

## ■ Shifting focus from technology to product in software development

Anders Dinsen

Software is based on technology and software engineers love it. The most popular subject of discussions between programmers is which platform to choose for the next project; whether it should be Linux or Windows, or whether it is better to use an object oriented design, or a procedural design. New technologies are heavily accompanied by marketing campaigns that promise many good things if we base ourselves on this new thing.

In ASYM we find that technology fascination often leads us nowhere when we look at the end result, except into spending extra money and time.

This paper points out that technology in itself is never a solution, and that we always need to evaluate everything against its influence on the result of the project: The product. I present ASYMs product focused software development paradigm.

Focusing on the end product (and knowing how to focus) in software development projects, will lead to better products in any project in which it is done.

**Table of Contents**

1. Technology focus.....3  
 1.1. A manager in love with technology.....3  
 1.2. What went wrong?.....4  
 2. Good products.....5  
 3. Shifting focus: Provocations.....6  
 3.1. Kill your darling.....6  
 3.2. Reflect, sleep on it, make slow decisions.....6  
 3.3. Stop reading books.....7  
 3.4. Communicate directly.....7  
 4. Product focused software development paradigm.....9  
 4.1. Visualisation.....9  
 4.2. Simplification.....10  
 4.3. Multiple solutions.....10  
 5. Conclusion.....11  
 6. Appendix.....12  
 6.1. References.....12  
 6.2. About the author.....12

## 1. Technology focus

Traditional software development starts with an idea for a product that will fill a market niche. This idea is presented to the engineers, who start analysing the idea and come up with a suggested way of realising it. A management team is formed and team roles are quickly defined: UI team, power controller interface team, hardware team, function test team. Specialists and team leaders are called in, programmers are hired in, resources spent, people work long hours. Everything looks good and right by the book.

This is a well proven – and structured – way to do things: Focus on problem, find solution and start off. The quicker the better; and the more sure of the solution the engineers are, the more efficient they seem to be at their job. Most software engineers have learnt – by hard work and failures to convince managers of the value of their knowledge – to not present alternatives to a solution, to appear determined about the solution, and to give quick answers.

### 1.1. A manager in love with technology

*Back in the late 1990s, a medium sized Nordic company that made specific kinds of computer equipment was taken over by venture capitalists who saw big potential in it. The company was in need of new blood and new product innovation. The new CEO (who was one of the new owners) went on a private journey to the USA shortly after the take over. Over there, he came across a book about a new technology that was to change the way we interconnect our computers. This technology seemed right: It was cheaper, more flexible, and even more secure than any previous technology.*

*This was too good to ignore, so he took the book back home with him and found some people eager to work with this technology. He was enthusiastic, and they did not want to fail on him. His idea was to deliver quickly and in big scale since this would make max profit. He was even ready to share his profit, offering bonus schemes and stock programs to the employees. In his mind this made sense, because it would ensure high motivation to reach the goal.*

*The engineers started off quickly. They were chosen because they were the best in the industry, and they quickly went up in pace, worked long hours, made many decisions. They failed delivering on time. And when they delivered, there was no marketing strategy. And the product was impossible to use. This took a couple of years to cure, and obviously the product failed. And when they finally succeeded it was too late: 20010911 passed, the cash tank went empty, and the whole company (with many other activities) went belly-up.*

**Date**  
2004-03-04  
**Title**  
Shifting focus from technology to product in software development  
**Author**  
Anders Dinsen  
**Document id**  
A-ASYM-1-1.1-en  
**Copyright**  
© 2003, 2004 Anders Dinsen

## 1.2. What went wrong?

The case is interesting in itself. And since it's history, there is little we can do about it – we can't change it. On the other hand, here are three concrete things that were done wrongly in this project:

1. The manager was fascinated by the technology and the potential it had. He did not realise that it took more than technology to make a success. He said it was important to be 'first' and 'best', which is true, but he never defined what to be first with or how to be best.
2. R&D was run by engineers with great competence in developing complex technology. They liked the job of developing a new platform, they liked putting their favourite platforms into work, and the outlook to a wildly growing, successful product.
3. There were no users or existing customers to try the product out on. No close relations with the players on the product's future market to discuss features with.

The result: Technology ruled, product failed.

Later in this paper, I will present a paradigm that does not address these problems one by one, but attempts to change the way of thinking so that they are eliminated.

## 2. Good products

There are many indicators that identify a good product:

- Market success
- High revenues
- Market share
- Users like it
- Usability is high
- Cost of ownership is low
- Aesthetic design
- High quality
- Big potential

We need to discuss these indicators slightly, because if you as a developer or manager wants to focus on the product, you need to know what it is about the product you are focusing on.

Different organisations will have different views on the importance of each of these factors. A startup company, for example, will rate growth, user acceptance, potential and quality as most important, whereas a well established company will tend to focus on market share, revenue, ethics and usability. Personal taste also counts, and finally different lines of business will also weigh the factors differently. A medico technology company, for example, will focus on safety, quality, and usability, whereas a manufacturer of consumer electronics will focus on potential, user acceptance, growth, revenues.

