SENG 521
Software Reliability & Software Quality

Chapter 9: Strategies to Meet Reliability Objective

Department of Electrical & Computer Engineering, University of Calgary

B.H. Far (far@ucalgary.ca)
http://www.enel.ucalgary.ca/People/far/Lectures/SENG521

far@ucalgary.ca
Contents

- Fault tolerance techniques
  - Definition, goal and problems
  - History
  - Fault tolerance process
  - Recovery techniques
  - Design techniques
- Techniques to improve product and process
  - ISO 9000-3
Review: Terminology

Dependability
  - Treats
    - Failures
    - Faults
    - Errors
  - Attributes
    - Availability
    - Reliability
    - Safety
    - Confidentiality
    - Integrity
    - Maintainability
  - Means
    - Fault prevention
    - Fault tolerance
    - Fault removal
    - Fault forecasting
  - Models
    - Reliability Block Diagram
    - Fault Tree model
    - Reliability Graph
Section 1

Fault Tolerant Software Systems
Fault Tolerance: RAID

- Standalone
- Hot swap
- Cluster

- RAID 0
- RAID 1
- RAID 5
- RAID 0+1

**Fault Tolerance: RAID**
Fault Tolerance Terminology

- Recovery
  - Backward
  - Forward

- Redundancy
  - Architectural
    - Hardware redundancy
    - Software redundancy
    - Data redundancy
    - Temporal redundancy
  - Functional
    - Serial
    - Parallel
    - Sequential
A fault-tolerant computing system must be capable of providing specified services in the presence of a bounded number of failures.

Use of techniques to enable continued delivery of service during system operation

Based on the principle of

- *Act during operation* while
- *Defined during specification and design*
The failures could occur because faults are present in either the components of the system or in the system’s design.

Building large computing systems is a complex task; fault-tolerance requirements could make the task even more difficult unless appropriate system structuring concepts are utilized.

Reliability growth (modeling, computation and interpretation) of a system featuring fault tolerance is different from a system without such feature.
Problems …

- The traditional approaches to fault tolerance in hardware systems have been based on coping with the effects of well-understood failure modes of physical components.
- Conventional hardware fault tolerance methods (e.g., redundancy) are rarely powerful enough to cope with design deficiencies. E.g., designing a square wheel!
- Consequently, most hardware fault tolerance techniques cannot be applied directly in software, where almost all faults are design faults.
Example: Consistency Check

\[ \mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{6} x_i \cdot y_i \]

\[ \mathbf{x} = \left( 10^{20}, 1223, 10^{24}, 10^{18}, 3, -10^{21} \right) \]

\[ \mathbf{y} = \left( 10^{30}, 2, -10^{26}, 10^{22}, 2111, 10^{19} \right) \]

- The correct answer should be 8779.
- But ordinary implementation of this will return zero due to rounding and large differences in the order of magnitude of the summands.
History ...

- **Defensive programming:**
  - Implementing relatively ad-hoc methods used to minimize the damage which could arise from the damage of presence of residual bugs.

- **Dual software technique:**
  - Implementing two distinct versions of the same software and executing them. Any discrepancy in the outputs of the two versions may trigger an alarm.

- Etc.
Fault Tolerance Process

1. Detection
   - Identify faults and their causes (errors)

2. Assessment
   - Assess the extent to which the system state has been damaged or corrupted

3. Recovery
   - Remain operational or regain operational status

4. Fault treatment and continued service
   - Locate and repair the fault to prevent another occurrence
Fault Tolerance Phases /1

- **Phase 1: Fault detection**
  - For a fault to be tolerated, it must first be detected. Thus the starting point for fault-tolerance techniques is observing failures.

1. **Concurrent**
   - Look for errors during service delivery
   - e.g., self-testing techniques: duplicate codes, module pairs

2. **Preemptive**
   - Look for errors when service is suspended
   - e.g., spare-checking, audit program
Phase 2: Damage assessment

It is necessary to assess the extent to which the system state has been damaged or corrupted.

