Value Driven Development

—

Delivering more value with less effort

A Leif Trulsson eBook

Leif G. Trulsson

February 2012

www.leiftrulsson.com

Contents

Introduction 4

Some Of The Underlying Problems 6

Project resolution 7

Features and functions used in a typical system 8

Production analogy 9

Changes 11

- Changing requirements 11
- Changes due to defects 12

Complexity 13

Lack of separation of concern 14

Tried Solutions 17

Myths and Misconceptions 19

The Value Driven Development Principles 25

Value Generation 27

How do we measure the value? 30

- The Kano Model 30
- The Net Promoter Score 31

Copyright © 2012 Leif Trulsson. All rights reserved.

Protecting Value 32

Leadership and People 35

Leadership 37

Stakeholders 38

Roles and Responsibilities 39
- The Coach/Manager 39
- The Chief 39
- Our most valuable and critical resources/assets 41

Value Flow Management 53

Waste Management 57

Process Efficiency 58
- Managing constraints 59
- Time-to-Value 60

Conclusion 62

About the Author 64

References 65

Copyright © 2012 Leif Trulsson. All rights reserved.

Introduction

"Creating value should be our highest priority."

The purpose of Value Driven Development is to increase value and profitability and the ability to compete in a world of hyper-competition and with an ever-increasing pace to market.

For years, enterprises have been told that to cope with increased complexity in software development, they need to add more measurements, more controls, more checks and balances, and more rigors.

Studies by the Standish Group shows that 64% of delivered features are rarely or never used. Less than 30% of software development projects are successful in terms of time, costs, and delivered functionality.

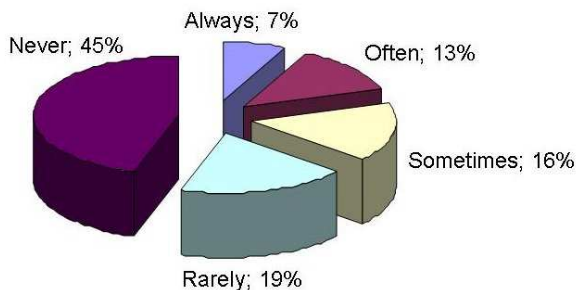


Figure 1. Implemented features and functions usage

Why is this so? Why haven't all maturity, project management, and unified process efforts managed to resolve this dilemma? Because they all fail to focus on the most important thing – **value**!

Copyright © 2012 Leif Trulsson. All rights reserved.

This book will introduce you to Value Driven Development and its main principles. I will start by looking at some of the underlying problems of software development, tried solutions, and flourishing myths, and then I will take a closer look at the main principles behind Value Driven Development.

This book has been in the development for many years and started as a presentation about *Value Driven Development* back in 2005. I hope you find the thoughts and ideas presented herein valuable and inspiring.

Leif Trulsson
Ljunghusen, February 2012

Copyright © 2012 Leif Trulsson. All rights reserved.

Some Of The Underlying Problems

”There is no universal truths except, of course, this one” — David Thewlis

I’ve many times been asked why it’s so hard to develop good user friendly and bug-free software, on time and within budget and with expected functionality. And why it’s so hard to estimate the development effort? Why haven’t software development followed the same trend as hardware development, with its dramatic increase in performance?

There are many answers to these questions and in this and the following chapters about *Tried Solutions* and *Myths* I will take a closer look at some of the problems and some of their causes.

Some of the problems and causes that software development projects often face and run into are for example:

- Project resolution
- Features and functions used in a typical system
- Production analogy
- Changes
    - Changing requirements
    - Changes due to defects
- Complexity
- Lack of separation of concern

”The creation of good software demands a significantly higher standard of accuracy than other things to do, and it requires a longer attention span than other intellectual tasks.” – Donald E. Knuth

Copyright © 2012 Leif Trulsson. All rights reserved.

Project resolution

Less than 30% of software development projects are successful in terms of time, costs, and delivered functionality. There are many factors influencing the outcome of a project but usually it is some of, or a combination of the following factors:

- Lack of end-user involvement and/or organizational commitment
- Lack of requirements and specification analysis
- Changing needs and requirement specifications
- Lack of sponsorship
- Lack of resources
- Immature or unproven technology
- Unrealistic expectations
- Unclear goals and objectives
- Lack of planning
- Best before date expired
- Lack of IT leadership
- Bad attitude between team members/developers

These are the Critical Impair Factors that are explained in more details in *"The Art of Project Management — How To Increase Business Values With Efficient Project Management"* [4].

Is there any recipe that will increase the probability for success? The answer is yes! Even though the chance is less than a third, there are projects that succeed. But as with all recipes, it is how you mix and balance the ingredients that will affect the final outcome. There are also a few project gurus that are masters in managing projects. They are not many, but they do exist and they are worth every penny. The following are some of a project's Critical Success Factors [4]:

- Involve the end-user
- Secure executive sponsorship
- Assign an experienced project manager

Copyright © 2012 Leif Trulsson. All rights reserved.

- Define clear business needs and objectives
- Minimize scope
- Build on standard software infrastructure
- Use proven methodology
- Foster team spirit
- Choose technology not competence
- Pull the plug if necessary

Features and functions used in a typical system

Studies by the Standish Group shows that 64% of delivered features are rarely or never used. The Pareto principle tell us that 20 percent of the features delivers 80 percent of the perceived value, so why do we still invest time and effort in adding the features that are rarely or never used?

A few years back I was head of development at a software company in Sweden. One of the reoccurring issues was with one of the sales representative who repeatedly came up with new features that had to be in the next release; otherwise he wouldn't be able to sell our product to his customers.

The *must-have* feature list grew longer and longer, and in an effort to try balance our limited development resources against business priorities and possible revenue streams, I asked the sales man to produce some tangible figures on expected return on investment. He couldn't. And in the end he didn't manage to sell to any of his potential customers, and we ended up with features that none of the existing customers had asked for. In fact, some of our existing customers expressed concerns about having sponsored features that they didn't need or wanted instead of more pressing needs.

This story shows how easy it is to loose track between what is **needed** and what is wanted. It is better to deliver features and functionality that

Copyright © 2012 Leif Trulsson. All rights reserved.

satisfies the need 80 percent of the time. And then, if any extra features are needed, have a customer pay for the extra development effort, or add it to the next release or version of the product when you know that an existing customer needs it.

Production analogy

Building software is not like building bridges or houses. Many attempts have been made to view software development in terms of traditional engineering terms. Through the years, efforts have been made to mimic traditional engineering processes, but these efforts have just failed in delivering on promise.

In traditional engineering, you would develop a detailed plan of activities, track that plan's execution, and adjust for variances between planned performance and actual performance. The plan's quality was considered to be in direct relation to its level of detail. Loads and tensions follow the laws of physics, and you would know before the bridge was finished what level of environmental forces the bridge would be able to withstand.

The tools and equipment used to build the bridge would be the same through out the building process. Also, the properties of materials and maturity of building codes and practices would not change during the project. So the success of the project would more depend on proper resource management and proper execution of the plan.

It is very tempting to compare software construction to the building of a bridge. Both projects involve requirements management, design, construction, scheduling, special teams, and inspections.

But in software development, these traditional methods just do not work. In fact, they are the reason why many software-development projects fail or are challenged. Traditional sequential, activity-based construction approaches (following what we call the waterfall model) just do not deliver. The success rate (meeting the time frame and

Copyright © 2012 Leif Trulsson. All rights reserved.

budget) for software projects following the waterfall model is about one in ten.

Software Development is, as the words say, **development.** Product development is completely different from production (see Table 1). Product development is very much a mental process, especially if the development effort is exploring new fields of application. Software development is an even more mental process, as the software under construction is very hard to visualize, especially if it's a complex real-time system with many parallel processes. Software development is (or should be) a highly iterative and incremental process, where the required functionality is gradually grown into a full-fledged application or system.

Development	**Production**
Designs the Recipe • Quality is fitness for use • Variable results are good • Iteration generates value	*Produces the Dish* • Quality is conformance to requirements • Variable results are bad • Iteration generates waste (called rework)

Table 1. Development vs Production

Not only software projects run into trouble. In southern Sweden we have been witnessing an epitomical example of a projects nightmare — the black-hole of tunnel digging. For a number of years, the state owned company Banverket, responsible for railway tracks in Sweden, has been trying to penetrate the mountain and ground of Hallandsåsen.

Hallandsåsen is a ridge between the provinces of Scania and Halland and the digging through this ridge has turned out to be a tremendous challenge and nightmare for all involved stakeholders. The mountain has turned out to be very soft and porous, which has caused all sorts of troubles; including a poison scandal affecting rivers and waters on the ridge. We have also seen the dramatic decrease in water levels causing many local water wells to run out of water.

The project is still under way and nobody really knows when it will finish and what the final cost will amount to.

Copyright © 2012 Leif Trulsson. All rights reserved.

Changes

"The only constancy is change itself" — Fred Brooks, The Mythical Man-Month

The one thing that we can always be sure about is that things will change, and we need to be able to handle those changes. What differentiates software from most other produced entities, is how relatively easy it is to change. When it comes to the hard stuff of man-made constructs like buildings, cars, bridges, or tunnels, change can be very hard and many times very costly.

Every now and then we hear about cars being called-back and we might consider some remodeling on our house. These changes differ greatly from software changes in that they are most of the time easily understood and controllable and even touchable, and can sometimes be quite hard to accomplish. Software changes on the other hand, are easily performed, but not so easily understood, and many times not so easily controlled.

Another differentiating factor is the frequency of change. Software changes occur on a much more frequent rate than hardware changes and are much easier done. And as Fred Brooks [1] points out, the software in a system embodies its function and is made up of pure thought-stuff that is infinitely malleable. Software is embedded in a mix of its application and use, users, regulations, and hardware. This environment is in continues change and in turn force change upon software.

