# Chapter 14
# Structured Approach to Build-in Design Robustness to Improve Product Reliability


**Vic Nanda and Eric Maass**

**Abstract** Robustness is defined as the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions. The objective of robustness is to deliver high reliability to customers. Robustness ensures that product design is immune to and can gracefully handle invalid inputs and stressful environmental conditions without any disruption or degradation of service to the end user. Robustness can be systematically built into any system, regardless of hardware or software, by following an end−to−end approach that encompasses product requirements, design, development, and testing. This chapter provides a structured approach to design in robustness by mapping baseline use case scenarios as 'sunny day' scenarios, identifying potential failures using P−diagrams and Design Failure Modes & Effects Analysis (or, "rainy day" scenarios), and proactively embedding design controls to strengthen product robustness and minimize field failures. The authors describe an innovative way to prioritize design improvements not just by traditional Risk Priority Number (RPN) of design failures but by considering the actual magnitude of risk reduction, as well as by factoring in cost of design improvements in prioritization decisions. Robustness once built−in product design must be validated through vigorous robustness testing to provide objective evidence of design robustness and support decision-making regarding product readiness for release. A comprehensive approach to robustness testing is described along with guidance on how to design a comprehensive suite of robust test cases for high reliability.

**Keywords** Robustness · Robust design · P-diagram · DFMEA · Robust testing

V. Nanda (✉)
NOKIA, Espoo, Finland
e-mail: vic_nanda@hotmail.com

E. Maass
Medtronic Restorative Therapy Group, Tempe, USA



K. B. Misra (ed.), *Handbook of Advanced Performability Engineering*,
https://doi.org/10.1007/978-3-030-55732-4_14

**Fig. 14.1** Mars opportunity rover

## 14.1 Introduction to Robustness

Robustness is defined as the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions (IEEE [1]). The objective of robustness is to deliver products with high reliability to customers and end users.

One of the best examples of robustness is the Mars Opportunity rover that was launched on July 3, 2003 and landed on Mars on January 25, 2004, with a planned 90−sol duration of activity (slightly more than 90 earth days), yet it remained operational until June 10, 2018 and exceeded its operating plan by 14 years, 46 days in earth time, or 5111 sols, 55 times its designed lifespan! It withstood hard environmental conditions and stress including Martian dust storms far beyond its intended lifespan and performed remarkably well beaming stunning visuals of the Martian landscape and provided wealth of scientific data (Fig. 14.1).

Here on earth, there are several examples of robust products such as consumer products that work faithfully under all reasonable operating and environmental conditions such as mobile phones, personal computers, automobiles, and so on.

## 14.2 Why Robustness Matters?

When products are not robust, it can cause immense harm and inconvenience to end users and damage to company reputation. For example, in 2017, Amazon Web Services (AWS) experienced an 11-h outage due to a simple human error that crippled popular websites like Netflix and top 100 online retail websites. AWS was forced to issue a public apology and a detailed post-mortem that stated an authorized employee

executed a command that was supposed to remove a small number of servers for one of the AWS sub-systems for maintenance but one of the parameters for the command was entered incorrectly and took down a large number of servers that supported critical services—the engineer intended to decommission 0.03% of the servers for maintenance but inadvertently entered 30%, that knocked out large part of the ASW network as shown in Fig. 14.2. Clearly, the system was not robust in that it had no designed-in control mechanism to preclude such a human error, and further it pointed to a deficiency in the AWS server test processes in that there was no test case that exercised this scenario. Consequently, AWS made changes to its internal tools and processes, so that servers were taken down more slowly and blocked operations that reduced server capacity below safety check levels.

Another robustness failure had even more catastrophic consequences—In 2015, an Airbus 400 M military transport aircraft as shown in Fig. 14.3 crashed due to a faulty software configuration. The crash investigation confirmed that there was no structural defect in the airplane but a pure software defect in the configuration settings programmed in the electronic control unit (ECU) in three of the aircraft's four ECUs—a file-storing torque calibration parameters for each engine were somehow 'accidentally wiped' when the software was being installed. As a result, three of the aircraft's engines automatically shut down in flight. Worse, as per the design of the software, the pilot of the A400M would not have gotten an alert about the missing data until the aircraft was already at an altitude of 400 feet. No cockpit alert about the data fault would appear while the aircraft was on the ground. The A400M, which was on a final test flight before delivery to the Turkish Air Force, reached an altitude of 1,725 feet after takeoff before three of the engines stalled and it crashed during an attempted emergency landing. There were no survivors.