If the delay involved between the manifestation of a fault (failure) and the detection of its cause (error) is large then it is likely that the damage to the system state will be more severe than if the latency interval were shorter.
Phase 3: Recovery

- Error recovery techniques must be utilized in order to obtain a normal, error-free system state.
- There are two different kinds of recovery technique.

1. Backward recovery technique consists of discarding the current (corrupted) state in favor of an earlier state. Therefore, mechanisms are needed to record and store system states. e.g., roll-back.

2. Forward recovery technique involves making use of the current (corrupted) state to construct an error-free state.
Phase 4: Fault treatment & continued service

- Once recovery has been undertaken, it is essential to ensure that the normal operation of the system will continue without the fault immediately manifesting itself once more.
- The first aspect of fault treatment is to attempt to locate the fault.
- Following this, steps can be taken either to repair the fault or to reconfigure the rest of the system to avoid the fault.
Example

- **Fault tolerance:** simple bug tracking
  1. **Detection:** acceptance test (a Boolean expression) is used.
  2. **Damage assessment:** only the program in execution is assumed to be affected.
  3. **Recovery:** (backward in this case) consists of recovering the state of the executing program to that at the beginning of the recovery block.
  4. **Fault treatment:** the program in execution (primary or alternative) is assumed to be faulty, so its faults are avoided by executing the next alternative (if any).
Section 2

Fault Tolerance: Techniques
Definitions

Recovery
- Actions to restore the system state to a “correct state”
- Recovery usually requires consistency checking
- Usually the first choice in software

Redundancy
- Designing the system with multiple components with the same functionality
- Usually the second choice in software
Consistency Check

- A program-specific error detection mechanism to check on the results of program execution.
- Usually evaluates to either “true” or “false”.

\[ \text{ensure}<\text{acceptance test}> \text{by } P_0 \text{ else-by } P_1 \text{ else fail} \]
Example: Consistency Check

- Checksums for program parts or split packages
- Internal check points:
  \[ \text{ABS}[(\text{SQRT}(x)\times\text{SQRT}(x)) - x] < E \]
- Exception signal when dividing by zero
- Integer overflow signal
- Interrupt signal for program loop
- Float point numerical failure check
Example: Consistency Check

\[ \mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{6} x_i \cdot y_i \]

\[ \mathbf{x} = \left( 10^{20}, 1223, 10^{24}, 10^{18}, 3, -10^{21} \right) \]
\[ \mathbf{y} = \left( 10^{30}, 2, -10^{26}, 10^{22}, 2111, 10^{19} \right) \]

- The correct answer should be 8779.
- But ordinary implementation of this will return zero due to rounding and large differences in the order of magnitude of the summands.
Backward Recovery

- Roll back the system to a previously saved correct state

Laura L. Pullum: Software Fault Tolerance Techniques and Implementation, Artech House, 2001
Domino Effect

- Why backward recovery is not always possible?
- **Domino Effect**: successive rollback of communicating processes when a failure is detected in any one of the processes.

Laura L. Pullum: Software Fault Tolerance Techniques and Implementation, Artech House, 2001
Forward Recovery

- Use redundancy to recover from a failure

Laura L. Pullum: Software Fault Tolerance Techniques and Implementation, Artech House, 2001
Forward Recovery: Pros & Cons

Advantages:
- Forward recovery is fairly efficient in terms of the overhead (time and memory) it requires. This can be crucial in real-time applications where the time overhead of backward recovery can exceed stringent time constraints.
- If the fault is an anticipated one, such as the potential loss of data, then redundancy and forward recovery can be a useful and timely approach.
- Faults involving missed deadlines may be better recovered from using forward recovery than by introducing additional delay in roll back and recovering.