- **Changing requirements**

Changing requirements is as certain as tomorrow comes after today. Business needs changes, technology changes, and regulatory requirements changes. Companies are merged, divested, and acquired; thus spurring new needs and requirements.

Studies by Ralph Young [7] show that 85 percent of defects in software are due to inadequate or bad requirements. Other studies by

Copyright © 2012 Leif Trulsson. All rights reserved.

Ivy Hooks and Kristin Farry [8] show that common types of requirements errors are:

- Incorrect assumptions (49%)
- Omitted requirements (29%)
- Inconsistent requirements (13%)
- Ambiguities[1] (5%)

- **Changes due to defects**

”*Everything that can go wrong will go wrong*” — *Murphy’s Law*

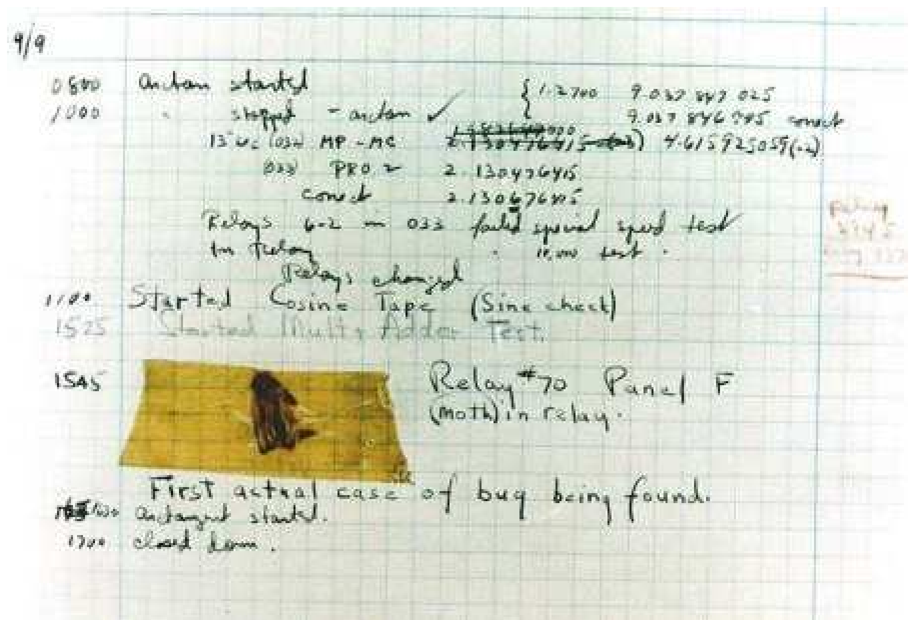


Figure 2. The first ever found bug in a computer

Who hasn’t heard about a bug in a computer? The term is coined from a moth being trapped in a relay on an IBM Mark II in 1946 (see Figure

---

[1] A word, phrase, sentence, or other communication is called **ambiguous** if it can be reasonably interpreted in more than one way. The simplest case is a single word with more than one sense: The word "bank", for example, can mean "financial institution", "edge of a river", or other things.

Copyright © 2012 Leif Trulsson. All rights reserved.

2). And stemming from this first bug, today we call errors in a program or system a bug.

Many attempts have been made to minimize the occurrence of bugs or defects in software. The late Harlan Mills suggested the Clean room [9] concept, which reflects an emphasis on defect prevention rather than defect removal. The name "Clean room" was taken from the electronics industry, where a physical clean room exists to prevent introduction of defects during hardware fabrication.

The cost of defects varies from discretionary money to the cost of lives. A defect in an air-traffic-control system or a missile guidance system can have devastating effects. Whereas NASA's $150 million Mars Lander mission with a faulty measurement conversions “bug” was the loss of some serious money.

Complexity

“The challenge over the next twenty years will not be speed or cost or performance; it will be a question of complexity.” — Bill Raduchel, Chief Strategy Officer, Sun Microsystems

In the case of the Mars Lander the bug was caused by a logical error. In other cases bugs often appear due to shear complexity. Software systems are much more complex for their size than any other human constructs. In a software system no two parts are alike, and software systems have orders of magnitude more states than any other man-made devices, machines, or even computers. This is also why software systems totally differ from other constructs like buildings, computers, or cars, where we have an abundance of repeated parts.

Brooks suggest that, “The complexity of software is an essential property, not an accidental one” [2]. Meaning, the essence of software entities being constructs of interlocking concepts like data sets, relationships among data items, algorithms, and invocations of functions. The conceptual construct is abstract and can be represented in many ways. We can raise the level of abstraction, but we can never

Copyright © 2012 Leif Trulsson. All rights reserved.

abstract away the complexity. When software grows, complexity grows in a non-linear fashion as the interactions between all the different parts also grow in a non-linear fashion.

Grady Booch observers that the "inherent complexity derives from four elements: the complexity of the problem domain, the difficulty of managing the developmental process, the flexibility possible through software, and the problem of characterizing the behavior of discrete systems" [10].

Today, we can model almost anything. With software we can model the weather, how collisions affect the superstructure of a car, and how molecules interact in complex molecular structures. Software has allowed us to create virtual realities that are used to support doctors performing remote surgery and moviemakers to resurrect the dinosaurs, to create the world of Star Wars and Star Trek, and the visual effects in the Lord of the Rings trilogy, Avatar and the Matrix movies.

Large software systems are the most complex man-made entities, they span over a multidimensional domain, they are invisible, and possesses inherent complexity.

"Everything should be as simple as possible, but no simpler." — Albert Einstein

Lack of separation of concern

When we design our systems, we try to do it with a clear separation of concern, so that what happens in one part of the system has no or minimal impact on other parts of the system. And still, all too often this is where it starts to go wrong.

Nearing the end of my first year at Malaco, I got a request for change in our sales-force system. It wasn't a big change; it "just" involved a change to the serial numbers of our sales-force. As it turned out, the Swedish sales-force had serial numbers based on geographic region and market and customer segment. In other words, we had built in

Copyright © 2012 Leif Trulsson. All rights reserved.

> logic in the sales-force serial numbers.
>
> I was stunned. To have this kind of built in logic in the core information might be valid in a closed system, like representing the dates of the year, but not in an opened-ended system where we didn't know much about tomorrow, and much less about the next year and the year after that. But as we all know, not even the "well-known" domains come without surprises — who haven't heard about the year 2000 problem.
>
> In the case of our sales-force system, changing the sales representative's serial numbers every time we reorganized our sales organization caused rippled effects in our information databases. The information in the sales-force system together with the information in our order system was the basis for our executive information system. This in turn meant that every time the serial numbers changed, we changed the history — we couldn't do comparisons between different years based on the serial numbers. It was just like if you every year decided to reshuffle everyone's social security number. Not a very good idea!

A clear separation of concern means separating the business logic from the information model and the presentation model. This is also why you should program to an interface, not an implementation and favor object composition over class inheritance.

One of the top issues for Chief Information Officers is integration. And tight integration is what many enterprise resource planning (ERP) system vendors use as a selling argument. But integration cannot and must not come on the expense of a clear separation of concern. And just as you don't want to replace the kitchen just because you need to replace the microwave oven, you don't want to upgrade or replace your whole system just because you want to replace a small module.

But the lack of a clear separation of concern is in reality causing a lot of aggravation and costs a lot of money every year. The one thing that affects us the most and that everyone has heard about is the threat from vicious computer viruses. And one of the main reasons that these viruses cause such harm is the lack of a clear separation of concern in the Microsoft Windows products. But some might argue, that viruses

Copyright © 2012 Leif Trulsson. All rights reserved.

can affect other operating systems as well, and the reason that Windows is more prone is that there are so many of them.

Yes and no. A virus in a Linux or Unix system could potentially cause some damage but only if the virus has access to the root account. And in mainframes or systems like the IBM AS400, viruses are unheard of because these systems posses a very clear separation of concern. In fact, the best ever clear separation of concern I've seen is IBM's VM System, the ancestor of all virtual machine systems. In the VM System you can run other operating systems inside the VM System and totally unaware of each other. Every virtual machine is a complete mimic of the underlying hardware, and the system that you run under the VM System believes that it is running on a stand-alone machine. Internally, IBM has been using the VM System to develop and test new versions and releases of mainframe operating systems almost since the inception of the VM System back in the sixties.

With Linux and the Open Source there is an exiting twist here. With a single mainframe running the VM System, you can run tens of thousands of Linux systems. Thus, reducing the management, service, and support issues involved in a server-farm. And it's just a matter of seconds to create a new Linux system.

Copyright © 2012 Leif Trulsson. All rights reserved.

Tried Solutions

"There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity" — Fred Brooks [2]

In the Introduction chapter I mentioned that for years, enterprises have been told that to cope with increased complexity in software development, they need to add more measurements, more controls, more checks and balances, and more rigor**.** Significant resources have been spent on process compliance, up front planning, and associated change request processes. And still there is no proof that the organizations that have invested most heavily in methodology and process are the major beneficiaries. The reason for this is that we are encountering a number of paradoxes about processes:

When process is improved, work becomes harder. Brook's [2] thesis about the *essence* of software development, says that there is an irreducible core of work that is not subject to improvement and that cannot be mechanized. Process does not help, it just adds to the work. This is also one of the key findings among the Agile methodologies and one of the key principles of the Agile Manifesto – "*Individuals and interactions over processes and tools"*.

Process focus tends to make an organization risk averse. Better safe than sorry doesn't always deliver the expected outcome. In today's rapidly changing world, agility is the name of game, not rigidity. Time-to-market is a key leverage in fierce competition. Having an ISO or CMM certification can be a key to entry, but not a market advantage.

Copyright © 2012 Leif Trulsson. All rights reserved.

You can have the best ISO certified process manufacturing life-vests made of concrete, but your product will probably not save many lives.

