| **Testing** | **Debugging** |
| --- | --- |
| [Testing](https://www.geeksforgeeks.org/software-testing-basics/)is the process to find bugs and errors. | [Debugging](https://www.geeksforgeeks.org/software-engineering-debugging/) is the process of correcting the bugs found during testing. |
| It is the process to identify the failure of implemented code. | It is the process to give absolution to code failure. |
| Testing is the display of errors. | Debugging is a deductive process. |
| Testing is done by the tester. | Debugging is done by either programmer or the developer. |
| There is no need of design knowledge in the testing process. | Debugging can’t be done without proper design knowledge. |
| Testing can be done by insiders as well as outsiders. | Debugging is done only by insiders. An outsider can’t do debugging. |
| Testing can be manual or automated. | Debugging is always manual. Debugging can’t be automated. |
| It is based on different testing levels i.e. unit testing, integration testing, system testing, etc. | Debugging is based on different types of bugs. |
| Testing is a stage of the software development life cycle (SDLC). | Debugging is not an aspect of the software development life cycle, it occurs as a consequence of testing. |
| Testing is composed of the validation and verification of software. | While debugging process seeks to match symptoms with cause, by that it leads to error correction. |
| Testing is initiated after the code is written. | Debugging commences with the execution of a test case. |
| Testing process based on various levels of testing-system testing, integration testing, unit testing, etc. | Debugging process based on various types of bugs is present in a system. |

* [Software Testing Models](https://www.edureka.co/blog/software-testing-models/#softwaretestingmodels)
  + [Waterfall Model](https://www.edureka.co/blog/software-testing-models/#waterfallmodel)
  + [V Model](https://www.edureka.co/blog/software-testing-models/#vmodel)
  + [Agile Model](https://www.edureka.co/blog/software-testing-models/#agilemodel)
  + [Spiral Model](https://www.edureka.co/blog/software-testing-models/#spiralmodel)
  + [Iterative Model](https://www.edureka.co/blog/software-testing-models/#iterativemodel)

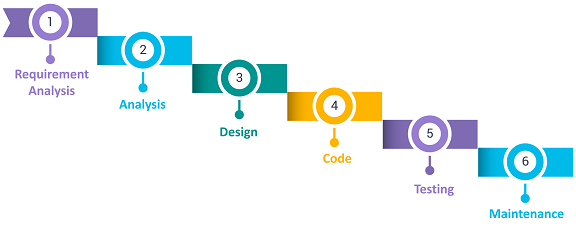
**Software Testing Models**

[Software Testing](https://www.edureka.co/blog/what-is-software-testing/" \t "_blank) is an integral part of the [software development life cycle](https://www.edureka.co/blog/software-testing-tutorial/). There are different models or approaches you can use in the software development process where each model has its own advantages and disadvantages. So, You must choose a particular model depending on the project deliverables and complexity of the project.

The different Software Testing Models are:

**Waterfall Model**

This is the most basic [software development life cycle](https://www.edureka.co/blog/software-testing-life-cycle/) process which is followed broadly in the industry. In this model, the developers follow a sequence of processes downwards towards the ultimate goal. It is like a waterfall where there are various phases involved.

The different phases in the [**waterfall model**](https://www.edureka.co/blog/waterfall-vs-agile/)are:

* Requirement Analysis
* Analysis phase
* Software design
* Programmed implementation
* Testing
* Maintenance

**Advantages**

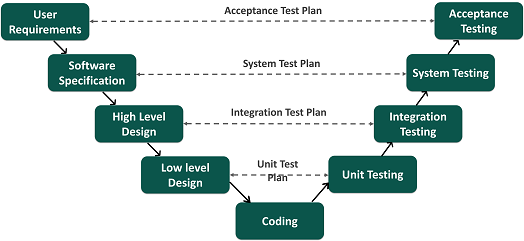
* It is easy to implement and maintain.
* The initial phase of rigorous scrutiny of requirements and systems helps in saving time later in the developmental phase.
* The requirement of resources is minimal and testing is done after each
* phase has been completed.

