IT Project Management with System Thinking

Mirosław Kordos

still in preparation, last update: November 15, 2010

Abstract

System thinking is an approach that consideres the results of every activity for the entire system, not only on the part the activity refers directly to. The best results to the entire system are obtained when the company vision can be implemented using the personal mastery and intrinsic motivation of employees. The main emphasis in this work is put on the human factor in IT project management, because it is the area when most improvements can be done. Most project management methodologies do not consider the human factor (peopleware) promptly either taking for granted that managers know how to handle their teams or assuming that this is neither a complex nor important issue. The truth is however that managers frequently have no idea about peopleware and this really is a complex and important issue. Nevertheless, still some formal methods to systematize the management of software development projects would be useful. The work proposes a merge of critical chain method with agile approaches based on extreme programming.

Contents

1. Introduction

What is a successful IT project? Is it a project closed on time, or within the initial budget or implemented according to the customer specification? To put it simply a successful project is a project during which and first of all upon completion of which the project manager, the developers and the customer are happy from the project results. That maybe within time, budget and specification, but doesn't have to be. This work discusses some issues helpful in leading successful IT projects.

Strict methodologies were written by few smart people for many less smart people, who don't have enough knowledge and talent to act on their own. Sticking to the methodologies is by no means as effective as being smart, but is for sure more effective than acting without any own thinking and without guidelines. In this work I discuss some of the methodologies, pointing out in which areas they can be used as a backbone of the project (in order not to waste time for re-inventing the wheel) and why system thinking should always take precedende over any methodology.

First of all, project management can't be isolated from the company strategy and from employee management. How the project is managed depends not only on the project itself, but also on the company long term strategy (or what unfortunately sometimes seems to happen on the lack of it). The project is implemented by employees. Factors such if they are happy with the job, how they are motivated to work on the project, what is the exchange of information between them and between the company management and how the responsibilities and decision power are distributed have an overwhelming effect on the project effectiveness. One must remember that software development is quite a complex task. It requires much knowledge and talent and can't be organized in such a simple matter as hamburger production at fast food restaurant. In McDonalds the workers have a set of rules to follow, what enables them to produce hamburgers of desired properties. You can't prepare such a set of rules for programmers, because their job is to think on their own to invent solutions no-one has invented before. It is a creative work, not a reproductive one. Also assessing the quality of programmers work is not so

straightforward. You can assess how many hamburgers employee produced and sold with a very simple measure. There is no such a simple measure for programmers, because each task is different and unrepeatable. That also makes assessing the required time, cost and resources for the whole project more difficult. Another complex problem here is the interaction with customers.

This work is based not only on theory but as well on my own observations and on the solutions I'm introducing currently in my business. After graduating from the university I first worked for polish a company of 40 persons (which I call "Company One"), than for another polish company of 300 persons ("Company Two") and which was declared bankrupt last year. Then for an international corporation with 10,000 employees ("Company Three") and finally since 2007 I've been building my own company. Each of the four companies used a different approach to IT project management. My approach presented in this work is based not only on theory but also on what I learned in the three previous workplaces about what you should do, what you should not do and what will be the most likely short and long term results of all of that. However, I'm sure it is still not the best possible way of IT project management. If a smart person used to work at the three businesses and then at my company, he would finally come out with something better.

As well the activity of a company as of an individual can be decomposed into three pieces: strategy, tactics and operations.

	Company	Individual
Strategy	What is our company mission? (e.g. To deliver the best software for technological process optimization -10 years vision) How do we want to achieve this? (e.g. by System Thinking)	Whom do I want to be next 20 years? (e.g. the head of big IT corporation.) How do I want to achieve this?
Tactics	In which particular IT project should we engage to fulfill our mission?	Which job should I take now that will lead me to my strategic goal?
Operations	Project management (this is what the work is about)	Solving particular problems within the current project.

Strategy

Strategy is the most important factor, because all the lower levels originate from it. Mistakes in strategy are most costly. The second most important piece is tactics. Operations are just the day-to-day activities including project management. But in spite of being at the bottom of the ladder usually most attention is paid to them. That is of course not the right approach. Thus, in some corporations there are two different positions: the CEO of strategy and the CEO of day-to-day operations. It is difficult to embed strategy into a strict methodology; very much depends on the CEO personality, knowledge, prediction ability and innovation. It is possible to embed project management into a strict methodology, but it is highly inefficient and should be discouraged.

Systems Thinking is as an approach to problem solving, by viewing problems as parts of the entire system, rather than reacting to problem itself. (http://en.wikipedia.org/wiki/ Systems_thinking) . The total output of the system should be maximized and local maxima of particular parts of it frequently do not lead to the global system maximum. Peter Senge in his ground-breaking book "The Fifth Discipline" (Senge 1993) introduces the term of system thinking: "Systems thinking needs the disciplines of building shared vision, mental models, team learning, and personal mastery to realize its potential. Building shared vision fosters a commitment to the long term. Mental models focus on the openness needed to unearth shortcomings in our present ways of seeing the world. Team learning develops the skills of groups of people to look for the larger picture beyond individual perspectives. And personal mastery fosters the personal motivation to continually learn how our actions affect our world."

In modern intelligent organizations the market success is not a result of planning and control but of an organizational culture that fosters freedom, creativity, experimentation, convergence of individual's and organization's goals, openness for different views and teamwork (Śliwa 2001).

Tactics

The role of tactics is to answer the question what particular steps should be taken to get the projects we want and then to assess the time, cost and risks of the project and decide if we should take this project. The decision is based not only on the profitability of the single project but also on the whole potential of the interaction with a given customer. In particular even a small, unprofitable project may open the door to big profitable contracts.

There are some measures of the software size and methods of assessing the size and cost of a project. The two most frequently measures of the system size are KLOC - thousands of Lines of Code and FP - Function Points (Flasiński 2006). FP can also be used to assess the system size. FP is a number proportional to the sum of system inputs, outputs, user interactions, files and external interfaces, each of them multiplied by a weight expressing its complexity. The size of the project in persons per month can be assessed with decompositional and empirical methods, e.g. COCOMO II - **Co**nstructive **Co**st **Mo**del (Flasiński 2006). The methods use a set of mathematical equations and discussing them is out of scope of this work.

Operations – Project Management

After the short introduction which places project management in the whole picture, software development project management approach based on system thinking and peopleware management is discussed in this work. The merge of critical chain and agile methods based on extreme programming is proposed as framework for practical implementation of the system thinking idea.

2. Traditional Project Management Methodologies

The Project Management appeared as a new branch of economic science in 1950s. Two project management methodologies were developed at that time. The Critical Path Method (CPM) and the Program Evaluation and Review Technique (PERT), giving an impulse to start intensive project management research.

Many other methods emerged in the following years, including complex project management methodologies such as PMBOK or PRINCE2, which specify the whole organization of the projects (who is responsible for what, what kind of documents, meeting and reports are required, etc.).

There are two main classes of approaches to IT project management: waterfall-based ones, including the Critical Chain Project Management (CCPM), which in my opinion is the best of them and agile-based one, like Extreme Programming, Scrum and others. CCPM proved to be a more effective project management method than CPM and PERT. Agile methodologies in most cases are more efficient that waterfall methods. However, there is a fundamental difference in the objective of CCPM which is to get the project completed and of agile methods which is to make incremental improvements, with each of them making a positive change to the customer.

2.1. Critical Path Method

Critical Path Method (CPM) is one of the most popular methods used in project management. CPM was developed in the late 1950s by Morgan R. Walker and James E. Kelley in order to optimize production processes (en.wikipedia.org/wiki/Critical_path_method). CPM allows for graphical presentation of the consecutive activities with the planned time of each activity and their sequence. CPM can be used when the times of particular activities are a priori known.

Critical path is the longest sequence of the activities required to complete the project. Thus, it also determines the shortest possible time of the project. The critical path (in pink-red in fig. 1) consists of critical activities, this is activities, which cannot be delayed without making the whole project last longer.

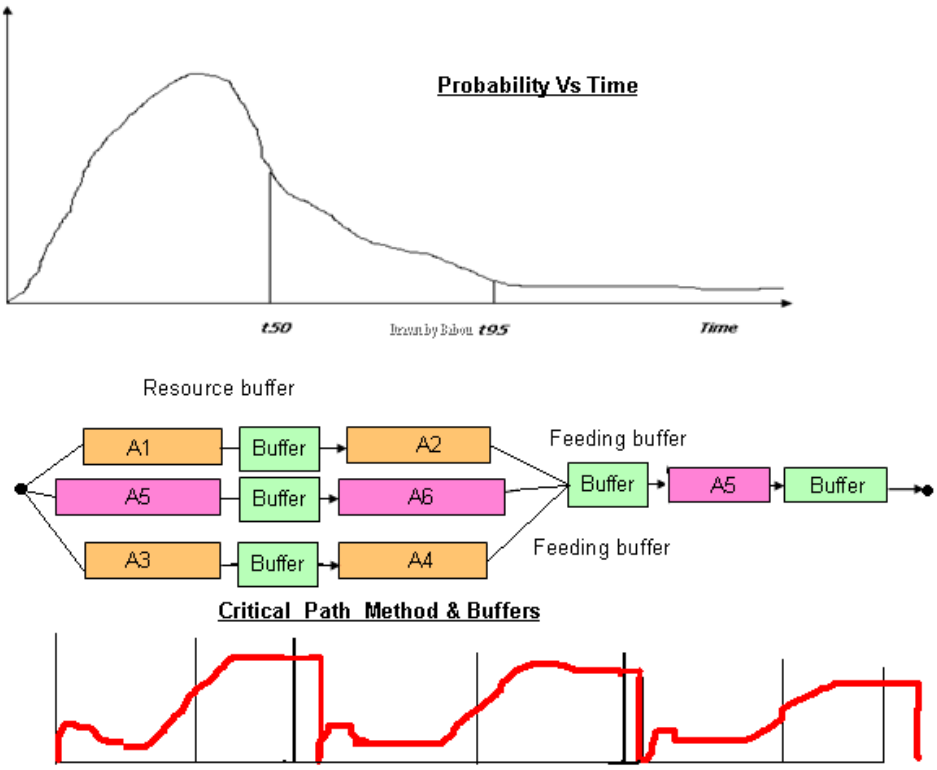


Fig.1. Critical Path Method (based on an unknown source)

A CPM graphs consists of vertices and edges (fig. 1). Each vertex represents the beginning of one activity, while each edge represents the activity duration. There is only one beginning and one ending vertex.

In practice the durations of particular tasks usually cannot be precisely determined, because they depend on many factors that are out of control of the project team. That can be for example the complexity of some algorithms in IT projects, which cannot be a priori estimated before approaching this activity. For that reason product managers usually add some additional time

buffers to each activity. For example: we suppose that we can write some procedures in one month, but we don't know all the details, which we will learn during the detailed analysis of some technological processes and which can influence the complexity of these procedures and thus the time required to create them. For that reason we allow for an additional buffer of two weeks after the activity.

In practice it turns out that assessing proper time buffers is one of the biggest concerns with CPM. The product manager should be a specialist in a given domain (e.g. a developer himself in the case of a software development project), otherwise there is a great risk (and that happens frequently in practice) that his subordinates will not respect him and will try kidding him. In such a case the project manager could not estimate the time of writing the procedure, because he lacks the technical knowledge to do it. So he will ask his team members and in a great many of cases they will overestimate the time in order not to work very hard. That of course depends on many factors, relationships and motivation system within the company. But the rule of thumb is that technical people will see a boss without technical background as someone inferior and it will be very difficult for him to build and effective and loyal team.