The promise and difficulties of software reuse. Software reuse and object-orientation have claimed promises of productivity gains. But as Brook's [2] points out, they can only "remove all accidental difficulties from the expression of the design", not the essential of the design. There is no correlation between the success of an application and the libraries or classes being used. Investments in infrastructure code on the other hand, can save a lot of time and money. Estimates have been made that infrastructure code make up about 70 percent of the total code. So why invent the wheel over and over again when you can buy it from someone else?

Our ability to adapt to rapid change compared to slow change. Change is inevitably.

In the early and mid eighties I was working for IBM in Saudi Arabia. Many of the countries in the region had seen some tremendous change in just a few decades especially in the standard-of-living and in terms of technology advancements. But culturally, socially, and religiously they were still in the middle ages.

When we were living in Al Khobar and we wanted go to the beach, we had to take our four-wheel drives and go off-road to find a spot where the Muttawa, the religious police, couldn't find us. Because in Saudi Arabia women where not allowed to show any naked skin in public. Most western women like to tan their skins — my wife being no exception, but the penalty for being caught doing so was sever, and not something taken lightly.

Humans tend to find it easier to adapt to changing technology than to changing social structures. Technology often follows a continuous flow of changes, whereas social structures tend to go through stepwise changes, like revolutions or wars.

Copyright © 2012 Leif Trulsson. All rights reserved.

Myths and Misconceptions

"I think there is a world market for maybe five computers" – a comment attributed to Thomas Watson, Chairman of IBM, 1943

The above quote is probably one of the most widespread myths and die-hard quotes ever fabricated. Kevin Maney, the author of *"The Maverick and His Machine"*, found no such evidence in his research for his highly acclaimed book, and he went through all the archives of what Watson said and wrote.

Like the Watson quote, there are a number of myths and other misconceptions in regards to software development. In the following I will take a look at some of the flourishing myths and misconceptions that sometimes tend to become truths.

The Man-month: Brook's Law [1] states: *"Adding manpower to a late project makes it later."*

As Brooks [1] points out, men and months are only interchangeable when a task can be partitioned among them and they don't need to communicate between each other. Like carrying bricks from point A to point B or harvesting grapes in a wine-yard. But it's not true for tasks that cannot be partitioned. Carrying a child takes nine months, no matter how many women you assign to the task. The same is true for many tasks in software development because of their sequential nature. You code before you test and debug.

The load of communication and management also affects the tension and efficiency. As the number of people increases, so does the need for administration, communication, and management. Intercommunication

Copyright © 2012 Leif Trulsson. All rights reserved.

is severely affected by adding more people to a software project. As Brooks [1] shows — *"If each part of a task must be separately coordinated with each other part, the effort increases as n(n-1)/2"*.

Attempts have been made to invalidate Brook's Law, but nothing in my own more than thirty-four years of professional experience contradicts it. Though, there are ways to moderate the burden of the added administration, communication, and management. Keeping the teams small and limiting the communication channels is one way, which I will talk about in a later chapter.

Early specifications reduce waste: We need specifications there is no doubt about that. The question is rather what level of detail that we should provide upfront. We grow software and as I said earlier about designing the recipe in the chapter about Production analogy; quality is fitness for use, variable results are good, and iteration generates value. This means that we early on should define the boundaries, but not the complete inner workings. These are grown into fit for use and purpose through the iterative and incremental process and continuous feedback from the user.

A well-written, comprehensive requirements specification is all you need! This is another misconception about specifications. As I've mentioned before, 85 percent of defects in software are due to inadequate or bad requirements [7]. And one of the major reasons for this being to detailed requirements specifications. A requirement specification should describe *what* is required, not *how* it should be implemented, but all too often this is what happens.

Also, it's much better to create some kind of prototype and let the user try it out and provide feedback. For most people, learning is improved if we can test or try the product or process at hand. You don't learn how to fly or drive a car by reading the specifications for the vehicle.

Formal system specifications eliminate problems. The following quotes says it all:

Copyright © 2012 Leif Trulsson. All rights reserved.

"The Requirements Uncertainty Principle: For a new software system, the requirements will not be completely known until after users have used it." — W.S. Humphrey

"Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence." — Edsger W. Dijkstra

"There is no such thing as an absolute proof of logical correctness. There are only degrees of rigor, such as 'technical English,' 'mathematical journal proof,' 'formal logic,' etc., which are each informal descriptions of mechanisms for creating agreement and belief in a process of reasoning." — Harlan D. Mills

Formal methods do not find all gaps in understanding! It is very difficult for the user to recognize the lack of functionality. It's like the customer who wants to buy a car, but has never heard about automatic transmissions, and as long as the car salesman doesn't inform him about the existence of such a "wonderful" function he will never ask for it.

Theory of Games says – *"There's no point in using exact methods where there's no clarity in the concepts and issues to which they are to be applied."*

Predictions create predictability: Try the following assertion: The number of defects in a system is a good predictor of the systems reliability. In other words the fewer the defects we find in the system, the greater the reliability of the system will be. Not likely! There is no evidence that we can measure defects during development as evidence of code reliability and use this to calculate the likelihood of reliability.

"I can build a reliable system with thousands of bugs, if you let me choose my bugs carefully." - David Parnas

Haste makes waste: For anyone who has ever witnessed a F1 or Indy 500 team handle a pit-stop know this isn't true. In less than ten seconds 10-12 people change tires, fill-up the tank, and serve the car and driver.

Copyright © 2012 Leif Trulsson. All rights reserved.

The speed of the pit-stop team is a key competitive advantage, and a lost second can mean the difference between victory and defeat.

Second place is the first looser, and for most companies time-to-market is a matter of survival. Remember, it's better to be 80 percent right than 100 percent wrong.

Get it right the first time: When John Opel took over as CEO at IBM he proclaimed the *"Right the First Time"* principle. Before that, *"THINK"* had been the guiding principle ever since Thomas Watson Sr. The paradox about the *"Right the First Time"* era is that this is the time when things started to go really wrong within IBM.

This is when IBM laid the foundation for companies like Microsoft and Intel, and when it changed its revenue principles from rental to selling only — a move that totally changed IBM's cash flow and revenue streams, and that together with lack of customer focus brought IBM to its knees.

Of course there are moments when *right the first time* is crucial: when you are parachuting, bungy jumping, or doing anything else where you only get one try. Developing software is not one of these one-time-shots. Software development is as the word says **development**, and sometimes this tends to be forgotten.

There is one best way: There will always be promises of the wholly grail. But it is more like everybody is talking about it, but nobody has seen it. There is no "grand unifying theory" in software development and no "one-size-fit-all". There can only be guiding principles and you have to adapt these principles to your own needs and circumstances. For example Extreme Programming (XP) requires you to do pair programming. But pair programming doesn't work for everybody in the long run. Personally, I'm very reluctant to do pair programming for longer sessions — it just doesn't fit my personality and need to control the creative process.

All programmers are the same: Already in their *"An exploratory investigation of programmer performance under online and offline*

Copyright © 2012 Leif Trulsson. All rights reserved.

conditions. SP-2581, System Development Corp., Santa Monica, Calif., Sept. 2, 1966", Sackman, Erikson, and Grant revealed that programmers with the same level of experience exhibit variations of more than 20 to 1 in the time required to solve particular programming problems. Other studies show similar results and my own experience show that the variance can be as much as much as 100 to 1.

You will also find large variations among experienced programmers. Experience will only take you this far, the rest depends on talent and talent comes in many variations. Some are just better at problem determination, conceptualizing, and making the logical constructs that are required.

The program is 95% done! I once had a colleague who when asked about completeness of a job always answered — 70 percent is done. After a while we found out that he was always "70 percent done", whether he had started or not. Some people are more optimists than others and programmers are probably among the most optimistic people in the world. Some programmers believe in their ability to solve any problems on the fly and instantly even in the face of continued, repeated evidence of the contrary. But the only true evidence of the job is done is working code — period!

Tools will solve all our problems: Ivar Jacobson, the father of Use Cases, wants us to believe that what we need is the "right" tool providing explicit knowledge to inexperienced programmers. Coding he says — "is a no-brainer work"; anyone can do it with the right tool.

This is in stark contrast to my own experience and what Brooks [2] says – "There is no silver bullet!" The essence of building software is devising the conceptual construct itself. This is very hard. Even the best tool will not make a good programmer out of a bad one. In fact, a bad programmer will use good tools to turn out worse programs more quickly than ever before.

A high CMM rating makes you a good software development company: As I showed in the chapter Tried Solutions, there is really no correlation between a certification and the end result. The same

Copyright © 2012 Leif Trulsson. All rights reserved.

holds true for any form of maturity or other ratings. A CMM rating is only a prediction – there is no experimental verification of the prediction. Just as with the life-vests made of concrete, if an organization is good it will score high, but if an organization scores high doesn't mean its good.

The prototype can always be extended: I like prototypes, but sometimes prototypes can be deceptive. They can give the end-user a false impression of what the end-result will look like. Also, extending a prototype is a good way to preserve lousy design decisions. Sometimes it's better to use storyboards or other methods of visualization just to avoid the risk of an early prototype ending up as the final design.

Copyright © 2012 Leif Trulsson. All rights reserved.

The Value Driven Development Principles

"Simple rules and purpose give intelligent behavior, complicated rules give stupid behavior."

We have looked at some of the underlying problems, we have looked at tried solutions, and we have looked at myths and misconceptions. So what do we need?

We need a no-nonsense way to deliver more value with less effort and less friction!

We need some principles that are easy to understand and easy to implement. We need principles that allow us to address the underlying problems. We need simple principles that help us to increase value and profitability and the ability to compete. We need principles that help us to deliver more with less!

We need principles that focus on:

- Value Generation
- Leadership and People
- Value Flow Management

These are the main principles of Value Driven Development. They are well known and they have been around for a very long time. They are proven and they work.