**Disadvantages**

* It is not possible to alter or update requirements.
* Once you move into the next phase you cannot make changes.
* You cannot start the next phase until the previous phase is completed.

**V Model**

The V Model is considered superior to the waterfall model. In this model, the development and test execution activities are carried out side by side in the downhill and uphill shape. Also, testing starts at the unit level and spreads towards the integration of the entire system.



**Advantages**

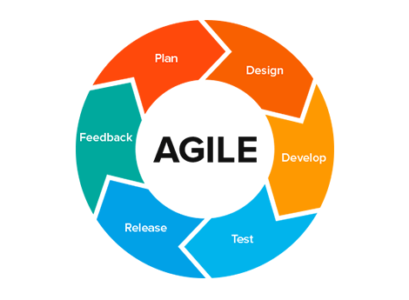
* It is easy to use since testing activities like planning and test designing are done before coding.
* This model enhances the chances of success and saves time.
* Defects are mostly found at an early stage and downward flow of defects is generally avoided.

**Disadvantages**

* It is a rigid model.
* The software is developed during the implementation phase so early prototypes of the product are not available.
* If there are changes in the midway, you need to update the test document.

### ****Agile Model****

In the Agile model, requirements and solutions evolve through collaboration between various cross-functional teams. It is also known as an iterative and incremental model. The agile software testing model focus on process adaptability and customer satisfaction by rapid delivery of working software product and by breaking the product into small incremental builds.



**Advantages**

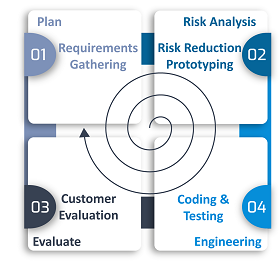
* It ensures customer satisfaction with rapid and continuous development of deliverables.
* The continuous interaction between the customers, developers, and testers makes it a flexible model.
* You can develop the working software quickly and adapt to changing requirements regularly.

**Disadvantages**

* It is difficult to assess the effort required at the beginning of the cycle for large and complex software development cases.
* Due to continuous interaction with the customer, the project can go off track if the customer is not clear about the goals.

**Spiral Model**

This software testing model is similar to the [Agile model](https://www.edureka.co/blog/what-is-agile-testing/), but with more emphasis on risk analysis. The different phases of the spiral model include planning, risk analysis, engineering, and evaluation. In this case, you need to gather the requirements and perform the risk assessment at the base level and every upper spiral builds on it.



**Advantages**

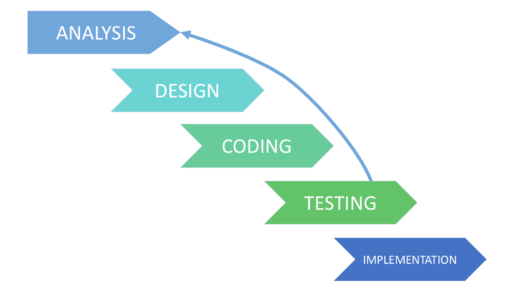
* It is suitable for complex and large systems.
* You can add functionalities depending on the changed circumstances.
* Software is produced early in the cycle.

**Disadvantages**

* It is a costly model which requires highly specialized expertise in risk analysis
* It does not work well on simpler projects.

**Iterative Model**

The Iterative model does not need a full list of requirements before beginning the project. The development process starts with the requirements of the functional part, which can be expanded later. The process is repetitive and allows new versions of the product for every cycle. Every iteration includes the development of a separate component of the system which is added to the functional developed earlier.



**Advantages**

* It is easier to control the risks as high-risk tasks are completed first.
* The progress is easily measurable.
* Problems and risks defined within one iteration can be prevented in the next sprints.

**Disadvantages**

* Iterative model requires more resources than the waterfall model.
* The process is difficult to manage.
* The risks may not be completely determined even at the final stage of the project.

These are the different software testing models involved in the software development life cycle. I hope you understood how each of these models is used in software testing.