Disadvantages:
- Application-specific, that is, it must be tailored to each situation or program.
- Can only remove predictable errors from the system state.
- Requires knowledge of the error.
- Cannot aid in recovery if the state is damaged beyond recoverability.
- Depends on the ability to accurately detect the occurrence of a fault (thus initiating the recovery actions.)
Redundancy

- Redundancy: designing the system with multiple components with the same functionality

- **Redundancy techniques:**
  - Implementing two (or more) distinct versions of the same software and executing them for the same set of inputs. Any discrepancy in the outputs of the two versions may trigger an alarm.

- Redundancy techniques’ efficiency depends on *coincident* and *correlated* faults.
Types of Redundancy

- **Hardware redundancy**
  - Replicated and supplementary hardware added to the system to support fault tolerance.

- **Software redundancy**
  - Also called program, modular, or functional redundancy, includes programs, modules, functions used to support fault tolerance.

- **Data redundancy**
  - Using additional forms of data to assist in fault tolerance.

- **Temporal redundancy**
  - Using additional time to perform tasks related to fault tolerance, i.e. repeating an execution using the same software and hardware resources involved in the initial, failed execution.
1. Coincident Faults

- **Coincident Faults**: when two or more functionally equivalent software components fail on the same input.

- When two or more software versions give the same incorrect response, an *identical-and-wrong* (IAW) answer is obtained.
2. Correlated Faults

- **Correlated Faults:** Two faults are correlated when the measured probability of the coincidence failures is significantly higher than what would be expected from the individual failure.

\[
p(i\_fails | j\_fails) > p(i\_fails)
\]

There will be no failure independence.
Failure Scenario

- What if the software components produce doublet or triplet identical-and-wrong (IAW) responses?

Adjudication Algorithm

- Doublet & triplet
- IAW faults

Input space for each procedure
Adjudication by Voting

- A voter compares results from two or more functionally equivalent software components and decides which of the answers provided by those components is correct.

- Various versions of voting algorithm:
  - Majority voting
  - Consensus voting
  - 2-of-N voting
Majority Voting

- Several identical components are structured in parallel and all are active. If the component outputs are not identical, the minority components are ignored (i.e., disabled or switched off).

- Majority voting:

\[ m \geq \left\lceil \frac{(N+1)}{2} \right\rceil, N > 1 \]

- System reliability \((R_{\text{system}})\) for majority voting (assuming components with identical reliability \(R_c\))

\[
R_{\text{system}} = 1 - \left(1 - R_c\right)^m \quad \text{where} \quad m = \left\lceil \frac{N+1}{2} \right\rceil
\]
Consensus Voting

- If majority agreement is achieved, select this answer
- If unique maximum agreement is achieved but 
  
  \[ m < \left\lceil \frac{(N+1)}{2} \right\rceil, \]
  select the unique maximum (\( m \) is the ceiling value)
- If tie in the maximum agreement number is achieved, 
  select randomly
- System reliability (\( R_{\text{system}} \)) for consensus voting (assuming components with identical reliability \( R_c \))

\[
R_{\text{system}} = 1 - \left(1 - R_c\right)^m
\]

\( m \) is the number of unique maximum components
2-of-N Voting

- Agreement number $m$ can be set to 2 if the output space is large and statistical independence of variant failures can be assumed.

- System reliability ($R_{\text{system}}$) for 2-of-N voting (assuming components with identical reliability $R_c$)

\[
R_{\text{system}} = 1 - \left(1 - R_c\right)^2
\]
Design Techniques

1) Robust software systems
2) Recovery blocks
3) N-version programming
4) Consensus recovery block
5) Acceptance voting
6) N-self-checking programming
Robust Software Systems (Anderson and Lee 1981, etc.):

Construction of a robust module requires:

- **Exception handlers** for coping with exceptions propagated from lower levels; and
- **Boolean expressions** for detecting exceptions arising in the module itself, and their exception handlers.

