**CLASSIFICATION OF QA ACTIVITIES:**

Based on how different QA alternatives deal with defects can yield a generic classification scheme that can be used to help us better select, adapt and use different QA alternatives and related techniques for specific applications

**Classification Scheme:** We can view different QA activities as attempting to prevent, eliminate, reduce, or contain various specific problems associated with different aspects of defects. We can classify these QA alternatives into the following three generic categories:

1. Defect prevention through error blocking or error source removal
2. Defect reduction through fault detection and removal
3. Defect containment through failure prevention and containment

**Defect prevention through error blocking or error source removal:**

These QA activities prevent certain types of faults from being injected into the software. Since errors are the missing or incorrect human actions that lead to the injection of faults into software systems, we can directly correct or block these actions, or remove the underlying causes for them.

Therefore, defect prevention can be done in two generic ways:

* Eliminating certain error sources, such as eliminating ambiguities or correcting human misconceptions, which are the root causes for the errors.
* Fault prevention or blocking by directly correcting or blocking these missing or incorrect human actions.

**Defect reduction through fault detection and removal:**

These QA alternatives detect and remove certain faults once they have been injected into the software systems. In fact, most traditional QA activities fall into this category. For example,

* Inspection directly detects and removes faults from the software code, design, etc.
* Testing removes faults based on related failure observations during program etc. execution.

**Defect containment through failure prevention and containment:**

These containment measures focus on the failures by either containing them to local areas so that there is no global failures observable to users, or limiting the damage caused by software system failures.

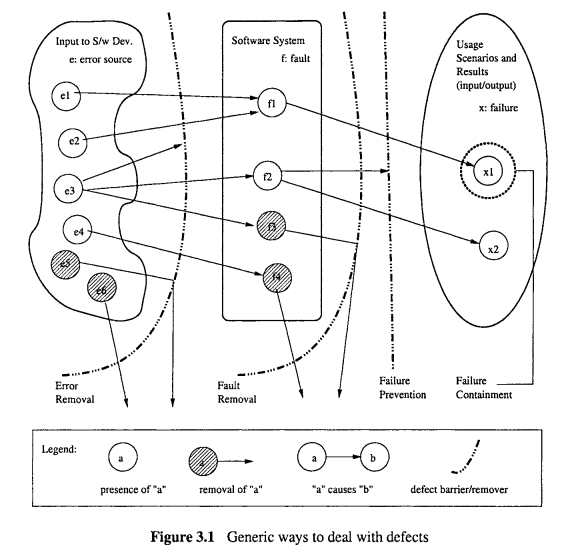
Therefore, defect containment can be done in two generic ways:

* Using fault-tolerance techniques, break the causal relation between faults and failures so that local faults will not cause global failures
* An extension to fault-tolerance is containment measures to avoid catastrophic consequences like damage control in case of failures. For example, failure containment for real-time control software used in nuclear reactors may include concrete walls to encircle and contain radioactive material in case of reactor melt-down due to software failures, in order to prevent damage to environment and people’s health.

**Dormant defects:** The faults left in the finished software products are often called “dormant defects”, which may stay dormant for some time, but have the potential of causing problems to customers and users of the products.

**Pre/Post-release defect prevention and reduction activities:**

Pre-release defect prevention methods are implemented to reduce the number of defects in the software release. After product release, the failures observed and problems reported by customers and users also need to be fixed which is called post release defect reduction, which could lead to reduced defects and improved product quality. But the cost of fixing defects after product release is significantly higher than before product release due to the numerous installations. In addition, the damage to software vendors’ reputation can be devastating.



**DEFECT CONTAINMENT:**

For software systems where failure impact is substantial, such as many real-time control software sub-systems used in medical, nuclear, transportation, and other embedded systems, this low defect level and failure risk may still be inadequate. These 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. Instead, some other means need to be used to prevent failures by breaking the causal relations between these faults and the resulting failures, thus “tolerating” these faults, or to contain the failures by reducing the resulting damage.

1. Software fault tolerance
2. Safety assurance and failure containment

**Software Fault Tolerance:** Software fault tolerance ideas originate from fault tolerance designs in traditional hardware systems that require higher levels of reliability, availability, or dependability. The primary software fault tolerance techniques include recovery blocks, N-version programming (NVP), and their variations.

* **Recovery Blocks:** uses repeated executions as the basic mechanism for fault tolerance. If dynamic failures in some local areas are detected, a portion of the latest execution is repeated, in the hope that this repeated execution will not lead to the same failure. Therefore, local failures will not propagate to global failures, although some time-delay may be involved.
* **N-Version Programming:** NVP uses parallel redundancy, where N copies, each of a different version, of programs fulfilling the same functionality are running in parallel. The decision algorithm in NVP makes sure that local failures in limited number of these parallel versions will not compromise global execution results.

**Note:** most fault tolerance techniques, faults are not typically identified, therefore not removed, but only tolerated dynamically

**Safety assurance and failure containment**

For safety critical systems, the primary concern is our ability to prevent accidents from happening, where an accident is a failure with a severe consequence. So, various techniques are needed for safety critical systems based on analysis of hazards, or logical pre-conditions for accidents.

1. **Hazard elimination:** These techniques reduce certain defect injections or substitute non-hazardous ones for hazardous ones. This can be achieved through substitution, simplification, decoupling, elimination of specific human errors, and reduction of hazardous materials or conditions.
2. **Hazard reduction:** These techniques are similar to the fault tolerance techniques considering hazardous situations where local failures are contained without leading to system failures. This can be achieved through design for controllability, use of locking devices, and failure minimization using safety margins and redundancy.
3. **Hazard control:** These techniques reduce the severity of failures, therefore weakening the link between failures and accidents through reducing exposure, isolation and containment, protection systems, and fail-safe design.
4. **Damage control:** These techniques reduce the severity of accidents, thus limiting the damage caused by these accidents and related software failures through escape routes, safe abandonment of products and materials, and devices for limiting physical damages to equipment or people.