SOFTWARE QUALITY Assurance & Test

LECTURE # 7

QUALITY ASSURANCE

chenbo@etao.net

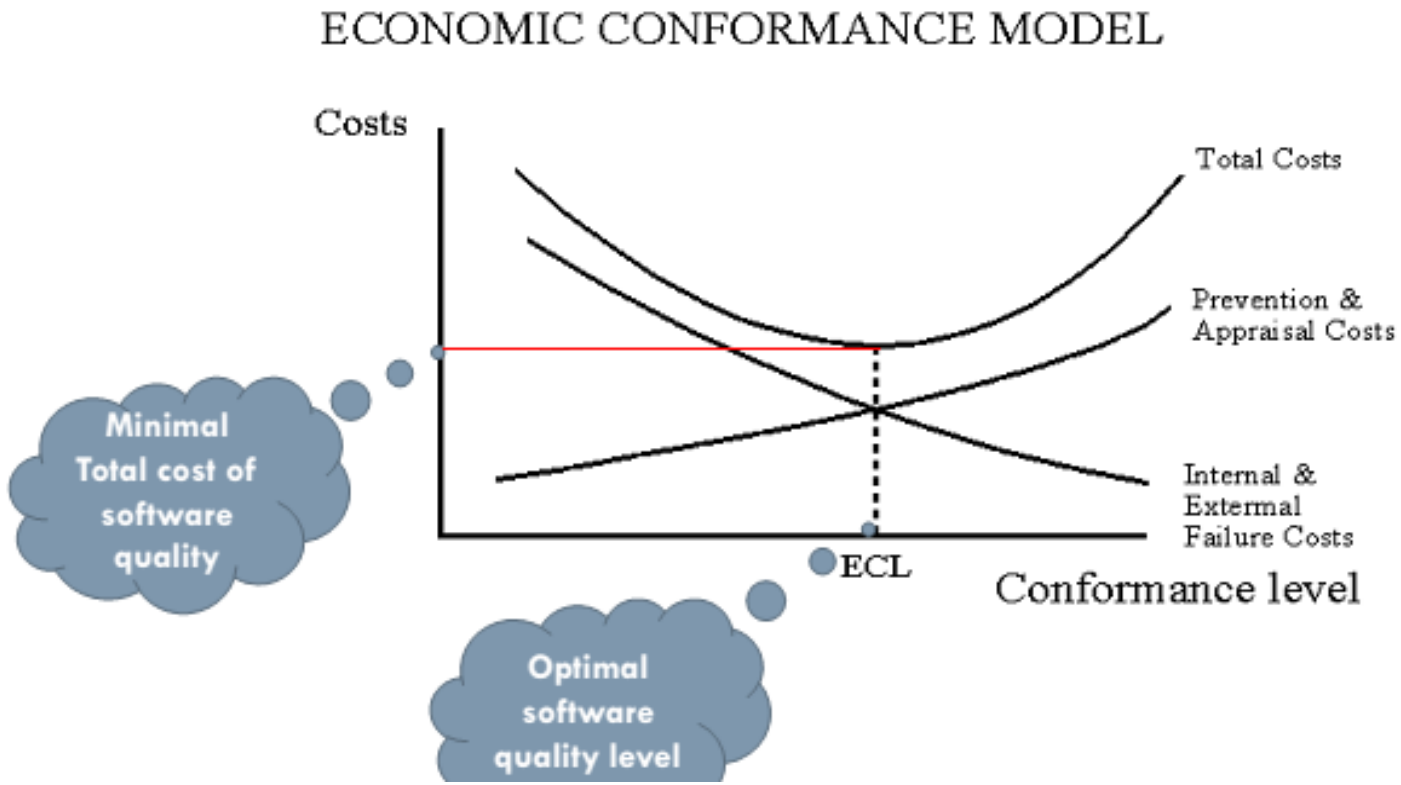
Summary of Previous Lecture

- **Quality Models**
  - **ISO 9126**
  - **McCall's**
  - **FURPS**
  - **Boehm's**
  - **GQM**
- **Cost of Quality**
  - **Prevention Cost**
  - **Appraisal Cost**
  - **Failure Cost**
- **Quality Cost Conformance Model**

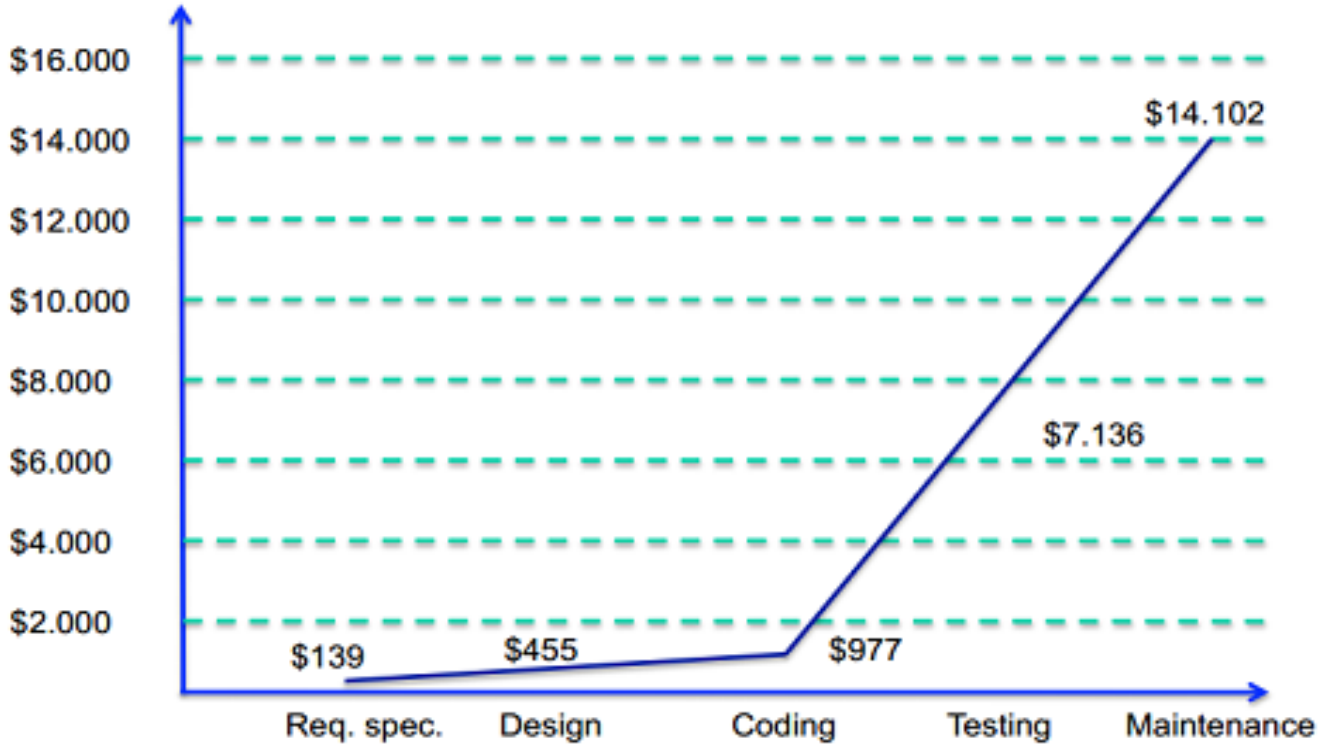
Quality Cost Conformance Model

- The quality cost conformance model provides an example of a constrained optimization approach. In this model the economic conformance level (ECL) is obtained where prevention and appraisal costs are equal to external and internal failure costs.
- Prevention and appraisal costs increase as the level of conformance of quality increases.
- Failure costs are expected to decrease as the level of conformance of quality increases. Therefore, the total costs associated with conformance of quality will be U-shaped as indicated in the figure.

