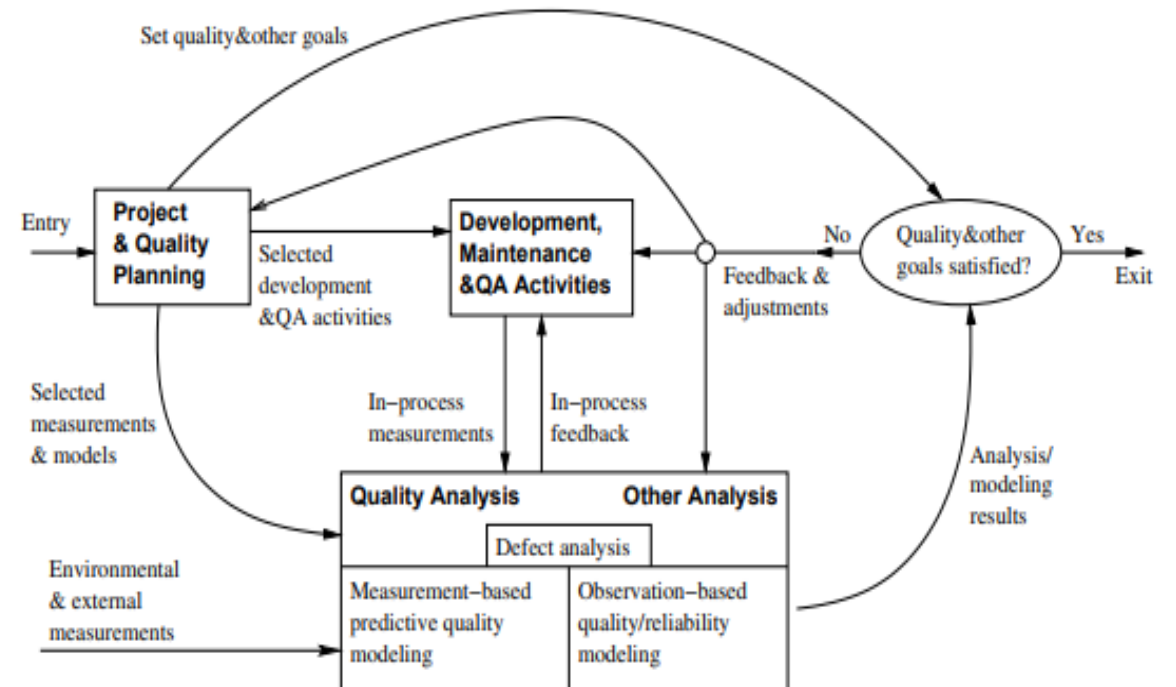
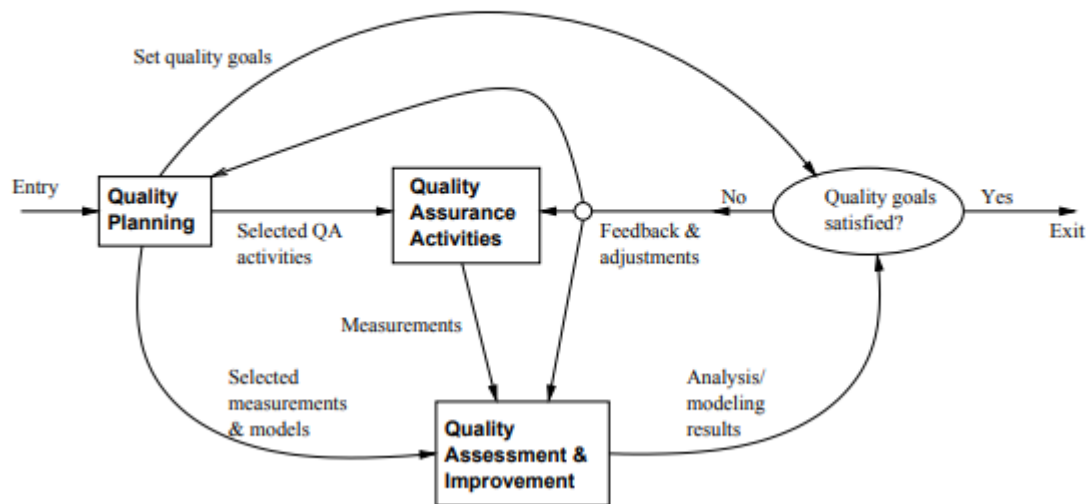
QUANTIFIABLE QUALITY IMPROVEMENT

- To support quantifiable quality improvement, various parallel and follow-up activities to the main quality assurance (QA) activities are needed, including:
- Monitoring the specific QA activities and the overall software development or maintenance activities, and extracting relevant measurement data.
- Analyzing the data collected above for quality quantification and identification of quality improvement opportunities.
- Providing feedback to the QA and development/maintenance activities and carrying out follow-up actions based on the analysis results above.

QUANTIFIABLE QUALITY IMPROVEMENT

- These activities also close the quality engineering feedback loop discussed in part I and refine it into Following:

Software quality engineering (SQE)



IMPORTANCE OF FEEDBACK LOOP

- All QA activities covered in Part II and Part III need additional support: .
- Planning and goal setting
- Management via feedback loop
 - When to stop?
 - Adjustment and improvement, etc.
 - All based on assessments/predictions

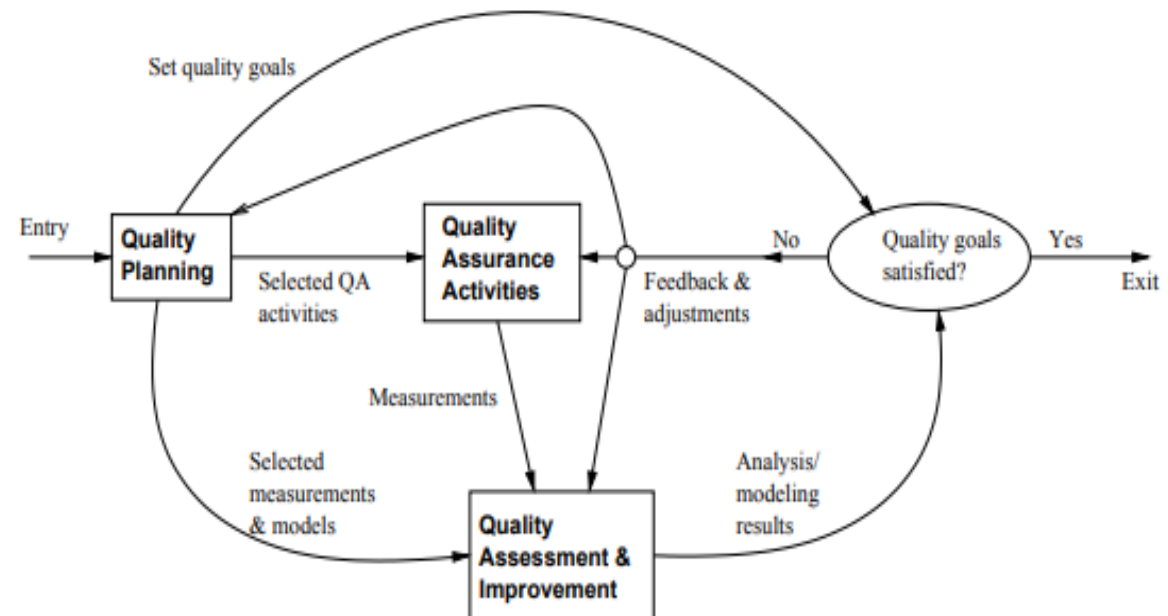
IMPORTANCE OF FEEDBACK LOOP

- Feedback loop for quantification/improvement: i.e. Focus of Part IV
 - mechanism and implementation
 - models and measurements.
 - defect analyses and techniques
 - risk identification techniques
 - software reliability engineering