In the following chapters I will take a closer look at these main principles of Value Driven Development and the *why, what, who,* and

Copyright © 2012 Leif Trulsson. All rights reserved.

how of these principles. **Why** we should use them, **what** they focus on, **whom** they involve, and **how** they are realized.

Copyright © 2012 Leif Trulsson. All rights reserved.

Value Generation

"Value is determined in the eyes of the beholder."

Value, by definition, is a fair return in goods, services, or money for something exchanged, the monetary worth of something, or something (as a principle or quality) intrinsically valuable or desirable. And according to generally accepted accounting principles, fair market value is quantified based on the perceived value of that which is given up in exchange for that which is received. The perceived value is relative and only the recipient can assess the relative value in something he or she receives.

Obviously, we can value something we provide for someone, but most likely we are doing so from our own perspective. In other words, this is relative to our own perception and usefulness to us. What this means is that the usefulness to us doesn't matter; the receiver of the value, not the giver, is the one whose assessment of value is what matters.

Value Generation focuses on:

- Why?
    - Because we want to increase value and profitability and our ability to compete
- What?
    - Focus on value
- Who?
    - Focus on who we serve and the people involved in the process
- How?
    - By realizing that the customer defines the value, and it is the perceived value that matters.

Copyright © 2012 Leif Trulsson. All rights reserved.

- By understanding that value is what the customer defines and is willing to pay for or is willing to promote to others!
- By building on both conceptual and perceived integrity
- By doing Value Assessment
- By managing our most valuable assets
- By Value stream mapping
- By delivering Value faster

"The customer is king."

In 1984, on our way to Australia, we stayed a couple of days in Kuala Lumpur in Malaysia. After a nice dinner one evening, I asked the waiter if they had any cigars. He nodded and disappeared. After a while he returned together with an assistant and a small table with various utensils.

What then unfolded was a superior performance in how to prepare a cigar for smoking. The waiter and his assistant rolled, massaged, and heated the cigar over open flame. We were all spell bound by the performance, and as a grand final the cigar was lit from the open flame and then the tip was quickly dipped in some sweet liqueur, and was then handed over to me.

This is the best cigar experience I ever had.

What the waiter and his assistant in Kuala Lumpur provided was a total experience. They were delivering a packaged solution that went far beyond my expectations, and in return they got a very satisfied customer.

For example for an IT organization, the business and its users, customers, and partners are the customers. The challenge for the IT organization is to deliver a total experience that has both real and perceived value. It doesn't matter what we think of our solutions and services, as long as the end-users don't perceive them as providing value. If systems get more in the way than provide support in the end-

Copyright © 2012 Leif Trulsson. All rights reserved.

users daily work, the end-user will soon find a way to shortcut the system. And if the services and solutions provided are not perceived as providing any real value, outsourcing and other more threatening issues will soon be raised.

Customer satisfaction is what matters. Customer satisfaction has nothing to do with the latest or hottest technology or fad. Customer satisfaction is about the customer's perception of qualities like usability, response times, fit for purpose, and level of service. If the customer is satisfied, then we have delivered **the right product, solution, or service at the right time**.

The Fans

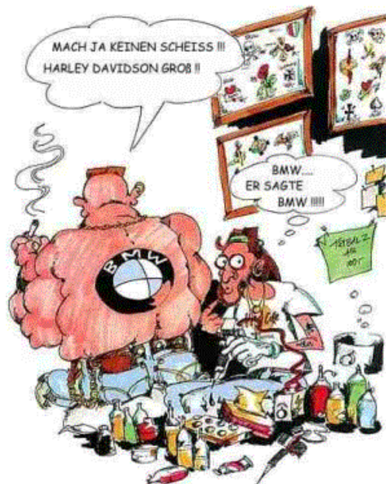


Figure 3. Products with high-perceived value have fan clubs

Copyright © 2012 Leif Trulsson. All rights reserved.

So what is value?

Value is:

Everything that *improves* the fit, form, or function of the product or service and **what the *customer* is *willing* to pay for or promote to others**

How do we measure the value?

Value can be measured in a number of ways and here I'm just mentioning two models: The Kano Model and The Net Promoter Score.

- **The Kano Model**

Developed in the 80's by Professor *Noriaki Kano*, the Kano model is based on the concepts of customer quality and provides a simple ranking scheme, which distinguishes between essential and differentiating attributes. The model is a powerful way of visualizing product characteristics and stimulating debate within the design team.

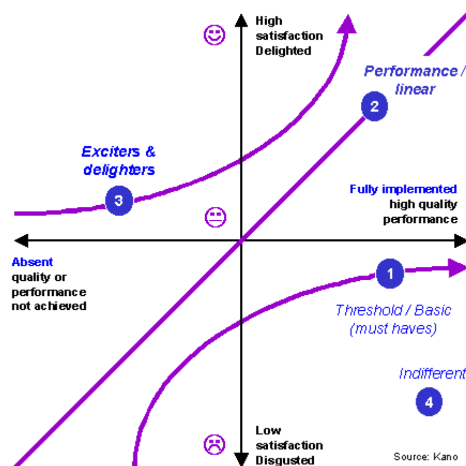


Figure 4. The Kano Model

Copyright © 2012 Leif Trulsson. All rights reserved.

Kano also produced a rigorous methodology for mapping consumer responses onto the model. Product characteristics can be classified as:

Threshold / Basic attributes
Attributes which must be present in order for the product to be successful, and can be viewed as a 'price of entry'. However, the customer will remain neutral towards the product even with improved execution of these threshold and basic attributes.

One-dimensional attributes (Performance / Linear)
These characteristics are directly correlated to customer satisfaction. Increased functionality or quality of execution will result in increased customer satisfaction. Conversely, decreased functionality results in greater dissatisfaction. Product price is often related to these attributes.

Attractive attributes (Exciters / Delighters)
Customers get great satisfaction from a feature - and are willing to pay a price premium. However, satisfaction will not decrease (below neutral) if the product lacks the feature. These features are often unexpected by customers and they can be difficult to establish as needs up front. Sometimes called unknown or latent needs.

- **The Net Promoter Score**

Fred Reichheld's Net Promoter Score, shows with one number what generates value. It was introduced by Reichheld in his 2003 Harvard Business Review article *"The One Number You Need to Grow"*[11]. The benefits of this method lie in the simplification and communication of the objectives of creating more "Promoters" and less "Detractors. In addition, the Net Promoter method reduces the complexity of implementation and analysis frequently associated with measures of customer satisfaction. As such, the Net Promoter Score provides a stable measure of performance that can be compared across business units and even across industries, and increasing interpretability of changes in customer satisfaction trends over time.

Copyright © 2012 Leif Trulsson. All rights reserved.

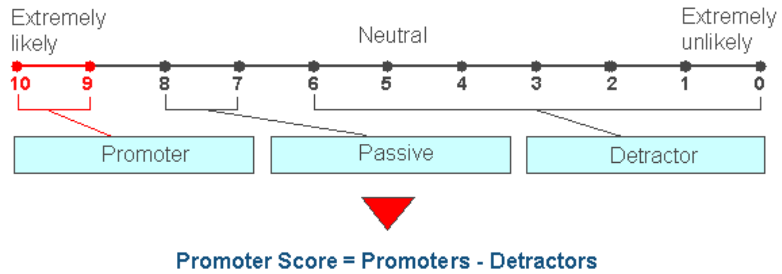


Figure 5. The Net Promoter Score

For a development team the Net Promoter Score might be just a little bit too simplistic in its approach and the Kano Model might give you a better feeling for what features and functions should go into the product under development. Another important aspect is time-to-value which I will come back to in the chapter Value Flow Management.

Protecting Value

The cost of change varies and the cost of change for *High Stake Constraints* increases dramatically as time progresses. Whereas other changes, like simple modifications like repositioning a user interface logo or a button, have a low cost-of-change penalty.

Copyright © 2012 Leif Trulsson. All rights reserved.

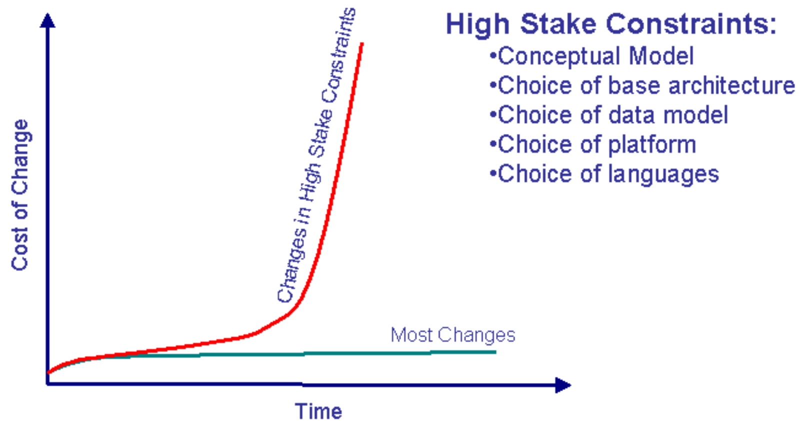


Figure 6. Cost of Change

To protect the value and ensure value creation, we need to build on conceptual integrity and clear separation of concern. Conceptual integrity is the most important consideration in system design. Conceptual integrity means hiding the "how things are made to happen" from "what happens". It means achieving a high level of ease-of-use compared to functionality. In other words achieving a high level of transparency between the user interface and the workings underneath the covers. Thus conceptual integrity goes hand in hand with clear separation of concern. E.g. the conceptual model of the analog clock displays a very high level of conceptual integrity and a clear separation of concern.

As requirements change, so do design and code. The essence of building software is devising the conceptual construct itself. This is very hard because we are dealing with:

- Arbitrary complexity

Copyright © 2012 Leif Trulsson. All rights reserved.

- Conformity to given world
- Changes and changeability
- Invisibility

By building on a high level of conceptual integrity and clear separation of concern, we are in a better position to handle these challenges.