The Quality Cost Conformance Model



Repair Cost of Defects



Repair costs of defects (Pressman 2010, Boehm & Basili 2001)

Topics to Cover - Today

- Quality Assurance
- Qualification Scheme for Quality Assurance
  - Defect prevention
  - Defect Reduction
  - Defect Containment

Quality Assurance

- Quality assurance can be defined as, „the establishment of framework of organizational procedures and standards that lead to a high quality software"
- Quality assurance is the function responsible for managing quality. The word "assurance" means that if the processes are followed, management can be assured of product quality.
- About quality assurance:
  - The first formal quality assurance and control function was introduced at Bell Labs in 1916 in the manufacturing world.
  - During the 1950s and 1960s, the programmers controls their product quality.
  - During the 1970s, quality assurance standards were introduced first in military contract software development.

Classification Scheme for QA as Dealing with Defects

- Defect Prevention

- Defect Reduction
  (Defect Detection and Removal)

- Defect Containment

Defect Prevention

- Prevent faults from being injected into the software through error blocking or error source removal
- Eliminating certain error sources, such as correcting human misconceptions
- We can analyze the reasons behind the missing or incorrect human actions and deal with the root causes or the error sources instead. This generic approach is called error source removal.
- The focus of these activities is typically on the people and their conceptual mistakes, which may lead to the selection and use of inappropriate development methodologies, languages, algorithms, QA strategies, etc. Such inappropriate selection may lead to numerous fault injections.

Defect Prevention Techniques

- Education and training

- Process conformance and standards enforcement

- Tools/technologies and techniques

Education and Training

- Education and training of software professionals can help them control, manage, and improve the way they work. Such activities helps to ensure that they have few if any misconceptions related to the product and the product development.

- The elimination of these human misconceptions will help prevent certain types of fault from being injected into software products.

Education and Training

The education and training effort for error source elimination should focus on the following areas:

- **Product and domain specific knowledge**
  - If the people involved are not familiar with the product type or application domain, there is a good chance that wrong solutions will be implemented
- **Knowledge about the specific development/testing tools used by the organization**
  - Also plays an important role in developing high- quality software products.
- **General Software Development Knowledge and expertise**
  - It plays an important role in developing high quality software products.
  - For example lack of expertise with requirements analysis and product specification usually leads to many problems and rework in design, coding and testing phases
- **Development Process Knowledge used by the organization**
  - For example if the people involved in incremental software development do not know how the individual development efforts for different increments fit together, the uncoordinated development may lead to many interface or interaction problems.

Tools/Technologies and Techniques

- Appropriate use of software methodologies can also help reduce the chances of fault injections. Many of the problems with low quality "fat software" could be addressed by disciplined methodologies and return to essentials for high quality "Lean software"

- Specific software tools can also help reduce the chances of fault injections. For example Syntax directed editor that automatically balances out each open parenthesis, "{", with a close parenthesis, "}", can help reduce syntactical problems in programs written in the C language

Defect Reduction

- **Defect reduction through fault detection and removal**
  - Detect and remove faults once they have been injected
  - Two categories:
    - **Static Testing**
      (Inspection of software code, design etc., Reviews, Walkthroughs, desk checking etc)
    - **Dynamic Testing**
      Test a program by executing test scripts e.g White box testing, black box testing etc.

Static Testing

Inspection

- Software Inspections are critical examinations of software artifacts (code/design/test-cases/etc.) by human inspectors aimed at discovering and fixing faults in the software system
- The basic idea of inspections are listed below:
  - Inspections are critical reading and analysis of software code or other software artifacts, such as requirements, designs, test plans, etc.
  - Inspections are typically conducted by multiple human inspectors, through some coordination process.
  - Faults are detected directly in inspection by human inspectors, either during their individual inspection or various types of group sessions.

Static Testing

- Identified faults need to be removed as a result of the inspection process, and their removal also needs to be verified.
- The inspection processes include  g and follow-up activities.

Static Testing

Walkthroughs

- The code walkthrough, like the inspection, is a set of procedures and error detection techniques for group code reading. It shares much in common with the inspection process, but the procedures are slightly different, and a different error-detection technique is employed.

Desk Checking

- A desk check can be viewed as a one-person inspection or walkthrough: A person reads a program, checks it with respect to an error list, and/or walks test data through it.

Dynamic Testing

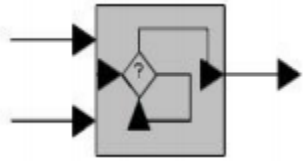
- Testing is one of the most important parts of QA and the most commonly performed QA activity

- Dynamic Testing involves the execution of software and the observation of the program behavior or outcome

Dynamic Testing

- **black-box/functional testing**
  - verifies the correct handling of the external functions provided by the software or whether the observed behavior conforms to user expectations or product specifications
  - The emphasis is on reducing the chances of encountering functional problems by target customers.

- **white/clear-box/structural testing**
  - verifies the correct implementation of internal units, structures and relations among them
  - When white box testing is performed ,failures related to internal implementations can be observed, leading to corresponding faults being detected and removed.



Defect Containment

	Stage Originated							
Stage Detected	SW Reqs Analysis	Design	Code & Test	SW Integ	SW Qual Test	System Integ	Maint & Post-Rel	**Total Detected**
SW Reqs Analysis	730							730
Design	158	481						639
Code & Test	19	2	501					522
SW Integration	15	0	12	321				348
SW Qual Test	25	4	35	63	9			136
System Integration	4	0	7	19	4	2		36
Maint & Post-Release	48	2	0	36	0	0	17	103
Total Originated	999	489	555	439	13	2	17	2,514
Total Out-of-Phase	269	8	54	118	4	0	0	453 (18%)

Defect Containment

- **Defect containment through failure prevention and containment**
  - Containing the failures to local areas
  - Limiting the damage

  - **Fault tolerance** techniques to break the causal relation between faults and failures so that local faults will not cause global failures, thus "tolerating" these local faults
  - **Failure Containment** measures to avoid catastrophic consequences, such as death, personal injury and severe things in case of failures

Defect Containment

- Software defects cannot be eliminated completely because of the large size and high complexity of software systems in use today.
- The remaining faults may be triggered under rare conditions or unusual dynamic scenarios, making it unrealistic to attempt to generate the huge number of test cases to cover all these conditions or to perform exhaustive inspection based on all possible scenarios.
- Break the causal relation between these faults and the resulting failures, thus "**tolerating**" these faults or "**contain**" the failures by reducing the resulting damage

Software Fault Tolerance

- **Fault-tolerance** or **graceful degradation** is the property that enables a system to continue operating properly in the event of the failure of some of its components. If its operating quality decreases, the decrease is proportional to the severity of the failure.
- Software fault tolerance makes the assumption that the system has unavoidable and undetectable faults and aims to make provisions for the system to operate correctly even in the presence of faults.
- Using a variety of Software methods, faults are detected and recovery is accomplished
- All fault tolerance activities are based on redundancy either for detection and recovery
- The process of fault tolerance involves many defined stages, from the phase of errors detection, through the confinement phase and evaluation of the propagation of erroneous information until the recovery phase