QE ACTIVITIES AND PROCESS REVIEW

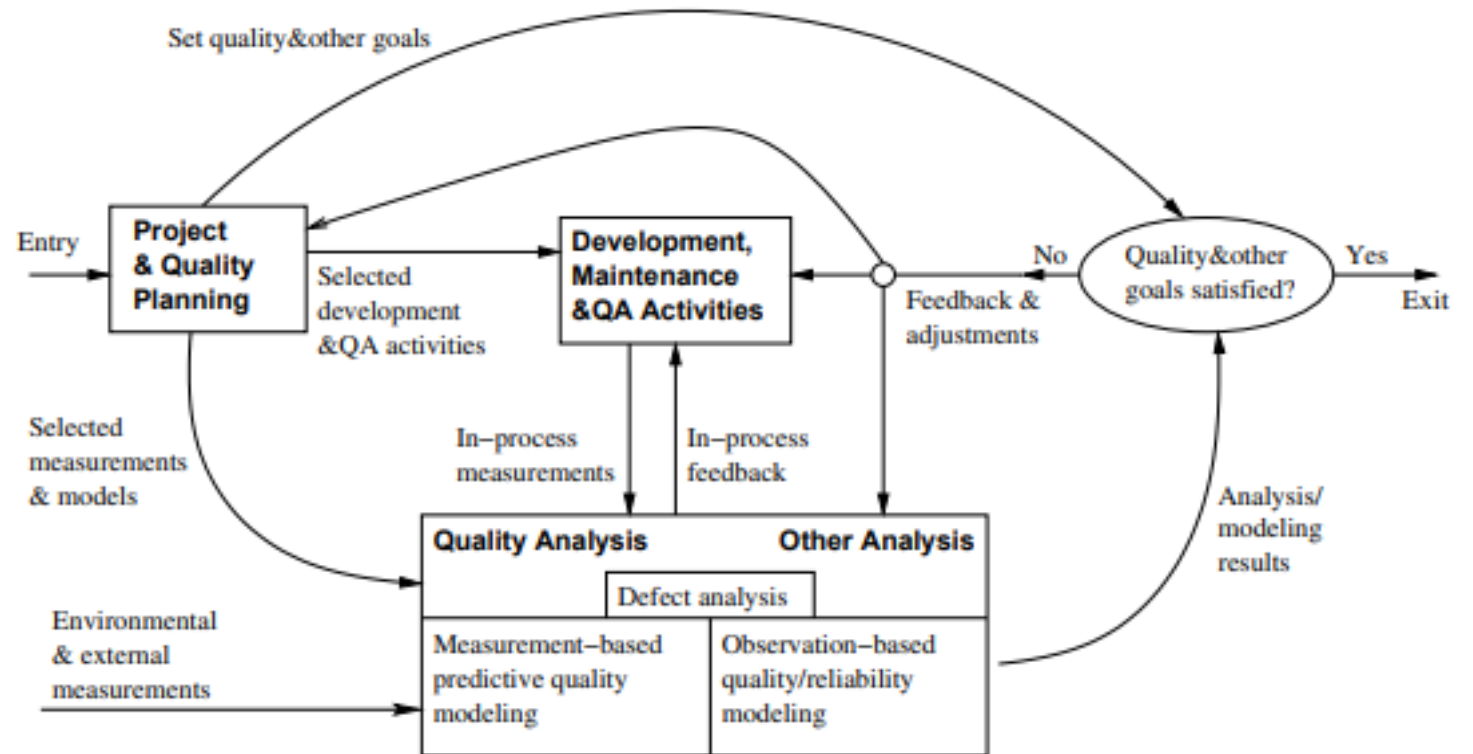
Software quality engineering (SQE)

- Major activities:
- Pre-QA planning (Part I)
- QA (Part II and Part III)
- Post-QA analysis & feedback – Part IV (maybe parallel instead of “post-”)



QE ACTIVITIES AND PROCESS REVIEW

- Feedback loop zoom-in:
- Multiple measurement sources.
- Many types of analysis performed.
- Multiple feedback paths.



FEEDBACK LOOP RELATED ACTIVITIES

- Monitoring and measurement: .
 - defect monitoring \in process management
 - defect measurement \in defect handling
 - many other related measurements.
- Feedback and follow-up
 - Frequent feedback: assessment/prediction
 - Possible improvement areas identified
 - Overall management and improvement.
- Analysis modeling:
 - Historical baselines and experience
 - Choosing models and analysis techniques
 - Focus on defect/risk/reliability analyses
 - Goal: assessment/prediction/improvement.

The primary purpose of QA monitoring is to ensure proper execution of the planned QA activities through the use of various measurements. These measurements also provide the data input to subsequent analysis and modeling activities.

QUALITY MONITORING AND MEASUREMENTS

- Quality monitoring needs:
 - Quality as a quantified entity over time
 - Able to assess, predict, and control
 - Various measurement data needed
- Some directly in quality monitoring
- Others via analyses to provide feedback.
- **Direct quality measurements:**
 - Total number of defects, fault count, reliability, Result & Defect measurement, impact and related info. e.g., success vs. failure
- Defect information: directly monitored.
- Mostly used in quality monitoring.

INDIRECT QUALITY MEASUREMENTS

- Indirect quality measurements: Why?
- Other quality measurements (reliability) need additional analyses/data.
- Unavailability of direct quality measurements early in the development cycle \Rightarrow early (indirect) indicators.
- Used to assess/predict/control quality. (to link to or affect various direct quality measurements)
- Types of indirect quality measurements:
 - Environmental measurements.
 - Product internal measurements.
 - Activity measurements.

INDIRECT MEASUREMENTS: ENVIRONMENT

- Process characteristics
 - The process used: waterfall, iterative, spiral etc.
 - Activities and their relationships
 - Preparation, execution and follow-up
 - Development Techniques used
- People characteristics
 - Skills and experience
 - Roles: planners/developers/testers
 - Process management and teams
- Product characteristics
 - general expectations of the target users
 - high-level product functionality
 - Product/market environment
 - Hardware/software configuration environment

INDIRECT MEASUREMENTS: INTERNAL



- Product internal measurements: most studied/understood in SE
- Software artifacts being measured:
 - Mostly code-related
 - Sometimes SRS, design, docs etc.
- Product attributes being measured:
 - Control: e.g., control flow paths
 - Data: e.g., operand count
 - Presentation: e.g., indentation rules

INDIRECT MEASUREMENTS:ACTIVITY

- Activity measurements directly measure specific software development and maintenance activities and the associated effort, time, and other resources: can be done at different levels of granularity:
- **Coarse-grain activity measurements** for the whole project. For example, total effort and cycle-time can be used in various models for overall quality assessment and project release decisions.
- **Medium-grain activity measurements** for individual development phases, sub-phases, or time periods such as weeks or months. For example, defect profiles over development phases are commonly used in various models for quality assessment, resource allocation, and project management.
- **Fine-grain activity measurements** for individual activities. For example, test work load assessments for individual test cases are used in various reliability models to provide valuable feedback to the software development process and QA activities

INDIRECT MEASUREMENTS:ACTIVITY

- Execution/activity measurements:
 - Overall: e.g., cycle time, total effort.
 - Phased: profiles/histograms.
 - Detailed: transactions
- Testing activity examples:
 - Timing during testing/usage
 - Path verification (white-box)
 - Usage-component mapping (black-box)
 - Measurement along the path
- Usage of observations/measurements: observation-based and predictive models

- 
- 
- Most of these direct and indirect quality measurements can be obtained from the QA activities or from the overall software development or maintenance processes. Therefore, they are classified as in-process measurements.