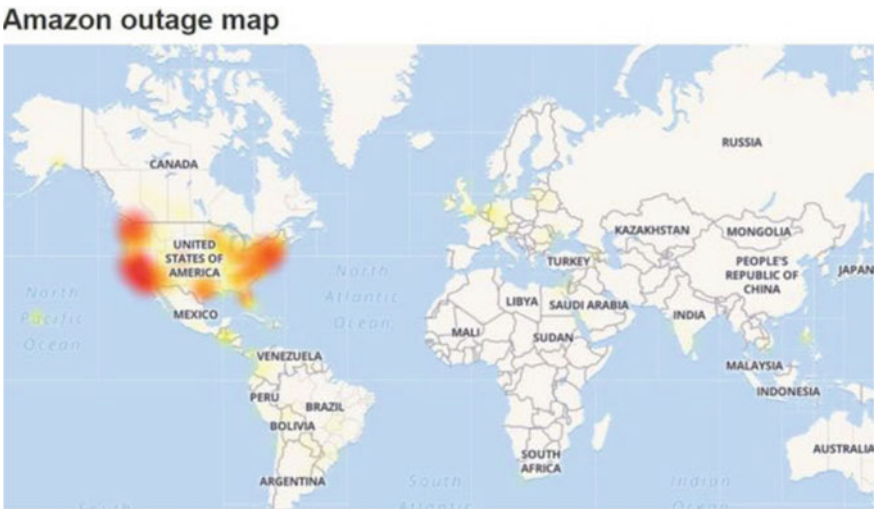


**Fig. 14.2** AWS outage impact (2017)

**Fig. 14.3** Airbus A400M military transport aircraft that crashed in 2015 due to software design defect

## 14.3 Benefits of Robustness

Robustness in products offers compelling benefits to companies producing those products as well as the customers and end users. It reduces Cost of Poor Quality (COPQ) from internal and external failures that directly contributes to reduced operating expenses and *improved bottom-line* for the producer. Reduction in customer-reported defects results in improved customer satisfaction and loyalty, improved customer retention, and sales growth from existing and new customers. Improved quality therefore directly contributes to *improved top-line*. These twin benefits of reduced operating expenses by virtue of reduced COPQ and improved top-line by virtue of sales growth from current and new customers *improve business profitability*.

As an example, in the software industry, according to The Cost of Poor Quality Software in the USA: A 2018 Report, the cost of poor quality software in the USA in 2018 is approximately $2.84 trillion (COPQ [2])!

## 14.4 Robustness Strategy

Robustness in products cannot be an afterthought and designed outside in—one cannot assure that a product is robust by merely testing for robustness. Robustness must be thought of proactively, it must be planned for all phases in a project. This is the fundamental concept of *shift left quality*—that is to prevent and find defects early in the product design and development process.

Robustness can be systematically built into any system, regardless of hardware or software, by following an end-to-end approach that encompasses product requirements, design, development, testing, and customer deliverables. In other words, robustness must be designed inside out.

## 14.5 End-to-end Robustness Lifecycle

The end-to-end lifecycle to plan, design-in and validate robustness in products covers four phases:

1. Robustness specifications
2. Robust architecture and design
3. Robust development
4. Robust testing.

We provide an overview of each of these phases before covering each phase in detail.

### *14.5.1 Robustness Specifications*

For any product, the design and development lifecycle begins with requirements specifications, and planning for robustness starts with:

- Documenting robustness requirements that specify anticipated product behaviour in the event of:
  - Incorrect, incomplete, delayed, or missing inputs
  - Execution errors
  - *Edge cases*—errors associated with input(s) at maximum or minimum value
  - *Boundary cases* when one input is at or just beyond maximum or minimum limits
  - *Corner cases* that are outside normal operating parameters and indicate a stress scenario when multiple environmental variables or conditions are simultaneously at extreme levels even though each parameter is within the specified range for that parameter
- Specifying fault tolerance requirements, including failover requirement for a faulty system to 'failover' or gracefully handoff to a backup system in the event of a failure
- Documenting traceability of robustness requirements to design specifications and test cases to assure that the robustness requirements are designed in and validated during testing, and
- Requirement reviews including verification of robustness requirements and traceability.

### 14.5.2 Robust Architecture and Design

Robust architecture and design is about having a design that can gracefully handle invalid inputs, internal errors and failures without unrecoverable catastrophic product failure. The objectives of robust architecture and design are to:

- Conform to robustness specifications, so that subsequent product design and development can successfully be validated against requirements and designs for robustness, respectively
- Identify potential vulnerabilities or high-risk 'failure points'
  - 'Hot spots', or design elements with intense use and therefore higher risk of failure,
  - 'Weak spots', or design elements that are known to be fragile from historical defect data
  - 'Critical interfaces' between sub-systems and modules
- Gracefully handle invalid inputs, processing errors and failures
- Verify that the robustness requirements have been adequately met (verification is performed in design reviews).