Just as you don't want to change your whole kitchen just because you need to replace the microwave oven, you don't want to change your whole system because of a minor upgrade or change request.

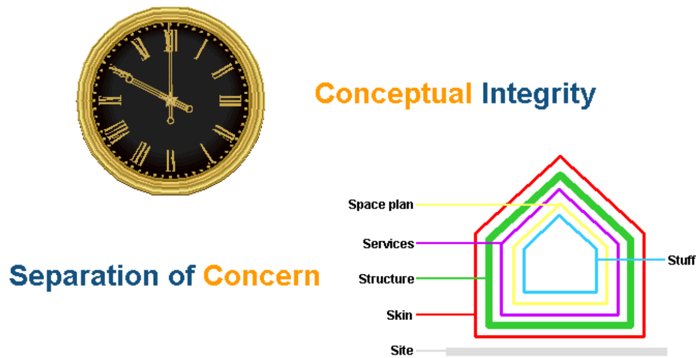


Figure 7. Value Protection

Copyright © 2012 Leif Trulsson. All rights reserved.

Leadership and People

From clocks, conceptual integrity, transparency, clear separation of concern, and usability – leads us to the human aspect of software development.

Leadership and people focuses on:

- Why?
    - Because people are the ultimate source of value
- What?
    - Focus on leadership, competence, empowerment, team performance, commitment, execution, reflection, learning
    - Focus on leadership through others
    - Focus on clear and simple rules and goals
- Who?
    - All stakeholders
- How?
    - By building on trust, respect, honesty, and reward
    - By recognizing that experience, skill, and knowledge matters
        - 97% of all successful projects have an experienced PM at the helm
        - The need for an Chief Architect, responsible for the conceptual design and the advocate of the customer
        - The Chief Programmer (Mills)
    - By understanding that the project manager's/team leader's primary responsibility is to create an environment in which the team can excel in value delivery performance

Copyright © 2012 Leif Trulsson. All rights reserved.

  - By understanding that value is created in close collaboration with all stakeholders
  - By understanding team dynamics – the pairing of masters and apprentices
  - By celebrating team success

Tom DeMarco says:
”the very best technology never has as much impact as girlfriend or boyfriend trouble.”
”...the project’s sociology will be more important to eventual success and failure than the project’s technology.” The sociology and politics of the team will make or break the team.

Jumphrey, Kitson, and Kasse report that:
”for low maturity organizations, technical issues almost never appear at the top of key priority issues list, ...not because technical issues are not important but simply because so many management problems must be handled first.”

Fred Brooks says it to:
”People are everything”

- The issues are managerial, not technical
- Every study shows the crucial importance of people
- Projects don’t move; only goals move!

”the central question in how to improve the software art, centers, as it always has, on people.”

Barry Boehm – *”Personnel attributes and human relations activities provides by far the largest source of opportunity for improving software productivity.”*

Given a choice between investing in talented, expensive people and good, expensive tools, go for the talented people even though they are more expensive than the expensive technology.

Copyright © 2012 Leif Trulsson. All rights reserved.

Watts Humphrey, who wrote the book on software processes says: *”While technology offers considerable potential for improvement, in many organizations the software process is sufficiently confused and incoherent that non-technological factors impede the effective application of technology.”*

What this means is that a team of highly competent programmers who are also highly territorial, egotistical politicians will fail while a team of equally competent programmers, who are also ego-less, cooperative, team players will succeed.

Leadership

Tom Peters, author, speaker, learner, and listener, says that leadership is the scarcest commodity. I couldn't have said it better myself. Time and talent are the most critical resources now and in the future. And among the talents, it's leadership talent that is the scarcest and most critical resource.

Everybody wants great leaders, but great leaders are hard to come by. Great leaders are shaped by circumstances; they grow into their leadership roles. You can train people in management and leadership, but still that doesn't mean that they will be great leaders. Also, great leaders are not necessary great leaders on all levels.

A leader’s chief responsibility is to rally people to a better future. Great leaders find what is universal and capitalize on it.

”Clarity is the answer to anxiety, so effective leaders are very clear.”

Great leaders provide clear answers to the following four key questions:

1. Who do we serve?
2. What is our core strength?
3. What is our core score?
4. What actions can we take today?

Copyright © 2012 Leif Trulsson. All rights reserved.

Great leaders also develop three important disciplines:

1. They muse
2. They pick their heroes with great care
3. They practice their words, phrases, and stories

Stakeholders

There are different stakeholders involved in the development effort. These stakeholders have different voices, requirements, and expectations. To create true and lasting value we have to take into account the voices of all involved stakeholders.

The voice of the end-users:

- Provides the input/requirements to the development process
- Uses the developed application or system
- Are affected by the application or system

The voice of the sponsor:

- Sets the vision
- Provides the funding
- Determines priorities

The voice of the technical community:

- Develops the software
- Runs the software
- Maintains and supports the software

Copyright © 2012 Leif Trulsson. All rights reserved.

Roles and Responsibilities

The development effort requires different roles and responsibilities depending on size and exposure/criticality of the project. Depending on the size, we need different levels of leadership just like for example:

- Jazz Ensemble - Set the tempo and start the piece; rotates among members
- Jazz Band - Direct the band, but not the featured instruments
- Orchestra - Direct the orchestra and interpret the piece

The Coach/Manager

As a manager, I have always seen my own role as a facilitator of the best possible environment and tools for the team to be able to perform their best. This also means that I sometimes have had to shield the team from a "hostile environment".

The chief responsibility of the coach/manager is to turn a person's talent into performance. Great coaches/managers:

- Finds what is unique about each person and capitalize on it
- Focuses on strengths
- Releases peoples talents

• The Chief

The Chief Architect/Designer/Programmer, the master who provides the technical leadership and guidance based on talent, skill, and experience. Some development methodologies do not focus on the chief role and in some cases, like Scrum, doesn't advocate this kind of role. Even though I'm a certified Scrum Master, I personally believe in the role of the chief architect/designer in development projects. You need someone who is responsible for and owns the overall picture and the conceptual integrity.

In his book "*The Mythical Man-Month*"[1], Fred Brooks elaborates on how to break down a large project into smaller, more manageable

Copyright © 2012 Leif Trulsson. All rights reserved.

teams. And he proposes a solution that he calls "*The Surgical Team*", a concept originally proposed by Harlan Mills[2]. Mills proposed that each segment of a larger effort is taken on by a smaller team, much like a surgical team, as Brooks notes. The team is in Brooks' analogy led by "*The surgeon*", and what Mills called a *Chief Programmer. "He personally defines the functional and performance specifications, designs the program, codes it, tests it, and writes its documentation."*[1]

As Brooks notes, in conventional teams the work is divided among the developers and each is responsible for design and implementation of their part of the work. This is also true for pair programming, frequently used in several Agile methodologies. But, as Brooks notes, the differences in judgment, interest, and interpretation of the overall strategy has to be settled which often leads to a compromise. In the surgical team, there is no such conflict of interest, and it also ensures the conceptual integrity of the work.

At Toyota the Chief Engineer is responsible for:

- Business Success
- Deep Customer Understanding
- To develop the Product Concept
- Creating The High Level System Design
- Setting the Schedule
- Understanding what customers will value and conveys this to the engineers making day-to-day tradeoffs
- Arbitrating trade-offs when necessary
- Defending the Vision

Apart from the Chief, the team should also display different roles and responsibilities depending on size, effort, and individual talent, strength, and abilities.

---

[2] **Harlan Mills** was the great Systems Engineer, who over and over showed that even large projects could and should use an iterative and incremental development process.

Copyright © 2012 Leif Trulsson. All rights reserved.

- **Our most valuable and critical resources/assets**

Time and **Talent** are our most valuable and critical resources/assets. They also posses some common features, they can't be reproduced and they can't be recycled (at least not yet). When they're gone, they're gone. Wasted time and talent are gone forever. Their level of uniqueness is extremely high.

Alistair Cockburn[13] says ***"People are a first-order driver of a project's trajectory"***.

I came to the same conclusion several years ago. And during a turbulent merger between two major Scandinavian competitors in the food industry, where I was head of IT for the bigger of the two, I came up with what I used to call **"The first Team Law"**, but today I call it "**The Success Formula**" because it's really a formula about success and how to achieve success.

Figure 8. The Success Formula

Copyright © 2012 Leif Trulsson. All rights reserved.

The Success Formula states that everything starts with **clarity**. A good friend and former colleague of mine use to say, that everything starts and ends with leadership. This also happens to be a vital part of **The Success Framework**[3] and it's also what **Clarity** is all about. True leadership is about formulating and making very clear a number of key components. Because without **Clarity** you and your team/organization are fumbling in the dark.

Clarity is about formulating:

- Clear leadership
- Clear vision
- Clear mission/purpose
- Clear values
- Clear goals
- Clear strategies
- Clear roles and responsibilities
- Clear and transparent information and communication
- Clear rules, policies and principles

Clarity: Is about eliminating any doubts, and makes life so much more easy. And still, this is where so many organizations fail. In fact, I know companies that for many years didn't have any overall goals, or any strategies for that matter. And of course they failed - miserably to say the least. So instead off being on the path from *good to great*, they were on the path *down mediocrity way*.

Some executive managers that I've met through the years, even believes in *Mushroom Management*, in other words keep everybody in the dark and spread the shit on them. Personally, I believe in full transparency. If anyone is old enough, and skilled enough to work for the organization, they are also capable of hearing the truth.

[3] **The Success Framework** is framework developed by Leif Trulsson to analyze a business or an organization and help them to achieve better results. More information about The Success Framework can be found at www.leiftrulsson.com.

Copyright © 2012 Leif Trulsson. All rights reserved.