IMMEDIATE FOLLOW-UP AND FEEDBACK

- Immediate (without analyses):Why?
 - Immediate action needed right away:
 - critical problems ⇒ immediate fixing
 - most other problems: no need to wait
 - Some feedback as built-in features in various QA alternatives and techniques.(Process brain storming in Gilb inspection)
 - Activities related to immediate actions.
- Testing activity examples:
 - Shifting focus from failed runs/areas.
 - Re-test to verify defect fixing.
 - Other defect-related adjustments.
- Defect and activity measurements used.

ANALYSES AND FOLLOW-UP ACTIONS

- Most feedback/follow-up relies on analyses.
- There is a tight connection between the types of analyses performed and the feedback and follow-up activities in response to the specific analysis results. Therefore, we examine them together, linking specific analyses to specific feedback and follow-up actions.
- Types of analyses:
 - Product release decision related.
 - For other project management decisions, at the phase or overall project level.
 - Longer-term or wider-scope analyses.
- Types of feedback paths:
 - Shorter vs. longer feedback loops.
 - Frequency and time duration variations.
 - Overall scope of the feedback.
 - Data source refinement.
 - Feedback destinations.

ANALYSIS FOR PRODUCT RELEASE DECISIONS

- Most important usage of analysis results
 - Prominent in SQE and modified SQE Figure
 - Related to: “when to stop testing?”
- Basis for decision making:
 - **Without explicit quality assessment:**
 - Implicit quality assessment : planned activities,
 - Indirect quality assessment : coverage goals,
 - other factors for quality assessment : time/\$-based.
 - **With explicit quality assessment:**
 - failure-based quality assessment : reliability,
 - fault-based quality assessment : defect count & defect density.
- Criteria preference:
 - reliability – defect – coverage – activity.

ANALYSES FOR OTHER DECISIONS

- Transition from one (sub-)phase to another:
 - Later ones: similar to product release.(the focus is typically on the overall operation of the software system from the users' perspective.)
 - Earlier ones: reliability undefined (complete system not be operational yet. So the system reliability from a users' perspective may not be defined nor can it be assessed
 - defects – coverage – activity,
 - inspection and other early QA
- The quality assessment and modeling results can also be used to support various other management decisions, primarily in the following areas:
 - Schedule adjustment.
 - Resource allocation and adjustment.
 - Planning for post-release support.
 - Planning for future products or updates.
- These are product-level or sub-product-level decisions and activities

OTHER FEEDBACK AND FOLLOW-UP

- Other (less frequent) feedback/follow-up:
 - Goal adjustment (justified/approved).
 - Self-feedback (measurement & analysis)
 - Longer term, project-level feedback.
 - May even carry over to follow-up projects.
- Beyond a single-project duration/scope:
 - Future product quality improvement
 - overall goal/strategy/model/data,
 - especially for defect prevention.
- Process improvement.
- More experienced people

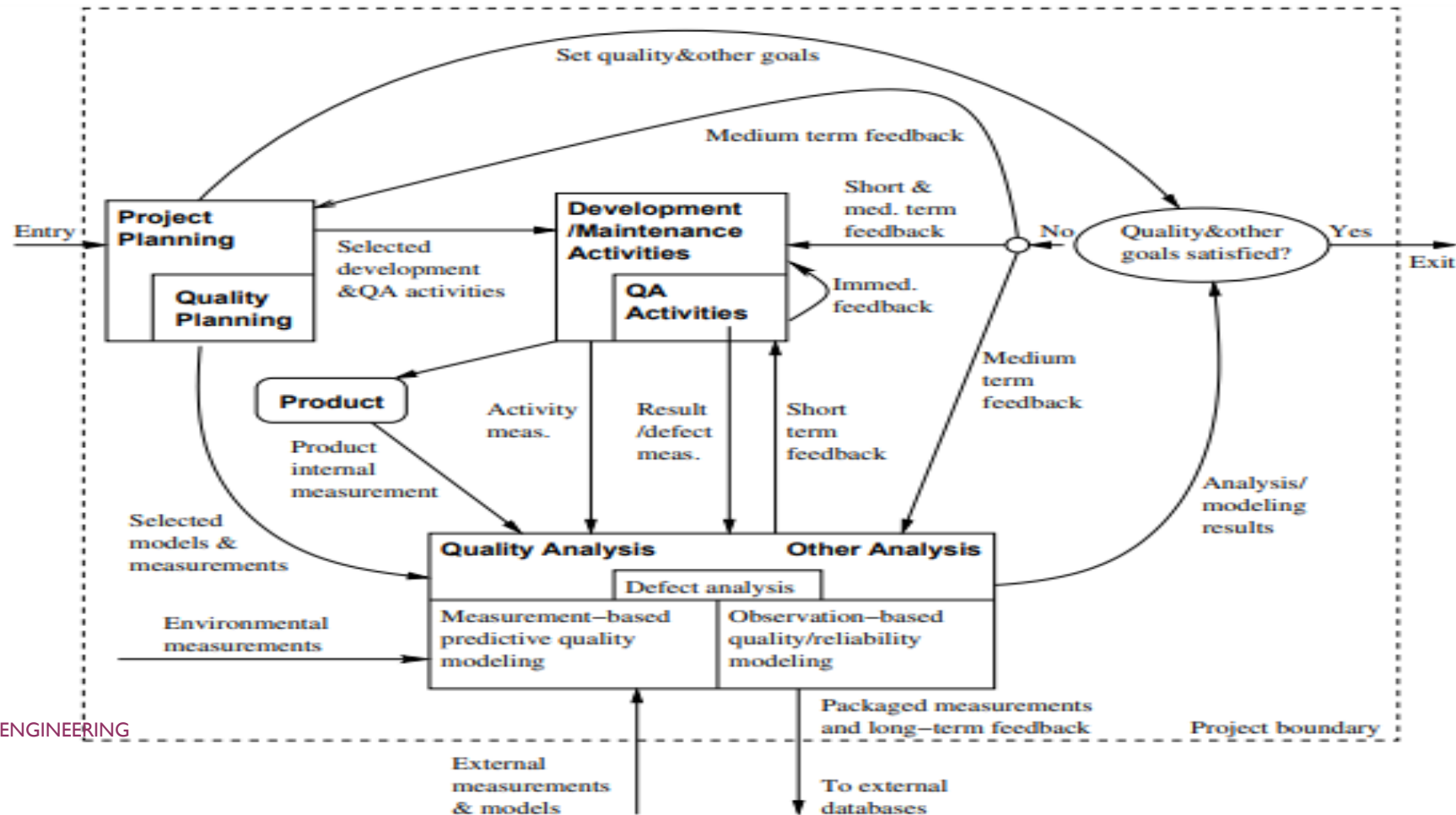
FEEDBACK LOOP IMPLEMENTATION

- Key question: sources and destinations. (Analysis and modeling activity at center.)
- Sources of feedback loop = data sources:
 - Result and defect data:
 - the QA activities themselves.
 - Activity data:
 - both QA and development activities.
 - Product internal data: product. (produced by development activities)
 - Environmental data: environment.
- Additional sources of feedback loop:
 - From project/QA planning.
 - Extended environment: measurement data and models beyond project scope.

