Design for Six Sigma (DfSS) in Software

Ajit Ashok Shenvi
Philips Innovation Campus
India

1. Introduction

It is difficult to imagine a product or service these days that does not have software at its core. The flexibility and differentiation made possible by software makes it the most essential element in any product or service offering. The base product or features of most of the manufactures/service providers is essentially the same. On one hand, the differentiation is in the unique delighters, intuitive user interface, reliability, responsiveness etc i.e. the non-functional aspects and software is at the heart of such differentiation. On the other hand being such a crucial aspect of a product or a service, failures or quality issues and correspondingly, user dissatisfaction often get associated with software.
Compared to other fields such as civil, electrical, mechanical etc, software industry is still in its infancy. Although there are some architecture patterns, design rules, coding guidelines, software development is not yet a fully mature engineering discipline. The fundamentals principles of PDCA (Plan-DO-Check-Act) do apply, however intangibility associated with the output and absence of samples makes the in-process verification at every stage a very and challenging and person dependent affair.
Historically the development of software has been governed by "software engineering principles" applied though application of SEI-CMM^R model. This is a collection of best practices across the industry that has been compiled and structured into a framework and has strong emphasis on a process based philosophy. The essential idea is that when the input to the process as well as process steps are controlled, the output of the process is expected to be predictable and of good quality. On the other hand, the core principle of DfSS is to minimize variation. As it is often said – ***"Variation is an enemy of Quality"***. However, software is mainly digital in nature (it works or it does not, yes/no) and is expected to be 100% predictable with "no inherent variation" by itself –i.e. the same software would have exactly the same behaviour under same environmental conditions/inputs. In addition, there is nothing like "samples" with respect to software development process. It is the same piece of code that evolves right from development phase to maturity phase. With all this, the very concept of "statistics" and correspondingly the various fundamental DfSS metrics like the Z-score, etc start to become fuzzy in case of software. Moreover "software does not wear out with time, it only becomes obsolete.

This leads to lot of questions....

- Can the time tested principles of DfSS be really applied in software development?
- If yes then how does it fit into the overall framework that is CMMI based?
- What kind of tools and statistics would really help?
- Are software requirements always digital or they can be converted into Continuous parameters?
- What does it mean to say a product / service process is six sigma?
- And so on …

This chapter is an attempt to answer these questions by sharing experiences of applying the DfSS methodology in real-life software development of an embedded consumer product. Although the chapter exemplifies the case study of a product, the same concepts could be easily applied to "service" also. This chapter is divided into 3 parts --

1. Part-1 is briefly introducing the fabric of the ***DfSS methodology*** – the DIDOVM phases of DfSS, the deliverables of these phases, spectrum of tools, the training required, the Green belt mechanism etc
2. Part-2 is a ***Case study*** of the application of the various DfSS concepts such as Voice of customer (VOC), Critical to Quality parameters (CTQs), Failure modes and Effects Analysis (FMEA) in the software development life cycle of an embedded product - the DVD-Hard disk combi recorder. Here it gives few examples of starting from Voice of customer, translating them into CTQs, quantifying the various non-functional elements such as Usability, Reliability and mapping to the Voice of customer, the way it was designed in the software, the results seen
3. Part-3 of the chapter revolves around some ***challenges of DfSS in software*** such as mapping into CMMI, elements to be careful about software FMEA and use of statistics in context of software.

Most of the contents of this chapter is taken from the author's published paper at the 1st India Software Engineering conference (Ajit Shenvi, 2008) , "[Design for Six Sigma: Software Product Quality] © 2008 Association for Computing Machinery, Inc. Reprinted by permission. http://doi.acm.org/10.1145/1342211.1342231.

2. DfSS Methodology

2.1 The phases

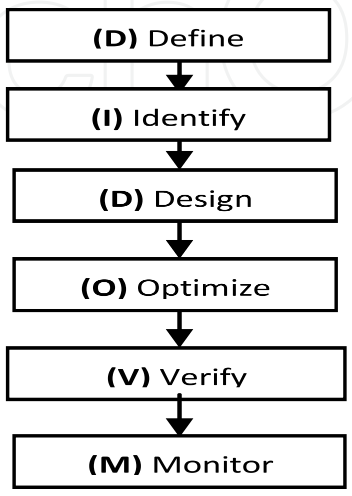


Fig. 1. The DfSS phases: DIDOVM

DfSS is a scientific methodology and as with any methodology, there are various phases and deliverables associtaed with these phases. Number of DfSS flavours are prevalent in the industry and DIDOVM is one of them as depicted in the Figure 1.

Define: In this phase, the problem area or opportunity is defined. The main deliverable of this phase is the *"project charter defined with management sign-off"*. This is a crucial step because lot of focussed effort and management support would be spent in the area scoped in the charter.
Identify: As with all flavours of Six sigma project, the DFSS projects start with identifying the Voice of Customer (VOC); This phase concentrates on translating the VOC i.e. the customer needs into "Critical to Quality" parameters. This VOC in customer language is translated into *"Critical to Quality" parameters (CTQs) with clear targets* and defined specification limits. Also the mechanism to measure these CTQs need to be articulated very clearly.
Design: This is the phase where the design takes place with CTQs as the basis. The *"best designs"* are chosen that will help meet the CTQs. The top level CTQs (Y) are broken down into lower level input factors (Xs) and *"transfer functions"* are developed. At this stage based on the transfer functions, it is possible to predict the capability of CTQ (Y) in terms of Z-score, given the variations of Xs. *Failure Mode & Effects Analysis* (FMEA) is also done in this phase to make the design robust and insensitive to noise factors.
Optimize: There could be same Xs impacting multiple CTQs in contradictory ways. Hence *trade-off decisions*, as well optimizing the design to meet the CTQs takes place in this phase. The tracking of actions from FMEAs for risk reduction is also done.
Verify: The optimized CTQs are now verified on the pilot products/systems. Here-in, when the products are beginning to be manufactured in large quantities, it is crucial to ensure that the CTQs are still within the specified limits on the produced items.
Monitor: Finally the performance of the CTQs is monitored in the field to ensure the real customer satisfaction.
In summary, the entire development, and testing effort of the product is centred on the ***"Voice of the customer" through the mechanism of "CTQs".***

2.2 The roles

A typical DfSS fabric is anchored on *"leadership commitment"*. As described earlier, DfSS is not only about a methodology but also a philosophy change; hence certain roles as described below are pivotal for the successful execution of DfSS projects
Champion/Sponsor: This person from senior management, champions the overall cause, provides budget, and removes bottlenecks if any. Champion has belief in the cause and stands-by the teams in the initial days when there is lot of dis-belief from the project teams and resistance to change in the system.
Master Black Belt (MBB): The MBB identifies, trains, coaches, and guides the Black belts towards the overall goal of the organization.
Black Belt (BB): This is a real *"change agent"* who is trained in DfSS methodology and gets the break-through improvement done. Since the BB is the real change agent, it naturally follows that this person should be sufficiently senior and influential in the team. The best candidate then would be the project architect or Function owner etc. The BB ensures that the VOC, CTQs and other techniques get deployed in the project, escalate issues if necessary,

keep the senior management and champion/sponsor updated on the progress. Training the Green belts is also responsibility of the BB.

Green Belt (GB): These are the *functional members* who deploy the DfSS tools and techniques in their respective functions. The essential difference between the BB and GB is the scope of deployment. A BB project has a very wide scope whereas GB project is on a smaller scope such as a functional area. The financial savings could also be a factor that distinguishes a BB project from a GB project. For e.g. a GB project could typically save 250 K Euros whereas a BB project would in order of 1 Million Euros.

2.3 The Training

The BB and GBs have to be trained on the DfSS concepts and tools. The training is not just classroom training, but a workshop type where the selected participants identify problem/opportunity areas to work on before they start on the training. These trainee BBs/GBs are then expected to deploy the applicable tools and techniques in their respective projects once they go back to their work. The results, issues faced, improvisations done, progress made are then presented by the BBs/GBs for review before they come for the next session. To maintain the seriousness of the trainings and the philosophy, this is an important pre-requisite condition for continuation of the BB/GB training. Around 4-6 weeks gap is required to be planned between the sessions for people to get adequate time to deploy the techniques, track progress and monitor results. The MBB has to work closely with the BB and the BB has to closely with the GBs during this time to coach, mentor, course-correct and steer the GBs towards the goal. The Figure 2 below depicts the GB training structure (Philips-SigMax DfSS Training material, 2005).

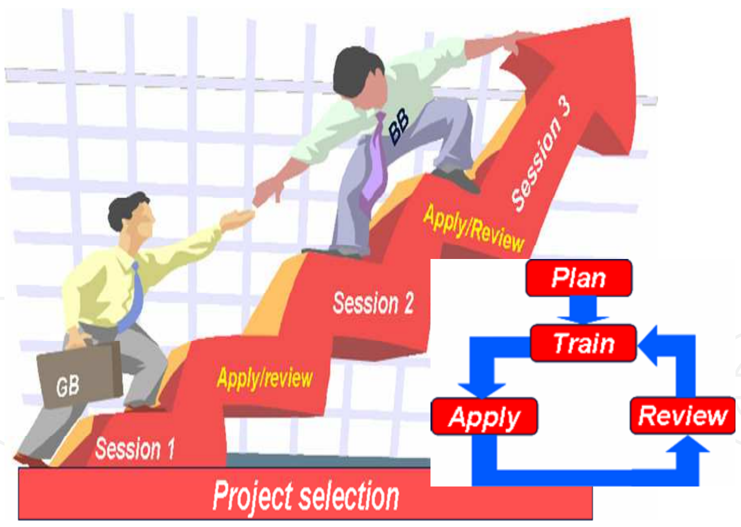


Fig. 2. The Green Belt Training Structure

For a GB training the typical training could be planned for 6.5 days and spread over duration of 3-4 months with the following sessions:

- 0.5 day brief on the need for change and DfSS overview
- 2 days for Define and Identify phase
- 2 days for Design phase
- 2 days for Optimize, Verify and Monitor phase.