One of the keywords here is **purpose**. You can have very clear goals, but if the purpose is murky or if there is not defined a clear purpose at all, the goals are really irrelevant. As they say at Toyota – always ask **why** five times. Why are we doing this? Why are we going in the direction that we are going? Why are we having these rules, policies, and principles? Why are we hiring this person? Why, why, why, why, why?

Asking **WHY** helps eliminate any doubt and makes things even more clear. So be like the young child – ask **why**!

If clarity is the first pillar in the team foundation, **respect**, **trust**, **honesty**, and **reward** are the corner stones of the foundation.

Respect: When I came to IBM back in 1977, one of the three *Basic Beliefs* was **Respect for the Individual**. The other *Basic Beliefs* where: The best Customer Service and Pursuit of Excellence, and together they formed the foundation for IBM growing into one of the worlds true and great industrial giants, and by many viewed as a “best managed company”.

The following are three quotes on respect from **Thomas Watson Jr.**, the former chairman and CEO of IBM.

“There are many things I would like IBM to be known for, but no matter how big we become, I want this company to be known as the company which has the greatest respect for the individual.” – Thomas Watson Jr. (1957)

“If IBM is to continue to be strong, to grow, and to bring profit to all of us in the company and to our customers and stockholders, we must be certain — constantly — that we are headed in the right direction, making the right decisions, and treating every employee with respect.” – Thomas Watson Jr. (1961)

“We accept our responsibilities as a corporate citizen in community, national and world affairs; we serve our interests best when we serve the public interest. We believe that the immediate and long-term public

Copyright © 2012 Leif Trulsson. All rights reserved.

interest is best served in a system of competing enterprises. Therefore, we believe we should compete vigorously, but in a spirit of fair play, with respect for our competitors, and with respect for the law. In communities where IBM facilities are located, we do our utmost to help create an environment in which people want to work and live. We acknowledge our obligation as a business institution to help improve the quality of the society we are part of. We want to be in the forefront of those companies which are working to make the world a better place." – Thomas Watson Jr. (1969)

Though many viewed the three *Basic Beliefs* to be part of the reasons why IBM started to fumble back in the late 1980's and early 1990's, I personally don't believe that at all. I firmly believe that it was rather the loss of focus on the three *Basic Beliefs*, together with a change in the core business model that was the root of the problems. The key to IBM's success was never about computers or technology, but about people and processes, and a mindset in the "pursuit of excellence". As **Jim Collins**, the author of *Good To Great* [12], points out: "IBM stumbled badly in the late 1980s because it drifted from its core values (which it should never have abandoned) while remaining too rigid in its strategies and operating practices (which it should have changed far more vigorously)."

Personally, the three *Basic Beliefs* of IBM are so ingrained in me, and I strongly believe that if you adhere to these beliefs you just can't fail. But this is where so many organizations fail. Since leaving IBM in 1997, I have come across a number of executives and companies that haven't got a faintest idea what **respect** stands for or is about. Just as the three *Basic Beliefs* of IBM was part of IBM's DNA and made it a great company, the lack of respect in an organizations DNA is a severe handicap.

So what does **Respect** mean?

"Do unto others as you would have them do unto you." In other words treat others the way you yourself want to be treated. Unfortunately though, many employers regard their employees the same way as they regard any other assets or products that may be used, abused or

Copyright © 2012 Leif Trulsson. All rights reserved.

discarded at will. But on the other hand, some employers like i.e. IBM once did have embraced their employees and made them partners in the growth of the company.

As I said earlier, everything starts and ends with leadership. So also when it concerns respect. It is my duty as leader to instill the values and create the circumstances/environment for the team or organization to succeed. If I have done that, I can expect the team members to function to the best of their ability and fulfill established, well defined, and obtainable goals. Under these circumstances, we may very well expect success from the individual or the team. However, if we for various reasons, *place an individual or team in an untenable or impossible situation in which the individual or team will fail to fulfill expectations* that would be an act of disrespect.

I have met business leaders, that treat their employees with utterly disrespect, and as though they are less significant than themselves. They lie, cheat, deceive, and harass others as though they are supreme and more important than others. They don’t seem to understand, that every time they are behaving deceptively or cruelly they are acting with disrespect and disregard for others.

Trust: All healthy and sound relationships are built on trust. Trust is like a bond or a lifeline, and trust needs to be earned. It takes time to build trust, but it can be ruined and gone in a fraction of second. Trust is also tightly intertwined with both respect and honesty. Take away any of these core values and what ever you are trying to build will come tumbling down.

Trust is such a vital part of our daily life that we don’t even think about it. When we take our car to the road we trust the road signs and that other drivers stick to the traffic rules. When we board an airplane we trust the pilot and crew that they will safely bring us to our destination and we entrust them with our most valuable possession, our lives.

We hopefully also trust our government and other local or federal authorities, but in many countries that’s not the rule of thumb. Even in a “developed” democracy it’s often more a rule than an exception, that

Copyright © 2012 Leif Trulsson. All rights reserved.

we really cannot trust our so-called leaders. Unfortunately powers corrupt, and real leadership is probably the scarcest resource of all.

But to be successful in almost any endeavor we need to build upon trust. In any successful team or organization the level of trust is very high. We trust the leadership and fellow teammates that they will always do their very best for the team/organization. Also on an individual level we need to trust our own abilities, skills, and knowledge. High achievers always have a very high level of trust in their own ability, and success can never be attained if we lack in trust.

When I joined the Customer Engineering department at IBM back in 1978, my first manager was what I call a *rug-sack manager*. He was not very supportive, and when you where on a customer call with a machine failure, he would call you every 15 minutes to check how things where going. This was really annoying, because every time he called you, your thought process was interrupted and you almost had to start the problem determination process all over again. He would also be easily fired up if the customer called him and expressed concerns about the fix progress. The problem he had was that he didn't trust you, at least not well enough to let you do your job in the best possible way.

My second line-manger was the complete opposite. He was always 100% supportive and had complete trust in you. He knew that if you couldn't fix it within the expected time frame, you would call for help. This he also reassured any worrying customer that would call him. The complete trust that my second line-manager showed me and the other Customer Engineers in our group, was to become my own guiding principles years later as I became a manager myself.

Sometimes however, trust can almost become a burden. When I, during a number of years during the 1980s, was Country Specialist for IBM in Saudi Arabia, I was bestowed which such levels of trust, that I more or less received a cult status. Some of my colleagues had such great faith and trust in me, that they where totally convinced that if only I showed up at a site that had a serious system down problem, all problems would be solved. And sure enough, I was lucky, because I managed to live up to these expectations every time I was called in as second line support, even when we faced some very difficult cases.

Copyright © 2012 Leif Trulsson. All rights reserved.

But not everyone is able to live up to the expectations that are put on their shoulders. We have seen this for example in the sports world, where not everyone has the right mental strength to live up to the trust that has been bestowed upon him or her. So we need some moderation. Trust is good, but we also need to offer assistance to those whom we trust and have faith in.

Honesty: Should be everybody's constant core value, but unfortunately it seldom is. Honesty is the bond between Respect and Trust, and when that bond brakes the whole foundation that success rest upon tumbles.

Honesty is the guardrail and an essential quality of clear communication. By being honest both to others and ourselves we show integrity. By being honest we play by the rules and we do not try to taint or mislead in any way. Sometimes honesty hurts, but at the end of the day it's the only way to go.

In his book "*Good To Great*"[12], **Jim Collins** concludes that the Great companies by "confronting the brutal facts" are brutally honest with themselves. They do not let personal investments in ideas or past practice get in the way of reality. They create what Collins calls, "a climate of truth". But at the same time they have an unwavering faith that they can succeed.

In "*Good To Great*"[12], **Jim Collins** also lists four basic practices in creating a climate where truth is heard:

1. "Lead with questions not answers."
2. "Engage in dialogue and debate, not coercion."
3. "Conduct autopsies, without blame."
4. "Build red flag mechanisms that turn information into information that cannot be ignored."

Unfortunately though, the same does not hold true for mediocre companies or organizations.

Copyright © 2012 Leif Trulsson. All rights reserved.

“*Oh, what a tangled web we weave, when first we practice to deceive!*” [Sir Walter Scott]

”*Honesty is the first chapter of the book of wisdom.*” [Thomas Jefferson]

Honesty must be more than a policy; it must really be one of your constant core values.

Reward: When I talk about reward, I’m not referring to the kind of hefty bonuses that we’ve seen in the last decade or so. That kind of reward is definitely not good for either the moral nor what’s best for the business or organization as a whole. That’s pure greed and nothing good has ever come out of greed.

Reward in this context is on the other hand good for the morale. To celebrate success and advancement strengthens team spirit and can act as the glue between project team, users, and sponsors. Many are the projects where a good get-together has worked as a catalyst and platform for further communication and collaboration in the project and thus contributing to the success of the project. But even a tap on the shoulder or a word of appreciation works wonders.

In the “old” IBM, the IBM that the Watsons built, reward and incentives were an important part in motivating and rewarding the employees for their contributions. Also spouses where, at least once a year on a local level, part of this appreciation.

Reward and appreciations don’t have to be on the grand scale like the old IBM HPC (Hundred Per Cent Club; for successful sales personnel), but could be a modest *Dinner for Two* or some other more modest token of appreciation. It was never the financial size of the appreciation that mattered, but the appreciation it self.

Confidence: Is the launch pad that enables us to get off to a good start in our quest for success. Clarity, respect, trust, honesty, and reward create a sphere of comfort and security and are the pillars and

Copyright © 2012 Leif Trulsson. All rights reserved.

foundation for achieving success. Together they make us feel confident and secure in our roles and abilities, and the team's ability as whole.

Confidence means trusting our self and/or our team's ability to succeed. Confidence is a state of mind, and an essential ingredient in achieving success.

This is a good beginning, but it's not all. To complete the formula we also need three more very important ingredients.