Software Fault Tolerance

- Error Detection
  - A starting point of fault tolerance technique, you have to implement verification tests to detect errors in the applications
  - The redundancy exists in this approach is active redundancy
  - **Active redundancy** is a design concept, in fault tolerance it is the use of redundant elements operating simultaneously to prevent, or permit recovery from failures thus increases operational availability
  - Forms of Redundancy
    - Hardware
    - Software
    - Time
    - Information

Software Fault Tolerance- Error Detection Approaches

- Replication
  - In these tests the copies of the system are used
  - Checks the outputs while submitting the same inputs
  - In case the results are distinct, a mistake is detected

- Temporal
  - These tests watch for measurement of an interval of time expected
  - Mistake is detected in case watch expires without system shows any response

Software Fault Tolerance-Recovery Types

- **Backward Recovery**
  - This recovery type undoes the executed actions and its come back to the consistent state which already has past, in a way to continue starting from that point.
  - The backward recovery process demands the establishment of a point, during the execution of the system, denominated recovery point, which is responsible for the preservation of appropriate information for subsequent recovery, in case of error.
- **Forward Recovery**
  - The forward recovery acts when an abnormal state is detected with the suspension of the processing to that more favorable conditions are found
  - This recovery type ignores part or the whole work, and it continues with what it is supposed correct, it corrects the erroneous behavior for the sending of compensation information

Software Fault Tolerance-Fault Injection for Fault Tolerance Assessment

- **Software Fault Injection** is the process of testing software under anomalous/abnormal circumstances involving erroneous/invalid external inputs
- The main objective is to test the fault tolerance capability through injecting faults into the system and analyze if the system can detect and recover from faults as specified by the system
- The results can lead to either fixing of individual software bugs or discovery of design deficiencies in system fault tolerance

Software Fault Tolerance Techniques-Data & Design Diversity

- **Multiple data representation environment:**
  - Data diverse techniques are used in a multiple data representation environment
  - Utilize different representations of input data to provide tolerance to software design faults
- **Multiple version software environment:**
  - Design diverse techniques are used in a multiple version software environment
  - Use the functionality of independently developed software versions to provide tolerance to software design faults

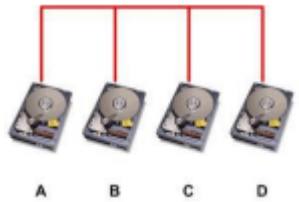
Software Fault Tolerance Techniques-Data & Design Diversity

- Two or more variants of software developed by different teams but to a common specification are used.
- These variants are then used in a time or space redundant manner to achieve fault tolerance.
- Disadvantages of design diversity is the high cost involved in developing multiple variants of software

Software Fault Tolerance Techniques-Data & Design Diversity

- Popular techniques which are based on the design diversity concept for fault tolerance in software are:

  - Recovery Block



  - N-Version Programming

  - N-Self-Checking Programming

Design Diversity technique

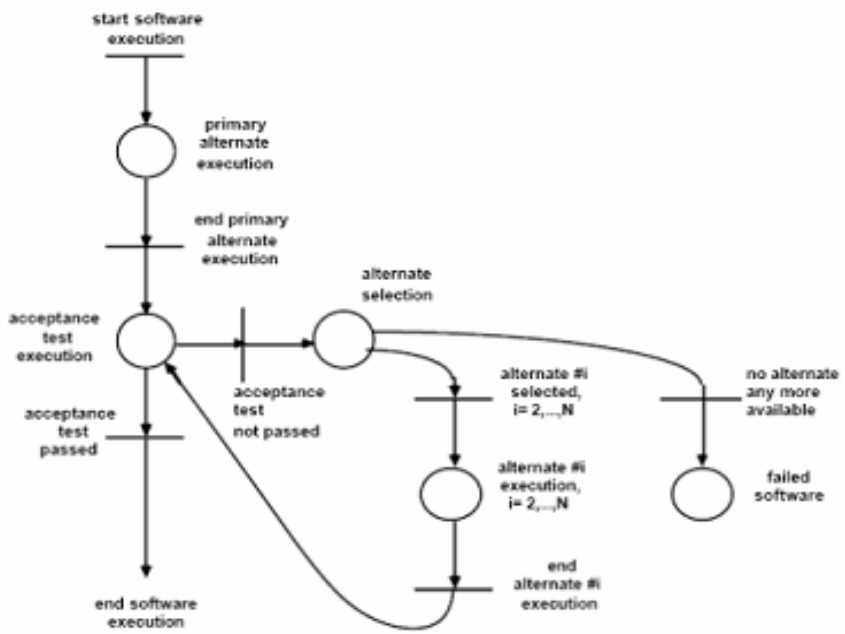
- *Recovery Blocks*

  - It was introduced in 1974 by Horning, with early implementations developed by Randell in 1975 and Hecht in 1981
  - Recovery blocks use repeated executions (or redundancy over time) as the basic mechanism for fault tolerance.
  - The basic recovery block is related to sequential systems
  - Several alternatives are proposed for a certain block of instructions, incase of mistake there will be activation of alternatives one at a time under acceptance routine until the correct execution or error message

Design Diversity technique

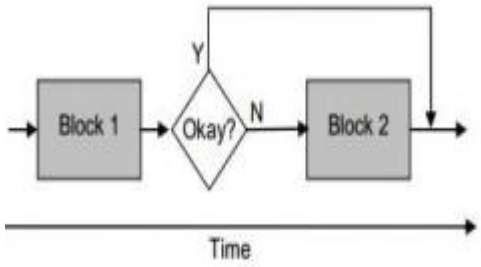
- *Recovery Blocks*
  - A recovery block consists of a conventional block which is provided with a means of error detection (an acceptance test), Primary alternate and zero or more stand-by spares (the additional alternates). A possible syntax for recovery blocks is as follows:

    - ensure acceptance test
    - by primary alternate
    - else by alternate 2
    - else by alternate n
    - else error



Design Diversity technique

- *Recovery Blocks*
  - The *primary alternate corresponds exactly to the block of the equivalent conventional program, and is* entered to perform the desired operation. The acceptance test, which is a logical expression is evaluated on exit to determine whether the alternate has performed acceptably.
  - A further alternate, if one exists, is entered if the preceding alternate fails to complete (e.g., because it attempts to divide by zero, or exceeds a time limit), or fails the acceptance test.
  - However before an alternate is so entered, the state of the process is restored to that current just before entry to the primary alternate.