All this investment in time, effort and cost makes the project selection very crucial. Every project cannot be a GB project. It has to be one where enough problems/opportunities exist to be able to classify it as a candidate for break-through improvement. Also the improvement has to be critical to business and sensed so by the sponsor. Hence it should be mandated that GBs prepare a project charter signed off by the sponsor before they could start on the GB training. The charter should have a clear problem/ opportunity statement, scope, targets, top level dates, resource requirement and operating principles of the GB project team. As a thumb rule, an improvement of *"atleast 50% improvement"* on chosen areas be demonstrated to be able classify it as a GB project. The successful completion of the BB/GB project with demonstrated results as mentioned in the charter would qualify for BB/GB certification.

2.4 Change Management

For successful deployment of any initiative it is important to identify the customers and the stakeholders and get them involved. The purpose of identifying and mapping stakeholders is important from *"Change Management"* perspective. Any break through initiative is bound to introduce number of changes and these changes are bound to meet with lots of resistance. So to manage this, the stakeholders especially the project team has to be sold on this idea as they are the ones finally implementing the changes. The mapping could be done into three categories:-

Blockers: who are against the idea and will try to resist the change either with valid or personal reasons

Floaters: who are on the fence and do not have particular opinions either ways

Movers: who are the supporters and are enthusiastic about the change

The Structure shown in Figure 3 can be used to plan and track the Stakeholder involvement from Change Management perspective.

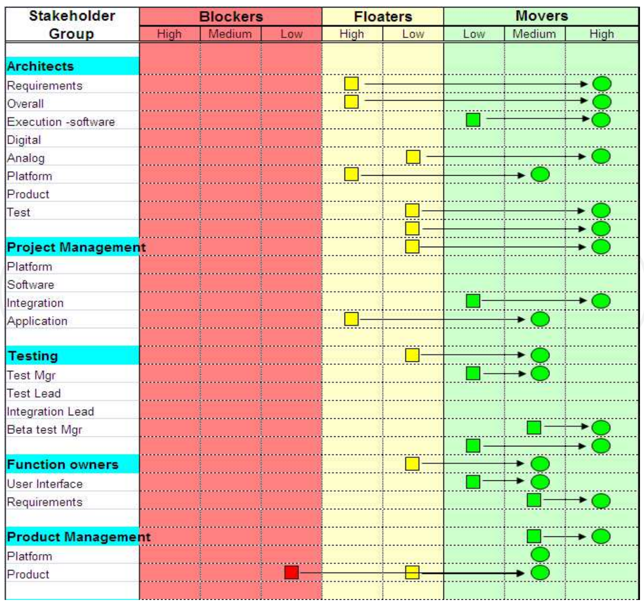


Fig. 3. The Change Management Structure

Movers can be used to convince Blockers about the need for change and get them on your side. Current state and the desired state for each of the stakeholders and actions to facilitate this movement need to be identified as depicted in Figure 3 above. Such actions need to be identified for each and every stakeholder including senior and middle management members and tracked on a periodic basis. The goal in doing this exercise is to ensure that adequate support and push is available from all sides to bring about break-through changes in the Way of Working.

3. Case Study

Philips Innovation Campus (PIC), Bangalore is a division of Philips Electronics India Limited, owned by Royal Philips Electronics N.V., The Netherlands. There are various groups in PIC, that develop embedded software including user interface for consumer electronics (CE) devices such as Televisions, DVD players & recorders, Juke boxes, Set top boxes etc. These CE products like any other go through the typical product life cycle of inception, growth, maturity and decline. This transition is very rapid, due to which the industry is extremely competitive. The margins on the product are very small and it is only through volume sales that the CE companies are able to make any profit. Moreover the base product and features of almost all the manufacturers is essentially the same. What differentiates them then is some unique delighters, intuitive user interface, responsiveness i.e. often the non-functional requirements. Software is at the heart of such differentiation. On the flip side since software is such an important element of the embedded product, it is also cause of failures, user dissatisfaction (perceived as well as real).
One such product range that had just entered from inception phase to growth phase is the DVD-Hard disk recorder. This product with all its combinations of use cases makes it a very complex product. Correspondingly it has a potential of having field issues and user complaints leading to Non-Quality Costs that would ultimately eat into the profit margins of the current business. Loss of brand image arising out of this would also affect the future business as well. Hence it was decided to use DfSS techniques as a focussed approach in the development to ensure good software quality product.

3.1 The Product

Fig. 4. The Product: DVD-Hard disk recorder

This is a product that records and plays DVD, VCD and many other formats. It has an inbuilt hard disk that can store pictures, video, audio etc. Due to the presence of the hard disk, it is possible to a pause the live-TV and resumes it later from the point it was paused. The product is packed with many (more than 50) features. All these features and their associated use-cases with some of them in parallel make this a very complicated product. Also because of the complexity, the intuitiveness of user-interface assumes enormous importance to address the usability of the product. For convenience sake, let us call this product XYZ.

3.2 Customer Identification

The DFSS methodology is strongly anchored on listening to the "Voice of customer (VOC)" and ensuring that this voice is satisfied throughout the development life cycle. For this DVD product, the external customers are very clear and they are the end users of this consumer product and the retailers/dealers who stock them. So when VOC is being referred to in this context, it is this group that we refer to.

At the same time, being a development community it is also imperative to understand that there are other set of internal customers as well, whose voice also needs to be heard. They are the sales group who face the end users on a day-to-day basis, product management who decide what features get into which products and factory where the products actually get produced. For example - factory "VOC" could be to make it simple to produce the sets and a related CTQ could be number of times the hard disk needs to be formatted on the production line.

3.3 Stakeholder Analysis

Stakeholders also needed to be identified as they are directly linked to the success/failure of our DfSS project with the project team being the most important one. Therefore, as indicated in section 2.4, it is advisable not only to identify the stakeholders but also to map them into various categories and have actions to facilitate their movement from blockers to enthusiastic supporters. These are some of the "change management" techniques that could be used. The customers and stakeholders identified for this project are represented below in the Figure 5.

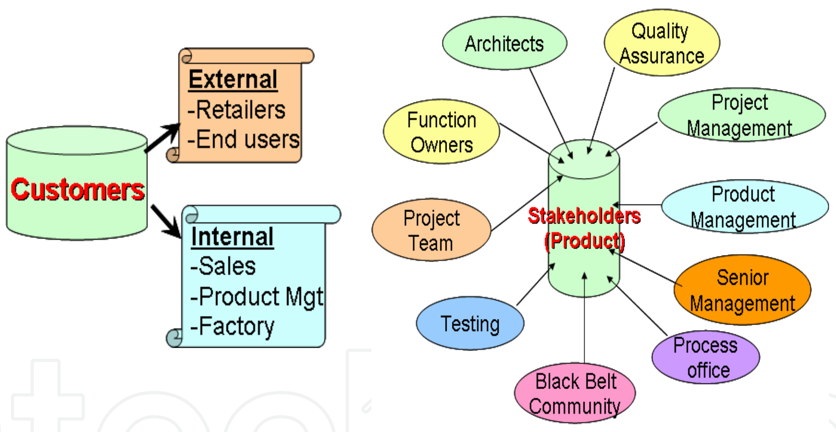


Fig. 5. Customers and Stakeholders

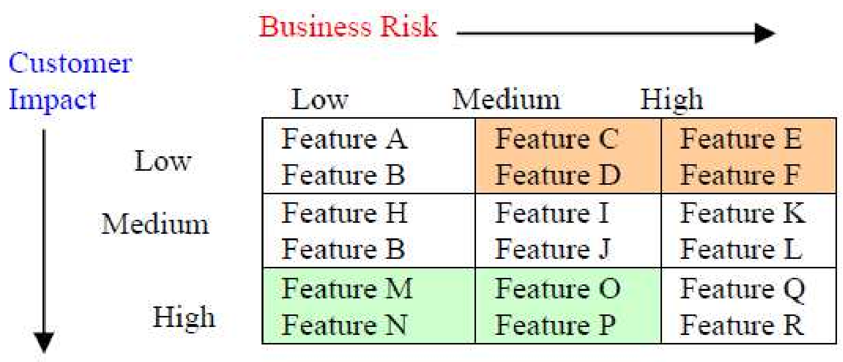
3.4 Define and Identify phase

As mentioned in section 3.1, this product range due to its complexity and large software base has a potential of high Non Quality costs due to field issues and usability calls. The DfSS Black belt project therefore had a charter of preventing this. In other words, both limiting the Non-Quality costs due to quality issues as well as usability enhancement was the target of this project.