Different Types of Bugs in Software Testing:

No matter the software type, software bugs are categorized into three types; Nature, Priority, and Severity. Classification of bugs in software testing is done on the basis of their nature and impact on the user experience.

Software Bugs by Nature:

Software bugs have different natures where they affect the overall functioning of the software differently. Though there are dozens of such bugs existing currently, you may not face them frequently. With that in mind, here are the most common software bugs categorized by nature that you are most likely to witness at some point in your software development career.

Performance Bugs:  
No user wants to use software with poor performance. Software bugs that lead to degraded speed, stability, increased response time, and higher resource consumption are considered performance bugs. The most significant sign of any such bug in software is by noticing slower loading speed than usual or analyzing the response time. If any such sign is found, the developer may begin diagnosing a performance bug. The [performance testing](https://www.thinksys.com/performance-testing-services/) phase is part of the development process where every such bug is detected in the software.

Security Bugs:  
While using software, security is the biggest concern of a user. Software with poor security will not only put the user’s data at risk but will also damage the overall image of the organization which may take years to recuperate. Due to their high severity, security bugs are considered among the most sensitive bugs of all types. Though it is self-explanatory, security bugs may make the software vulnerable to potential cyber threats. Sometimes, the software organization may not notice such attacks whereas in some cases, these attacks could cause monetary loss to the users, especially small and medium-scale businesses. XSS vulnerabilities, logical errors, and encryption errors are some of the commonest security bugs found in the software. Developers put special focus on checking the code to find any underlying [security bug](https://www.thinksys.com/security-testing-services/) to minimize the risk of cyber-attacks.

Unit Level Bugs:  
Unit level bugs are fairly common in software development and do not cause much damage to it as well. Facing basic logic bugs or calculation errors are considered unit-level bugs. The [testing team](https://www.thinksys.com/software-testing-services/) along with the agile team test a small part of the code as a whole. The reason why this testing method is preferred is to make sure that the entire code is working as it is meant to. While testing, the team may encounter unit-level bugs which can be fixed easily as the team is only working with a small code.

Functional Bugs:  
Software is as good as the feature it provides. If any of the functionality of a software is compromised, the number of users will start to decline drastically until it becomes functional again. A functional bug is when a certain feature or the entire software is not functioning properly due to an error. The severity of such bugs depends on the feature they are hampering. For instance, an unresponsive clickable button that is not functioning is not as severe as the entire software not working. [Functional testing](https://www.thinksys.com/functional-testing-services/) is done by the testing team to identify any such software bug causing functionality errors. Once identified, the team decides its further classification and severity.

Usability Bugs:  
Probably one of the most catastrophic bugs for software, a usability bug or defect can stop the software from working to its potential or make it entirely unusable. Examples of this bug in software testing are the inability to log in to the user account or the inefficient layout of the software for the user. The bottom line is that this type of defect or bug can make it complex for the user to use the software efficiently. The developers and engineers have to look out for the right [usability requirements](https://www.thinksys.com/qa-testing/complete-guide-to-usability-testing/) while testing the code to identify such bugs.

Syntax Errors:  
Syntax errors are among the commonest software bug types and do not allow the application to be compiled appropriately. This bug occurs due to an incorrect or missing character from the source code due to which the compiling will be affected. A small error like a missing bracket could lead to this problem. The development or testing team will get to know about this bug during compiling and will further analyze the source code to fix the missing or wrong characters.

Compatibility Errors:  
Whenever a software or an application is not compatible with hardware, or an operating system, it is considered as incompatible software or a compatibility error. Finding a compatibility error is not a common practice as they may not show up in the initial testing. Due to this reason, the developers should go for compatibility testing to make sure that their created software is compatible with common hardware and operating systems.

Logic Bugs:  
Another one of the most frequently found bugs in a software code, logic errors make the software give wrong output, software crash or failure. In the majority of cases, these bugs are caused due to coding errors where it may make the software stuck in a never-ending loading loop. In that case, only an external interruption or software crashing are the two only things that can break the loading loop.