Each project, then, has its own view on what a good product is. This view combined with the features of the new product constitutes what I call the vision of the product.

Do people "know" the vision before they start? No, often they don't, and that leads to problems. The CEO in the case mentioned earlier probably did indeed have a good vision about the product. This vision was probably a source for his enthusiasm. Unfortunately he forgot to pass on this vision to the people defining and creating the product, and maybe he also neglected to verify his vision: Can it be realised at all? Will it have the value I dream of? Many people tend stick to their visions so strongly that they refuse to question them at all.

### 3. Shifting focus: Provocations

One might think that with the end of the dot-com era, no-one will ever again make the error of mindlessly chasing technology at the quickest possible pace. Things are not that easy though, we need to do something to actively shift the focus. A bit of provocation is always good.

*[...] I swear as a director to refrain from personal taste! I am no longer an artist. I swear to refrain from creating a "work", as I regard the instant as more important than the whole. My supreme goal is to force the truth out of my characters and settings. I swear to do so by all the means available and at the cost of any good taste and any aesthetic considerations.*

(From the Dogme95 Vow of Chastity.)

The above sentence is the result of two film makers' (Thomas Vinterberg and Lars von Trier) wish to shift focus from all the stuff making up a film (art, technology, aesthetics...) to the truth – nothing less. This is a very ambitious goal. A goal that I share in the sense that it is impossible to know the absolute truth. But you can get pretty close.

The Dogme Vow is about shifting focus. From the means to the result. And as such, the Dogme95 initiative can serve as inspiration to us software engineers.

Here is another provocation (from the same world):

#### 3.1. Kill your darling

Instructors of theatre and film use this statement to remind them that they easily end up putting all their energy into one particular scene or actor. The statement tells them to cut out the best scene of the play or film, their personal favourite. Or to take out the best actor from the cast.

This is not very engineering-like! Engineers calculates quality of a product as the sum of qualities of the elements making it up. Using that definition, this is the best way to make a bad film!

But it's not.

Love controls us. Turns us into instinct-following little animals. Forces us to forget the whole by directing all of our attention to the item of our love. An instructor with a darling is controlled by that love. What happens is that other scenes tend to "build up" around the "darling" instead of playing their own parts.

I'd like to see this implemented in software development: Try imagining the project manager who lays off his best favourite developer? Or the expert who refuses to apply his expertise? Or a project with '.NET'? Are you ready to do that? Will it help? Sometimes it will.

#### 3.2. Reflect, sleep on it, make slow decisions

The human mind can learn to provide quick answers to questions, but quick answers are known answers, and that is not what we need in software development where innovation is so important. What do you think would happen if in your next project you

decided *not to decide* the software platform for until after one third of the project time?

I can hear engineers complaining: "How can we design when we do not know what we are designing for?" I can argue: "You are designing for the customers, so what is the problem?"

Can that be done? Will it make the product better or worse?

### 3.3. Stop reading books

Some books promise all kinds of good things when you look at the back cover. All you need to do is to read the book and apply everything it says to your work. "It's as easy as 1-2-3!"

Here are some real examples:

"[This book] provides step-by-step information about which methods to use at various stages during the development life cycle. [...] You do not need to have previous knowledge of usability to implement the methods provided, yet all of the latest research is covered."<sup>1</sup>

"Capturing a wealth of experience about the design of object-oriented software, four top-notch designers present a catalog of simple and succinct solutions to commonly occurring design problems."<sup>2</sup>

Like I said before, what they offer is probably good and useful in some applications, but you will not know whether it works or not until you have tried it. Think about it: The guy writing the book is trying to sell what he is writing about. Otherwise he won't make money, and otherwise he wouldn't have been able to do the (big) job of completing the particular book. He was motivated when he wrote it, and his motivation will affect you. Don't let him away with it uncritically.

### 3.4. Communicate directly

Communication between developers is necessary to ensure that problems are foreseen and opportunities discovered. That is probably pretty obvious. But far too often in real development, communication is channelled through a narrow path of a single person.

This is most visibly seen in outsourcing relationships, where both producers and procurers usually establish quite rigid communication facilities. The argument is usually to avoid misunderstandings, but most often there is also a political and contractual limit on what can be said to the other party unless you want to start losing money.

Since managers generally can't trust business to engineers, it's better to monopolize communication with the partner within a single person or department that can be much better controlled.

The funny (sic!) thing is that it is the cause for an enormous amount of misunderstandings! Usually, questions aren't answered in a way that really helps the person

---

1 Jakob Nielsen: Usability Engineering, Academic Press 1993.  
2 Erich Gamma et al: Design Patterns. Addison Wesley, 1993.

asking, and it takes a very long time to get the answers in the first place. So a lot of communication back and forth is needed, often for problems that – if they were asked in person – could be solved in five minutes only!

Think what would happen if a procurer sent off an engineer to the producer, who had the power to make decisions and who would spend his time talking to the real developers and solving the real problems (instead of just the managers)?

Chaos? No, agility!

#### 4. Product focused software development paradigm

After some provocations, I am now ready to present ASYMs software development paradigm.