### 14.5.3 Robust Development

Robust development entails developing the product in accordance with the robustness requirements and detailed designs, to:

- Ensure the design for robustness is fully implemented, including following development guidelines and best practices to minimize failures
- Include development reviews to verify that the documented robust design has been implemented.

### 14.5.4 Robust Testing

The purpose of robust testing is to inject invalid inputs, introduce error and failure scenarios and stressful environmental, and use conditions to verify whether the system behaves as accepted by the customer. Robust testing is during the entire product development lifecycle from individual modules, sub-systems, to the integrated system to ensure robustness at all levels. Robustness testing validates that all the robustness requirements have been implemented.

To summarize, from a robustness perspective, the primary goal during requirements specification, architecture, design and development is *fault prevention and fault tolerance*, while the goal during robust testing is *fault identification and removal*.

We will now look at each of these phases in detail, starting with robustness specifications.

### *14.5.5 Creating Robustness Specifications*

The first step to defining robustness specifications is to map the baseline use case scenarios—understand the ideal use cases of the overall system with the help of a block diagram that depicts the end-user, inputs and interactions, to show what is the default scenario(s) assuming no exceptions and errors. Such a use cases is commonly referred to as the 'happy path' or 'sunny day scenario'. At this time, we assume no errors or failures. For example, for a customer using an A™ to withdraw money, we first assume that the customer correctly enters the PIN code and correctly makes the right user selections to withdraw the money.

Next, identify potential failures in the sunny day scenario—the 'what if' scenarios—what if the input is incorrect? What if the input is incomplete? While both of these are failure scenarios, but they are distinct and constitute individual *failure modes*—one when the input is incorrect or the other when it is incomplete. Likewise, there may be additional failure modes pertaining to the inputs. In the A™ example, what if the customer entered the PIN incorrectly? What should the A™ do? What does the customer reasonably expect it to do? Should it reject the transaction and return the card? Should it display an error message and allow the customer to make a second attempt?

Failure modes are not restricted to user or data inputs (from other systems) alone. What if there is an internal failure or execution error at a module or sub-system level? What should happen in these failure scenarios? 'What should happen' equates to the requirement specification for how the failure is expected to be handled (as expected by the customer and end user).

Figure 14.4 shows an example of a basic block diagram with some but not all potential points of failure. Such use cases that depict failure modes are also called 'rainy day scenario' or 'unhappy path'. Such block diagrams can be built at system, sub-system, sub-assembly and module or component level to fully map out potential failures at all levels of design.

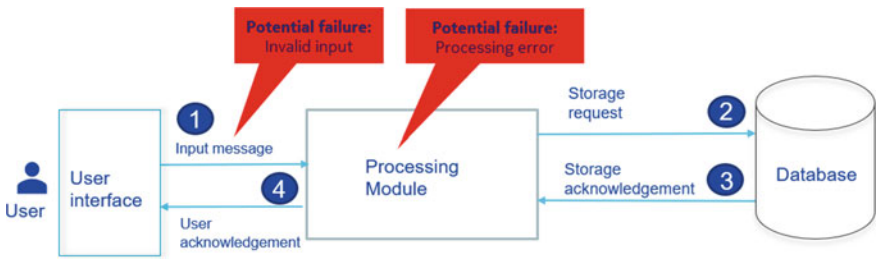


**Fig. 14.4** Mapping of baseline use case with potential failure scenarios

## 14.6 Parameter diagram (P-diagram)

In his book 'Quality Planning & Analysis', Juran describes the purpose of P-diagrams as follows: "the most basic product feature is performance, i.e. the output—the colour density of a television set, the turning radius of an automobile. To create such output, engineers use principles to combine inputs of materials, parts, components, assemblies, liquids, etc. For each of these inputs, the engineer identifies parameters and specifies numerical values to achieve the required output of the final product" (Juran [3]).

The Parameter Diagram (P-Diagram) takes the inputs from a system/customer and relates those inputs to desired outputs of a design that the engineer is creating, also considering non-controllable outside influences (Fig. 14.5).

During requirement flow-down, potential problems can be anticipated at the sub-system, sub-assembly and the component levels. The system-level flow-down will involve a birds-eye view of failure modes and will involve a broader cross-section of expertise for this purpose—but the anticipation of failure modes and mechanisms at sub-system and component levels will involve a more focused set of experts to dissect the potential problems involved at that deeper, more detailed level. Essentially, at these subsequent iterations, the sub-system and component under consideration become 'the system' for the team. It is worth noting that many of the sub-systems for complex products could literally *be* the 'system' or product for the same or for other companies. For example, many cellular phones include digital cameras—and digital cameras are products for camera manufacturers.

