# Appendix A

# Questionnaire

# QUESTIONNAIRE

Dear Madam/Sir,

I am a student reading for a MBA in Management of Technology at the University of Moratuwa. As part of the course requirement, I am doing a research project on "***A Study on Software Product Development Approaches in the Sri Lankan Software Industry***". Through this questionnaire I hope to gather information on the existing Software Development Life Cycle Models being used. I shall be grateful if you could spare a few moments of your valuable time to complete this questionnaire, as it will be very helpful in the successful completion of my project.

- If you feel some questions are of confidential nature, you may skip and continue with the rest.
- Where more than one answer is correct for a particular question, please indicate all such answers.
- If you do not agree with any of the answers, please state your views in the space provided for comments.

Thank you

Yours sincerely

---

Vignarajah Manoharan

Name : _____

Name of the Organisation : _____

Designation : _____

1). Is your organization into Product business, Project business or both?

Product ☐ Project ☐ Both ☐

2) How long has your organization been in the IT Industry?

☐ Under 3 years ☐ 3-5 years ☐ 5-10 years
☐ 10-15 years ☐ 15-20 years ☐ Above 20 years

3) How long has your organization been in the Software Product business?

☐ Under 1 year ☐ 1-3 years ☐ 3-5 years
☐ 5-7 years ☐ 7-10 years ☐ Above 10 years

4) How many Software Products your organization has developed and sold?

☐ 1 ☐ 2 ☐ 3
☐ 4 ☐ 5 ☐ Above 5

5). What is the roll that you play in the Software Development Life Cycle (SDLC) Model followed by your organization?

☐ Project Manager ☐ Business Analyst ☐ Analyst Designer / Architect
☐ S/W Developer ☐ Quality Assurance ☐ Implementer
☐ Trainer ☐ Database Administrator ☐ Trainer ☐ Other

6). Which SDLC Model is followed by your organization?

☐ Waterfall ☐ Rational Unified Process ☐ Prototyping
☐ Exploratory ☐ Spiral ☐ Ad-hoc development

☐ Extreme Programming ☐ Other

7) Please state the rate of problems faced in the following areas

| | Area | Problems | | | | |
|---|---|---|---|---|---|---|
| | | **Very High** | **High** | **Low** | **Very Low** | **Not at all** |
| 7.(a) | Requirement Gathering | ☐ | ☐ | ☐ | ☐ | ☐ |
| 7.(b) | Design | ☐ | ☐ | ☐ | ☐ | ☐ |
| 7.(c) | Coding | ☐ | ☐ | ☐ | ☐ | ☐ |
| 7.(d) | Business Knowledge | ☐ | ☐ | ☐ | ☐ | ☐ |
| 7.(e) | Change Management | ☐ | ☐ | ☐ | ☐ | ☐ |
| 7.(f) | Communication of Solution to Customer | ☐ | ☐ | ☐ | ☐ | ☐ |
| 7.(g) | Process Monitoring | ☐ | ☐ | ☐ | ☐ | ☐ |

8). What rate of success your organization has achieved by using the currently available SDLC models for Software Product development?

☐ Extremely Good ☐ Very Good ☐ Somewhat Good
☐ Not Very Good ☐ Not at all Good

9). How do you rate the suitability of the currently available SDLC for Software Product development?

| | SDLC | Suitability | | | | |
|---|---|---|---|---|---|---|
| | | **Extremely Good** | **Very Good** | **Somewhat Good** | **Not Very Good** | **Not at all Good** |
| 9.(a) | Waterfall | ☐ | ☐ | ☐ | ☐ | ☐ |
| 9.(b) | Rational Unified Process | ☐ | ☐ | ☐ | ☐ | ☐ |
| 9.(c) | Prototyping | ☐ | ☐ | ☐ | ☐ | ☐ |
| 9.(d) | Exploratory | ☐ | ☐ | ☐ | ☐ | ☐ |
| 9.(e) | Ad-hoc development | ☐ | ☐ | ☐ | ☐ | ☐ |
| 9.(f) | Extreme Programming | ☐ | ☐ | ☐ | ☐ | ☐ |

10). Do you think that there should be a new SDLC model for software product development?

☐ Yes ☐ No.

***Additional comments or suggestions***

Thank you for the courtesy, patience and most importantly for spending your valuable time in filling this questionnaire.

**Vignarajah Manoharan**

# Appendix B

# Role of Design in Software Product Development

By William Buxton

# The Role of Design in Software Product Development

## Introduction

For a number of years software developers have struggled with a frustration with their own industry, software product development. The roots of this frustration lay in two basic questions:

1. Why were we so bad at bringing out new products and by this what is meant is new as opposed to "n+1" products, or upgrades?

2. Why were all of our development efforts (both new and n+1) always late, over budget, and so far in quality from what we wanted them to be, and knew they should be?

At first, the author took this frustration personally, both as an individual, and as an executive of a software company. But gradually, the author looked around and realized that it wasn't just us. The whole industry seemed to suffer from the same malaise. In fact, despite our failings, we were better than many. However, this was small comfort. After all, how good can you feel about your own performance if the best that you can say is that you are not as bad as some of your competitors?

The reality was that, standards should be set according to what you knew you could and should be doing, not by industry norms that fall well below that mark. Despite new approaches to code reviews, agile programming, iterative design, object oriented design, and the like, there had to be a better way.