Priority-Based Software Bugs:  
The foremost category here is priority-based software bugs. These are based on the impact these bugs leave on the business. Here, the developers will analyze the bug to determine its impact and its defect priority. Afterward, the timeline is given to each bug where it should be rectified within the stipulated time frame to minimize the bug effect on the user. Here are the four types of priority-based software bugs.

Low-priority defects:  
Low priority defects do not cause much impact on the functioning of the application. Rather, they are more about software aesthetics. For instance, any issue with the spelling or the alignment of a button or text could be a low-priority defect. The software testing will move to the exit criteria even if the low-priority defects are not fixed, but they should be rectified before the final release of the software.

Medium-priority defects:  
Akin to low-priority defects, medium-priority defects do not cause any significant impact on the software, but they should be fixed in any subsequent or upcoming release. Such defects may not have the same effect for every user and it may vary with the device as well as specific configuration they have.

High-priority defects:  
Unlike the previous two, the exit criteria of high-priority defects are not met until the issue is resolved. Every bug falling in this category may make certain features of the software unusable. Even though it may not affect every user, it is mandatory to fix these bugs before any further step is taken in software development or testing.

Urgent Defects:  
As the name suggests, all bugs that should be dealt with utmost urgency fall under this category. Urgent defects may leave a lasting impact on the brand image as well as affect the user experience drastically. The stipulated timeline for fixing these bugs is within 24-hours of reporting.

Software Bugs by Severity:  
Depending on the technical effect that the bug will cause on the software, the bugs are categorized into four categories.

Low Severity Bugs:  
Low severity bugs do not cause much damage to the functioning of the software as their primary target is the user interface. For instance, the font of the text on the program differs from what was used. These bugs can be fixed easily and are nothing to worry about.

Medium Severity Bugs:  
Every bug that can affect the functionality of the software a little bit is considered a medium severity bug. All such bugs make the software function different from what it is supposed to function. Though they are not also major for the program, they should be fixed for a better user experience.

High Severity Bugs:  
High severity bugs affect the software functionality, making it behave differently from what it was programmed for. Not only are such bugs damaging for the software, they sometimes make the entire software unusable for the user.

Critical Bugs:  
Critical bugs are the most damaging bugs in the category that can hinder the functionality of the entire software. The reason why critical bugs are considered the most damaging is that further testing on the software becomes impossible till such bugs exist in the software.

## ****Software Test Design Techniques****

1. Introduction to Software Test Design Techniques

2. Static and Dynamic Test Design Techniques

3. Black-box Test Design Techniques

a.Equivalence Partitioning (EP)

b. Boundary Value Analysis (BVA)

c. Decision Table Testing

d. State Transition Testing

**Why do we need to use Test Design Techniques?**

Exhaustive Testing is not possible, so we need to use Test Design Techniques in order to reduce the size of the input.

Exhaustive Testing is a Test approach in which the test suite comprises all combinations of input values and preconditions.

Exhaustive Testing is not recommendable due to Time and Budget considerations.

**Categories of Test Design Techniques?**

There are two main categories of Test Design Techniques, They are:

a. Static Techniques  
b. Dynamic Techniques

#### **2. Static and Dynamic Test Design Techniques**

**A. Static Techniques**

> Testing of the software documents manually or with a set of tools but without executing the Software.

Two types of static testing techniques

i) Reviews (Manual Examination)

ii) Static Analysis (Automated Analysis)  
——————————-  
**i) Reviews**  
Types of Reviews

a) Informal Review  
b) Walkthrough  
c) Technical Review  
d) Inspection

**ii) Static Analysis**  
Static analysis tools are typically used by developers, Compilers offer some support for Static analysis,

#### **B. Dynamic Test Design Techniques**

> The software is tested by executing it on a computer.