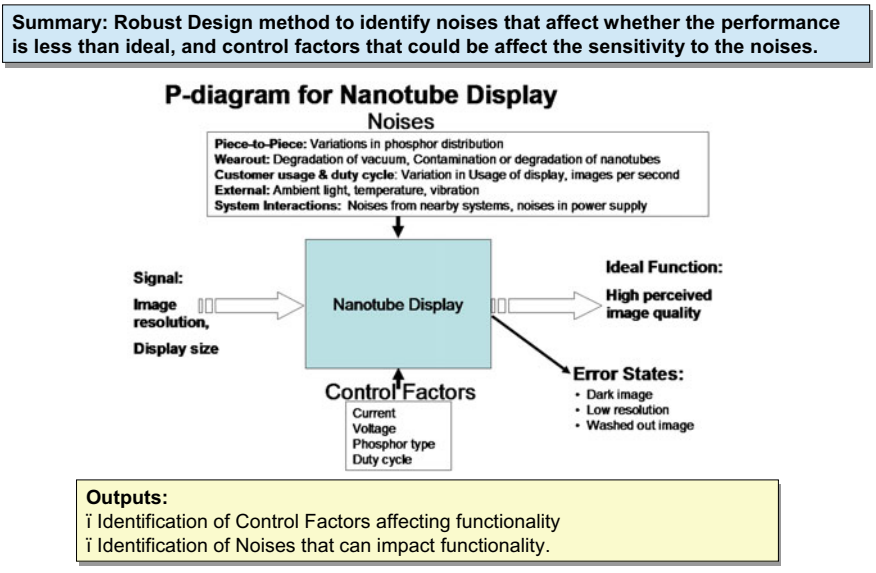


**Fig. 14.5** Example of P-diagram

Either as an integrated sub-system, or as a separate product, anticipation of potential problems is a vital first step towards prevention of problems. The P-Diagram is valuable in anticipating and preventing problems if not adequately addressed could impact the success of the product. It does this by enabling engineering teams to document input signals, noise factors, control factors, error states, and ideal response:

- Input signals are a description of the inputs received by the system, which are processed to generate the intended output.
- Control factors are *parameters or design improvements that can potentially prevent, mitigate, and/or detect the failure modes.*
- Error states are any undesirable and unintended system outputs. These are referred to as *Failure Modes—'Real-life' failures that can happen when the system is deployed.*
- Noise factors are *environmental factors or internal factors that can potentially impact normal system operations and result in failure modes.* The design must be robust against the expected noise factors.
- Ideal response is the intended output of the system.

Steps to complete the P-diagram are:

1. Identify signal
2. Identify intended function or result
3. Identify noise factors
4. Identify 'real-life' failure modes from the noise factors
5. Identify control factors.

A P-Diagram can help with the development of the Design Failure Mode and Effects Analysis (DFMEA), in which the error states or deviations from the ideal function (at the lower right of P-diagrams) could suggest failure modes to be included in the DFMEA, and the noises (at the top of the P-Diagrams) could suggest potential causes for the failure modes.

The team approach for identifying control and noise factors used in developing the P-Diagram can be leveraged in the flowing down requirements to the next level. The P-Diagram can also prove useful in generation and subsequent evaluation of alternative concepts for the sub-system, module or component, particularly in terms of considering the noises that can affect performance when brainstorming potentially robust design approaches—and the relative insensitivity of the alternative concepts to those noises can and should be considered in selecting a superior concept for the sub-system, module, or component.

The P-Diagram can also prove valuable during transfer function determination and initializing the identification of control and noise factors to use in an experimental design approach. The P-Diagram will also prove valuable for evaluating and optimizing robustness against the noises and verification of reliability, and some of the noises from the P-Diagram can be used as stress factors for reliability evaluation (Maass [4]).

## 14.7 Identifying Detailed Failure Modes with D-FMEA

The primary objective of DFMEAs is to help analyze, improve, and control the risk of product or service or feature or functional area design failure in meeting customer needs or expectations. It is supposed to be a living document that is initially created during product design and then maintained after product release as fixes are made to the product and enhancements are made in future releases.

DFMEA is essentially a risk identification and mitigation tool and it progresses through the following phases: risk identification, risk characterization, risk aversion, and improvement actions prioritization.