FEEDBACK LOOP IMPLEMENTATION

- Feedback loop at different duration/scope levels.
- Immediate feedback to current development activities (locally).
- Short-term or sub-project-level feedback:
 - most of the feedback/follow-up
 - transition, schedule, resource,
 - destination: development activities.
- Medium-term or project-level feedback:
 - overall project adjustment and release
 - destination: major blocks
- Longer-term or multi-project feedback:
 - to external destinations

FEEDBACK LOOP IMPLEMENTATION



IMPLEMENTATION SUPPORT TOOLS

- Type of tools:
 - Data gathering tools.
 - Analysis and modeling tools.
 - Presentation tools.
- Data gathering tools:
 - Defects/direct quality measurements:
 - from defect tracking tools.
 - Environmental data: project db.
 - Activity measurements: logs.
 - Product internal measurements:
 - commercial/home-build tools.
 - New tools/APIs might be needed.

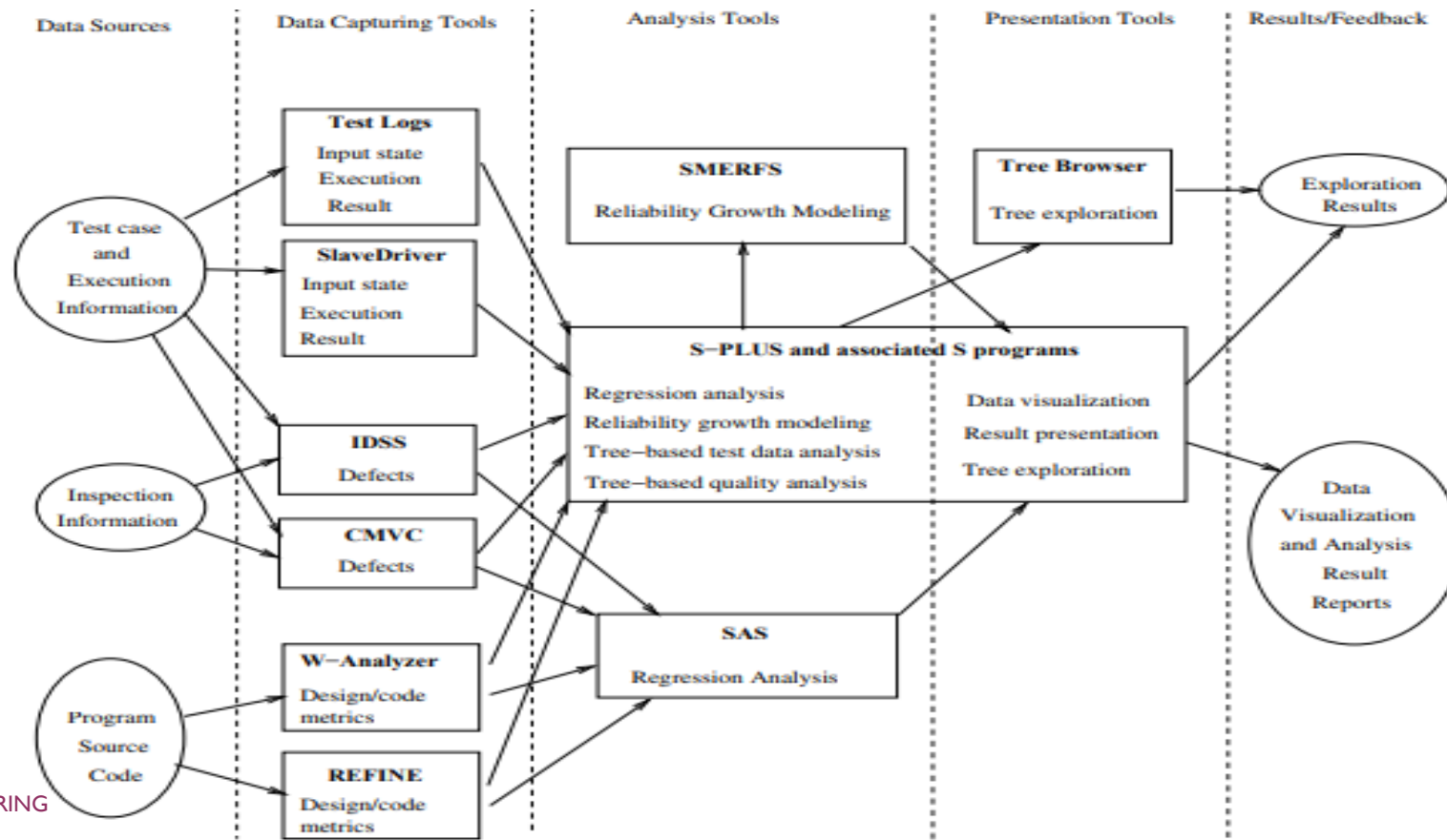
IMPLEMENTATION SUPPORT TOOLS

- Analysis and modeling tools:
 - Dedicated modeling tools:
 - e.g., SMERFS and CASRE for SRE
 - General modeling tools/packages:
 - e.g., multi-purpose S-Plus, SAS.
 - Utility programs often needed for data screening and processing.
- Presentation tools:
 - Aim: easy interpretation of feedback \Rightarrow more likely to act on.
 - Graphical presentation preferred.
 - Some “what-if”/exploration capability.

STRATEGY FOR TOOL SUPPORT

- Using existing tools \Rightarrow cost↓:
 - Functionality and availability/cost.
 - Usability.
 - Flexibility and programmability.
 - Integration with other tools.
- Tool integration issues:
 - Assumption: multiple tools used. (All-purpose tools not feasible/practical.)
 - External rules for inter-operability,
 - common data format and repository.
 - Multi-purpose tools.
 - Utilities for inter-operability.

TOOL SUPPORT EXAMPLE



QUALITY MODELS AND MEASUREMENT

- The primary purpose of the measurement and analysis activities is to provide feedback and useful information to manage software quality, the **quality engineering process**, and the **overall software development/maintenance** process and activities.
- The feedback and information provided are based on the analysis results using various models on the data collected from the quality assurance (QA) and the general development activities.
- In this lecture we examine and **classify** these models, **relate** them to the required measurements, **compare** the different models, and outline a **general strategy** to select appropriate models and measurements to satisfy specific quality assessment and improvement goals under specific application environments.

SOFTWARE QUALITY MODELS(BONUS ACTIVITY)

- Software Quality Models are a standardised way of measuring a software product.
- With the increasing trend in software industry, new applications are planned and developed everyday.
- This eventually gives rise to the need for reassuring that the product so built meets at least the expected standards.
- During this course, we have discussed some of the following main categories of software quality models:
- Quality Definition Models:
 - McCall
 - FURPS
 - ISO 9126
 - GQM
 - CMMI
- Your task is to compare the above models and provide a critical analysis based on comparisons.
- Handwritten submission only.