Categories of Dynamic Test Design Techniques

**i) Specification-based or Black-box Techniques**

a) Equivalence Partitioning (EP)  
b) Boundary Value Analysis (BVA)  
c) Decision Table Testing  
d) State Transition Testing  
e) Use Case Testing Etc…

**ii) Structure-based or White-box Techniques**

a) Statement Testing and coverage  
b) Decision Testing and Coverage  
c) Condition Testing, Multi Condition Testing etc…

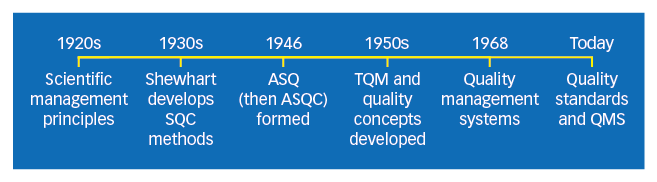
**iii) Experience-based Techniques**

a) Error Guessing  
b) Exploratory Testing

# **HISTORY OF TOTAL QUALITY MANAGEMENT**

The history of [total quality management](https://asq.org/quality-resources/total-quality-management) (TQM) began initially as a term coined by the Naval Air Systems Command to describe its Japanese-style management approach to quality improvement. An umbrella methodology for continually improving the quality of all processes, it draws on a knowledge of the principles and practices of:

* The behavioral sciences
* The analysis of quantitative and nonquantitative data
* Economics theories
* Process analysis

****

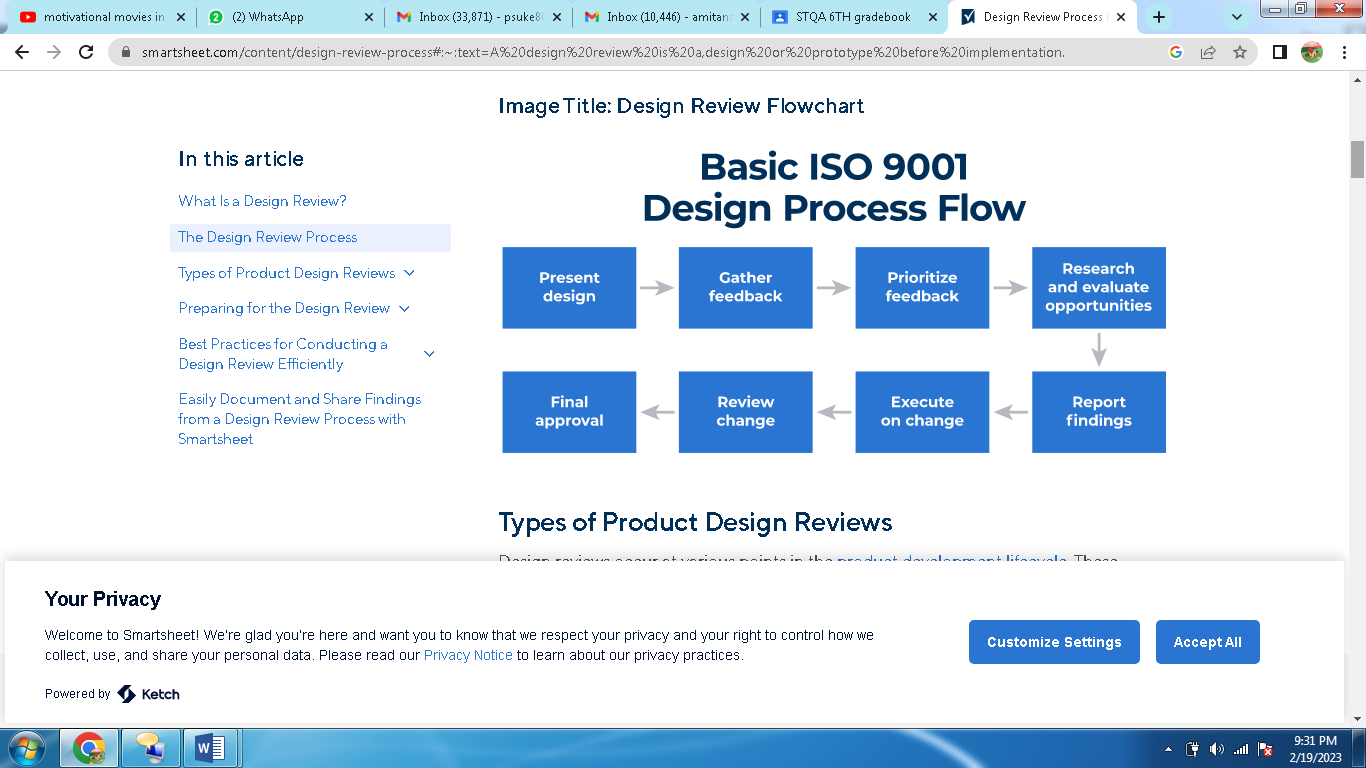
**History of Total Quality Management (TQM)**