It is often possible (and desirable for the sake of simplicity) to map several exceptions onto a single handler.
1) Robust Software Systems

- For a given module, carefully analyze the cases that could prevent the module from providing the desired normal services.
- Make use of exception handlers either to mask the effects of such undesired, but expected, exceptions or to signal an appropriate exception to the caller of the module.
- Make use of default exception handlers or recovery blocks to obtain a measure of tolerance against design faults.
2) Recovery Blocks (RB)

- Using multiple versions of software module and acceptance test.
- The output of the 1\textsuperscript{st} module is tested for acceptability and if fails, the 2\textsuperscript{nd} module is executed after backward state recovery.
- The system fails only if all modules fail on their acceptance tests.
3) N-Version Programming (NVP)

- Parallel execution of N independently developed functionally equivalent modules.
- Adjudication is via voting.
- The voter accepts all N outputs and selects the correct one among them, i.e., the one that meets the specification.
- **Advantage of NVP:** no service interrupt.

Figure from *Reliability Engineering Handbook*
4) Consensus Recovery Block

- Combination of N-version programming (NVP) and recovery blocks (RB).
- IF NVP fails, the system reverts to RB using the same blocks.
- **Advantage:** highest possible system reliability.
5) Acceptance Voting

- Like N-version programming (NVP) all versions are executed in parallel.
- The output of each module goes to an acceptance test.
- If acceptance test is successful, the output goes to a voter.

Figure from Reliability Engineering Handbook
6) N-Self-Check Programming

- In N-Self-Check Programming (NSCP), N modules are executed in pairs.
- The pairs’ outputs can be compared or accessed for correctness.

Figure from Reliability Engineering Handbook
Discussion

- The capability of tolerating design faults rests largely on the ‘coverage’ of run-time checks (i.e. acceptance tests) for detecting errors.

- Often, it is not possible to check completely within a procedure that the results produced have been according to the specification (e.g., for a “sort” algorithm that sorts its input, the check that the output has been sorted correctly would be as complex as the “sort” algorithm itself).

- Hence run-time checks are often limited to checking certain critical aspects of the specification.

- This means that the possibility of undetected failures cannot be ruled out entirely.
Fault Tolerance: Adjudication by voting
Section 3

Software Product & Process Improvement using ISO 9000-3
How to Develop Better Software?

- Define development process
- Establish standards (requirements, design, coding, testing)
- Follow best practices
- Review & audit
- Follow ISO 9000-3 guidelines
The Software Engineering Institute (SEI) Capability Maturity Model (CMM)
Seven Development Tips

1. Keep the human network up and running
2. Constantly look for and plug time/effort leaks
3. Establish functional contracts (who checks what)
4. Test early, but not too early
5. Support manual testing with automated tools
6. Use automated code checkers/generators
7. Write stub code where possible
Review: Fault Prevention

- To avoid fault occurrences by construction.
- Activities:
  - Requirement review
  - Design review
  - Clear code
  - Establishing standards (ISO 9000-3, etc.)
  - Using CASE tools with built-in check mechanisms

All these activities are included in ISO 9000-3
ISO 9000 Family

- **ISO 9000**: Quality management and quality assurance standards: guidelines for selection and use
- **ISO 9000-1**: Revision of ISO 9000 (1994)
- **ISO 9002**: Quality systems: models for quality assurance in production, installation and servicing (1994)
ISO 9000 Family /2

- **ISO 9003**: Quality systems: models for quality assurance in final inspection and test (1994)
- **ISO 9004**: Quality management and quality system elements —guideline— (1987)
- **ISO 9000-3**: Guidelines for application of ISO 9001 to the development, supply and maintenance of software (1991)
My First Impression
Overview

QMS Continual Improvement

CUSTOMERS

Requirements

Management responsibility

Resource management

Measurement, analysis, improvement

Input

Product realization

Output

Product

Suppliers

CUSTOMERS

Satisfaction
ISO9000-3: Organization of Guidelines

1) Scope
2) Normative References
3) Definitions
4) Quality System: Framework
5) Quality System: Life-Cycle Activities
6) Quality System: Supporting Activities

Core parts
1. Scope

- This part of ISO 9000 sets out guidelines to facilitate the application of ISO 9001 to organizations developing, supplying and maintaining *software*.