But what in the case the team leader is a good developer himself? Still two problems remain. First, he can only roughly assess the resources required for some activities, because he does not know all the details (and he never can know all details of big complex projects), so he must still trust his team. However, if he treats them fairly and has esteem with them, they will usually tell him the truth. Second, and this is a much greater issue, people will tend to overestimate the time buffer just not to get into trouble in the case of exceeding the activity time. The good and knowledgeable project manager will know that they overestimate the time buffer. Moreover, they will tell it him explicitly. But he has no power to force them to reduce their estimation, because if really something goes wrong not only will they get into trouble but he as well, because the entire project will be delayed. So they estimate the duration of the activity with probability of 50% (see picture below) and the time buffer with total probability of 95%. So who is to blame here; the project manger or his team members? Nobody. Just the CPM methodology itself. Is there a way to make the time buffer shorter and still have a high probability of keeping the project deadline? Yes, and it was already solved in a methodology called Critical Chain Method that will be discussed further.

Another problem rising from buffer overestimation is that this tends to give grounds for phenomena called Student Syndrome, Parkinson's Law, formulated as "Work expands so as to fill the time available for its completion." The red lines on the bottom in fig. 1show the work intensity of people if they know they have still a lot of time. Of course the Parkinson's law does not apply to all employees. In particular it does not fully apply to employees who are well managed, so that the work is pleasure for them and they are not interrupted to work according to their intrinsic feeling of quality. That are rare cases will be discussed firther in this work.

PERT (P*rogram Evaluation and Review Technique*) is a stochastic technique of project management (as opposed to CPM which is deterministic). PERT was developed by US Department of Defense in 1950s to be used by US Navy while they were building the new type of rockets (en.wikipedia.org/wiki/Program_Evaluation_and_Review_Technique).

Like in CMP, the core of PERT is the critical path analysis. The difference is that in PERT the duration of particular activities is given by a random variable rather than by a deterministic number. Such an approach allows using statistical methods to assess the probabilities of completing particular activities and the entire project within a given time frame and asses the optimistic, most likely and pessimistic time estimate. In a matter of fact, CPM is a special case of PERT in which all the three estimates are equal.

CPM and PERT are good for "well-known-what-to-do" projects but less useful for conceptual or "fuzzy" projects where agile methods would work better.

2.2. Critical Chain Method

Critical Chain Project Management (CCPM) was developed by Eliyahu M. Goldratt from his Theory of Constraints (Goldratt 1997) and introduced in 1997 in his book "Critical Chain". puts the main emphasis on the resources required to execute project tasks. This is in contrast to the more traditional Critical Path and PERT methods, which emphasize task order and rigid scheduling. Application of CCPM has been credited with achieving projects on average 30% faster and/or cheaper than the CPM and PERT.

With CPM, 30% of time and resources are typically consumed by wasteful techniques such as bad multi-tasking, Student syndrome and lack of prioritization.

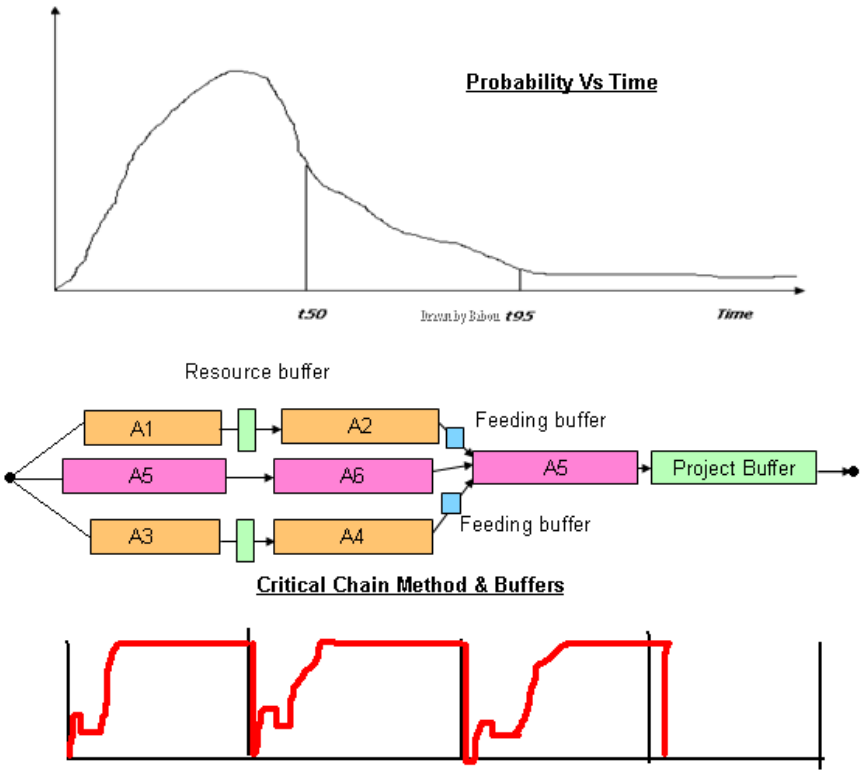


Fig.2. Critical Chain Method (based on an unknown source)

The main differences between the critical chain (CCPM) and critical path methods are:

- The use of resource dependencies in CCPM. In CCPM is that the critical resources should be assigned to the critical path.
- There is no search for an optimum solution in CCPM. A suboptimal solution is acceptable because of the uncertainty in estimates is much greater than the difference between the optimum and sub-optimum solution.
- In CCPM a project buffer is added at the end of the project. Activity buffers are eliminated, leaving only feeding buffers. In this way CCPM aggregates the large amounts

of safety time added to many activities in a single project buffer. That buffer can be shorter than the sum of buffers in the critical path. That also avoids wasting time through bad multitasking and Parkinson's Law (the red line at the bottom in fig. 2 shows the work intensity during the project.).

- In CCPM the project progress is monitoring rather by the consumption rate of the project buffer than individual task performance.

A CCPM project plan is created in a similar way to the critical path project plan. Starting from the completion date, the plan is traversed backward with each activity starting as late as possible. Two times are used for each task: the p=0.5 (50% probability) estimate and p=0.9 (a 90% or perhaps 95% probability) estimate. The last one is the "safe" duration of the task. The probability that a single task will consume its entire buffer is 1 - 0.9=0.1 (10%). As we know from a statistical theory, to find the probability of two independent events that occur in sequence, we must find the probability of each event occurring separately, and then multiply the probabilities. Thus, if the critical chain consists of three such tasks, the probability that all the task buffers will be fully consumed is 0.1*0.1*0.1=0.001 or 0.1%. On the grounds of that the project buffer can be shorter than the sum of task buffers to keep the project on time completion probability at 0.9. (http://en.wikipedia.org/wiki/Critical_Chain_Project_Management)

2.3. PMBOK and PRINCE2

This chapter is contained within the work to give the reader an idea how about how PRINCE2 and other similar project management methodologies are composed in the all project management landscape.

In 1969, the Project Management Institute (PMI) was formed in the USA. Nowadays PMI publishes A Guide to the Project Management Body of Knowledge (PMBOK Guide), which describes project management practices that are common to most projects, most of the time. PMBOK recognizes five basic process groups and nine knowledge areas typical of almost all projects.

As there has always been competition between the USA and Europe ("American Dream vs. "European Dream"), PRINCE2 (**PR**ojects **IN C**ontrolled **E**nvironments) was developed as a European response to the American PMBOK. PRINCE2 was developed from PRINCE in 1996 by Central Computer and Telecommunications Agency (CCTA) in UK (www.prince2.com).

The basic difference between these two methodologies is that "PRINCE2 is a project management method and PMBOK encapsulates general theory on project management." (http://www.pmacademy.co.za/prince2-vs-pmbok) In other words it is easier to implement PRINCE2, than to follow PMBOK theory. "PRINCE2 provides a SINGLE STANDARD APPROACH for the management of a project, whereas PMBOK gives each project manager the freedom to decide on their approach. Many government and global organizations have preference for a single standard approach to be used for all their projects and are adopting PRINCE2 as a result. The other advantage is that people with limited experience can use PRINCE2 whereas PMBOK requires a level of experience to apply its knowledge areas appropriately. (http://www.articledashboard.com/Article/PRINCE2-vs-PMBOK/478463)".

Charles Fox, the author of "Prince2. A No Nonsense Management Guide "puts this in the following words: "People sometimes say that PRINCE2 is bureaucratic and involves too much paper. This is a misunderstanding. The vast majority of the method is highly tailorable and can easily be merged with different working cultures. The 'documents' referred to in the method are just sets of information, they do not need to be on paper at all. Poor implementation can be bureaucratic and excessively paper based, however the method itself is not. The PRINCE2 manual contains extensive description rather than excessive prescription."

The three main sections in the PRINCE2 manual are:

- Eight Components (Business Case, Organization, Plans, Controls, Management of risk, Quality in a project environment, Configuration management, Change control)
- Eight Processes (Starting up a project, Directing the project, Initiating the project, Managing stage boundaries, Controlling a stage, Managing product delivery, Planning, Closing a project)
- Three Techniques (Product based planning, Change control, Quality review)

The detailed description of all the PRINCE2 structure is out of scope of this work and can be found in many textbooks, while a good overview of PRINCE2 can be found at http://corporategeek.info/Prince2-Certification-Nutshell.

The book "Prince2. A No Nonsense Management Guide." by Charles Fox lists among others the following benefits of PRINCE2:

- Greater predictability in management of change
- Direct link between strategic objectives and operational change
- Clarity of what will be achieved
- Immediate escalation if business benefits are threatened
- Establishes best practices
- One set of principles that can be applied to any size of project in any environment
- Clear expectations for what to deliver, when and why
- Deliverables details can be agreed before commencement of activity planning
- Team Involvement in decisions
- Clear decision making authority and escalation procedures

There are also many drawbacks of PRINCE2:

- Many organizations seem to suffer from what the PRINCE2 experts call "PINO Syndrome" (Prince Only In Name). Those organizations chose only some points of the PRINCE2 methodology without paying attention to the core of PRINCE2.
- PRINCE2 makes an important point of documentation as a way of controlling the project progress. However, in come organizations the documents tend to be a goal in itself. Everything looks fine in the documentation while the real project becomes a total failure
- PRINCE2 recommends regular information exchange, what in practice frequently means long, unproductive meetings.
- PRINCE2 doesn't define the requirement analysis, thus a wrong assumptions may be made leading to the project failure.
- Following PRINCE2 too strictly without the proper adaptation to the local conditions may cause too high work overhead, especially in the case of small products.

- PRINCE2 is not an agile methodology (However there are known merges of PRINCE2 with agile software development methods, where PRINCE2 was used for high-level management and agile methods for day-to-day software development (www.best-management-practice.com))

Special certifications exist in PRINCE2 as well as in PMBOK, which maybe helpful with career at big corporations. There are two certification levels in PRINCE2: Foundation and Practitioner. The PRINCE2 Foundation test for example is a multi-choice test. The passing mark is 38 out of 75 questions. It's a closed book exam, but all you need to pass it is to learn the official PRINCE2 manual. Below is one sample question from the test:

> In PRINCE 2 what is the name of the product which is used to define the information which justifies the setting up, continuation or termination of the Project?
>
> a) Project Initiation Document
> b) Business Case
> c) End Stage Approval
> d) Project Brief
> e) Project Mandate

You can find more sample PRINCE2 exam questions for example at http://prince2.technorealism.org/index.php?page=Prince2_Exam_Questions. The sample test question and the listed components, processes and techniques are quite a useful overview of what is PRINCE2 about.