|  |  |
| --- | --- |
| **1920s** | * Some of the first seeds of quality management were planted as the principles of scientific management swept through U.S. industry. * Businesses clearly separated the processes of planning and carrying out the plan, and union opposition arose as workers were deprived of a voice in the conditions and functions of their work. * The Hawthorne experiments in the late 1920s showed how worker productivity could be impacted by participation. |
| **1930s** | * [Walter Shewhart](https://asq.org/about-asq/honorary-members/shewhart) developed the methods for statistical analysis and control of quality. |
| **1950s** | * [W. Edwards Deming](https://asq.org/about-asq/honorary-members/deming) taught methods for statistical analysis and control of quality to Japanese engineers and executives. This can be considered the origin of TQM. * [Joseph M. Juran](https://asq.org/about-asq/honorary-members/juran) taught the concepts of controlling quality and managerial breakthrough. * [Armand V. Feigenbaum](https://asq.org/about-asq/honorary-members/feigenbaum)’s book *Total Quality Control,* a forerunner for the present understanding of TQM, was published. * [Philip B. Crosby](https://asq.org/about-asq/honorary-members/crosby)’s promotion of zero defects paved the way for quality improvement in many companies. |
| **1968** | * The Japanese named their approach to total quality "companywide quality control." It is around this time that the term [quality management systems](https://asq.org/quality-resources/quality-management-system) arises. * [Kaoru Ishikawa](https://asq.org/about-asq/honorary-members/ishikawa)’s synthesis of the philosophy contributed to Japan’s ascendancy as a quality leader. |
| **Today** | * TQM is the name for the philosophy of a broad and systemic approach to managing organizational quality. * Quality standards such as the [ISO 9000 series](https://asq.org/quality-resources/iso-9000) and quality award programs such as the Deming Prize and the [Malcolm Baldrige National Quality Award](https://asq.org/quality-resources/malcolm-baldrige-national-quality-award) specify principles and processes that comprise TQM. * TQM as a term to describe an organization's quality policy and procedure has fallen out of favor as international standards for quality management have been developed. Please see our series of pages on [quality management systems](https://asq.org/quality-resources/quality-management-system) for more information. |

# TQM’s seven basic elements

Seven basic elements capture the essence of the TQM philosophy: customer focus, continuous improvement, employee empowerment, quality tools, product design, process management, and supplier quality.