Belief in what we are doing: "*You can be anything you want to be, if you only believe with sufficient conviction and act in accordance with your faith; for whatever the mind can conceive and believe, the mind can achieve.*" - Napoleon Hill

No matter how confident we are or how secure we feel, if we don't believe in what we are doing we will not succeed. We have to feel it in our hearts that it's the right thing to do. You might have the skill and talent to do or achieve what it is that you set out to do, but if you don't believe that you will succeed you wont. The lack of belief and trust in your own (or the teams) ability becomes a self-fulfilling prophecy.

Commitment: "*I believe that this nation should commit itself to achieving the goal, before this decade is out, of landing a man on the Moon and returning him safely to the Earth. No single space project in this period will be more impressive to mankind, or more important in the long-range exploration of space; and none will be so difficult or expensive to accomplish.*" — President John F. Kennedy in a special address to a joint session of Congress on May 25, 1961.

As I mentioned before IBM's three *Basic Beliefs* were *Respect for the Individual*, *The best Customer Service*, and *Pursuit of Excellence*. These three beliefs were also a commitment to both its employees and its customers. These were really strong commitments not only on a corporate level, but also very much so on an individual level, and the customer's needs really did come first.

Copyright © 2012 Leif Trulsson. All rights reserved.

The commitment to quality and to excellence gave IBM its unique strength. Even though cost was never an issue, cost issues were never ignored, but the focus was rather on price/performance. This meant that every option was considered in ensuring that the optimal solution was achieved. This was also the essence of the famous THINK campaign created by Thomas J Watson. The result was a unique degree of flexibility, coupled with intellectual application, which could then be overpowering to the outside world.

”*Just because you are a character doesn't mean that you have character.*” *— The Wolf, Pulp Fiction*

The two most important key success factors in running IT projects are user involvement and sponsor support. Both these two factors are also commitments on the part of the users and on the part of the sponsor. Projects that lack these two commitments are more likely to fail than projects that have these commitments.

In January 2009 a new CEO took the helm at Procurator. His experience of successful IT projects was not very good to say the least. What he didn’t understand though, was the critical impair and success factors involved, and the very important role that he himself played in the make it or break it of projects.

Without the right sponsor support, any significant project is almost certainly doomed to fail. It’s the sponsor that sets the agenda and if the CEO is not backing up corporate wide projects in a clear and active way, the organization will take notice of this and will behave accordingly.

The new CEO at Procurator had never, according to himself, experienced any successful IT project. However, before he arrived at Procurator there had been more than 20 successful corporate wide projects during the previous three years. And sure enough, the first major project during his watch failed and he was personally very much responsible for the failure, as he made all the mistakes in the book and then some. In fact, it was totally impossible to succeed under the circumstances that he created. It’s funny how often self-fulfilling

Copyright © 2012 Leif Trulsson. All rights reserved.

prophecies becomes just that – self-fulfilling.

For me personally it was a double defeat. It was the first time in more than thirty years that a project that I was leading failed. The cost to me personally was even higher, as I due to the environmental factors ended up in the ER with a stress related heart arrhythmia, which later forced me to retire from my job due to health reasons.

Commitment is the for-better-or-for-worst part. We need the whole heart to be with us. We need to be committed. We can believe in anything we want, but if we don't put our heart and mind to it, the chance of success is very dim.

Inspiration: Inspiration and motivation is the fuel that makes us tic. Money is a motivator if you don’t pay enough, but you can take the money out of the equation by paying people enough. Researchers at MIT, Chicago School of Economics, and at Carnegie Melon University have found that there are three factors that lead to better performance and personal satisfaction:

- Autonomy
- Mastery
- Purpose

Autonomy, mastery, and purpose are what fuels inspiration.

During my professional career, I have always viewed every position or job I had as my company. I was running the company “Leif Trulsson” (autonomy) and my goal was to excel in everything that I did (mastery) and deliver the best possible service (purpose). In the beginning it was a one man company, but as I later became a CIO I viewed the whole IT department as an autonomous company and my aspirations was for the whole department to excel and under the given circumstances deliver the best possible service to our customers and at an unbeatable price/performance. And we did, and in the process we had a lot of fun.

Copyright © 2012 Leif Trulsson. All rights reserved.

So, feel inspired and have **FUN**! Some might even say if it's not fun it's not worth doing. In fact, inspiration fuels a whole industry, the entertainment industry.

The Success Formula — lays the foundation for success.

We need to increase both the Return on Investment for the business and the Return on Experience for the individual and the team. That's the true reward.

And remember, we need to celebrate!

Copyright © 2012 Leif Trulsson. All rights reserved.

Value Flow Management

Value flow management is about managing the value from idea to realization. The purpose is to shorten time-to-value by minimizing waste, speeding up the flow of value, and delivering tangible value.

Value Flow Management focuses on:
Why?

- Because we want to shorten time-to-value

What?

- Focus on the whole and on throughput, feature size (Minimum Marketable Feature), value generation, satisfaction, cycle-time, process efficiency

Who?

- The developing organization and the end-user/customer

How?

- By eliminating anything that does not add value to the "paying customer" (Waste Management)
- By Value stream mapping
- By managing constraints
- By shorter iterations with shorter release-cycles and shorter feedback loops
- By incremental development, gradually improving the product (growing the product), and involving the customer
- By simple and clear measurements
- By focusing on throughput and decreasing **Time-to-Value**

Many of the ideas behind Value Flow Management originate in LEAN thinking. LEAN thinking traces its roots back to the turn of the 20th

Copyright © 2012 Leif Trulsson. All rights reserved.

century and Sakichi Toyoda. In his textile factory they had problems with looms that stopped themselves when a thread broke. To handle this, Sakichi Toyoda developed the auto-activated loom that automatically and immediately stopped the loom if the vertical or lateral threads broke or ran out.

This became the seed of autonomation and Jidoka at Toyota. The purpose of autonomation is to rapidly or immediately address, identify, and correct mistakes that occur in a process. Autonomation thereby relieves the worker of the need to continuously judge whether the operation of the machine is normal or not and the worker is now only engaged when there is a problem alerted by the machine.

When Toyota in the 1930s moved from textiles to car production, Kiichiro Toyoda, founder of Toyota, discovered many problems in the engine casting. He decided to intensely study each stage of the process in an effort to eliminate the repairing of poor quality. This later evolved into the "Kaizen" improvement teams.

As the levels of demand in the Post War economy of Japan were low and the application of mass production on lowest cost per item via economies of scale therefore had very little or no relevance. After having visited and seen supermarkets in the USA, Taiichi Ohno recognized the scheduling of work should not be driven by sales or production targets but by actual sales. Given the financial situation during this period, over-production had to be avoided and thus the notion of Pull (build to order rather than target driven Push) came to underpin the production scheduling.

It was then in the late 1940s and with Taiichi Ohno at Toyota that these themes came together. Ohno built on the already existing internal processes of thoughts and refined them into what has now become the Toyota Production System (TPS). It is in part from the TPS, but also from other sources, that Lean Thinking has developed.

The scale, rigor and continuous learning aspects of TPS have made it a core concept of Lean. At the core lies the fundamental lean principle: *Eliminate waste.*

Copyright © 2012 Leif Trulsson. All rights reserved.

The elimination of waste is the goal of Lean, and Toyota defined three broad types of waste: *muda*, *muri* and *mura*. In reality and in most Lean implementations only the first waste type is identified. Also, the elimination of waste may seem like a simple and clear task, but waste is often very conservatively identified. This then hugely reduces the potential of waste elimination. But as Shigeo Shingo[4] observed, only the last turn of a bolt tightens it—the rest is just movement. The clarification of waste is the key to establishing distinctions between value-adding activity, waste, and non-value-adding activity. Non-value adding activity is waste that must be done under the present work conditions.

Table 2 is a comparison between the original seven mudas and the seven wastes of software development.

[4] Shigeo Shingo was a Japanese industrial engineer who distinguished himself as one of the world's leading experts on manufacturing practices and the Toyota Production System.

Copyright © 2012 Leif Trulsson. All rights reserved.

The original seven Mudas	The seven Wastes of Software Development
Transport (moving products that are not actually required to perform the processing)	Handoffs
Inventory (all components, work in process and finished product not being processed)	Unfinished work
Motion (people or equipment moving or walking more than is required to perform the processing)	Task switching
Waiting (waiting for the next production step)	Delays
Overproduction (production ahead of demand)	Extra features
Over Processing (resulting from poor tool or product design creating activity)	Extra processing/Paperwork
Defects (the effort involved in inspecting for and fixing defects)	Defects

Table 2. The Seven Wastes

Copyright © 2012 Leif Trulsson. All rights reserved.

Waste Management

Value Flow Management then in part becomes Waste Management and waste can then be identified as:

- ***Anything that does not add customer value based on a deep understanding of what customers value***
- Anything that has been started but is not being used in production
- Any extra features that are not needed now
- Anything that delays development or keeps people waiting
- Creating documents that are not read
- Unnecessary measurements
- Making the wrong thing or making the thing wrong
- Any activity which absorbs resources but does not create value
- Anything that does NOT improve the fit, form, or function of the product or service
- Waiting

To further improve the process, we also constantly need to ask the following three questions:

- Does the activity create *value?*
- Does it *improve* the fit, form, or function of the product or service?
- Is the *customer* willing to pay for it?

We can then use time and waste elimination as competitive leverage. Mapping the value stream also enables us to identify potential waste candidates. Some Value Stream mapping techniques only focuses on work time and wait time, but I like to map three categories:

- **Customer Value-Add (CVA)** – including specifications, working code, and manuals
- **Business Value-Add (BVA)** – including financial reporting, risk management, project management, requirements management, and configuration management

Copyright © 2012 Leif Trulsson. All rights reserved.