Susan de Sousa, a practicing project manager with PRINCE2 certifications writes that: "Lastly, in my experience learning PM theory is all well and good, but you will find it doesn't help in real life. After all most PM's in the public sector have at least one if not several PM qualifications, and the track record of IT projects in this sector is appalling. In fact in the UK we have wasted conservatively £18 Billion on failed NHS IT projects all of which were managed by PM Qualified personnel. The most important traits in my book are common sense, good people skills, lateral thinking and experience. With these tools you can learn to deliver just about any project successfully to time and budget."(http://www.linkedin.com/answers/business-operations/project-management/OPS_PRJ/440462-863114?browseIdx=0&sik=1243922341320&goback=.amq)

That is not to say that using PRINCE2 is useless. Using PRINCE2 or other project management methodology maybe the only reasonable way of implementing big projects in big organizations at the absence of smart and talented people and at the presence of the assessing system that rewards employees based on how well they stick to procedures and how many somehow measured "points" they achieve. Such methodologies can be also a source of reference for smart people that are given freedom at their work. They, however, are not obliged to implement the methods and usually they don't.

There are of course several more methodologies, e.g. ITIL (Information Technology Infrastructure Library), which is a set of concepts and practices for IT companies. However, discussing them would not add any value to this work.

2.4. Software Engineering: Waterfall Method

The waterfall software development methodology works in a sequential manner. Each stage is started after the previous step is completed, as shown in the figure below.

Though this can seem quite logical, there are several limits to the methodology. First of all it assumes that the project objectives remain unchanged during the project realization. That may be true only in two circumstances: when the project is small and simple and when really the customer knows from the very beginning what exactly he wants. Then the system requirements are correctly specified and finally the system design is done correctly at the first approach. Many companies try to stick to the methodology believing that all this can be done. However, in practice it can be achieved very seldom.

Even, if one of those two cases occurs, we have still one serious problem: a mistake made at an earlier stage of the project automatically propagates to all later stages. That of course requires repeating all the stages starting from the detected mistake to the very end. Thus, especially the mistakes made at the early project stages may be very costly. It is of course impossible to make a project without mistakes. So it seems there can't be any reason to justify the use of waterfall

methods for big IT projects. Yes and no. Yes in the basic form of the waterfall scheme and no after some modifications are applied, about which I will write in the following chapters.

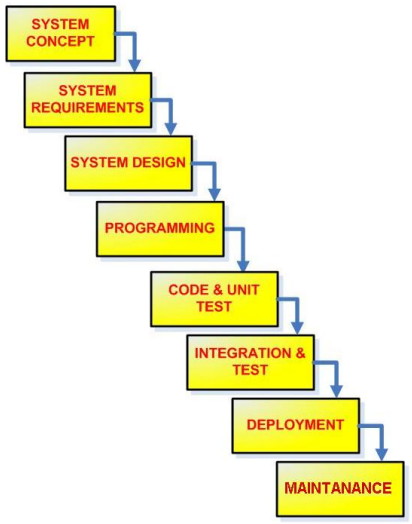


Fig.3. Waterfall Software Development Project Approach.

The frequent problems in my practice were customers who did not know what they wanted. After we agreed with the customer the program functionality and well advanced our work, he used to come to us and say that he had changed his mind and some things must be done differently. So what we did? Of course we applied exactly the same procedure we would have applied after detecting a mistake: repeating all the stages from system requirements downwards. That was of course faster than doing it from scratch, because only the modifications had to be applied. Nevertheless, the waste of time was quite significant.

Still another kind of problems, which will happen almost for sure, is the difference of understanding of some concepts among the customer, the system analyst, system designer and developers. That is well depicted by the picture below.

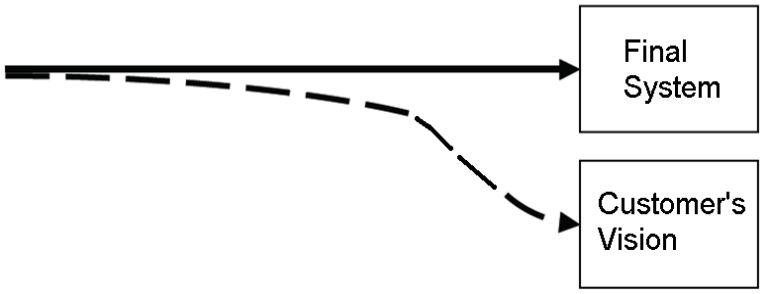


Fig. 4. What the customer wanted and what he get.

Let's have a look at the next picture. Maybe it does not describe precisely that phenomenon, because what really happens, is that the changes in concept understanding gradually grow from one phase to the other, but I like it, because it shows well something else. It shows that the first picture and the last one are not the same. What does it mean? That means exactly that the customers don't know what they want. Quite a common problem. It is the role of the business analyst to get what the customer really wants and not what he thinks he wants. Sounds fine, but is it possible to do it? Yes, and moreover it is quite simple. Yet, not many business analysts ever try it.

Let's assume the customer produces some chemical substance. He knows that the ingredient X is extremely expensive and he wants us to write software that will assists in dosing X in the process so as little X as possible is used. Sounds reasonable, yes? No! Does he really want to reduce X usage or rather decrease the total cost of production? Is he sure that decreasing X usage will also decrease the total costs? It is what he thinks. So what we can do is to help him formulate that primary goal by asking him why he wants to reduce X usage. Then we will probably find out that what he really wants is to reduce total production cost. Now we can ask him if he is for 100% sure that X reduction is the best way to do it. That may lead him to consider alternative ways of cost reduction which may turn out to be even more effective and/or much easier to implement or to achieve. That also totally changes the purpose of our software: to optimize total production cost, not to minimize X usage.

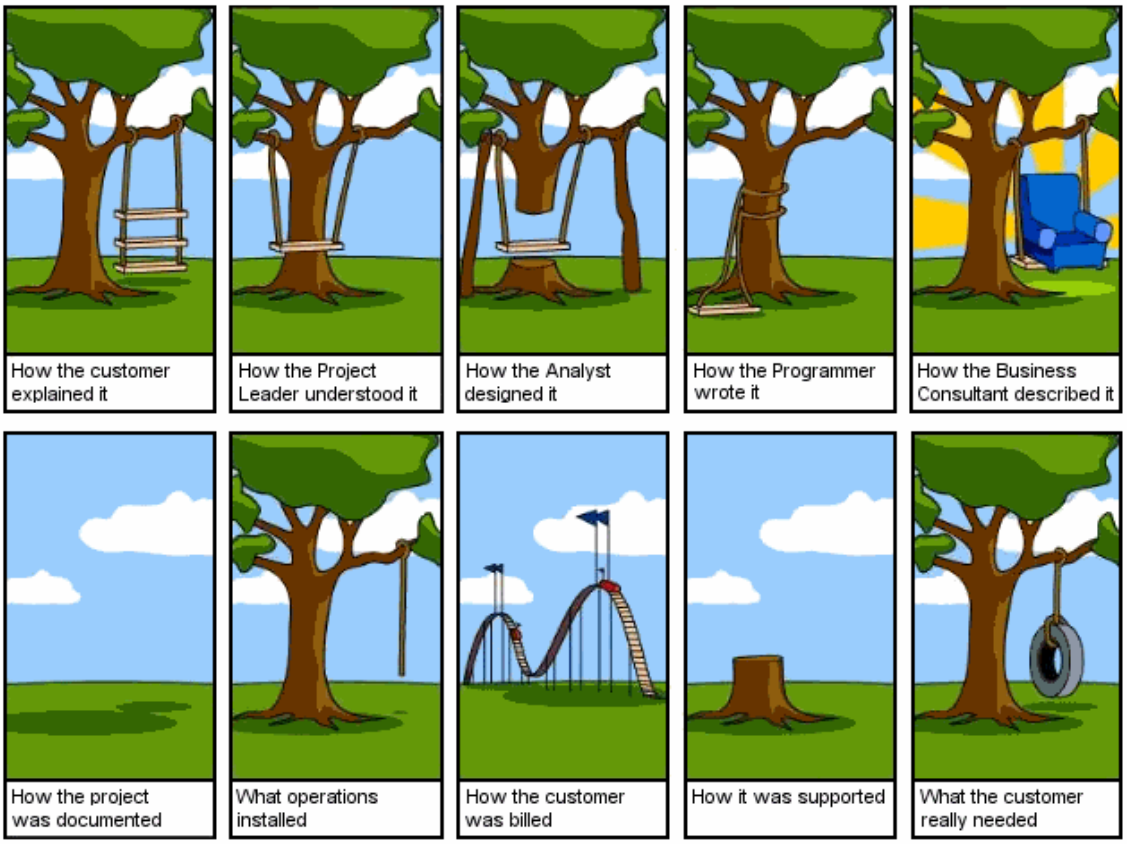


Fig. 5. A not quite true story about the waterfall project management methodology. This picture can be found on many internet sites. I was unable to find out who is the original author of it.

This happens almost always when the customer proposes particular software solutions. E.g. when he says: "I need three databases: one for that, one for that and one for that". In this case it is almost sure that he really needs something different, but because his old application once had three databases, he is now convinced that the only way to solve the problem is to have three databases. Ho John Lee (http://www.hojohnlee.com/weblog/archives/2005/10/02/ten-views-of-project-management) writes that "Various versions of these observations (Shown in fig.5) on project management have been kicking around for a long time, probably for as long as there have been large scale projects. I wouldn't be surprised if they eventually found something like this painted on a cave wall or inside one of the Pyramids at some point."

3. Peopleware

3.1. Managing People as the Most Important Area.

Each IT project consists of software, hardware and peopleware. The peopleware is the most important part of that. However, this is also the most difficult part to deal with. Since managers usually prefer easier tasks, more attention is paid to technology than to people.

Tom DeMarco and Timothy Lister in their book "Peopleware: Productive Projects and Teams" put this in the following words: "If you [as a project manager] find yourself concentrating on the technology rather than the sociology, you're like the vaudeville character who loses his keys on a dark street and looks for them on the adjacent street because, as he explains, the light is better there."

Many project managers admit that they have more troubles with people than with technology, but at the same time they put much more efforts to dealing with technology than with people. Another tendency of many managers is imposing given technical solutions on their teams, instead of leaving that for the most knowledgeable engineers and managing the team and not the technology. That is devastating for two reasons: the technological aspects are not decided by the most competent persons and the teams are not managed smoothly.

The typical problems associated with IT projects conducted with the waterfall method are:

- budget exceeded
- time exceeded
- buggy software
- software created not as customer wanted

What is the reason of the problems? Usually pure technological approach to the projects, without considering the human factor. In other words, when the project is planned, the managers think about hardware and software without paying the due attention to peopleware.

But what happens then, when they see that the project went into troubles? At this moment they usually realize that the human factor was not considered and they try to fix this. There are in general two ways of fixing the problem: the stupid one and the wise one.