QUALITY ANALYSIS

- Analysis and modeling:
- . Quality models: data \Rightarrow quality
 - a.k.a. quality assessment models or quality evaluation models
- Various models needed for .
 - Assessment, prediction, control
 - Management decisions
 - Problematic areas for actions
 - Process improvement
- Measurement data needed
- Direct quality measurements: success/failure (& defect info)
- Indirect quality measurements:– activities/internal/environmental. Indirect but early quality indicators.
- All described in Chapter 18

MODELS FOR QUALITY ASSESSMENT

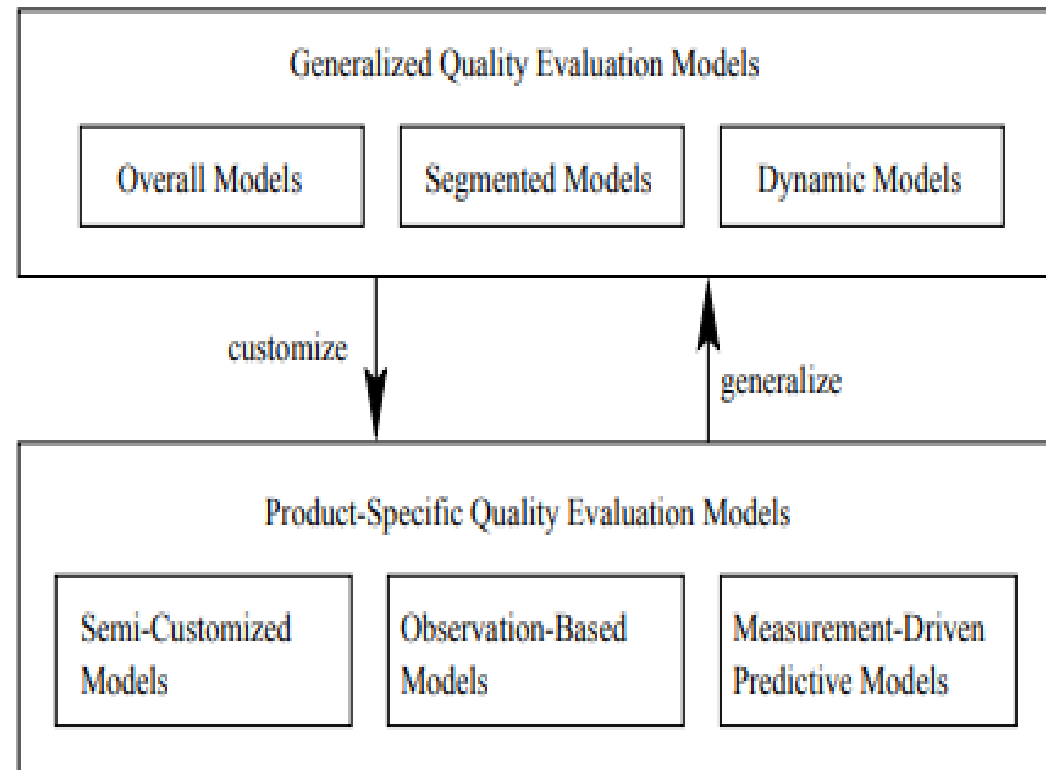
- Quality assessment models are analytical models that provide quantitative assessment of selected quality characteristics or sub-characteristics based on measurement data from software projects.
- Such models can help us obtain an objective assessment of our current product quality, in contrast to the often unreliable subjective assessment based on personal judgment or imprecise qualitative assessment.
- When applied over time, these models can provide us with an accurate prediction of the future quality, which can be used to help us make project scheduling, resource allocation, and other management decisions.

QUALITY MODELS

- Practical issues:
 - Applicability vs. appl. environment
 - Goal/Usefulness: information/results?
 - Data: measurement data required
 - Cost of models and related data
- Type of quality models
 - Generalized: averages or trends
 - Product-specific: more customized
 - Relating to issues above

GENERALIZED MODELS: OVERALL

- Model taxonomy:
- Generalized:
 - overall, segmented, and dynamic
- Product-specific:
 - semi-customized: product history
 - observation-based: observations
 - measurement-driven: predictive



GENERALIZED MODELS: OVERALL

- Key characteristics
 - Industrial averages/patterns: \Rightarrow (single) rough estimate.
 - Most widely applicable.
 - Low cost of use.
- Examples: Defect density.
 - Estimate total defect with sizing model.
 - (counting in-field unique defect only)
- Non-quantitative overall models:
 - As extension to quantitative models.
 - Examples: 80:20 rule, and other general observations.

GENERALIZED MODELS: SEGMENTED

- Key characteristics:
- Estimates via product segmentation.
- Model: segment \rightarrow quality.
- Multiple estimates provided.
- Other applications.
- Commonly used in software estimation.
- Example: COCOMO models.

A segmented model for reliability level estimation

Product Type	Failure Rate (per hour)	Reliability Level
safety-critical software	$< 10^{-7}$	ultra-high
commercial software	10^{-3} to 10^{-7}	moderate
auxiliary software	$> 10^{-3}$	low

GENERALIZED MODELS: DYNAMIC

- Example: Putnam model
- Rayleigh curve for failure rate: $r = 2Bate^{-at^2}$
- Overall/average trend over time.
- Often expressed as a mathematical function or an empirical curve.
- Combined models possible,
- e.g., segmented dynamic models.

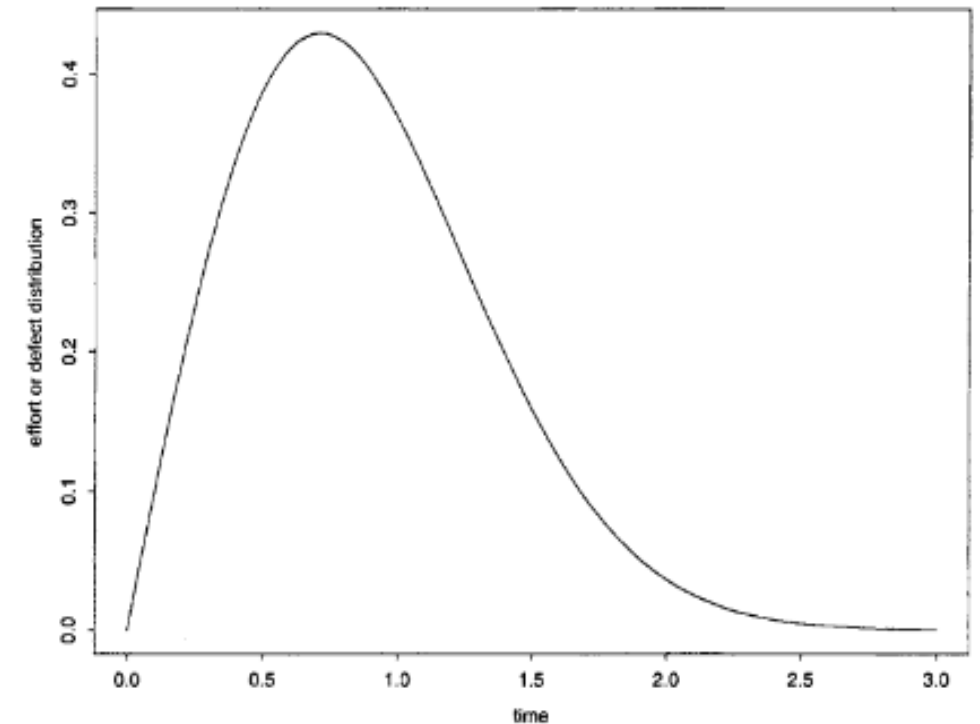


Figure 19.2 Effort or defect profile in the Putnam Model

PRODUCT-SPECIFIC MODELS (PSM)

- Product-specific info. Used (vs. none used in generalized models)
- Better accuracy/usefulness at cost ↑
- Three types:
 - semi-customized
 - observation-based
 - measurement-driven predictive
- Connection to generalized models (GMs):
 - Customize GMs to PSMs with new/refined models and additional data.
 - Generalize PSMs to GMs with empirical evidence and general patterns.

SEMI-CUSTOMIZED MODELS

- Semi-customized models:
 - Project level model based on history.
 - Data captured by phase.
 - Both projections and actual.
 - Linear extrapolation.