**Risk Identification**: The failure modes from the P-diagrams are populated in the DFMEA table and these are further expanded to identify even more failure modes (from team brainstorming, past defects) which may not otherwise be possible to depict in the P-diagram (in order to minimize complexity of the P-diagram).

**Risk Characterization**: After listing each failure mode, the team lists the effect of each failure and scores:

- The severity of the risk on a 10-point scale with 10 indicating most severe,
- Likelihood of occurrence on a 10-point scale with 10 indicating most likely, and
- Detection mechanisms to detect or prevent the failure mode with 10 indicating no ability to detect or prevent and therefore risk of defect escape to the customer, and 1 indicating strong detection mechanism to prevent the defect escape.

The RPN (Risk Priority Number) score is then computed and it is the product of severity of impact, likelihood of occurrence and detection score, and it provides a risk score for each failure mode and helps assess the relative risk of each failure. Because each of the three relative rankings is on a scale that ranges from 1 to 10, the overall RPN will always be between 1 and 1000, with 1000 indicating the failure mode with the highest risk.

**Risk Aversion**: In order to avert design risks, one must completely understand all the root causes (Fig. 14.6, column 5) and assess each failure mode and its root causes individually with by assigning them separate RPN scores. Risk aversion strategies include:

1. Risk mitigation (acting to reduce the risk probability and impact),
2. Risk avoidance (choosing a course of action that eliminates the risk),
3. Risk transfer (transferring the risk to another more appropriate product team to own the risk), and
4. Risk acceptance (plan a contingency to contain the risk in case the risk is realized).

As a general rule, improvement action plans from DFMEA aim to minimize the RPN score, typically below a threshold of 100 by reducing likelihood of occurrence and improving detection and prevention (control), while reduction in severity of impact is possible only be altering the design. For example, if the brakes in a car fail, improvements will focus on reducing the likelihood of failure and improving detection of brake failure, but the severity of impact on the passengers (in this case,

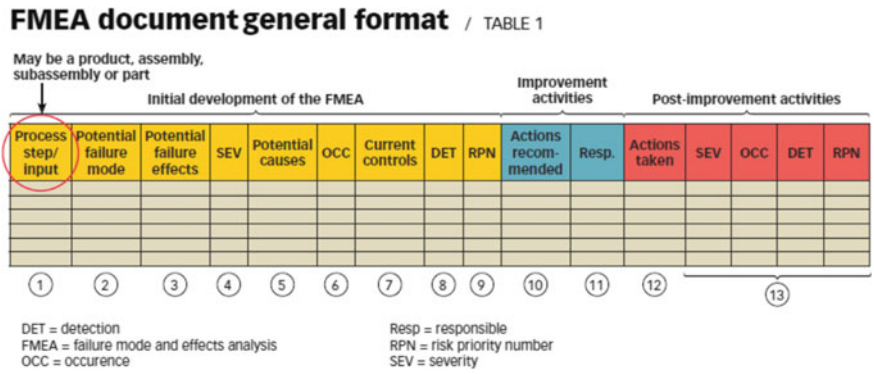


**Fig. 14.6** FMEA template

risk of injury) in the event that they do fail would be the same unless design changes are made to lower the risk of injury, for example, adding airbags and crumble zones in the car to minimize injury to passengers (and thus lower the score for severity of impact).

After the improvement actions are identified, a new RPN is computed and compared with the original RPN to assess the % reduction in risk and to assess if the new RPN is below the acceptable threshold.

## 14.8 Improvement Actions Prioritization

The original RPNs are sorted in descending order in order to define prioritized improvement actions starting with the highest risk failure modes down to the threshold below which the risks are considered low risk (typically, failure modes with RPN score of 100 or less). This prioritized list of improvement actions is then translated into execution plans for 30–60–90 days (or other timeframes).

This approach does have some major limitations. It does not factor in the extent of business impact, or extent of reduction in risk exposure as a result of risk aversion actions, and it does not factor in cost, difficulty or effort to reduce the risk. Figure 14.7 shows risk prioritization approach that factors in reduction in risk exposure and cost to prioritize the risks. All failure modes that are above the threshold (old RPN) are reviewed for reduction in RPN after proposed improvement actions and the magnitude of reduction in RPN is plotted on the Y-axis. The scale may be numbered on the Y-axis from the least to the most RPN reduction seen in the project, for example, H on the Y-axis would be max RPN reduction and L would be least reduction in RPN. Likewise, the X-axis can be numbered according to the maximum cost of improvement actions to the least cost or may be used simply with low–medium–high cost variable. Alternatively, cost may be replaced with 'difficulty' in implementing the improvement actions.