The stupid solution seems usually more intuitive, so it is used more frequently. The wise solution requires some knowledge and wisdom from the project manager. Had he the knowledge and wisdom he would consider peopleware from the very beginning and never allow such problems to occur. That is an example that intuition without proper research and knowledge is worth nothing. The stupid solutions simply don't work or even worse; they work in the opposite direction as it was intended.

Let's start from the stupid ones. They are: higher work discipline (in the wrong understanding of this term), more strict sticking to procedures and local indicators.

So, the budget is exceeded, the time also and the software quality is poor. Who is to blame? The managers claim that the developers of course. It is obvious for them that the project was designed well and could be implemented successfully if only the developers worked well. But for the developers it is clear that they couldn't have worked better, because the managers disintegrated all their efforts.

The managers seem to behave as the Polish government in 1980s, when one of the prime ministers told that the reforms failed not because the government organized them in a wrong way, but because the society was not intelligent enough to accept them.

Of course it is the government in the country and managers in the company, who are responsible for the success of reforms and projects. Now, let's have a closer look at this.

The managers say that the work discipline must be improved and they begin to require the developers to come to office at 8 a.m. and work to 4 p.m. But they forget about one fundamental

issue: programming is a creative job; it is science and art, the same way as composing concertos. It cannot be measured in hours, lines of code, number of bugs and so on.

Below three methods of management are discussed: The Command and Control Management Method, The Measurement Based Management Method and The Identity Management Method.

3.2. Command and Control Management

The Command and Control Management Method is based on military management. But rather on the old-style military management in a poorly performing army. The American Army managers for example are encouraged to consult The U.S. Army Leadership Field Manual (The U.S. Army Leadership Field Manual 2004), which is constantly updated and which contains much more intelligent approaches than the old-fashioned Command and Control Management Method.

In the Command and Control Management Method as Joel Spolsky puts it, "the idea is that people do what you tell them to do, and if they don't, you yell at them until they do, and if they still don't, you throw them in the brig for a while, and if that doesn't teach them, you put them in charge of peeling onions on a submarine"

There are, it turns out, three drawbacks with this method in an IT team.

First, people don't really like this kind of management. People, especially software developers and other highly qualified staff don't like to be treaded as if they had nothing to say and had to follow commands, which may seem to them unreasonable.

According to the research that I was presented with at one seminar lead by an American Career Advisor, the most frequent reason of changing the job in the USA is that people are at odds with their bosses. And the situation gets so stressful with time that they finally decide to look for another job. That is the cause number one of changing jobs and not "pursuing new career

opportunities". Who decides to leave first if people feel uncomfortable with their bosses? The best qualified employees of course, because for them it is the easiest to find a new job.

It also happened once in my life. At that time I'd been working for over three years at Company One. I was very happy with that job. First of all, I had a very good boss. He had a high level of technical skills himself, but every time when he wanted me to do something, we together decided what should be done or perhaps that we would better skip this opportunity. It also happened from time to time that I refused my boss to do something, explaining him that either I don't have enough technical knowledge in this field or that I think I shouldn't be done this way and he always agreed. When I was at a customer and the situation developed in an unexpected way I had always freedom to decide what to do. In general the customers were very happy with my services, so was my boss. Very frequently he didn't know where I was at a given time of a day and what I was doing. But he saw that the results of my work were excellent so he never bothered about finding out what I currently did and where I was, unless he needed me at that time. I was never controlled how many hours I work, what for I use the company car, company notebook, etc. Only the results measured by customer satisfaction and how much they were willing to pay in a long run counted. I must say he was an excellent boss. He used the same approach to customers as to employees and the business was thriving.

All of the sudden the situation changed in summer 2000. After my summer vacations I came to the office and instead of my old boss I met a new one there. He began to implement to Command and Control Management Method. He was controlling tightly me and all my colleagues. When I was not in the office, but at a customer's site, he called me several times a day to check what I do. Once he called me at 3:40 p.m. and I told him that I'd just finished and I'd go home. He replied that I had to back to the office, because the working hours are to 4:00 p.m. He didn't trust me and any other person. He didn't allow us to pick up the phone, when customers called us. He wanted to pick up all phones himself, because he was afraid we would cheat him. It took me two weeks to find a new job. Then I went directly to the company CEO and told him that I'm leaving because I'm fed up with the new manager of our department. There were 13 employees in our department. In the first two months with the new boss 12 of us found new jobs and left. We not only left the company, but many of the customers were so satisfied with our previous service that they wanted the same persons to provide the service. As a result also about 80% of

the customers left the company and became customers of our new employers. So that are the results of Command and Control method.

The second practical drawback with Command and Control is that “management simply does not have enough time to micromanage at this level, because there simply aren’t enough managers.”

The third one is that people at the down of the corporate ladder frequently know better how to solve particular problems than the higher managers, who just don’t know enough details of the situation.

3.3. Measurement Based Management

3.3.1. Principles and Outcomes

The measurement based management is tightly connected with Taylors's economic theory of motivation that states in short that everyone is motivated by money, and that the best way to get people to do what you want them to do is to give them financial rewards and punishments to create incentives.

There are a number of beliefs that F. W. Taylor expressed:

- Fundamentally people dislike work and have to be pressured into doing it.
- Employees are untrustworthy and unreliable, and hence have to be constantly supervised and directed.
- For maximum productivity, it is necessary to standardize jobs by dividing them into tasks and sub-tasks. Each of these must be allocated to a different person.
- Discipline is necessary to ensure that work is done, a system of hierarchical authority wisrequired to implement management’s policy.
- The task to be carried out have to be carefully studied, and the ‘one-best-way’ discovered and taught to employees. Each task have to be carefully selected.
- The tools with which to do the tasks had to be carefully chosen.

- Ensure that employees use the ‘one-best-way’ by using a payment by results system – the more you produce, the more you earn.

But what one should remember is that though this theory might apply to unskilled workers who were coming to the USA from different countries one hundred years ago, it doesn’t mean it should be applied to highly skilled professionals nowadays.

For example, software companies tend to give bonuses to programmers who create the fewest bugs or write the most pieces of code. Unfortunately this replaces intrinsic motivation with extrinsic motivation.

Intrinsic motivation is a human own, natural desire to do things well. People usually start out with a high level of intrinsic motivation and they really want to do a good job, to write less-buggy code. Extrinsic motivation is a motivation that comes from outside, like when you are paid to achieve something specific. Intrinsic motivation is much stronger than extrinsic motivation. People work much harder at things that they actually want to do (Spolsky 2004).

According to my own experience, because of intrinsic motivations and no need to comply with company standards and procedures, programmers who work on their own are on average five times more productive than programmers at big corporations. That means if both have the same level of technical skills, the software that the independent programmer creates in 100 hours would require 500 hours of work from the corporate programmer.

Joel Spolsky (Spolsky 2004) writes that when people are offered money to do things that they wanted to do, anyway, they suffer from something called the “over justification effect”: “I must be writing bug-free code because I like the money I get for it,” they think, and the extrinsic motivation displaces the intrinsic motivation. Since extrinsic motivation is much weaker, that results in a decreased desire to do a good job. And there’s no way back. When they are no longer paid the bonus, they will not regain the intrinsic motivation, they will just lose the eternal motivation either.

Another big problem with Measurement-based Management is that this leads to the optimization of local maxima. When the manager uses Measurement-based Management, he encourages developers to focus on what is being measured and not on doing the job well.

Joel Spolsky (Spolsky 2004) writes that "Software organizations tend to reward programmers who write lots of code and fix lots of bugs. The best way to get ahead in an organization like this is to check in lots of buggy code and fix it all, rather than taking the extra time to get it right in the first place. When you try to fix this problem by penalizing programmers for creating bugs, you create a perverse incentive for them to hide their bugs or not tell the testers about new code they wrote in hopes that fewer bugs will be found. You can't win. It seems like any time you try to measure the performance of knowledge workers, things rapidly disintegrate."

Robert Austin (Austin 1996) calls this "measurement dysfunction". When managers like to implement measurements, employees have an incentive to "work to the measurement" making all their efforts solely to increase the measured values and not to increase the actual value or quality of their work.

He says that there are two phases when introducing performance metrics. At first, the manager can get what he wanted, because nobody has find out how to improve the measured results. That phase however last very shortly. Several days on the average, sometimes one day only, if this is not the first measurement introduced. In the second phase, the situation get worse than it was before the introduction of the metrics, because everyone has already figured out how to maximize what is measured, even at the cost of ruining the company. Measurement-based managers think usually that they can avoid this problem by tweaking the metrics. But they can't. It never works, but always backfires. People are not so stupid and will always find a way.

In a matter of fact, measurement-based management is a refusal to find out how to make things work. Even if managers see that what they do doesn't work, they try the same kind of tricks again and again. Does an intelligent person do anything again and again when he notices it doesn't work?

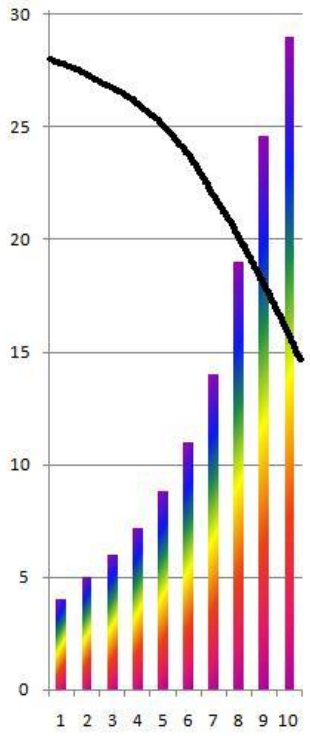


Fig. 6. Growing local indicators and falling company results (black line).

Mike Daisey (Daisey 2001) writes that customers calling Amazon.com were frequently cut off right after "How, may I help you?". The reason was that Amazon rated their call center workers based on the number of calls taken per hour. The best solution in this situation was to disconnect right after the talk started, thus increasing the number of calls taken per hour. I'm not sure if this is true, because he had some personal argument with Amazon.com, but nevertheless, the "measurement dysfunction" is very likely to work this way in this situation.

The problem with metrics management is that it subverts intrinsic motivation. And "there is nothing more discouraging to any worker than the sense that his own motivation is inadequate and has to be supplemented by that of the boss". The Identity Method is a way to nurture the intrinsic motivation.

Alfie Kohn, in a Harvard Business Review article, wrote: “At least two dozen studies over the last three decades have conclusively shown that people who expect to receive a reward for completing a task or for doing that task successfully simply do not perform as well as those who expect no reward at all.”

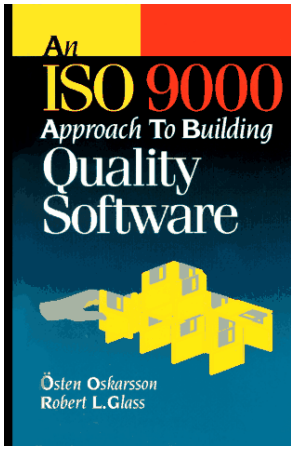
3.3.2. Case Study: Company Two.