* **Customer focus**: Decisions of how to organize resources to best serve customers starts with a clear understanding of customer needs and the measurement of customer satisfaction. For example, the Red Cross surveys its blood donors to determine how it can make the blood donation experience more pleasant and convenient. It collects information on the place, date and time donors came in, and asks donors questions of whether the donation time was convenient, whether they were treated with respect and gratitude, how long they had to wait to donate, and whether parking was adequate. By understanding donors’ needs and experiences, Red Cross managers can determine strengths and weaknesses of the donation service process and make adjustments if necessary.
* **Continuous improvement:** An organizational culture that promotes continuous learning and problem solving is essential in the pursuit of zero defects. The Toyota Production System (TPS) is a universal continuous improvement system that has been effectively applied to many different types of organizations, including the health care industry. Essential elements of the TPS culture include studying process flow, collecting data, driving out wasteful non-value-added activities, and making everyone responsible for quality improvement. In the case of health care, the TPS approach enabled one hospital to analyze the causes of patient infections from catheters and pneumonia in patients on ventilators. With simple changes in procedures that prevented patients from getting these secondary illnesses, the hospital was able to save USD 40,000 per patient in these cases.
* **Employee involvement**: Employees in a TQM environment have very different roles and responsibilities than in a traditional organization. They are given responsibility, training, and authority to measure and control the quality of the work they produce, they work together in teams to address quality issues, they are cross-trained to be able to perform multiple tasks and have a greater understanding of the total production process, and they have a more intimate understanding of the operation and maintenance of their equipment. Employees are essential to the building of a continuous improvement organization.
* **Quality tools:** Discussion of the details of quality tools extends beyond the scope of this chapter, but there are seven basic quality tools that are used by front-line workers and managers in monitoring quality performance and gathering data for quality improvement activities. These tools include: cause-and-effect (fishbone) diagrams, flowcharts, checklists, control charts, scatter diagrams, Pareto analysis, and histograms. The beauty of these tools is that they are easy to understand and apply in on-going quality efforts.
* **Product design**: Product design is a key activity to avoid costly internal and external failure costs. For example, when a dental office designs the service process, it might have patients fill out a form that covers important information on general health issues, allergies, and medications. This helps to avoid future complications and problems. Staff, hygienists, and dentists are highly trained to follow proper procedures, the facility is both functional and pleasant, and the equipment and tools are state of the art to ensure that the patient’s desired outcome is achieved. In a manufacturing setting, products should be designed to maximize product functionality, reliability, and manufacturability.
* **Process management:** “Quality at the Source” is an important concept in TQM. It means that managers and employees should be focused on the detailed activities in a process where good or bad quality is created. For example, in a Toyota plant in the United States in Georgetown, Kentucky, one of the work stations was responsible for installing seat belts and visors in every vehicle that came along the assembly line. There were 12 possible combinations of visors and seat belts that would go into any particular vehicle and the worker had to select the right combination and install the items in the vehicle in 55 seconds. Even the best workers made several errors during a shift on this activity. After studying the process, the workers came up with an idea to put all the items for a particular vehicle model in a blue plastic tote. With this change, the worker only had to make one decision per vehicle. Almost all the errors from the previous system were eliminated with this simple solution.
* **Supplier quality**: The focus on quality at the source extends to suppliers’ processes as well, since the quality of a finished product is only as good as the quality of its individual parts and components, regardless of whether they come from internal or external sources. Sharing your quality and engineering expertise with your suppliers, having a formal supplier certification program, and including your suppliers in the product design stage are important measures to take to ensure that quality at the source extends to the supplier network.
* The Design Review Process
* During the design review process, stakeholders assess the design in question. In a design review meeting, designers present the design, and participants provide feedback in the form of questions, challenges, and suggested improvements.
* Following the design review meeting, the designer and immediate team consolidate and prioritize feedback. Designers use the feedback to evaluate opportunities and report back to the larger group on findings and action items.
* Image Title: Design Review Flowchart



### Engineering Design Review Procedure

Systems engineering conducts design reviews in a structured and specific order. The goal is to optimize time by reviewing design at critical junctures: design review, test readiness, and system verification.

For example, systems engineering typically conducts design reviews as follows:

* **Alternative Concepts Review (ACR):** Review alternative ideas.
* **System Functional Review (SFR):** Ensure the system can satisfy initial capabilities.
* **Software Specification Review (SSR):** Formally define software requirements.
* **System Design Review (SDR):** Ensure the systems will satisfy the initial requirement within budget and on time.
* **Preliminary Design Review (PDR):** The PDR ensures the system is operationally useful.
* **Critical Design Review (CDR):**This technical review ensures a system can proceed to the next steps.
* **Test Readiness Review (TRR):** Determine if the system in review is ready to move into formal testing.
* **Final Design Review (FDR):**This review takes place when most design work and testing are complete. This review is the last opportunity to provide feedback on the design.
* **System Verification Review (SVR):** Verify the final product performance.
* **Production Readiness Review (PRR):** Determine if the product is ready for production.
* **Physical Configuration Audit (PCA):** Examine the configuration item against the documentation to verify the product meets product baseline specifications.