## Film and Industrial Design

Why might film and industrial design have some relevance to software product design?

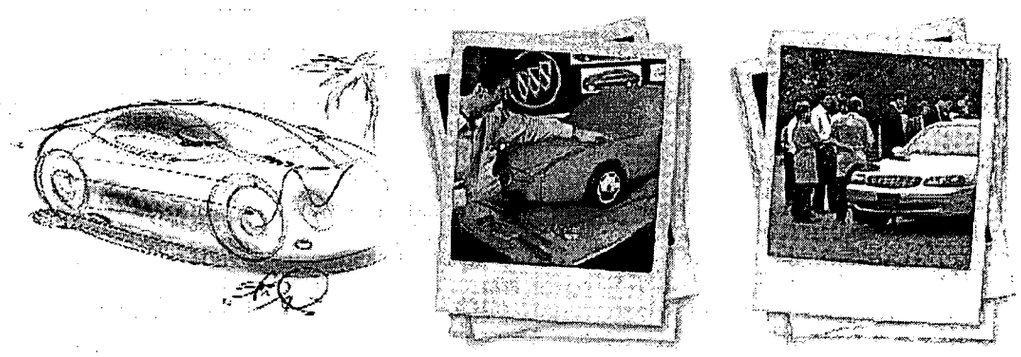
While perhaps seeming ordinary, the reason that their process interested the author was that they generally succeeded in exactly the areas where we(from the IT industry) were weak, at best, and failing, at worst: namely, getting new products out on time, on budget, and with a high standard of quality and innovation.

What is understood from the above is the assumption that the process of both filmmaking and industrial design had something in common.

In film, this phase of the process (Design) is called "preproduction." Most film buffs will know this term and may even be familiar with some of its components, such as story development and assembling the key players. But it actually goes much further than that. Typically, you only consider a film for green-lighting after it has assembled a complete "package." This package not only includes script, director, and key cast members but also budget, release date, marketing plan, and a detailed production plan. In other words, before committing the resources to actually shoot the film, there must first be an extremely detailed creative, production (technical), and business plan in place. While this represents a significant up-front investment before even knowing if the film will happen, the costs and competitive nature of the industry are such that it is unthinkable to commit to something where these things are not nailed down in advance. There is still room for creativity and innovation during production. However, it could be argued that it is precisely the up-front planning in the pre-production phase that provides this freedom, and lets the director and producer manage it, while staying within the framework of what was approved.

Likewise, when a company like BMW or General Motors wants to bring a new car to market, there is a very similar process that precedes making the decision to put the car into production. This phase involves people from engineering, marketing, and design. By

the time that the executive team meets to decide whether to green-light the project or not, they know what they are going to build, how and where they are going to build it, how much it will cost, who will want it, and how they are going to sell it. But what struck the author most strongly about the process was this: The design process was such that at the end, before making the decision to go into production, they output a model of the proposed vehicle that even experienced people could not tell if it was the final product or not. And yet, despite this refinement, if the vehicle were approved for production, there would still be at least a year of engineering before it could be manufactured. And, as in film making, during the product engineering and manufacturing phases, there was still room for innovation and improvement.



(a) (b) (c)

**Figure B.1: Three of the many phases of the automotive design process**

(a) Early "iteration", or concept sketching;
(b) Design refinement using 3D clay scale models;
(c) Design review using full size clay model. Credit:

It is this front-end process that we see as generally missing in the software industry, and it is the belief that its absence is largely responsible for the industry's poor performance with respect to the quality of the products and bringing new products to market.

## Design and Engineering: A Social Contrast

The focus of the discussion so far has been on various industries and their processes. However, the over-riding concern in all of this is human. Not only are people, in the form of users, the ones who appreciate and are affected by product quality, but it is also people who populate the processes that create it. Consequently, to discuss a process, or change in process without taking into account the people who carry it out would be to violate the basic tenants of a human-centric perspective.

Design is a separate process from engineering. More to the current point, design takes a very different mind-set and training than does engineering. For example, employees who were a disaster in product engineering were outstanding in the design phase and vice versa.

In a product engineer we require someone who is extremely well organized, and who is technically strong. Engineer should not be innovating with the product concept. What is required is their creativity directed at improving the quality of the product and how its vision can be effectively brought to market. However, for this to happen, the engineer needs to know what that vision is. Therein lies the problem in most of our software products, since (more often than not) the engineer is making design decisions at the same time that the product is being built. And, to make matters worse, since there is no centralized responsibility for design (much less an explicit design process), various engineers are making design decisions that are consistent with their own model of the product, but inconsistent with those of other engineers. The best example is a film that has no director, much less a script, storyboards, or production design.