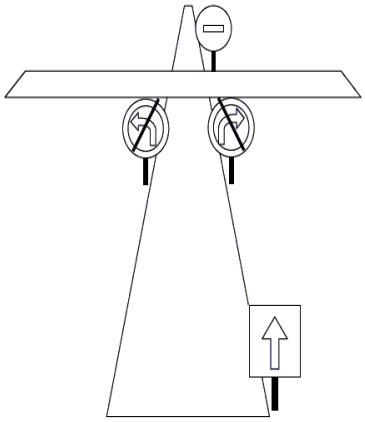
In years 2000-2005 I was working at Company Two. People employed at the company originally displayed some initiative, however the supervisors rather than appreciating us for that tended to criticize us for the initiative not being perfect. As a result we learned that the best course of action is not to do anything unless you are explicitly told so. Because the company was awarded ISO 9000 certificate, lot of time was wasted on paperwork. Much of the paperwork could be easily automated, what was obvious to all employees, but after some initial initiative to do so was ignored by the supervisors and the employees were told not to exceed their range of responsibilities, no-one more bather with that.

At Company Two we were assessed by the local indicators, as the number of hardware repairs people performed in the computer service department or number of procedures the software developers wrote. So we began to work solely to boost the statistics and not to have the work well done. For example, we learned that it was better for us to persuade a potential customer with a complex problem to go to another service provider or not to do the intended service at all. In this way we were trouble-free and instead of one complex computer repair we could do several simple repairs in the same time, thus increasing the statistics (although generating less income). Also whenever possible we were splitting one service action into several smaller actions (the same way one programming task into several smaller tasks), resulting in several invoices, which had o lot of time overhead due to the paperwork, but also increased our statistics. The carrot was not used in the company at all, only the stick, when the number of repairs fell below a certain threshold. Some workers with the lower scores were threatened to be fired and five finally were asked to leave the company. The reason of the leave was however not their poor performance but

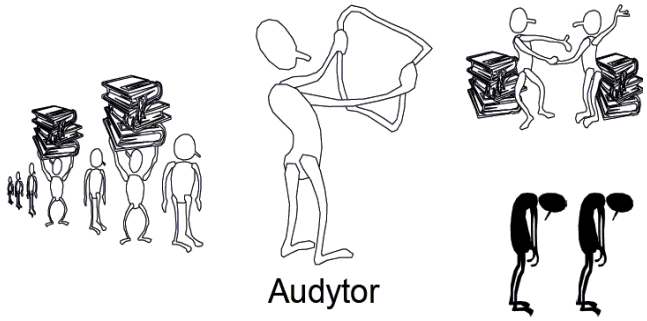
the poor financial condition of the company. The number of repairs was only the key to select the employees that will have to leave. Customer satisfaction began to fall due to these practices and several customers left the company. Gradually also the best professionals began leaving. Some of them went to another IT companies taking the best customers with them (exactly as it took place when I was leaving Company One).

The management at Company Two was likely to believe in Taylor's economic theory of motivation (ETM). However, the theory was not fully applied, since the increase of the employees' productivity was not reflected in their earnings. The ETM stated that people dislike their work and have to be pressured into doing it. It was exactly the case in the company. However, our dislike of the work rather than being inborn grew in response to our supervisors' policy.





ISO 9000



Audytor

Fig.7. ISO 9000 Approach to Building Software

The management standardized jobs and divided them into tasks, but rather than for maximum productivity they did so for the purpose of statistical assessments and of complying with the ISO 9000 procedures. Each of these tasks was really allocated to a different person. But that resulted in increasing the time in which the job was done, because each person in the chain had to wait for the preceding person. For instance, to exchange a processor in a computer: the first person (salesperson) initiated the action and filled one document in a computer system and printed two copies, then the storekeeper received one of the printed copies and signed the other one and returned it to the salesperson, logged into the system, ordered that the computer and processor be sent to the service department and printed two copies of another document. The third person was the man who transported the equipment from the store to the service. In the service department the fourth person logged into the system and printed two copies of the next document (as always one for him and one for the next person in the chain) and brought the equipment to the service technician. The technician did the work and logged into the system to generate the next document and brought the equipment back to the previous person. He again generated the next document and printed two copies of that. Then the previous person took the old processor back to the store. The storekeeper generated another document and printed two copies. Finally the salesperson generated his document, printed two copies and finally the modified computer was ready to be sold. The entire operation took two hours, of which 10 minutes was the physical exchange of the processor. If one of the links in the chain was broken, what happened sometimes, than the operation was ready on the next day.

Companies that are proud to have ISO 9000 certificates should be rather ashamed of that, since ISO 9000 proves nothing else but that the company has mastered wasting time and money on paperwork.

If the economic theory of motivation would be strictly followed, several aspects could be improved if:

- The measurement indices were the income the company can make on repairs performed by an employee and not the number of repairs

- Employee's salary would be proportional to that index
- The salary of non-technical employees for which the index could not be directly applied would be proportional to the average index achieved by the technical workers in a given month
- That could also encourage the employees to optimize and automate the paperwork as far as the management allows and thus allocating more time to income generating activities.

I left Company Two in 2005, where the decline was already clearly visible. Company Two was finally declared bankrupt in 2009.

The common practice in Company Two was that if something was done well, there was no feedback at all. ("No news means good news"). If something was done the wrong way, the employees rather tended to hide their fault from the supervisor or to make it someone else's fault. Sometimes they succeeded at this. That however frequently did not allow fixing this mistake before it was too late. Most of the customers would probably be happy if the engineer notices his mistakes before the customer does it. That is nevertheless frequently impossible without informing the supervisor about that (e.g. the engineer has to go once again to the customer's site to fix it.). The employees didn't have a sense of common company good - they didn't identify company goals with their goals at all. Even worse what was sometimes good for the company frequently turned out bad for them, e.g. admitting to the mistake or spending more time on some tasks to do it perfectly. That was solely the result of the management's policy.

The management of Company Two wanted to shorten the time of the project, reduce costs, reduce bugs and increase functionality using only one method for that: a set of stupid metrics to measure employees' performance. The results were exactly as shown in fig. 6 – growing metrics and falling quality and productivity.

Projects work as a hydrodynamic model (fig. 8). You can't shorten the delivery time, reduce cost, improve quality and enhance functionality at the same time, at least with the measurement based management method. When you press on one piston, the others go up. So is there no way

do have everything better? The good news is: there is a way - the identity management method, as I call it here and as Joel Spolsky called this.

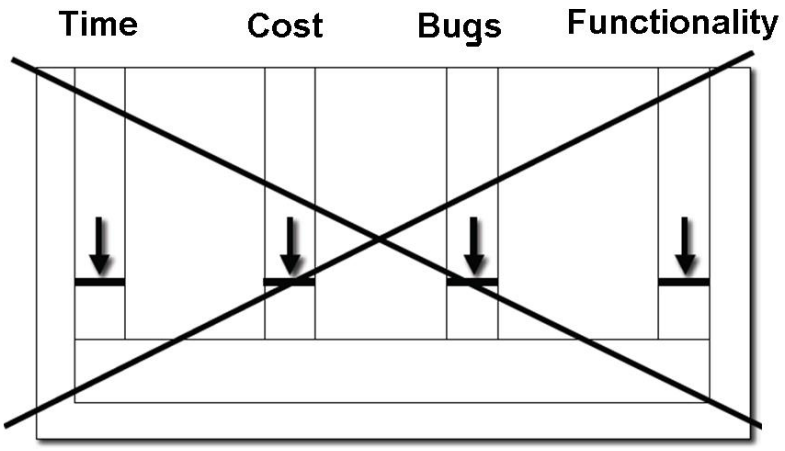


Fig. 8. Hydrodynamic Project Model.

3.4. The Identity Management Method

The developers who work on their own are, according to my own research conducted among my friends and according to the opinions that can be found on the internet, on average five times more productive than developers who work at a big corporation. This chapter and the following ones will try to answer why this is so and will try to give some recommendations to project managers how their teams can get closer to the productivity level of independent developers.

Since the command and control management and metrics based management both fail, another method must be implemented. Herzberg in his theory of motivations (Herzberg 1966) distinguished motivators and hygiene factors. As opposite to the Taylor's theory discussed in the previous chapter, it seems quite well matched to software developer teams.

Herzberg writes that there are at least five factors (motivators) that determine job satisfaction:

- achievement
- recognition
- work itself
- responsibility
- advancement

Where he asked his responders if they were satisfied with their jobs they mentioned these factors as a reason of the satisfaction. But when they were dissatisfied they mentioned quite a different set of reasons, which Herzberg called demotivators or hygiene factors:

- company policy and administration
- supervisors
- salary
- interpersonal relations
- working conditions

I would add to the motivators list (in front of all the other factors) the E-factors: enthusiasm, energy, emotion, excitement.

Even if the list is not complete, it is a very good clue for the managers. At this point it is already known what to do. Now we must consider how to do it.

Steve McConell (McConell 1997) presents an interested modification of the Maslow's hierarchy of needs. Many managers in "good organizations" know about the hierarchy and try to bring their team members to the fifth level, because the think it is the very top. And that works. But still the highest level is missing in the picture (fig. 9): identity with E-factors. Unfortunately not many managers know that the sixth level exist and that bringing developers there works even much better.

I was wondering whether to use the term "identity management method" as Joel Spolsky calls it or the term "system thinking management method" – the term used by Peter Senge (Senge 1993). System thinking is close to the identity management, however that is an approach that wider includes the company, as opposed to identity management that focuses mostly on the employee.

From the manager's viewpoint system thinking should be used, but the employees can see it as identity management.

In The Identity Method, the goal is to manage by making people identify with the goals your company is trying to achieve. So that while their work is powered by the E-factors and motivators, the results of the work are the goals of the company. The personal goals and the company goals maybe different, but the crucial point is that by pursuing the personal goals the employees also pursue the company goals.

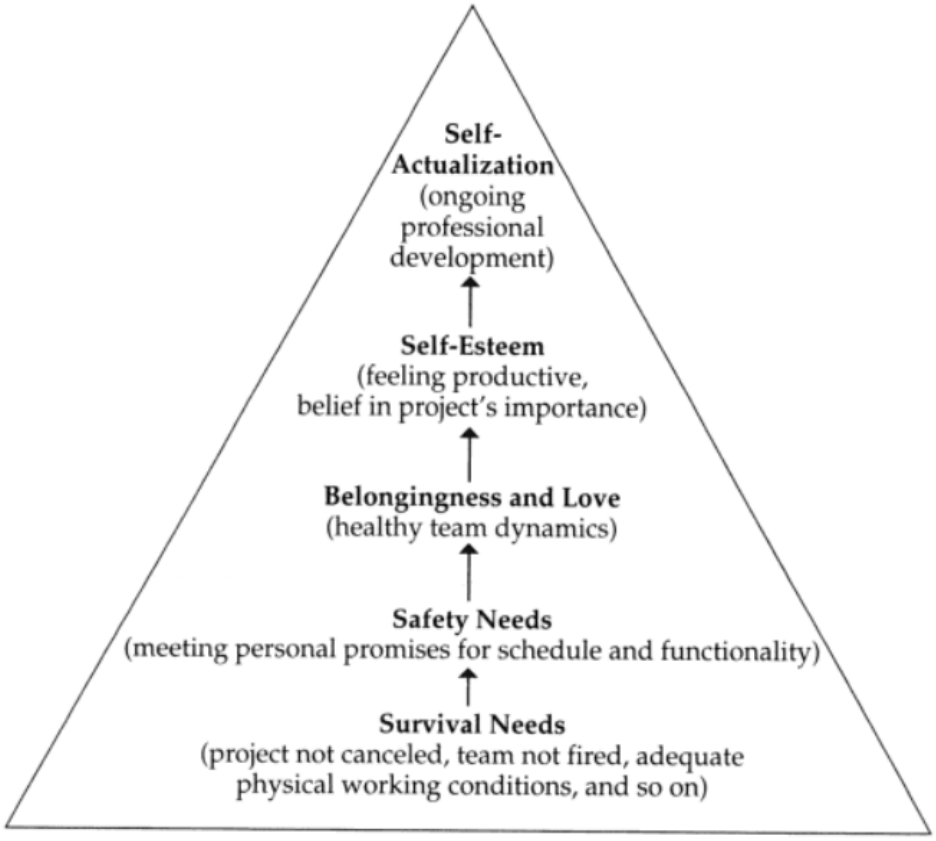


Fig. 9. Hierarchy of programmer's needs as seen in most "good organizations" (source: McConell 1997). The highest level: **identity with E-factors** is missing.