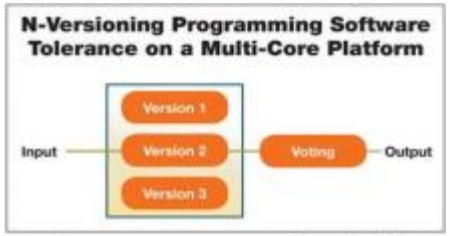



```
ensure      Successful Execution
by          executing primary block
else by     executing secondary block #1
else by     executing secondary block #2
...
else by     executing secondary block #n
else trigger exception ().
```

Two representations of Recovery Blocks

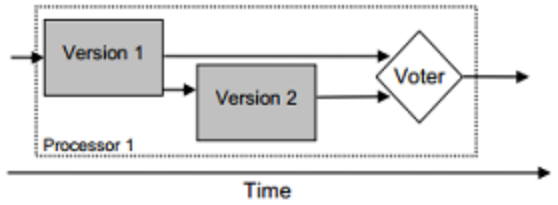
Design Diversity technique

- *NVersion Programming (NVP)*
  - NVP was suggested by Elmendorf in 1972 and developed by Avizienis and Chen in 1977–1978
  - NVP uses parallel redundancy, where N copies each of a different version (variants), of programs fulfilling the same functionality are running in parallel to fulfill the same requirement and the decision of output correctness is based on comparisons of all outputs.
  - The decision algorithm (voter) selects the correct output
  - The principle objective of N-Version Programming is to mask the effects of software flaws in the exits of the module



Design Diversity technique

- *NVersion Programming (NVP)*
  - Compared with RcB, NVP is s a static technique. That means a task: is executed by several processes or programs and a result is accepted only if it is judged as an acceptable result, usually via a majority vote.
  - Voter is the fundamental difference between NVP and RB
  - The decision algorithm in NVP makes sure that local failures in limited number of these parallel versions will not compromise global execution results
  - Since the entire versions are built to satisfy the same requirement, N version programming thus requires considerable development effort

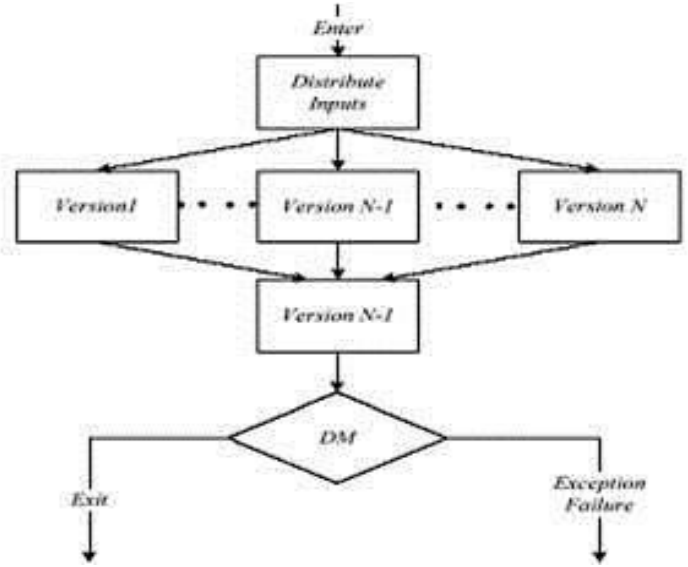


Multiple serial executions with voting

Design Diversity technique

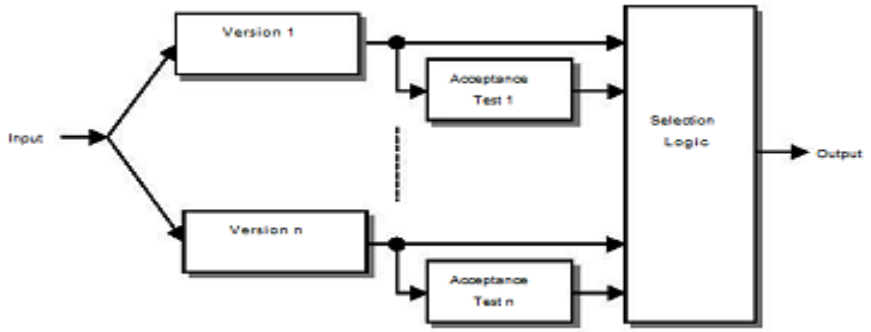
- *NVersion Programming (NVP)*

  General syntax:
  run Version 1, Version 2, ..., Version n
  if (Decision Mechanism (Result1, Result2,..., Result n))
  return Result
  else failure exception



Design Diversity technique

- *N Self Check Programming*
  - N Self Checking programming is the use of multiple software versions combined with structural variations of the Recovery Blocks and NVP.
  - N Self Checking programming using acceptance tests is shown on Figure.
  - Here the versions and the acceptance tests are developed independently from common requirements. This use of separate acceptance tests for each version is the main difference of this N Self Checking model from the Recovery Blocks approach.
  - Similar to Recovery Blocks, execution of the versions and their tests can be done sequentially or in parallel but the output is taken from the highest-ranking version that passes its acceptance test.



Data Diversity technique

- *Data diversity*, a technique for fault tolerance in software, was introduced by Amman and Knight.
- While the design diversity approaches to provide fault tolerance, rely on multiple versions of the software written to the same specifications, the data diversity approach uses only one version of the software.
- This approach relies on the observation that a software sometime fails for certain values in the input space and this failure could be averted if there is a minor change of input data which is acceptable to the software.
- This technique is cheaper to implement than the design diversity technique.
- Popular techniques which are based on the data diversity concept for fault tolerance in software are:
  - Retry Blocks
  - N-Copy Programming

Data Diversity technique

- Retry Blocks
  - A retry block developed by Ammann and Knight [Ammann and Knight 1987; Ammann and Knight 1988] is a modification of the recovery block scheme that uses data diversity instead of design diversity.
  - Data diversity is a strategy that does not change the algorithm of the system (just retry), but does change the data that the algorithm processes. It is assumed that there are certain data which will cause the algorithm to fail, and that if the data were re-expressed in a different, equivalent (or near equivalent) form the algorithm would function correctly.
  - A retry block executes the single algorithm normally and evaluates the acceptance test.
  - If the test passes, the retry block is complete. If the test fails, the algorithm executes again after the data has been re-expressed.

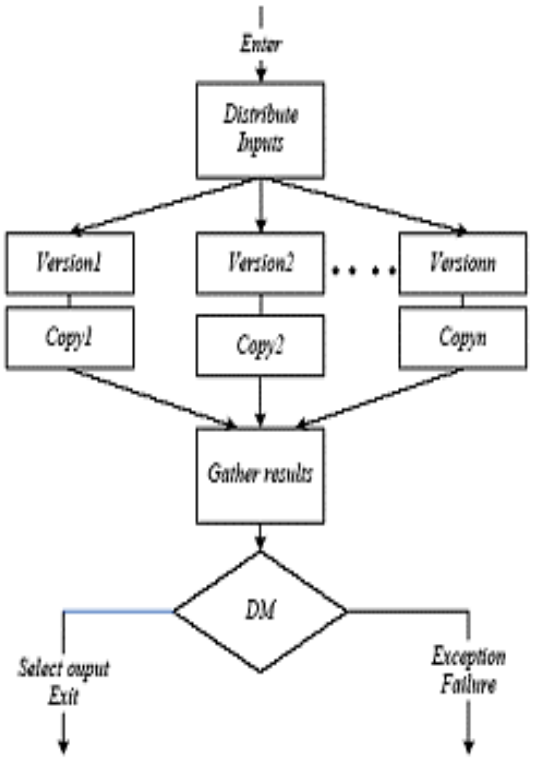
Data Diversity technique

```
General syntax:
ensure Acceptance Test
by Primary Algorithm (Original Input)
else by Primary Algorithm (Re-expressed Input)
else by Primary Algorithm (Re-expressed Input)
...
... [Deadline Expires]
else by Backup Algorithm (Original Input)
else failure exception
```

- The RtB syntax above states that the technique will first attempt to ensure the AT by using the primary algorithm.
- If the primary algorithm's result does not pass the AT, then the input data will be re expressed and the same algorithm attempted until a result passes the AT or the deadline expires.
- If the deadline expire, the backup algorithm is invoked with the original inputs.
- If this backup algorithm is not successful, an error occurs.