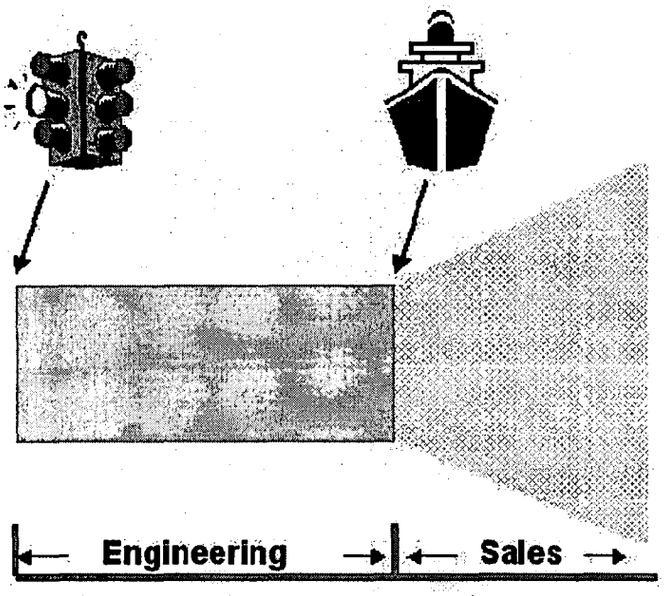





**Figure B.2: The status quo in software product development.**

Projects get a green light right at the start, and go directly to engineering. The next phase is when they slip usually late, with bugs, and over budget and missing functionality.

Designers are a different breed altogether. They went to Art College rather than engineering school. They tend to work different hours, dress differently, and use different references. Their primary skill is not making things that work, that can be maintained, or that can be built. Rather, it is sketching. That is, quickly working up product concepts into a concrete form sufficient to serve as the basis for discussion, evaluation and questioning. From the perspective of quality, the metric is in terms of concept and speed, not refinement of execution.

There are three key observations:

1. Regardless of how you characterize the two, design and engineering are very different skill sets, and the two professions are made up of two very distinct populations.

2. Each skill set is essential for the production of quality products, but neither is sufficient on its own.

3. For the most part, software companies and their process have a huge shortfall in the design arts and process, and much of their failure is due to their attempting to integrate design into the engineering processes, and have it done by engineers.

If you asked both a designer and engineer involved in software design, you would find that they both advocated the idea of rapid prototyping and iterative design. "So!" you might say. "Here is a similarity, not a difference." Well, here is what the author has experienced.

One company had an idea for a new product. After looking at the problem that the product was trying to address, and consulting with customers, the design group started doing some concept sketches. These evolved into working prototypes that captured the essence of the evolving design, in particular in terms of its feel and interaction involved in performing the desired function. At the end of the process, which included customer validation, the result was a prototype that was, to the intended software product, what a full-sized clay model would be in automotive design.

Based on this, the project was green-lighted and the product taken over by product engineering. Partially because of the reliability of the prototype to the basic concept and intent, what emerged (in record time), was a product that not only was faithful to the prototype, but was better. Yet, it was on time, met the specifications, and was as usable as it was free of bugs.

You might conclude, therefore, that this was an example of a successful interaction between design and engineering. Well, not quite. In fact, there was a significant part of the engineering team that felt that the design group had failed, since they could not use any of the code from the design prototype in their product. The resulting

reimplementation was seen as a waste, and just another example of the design team's technical incompetence.

It is commonly said that "Engineers view prototypes as part of the process of building things. Designers build prototypes to criticize and tear apart."

In this case, the engineers in question viewed the prototype in engineering terms, as a piece of not-production-quality software. The designers, on the other hand, viewed it as an "interactive sketch." It was to the final product what a rough sketch of a car is to the final vehicle. The key to appreciating the underlying point here is that the idea of sketching was extended to include feel, and not just look; to serve its purpose, the sketch had to be interactive. However, since the medium that the sketch used to accomplish this included some of the same tools and technology that would also be used in the engineering phase, it was prone to a misinterpretation of its intent.

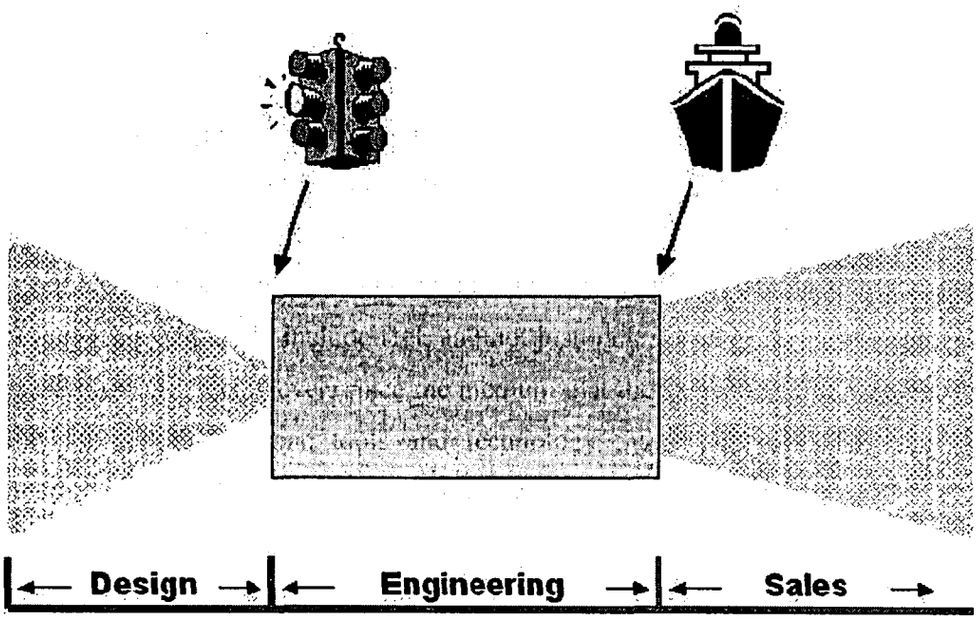



**Figure B.3: Inserting an explicit design process at the front end, prior to green lighting the project. The process is represented as a funnel, since the number of concepts to emerge is always anticipated to be fewer than enter.**

Still, it is unfair to blame the engineers, as there is typically nothing in their training or experience that would lead them to think any differently. Design, its role, and its process are simply not part of the culture or educational process of software engineering, all of which leads us to the situation in which we find ourselves today.