Having defined the charter the next step was to identify the ***Voice of the Customer*** (VOC) for this product. The various techniques to get this VOC are focus group interviews with consumers/dealers, surveys, benchmarking etc. As a development community, this activity had already been done by the market intelligence and product management community. The VOC information was available in the form of consumer requirements specifications (CRS), and Product Value Proposition House. These were validated and assumptions challenged using some DfSS techniques such as Risk Benefit matrix, Kano analysis and mechanism of identifying CTQs. These are the tools that can be used in Requirements management phase of the software development to enhance requirements analysis and prioritization.

3.4.1 Risk Benefit Matrix

Products are often packed with lots of features making it complicated to use. Typically 80% of users do not use more than 30% of the features.
Risk-Benefit is a simple matrix that can challenge each and every requirement or feature and its need. The matrix has 2 axes: one axis represents the ***"customer impact"*** attribute and the other axis is the ***"business/development risk"*** attribute. Each of these attributes has 3 levels: high, medium and low. The Figure 6 below shows such a matrix template.


Business Risk →

Customer Impact ↓

	Low	Medium	High
Low	Feature A Feature B	Feature C Feature D	Feature E Feature F
Medium	Feature H Feature B	Feature I Feature J	Feature K Feature L
High	Feature M Feature N	Feature O Feature P	Feature Q Feature R

Fig. 6. The Risk-Benefit Matrix

All the features identified for the product in the Consumer Requirements Specifications are then filled in the appropriate cell of this matrix. For maximum benefit, this should be a joint exercise between the development community and the product management. All those features in the low customer impact and high/medium business risk could simply be removed.
For the XYZ product, a similar exercise was done and the product management was challenged on the position of each feature in the above matrix. Some of the features in *"High risk-Low impact"* zone got dropped in the process making the product simpler than what it was already conceived, without even getting even a single step into development.
Previously also similar exercises took place but they were adhoc and sometimes too late in the development cycle after lot of effort were already spent in designing and coding the high risk features. The risk benefit gave a very structured mechanism to prioritize the features. Another added benefit was that, it helped improve the communication between the product management and the development community by giving an appropriate platform to debate and discuss. Development community could now think from the problem domain perspective and product management from the solution domain.
Having done the filtering of features, Kano analysis can then be done to prioritize the requirements further.

3.4.2 Kano Analysis

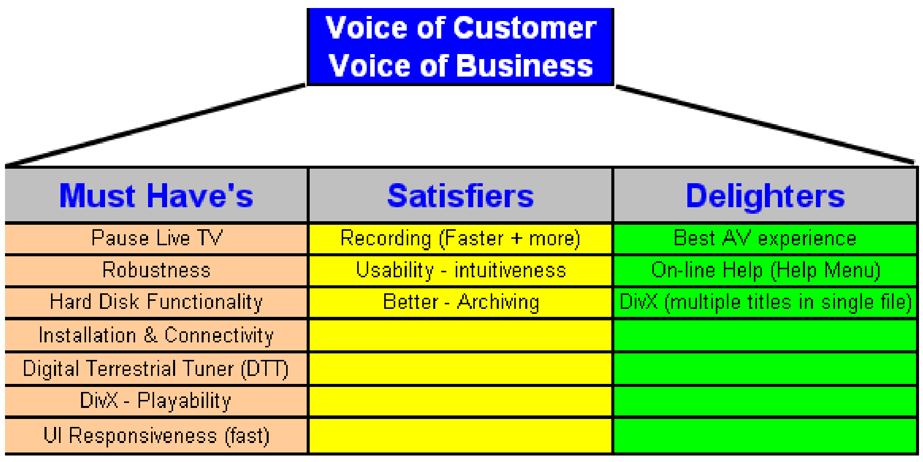
Kano Analysis is one of the techniques that classifies the features/requirements into 3 categories namely-

Must Haves: These are the basic needs that the customers expect in a product/service and therefore take it for granted. Its absence would cause extreme dissatisfaction, however more of it does not guarantee increased satisfaction.

Satisfiers: These are the specifications that increase customer satisfaction as more and more is added. For example-higher speed has better satisfaction.

Delighters: These are the real value add attributes that act as differentiators. The customer is not expecting them but their presence gives a "WOW" feeling.

Part of Kano analysis done for this XYZ product can be seen in Figure 7. This prioritization done through Kano analysis, then helps to allocate effort and bandwidth when identifying CTQs and focusing development.



Voice of Customer
Voice of Business

Must Have's	Satisfiers	Delighters
Pause Live TV	Recording (Faster + more)	Best AV experience
Robustness	Usability - intuitiveness	On-line Help (Help Menu)
Hard Disk Functionality	Better - Archiving	DivX (multiple titles in single file)
Installation & Connectivity		
Digital Terrestrial Tuner (DTT)		
DivX - Playability		
UI Responsiveness (fast)		

Fig. 7. The Kano Analysis for the XYZ Product

3.4.3 The CTQs (Critical To Quality)

CTQs are the Critical to Quality parameters as called in the DfSS jargon. Basically CTQs are those parameters that are directly linked to Voice of customer. CTQs are of 3 types:

Continuous: These are the quantitative ones that can be measured using gauges and instruments.

Discrete: These are the ones that can be classified into Pass/Fail, Yes/No category.

Critical factors: These are CTQs that are either present or absent. For example - Wi-Fi compliance is a critical factor. Either all sets are compliant or not.

The real crux of the DfSS BB/GB project lies in identifying the right CTQs that are mapped to the VOC which are the needs of the customer. Effort available is limited and if spent on unwanted CTQs is actually wasted. So once the VOC was identified through CRS, Value proposition, Risk benefit and Kano, the challenge then was to identify the right software CTQs for this product XYZ.

The complete landscape that could lead us to the right CTQs was analyzed. The CRS and value proposition was an obvious starting point. The VOC expressed as field complaints and feedback of previous products, both from external customers (consumers) and internal

customers (sales, factory) turned out be another valuable input in determining these CTQs. An often after-thought element ***"non-functional requirements"*** such as responsiveness etc. was another dimension to look at. Thinking about VOC made us also look at competitor products for determining CTQs via benchmarking. Last but not the least each of us in the development community was also a consumer and wearing an end-user hat changed our perspective when we were trying to identify the CTQs.
All these inputs as shown in Figure 8 were used in iterations along with product management to churn out the CTQs for this product XYZ.

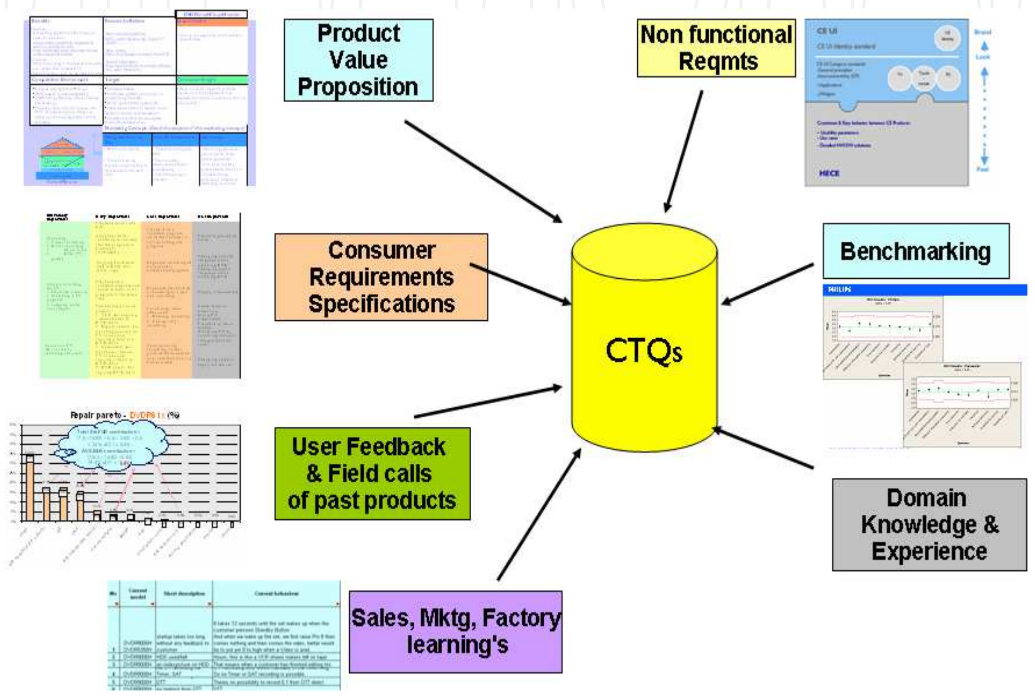

Product Value Proposition
Non functional Reqmts
Consumer Requirements Specifications
Benchmarking
CTQs
User Feedback & Field calls of past products
Domain Knowledge & Experience
Sales, Mktg, Factory learning's
Repair pareto

Fig. 8. Inputs for CTQ Identification

For each of the continuous and discrete CTQs, the measurement method, target and specification limits must be clearly identified. For the critical factor CTQs, since quantitative measurements or classification is not available, the verification criteria, method and risk should be elaborated instead.
This Identify stage turned out to be one of the most difficult phase when it comes to software development. Everything is digital in software – it works or does not, so all the CTQs we identified from the VOC started becoming *"Critical factors"*. Identifying test cases as verification criteria was what we did previously also in the software development life-cycle.
So we started challenging them by further breaking down to lower levels and identifying some numbers and measurements with it. The 2 examples below on DivX and USB elaborate this approach.