Data Diversity technique

- N Copy Programming
  - N Copy Programming (NCP) is the data diverse variant of NVP
  - The Technique use one or more Data re-expression algorithm's (DRA's) and at least two copies of a program
  - The system inputs are run through the DRA(s) to re-express the inputs.
  - The copies execute in parallel using the re-expressed data as input.
  - A DM examines the results of the copy executions and selects the "best" result, if one exists.



Data Diversity technique

The basic NCP technique consists of an executive, 1 to n DRA, n copies of the program or function,
and a DM. The executive orchestrates the NCP technique operation, which has the general syntax:
run DRA 1, DRA 2, ..., DRA n
run Copy 1(result of DRA 1),
Copy 2(result of DRA 2), ...,
Copy n(result of DRA n)
if (Decision Mechanism (Result 1, Result 2, ...,Result n))
return Result
else failure exception

The NCP syntax above states that the technique first runs the DRA concurrently to re-express the input data, then executes the n copies concurrently.
The results of the copy executions are provided to the DM, which operates upon the results to determine if a correct result can be adjudicated/judged/decided/resolved.
If one can (i.e., the Decision Mechanism statement above evaluates to TRUE), then it is returned.
If a correct result cannot be determined, then an error occurs.

Decision Mechanism

- Voters
  - Voters compare the results from two or more variants
  - If there are two results to examine, the decision mechanism is called the comparator
  - The Voter decides the correct results if any exists
- Acceptance Tests
  - Verification of the last instance
  - Not important to know which alternative generates the results but necessary that these are inside the acceptance limit
  - Must be Simpler
  - If not projected conveniently then two flaws occurs
    - Disregard of correct alternative
    - Acceptance of incorrect alternative

Failure Containment

- With all the fault prevention and fault tolerant techniques, unfortunately, we will still have faults. In that case, can we

  (a) "prevent accidents" ? and can we

  (b) "reduce the damage of the accidents"?

- We already know that we can not prevent all accidents. But we can analyze the hazards of an accident and hopefully contain or limit the damage.

Software Fault Tolerance Techniques-Data & Design Diversity

- **Multiple data representation environment:**
  - Data diverse techniques are used in a multiple data representation environment
  - Utilize different representations of input data to provide tolerance to software design faults
- **Multiple version software environment:**
  - Design diverse techniques are used in a multiple version software environment
  - Use the functionality of independently developed software versions to provide tolerance to software design faults

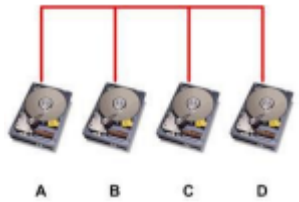
Software Fault Tolerance Techniques-Data & Design Diversity

- Two or more variants of software developed by different teams but to a common specification are used.
- These variants are then used in a time or space redundant manner to achieve fault tolerance.
- Disadvantages of design diversity is the high cost involved in developing multiple variants of software

Software Fault Tolerance Techniques-Data & Design Diversity

- Popular techniques which are based on the design diversity concept for fault tolerance in software are:

  - Recovery Block



  - N-Version Programming

  - N-Self-Checking Programming

Design Diversity technique

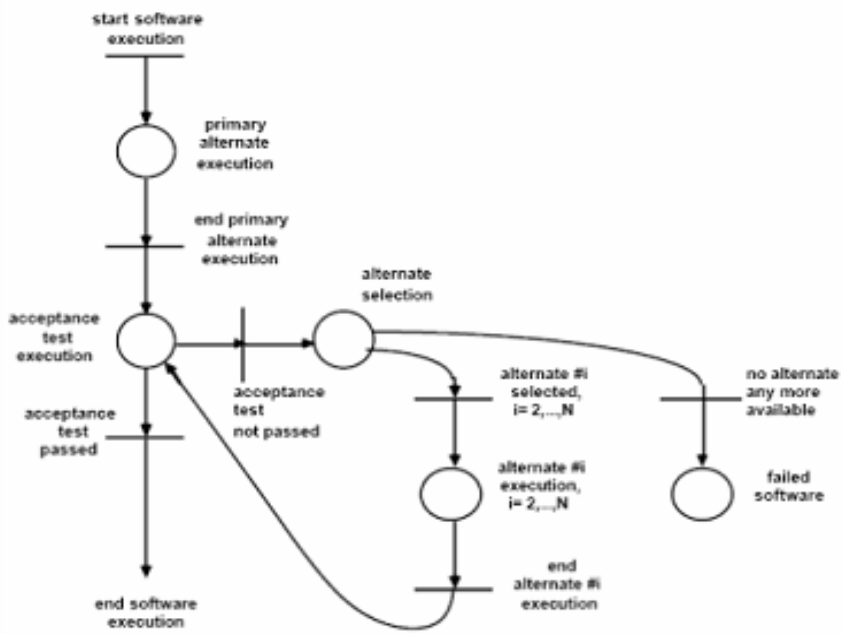
- *Recovery Blocks*

  - It was introduced in 1974 by Horning, with early implementations developed by Randell in 1975 and Hecht in 1981
  - Recovery blocks use repeated executions (or redundancy over time) as the basic mechanism for fault tolerance.
  - The basic recovery block is related to sequential systems
  - Several alternatives are proposed for a certain block of instructions, incase of mistake there will be activation of alternatives one at a time under acceptance routine until the correct execution or error message

Design Diversity technique

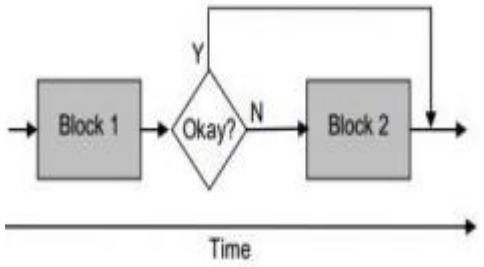
- *Recovery Blocks*
  - A recovery block consists of a conventional block which is provided with a means of error detection (an acceptance test), Primary alternate and zero or more stand-by spares (the additional alternates). A possible syntax for recovery blocks is as follows:

    - ensure acceptance test
    - by primary alternate
    - else by alternate 2
    - else by alternate n
    - else error



Design Diversity technique

- *Recovery Blocks*
  - The *primary alternate corresponds exactly to the block of the equivalent conventional program, and is* entered to perform the desired operation. The acceptance test, which is a logical expression is evaluated on exit to determine whether the alternate has performed acceptably.
  - A further alternate, if one exists, is entered if the preceding alternate fails to complete (e.g., because it attempts to divide by zero, or exceeds a time limit), or fails the acceptance test.
  - However before an alternate is so entered, the state of the process is restored to that current just before entry to the primary alternate.