## A Sketch of the Process

What we can do is consider this explanation, and the resultant level of detail, as analogous to a sketch, rather than a final rendering.

In the simplest terms, the difference between Figure 2 and Figure 3 illustrates the basic argument that the author is making: the insertion of a design process at the front end of product development. The primary assumption in advocating doing this is that the cost and time lost due to this additional stage will be significantly less than the cost and time lost due to the poor planning and over-runs that will inevitably result if it is not included.

The author's perspective is that the bulk of software industry is organized around the demonstrable myth that we know what we want at the start, and how to get it, and therefore build our process assuming that we will take an optimal, direct path to get there. This is not correct. The process must reflect that we don't know and acknowledge that the sooner we make errors and detect and fix them, the less (not more) the cost.

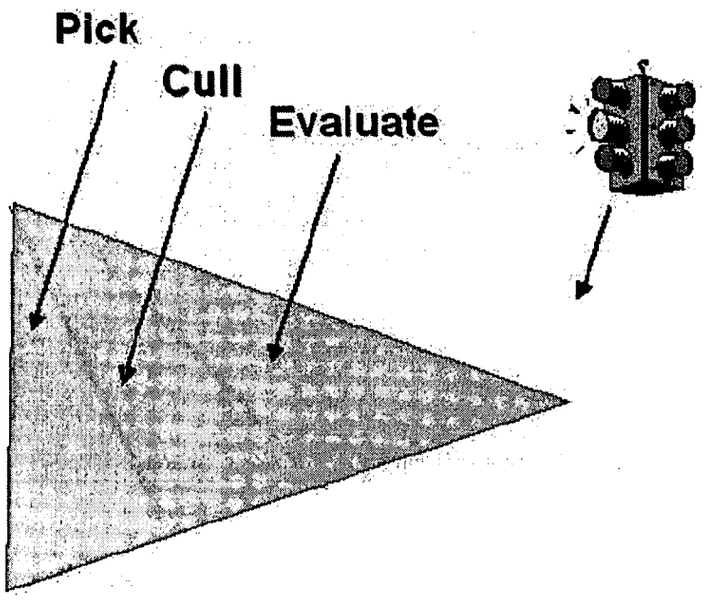



**Figure B.4: Changing criteria for evaluating ideas at one progresses through the funnel.**

While the back-end engineering process is about getting things right, the front end is loaded towards making errors sooner rather than later, detecting them, and learning from them. One key to "sketching" in this context, is ensuring that the cost of errors, and the subsequent lessons, is contained within acceptable limits. The other key is recognizing that the generation of ideas is targeted at converging on a design, rather than adding diversity for novelty's sake. Hence, the process is represented as a funnel, in which ideas are explored and refined. At the beginning, ideas are "cheap" and lightweight processes are all that are needed to evaluate them. As they become more and more refined, so does the investment that we have in them, so therefore the strictness with which they are examined increases. This continues up to the point where they are brought forward for formal go/no go evaluation as part of the green light process.

What is critical to note in all of this is that despite breaking the product lifecycle into three basic phases, as shown in Figure 3, this does not mean that the process should be viewed as three silos, roughly partitioned among design, engineering, and sales. One only has to go back to our précis of the process used in film to see this point. The "Design

Phase" as the author intend it to be understood, does not imply just the design of the actual product. Rather, as in film, it also includes the design of the engineering process (the technologies that will be used that is, the engineering/ manufacturing plan), the design of the marketing plan, and the refinement of the overall business model.

Likewise, green-lighting a project is not a signal that the designers are now off the project, or that they in any way "throw the project over the transom to engineering. Figure 5 describes the ongoing responsibilities and involvement of the key factions throughout the lifecycle of a product.

One can think of the ongoing relationship and distribution of responsibilities between design and engineering by similarity to the role of the architect and structural engineer in designing a building. The architect has prime responsibility at the front end but must clearly work with the engineer in order to ensure that the design can be built. On the other hand, the architect has ongoing responsibilities, after green-lighting, for the evaluation and monitoring of the product as it moves through the construction phase, ensuring that the design intent is respected.

Returning to the comparison with film, if we consider the director as the head of the main creative department of a film, a useful exercise or test on any software product would be to ask: "Who is the equivalent of the director?" "Do they have comparable power and responsibility? If not, why not?" "If not, why do we believe that the integrity of the design will be maintained?