3.4.3.1 Feature DivX

DivX was one of the new features for the product and an important VOC. But as a CTQ, its value in guiding software development was not high. So we started asking what in DivX is important to the user, in a brainstorming exercise with product management. After some deliberation we came up with 3 main things:
a) DivX playability – a measure of how well our product can play all flavours of DivX content available in the market.
b) DivX playback time – How fast can our device respond and start playing once the user presses "Play" and other related operations.
c) DivX certification – this is Voice of business and not really Voice of consumer. To put a DivX logo on the product we need to get the product certified from the standardizing body. Each round of certification costs a lot of money. Moreover time-slots have to be booked in advance with the standardizing body. So an unsuccessful round is not only an immediate financial loss but also an opportunity lost due to loss of time.

The next step was to set targets for each of these CTQs so that the architecture/design and implementation can be guided by the same.
For Playback-time the case was simple. There was already research papers available on what are the typical human irritation thresholds when they interact with devices. A compilation of these for different use-cases was already available, so we decided to use the same. So DivX playback time became our continuous CTQ and we could easily give a target number with upper limit and measurement method to it.
DivX certification on the other hand was a Critical factor but we still wanted to treat it as a measurable CTQ. So we set a target to achieve DivX certification on the first try itself as that would save us lot of costs of re-certification trials.
DivX playability was an interesting case. An end user would typically want everything that is called as DivX content to play on his device. This is a free content available on the internet and it is humanly impossible to test all. To add to the problems, users can also create text files and associate with a DivX content as "external subtitles". Defining a measurement mechanism for this CTQ was becoming very tricky and setting target even trickier. So we again had a brainstorming with product management and development team, searched the internet for all patterns of DivX content available, and created a repository of some 500 audio-video files. This repository had the complete spectrum of all possible combinations of DivX content from best case to worst case and would address at least 90% of use cases. The success criterion then was to play as many of these 500 files and the target defined was at least 90% of them should play successfully. So DivX playability then became our discrete CTQ with a measurement method of verifying the % of files the product XYZ was able to play and the target was 90% with a lower limit of 80%.
All the exceptional use cases identified in the meeting were then used for conducting the Failure Mode and Analysis (FMEA) to ensure robustness and graceful exits in case of feature abuse.

3.4.3.2 Feature USB

USB (Universal Standard Bus) was another very important feature from VOC as users use USB as a medium to copy/transfer content. USB 2.0 is a standard which we wanted to comply with.

So here again it was very easy to consider this as a critical factor and go ahead with the development. But just as in DivX case, we wanted to dwell deeper to understand what could be the CTQs. From DivX experience, few straightforward ones we could come up with were USB certification, USB notification time, content feedback time etc. The real challenge was to define how an end user would consider USB feature as successful and one of the ways is if the user is able to copy/playback content from the USB device on our product XYZ. In other words the CTQ parameter we identified is ***"Interoperability "***. This is easier said than done – just like DivX, there are at least thousands of USB manufactures and a variety of USB devices ranging from simple memory sticks to ipods, juke boxes to digital cameras. It is again humanly possible to verify the compatibility with each and every type. To add to the complications there are some device manufactures who sell USB those are not compatible with the USB 2.0 standard. Furthermore the market is flooded with devices that are USB 1.0, USB 2.0, High speed etc and one doesn't expect a user to check what version his USB is before plugging into our device. So to define this CTQ was really a challenge. So we decided to go back to basics. What is the single most important use case that any user would use the USB port of a DVD recorder extensively? In other words what is the real voice of consumer? The answer was obvious; to connect digital cameras to be able to view and store JPEG images. That limited the sample space of USB devices primarily to digital cameras. It reduced the complexity of the problem drastically but still did not solve it completely as types and makes of digital cameras itself easily runs into thousands. As a next step we scanned the market space, where this product XYZ was supposed to be launched to see the most popular digital cameras currently available and those that are in the pipeline to be launched in the timeframe of our product launch. We zeroed down on some 5-6 different brands with 6-7 different models in each brand. This list was augmented with some popular make memory sticks and juke boxes. The CTQ definition then was percentage of devices that could successfully interoperate with our product and target was at least 90%. The Figure 9 below describes the process pictorially.

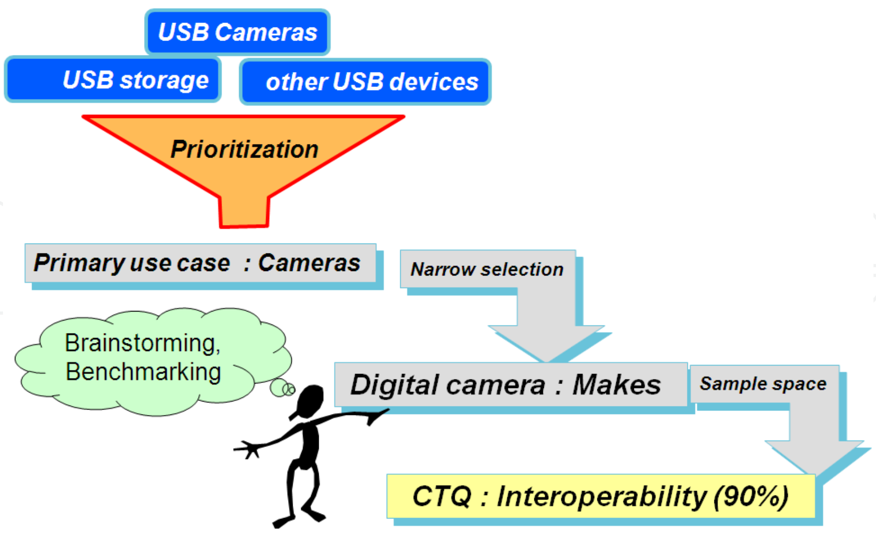


Fig. 9. USB Feature – Interoperability CTQ

As with DivX, all the exceptional conditions identified during these discussions were taken for doing the FMEA.

Similar approach was followed for all other features (VOC). At the end of the exercise we had list of all the CTQs mapped to VOC.
Continuous CTQs: mostly performance/responsiveness related e.g. startup time, content feedback time, USB notification time etc
Discrete CTQs: mostly on the "ilities" such as playability, interoperability, reliability, usability etc. Reliability and Usability being generic are further elaborated later in the chapter in the next section.
Critical factors: mostly compliance related such DivX certification, USB certification, FNAC 4 stars etc
For both Continuous and discrete CTQs clear targets and specification limits were identified as success factors. For critical factors only verification criteria and method were elaborated.

3.5 Design and Optimize phase

Once the CTQs are identified the next step is to guide the development cycle around these CTQs. Design phase has two primary goals – one is to select best design and second is to decompose the top level CTQ (Y) into its lower level factors (Xs) called as CTQ flow-down. These maps to architecture, design, implementation and integration phase of a typical software development life-cycle.
Tools like *Pugh matrix* can be used to select the best design form the alternative choices using CTQs as selection drivers. Appropriate weightage based on priority decided from Kano is given to the CTQs and all design choices are rated on their capabilities to meet the CTQs. Best design is then selected appropriately that has the highest score and minimum negatives.
Another aspect of design phase is to break down the CTQs (Y) into lower level inputs (Xs) and a make a transfer function. The transfer function basically helps to identify the *strength of correlation* between the inputs (Xs) and output (Y) so that we know where to spend the effort. Various statistical techniques as shown in Figure 10 below are available to make this transfer function such as Regression analysis if past data is available, Design of experiments (DOE) if past data is not available and of course domain knowledge of experts.

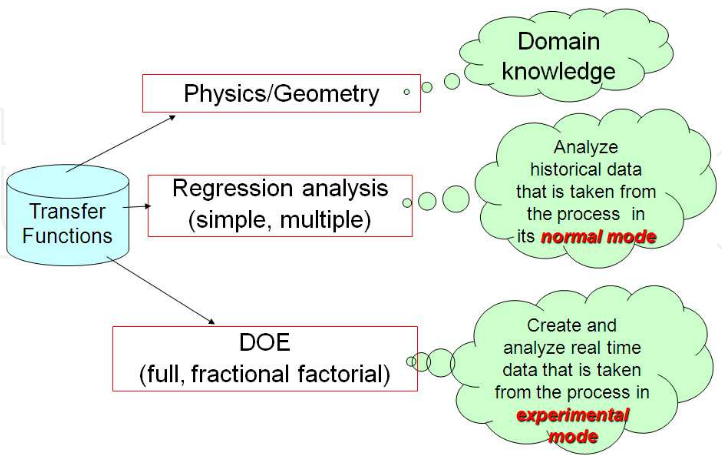

Domain knowledge
Physics/Geometry
Transfer Functions
Regression analysis (simple, multiple)
Analyze historical data that is taken from the process in its normal mode
DOE (full, fractional factorial)
Create and analyze real time data that is taken from the process in experimental mode

Fig. 10. Transfer Function Techniques

For many cases in software, the actual transfer function may not be necessary as the number of inputs and their combinations would be very high. What is more important to know is which are the input parameters (Xs) that can be controlled to meet the CTQs, which of the inputs are constants/fixed and which of them are noise parameters. An example of such a block for DivX is illustrated in the Figure 11 below.