That kind of management is much more difficult than the other methods, and it requires a high level of interpersonal skills. However, it brings the required results, and the other management styles don't.

How do you make people identify with the organization?

There are several ways to do it. Some f them I know from my own experience.

- The employee must work for his own account rather than for the company account. In this case in the documentation of the created software is mentioned, who created which part of the product and who is responsible for that. That is currently used in my company. Because we don't produce the mainstream software for finances, accounting, sales, ERP systems, etc., but highly specialized software for control and optimization of technological processes, we have very close interaction with the customer, because we have to consult lots of technological details. The person who talks to the customer represents himself, not only the company he works for. By no means is he an anonymous "ant", "rat" or whatever animals are used to work in corporations. So he personally feels responsible for what he does and he works fist of all for his own account.
- When software is created by several people, in some companies it is clearly mentioned who created which particular modules. It also a very good way of making people identify with the organization and make them more responsible for what they do. They also feel more appreciated of everyone can read that he personally developed this module and not the company as a whole.
- Employees must understand the purpose of their work. They can't do things, which they believe are useless or which are not done in the way they believe they should be done.
- Employees must be informed about why some decisions have been made. For example that the company has to release new product earlier at the cost of some features, because that will generate earlier higher income and the money is needed at the time to invest in something else. When they are told only: "release the product earlier at the cost of cutting off some features" with no explanation, they will simply think the board went crazy are will be only disappointed that they have to cut some features off.

- No methodology madness. No-one will identify with an organization that treats him this way. This issue is discussed in the next chapter.
- Natural authority. This issue is discussed in the next chapter.
- Good work conditions. This issue is discussed in the next chapter.
- No stupid performance measurements and reviews, but measuring the real goal – successful project completion. This issue already was but also will be discussed in one of the following chapters.
- Building Good Developer Teams. There is also a separate chapter on that issue.
- Keeping the “smart start-up” rather than “corporate bullshit” working style. There is one more chapter on that issue.

3.5. Methodology Madness

If the developers aren't smart enough to think on their own, the work will fail. No Methodology will help. So what helps? First of all, the right team members and the right manager. Most people are hired by recommendations. That is very good, because rather trusted than random people get the job. However it still sometimes happens that a person is found to be inadequate for the job. In such a case no incentives and no methodologies will make him adequate. The only solution is to help him make the decision of relocating to another company.

On the other hand methodologies can do devastating damage to efforts in which the people are fully competent. They do this by forcing the work into a fixed pattern that guarantees:

• a lot of paperwork (Tons of documentation are part of the problem, not of the solution.)
• a lack of responsibility.
• a lack of creativity.
• a general loss of motivation.
• the best employees quitting the company

- and finally decrease of the company income
- or (as it happened in T, where I was working) the company bankruptcy

"Those who build methodologies are tortured by the thought that thinking people will simply ignore them." (DeMarco 1999) That really happens in many organizations. But the opposite possibility also happens in many organizations; people do exactly what the methodology says to do, even when they know that doing this will be pure waste of time, money or other resources. That happened lots of times, when I was working at Company Two and at Company Three. For instance, the methodology required writing unit test for each function (blogs.msdn.com/b/cashto/archive/2009/03/31/it-s-ok-not-to-write-unit-tests.aspx). There are of course function that return always the same result or return nothing, so from the logical point of view the unit tests for them are useless. Nevertheless, the management required the developers to write unit test even for that functions. The management also knew that this didn't make sense, but they told that the procedure required that and if something went wrong they didn't want to be made guilty, because of not following all procedures.

ISO is one problem generator, which I already mentioned in the previous chapter. Another one is Capability Maturity Model (CMM). "A maturity model can be viewed as a set of structured levels that describe how well the behaviors, practices and processes of an organization can reliably and sustainably produce required outcomes". There are five levels of CMM (en.wikipedia.org/wiki/Capability_Maturity_Model):

1. Initial (chaotic, ad hoc, individual heroics) - the starting point for use of a new process.
2. Managed - the process is managed according to the metrics described in the Defined stage.
3. Defined - the process is defined/confirmed as a standard business process, and decomposed to levels 0, 1 and 2 (the latter being Work Instructions).
4. Quantitatively managed
5. Optimizing - process management includes deliberate process optimization/improvement.

Organizations are under the pressure to get the highest CMM level, as well as to get ISO certificate. DeMarco writes that "The officers of the corporation believe that the CMM does not

come from Pittsburgh, but from God's own lips.“ (DeMarco 1999). But frequently keeping ISO standards, climbing CMM ladder or even worse implementing PRINCE2 leads to beaurocracy, decrease of effectiveness, decrease of employees satisfaction and increase of costs. With CMM there is one more absurd. It is easier to achieve a higher level of CMM when doing simpler and easier projects. So the company that wants to achieve a high CMM level must engage in simple projects. But on the other hand taking simple project leads to a gradual loss of competencies and capabilities to take complex projects. And complex projects are usually more profitable. They also are more welcomed by the developers, because they give them the feeling of not being the “code monkeys” but “rocket scientists”. The theory says “effectiveness by implementing ISO and CMM” (even not mention PRINCE2, ITIL and other similar inventions) and the practice usually shows something quite opposite:

Effectiveness ***xor*** (ISO ***and*** CMM)

Better ways to achieve improvement in product quality are:

- Technical trainings for employees
- Providing employees with appropriate tools
- Peer review (e.g. programming in pairs, or exchanging code segments among programmers to be verified by a colleague)
- Interesting tasks, because people perform better if the work is interested
- Employees must be required to get things done – no excuses. But the thing to be done must be the right things which generate income, not the substitute metrics.
- And of course already mentioned intrinsic motivation coupled with reasonable management

In 2007 I spent three-month vacations in the USA. I bought an old car there for that time to have something to travel with. Soon there appeared a problem with the engine. I took the car to one car workshop and the mechanic told me, that this is a rare car model and there is no error code on his reader and therefore he can’t repair it, because all the procedures he has only tell what to do

depending on particular error codes. Then I took the car to the second workshop, then to the third one and the situation repeated every time. Finally I found on the internet a car workshop ranking and took the car to the workshop that was listed on the first place in Los Angeles for several years running. The mechanic told me that this is a rare car model and there is no error code on his reader and therefore he can't repair it at this moment, but if I let him keep the car for two days, he would call his fellow mechanics and would search for the solutions on the internet end hopefully he would repair it. So he did. It of course was not profitable for the workshop to spend so much time on the single repair, but thanks to that approach, the workshop is on the first place in the ranking for many years and consequently must have much more customers than all the workshops that work strictly to procedures and in which the workers are afraid to take a risk of thinking on their own. I'm sure the manager of that workshop doesn't punish employees for mistakes or for spending too much time on a single repair. For sure he also doesn't pay bonuses for the number of repairs made. That is also an excellent example how quality improves productivity.

Some managers believe there is a trade-off between quality and productivity. The case above is a counterexample.

In (Śliwa 2001) we can find the next counterexample: Toyota make the employees free from working in the rigid procedure-based rhythm, so they didn't have to adjust to the procedures. That in turn changed the core of the organizational behavior from labor to innovation – the average number of innovations per year at Toyota was between 450 and 650 per 1000 of employees. This is above 250% more than in American automobile corporations. At the same time Toyota eliminated between 4 and 7 spokes in the management ladders, which were widely regarded necessary. Toyota cars belong to the most reliable in the world and productivity at Toyota to one of the highest in automobile industry.

One of the best ways to enhance productivity is to allow the developers to create high quality software and never urge anyone to deliver the software earlier at the cost of quality. If the developer thinks he quality should be higher than the customer requires, let the developer do it. It will take a bit longer this time, but I will pay off in the future; once by a satisfied developer,

who will work with more enthusiasm and second with a good opinion about the company products. The first factor will increase the productivity and the second one will generate additional money. The result of requiring the developers to work permanent overtime will be just opposite.

3.6. Mistakes and Creativity

A good manager understands that creatively thinking developers are the most valuable company asset. So first of all the manager must not disturb them in thinking. For thinking workers, making an occasional mistake is a natural part of their work. But the mistakes are good, because they lead to finding effective solutions. Writing software is not hamburger production, where all mistakes should be avoided. The only way to limit the mistakes is to limit the creativity. And the result of that is finally limiting the company income, because innovative software is most valuable. Punishing developers for errors means acting against own company. An atmosphere that doesn't allow for errors makes people defensive. They even don't try new things. That has also the negative consequence for the staff morale. They simply don’t like such a work and at the first opportunity the best developers will quit. The cost of replacing any of them will be on average equal to his six month wages. Moreover, becase the best will quit first, finding equally good programmers will be really a big challenge.

There are come kinds of software development projects, where some bugs in the application are acceptable. For example, while working at Company Three, we created software for discovering new plastic substances. The purpose of the software was to check several million of chemical compounds and their possible combinations with some numerical and stochastic methods to discover which ones maybe promising candidates. If the software hung from time to time or occasionally produced wrong results, it didn’t matter. It was hundreds of times more important that it could find some compounds that would be further checked in the laboratory. In this case it would be a pure nonsense to punish the developers for mistakes. Moreover, no unit tests could be used in that software, because no-one knew what should be the result of the function. That was the purpose of the software: to discover the results and not to work correctly for known results.

Just imagine, what would have happened if our manager had used the measurement management method and would like to stick to ISO and CCM: the software would never be created, the new chemical substances would never be discovered and the production would never start!

3.7. Work Conditions

Office

An office should be arranged so as not to distract people from their work. No phone calls, no repair men visits in the working hours. Just no-one to disturb. Open spaces are very bad for that purpose. There is always buzz in the open space office. Too many distractions. Moreover, rooms without windows are like a prison. That has a disastrous influence on productivity. “There are a million ways to lose a work day, but not even a single way to get one back” (DeMarco 1999). When the office environment is frustrating, people look for a place to hide to be able to work effectively. They book the conference rooms or go to the library or to the coffee bar just to work without being distracted. A programmer to work effectively must be in a “comfort zone”. Entering the zone may take up to 15 minutes and the zone can be left be any disturbance, as a friend asking for something. Thus, what really counts are not the total working hours, but the uninterrupted hours (DeMarco 1999, Spolsky 2004). For that reason office rooms with doors that accommodate two or three persons are the preferred solution. They are more expensive for sure, but the money spent on them brings a very fast return on investment by enhancing the productivity dramatically. If someone is more effective while working from home and he prefers to work from home, he should be encouraged to do so.