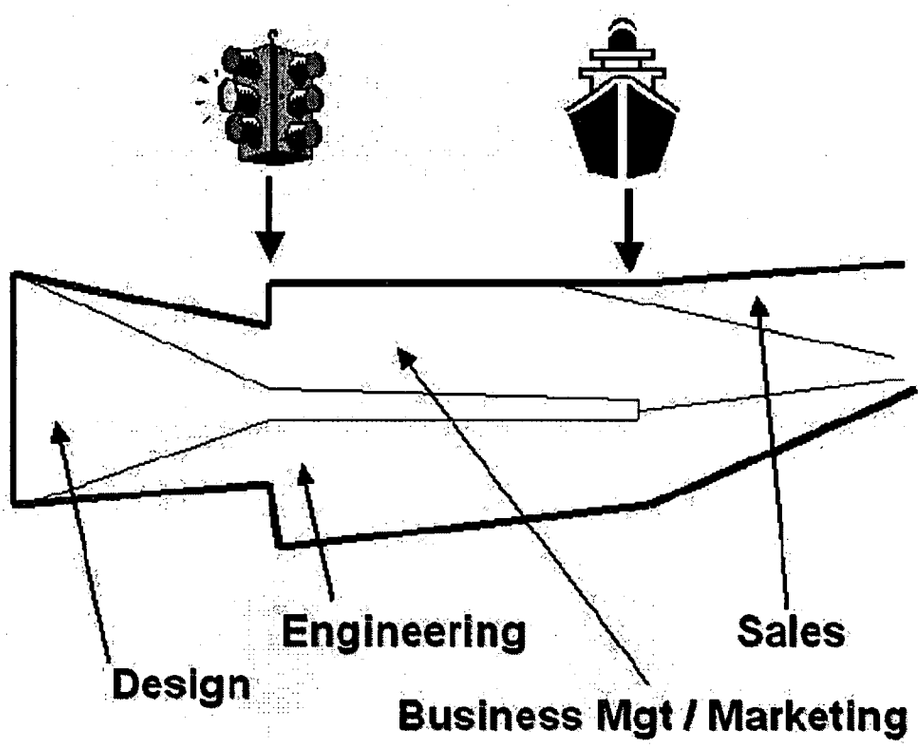



**Figure B.5:- No silos. Here we try to capture the nature of the ongoing responsibilities of the various teams throughout the overall process. Of central importance for our purpose is the belief for design includes the design for business/ marketing plans and engineering / manufacturing plans, as well as the product itself. In this it is no different from film making**

## Where are the Users in All of This?

At this point we risk being so bogged down in discussions about processes, roles, responsibilities, diagrams, and so forth, that we lose sight of why we are doing this in the first place: to design products that people want, need, like, and can use. Yet, so far the author has not said anything about user-centered or usage-centered design, usability testing, or any of the other associated buzz words. This may seem especially curious.

First, the absence is explained partially to simply take this part of the process for granted. The author feels that we should be approaching the point that we do not need to point out

such basic things any more than a paper on mathematics needs to include a discussion of basic arithmetic.

The second comment is that it is precisely a concern for users that underlies the value of the proposed approach. Techniques fundamental to the design phase, such as sketching and prototyping, mean that iterative user testing and validation can occur much earlier and in a form that captures the interactive nature of the system. The consequence is that user input can begin early enough to influence the design of the product.

User involvement then obviously continues through the engineering phase, in the form of usability testing.

## The analysis of a green Light

Perhaps the most important step in any of this, in terms of it working, is the nature of the green-light process, that is, the criteria and process whereby the decision is made to go to market or not. We tend to get what we measure and reward, so it is critical to ensure that these criteria are in alignment with our objectives. The process needs to ensure that the proposed product has a good fit to the intended market. But it also must ensure that everything has been done to make sure that it meets well defined measures of quality in the process.

Furthermore, the engineering and manufacturing plans must be presented in order to determine not only what is to be built and that it can be build, but also how it will be built. The higher the reliability of the final design prototypes, the more we know on the what side of the equation. And, if you don't have a very clear idea of what you are building, how can anyone responsibly or reliably say if it can be built, how it can be built, when it can be built, or how much it will cost to do so?

Likewise, the green-light process needs to include a serious evaluation of the analysis of the product market, and the sales and marketing plan and not just of the version one

product, but of the product over its larger life-cycle (including how well it scales, from a business perspective, and how it fits into other products offered by the company).

From the holistic perspective, one also needs to go beyond this analysis and determine what else one might otherwise be doing with the resources thus consumed, in terms of maximizing ROI as well as service to the customer.

The author sees the green-light process as the establishment of a number of "contracts" among the key executives responsible.

1. The Executive Responsible for Design: agrees to deliver products to the green-light process according to criteria established by the executive as a whole, and is evaluated on doing so.

2. The Executive Responsible for Engineering: on green-lighting a project, agrees to deliver that product, meeting specification, at a specified time and within a specified budget, and is evaluated on doing so.

3. The Executive Responsible for Sales: on green-lighting a project, agrees to sign up for a specific revenue target, in a specific quarter, if the product is delivered meeting specifications (including quality), at the specified time, and is evaluated on doing so.

The executive team, as a whole, takes responsibility for establishing the strategic direction of the product agenda and roadmap, as well as the green-lighting of projects.

However, while the above seems "common sense" and "obvious," see how much it contrasts with common practice, where we green-light products before we know what they are, and forecast revenue before there is any design, much less meaningful validation. The consequence is that we "have" to ship whatever we have to meet our numbers, and the product that we start to engineer is seldom the same product that actually ships (if, in fact, it ships at all).