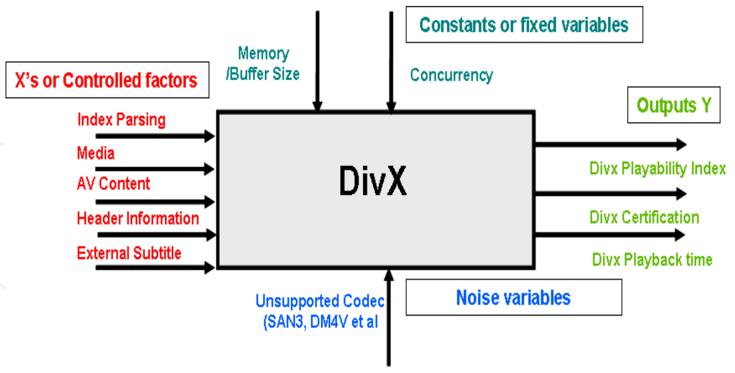


Fig. 11. DivX Feature : Transfer function

For performance CTQs, the actual transfer functions really make sense as they are linear in nature. One can easily decide from the values itself those Xs that need to be changed and by how much. For e.g.
Start-up time (Y) = drive initialization (X1) + software initialization (X2) + diagnostic check time (X3)
For other CTQs, main effects plot and interaction plots are sufficient enough to know the inputs to tweak. These plots can be made in any statistical tools such as Minitab (www.minitab.com) either from Regression analysis or conducting few Design Of Experiments (DOEs).
Main effects plot give an indication of the impact each of the Xs have on Y. For example- the Figure 12 shows the variation in USB copy speed CTQ (Y) for the variation in each of the Xs (buffer, device speed etc).

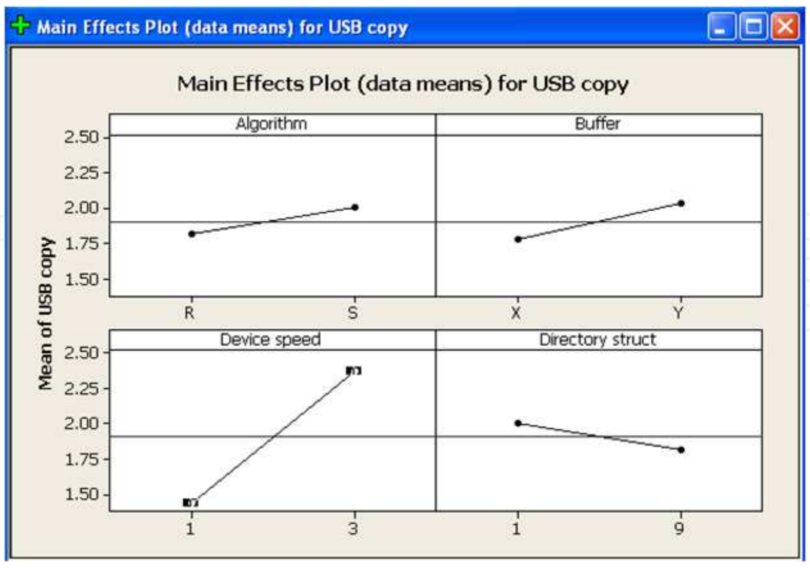


Fig. 12. USB copy: Main Effects Plot

The Interaction plot on the other hand will show the impact of the interactions of the Xs on the CTQ Y as shown in Figure 13.

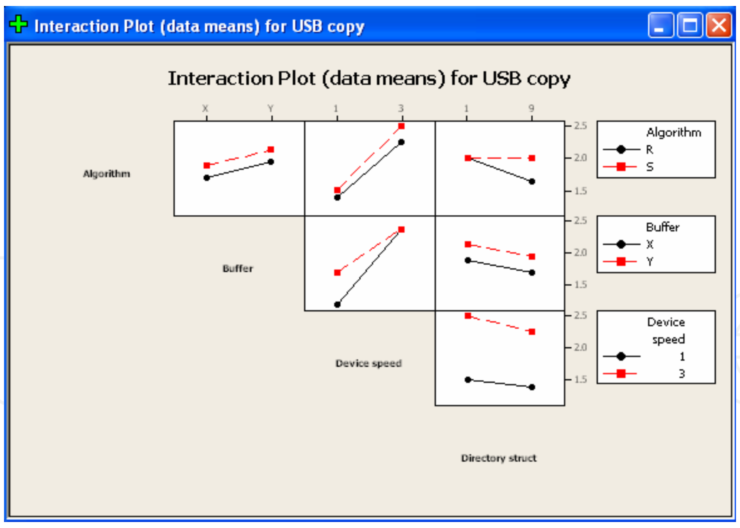


Fig. 13. USB copy: Interaction Plot

Knowledge on both the main effects and interactions helps in optimizing the design and trade-off decisions. Many a time in software, most of modules are interrelated, and a single X might impact multiple CTQs in opposite ways i.e. for meeting CTQ Y1, an increase in X may be necessary and for CTQ Y2 a decrease would be required. In such a case interaction plots can help to make trade-off decisions by masking input X with another interacting input X2.
Another important aspect of the Design and Optimize phase is the FMEA and mistake proofing i.e. to make designs resilient to failures or mask the users from making mistakes itself. A Standard FMEA template was tailored by mapping the definitions and scale of the "Severity", "Occurrence" and "Detection" parameters to software context. For example we already had severity definitions defined for classifying software bugs in our process framework. We used the same for FMEA severity attribute. Occurrence attribute was simplified to mean 1 in 3 chances as a high value for example and so on and so forth. Guidelines for detection were also accordingly simplified. More details on pitfalls and learning's of applying FMEA in software are discussed in detail in section 4.2.
The Risk priority number (RPN) from this FMEA after implementing the actions was tracked on a periodic basis. For software, the demarcation between the Design and Optimize is very thin as the same code base is used iteratively.

3.5.1 Software Reliability

Software by itself does not have a "Constant Failure rate"; hence defining MTBF (Mean Time Between Failure) for software alone starts becoming fuzzy. The typical bath-tub curve for software looks something like shown in Figure 14 (Jiantao Pan, 1999).

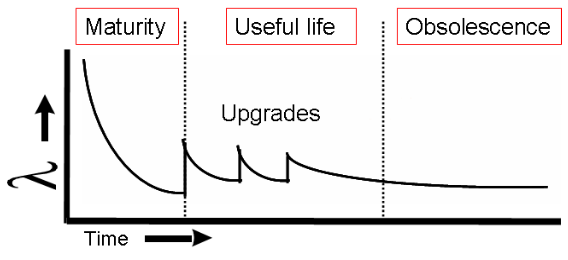


Fig. 14. Typical Bath-Tub curve for software (where λ is the failure rate).

One way to determine software reliability would be in terms of its robustness. We tried to define Robustness as a CTQ for the product XYZ. This was again turning out to be "critical factor". So we defined the CTQ (Y) in terms of "Number of Hangs/crashes" in normal use-case scenarios as well as stressed situations and target was set at 0.
The lower level factors (X's) affecting the CTQ robustness was then identified as:

- Null pointers
- Memory leaks
- CPU loading
- Exceptions/Error handling
- Coding errors

Robustness = f (Null pointers, Mem leaks, CPU load, Exceptions, Coding errors)
The exact transfer function could have been found out by doing a number of "Design of experiments". This would have consumed a lot of effort and would have turned out to be an academic exercise. The purpose of finding transfer function is really to find out which of the Xs are really correlating heavily with Y so that optimizing them would yield maximum benefits. In case of robustness however each of these Xs are equally important and all of them need to be optimized. So we decided to take actions on each of these input parameters in different ways.
Some of these actions are as follows:-

- A small tool was developed to find null pointers if any in the code stack. Most of these were then eliminated.
- Stringent limits set for memory allocation of subsystems. This was tracked at every release to ensure that all subsystems are within budget and that there is no overlap of memory space. (Subsystems come from different project teams and external suppliers).
- From programming experience, it has been found that CPU load > 65% makes the embedded system unstable and unpredictable. Hence different combinations of concurrent scenario's (stressed conditions) were chosen and CPU load tracked using a tool (proprietary) for every release as shown Figure 15. Any increase in the load led to code optimisation

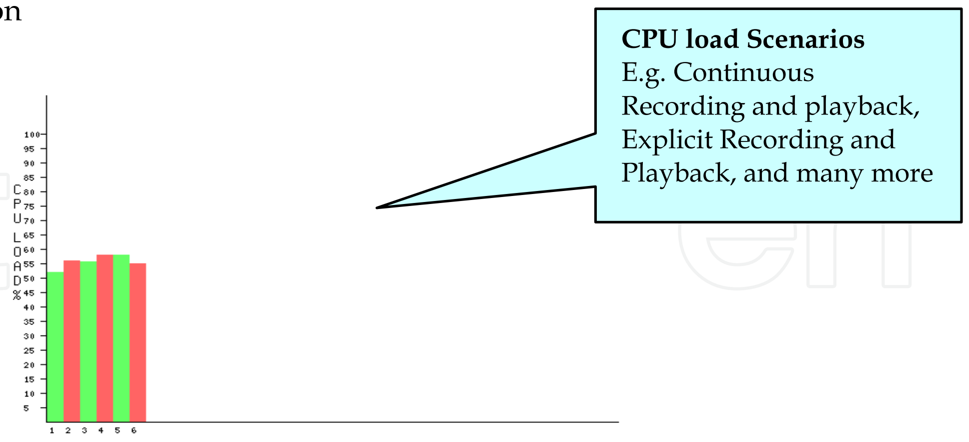


Fig. 15. CPU load tracking: Various scenarios

- FMEA was done to identify failure modes leading to exceptional conditions for new features. Graceful exits and error recovery mechanisms were implemented. For example- exit with an error message rather than be in a continuous loop when non-standard USB device is connected to the recorder.