Fixed vs. Flexible Work Hours

Fixed work hours frequently lead to the fixed body time only but not the fixed mind time. It is known that fixed work hours decrease the developer’s work effectiveness. There are several reasons for that:

- People tend to feel that if they are at work, they are OK just by the fact that they are at work. In this way they don't feel enough responsible for the effects of their work.
- People not always can work in the time when they are most productive.
- People feel they are not treated seriously. They rather feel to be treated as preschool boys than as rocket scientists. That has a devastating influence on their morale.
- People feel that the requirement to work fixed hours is nonsense (and they are right, because it is nonsense). The real requirement should be to work effectively, no mater at what time, from home, from office, from anywhere. Time and location really doesn't matter. If it matters for the manager, it means that he is not focused on the real goal - to generate income. If someone is more effective while working from home or at night, he should be encouraged to work from home or come to the office in the evening (if it is possible). Coming to office in the evening may have the additional advantage that this is probably the situation with the fewest possible number of distractions.
- That limits freedom. No-one likes to have limited freedom, especially the best developers. For that reason the best developers simply don't work in companies that require fixed work hours. Moreover, they don't work at companies that require a specific number of hours to work, but rather at companies that require specific effect to produce.

When you work at a fast food restaurant, you must work in fixed hours and follow the procedures. But even at the fast food restaurant an exaggeration with the procedures can made customers mad and cause them to go to the competition the next time. Some managers forget that software development and hamburger production require quite different management strategy.

I never had to work fixed hours and I was always assessed by the results of my work not by the hours I spent in the office. It happened only once, when the boss at Company One changed (see previous chapters). But I took me only two weeks to find a new job. If nowadays I was forced to work fixed hours, I would do exactly the same: find a new job.

3.8. Start-up Style

Joel Spolsky (Spolsky 2004) writes a story explaining why big corporations deliver poor consulting and poor software. He writes about a man called Mike, who had hired a huge company of IT consultants to build system for him. The consultants turned out to be highly incompetent and kept talking about "The Methodology". He spent millions of dollars on the project and they failed to produce a single thing. Mike was really unhappy.

Then Mike found a youthful programmer who was really smart and talented. He built the whole system just in several days for few hundreds of dollars. Mike was overjoyed and recommended the youthful programmer to all his friends.

Soon the youthful programmer had more work than he could handle, so he hired a bunch of people to help him. Because the best programmers wanted to earn too much, he decided to hire even younger programmers right out of college and train them quickly. It turned out soon, that they still didn't know how to work, so Youthful Programmer started creating rules and procedures. Over the years, the rule book grew to a six-volume manual called "The Methodology".

After a few years, Youthful Programmer is now a Huge Incompetent IT Consultant with an established Methodology and a lot of people who blindly follow the Methodology, no matter if it works or not, because simply they have no bloody idea what else can they do.

And now their customers are unhappy. And suddenly a new upstart talented programmer comes and takes away all their business. And the cycle begins anew.

The question is: does it have to be always this way?

When a new employee is hired he can be asked a question: what do you want to do in our company and should be allowed to do what he wants. I was asked the question after being hired by Company One and was being asked this question all the time while working at Company One (as I already mentioned). In happened also at Company Three what I was allowed to join a project which I wanted and to use any programming language I wanted. Upon my hiring at Company Three I was also asked if maybe I have a proposition of a brand new I project I would like to start at the company. The question can also be formulated: "try to define your own role and your range of responsibilities within our company". It doesn't make much sense to ask this question someone at the entry job level, but for a top class professional it is the right question.

People tend to work to the Parkinson's Law. But not all people and not under all circumstances. If someone works because of his intrinsic motivation driven be the E-factors (inner energy, enthusiasm, emotion and excitement), will he be a Parkinsonian worker? Where Parkinsonian workers Bill Gates, Larry Page and Sergey Brin at the very beginning of their corporations? People are frequently made Parkinsonian by setting deadlines upon them. But when they have strong intrinsic motivation they work at the fastest pace when there is no deadline at all, but when they are told: "I know you are responsible. Therefore there are no deadlines and no methodologies that you must use. Work in such a way that you are happy and that we are happy from your work". Will the developers work effectively? The self-fulfilling prophecy works also in this case. The developers became as their manager believes they are: smart, enthusiastic and responsible or stupid, lazy and requiring strict supervision. Most of them will work very effectively and feel very responsible for their work and for the whole project, but a few, despite of the prophecy, quite on the opposite. The solution is simple: these who can't work in such conditions must be helped in relocating to another company. We don't need poor managers, but we don't need poor employees either.

In general the knowledgeable workers should not be punished for anything, especially for the mistakes they didn't do intentionally. The way to have good employees is hiring by recommendation only. Only people about whom you know or a person you trust knows are excellent specialists and responsible persons should get the job. No job advertisements on the net, no resumes, cover letters, interview tests and all the other rubbish. That minimizes the chance of hiring the wrong person to the extremely low level. However, if it still happens that

someone turns out not to be a good employee, no system of incentives and punishments, carrots and sticks will make him talented and responsible. The only solution is to help him relocate to another company. I don't say to fire him. Just to help him find a less demanding job and organize the transfer to a company that will be happy with him. And for many companies that use for example the command and control or measurement based management he can be a very valuable employee, for that simple reason that this kind of managements impose lower requirement as well on managers as on employees. That is of course at the cost of productivity, job satisfaction and salary, but let others do what they want.

The purpose of business is to make money and the best way to do it is to let the employees have fun. That is exactly what Google tries to do: they let their employees to play some games during their work day, to spend 20% of their work time on what they want, Google offers excellent food for free, etc. However, the most important is to let them have fun from the work itself. When people have fun from what they do they work most efficiently. For example, if given software requires say 100 hours to be written according to the specification and the developer wants to spend 150 hours on that, because he wants to implement some additional fancy features, he should be allowed to do it. This will make him happy and more creative and more efficient and will pay off in the future. At Google over 50% of new products derive from the ideas their employees came up with in the 20% of time, when they could do what they wanted.

To have fun nobody should be forced to work with a programming language or database or operating system or whatever else he doesn't like. Also to have fun, you cannot work on uninteresting projects. Will you have fun writing the 10-th internet shop in the same technology? Not very likely. The projects must be challenging and innovative. For that reason in my company we don't take any mainstream projects, as web pages, ERP systems, etc. We specialize in software for optimization of technological processes. Every project is challenging and innovative and requires much knowledge, not only in IT, but in the given technological process, and frequently in advanced data mining and knowledge discovery, sometimes the time is critical and the result of complex calculations must be returned within several milliseconds, so we have optimization problems and so on. In general at starting a new project no-one has the knowledge required to complete the project. Because of all of that the projects are interesting. But that are also "high risk projects" because the chance that it turns out that something is either impossible

or financially unreasonable is relatively high. Moreover, it would be extremely difficult to stick to ISO9000 procedures or to raise the CMM level with such projects. For that reason not many companies are interested in that kind of projects.

But with that kind of projects it is much easier to make the developers happy. And making developers happy is the key to a successful business. Bill Gates, Larry Page and Sergey Brin worked also on the very high risk projects at the very beginning. And they have fun from that, what finally resulted in the explosive growth of their businesses. However, after reaching a certain number of employees no longer everyone can work with so much fun and second, there are not so many smart people that can be found. And the story told by Joel Spolsky repeats.

Some people assess the maximum size of that kind of an excellent IT company for 20 persons, others for 50 persons. But no-one believes that big corporations can work this way. But that doesn't mean it is impossible. Long time ago people believed that TV broadcast, flying planes and the demise of communism in Eastern Europe are impossible and then all that happened.

3.9. Authority

In the best organizations, there is a natural authority based on skills and competencies and not on positions in the company hierarchy. That authority works in all directions. The manager is known to be good at organizing work and removing obstacles, someone is known to be good at writing high quality code, someone is known to be good at generating new ideas and someone is known to be good at make others to work smoothly without conflicts. When the all areas of expertise are recognized, the team becomes very successful and it is a pleasure to work in such a team.

The project manager maybe considered by others a member of the team or maybe seen as someone out of the team. The position of the project manager may also be taken by a different team member for each project – that is probably the best solution, because it ensures that no-one will ever try to cheat the manager or to play any game with him and vice versa. It was used in the

Company One, where I was working and is now used in my own company. However, the two first approaches don't have to be bad – it all depends on the implementation.

3.10. Developer Teams

Belbin's Team Roles

As it was already mentioned in the previous chapter, different persons have different talents. Sometimes it happens, that someone doesn't write very excellent code and doesn't write very much code either. However, every project he is involved in turns out to be very successful. There are two clues from that for project managers:

- Assessing developers taking into account only the metrics (number of created functions, bugs, etc.) is a great mistake.
- A set of diverse roles is needed in a successful team.

Belbin (Belbin 1996) specified the following roles:

- **Resource Investigators** are full of enthusiasm when projects starts and eagerly search for and discover new resources and opportunities. They are excellent at networking and can copy ideas from other teams.
- **Plants** are creative and generate new ideas. Plants are good at inventing original solutions, however they maybe poor at practical implementations and communication with others.
- **Coordinators** are good as team leaders. They can have a global picture of the situation and recognize other's abilities.
- **Shapers** are task-focused and committed to achieving end goals. They can shape other team members to help the team achieve the aims.
- **Monitor Evaluators** are analytical observers of the situation, resources and possibilities. They take everything into account while coming to the right decision.

- **Team Workers** are good listeners and diplomats. They are talented at smoothing conflicts and helping other persons understand each other without confrontations.
- **Implementers** simply do what others ask for, and turn their ideas into action. They are self-disciplined and deliver results on time.
- **Completers/Finishers** make sure that things are getting done. They understand the need for accuracy and usually do not any encouragement from others.
- **Specialists** are great funs of their own domain of knowledge. Thus, they have the greatest depth of knowledge and skills, which they are constantly improving.

Belbin suggests that teams should be created so to contain persons with complementary features. For example it is better to have one plant, one coordinator and eight implementers in a team than ten implementers. The Belbin Team Inventory tests can assess who is good at what particular team role. But a good manager who knows his people doesn't need any testing tools. He just knows who performs what role in the team and whom to add to or remove from the team if necessary. That of course doesn't have to be decided by the manager but can be decided by the team members themselves.

Teams and Motivation

One thing is worth mentioning here once more. I wrote before that replacing intrinsic motivation with an external (performance reviews) one is very destructive for the single person. But it is also one of the most devastating things that can be done to the team, especially when it differentiates team members. Giving only the chosen workers extra money for "extraordinary" accomplishments or programs like "employee of the month" can ruin team cooperation. That is not to say that everyone should earn the same, but that management should encourage team cooperation and not competition. Competition between team members is deadly to the project and to the work atmosphere. Competition can lead to people playing against each other at the cost of the team productivity.

There is something in the human nature which tells us to look for the guilty rather than look for the solution. If the management enhances competition, people tend not to talk to each other, but to communicate by email every time there is some problem or disagreement. Moreover, they frequently place their manager on the CC. What should the manager do if he notices that his team members play against each other and try to involve him in their game? Probably the same as a parents should do if their children behave this way. Even if grownups should in theory be more intelligent than children and do not play this game, in practice frequently they are not.