## A Bit of History

In order to better guide future directions, as well as prevent us from lashing ourselves too much for our past behavior, it is worth briefly looking into what got us into this situation in the first place. Dan Olsen explains it in terms of three factors:

1. Immaturity of the field.
2. Flexibility and low manufacturing costs of software.
3. Moore's Law.

He describes the software industry in terms of the old American frontier, where the winners got there first without much concern with how they got there. Behavior was largely driven by the large rewards to be won, coupled with the fact that they could be (and were) won without such niceties as design or usability. Remember, safety standards and the rules of the road (much less comfort and styling) came after, not before, the introduction of the automobile.

Furthermore, unlike film or automobiles, where there is a significant cost of goods, including materials and manufacturing costs, software (while hard to write) was relatively fast to create, modify, and bring to market. With less upfront expenses, the rush to market was coupled with relatively less risk, thereby further fueling the rush to the frontier.

Finally, the reality of Moore's Law meant that there was a "perpetual frontier," and therefore little or no time for things to settle down to the point where such "niceties" as design, quality, and usability standards might become established. Again, during a gold rush, nobody involved thinks about the environment. Just think of this as a sustained gold rush.

From this characterization by Olsen, the author wishes to make two main observations. First, as long as the gold rush exists, things are more likely to change by introducing a

new process. Second, the economic downturn in the technology sector over the past few years can be largely interpreted as suggesting that the value of the gold is rapidly diminishing precisely due to the absence of associated quality of design, usability, and manufacturing. All of this argues, the author would suggest, all the more for the need for the changes that we have discussed.

# Appendix C

# A Framework for Managing Software Product Development

By H.Dayani-Fard

# Appendix C. A Framework for Managing Software Product Development

## Introduction

Software products have become an integral foundation of most businesses today. Most companies rely heavily on their database products, data mining, and other analysis tools to perform their day to day operations. Furthermore, the dynamic nature of businesses creates new requirements in terms of functionality, reliability, and performance (Lehman 1996). The result is that software development organizations are under daily pressure to improve the functionality and quality of their products to satisfy their customers' expectations while competing with other similar products.

In today's industry, examples of software products that have been in the market for over a decade are not rare. The long lifetime of software products and the continual growth of products, in conjunction with dynamics of development teams, have created considerable amount of management problems. The software is growing in size and complexity, while the development team is continually changing -- new members are added, others leave or change roles and responsibilities. Building a "good" product under such circumstances is a challenge.

In order to build a "good" product one must first define the meaning of "goodness" and, based on that definition, develop a management framework. Goodness is multidimensional. There are many stakeholders in the development team, each of whom has a different set of concerns that contribute to overall goodness.

## Release-Based Life Cycle

Software product development follows a release-based life cycle. Figure C.1 illustrates a typical product development cycle. After a version of the product has been released, there will be split in the code base. One side continues along the same line as the most recent release. This side embodies minor modifications such as error corrections, minor enhancement of features, and other regular periodic changes. On the other side, we have the development cycle for the next major release. This line represent a longer-term plan for a future release based on market input, customer requirements, and technological advancements. The key issue on this diagram is the feedback from one line to the other and previous releases. This picture is consistent with observation made by Parnas (Parnas 1979) who expressed that software products are a family, not just a single entity.

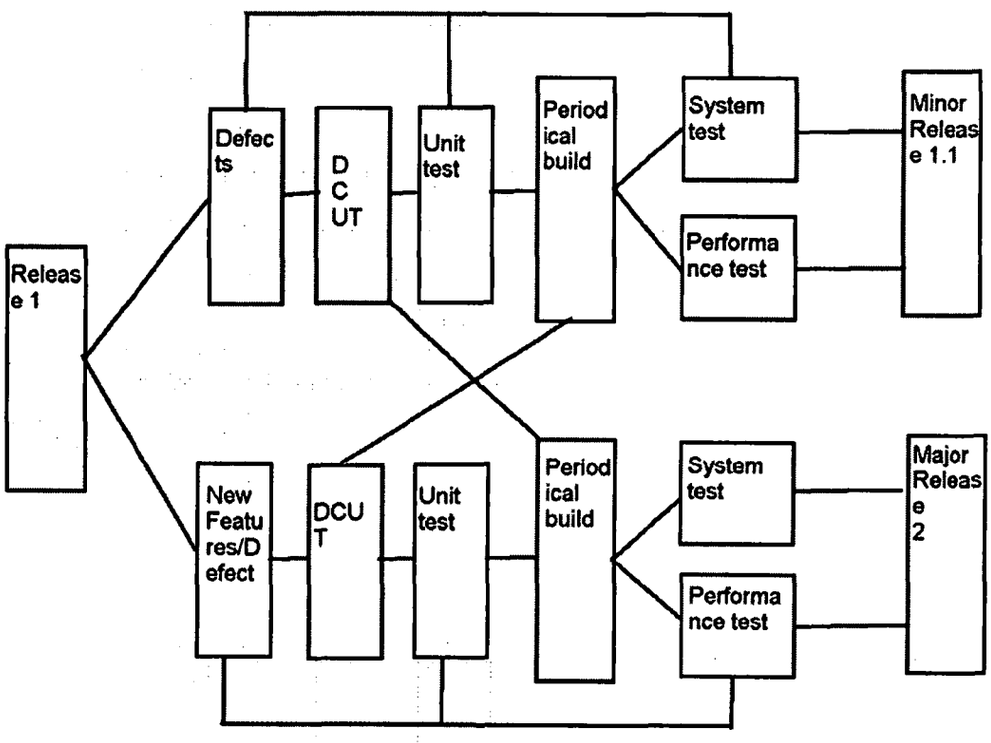



**Figure C.1 Software Product Release Life Cycle**

Most large products must evolve gradually. In other words, the lines drawn in Fig. 1 represent a small waterfall model that is embedded in a spiral model. At any given point, there are small changes that must be applied to the product: either fixing a defect or adding a feature. Such changes require specification, design, coding, and function testing. These changes are then included into a build -- the software product is built on regular intervals embodying the most recent changes (Cusumano and Selby 1997).