- It is intended to provide guidance where a contract between two parties requires the demonstration of a supplier’s capability to develop, supply and maintain software products.
1. Scope

The guidelines are applicable in contractual situations for software products when:

- The contract specifically requires design effort and the product requirements are stated principally in performance terms, or they need to be established; i.e., identifying product requirements in a quantifiable and testable manner.

- Confidence in the product can be attained by the adequate demonstration of a certain supplier’s capabilities in development, supply and maintenance.
2. Normative References

- This section identifies the provisions that are referenced in the text and are part of the ISO 9000.

- The references are as follows:
  - ISO 2382-1: Data processing Vocabulary Part 01 Fundamental terms (1984)
  - ISO 9001: Quality systems-Model for quality assurance in design/development, production, installation and servicing (1987)
3. Definitions

- **Software**: Intellectual creation comprising the programs, procedures, rules and any associated documentation pertaining to the operation of a data processing system.

- **Software product**: Complete set of computer programs, procedures and associated documentation and data designated for delivery to a user.

- **Software item**: Any identifiable part of a software product at an intermediate step or at the final step of development.
3. Definitions

- Development: All activities to be carried out to create a software product.
- Phase: Defined segment of work.
- Verification (for software): The process of evaluating the products of a given phase to ensure correctness and consistency with respect to the products and standards provided as input to that phase. Building the "right" thing
- Validation (for software): The process of evaluating software to ensure compliance with specified requirements. Building the thing "right"
4.1 Management Responsibility

- Both senior supplier (= developer) and purchaser (= customer) management are aware of their responsibilities.

- The purchaser management should
  - Ensure the supplier has all the purchaser-specific information needed to meet contractual obligations, and
  - Identify one purchaser representative responsible for supplier interface.
4.1 Management Responsibility

- The supplier (= developer) management must
  - Create an engineering environment with clearly identified roles and responsibilities for the engineers who work in the environment
  - Identify and provide the resources needed to verify the engineering work being performed is accurate and complete
  - Ensure that defined practices and procedures are being followed
  - Take part in the review of the engineering and engineering practices and procedures to ensure their suitability and effectiveness.
4.2 Quality System

Management must identify its organization’s goals and ensure the existence of an engineering environment where those goals can be reached in the most efficient manner.

Engineering environment should have:

- Defined processes and procedures;
- Development and maintenance plans based on the defined process and procedures;
- Reviews, audits, and tests to determine the quality of the product(s) being created and the process used to create those products;
- Corrective actions based on the information gained from reviews, audits, and tests.
4.3 Internal System Audits

- The supplier’s organization should have an *internal audit process* to ensure that the engineering process used by the company is in fact meeting the company’s product and process quality goals.

- Documented findings from the audits are to be reviewed and acted upon by those responsible for the areas or processes audited.
4.4 Corrective Action

- A “closed loop” management process should be in place to ensure that:
  - Causes of quality problems are determined
  - Actions are taken to control the problems with the product
  - Address the changes in practices and procedures required to avoid recurrence of the problem.
5. Quality System:
Life-Cycle Activities
5.1 General

- All development projects (and maintenance projects) should follow an organized life cycle (or process).

- Suppliers are free to use any life cycle (process) they deem best suited for the type of product being developed, or maintained, as long as consideration is given to the various activities referred to in Sections 5.2 through 5.10 as life-cycle activities.
5.2 Contract Review

- The following needs are identified:
  - The need for the purchaser and supplier to come to an agreement
  - The need to identify methods for resolving contract issues that may arise
- Both purchaser and supplier management need to understand
  - The scope of the contract
  - Its organization’s responsibilities
  - Risks to organizations (e.g., schedule, budget, legal)
  - Ownership of the product and by-products
5.3 Purchaser’s Requirements

- This section focuses on the need to identify
  - The product’s functional and technical requirements (e.g., performance, safety, reliability, etc.)
  - The product’s external interface
- The purchaser and supplier have a responsibility to work together to ensure these requirements are complete and quantifiable before product development begins.
5.4 Development Planning

- Once purchaser’s requirements have been identified, there needs to be a development plan to deliver a product that meets those requirements.