- Static analyzer tools such QAC was run and the target set was 85% code coverage. Errors and warnings were closed. For each code module, the QAC coverage was measured and improved before "build generation".
- An operational profile of typical user scenarios was created and the software run on the product with different profiles continuously for 4-days at elevated temperatures (Duration test). The results were verified every alternate week.
- Finally the overall CTQ of robustness – hangs and crashes were measured on weekly builds to verify the results in normal as well as stressed conditions. Refer Figure 16 below

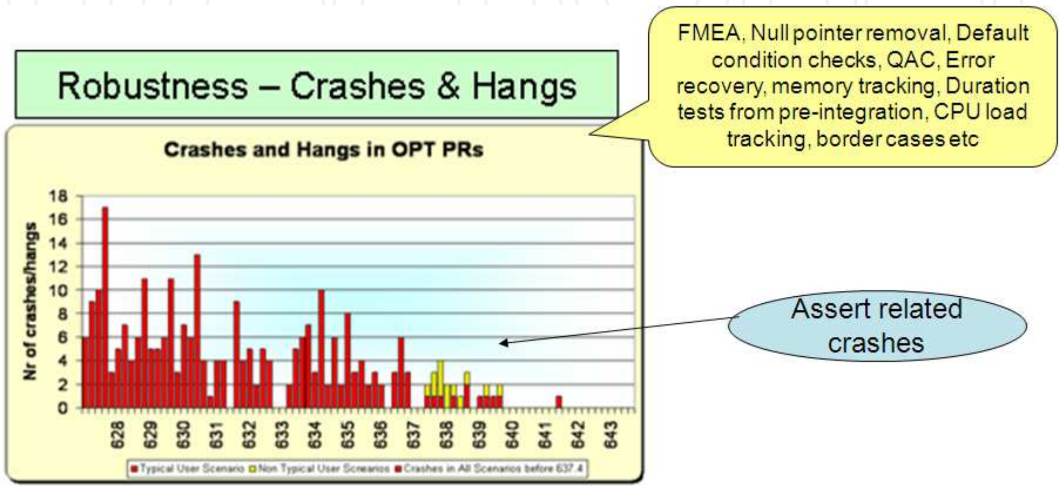


Fig. 16. Robustness CTQ Tracking

The stability of the set was well appreciated by sales and Field test personnel stating that it was indeed much better than previous sets. This was an early leading indicator, and the first feedback results coming from user tests, dealer meetings, field tests and market reviews were definitely indicating the improved robustness before the commercial release of the product.

3.5.2 Usability

Usability is very subjective parameter to measure and very easily starts becoming a critical factor. Living with the "Sense and Simplicity" theme of Philips, it was important that we treated it as a continuous CTQ and spend enough time to really quantify it.
A small questionnaire was prepared based on few critical features and weightage was assigned to them. A consumer experience test was conducted with a prototype version of product. Users with different age groups, nationality, gender, educational background were selected to run the user tests. These tests were conducted in home-like environment set-up as shown in Figure 17, so that the actual user behaviour could be observed.

Fig. 17. Consumer Experience Test Set-up

The ordinal data of user satisfaction was then converted into a measurable CTQ based on the weightage and the user score. This CTQ we called as ***"Usability Index"***. The Xs impacting this case were the factors such as Age, Gender etc. The interaction plot shown in the Figure 18 below helped to figure out and correct a lot of issues at a design stage itself.

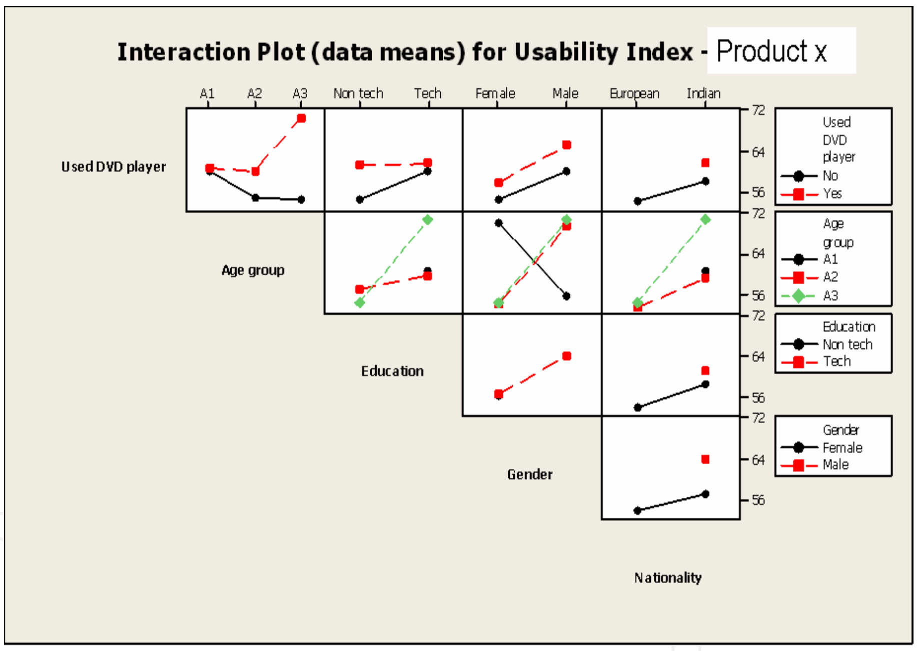


Fig. 18. Interaction Plot for Usability

For example - if the product is targeted towards European housewives of age group 25 – 40 who had no technical background and who had not used a DVD player before, we could already know what the usability issues could crop up in the field. Such feedbacks in different combinations were used to improve these issues at the design stage itself based on the target consumer group for this product XYZ.

3.6 Verify

This phase of DFSS maps to the testing phase of software development life cycle. Once the system is integrated, it is essential to verify that the CTQs that were achieved in the development environment during the design & optimize phase are still met on "production builds".

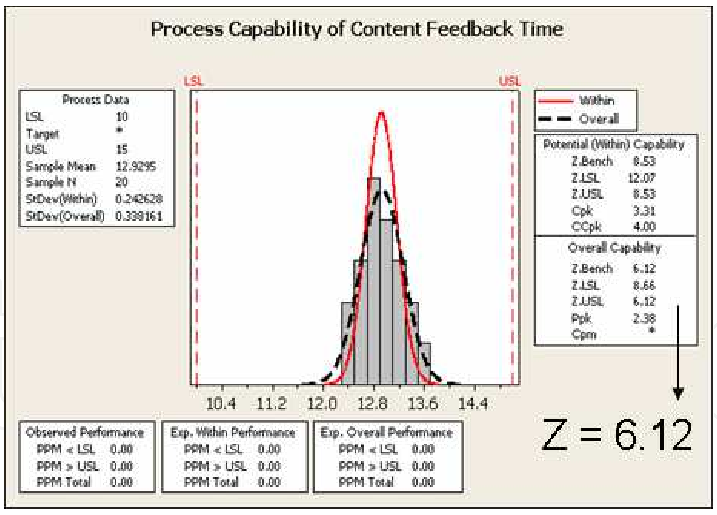
Here is where statistical tests and Z-scores could be used to verify that the CTQs have indeed been met on production sets. For all the performance (responsiveness) CTQs, we used statistical tests and Z-scores to verify the process capability. We configured 5 different sets with 2-3 operators measuring each of the performance CTQs for every release build. That makes it 10-15 samples for each CTQ per build. To avoid measurement errors simple set of clear instructions were made and explained to the testers doing the measurements. Similar stop watches from the same manufacturer were used to avoid any device related measurement errors creeping into the system. A few trials were done and observed before doing the actual measurements. In DfSS terms, this is referred to as ***"Gage Repeatability and Reproducibility"*** analysis.

When we talk about six-sigma, we are not only interested in mean but also the variation (standard deviation). Z-score is a measure that is a reflection of this variation.

$Z = \text{Abs}(SL - \mu) / \sigma$; where SL is the specification limit, μ is the mean and σ is the standard deviation.

By definition, anything outside specification limits is considered defect. From that perspective Z-score is also an indication of DPMO (defects per million opportunities). A 6 sigma process (i.e. Z = 6) has a DPMO of just 3.4

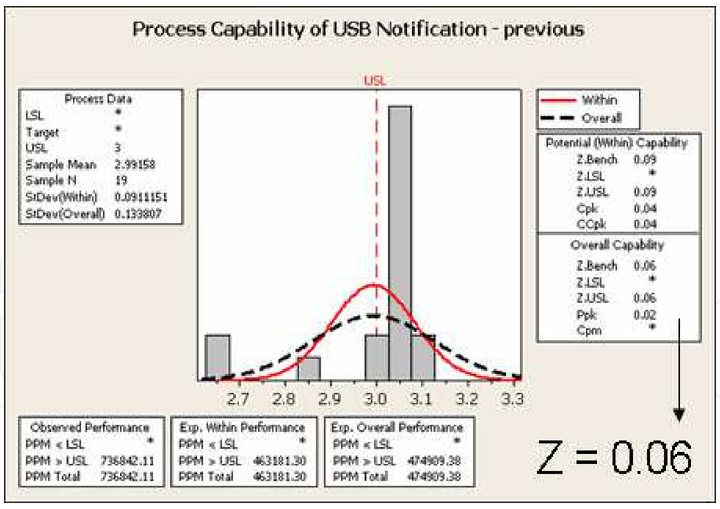
Figure 19 shows the process capability (Z-score) for one such CTQ – "Content feedback time"



```
Process Data
LSL              10
Target           *
USL              15
Sample Mean      12.9295
Sample N         20
StDev(Within)    0.242628
StDev(Overall)   0.338161

Potential (Within) Capability
Z.Bench   8.53
Z.LSL     12.07
Z.USL     8.53
Cpk       3.31
CCpk      4.00

Overall Capability
Z.Bench   6.12
Z.LSL     8.66
Z.USL     6.12
Ppk       2.38
Cpm       *

Observed Performance
PPM < LSL    0.00
PPM > USL    0.00
PPM Total    0.00

Exp. Within Performance
PPM < LSL    0.00
PPM > USL    0.00
PPM Total    0.00

Exp. Overall Performance
PPM < LSL    0.00
PPM > USL    0.00
PPM Total    0.00
```