Table 19.2 DRM (defect removal model): defect distribution for previous releases of a product

Requirement	Design	Coding	Testing	Support
5%	10%	35%	40%	10%

- The information from these semi-customized models can be directly used to predict defect distribution for the current release.

PSM: OBSERVATION-BASED

- Observation-based models relate observations of the software system behavior to information about related activities to provide more precise quality assessments.
- Examples of such models include various software reliability growth models (SRGMs) where observed failures and associated time intervals are fitted to SRGMs to evaluate product reliability.
- Focus on the effect/observations
- Assumptions about the causes
- Assessment-centric
- Example: Goel-Okumoto SRGM
 - functional relation: $m(t) = N(1 - e^{-bt})$
 - observed failures over time
 - curve fitting
 - reliability assessment/prediction
 - management decisions: exit criteria

PSM: PREDICTIVE

- Measurement-driven predictive models
 - Establish predictive relations between quality and other measurement on the historical data
 - Provide early prediction of quality, and identify problems early so that timely actions can be taken to improve product quality.
 - Modeling techniques: regression, TBM etc.
 - Risk assessment and management
- Model characteristics:
 - Response: chief concern
 - Predictors: observable/controllable
 - Linkage quantification

Table 19.3 High-defect modules for two products identified by tree-based

Product	Subset	#Modules	Mean-DF
LS	lrrr	16	9.81
	rlr	53	10.74
	rr	17	22.18
	whole product	1296	1.8
NS	rlll	8	55.0
	rr	5	77.0
	whole product	995	7.9

MODEL COMPARISON AND INTERCONNECTIONS

- Different types of quality assessment models and their relations can be compared by looking at their ability to provide useful information, their applicability to different project environments, and their inter-connections.
- we can compare the following:
 - Usefulness of the modeling results, in terms of how accurate the quality estimates are and the applicability of the models to different environments.
 - Model inter-connections, which can be examined in two opposite directions:
 - Customization of generalized quality models to provide better quality estimates when product-specific information is available.
 - Generalization of product-specific models when enough empirical evidence from different products or projects is accumulated.

SUMMARY OF QUALITY ASSESSMENT MODELS AND THEIR APPLICATIONS

The usefulness of a model mainly depends on two factors:

- Is it applicable to the specific development environment and the specific product under development or maintenance?
- If so, how accurate the model is in its quality estimates?

Table 19.4 Summary of quality assessment models and their applications

Model Type	Sub-Type	Primary Result	Applicability
generalized models		rough quality estimates	all or by industry
	overall	overall product quality	across industries
	segmented	industry-specific quality	within an industry
	dynamic	quality trend over time	trend in all
product-specific quality models		better quality estimates	specific product
	semi-customized	quality extrapolation	prev→cur release
	observation-based	quality assessments	current product
	measurement-driven	quality predictions	both above

MODEL COMPARISON AND INTERCONNECTIONS

- This usefulness can be weighted against its cost, in particular, the cost of collecting the required measurement data, which is typically the dominant part of the modeling cost.
- Generalized models provide rough quality estimates based on empirical data from industry.
- Product-specific models provide more precise quality assessments using product-specific measurements.
- However, generalized models are more widely applicable and less expensive to use than product-specific models, because they do not require product-specific measurements.
- Consequently, generalized models may be more useful in the product planning stage, and in the early phases of product development, when most product-specific data are unavailable.

MODEL COMPARISON AND INTERCONNECTIONS

- One exception to this general rule is when there exist historical data for the previous releases of the current product. Semi-customized models can be used to provide better estimates under this situation.
- As development or maintenance activities progress, more measurement data can be collected and various detailed quality models in the category of product-specific models, such as observation-based models and measurement-based predictive models, can be used to better manage the QA activities as well as the overall software development or maintenance processes.

SOFTWARE TESTING METRICS

- **Software Testing Metrics** are the quantitative measures used to estimate the progress, quality, productivity and health of the software testing process.
- Metric defines in quantitative terms the degree to which a system, system component, or process possesses a given attribute. E.g. Total number of defects
- **Types of Test Metrics**
- **Process Metrics:** It can be used to improve the process efficiency of the SDLC (Software Development Life Cycle)
- **Product Metrics:** It deals with the quality of the software product
- **Project Metrics:** It can be used to measure the efficiency of a project team or any testing tools being used by the team members



TEST METRICS GLOSSARY

- **Rework Effort Ratio** = (Actual rework efforts spent in that phase/ total actual efforts spent in that phase) X 100
- **Requirement Creep** = (Total number of requirements added/No of initial requirements)X100
- **Schedule Variance** = (Actual Date of Delivery – Planned Date of Delivery)
- **Cost of finding a defect in testing** = (Total effort spent on testing/ defects found in testing)
- **Schedule slippage** = (Actual end date – Estimated end date) / (Planned End Date – Planned Start Date) X 100
- **Passed Test Cases Percentage** = (Number of Passed Tests/Total number of tests executed) X 100
- **Failed Test Cases Percentage** = (Number of Failed Tests/Total number of tests executed) X 100
- **Blocked Test Cases Percentage** = (Number of Blocked Tests/Total number of tests executed) X 100
- **Fixed Defects Percentage** = (Defects Fixed/Defects Reported) X 100
- **Accepted Defects Percentage** = (Defects Accepted as Valid by Dev Team /Total Defects Reported) X 100
- **Defects Deferred Percentage** = (Defects deferred for future releases /Total Defects Reported) X 100
- **Critical Defects Percentage** = (Critical Defects / Total Defects Reported) X 100
- **Average time for a development team to repair defects** = (Total time taken for bugfixes/Number of bugs)
- **Number of tests run per time period** = Number of tests run/Total time
- **Test design efficiency** = Number of tests designed /Total time
- **Test review efficiency** = Number of tests reviewed /Total time
- **Bug find rate or Number of defects per test hour** = Total number of defects/Total number of test hours

SOFTWARE CONFIGURATION MANAGEMENT

- Configuration Management is the *control* of the evolution of complex software.
- SCM is the discipline that enable us **to keep evolving software products under control**, and thus contributes to satisfying quality and delay constraints.
- There is a need for storing the different components of a software product (artifacts) and all their versions safely.

SCM DEFINITION

- A discipline applying technical and administrative direction and surveillance to:
 - Identify and document the functional and **physical characteristics** of a configuration item
 - Control **changes** to those characteristics
 - Record and report **change processing** and implementation status
 - Verify compliance with specified requirements

Definition from IEEE

SOFTWARE CONFIGURATION MANAGEMENT

- In software engineering, software configuration management (SCM) is the task of **tracking** and **controlling** changes in the software.
- SCM practices include **revision control** and the establishment of **baselines**.
- If something goes wrong, SCM can determine **what** was changed and **who** changed it

VERSION CONTROL

- A component of software configuration management, version control, also known as revision control or source control, is the management of changes to documents, computer programs, large web sites, and other collections of information.
- Changes are usually identified by a number termed the "revision number", "revision level", or simply "revision".
- For example, an initial set of files is "revision 1". When the first change is made, the resulting set is "revision 2", and so on.
- Each revision is associated with a timestamp and the person making the change. Revisions can be compared, restored, and with some types of files, merged.

SOFTWARE CONFIGURATION MANAGEMENT

- **When you develop software, change is inevitable.**
- **Change increase the level of confusion among development team.**
- **Confusion arise when changes are not **analyzed** before they made, **recorded** before they are implemented.**

SOFTWARE CONFIGURATION MANAGEMENT



Why client is running the wrong version of the software ...