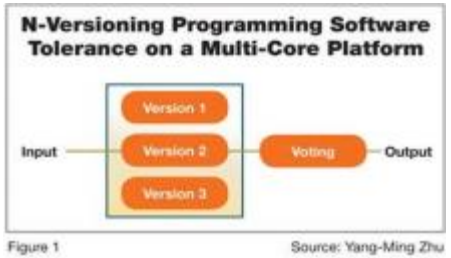



```
ensure      Successful Execution
by          executing primary block
else by     executing secondary block #1
else by     executing secondary block #2
...
else by     executing secondary block #n
else trigger exception ().
```

Two representations of Recovery Blocks

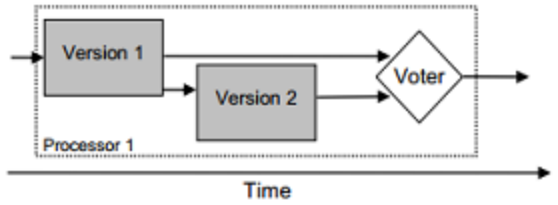
Design Diversity technique

- *NVersion Programming (NVP)*
  - NVP was suggested by Elmendorf in 1972 and developed by Avizienis and Chen in 1977–1978
  - NVP uses parallel redundancy, where N copies each of a different version (variants), of programs fulfilling the same functionality are running in parallel to fulfill the same requirement and the decision of output correctness is based on comparisons of all outputs.
  - The decision algorithm (voter) selects the correct output
  - The principle objective of N-Version Programming is to mask the effects of software flaws in the exits of the module



Design Diversity technique

- *NVersion Programming (NVP)*
  - Compared with RcB, NVP is s a static technique. That means a task: is executed by several processes or programs and a result is accepted only if it is judged as an acceptable result, usually via a majority vote.
  - Voter is the fundamental difference between NVP and RB
  - The decision algorithm in NVP makes sure that local failures in limited number of these parallel versions will not compromise global execution results
  - Since the entire versions are built to satisfy the same requirement, N version programming thus requires considerable development effort

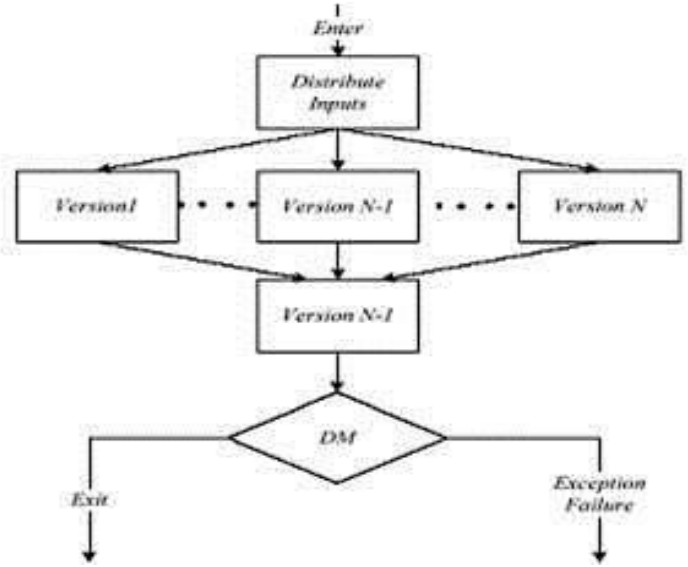


Multiple serial executions with voting

Design Diversity technique

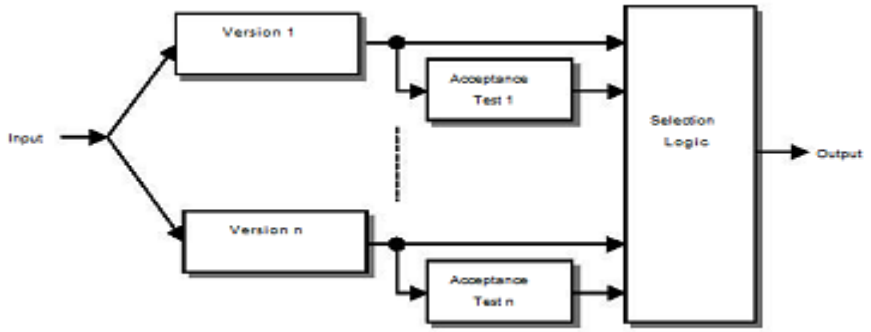
- *NVersion Programming (NVP)*

  General syntax:
  run Version 1, Version 2, ..., Version n
  if (Decision Mechanism (Result1, Result2,..., Result n))
  return Result
  else failure exception



Design Diversity technique

- *N Self Check Programming*
  - N Self Checking programming is the use of multiple software versions combined with structural variations of the Recovery Blocks and NVP.
  - N Self Checking programming using acceptance tests is shown on Figure.
  - Here the versions and the acceptance tests are developed independently from common requirements. This use of separate acceptance tests for each version is the main difference of this N Self Checking model from the Recovery Blocks approach.
  - Similar to Recovery Blocks, execution of the versions and their tests can be done sequentially or in parallel but the output is taken from the highest-ranking version that passes its acceptance test.



Data Diversity technique

- *Data diversity*, a technique for fault tolerance in software, was introduced by Amman and Knight.
- While the design diversity approaches to provide fault tolerance, rely on multiple versions of the software written to the same specifications, the data diversity approach uses only one version of the software.
- This approach relies on the observation that a software sometime fails for certain values in the input space and this failure could be averted if there is a minor change of input data which is acceptable to the software.
- This technique is cheaper to implement than the design diversity technique.
- Popular techniques which are based on the data diversity concept for fault tolerance in software are:
  - Retry Blocks
  - N-Copy Programming

Data Diversity technique

- Retry Blocks
  - A retry block developed by Ammann and Knight [Ammann and Knight 1987; Ammann and Knight 1988] is a modification of the recovery block scheme that uses data diversity instead of design diversity.
  - Data diversity is a strategy that does not change the algorithm of the system (just retry), but does change the data that the algorithm processes. It is assumed that there are certain data which will cause the algorithm to fail, and that if the data were re-expressed in a different, equivalent (or near equivalent) form the algorithm would function correctly.
  - A retry block executes the single algorithm normally and evaluates the acceptance test.
  - If the test passes, the retry block is complete. If the test fails, the algorithm executes again after the data has been re-expressed.

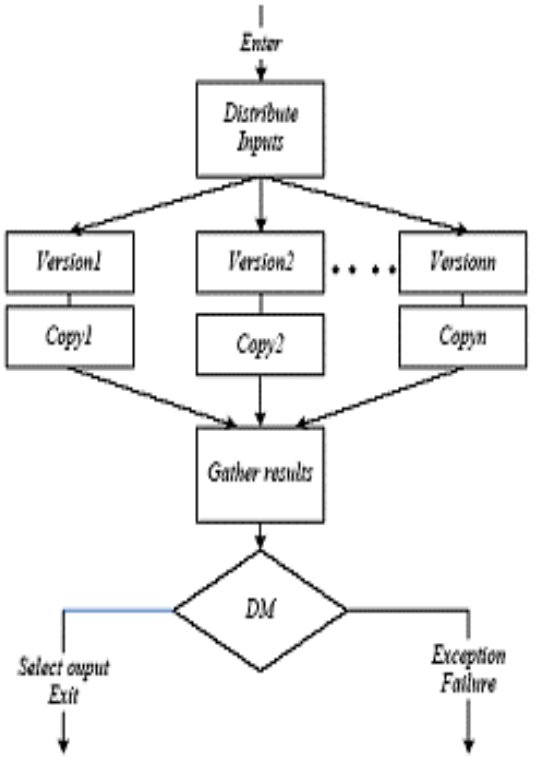
Data Diversity technique

```
General syntax:
ensure Acceptance Test
by Primary Algorithm (Original Input)
else by Primary Algorithm (Re-expressed Input)
else by Primary Algorithm (Re-expressed Input)
...
... [Deadline Expires]
else by Backup Algorithm (Original Input)
else failure exception
```

- The RtB syntax above states that the technique will first attempt to ensure the AT by using the primary algorithm.
- If the primary algorithm's result does not pass the AT, then the input data will be re expressed and the same algorithm attempted until a result passes the AT or the deadline expires.
- If the deadline expire, the backup algorithm is invoked with the original inputs.
- If this backup algorithm is not successful, an error occurs.