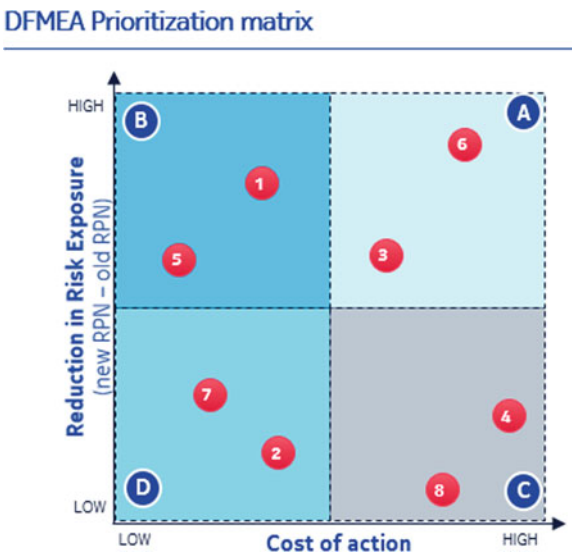


**Fig. 14.7** Failure mode prioritization based on reduction in risk and cost of action

The failure modes that must be addressed first lie in quadrant B (maximum reduction in RPN with least to moderate cost), followed by quadrant A or D (depending on if the team prioritizes reduction in risk or cost of action). Finally, failure modes in quadrant C are those with the least reduction in RPN and require significant cost to reduce risk below the acceptable threshold. This sequence can also be used to accordingly define the improvement timeframe starting with immediate action on the failure modes that offer the greatest reduction in risk exposure.

### *14.8.1 Embedding Design Controls for Robustness*

This step involves updating the original 'rainy day' block diagram to embed the design control actions from the DFMEA that mitigate the risks. It shows where the design vulnerabilities are and what control actions have been identified to avert those design risks. It therefore serves as powerful visual representation to view the landscape of design risks and actions identified to improve design robustness as shown in Fig. 14.8. Again, as previously mentioned, this analysis can be performed at sub-system and module level and all such block diagrams of lower levels of architecture and design would need to be updated to depict the control actions identified to avert risks in the entire product architecture and design.

While non-engineers tend to rely upon heuristic thinking for decisions, engineers traditionally use deterministic modelling in their tasks.

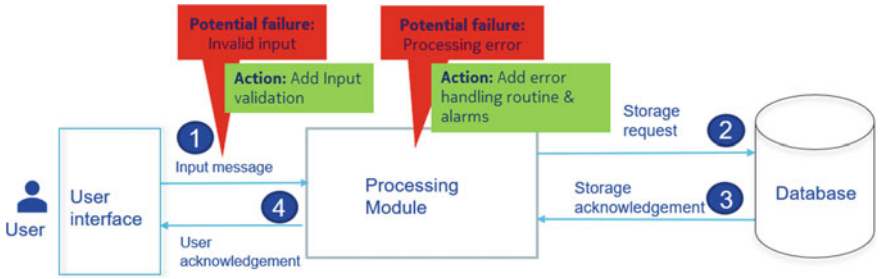


**Fig. 14.8** Rainy day scenario with embedded design controls to reduce risk of failure modes

Design robustness drives beyond deterministic to probabilistic or stochastic modelling (Maass [4]). The melding of engineering modelling with probabilistic methods is referred to as Predictive Engineering. Probabilistic methods include Monte Carlo Simulation, the generation of system moments method, and Bayesian Networks to predict the probability that the product will meet expectations over the range of noise factors.

On the right side of Fig. 14.9, parallel line segments represent the set of requirements for the product. Critical requirements, two line segments with arrows represent a subset of the requirements that are prioritized for the intensity of robust design. The goal is to predict the distribution for each critical requirement over the range of noise factors—represented as the distributions to the far right of Fig. 14.9.

To predict the distributions for the critical requirements, the critical requirements are flowed down to control and noise factors as represented by the line segments on

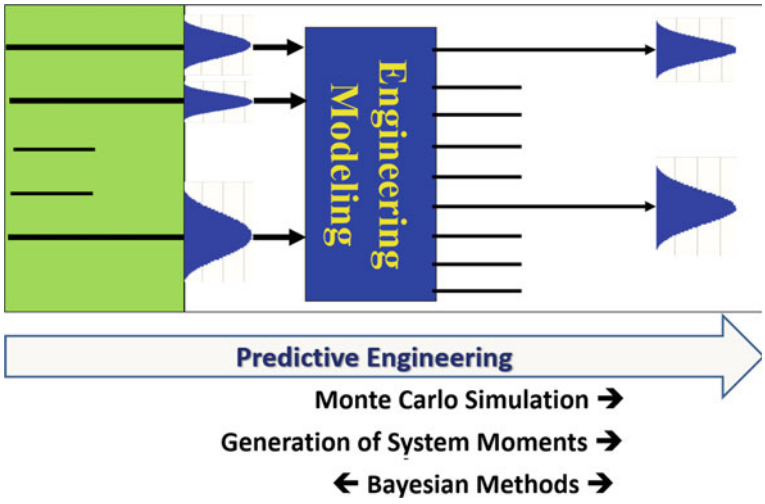


**Fig. 14.9** Robust design/predictive engineering as the melding of deterministic engineering modelling with probabilistic methods