Oops! a bug that was fixed in the software suddenly reappears ...

I don't know why a developed and tested functionality is missing ...

SCM CONCEPTS

- **Configuration**
- **Configuration Control**
- **Software Configuration Item (SCI)**
- **Baselines**
- **Version (or variants)**

SCM CONCEPTS

Configuration

- The functional and *physical characteristics* of hardware or software as set forth in technical documentation or achieved in a product. (ANSI/IEEE 828, 1990)

SCM CONCEPTS

Configuration Control

- An element of configuration management, consisting of evaluation, coordination, approval or disapproval and implementation of changes to configuration items *after formal establishment* of their configuration identifications.

(ANSI/IEEE 828, 1990)

SCM CONCEPTS

■ Configuration Item

“An aggregation of hardware, software, or both, that is designated for configuration management and treated as a single entity in the configuration management process.”

❖ Software configuration items are not only program code segments but all type of documents according to development, e.g

- ↙ all type of code files
- ↙ drivers for tests
- ↙ analysis or design documents
- ↙ user or developer manuals
- ↙ system configurations (e.g. version of compiler used)

❖ In some systems, not only software but also hardware configuration items (CPUs, bus speed frequencies) exist!

SCM CONCEPTS

Baseline

- **Baseline** is a specification or product that has been formally *reviewed* and *agreed* upon, that thereafter serves as the basis for further development and that can be changed only through formal change procedures.
- **e.g. SRS, Design Doc, Source Code etc**

SOFTWARE CONFIGURATION MANAGEMENT - WHY

- **New versions of software systems are created as they change**
 - **For different machines/OS**
 - **Offering different functionality**
 - **Tailored for particular user requirements**
- **Configuration management is concerned with managing evolving software systems**
 - **System change is a team activity**
 - **CM aims to control the costs and effort involved in making changes to a system**

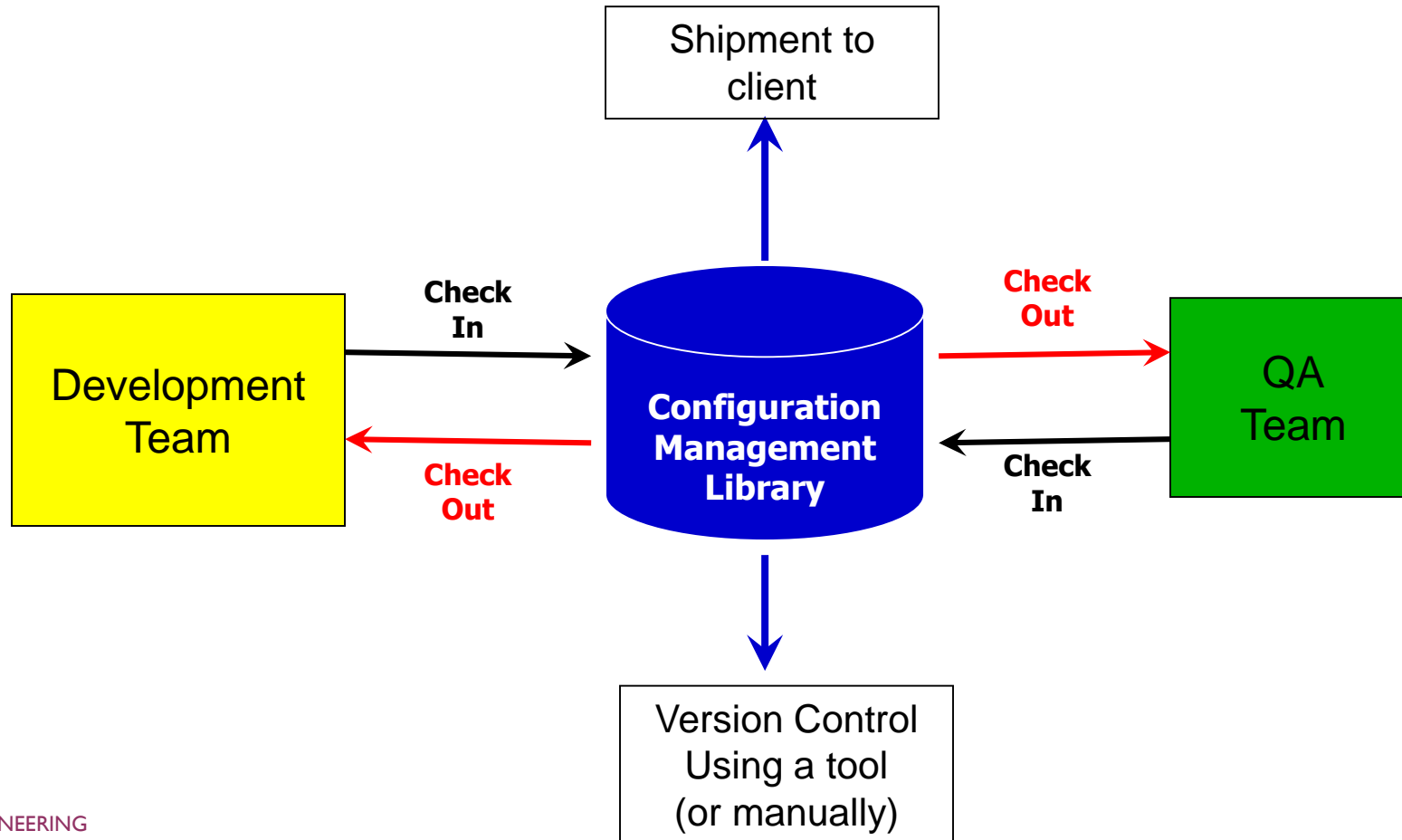
SOFTWARE CONFIGURATION MANAGEMENT – WHY?

- Involves the development and application of **procedures** and **standards** to manage an evolving software product
- May be seen as part of a more general quality management process
- When released to **CM**, software systems are sometimes called **baselines** as they are a starting point for further development

CONFIGURATION MANAGEMENT LIBRARY

- **The CM Library describes the central storage facility for all CI's.**
- **The CM Library contains the;**
 - **current baseline project,**
 - **approved documentation,**
 - **artifact files to keep the project running smoothly**