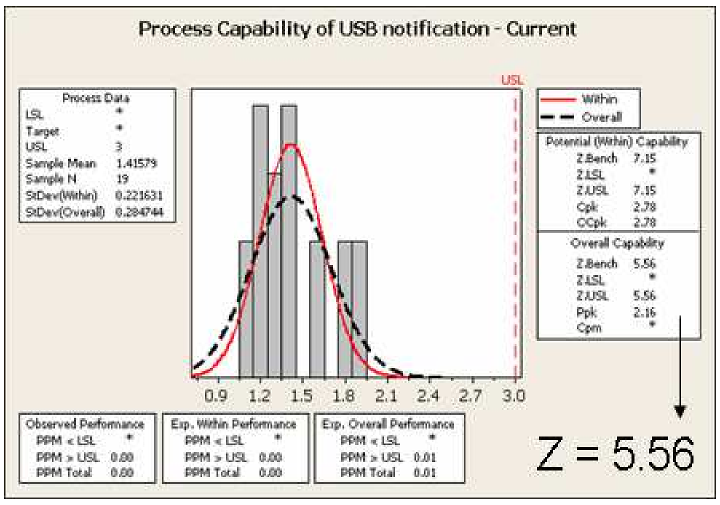
Fig. 19. Capability: Content Feedback Time

We also tried to see the change in Z-scores from initial condition to that after improvement. The 2 figures - Figure 20 and 21 show the previous and the current Z-scores for another CTQ – "USB notification time".



```
Process Data
LSL             *
Target          *
USL             3
Sample Mean     2.99158
Sample N        19
StDev(Within)   0.0911151
StDev(Overall)  0.133807

Potential (Within) Capability
Z.Bench   0.09
Z.LSL     *
Z.USL     0.09
Cpk       0.04
CCpk      0.04
Overall Capability
Z.Bench   0.06
Z.LSL     *
Z.USL     0.06
Ppk       0.02
Cpm       *

Observed Performance      Exp. Within Performance   Exp. Overall Performance
PPM < LSL   *             PPM < LSL   *             PPM < LSL   *
PPM > USL   736842.11     PPM > USL   463181.30     PPM > USL   474909.38
PPM Total   736842.11     PPM Total   463181.30     PPM Total   474909.38
```

Fig. 20. USB Notification Time: Before



```
Process Data
LSL             *
Target          *
USL             3
Sample Mean     1.41579
Sample N        19
StDev(Within)   0.221631
StDev(Overall)  0.284744

Potential (Within) Capability
Z.Bench   7.15
Z.LSL     *
Z.USL     7.15
Cpk       2.78
CCpk      2.78
Overall Capability
Z.Bench   5.56
Z.LSL     *
Z.USL     5.56
Ppk       2.16
Cpm       *

Observed Performance      Exp. Within Performance   Exp. Overall Performance
PPM < LSL   *             PPM < LSL   *             PPM < LSL   *
PPM > USL   0.00          PPM > USL   0.00          PPM > USL   0.01
PPM Total   0.00          PPM Total   0.00          PPM Total   0.01
```

Fig. 21. USB Notification: After

One catch here is that it is quite likely that in software when we measure different samples we would get almost the same readings as software by itself does not have inherent variation. So we would land up with very high Z-scores and things would look very rosy. Hence when we are verifying the CTQs it should be done on an integrated system along with hardware (i.e. on a product) and not for only software. That would give a realistic picture as a good software code should also be able to mask the small variations arising out of hardware.

In addition to Z-scores, a set of statistical tests can be used to ascertain that there is indeed an improvement that is statistically significant and not just due to random variation. There are number of tests that we used such as
1-sample T test to compare mean of a sample against a target,
2-sample T test to compare the means of 2 samples
F-test to compare the variation of 2 samples etc
In all these tests, the null hypothesis we used was "status quo" i.e. no change.
At the end of verify phase we had met most of the CTQs that we started with. For example: DivX playability > 90%, USB interoperability – 87%.

3.7 Monitor

This is the phase where the "proof of the pudding" is really available. The customers and stakeholder satisfaction is monitored in this phase. If our CTQs are right and if we meet them then customers should be satisfied, correspondingly the field calls and customer complaints should reduce leading to lower non-quality costs and improved usability scores.
Some proof points indicating the success of the journey:-

1) All the product certifications such as DivX certification etc. went off successfully on the first trial itself saving lost of recertification costs
2) Referring to section 3.2, the first set of customers were the Sales personnel and the dealers.
   Some Comments from sales personnel – *"Compared to previous sets, the stability of XYZ is better"*.
   Some reactions from dealer meetings – *"Today we have many customers visiting us to show our new products. Just wanted to let you know that we received great feedback on the new XYZ. Some couldn't even believe it! Sense and Simplicity brought to life....... Congratulations on a job well done!"*
   All these lead us to believe firmly that market response also would be very positive
3) After the release, the market response was closely monitored in terms of field complaints as well as user feedbacks in various Audio-Video forums such as web pages
   a. In most of the popular forums, the user views were extremely positive. The product XYZ always scored 8 or above (on a 10-point scale) on a comparative benchmark in the popular web forums
   b. The software related field calls were less than 2.5% translating to potential savings of 2 Million Euros

3.8 Conclusion

The case study described above serves as a good evidence to ascertain that the DfSS approach and tools can be used effectively also in a software development environment. The learning's are summarized in the following two sections as "Gains" at one end of spectrum and "Points to ponder" at the other end.

3.8.1 Gains

<u>Software engineering (addressing the pain areas)</u>

- Better Requirements Management in terms of specifications, prioritization, and traceability. Focus on non-functional aspects of usability, interoperability, performance, robustness
- Importance of Execution architecture (State-transitions, CPU loading, Memory)
- Increase in Design effort, upfront design discussions automatically leads to reduction of rework in the end
- Early involvement of test team – CTQ measurement mechanism
- CTQs as leading indicators of product quality

<u>Cross-functional approach</u>

- Reduction in communication barrier – way of working with marketing, product management e.g. challenging specifications, Kano, risk-benefit analysis
- Common language of CTQs with other disciplines for synergy

End-user perspective

- Development community sensitive to VOC – forces to think from problem domain rather than jumping into solutions.
- Measurement focus (variation rather than average), best case-worst case spectrum

System understanding (Product perspective)

- Transfer functions (CTQ flow-downs) triggers to understand the system better
- Understanding of noise and its effect on the system (usage environment)
- Trigger on "what could fail" (FMEA, mistake-proofing) improves robustness

Soft Aspects

- Mindset of "first time right" instead of "build-test-fix"

3.8.2 Points to Ponder

CTQ /domain dependency

- Everything is linked to CTQs so there is a chance of completely missing important ones
- Does not compensate for domain competency (Effectiveness of CTQ flow down, FMEA etc is determined by domain competency)
- Product management along with development community needs to be able to specify CTQs in a quantifiable way and as part of Consumer requirements specifications itself.

Software Engineering

- CTQs do not represent the complete requirements, hence the handling of other requirements, traceability, reviews, test coverage, regression need to be done also
- Configuration Management has to be addressed in the traditional way
- Additional effort needs to be budgeted in the initial phases (confidence of Project managers to plan more effort upfront that will compensate for effort saved in the end)

Use of Statistics in software

- Section 4.3 describes this topic in detail.