the left side of Fig. 14.9. The P-diagram provides a useful method for identifying these control and noise factors.

Screening methods such as fractional factorial experimental designs, Plackett–Burman designs, or Definitive Screening Designs can be used to determine a subset of the control and noise factors that dominate in terms of their impact on the critical requirements, as represented by the three arrows emerging from the shaded area to the left of Fig. 14.9.

Distributions of the range of values for each of these dominant control and noise factors are estimated as represented by the distributions to the left side of Fig. 14.9.

Engineering modelling is used to develop equations or transfer functions for how the subset of dominant control and noise factors affects each critical requirement. The engineering modelling can use theoretical methods, also called first principles modelling; examples include Ohm's Law, Voltage = Current x Resistance, or simple additive models like the total thickness of stacked layers or total delay from the sum of delays for each step.

Engineering modelling can also use designed experiments that vary the control and noise factors either through simulation or emulation. Mechanical simulation can use Finite Element Analysis; electronic simulation can use electronic circuit simulation tools. Emulation can use hardware that emulates the actual hardware and software system that may not yet exist in its final form.

Combinations of control and noise factors at varied settings can be defined according to an experimental plan such as Central Composite Design for Response Surface Modelling or Space-Filling Designs. These combinations of settings for control and noise factors are run through the simulation or emulation, and results for each critical requirement are obtained for each combination.

The combinations and the results can be analyzed using multiple regression to obtain empirical or semi-empirical equations of the form y = f(x1, x2 ,…, xp, noise1, noise2,…,noiseq).

The equation for each critical requirement obtained from engineering modelling, whether theoretical, empirical, or semi-empirical is deterministic at this point. Probabilistic methods (Monte Carlo Simulation, Generation of System Moments method or Bayesian Networks) are used in conjunction with the equations to predict the distributions of each critical requirement over the range of control and noise factors.

If the predicted distributions for each critical requirement are satisfactory—that is, with the specification limits for that critical requirement, robust design has been achieved. If the predicted distributions are not satisfactory, optimization methods such as Steepest Ascent, Simulated Annealing, the Genetic Algorithm, Branch and Bound, or Newton–Raphson can be used to explore and find more optimal settings for the control factors that render the design more robust to the noise factors.

## 14.9 Robustness Testing

In robust testing, the primary objective is to validate that the system or component can function correctly in the presence of invalid inputs or stressful environmental conditions, including misuse and abuse test cases. Planning for robustness starts with robustness test strategy and includes robustness test planning and execution.

### *14.9.1 Robustness Test Strategy*

Robust testing always starts from the inside, from the smallest component, module, sub-assembly, and it progresses outwards to greater aggregation of the product until the final finished product. Therefore, robust testing is the responsibility of the development and test organizations and not the test organization alone.

Robust testing begins with developing an overall robustness test strategy and plan, including identifying all phases of product development and test where robustness testing will be performed, identifying test resources, types of robustness testing, test cases, test procedures, test tools, problem management for defects reported from robust testing, defects database, and defect profiles from types of robust testing to inform future robust test strategy.

### *14.9.2 Robust Test Planning and Execution*

How should product teams design a comprehensive suite of robust test cases for high reliability? There are four potential sources of robust test cases:

1. **Historical Defect Data**: One can review past customer-reported and internally founds defects to gain insights into which product sub-systems, components, and sub-assemblies have been most prone to robustness defects. This requires reviewing all customer-reported defect data and categorizing all defects, with techniques such as affinity analysis (clustering) of keywords in defect reports to categorize defects that indicate poor design robustness. These can then be used to design new robust test cases.
2. **Requirement Specifications**: This involves reviewing the requirement specifications to understand how the product is expected to gracefully handle invalid inputs and environmental stresses, when such requirements are specified, and identifying robust test cases based on requirements by considering ‘what if’ scenarios discussed earlier. In addition, one can identify what features, components, sub-assemblies, and interfaces are new, unique, and difficult (complex), collectively referred to as NUD design elements that pose greater risk of failure, and design test cases to test them

3. **Product Architecture and Design**: Review of the product architecture and design help identify the product hot spots, weak spots, critical interfaces, as previously described.