- The development plan identifies the resources and schedule required to deliver a product.

- The resources and schedule are based upon a combination of:
  - Purchaser’s requirements
  - Engineering practices, and procedures used by the supplier to meet those requirements
  - Purchaser’s need date for the product.
5.4 Development Planning

- Development plan should show:
  - The phases of development
  - Inputs and outputs to each phase
  - Schedule and resources for each phase
  - Progress status and control
  - Tools and methods to be used, and
  - Verification procedures for each phase (reviews, audits, and testing)
5.4 Development Planning

- **Progress Control**
  - Briefly discusses the need to ensure the proper researching of a project throughout the entire life cycle.

- **Input to Development Phases**
  - Points out that each development phase should have defined inputs and that each product requirement should be stated in such a manner as to be quantifiably tested.
5.4 Development Planning

- **Output from Development Phases**
  - The outputs of one step in the software development process are the inputs into the next step.
  - These outputs should be validated to ensure that they meet the requirements of the next step.

- **Verification of Each Phase**
  - The output of each phase should be tested in some manner to ensure its fitness for use in the next phase.
5.5 Quality Planning

- Plans be defined to ensure that activities related to the quality of a development effort’s products or by-products take place.

- The plans for these activities can be a separate plan (software quality assurance plan) or incorporated in other plans like the development plan, test plan, and configuration management plan.
5.5 Quality Planning /2

- Typical list of quality planning activities:
  - Defining inputs and outputs for each development phase.
  - Identifying the types of test to be carried out.
  - Identifying the resources, schedules, and roles and responsibilities for carrying out the tests.
  - Configuration management.
  - Defect control and corrective action.
5.6 Design & Implementation

- Design and implementation are the processes that turn requirements into a product.
- Design is the technical kernel and to a great degree dictates the quality of the product.
- The design effort and the product itself would benefit from considering the following:
  - Design methodologies
  - Design rules and guidelines
  - Internal design (not seen by the user), comparison to previous designs
5.6 Design & Implementation

- **Implementation**
  - The supplier establish and use guidelines for subjects such as naming conventions, coding, and comments.

- **Reviews**
  - The supplier should review the products of analysis, design, implementation, and testing in order to ensure that the final product meets the purchaser's requirements.
  - Reviews and inspections are also meant to ensure that the methodologies and rules that were meant to be used during design were actually used.
5.7 Testing & Validation

- A multilevel testing process may be used to test a product and that a plan should be in place to support controlling the testing process.

- Test results should be recorded and used in order to:
  - Identify problems with the product being tested
  - Identify areas where tests need to be rerun
  - Determine the adequacy of the test process
5.7 Testing & Validation

Test Planning

There need to be plans in place to support this process. The plans should address:

- Types of testing
- Test cases
- Test environment
- Resources and schedule required to create the tests
- Resources and schedule required to execute the tests
- Test completion criteria
5.7 Testing & Validation

Validation

- Validation is the testing that is performed by the supplier on a version of the product that is intended to be delivered to the purchaser.
- Software testing can occur before validation, but that type of testing is verification of a product’s components as opposed to validation of an entire product.
5.7 Testing & Validation

- **Field Testing**
  - Field testing takes place at a site other than the supplier’s that is as close to an operational environment as possible.
  - The field tests need to be planned and that the supplier and purchaser may need to coordinate their efforts in the support of this type of testing.
5.8 Acceptance (Testing)

- Acceptance testing is performed by the purchaser.
- This should be a formal process planned well in advance of the actual testing.
- An acceptance test plan should identify the schedule, resources, roles and responsibilities, success criteria, and problem handling procedures.
- The supplier and purchaser have a shared responsibility for testing that goes on in this phase and must work closely together.
5.9 Delivery & Installation