SCM CONCEPT



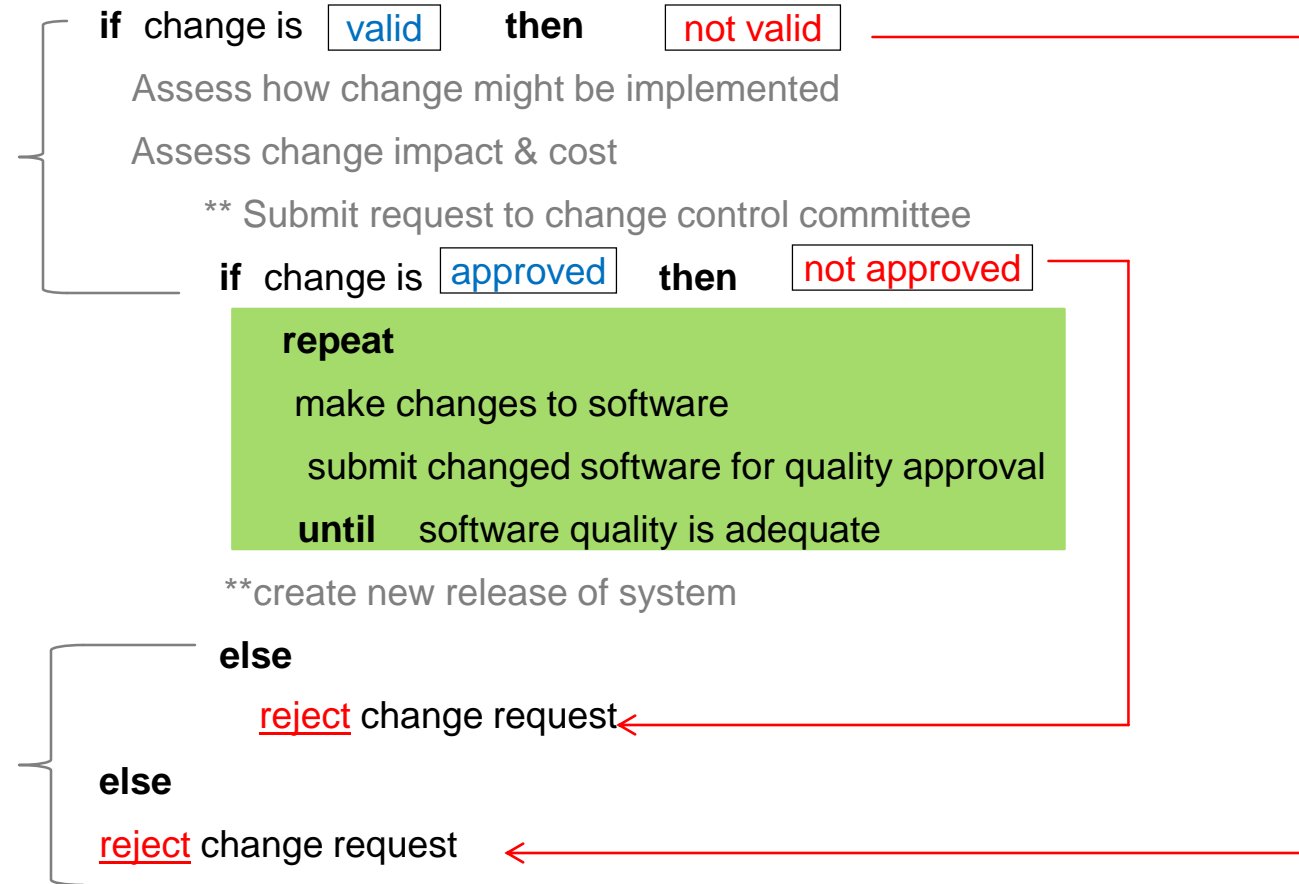
SCM ACTIVITIES

1. **Identify** the changes
2. **Control** the change
3. Ensure that change is **properly implemented**
4. **Report changes** to others involved in the project

CHANGE MANAGEMENT

- **Software systems are subject to continual change requests**
 - From users
 - From developers
 - From market forces
- **Change management is concerned with keeping track of these changes and ensuring that they are implemented in the most cost-effective way**

CHANGE MANAGEMENT PROCESS



SOFTWARE CHANGE STRATEGIES

- The software change strategies that could be applied separately or together are:
- **Software maintenance:** The changes are made in the software due to requirements.

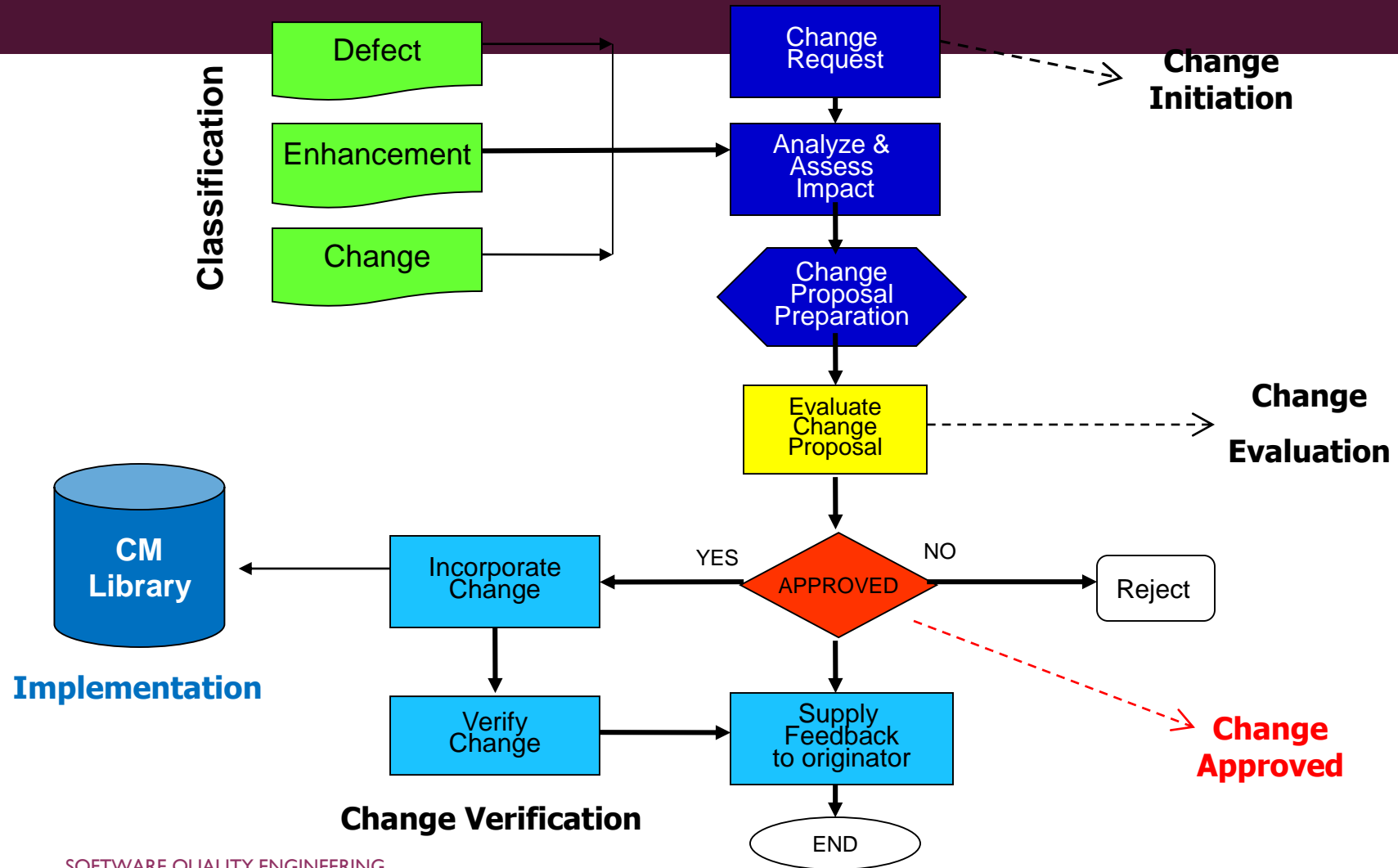
Types of maintenance?

- **Architectural transformation:** It is the process of changing one architecture into another form.
- **Software re-engineering:** New feature can be added to existing system and then the system is reconstructed for better use of it in future.

TYPES OF MAINTENANCE

- Maintenance is defined as the process in which changes are implemented by either modifying the existing system's architecture or by adding new components to the system.
- **Corrective Maintenance:** Means the maintenance for correcting the software faults.
- **Adaptive maintenance;** Means maintenance for adapting the change in environment.
- **Perfective maintenance:** Means modifying or enhancing the system to meet the new requirements.
- **Preventive maintenance:** Means changes made to improve future maintainability.

CHANGE CONTROL





That is all