The paradigm consists of three guidelines:

1. Visualise the end result
2. Simplify problems
3. Evaluate multiple solutions to every problem

##### 4.1. Visualisation

Visualisation is a buzz word these days. "I do not visualise what you say!" and "Please visualise your idea!" are common comments in modern organisations. Forgetting the buzz, assisting the developers in building an anticipation of the end result in a project is a well known management principle.

Although some may claim otherwise, I find that requirement specifications are usually of no help here. Requirement specifications are indeed a necessary management instrument in most development projects, but are of very little help in understanding anything, and never helps building visions. Use cases don't help much either, because they are far too abstract.

Good statements to build up a vision has the following properties:

1. They are qualitative: For example given two good descriptions of end users, it is uoquite easy to hypothetically ask them about their opinions about particular solutions. It may turn out that you do not know what they will say, then you must investigate further, but it usually turns out that it is quite easy to deduct their reactions from the knowledge you have about them.
2. They use everyday language. No abstract or technical language here.
3. They are concrete: "One user mentioned ...", "Marketing needs it by Christmas to ...", "Pete from support mentioned that the function was easy to understand because the button indicated what he should do."
4. They are motivating and enlightening.

The following sentences are examples of sentences used in conversations or presentations that will help developers build a vision of the result in their mind:

- "People quickly learn that it is both easier and more fun to use this web site than to do it in the traditional way."
- "Here are the persons that it helps: [Actual persons are mentioned]"
- "We had a conversation with one of our customers yesterday, and he said ..."
- "We discovered the following problems yesterday: ..."

Specifically it is *not* about making digits and numbers about everything. For example, I don't analyse the problem and tell the developers that "55% of the users are female in the age between 12 and 16." Sentences like that doesn't ring a bell unless you know which females in the given range of age we are talking about. Numbers like that are good for convincing people, not for instrumenting their visions.

It is also *not* about being very abstract: "It is our vision that the entity which is the ob-

ject of our development activities will improve society from an ecological perspective.” Or “We are on our way to something better, an improved life, supported by the product.” Such sentences don't ring a bell either, the first one one could describe anything from rat poison to operating systems, the other one says more about the person writing it than about the product.

## 4.2. Simplification

There are two ways to simplify something:

1. Divide it into smaller parts, each simple
2. Find high level models to explain the complexity

In software engineering, dividing problems into smaller parts is done by component technology or by modularisation. In ASYM, we find that it matters how components are structured:

- We try to limit components to embrace only a single level of abstraction. A data communication component should not work on both the protocol and the link layer.
- 'Responsibility' is a useful term: A component should have a clearly identified responsibility in the system. This enables others to anticipate the functions that it provides.
- 'Tradition' is another useful term. We always find functions and names for components that others can relate to. For example it is a good tradition that 'interfaces' only interface, they have no memory. We never try to incorporate or invent new traditions, except if the old ones are flawed.

## 4.3. Multiple solutions

Every time we face a problem, we look for multiple solutions to it, and judge each solution against what it does for the product.

A good (provocative) alternative is always: “Can we do without?” I have found that it often reveals interesting alternatives, both at the higher and the lower levels of a design.

Making up multiple alternatives is a way of understanding what we term 'the solution space'. A solution space is a mathematical multi-dimensional space which is populated by solutions to a problem. Solutions can be organised according to different parameters (e.g. development time, simplicity of interface, etc) and in that way they are located differently in the solution space.

There are two ways of constructing alternatives:

1. By unstructured means, relying on creativity. Brainstorming is a good and well known method for the unstructured development of ideas.
2. By structured means, relying on the systematic development of alternatives. An example of a structured method is the morphology where the problem is divided into organs and for each organ a number of different implementations are shown. The solution space is populated by all the valid combinations of implementations of the organs.

## 5. Conclusion

Structured development methods advocate structured ways of working, collaboration, precise specifications, just in time development, etc. While a structure is necessary in order to manage any development project, there is more to the project than the structure.

The substance of a project must be the end product. Often it is not, and in ASYM we believe that there are three reasons for this:

1. Software engineers are trained in methods and technologies.
2. A good in-depth coverage of popular methods and technologies is (incorrectly) seen as a secure way to get a job.
3. Engineers do not know how to focus on the end result.

In this paper, we have presented some hopefully eye-opening provocations, and a paradigm that we have implemented in ASYMs way of working on own projects and customer projects.

We find that this paradigm works just as well for a simple software project in which one person is assigned to implement the complete functionality of the product, as in large scale software development that we participate in as a resource.

## 6. Appendix

### 6.1. References

1. Steve C. McConnell: Rapid Development. Microsoft Press International, 1996.
2. M. Poppendieck, T. Poppendieck: Lean Software Development, An Agile Toolkit. Addison Wesley, 2003.
3. M. Myrup Andreasen, Lars Hein: Integreret Produktudvikling

### 6.2. About the author

Anders Dinsen is Master of Science of Engineering from the technical university of Denmark. He is the founder and managing director of ASYM APS, an independent research and development company. He has more than 9 years of experience with software development, both as a developer, designer, architect and project manager. His goal is to contribute positively to improving the quality of information technology products developed by ASYMs clients and thereby help increase their revenues.