- **None-Value-Add (NVA)** – including quality control, quality assurance, metrics gathering, defect repair, waiting, and any other form of waste

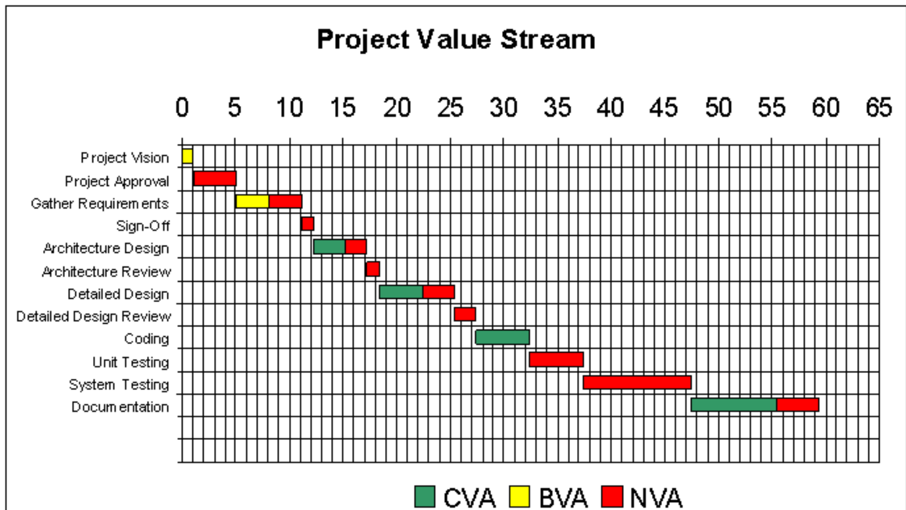


Figure 9. Value Stream Map

Process Efficiency

An expanded Value Stream Map can also measure:

- Tasks, Roles, and Work Products
  - Customer Value-Add (CVA)
  - Business Value-Add (BVA)
  - Non-Value-Add (NVA)
- Process Efficiency

To be able to raise the productivity, we need to increase the throughput. Process efficiency is the value of the output compared to the input. We can use the following formula to calculate the process efficiency.

Process Cycle Efficiency = (Value-added Time / Cycle Time)

Copyright © 2012 Leif Trulsson. All rights reserved.

- **Managing constraints**

To increase the process efficiency we also need to manage constraints. Managing constraints means elevating bottlenecks and exploiting the constraints in an effort to increase the throughput. A system only contains one bottleneck at the time, and this bottleneck defines the pace of the system.

If we are in a built-to-order setup and we can deliver in a faster pace than the demand from the customer, then customer is the bottleneck. If on the other hand the customer demand is higher than our capacity, then it's our bottleneck that will have to define the pace.

Build to order or Pull, which is what we want to achieve, means that nothing is done unless and until a downstream process requires it. The effect of *pull* is that production is not based on forecast and commitments are delayed until demand is present to indicate what the customer really wants.

This also means that decisions about design should be kept open as long as possible (last possible moment) – and when we know we go ahead. This could be compared to the landing of an airplane, where the pilot must be prepared to abort the landing to the last possible moment. Unnecessary commitments should also be avoided, as these will most likely result in unnecessary change requests anyway. In analogy with the landing of an airplane, only prioritized features in the feature buffer (see Figure 11) can and should be committed.

By using the constraint to set the tempo for the system we achieve maximum throughput. We can optimize development cycle time by:

- A steady flow of work based on capacity
- Limiting the number of things in process
- Limiting the size of things in process
- Establishing a regular tempo
- Having the flow of work being demand driven (pull)

Copyright © 2012 Leif Trulsson. All rights reserved.

- **Time-to-Value**

For a given product, the payback time is given. By packaging valuable functionality into ”value packets” or Minimum Marketable Features, and releasing it as early as possible we shorten time-to-value. With staged releases, we get faster time-to-value, faster payback time, and increased ROI (see Figure 10). By releasing functional value earlier we also achieve an increased accumulated profitability.

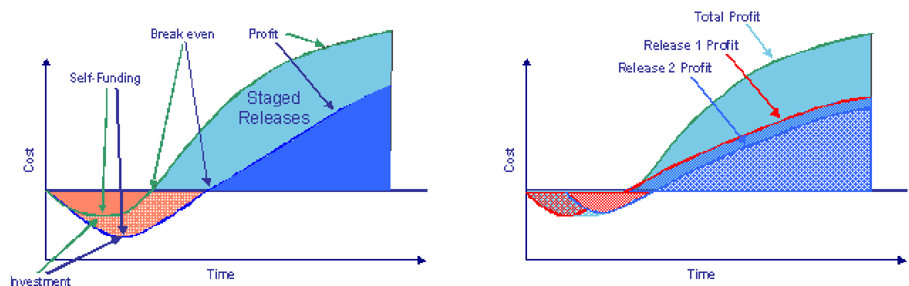


Figure 10. Staged Releases and Total ROI

Figure 11 describes the Value Driven Software Development process.

- Features are prioritized in a feature buffer
- Development is done incrementally in iterations
- The development team picks a number of features from the Feature Buffer and creates Use Cases/Features Stories
- The development progresses incrementally day-by-day, and a stand-up meeting is held daily to report progress and expose problems
- An iteration lasts for about 2-4 weeks and completed features are put into the Release Buffer
- After each iteration a couple of days are set aside for reflection
- Regression and integration testing is performed on the Release Buffer content
- At release date, everything that has been marked for release in the Release Buffer is released

Copyright © 2012 Leif Trulsson. All rights reserved.

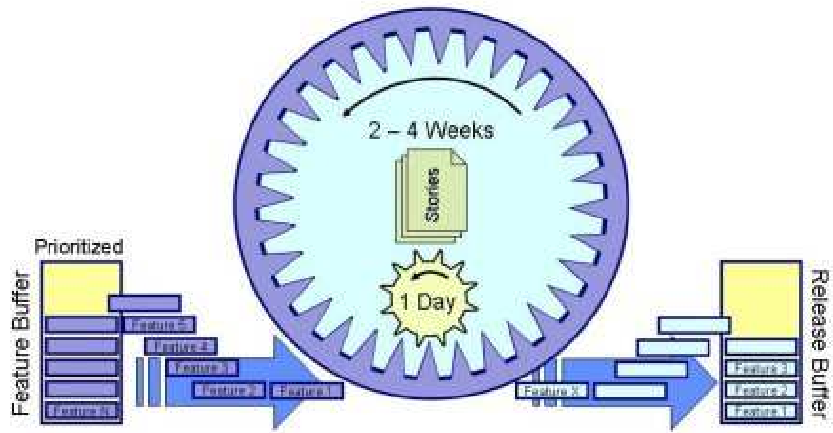


Figure 11. The Value Driven Development process

Copyright © 2012 Leif Trulsson. All rights reserved.

Conclusion

Value Driven Development focuses on three core principles:

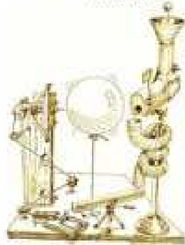
- Value Generation
- Leadership and People
- Value Flow Management

The goal of Value Driven Development is to delivers more value with less effort and increase value, profitability, and the ability to compete in a world of hyper-competition.

So how do we measure the effect and the progress of the Value Driven Development process? I suggest that you focus on three main measures (see Figure 12):

- Satisfaction
- Process efficiency
- Business value

Copyright © 2012 Leif Trulsson. All rights reserved.



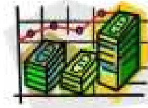


Figure 12. Measurements

"The most efficient route that nature has found from point A to point B is always the path of least resistance."

So remember, you get what you measure so be careful what measures you choose!

Copyright © 2012 Leif Trulsson. All rights reserved.

About the Author

Leif G. Trulsson joined IBM as a programmer in 1977 and has more than as 34 years of professional software development experience, and more than 19 years with IBM.

Leif, who is known as Doctor T. among friends and colleagues, is a software development addict and author, and has held a number of both hardware and software specialist positions within IBM. He is also a former project manager and software configuration management (SCM) product lead at the IBM International Technical Support Organization in San Jose, California. Leif has also published six IBM Redbooks on software development.

After leaving IBM in 1997, Leif became the first head of IT at the Malaco Group in Scandinavia, and in 1999 he became Director IT for MalacoLeaf Scandinavia. Since then Leif has held a number of C-level positions, and today Leif is an independent consultant and Internet entrepreneur.

Find out more about Leif and his work at:
www.leiftrulsson.com

Copyright © 2012 Leif Trulsson. All rights reserved.

References

[1] Fred Brooks; *The Mythical Man-Month*, 1975/1995

[2] Fred Brooks; *No Silver Bullet*, 1986

[3] Craig Larman and Victor R. Basili; *Iterative and Incremental Development: A Brief History*, 2003

[4] Leif Trulsson; *The Art of Project Management — How To Increase Business Values With Efficient Project Management*, 2005

[5] Leif Trulsson; *Increasing business values with efficient Software Configuration Management*, 2005

[6] Leif Trulsson et al; *Software Configuration Management: A Clear Case for IBM Rational ClearCase and ClearQuest UCM*, 2004

[7] Ralph R. Young; *Effective Requirements Practices*

[8] Ivy F. Hooks and Kristin A. Farry; *Customer Centered Products: Creating Successful Products through Smart Requirements Management*

[9] Mills, H.; Dyer, M.; & Linger, R.; *Cleanroom Software Engineering, IEEE Software 4, 5,* 1987

[10] Grady Booch; *Object Oriented Analysis and Design,* 1994

[11] Reichheld, Frederick F.; *The One Number You Need to Grow,* Harvard Business Review, December 2003.

[12] Jim Collins; *Good To Great*

[13] Alistair Cockburn; *People and Methodologies in Software Development,* thesis for Doctor Philosophiae at the Faculty of Mathematics and Natural Sciences University of Oslo Norway February 25, 2003

Copyright © 2012 Leif Trulsson. All rights reserved.