Taking into account multiple major releases, minor releases, and multiple builds of the products, the management of a software product's development has become so complex that even most experienced managers cannot effectively deal with various issues that may arise. The goal is to build a "good" product. The challenge is to define goodness precisely in terms of operational issues: progress, quality, backward compatibility, future extensions, and so forth.

The proposed framework is based on the observation that software development encompasses a multitude of concerns. The variation in concerns is the result of different tasks and responsibilities that members of team are assigned. Members who are directly involved with customers are mainly concerned with defect removal. Those who deal with nonfunctional requirements such as performance are mainly concerned with the response time of the product. The developer's concern is how to interpret an existing piece of code in order to make modifications. The executive's concern may be to deliver the new release on time for market share. In other words, goodness is a multidimensional parameter. We will take a look at a multidimensional framework that enables a development group to define their opinion of goodness, measure it, track the progress, identify deviations, and perhaps study causalities among different issues.

## Framework

As mentioned in the previous section, the framework is based on the observation of the multiplicity of concern in a software product's development. To manage such a multidimensional space, we divide our high level concerns into three axes:

- *Progress indicator*: this index represents how many new features have been completed
- *Quality indicator*: this index represents how the product will be viewed by customers
- *Health indicator*: this index represents how easily our code can be modified for future enhancements

As we can see, each axis focuses on a set of related concerns. For example, the health indicator focuses only on the non-functional requirements of a software product, such as understandability and maintainability. These concerns are organized an orthogonal axes to emphasize the relation among these concerns. A software product that has poor health is much more difficult to modify; hence, the progress will be much slower than expected and it is more likely that new features that are added will result in more defects. The status of a software project can now be represented as a point in this space as shown in Figure. C.2.

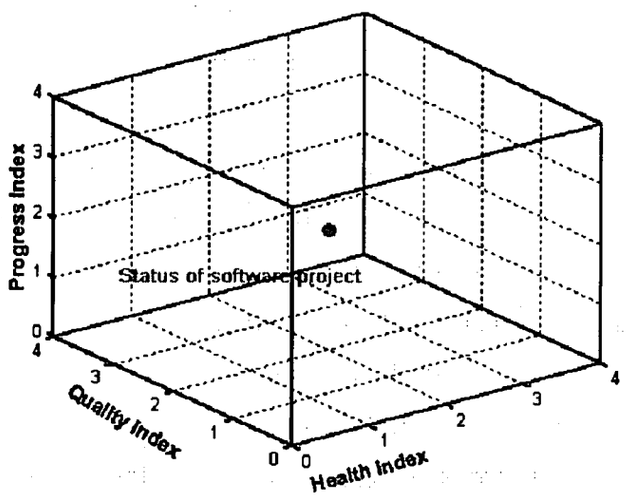



**Figure C.2 Three Dimensions of Software Project**

If the status of a software product can be represented by a point at any given point in time, we can determine the current status of a project and our goal as two points A=(p1,q1, h1) and B= (p2,q2,h2). Next we can define the ideal progress as a line connecting these two points as shown in Fig.C.3. It is unlikely that a particular software product's development progresses linearly as the ideal line; however, we can use this ideal line as our ultimate progress goal. For a more realistic picture, we can sample the status of our product at different points in time and fit a line through these points. The resulting curve represents an approximation of our real progress. Now we can define "risk" as the distance between the real progress and the ideal progress. This enables us to determine when our progress is deviating beyond our acceptable risk and a decision must be made on how to correct the problem.

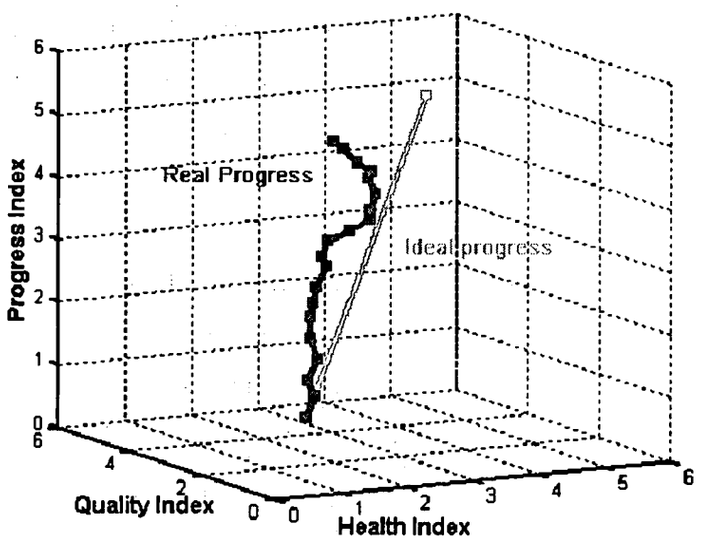



**Figure C.3 Ideal vs Real Progress**

The indicators on each axes represent a number. This number is an index of a set of values that in combination represent a concern. For example, we can define our quality index as a summation of a set of measures **mi**, each of which represents a quality concern. For example, the number of defects per some unit of programming can be a quality indicator. Furthermore, all quality indicators may not have the same importance. Hence, we assign a weight to each measure to take into account its importance. More precisely, we can write:

**Q = F(mi, wi)** for all i

where **Q** is quality index, **F** is a summation function, **mi** s are individual metrics, and **wi** s are their respective weights. We can similarly define other indicators. Depending on the depth of our definitions, each indicator can further be broken into subparts. For example, a software system usually consists of several components and each **mi** can be an index of a certain quality for all components. This recursive definition provides us a multidimensional data space.

## Implementation Issues

In order to be able to determine various indicators as described previously, we need to have access to various data sources. Typical data sources include, program source code for calculating health indicators; a configuration management and version control system for data about defects, features, and other data about the programs; and any other source of data that includes requirements, designs, or other project management data (e.g., data from case tools). A possible solution can be based on data warehousing technology. Figure 4. depicts one such implementation.

As we can see, the architecture of our data warehouse consists of three tiers. The *extraction and load* tier focuses on gathering data from various data sources, converting data to a uniform model, and cleansing the data to replace missing fields and/or unify the format of similar data. The *central data warehouse* tier is the main repository of all data about the software product and project. In this tier, we have all historical data that is uniform and cleansed. The central data warehouse enables the last tier to determine, based on its data needs, what to import from the warehouse to perform its functions.

The main advantage of such architecture is that the management model can be easily modified and improved based on feedback. The central data warehouse, in essence, reverses the traditional process-centric approach of software management to a data centric model. In typical management models, one must first determine the process, what is to be

done, then search for the data that is necessary for the process. In this approach, we gather all available data in the central data warehouse, while the processes are created at the third tier. Changing a process only requires implementation of a new process that imports its data from the central data warehouse.

At first glance, we can conclude that such an architecture will be expensive. The design of central data warehouse requires intensive data modeling. The extraction of data can be expensive and time- consuming. Finally, storage of such a large amount of data, despite the falling prices of storage, can put a dent into the budget of product development. The counter argument is based on paying off of the cost. Since software products continually evolve, their longer life time requires longer term planning. The cost of design of the warehouse and data extraction mechanisms over a long period of time can be significantly reduced. In terms of storage cost and the timing overhead associated with loading the central data warehouse, we need to base have a look on the benefits that are realized by having such a system. Using the data warehousing solution depicted in Figure C.4, we can determine the status of our project at the start, our final goal, and the current status. The fundamental management issue in project management is defining progress and quality in such a way that it can be measured and tracked.

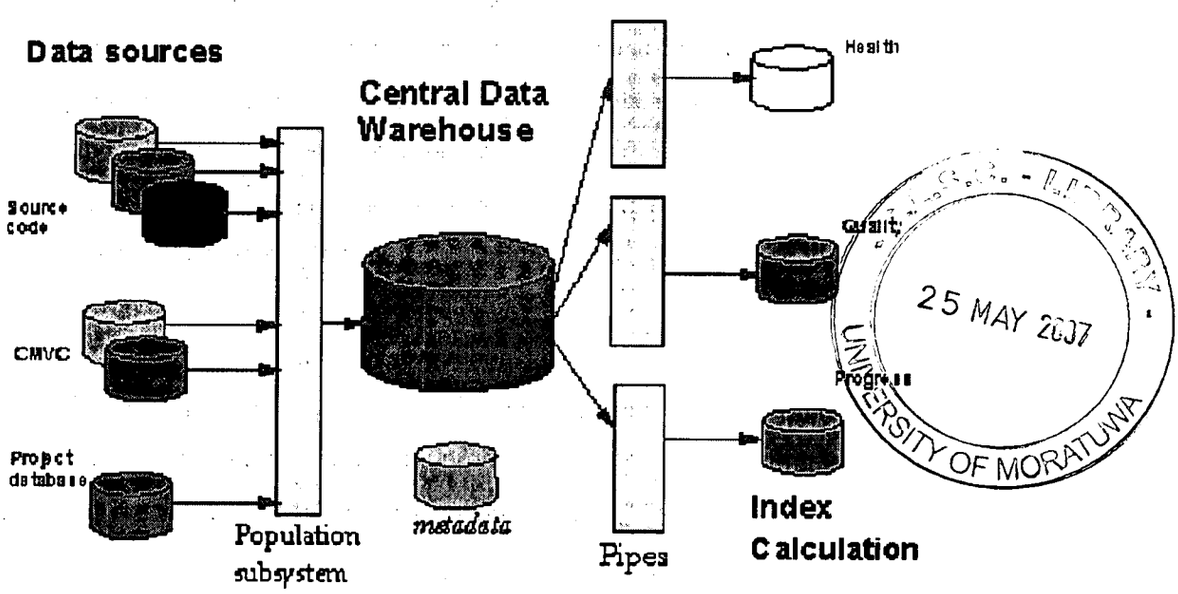



**Figure C.4. A Three-Tiered Data Warehousing Solution**