Data Diversity technique

- N Copy Programming
  - N Copy Programming (NCP) is the data diverse variant of NVP
  - The Technique use one or more Data re-expression algorithm's (DRA's) and at least two copies of a program
  - The system inputs are run through the DRA(s) to re-express the inputs.
  - The copies execute in parallel using the re-expressed data as input.
  - A DM examines the results of the copy executions and selects the "best" result, if one exists.



Data Diversity technique

The basic NCP technique consists of an executive, 1 to n DRA, n copies of the program or function,
and a DM. The executive orchestrates the NCP technique operation, which has the general syntax:
run DRA 1, DRA 2, ..., DRA n
run Copy 1(result of DRA 1),
Copy 2(result of DRA 2), ...,
Copy n(result of DRA n)
if (Decision Mechanism (Result 1, Result 2, ...,Result n))
return Result
else failure exception

The NCP syntax above states that the technique first runs the DRA concurrently to re-express the input data, then executes the n copies concurrently.
The results of the copy executions are provided to the DM, which operates upon the results to determine if a correct result can be adjudicated/judged/decided/resolved.
If one can (i.e., the Decision Mechanism statement above evaluates to TRUE), then it is returned.
If a correct result cannot be determined, then an error occurs.

Decision Mechanism

- Voters
  - Voters compare the results from two or more variants
  - If there are two results to examine, the decision mechanism is called the comparator
  - The Voter decides the correct results if any exists
- Acceptance Tests
  - Verification of the last instance
  - Not important to know which alternative generates the results but necessary that these are inside the acceptance limit
  - Must be Simpler
  - If not projected conveniently then two flaws occurs
    - Disregard of correct alternative
    - Acceptance of incorrect alternative

Failure Containment

- With all the fault prevention and fault tolerant techniques, unfortunately, we will still have faults. In that case, can we

  (a) "prevent accidents" ? and can we

  (b) "reduce the damage of the accidents"?

- We already know that we can not prevent all accidents. But we can analyze the hazards of an accident and hopefully contain or limit the damage.

Fault Tree Analysis

- Fault Tree Analysis (FTA) was originally developed in 1962 at Bell Laboratories by H.A. Watson, under a U.S. Air Force Ballistics Systems Division contract to evaluate the Minuteman I Intercontinental Ballistic Missile (ICBM) Launch Control System

- Fault tree analysis (FTA) is a top down, deductive failure analysis in which an undesired state of a system is analyzed using Boolean logic to combine a series of lower-level events. This analysis method is mainly used in the field of safety engineering and Reliability engineering to determine the probability of a safety accident or a particular system level (functional) failure.

The Fault Tree

- Begin Fault Analysis by identifying possible failures in design operation or maintenance

- Next build a graph whose nodes are failures
  - Single contents
  - System function
  - Entire system

- Edge = relationship among nodes by logical descriptor (AND,OR)