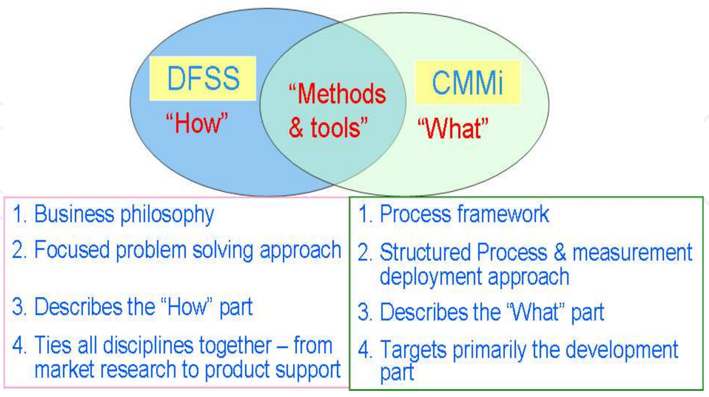
4. Challenges for DfSS in software

4.1 Software process framework

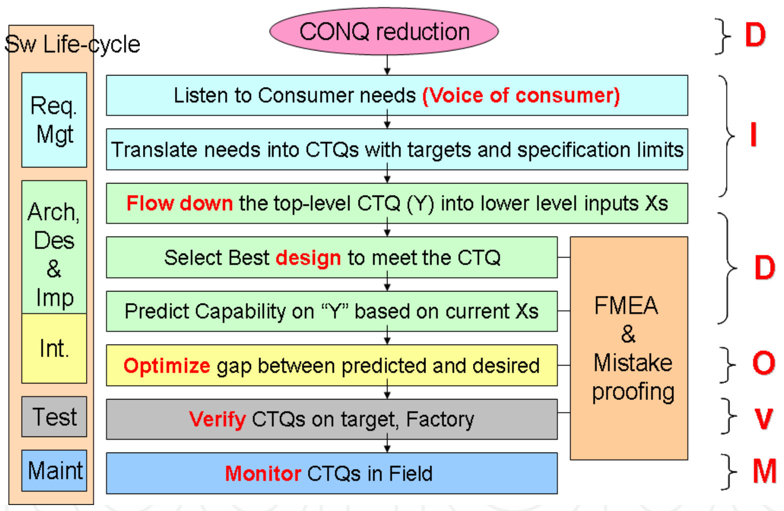
Software development organizations have traditionally followed the SEI-CMMI^R framework. Many of them have well established process framework and Quality management system built on CMM/CMMI model. So trying to deploy another methodology is a challenge in itself and cause confusion to the stakeholders. The obvious question in the minds of people is – we already have CMM and yet another model? Will it replace CMMI? Will it cause more overheads, more documentation? etc. It is important to answer these doubts upfront and rest assure the stakeholders that a happy marriage is indeed possible between DfSS and CMM/CMMI.

After studying both the models and also number of research papers (Jeannine & Forrester, 2004; Jeannine & Halowel, 2005), it is quite evident that both of them actually complement each other very well instead of working against one another. The CMM/CMMI addresses

the ***"What"*** (process) part with DfSS complementing it with the ***"How"*** part (methods and tools). So it is not one model versus another but rather a "methodology", that helps accelerate the CMM/CMMI journey by helping deploy processes better.
The Figure 22 below captures the essence of this thought process.


DFSS
"How"
"Methods & tools"
CMMi
"What"
1. Business philosophy
2. Focused problem solving approach
3. Describes the "How" part
4. Ties all disciplines together – from market research to product support
1. Process framework
2. Structured Process & measurement deployment approach
3. Describes the "What" part
4. Targets primarily the development part
Fig. 22. DfSS-CMMI Mapping

The other aspect is to ensure that the various phases and deliverables of DfSS are coupled with the internal milestones and processes as well as the overall goal of the organization. This is important part of Change management as described in section 2.4. The Figure 23 shows a broad level mapping of DfSS phases to the typical software development life cycle.


Sw Life-cycle
Req. Mgt
Arch, Des & Imp
Int.
Test
Maint
CONQ reduction
Listen to Consumer needs (Voice of consumer)
Translate needs into CTQs with targets and specification limits
Flow down the top-level CTQ (Y) into lower level inputs Xs
Select Best design to meet the CTQ
Predict Capability on "Y" based on current Xs
Optimize gap between predicted and desired
Verify CTQs on target, Factory
Monitor CTQs in Field
FMEA & Mistake proofing
D
I
D
O
V
M
Fig. 23. DfSS-Software Life cycle Mapping

4.2 Software FMEA

Failure mode and effects analysis (FMEA) is one of the well-known analysis methods having an established position in the traditional reliability analysis. The purpose of FMEA is to identify ***"UPFRONT"*** possible failure modes of the system components, evaluate their influences on system behaviour and propose proper countermeasures to suppress these effects. A failure mode and effects analysis (FMEA) can be described as a systematic way of identifying failure modes of a system, item or function, and evaluating the effects of the failure modes on the higher level. A bottom-up technique such as FMEA is an effective way to identify component failures or system mal-functions, and to "design rightly" the system under consideration (Pentti & Atte, 2002).

The standard guidelines provided by the FMEA cannot be directly used and would have to be tailored for applying it to software. Typically the best definition for ***"Severity"*** would be the one that the software teams use for their problem report classifications. Similarly for ***"Occurrence"*** and ***"Detection"*** it is better that the teams use their own tailored guideline based on a simplistic criteria of "Very high" to "Very Low".

By nature, software failure modes generally are unknown—"software modules do not fail in the literal sense as hardware failure, they only display incorrect behaviour"—and depend on dynamic behaviour of the application. The aim of the FMEA is to then uncover those situations.

The following are certain alerts/pitfalls/learning's to be aware of when doing software FMEA:-

1) Use case explosion – Software due to its very nature has many permutations /combinations of inputs and outputs which could be prone to failures. Hence FMEA would soon run into thousands of use-case combinations of failure-modes. Hence it is advisable to focus on failure modes associated with CTQs, Critical components/modules/functionalities etc
2) Capturing "Requirements not meeting" as failure modes e.g. set not recording as a failure mode for a DVD recorder etc. Recording is a basic requirement itself of a recorder so listing it as failure mode at a global level would not help. Instead the failure mode should delve deeper into the features
3) Not having the appropriate subject matter experts in the analyses. Failure modes largely dependent on competence, hence knowledge of domain (not software engineering but rather the usage of product in actual environment) is crucial
4) Attempting to perform FMEA on 100% of the design or code instead of sampling the design/code most likely to cause a serious failure
5) Excluding hardware from the analysis or isolating the software from the rest of the system as many of the failures result from the combination and not software alone
6) Typically for software, the severity "SEV" would remain unchanged and it is mainly the occurrence and detection that can be improved. For e.g. a hang/crash in a normal user operation is a severity "A" failure mode translating to a value of 8 for SEV. By taking various actions, its occurrence can be reduced/ eliminated or detectability can be improved. However even after taking actions, the severity would remain unchanged
7) The occurrence "OCC" value can be tricky sometimes for software. In a product development environment, normally a test will be done on few devices say 5 to 10 and issues do not surface out. When long duration tests are conducted in the factory on a

larger sample say 100 devices then the product starts failing. So OCC value could be different based on the sample taken and has to be accordingly adapted when validating the results

8) From software development life-cycle perspective, the DET value can take on different values for the same detection levels. For e.g. a control mechanism may have a high chance of detecting a failure mode making the DET value 4 as per the guideline. However based on whether that detection can happen in design itself or testing may vary the value. The team might give a higher vale for DET for something that can be detected only in testing as against that which can be detected in design.

4.3 Use of Statistics in software

Often this is one of most important challenge when it comes to using concepts like DfSS for software. Many software requirements fall into the Yes/No, Pass/Fail category so limit setting is fuzzy. Most of them would become critical factors (CFs) and not CTQs in the "continuous data" sense

- Predicting DPMO (defects per million opportunities) may be misleading (out of limits). This is because the specifications limits in cases like responsiveness are soft targets. Just because it takes 0.5 seconds more than Upper Specification Limit to start-up does not necessarily classify it as a defective product. In Six sigma terms anything beyond Upper spec limit and less than Lower spec limit becomes a defect
- Random failures due to only software are rare due to which concept like Mean-Time-Between-Failures (MTBF) for software alone is questionable, however it makes sense at overall product level
- No concept of samples – the same piece of code is corrected and used, so advanced statistical concepts have to be applied with discretion

However this does not mean that statistical concepts cannot be applied at all.

- The starting point is to challenge each specification to ensure if some numbers can be associated with it. Even abstract elements such as "Usability" can be measured as seen in section 3.5.2
- For many of the software CTQs, the Upper limits and lower limits may not be hard targets, nevertheless it is a good to use them as such and relax it during the course of the development
- The change in Z-scores over the releases would be more meaningful rather than absolute Z-scores
- All Statistical concepts can be applied for the "Continuous CTQs"
- Many of the Design of experiments in software would happen with discrete Xs due to nature of software. So often the purpose of doing these is not with the intent of generating a transfer function but more with a need to understand which "Xs" impact the Y the most – the cause and effect. So the Main effects plot and Interaction plots have high utility in such scenarios
- The hypothesis tests such as t-Tests, F-Tests, ANOVA are useful in the Verify and Monitor phase to determine if indeed there have been statistical significant changes over the life cycle or from one product generation to next etc.

- Statistical Capability analysis to understand the variation on many of the CTQs in simulated environments as well as actual hardware can be a good starting point to design in robustness in the software system.

5. References

Ajit Ashok Shenvi (2008). Design for Six Sigma : Software Product Quality, *Proceedings of the 1st India Software Engineering Conference*, pp. 97-106, ISBN:978-1-59593-917-3, Hyderabad, India, February 19 - 22, 2008. ISEC '08. ACM, New York, NY, DOI= http://doi.acm.org/10.1145/1342211.1342231

Haapanen Pentti & Helminen Atte, Stuk-yto-tr 190/August 2002. Failure modes and effects analysis of software based-automation systems

Jeannine M. Siviy and Eileen C. Forrester. (2004). Accelerating CMMi adoption using Six Sigma,Carnegie Mellon Software Engineering Institute

Jeannine M. Siviy (SEI), Dave Halowell (Six Sigma advantage). 2005. Bridging the gap between CMMi & Six Sigma Training. Carnegie Mellon Sw Engineering Institute

Jiantao Pan. 1999. Software Reliability. Carnegie Mellon http://www.ece.cmu.edu/~koopman/des_s99/sw_reliability/

Minitab tool – Statistical tool. http://www.minitab.com

Philips DFSS training material for Philips. 2005. SigMax Solutions LLC, USA