Examples of robust test cases include testing with invalid inputs, testing in unexpected environments, testing the product in stressful environmental conditions to predict how the product will perform while being exposed to expected stress levels or operating conditions above specification limits to create failure scenarios that would likely have occurred under normal stress over a period of time. Typically, this is done using one stress parameter at a time, such as vigorously shaking a mobile device at high intensity over a period of time, and this is referred to as Accelerated Life testing (ALT). Similarly, High Accelerated Life Testing (HALT) also tests a product for robustness to elevated stress beyond specification limits and may include multiple stress parameters such as shaking the mobile device while also raising the environmental temperature and continue to raise the stress up to the point of failure and discover the performance limit of the product (beyond the specification limit). This stress testing methodology is also referred to as 'test-to-fail' where a product is tested until its failure. Therefore, ALT helps answer the question when the product will fail and HALT helps define the difference between performance and specification limits to answer the question how much 'design margin' exists before eventual product failure.

## 14.10 Conclusion

Robust products can be designed by following a comprehensive end-to-end process as outlined in this chapter. To deliver reliable products, one must start with requirement specification, through design, development, and testing. Design robustness must be built inside out. Techniques described in this chapter can help proactively identify design vulnerabilities that can be systematically assessed, resolved through design improvements, and verified through robustness testing to assure customers that the product will perform reliably in the field.

## References

1. IEEE standard glossary of software engineering terminology, IEEE Std 610.12–1990.
2. https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2018-report/The-Cost-of-Poor-Quality-Software-in-the-US-2018-Report.pdf.
3. Juran, J. M., Gryna Frank, M. (1993). *Quality Planning and Analysis: From Product Development Through Use*, McGraw Hill.
4. Maass, E.C., McNair, P.D. (2009). *Applying Design for Six Sigma to Software and Hardware Systems*, Prentice Hall.

**Vic Nanda** is Head of Quality Capabilities Scaling & Nokia Quality Consulting at Nokia. He leads Nokia's Continuous improvement and Lean Six Sigma programs that have delivered over 1 Billion Euros of business impact from 2012–2019. His experience at major telecoms such as Motorola, Nortel, Ericsson and others spans operational excellence and quality management systems deployment using Lean Six Sigma, kaizen events, CMMi, ISO/TL 9000, and PMBOK practices. He has authored three books and several publications on Lean Six Sigma, quality management systems, and process improvement. He is a frequent industry speaker and has taught 1000+ Lean Six Sigma belts, coached 150+ Lean Six Sigma belts to certification, and delivered cumulative business impact in excess of 200M dollars. He is a Master Black Belt and holds 10 other quality certifications. He has consulted for the US Government, NASA (OSIRIS-REX mission to Asteroid Bennu), Arizona State University, and more.

He was awarded the American Society for Quality (ASQ) Golden Quill Award, and the ASQ Feigenbaum Medal by the ASQ for displaying outstanding characteristics of leadership, professionalism, and contributions to the field of quality. He has a Masters in Computer Science from McGill University, Bachelors in Computer Engineering from University of Pune, India, and executive education from Harvard Business School.

**Dr. Eric Maass** is Senior Director for DFSS/DRM for Medtronic Restorative Therapy Group. He is responsible for developing and leading the DRM strategic plan and focus for most of the company and has been the chief architect for Medtronic's DFSS/DRM BB and MBB programs. He was recognized with Medtronic's individual Star of Excellence award for 2012 and has been recognized as a Medtronic Technical Fellow. He joined Medtronic in October 2009, after 30 years with Motorola in roles ranging from Research and Development through Manufacturing, to Director of Operations for a $160 Million business and Director of Design and Systems Engineering for the Wireless group of Motorola SPS. He was a co-founder of the Six Sigma methods at Motorola, and had been the Lead Master Black Belt for DFSS at Motorola. His book, *Applying DFSS to Software and Hardware Systems*, provides clear step-by-step guidance on applying DFSS for developing innovative and compelling new products and technologies, while managing the business, schedule and technical risks. He received his Bachelor's degree in Biological Sciences from the University of Maryland Baltimore County, his Master's degree in Biomedical and Chemical Engineering from Arizona State University and his PhD in Industrial and Systems Engineering from Arizona State University. Dr Maass also currently serves as an Adjunct Professor at Arizona State University, and as chairman of the Industrial Advisory Board for the NSF-Sponsored B.R.A.I.N Industry/University collaborative research consortium.