The Fault Tree

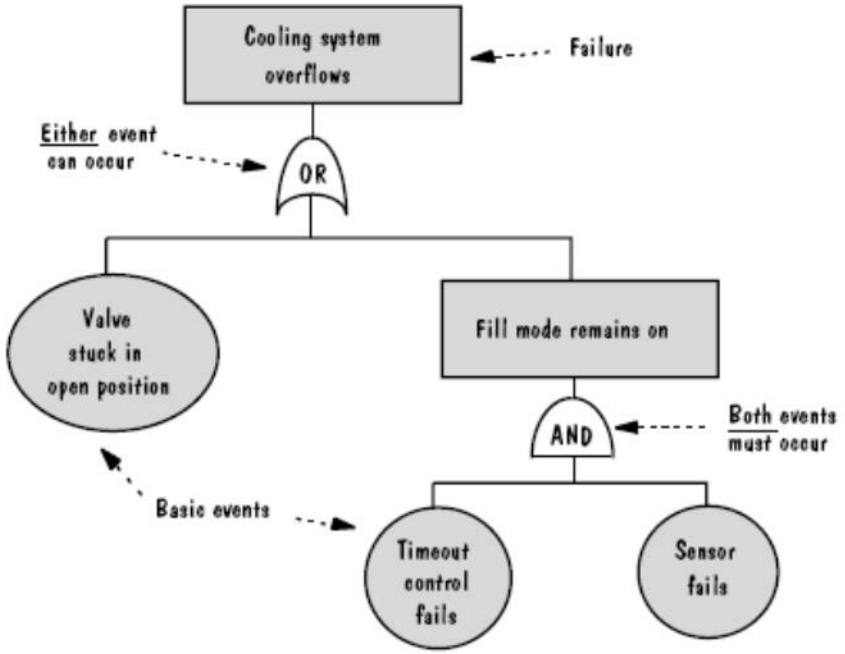


Figure 5.19 Portion of power plant control system

The Fault Tree

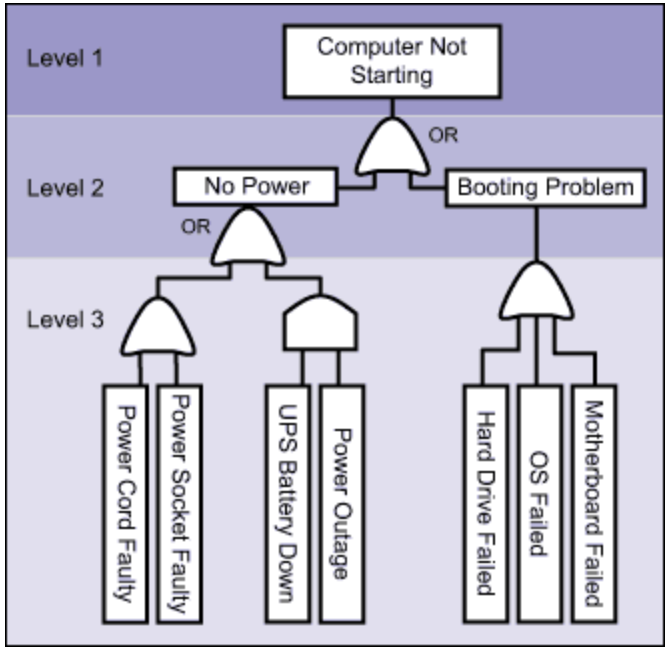
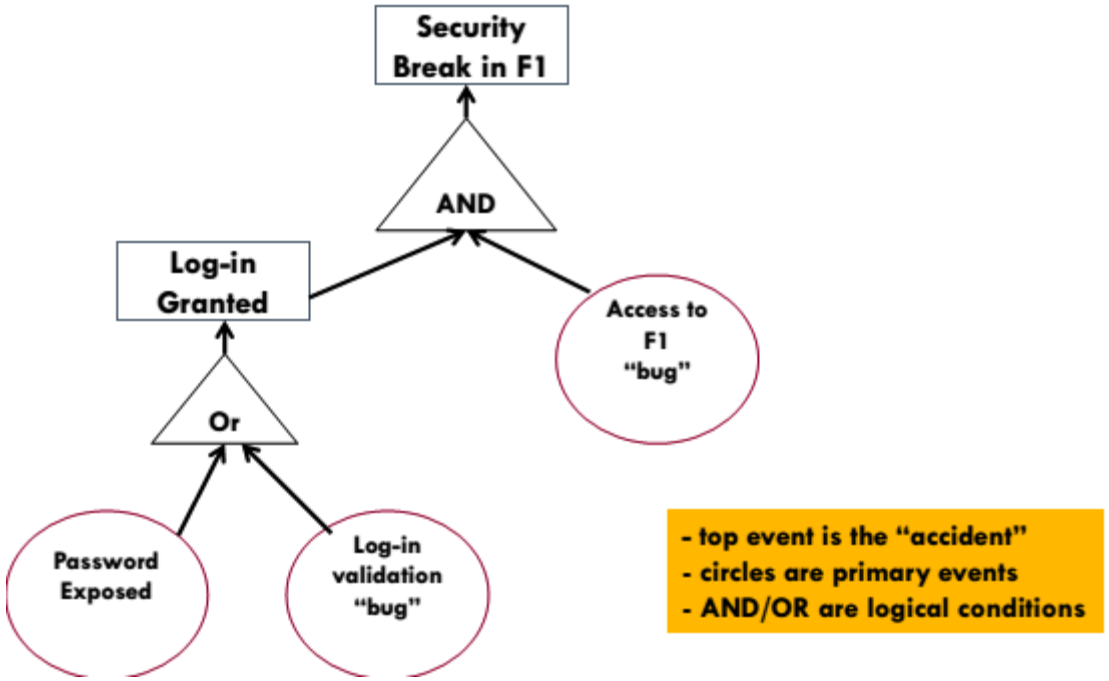


Figure: Demonstration of FTA

Fault Tree Analysis

- list the set of events that cause the "accident" or "failure"
- build an upside down tree that logically connects the events to the failure



Fault tree Analysis

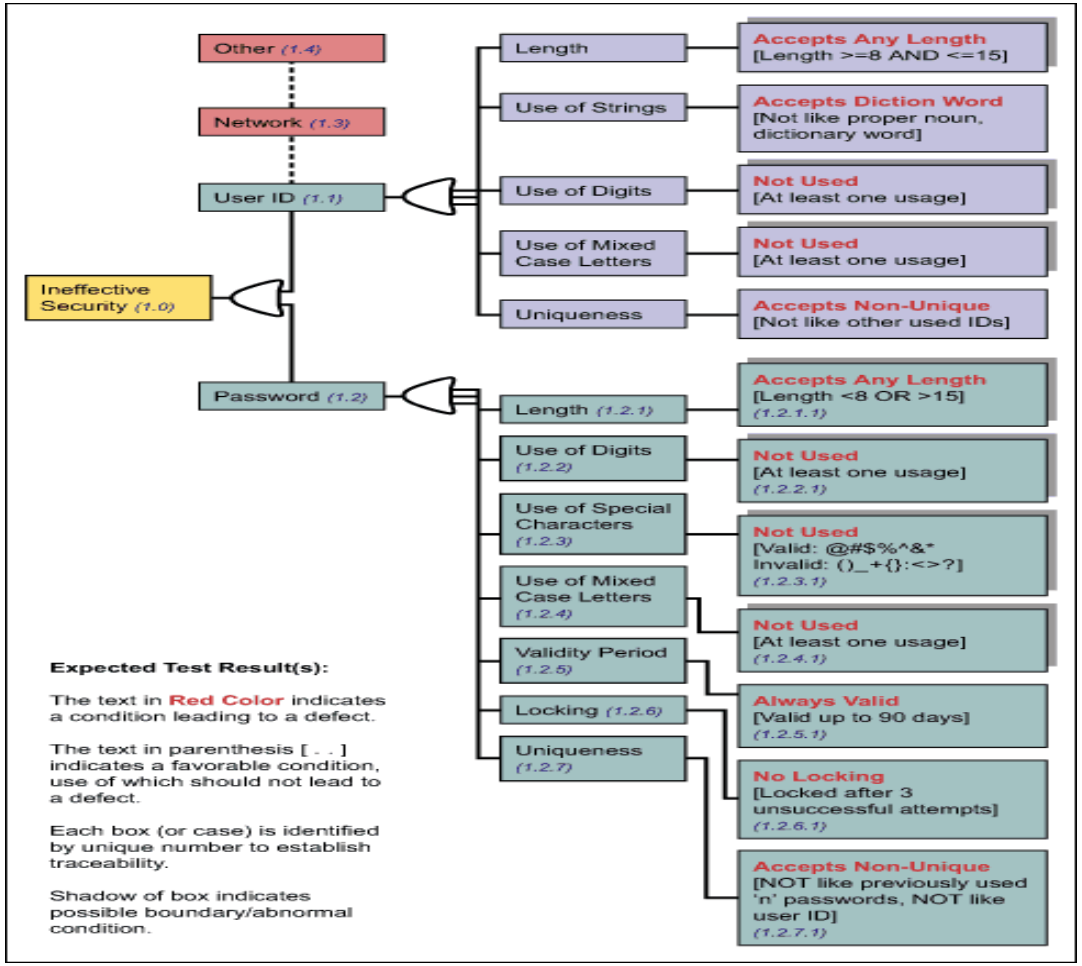


Figure: Illustration of FTA security of software application by using controlled access

Containment

- We use the fault-tree to analyze and understand the cause of the "accident."We may use it for:
  - ✓ Accident elimination
  - ✓ Accident reduction
  - ✓ Accident control

References

1. http://www.sqa.net/index.htm
2. Software Engineering by Roger Pressman
3. Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement: JEFF TIAN, 2005.
4. The Art of Software Testing by Glenford Myers
5. http://en.wikipedia.org/wiki/Black-box_testing
6. http://en.wikipedia.org/wiki/White-box_testing
7. http://www.cs.rutgers.edu/~rmartin/teaching/spring03/cs553/papers01/06.pdf
8. http://www.hillside.net/plop/2009/papers/Process/N-Version%20Programming.pdf
9. http://en.wikipedia.org/wiki/Active_redundancy
10. http://en.wikipedia.org/wiki/Fault-tolerant_system
11. https://www.google.com.pk/search?hl=en&noj=1&q=active+redundancy&tbs=dfn:1&tbo=u&sa=X&ei=s1BhUMmnAaOn4gTpulGABw&ved=0CB0QkQ4&biw=1366&bih=624
12. http://en.wikipedia.org/wiki/Fault_tree_analysis
13. http://www.pld.ttu.ee/IAF0030/Paper_4.pdf
14. http://www.isixsigma.com/tools-templates/risk-management/using-fault-tree-analysis-improve-software-testing/