- Replication and delivery are processes that are performed after a product has been developed or enhanced.
- Installation, may require coordination between the purchaser and the supplier.
- The level of coordination depends upon the complexity of the product and the number of purchaser sites that use the product.
- Installation planning should address schedules, available personnel, site access, availability, access to systems and equipment, and testing.
5.10 Maintenance

- Product maintenance, or enhancement, has the same components as product development.
- Analysis, design, implementation, and testing of changes to the product must all be planned, scheduled, and performed.
- The purchaser and the supplier must agree as to the timing and content of products releases so that both the supplier and purchaser can support the rate of product release.
6. Quality System:
Supporting Activities
Supporting Activities

- Configuration management
- Document control
- Quality records
- Measurement
- Rules, practices, and conventions
- Tools, facilities, and techniques
- Purchasing
- Included software
- Training
6.1 Configuration Management

- Configuration management is the process by which a product’s baselines (e.g., requirements, source code, test cases, test results, user documentation, etc.) are identified and changes to those baselines are controlled.

- The engineering organization has to identify, define, and plan for:
  - Identification of product baselines
  - Version control of the product baselines
  - Roles and responsibilities of the engineering organizations in the change process
  - Change control procedures
  - Status of the change control processes and baseline products
6.2 Document Control

- Engineering is a specification-driven process. There are a number of different engineering documents used in the development and maintenance of a software product.
- The engineering organization has to identify and control the use of these documents.
- Control is especially important for the initial approval and dissemination of such documents and the authorization and reissue of updated versions.
6.3 Quality Records

- Engineering organizations should maintain records that document the quality of their processes and products.
- Records of reviews, audits, and test results should be maintained to allow for their use in process and organizational improvement.
- The procedures and processes should be identified and implemented to control the accumulation, storage, and retrieval of such documents.
Engineering management should be a closed-loop process where measurements are taken to determine the quality of the **products** and **processes** used to create or manage those products.

The **product measurement** is based on purchaser/customer feedback and internal audits performed by the supplier organization.

These measurements are needed for product and process improvement.
6.4 Measurement

- The **process measurement** is used to determine whether schedule milestones are being met and whether the by-products of the process are meeting their quality goals.

- For measurements to be useful an organization needs to identify:
  - The current level of performance
  - Improvement goals
  - Measurement data to be collected
  - Actions to be taken based on measuring data against goals.
6.5 Rules, Practices & Conventions

- Supplier’s organization should
  - Document the engineering practices and procedures that are used to develop and maintain software products
  - Revise the practices and procedures as necessary.
6.6 Tools & Techniques

- Supplier’s organization should
  - Identify and use “tools, facilities, and techniques” in the development and management of software products
  - Replace or improve the tools, facilities, and techniques as necessary
6.7 Purchasing

- When purchasing third-party products, the supplier should apply many of the same management and engineering techniques as the purchaser uses in relation to the supplier.

- For example, the supplier (now acting as a purchaser) should ensure the subcontractor’s ability to perform the contracted work and validate the subcontractor’s products.
6.8 Included Software

- A supplier may have third-party products that need to be integrated with its own products.
- The supplier should identify procedures to ensure these products meet stated quality goals and that procedures and plans are in place for the storage, protection, and maintenance of the third-party products.
6.9 Training

- Engineers need to be trained in order to meet their responsibilities.

- An engineering organization must identify the techniques, tools, procedures, and methodologies that are used in the development and maintenance of a product.

- Once these have been identified, then a program can be instituted to ensure that engineers are proficient in their use.
Conclusions

- ISO 9000-3 *does not guarantee* quality software products.
- But those who follow guidelines and procedures of ISO 9000-3 are *more likely* to develop more reliable software products in a more consistent way.
Establishing standards (ISO 9000-3)