I think, that as well the project manager being in this situation as a parent should ask their employees (or children) to read the book "Feeling Good Together" by David D. Burns.

In many companies the conflict is not within a team (or department) but between different departments that are involved in the same project. For example the conflict between the Sales Department and the Development Department (or Service Department) more often exists than not.

3.11. Feeling Good Together

David D. Burns in his book "Feeling Good Together" (Burns 2008) performs a very interesting analysis of how do we treat other persons and what is the outcome of that. I dedicated the whole chapter to these issues, because the observations and suggestions of David D. Burns (as I already know from the experience of several people) can dramatically improve the quality of our life in the company and consequently the happiness and productivity.

He suggests performing a Blame Cost-Benefit Analysis to find out the advantages and disadvantages of blaming someone and consequently being with odds with him. Thus, "you can think of someone you are not getting along with, and list the advantages of blaming that person:

- You can tell yourself that you're being honest, because the other person is probably acting like a jerk. "Truth" will be on your side.

- You can look down the other person.
- You can feel a sense of moral superiority.
- You won't have to feel guilty or examine your own role in the problem.
- You can play the role of a victim and feel sorry for yourself.
- You won't have to change.
- You can try to bet back at the other person. After all he of she deserves it.
- You can be angry and resentful – anger is empowering.
- You won't have to feel guilty or ashamed.
- You can gossip about what a loser the other person is and get sympathy from your friends.

Of course some other advantages can be added to that list.
Then we can list the disadvantages of blaming the other person:

- You'll feel frustrated and resentful because nothing will change.
- The other person will feel judged and insist that it's all your fault.
- The conflict will be demoralizing and exhausting.
- You won't be able to get close to the other person.
- You won't experience any spiritual or emotional growth.
- People may get tired of your complaining.
- You won't experience any joy or intimacy because you'll be hopelessly enmeshed in the conflict.

Of course some other disadvantages can be added." (Burns 2008)

Then each advantage and disadvantage should be assigned a weight in points from 0 to 100, so that the weight sums up to 100. For example if the advantage of the blame seems much larger than the corresponding disadvantage assign to it 70 points and 30 points to the advantage. Then you should list the advantages in left column of the table ad the disadvantages in the right column, each in the decreasing order. You should also think about how the list feels as a whole,

disregarding particular weighs. Finally ask yourself whether the advantages or disadvantages of blame are greater (not which list is longer, because sometimes one advantage can outweigh many disadvantages of vice versa). Before reading further the reader is encouraged to do this exercise.

It turns out that almost every person notices that blaming others is very disadvantageous for him. Moreover, it can be easily seen that blaming others is a very ineffective way of getting things done. If an intelligent person sees that what he has been doing so far is ineffective, the only reasonable course of action is to change one's behavior.

Then David D. Burns presents The Five Secrets of Effective Communication. These "secrets" maybe used even to come to good relationships between someone from the development and someone from the sales department. The most important one is the "disarming technique".

1. The Disarming Technique. Find some truth in what the other person is saying, even if it seems totally unreasonable or unfair. It turns out (you can practice that if you don't believe) that the other person will no longer attack you, if you simply admit, that he is right. For example, he says: "Why haven't you written this procedure by now?". If you answer: "because you didn't give me the required information" or "because my boss asked me to do something else", you will get attacked again. But when you answer "Sorry, it's my fault. I didn't do it because yesterday I watched football world cup I totally forgot." – this will end up you quarrel. The other person seeing that you admitted will no longer try to blame you. Even if there were some other objective reasons besides the World Cup, don't mention them; it will look like you are defending yourself and because of the defense you will be attacked again. Just disarm, and it's likely that the other person first will notice that you are sincere and second it's likely that he will notice the other objective obstacles.
2. Empathy. You put yourself in the other person's shoes and try to see the word through him or her eyes.
    a. Thought Empathy. You paraphrase the other person's word.
    b. Feeling Empathy. You acknowledge how the other person is probably feeling, based on what he or she said.

3. Inquiry. You ask gentle, probing questions to learn more about how the other person is thinking or feeling.
4. “I feel” statements. You use “I feel” statements, such as “I feel upset”, rather than you statements, such as “You’re wrong” or “You’re making me furious”.
5. Stroking. You find something genuinely positive to say to the other person, even in the heat of a battle. You convey an attitude of respect, even though you may feel very angry with the other person.

You cannot use these techniques to manipulate people – that simple doesn’t work. They must reflect your thought and feeling and as such requires from you a genuine change of attitudes.

Also the Nonviolent Communication (NVC) developed by Marshall Rosenberg is worth having a look at (Rosenberg 2003) for those who are not familiar with that. Rosenberg writes that conflict between individuals or groups is a result of miscommunication about their needs, often because of coercive or manipulative language. And now let’s have a look at the plenty of courses for sales persons that teach them how to manipulate others to make them buy what you want. That is of course a one-time strategy. The manipulated customer even if buys something will never return. If the persons from the sales department are after such courses there is nothing strange in the situation that there are permanent conflicts between them and the programmers as well as between them and the customers. Well, some of them know that the “techniques” are rubbish and never use them, but still there is a conflict, because the sales department wants to sell everything to the customer without paying attention to all indirect costs and maintenance costs. On the other hand the programmers look at the whole picture. That is of course caused by the poor management that gives incentives to maximize local maxima, by rewarding the sales persons for what they sell and not for increasing total company income. But that is a frequent case. Rosenberg focuses on two things in the NVC: honest self-expression — exposing what matters to oneself in a way that's likely to inspire compassion in others, and empathy— listening with deep compassion. The result of following his advise maybe that programmers or persons form the technical department will understand persons from sales department and vice versa and they will ask together their managers or the company board to reorganize the incentive system. If their managers do this they will be happy. If their managers refuse, the awareness they can work at a better company that maximizes the global goal will give then an impulse to look for a new job and after finding it they also will be happy.

4. Agile Methods

4.1. Why Agile?

Storber and Hanssman (Storber 2010) proved that agile principles, which represent innovation and flexibility work better than traditional planning techniques (strict methodologies), but in spite of that most big corporation refuses to use them. The corporations use rather theory X (rewards and punishments for step-by-step implementation of the methodologies), than theory Y (which states that pursuing challenging goals is what people like and that is the real source of progress). In the official declarations of many corporate speeches theory Y is conjured, but at the same time, the working environment is based on theory X.

Fig. 10. Agile vs. Waterfall (source of the pictures is unknown)

Agile approaches don't solve all the problems and generate some new ones. However, they are definitely more efficient, at least in most cases. They include the human factor only to some degree, not in such details as it was considered in the last chapter. However, they enforce good communication, trust and responsibility and if implemented well they are good ground for the theory Y to flourish. The agile methods compare to the waterfall ones as the picture on the left below to that on the right.

The difference between agile and waterfall approach can be shown in fig. 10. The waterfall project looks like a cathedral. Everything is ordered according to the design. But what happens if a change is needed at the very low level? Everything above has to be disassembled to reach this level and then assembled again. That involves enormous costs. Another problem is that in every IT construction something will have to be reassembled for sure. Either before the construction is finished the customer will change his mind or it will turn out that the architect misunderstood the customer and then the bricklayers misunderstood the architect and the final construction doesn't match the customer's needs.

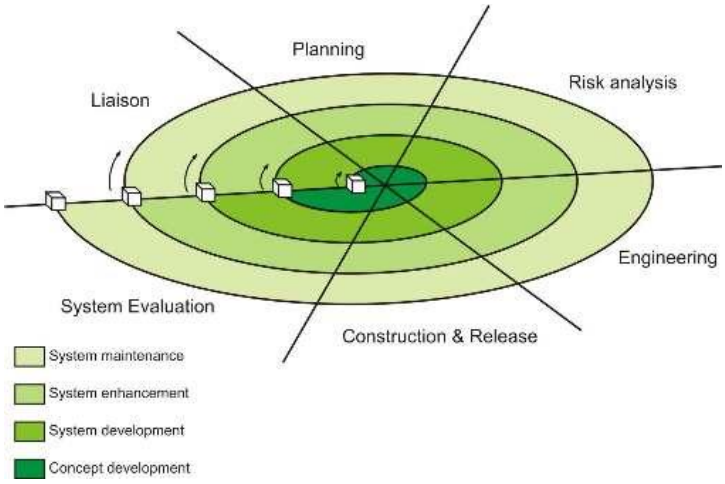


Fig. 11. The idea of Agile Software Development

On contrary in the market place on the right that symbolizes agile approach, the costs of rearrangement are minimal, also the time needed to do it is minimal. The builders also don't have to follow the rules so strictly. They have more freedom, so they can build a better construction, if they figure out how something can be done better. On the other hand it is more difficult to manage this project and also the workers must have more responsibility and creativity. The idea of agile software development is shown in fig. 11. Here all the phases shown in fig. 3. are merged together and performed in an iterative way. After every iteration the product gets better and better and more functionality is embedded into it.

4.2. Agile Manifesto

The Agile Manifesto was created to make better, more user-friendly and more effective software development environment. Four basic values stand behind the Agile Manifesto:

- Individuals and interactions are more important processes and tools
- Working software is more important than comprehensive documentation
- Collaboration with the Customer is more important than contract negotiation
- Responding to change is more important than following a plan

Based on these values the following manifesto was formed (http://agilemanifesto.org/ principles.html) by a team of developers including Kent Back, the author of the book "Extreme Programming Explained" and Alistair Cockburn, that author of the book "Agile Software Development":

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity - the art of maximizing the amount of work not done - is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

4.3. Extreme Programming

Extreme Programming is an idea of software development based on simplicity, communication, feedback, and courage (xprogramming.com/book/whatisxp). The Extreme Programming team keeps the system integrated and running all the time. The programmers frequently write production code in pairs, and all work together all the time. They code in a consistent style so after some effort put in the code understanding, everyone in the team can improve all the code as needed.

Extreme Programming is not a methodology in the sense of a lot of paperwork but rather in the sense of general rules: an iterative process for developing software, where everyone is responsible for the project and the project includes tight cooperation with a customer, who may even be one of the team mebers (www.jera.com/techinfo/xpfaq.html). XP is in fact a kind of methodology, although a lightweight one. Alistair Cockburn (Cockburn 2001) writes that XP is a “lightweight, low-ceremony, high-discipline methodology”.

12 core practices of XP:

- **Planning**: The developer team and the customer cooperate to produce software of the best business value as rapidly as possible. The customer comes up with a list of desired features for the system. Each feature is written as a user story. Developers estimate the effort of particular features and the customer decides which features he want implemented first.
- **Small Releases**: The working software is realesed early and often, adding a few features each time (not the entire ready system is released at once), see fig. 10.
- **System Metaphor**: a story that everyone - customers, programmers, and managers - can tell about how the system works.
- **Simple Design**: Always use the simplest possible design.
- **Continuous Testing**: First write a unit test and then implement the feature. As soon as the system passes the test, the feature is implemented correctly.
- **Refactor** every time you feel the code can be better.
- **Pair Programming**: All code is written by two programmers sitting at one machine.
- **Collective Code Ownership**: No single person "owns" a module. Any developer is expected to be able to work on any part of the codebase at any time.
- **Continuous Integration**: All changes are integrated into the main code at least daily.
- **40-Hour Work Week**: No overtime.
- **Customer is a member of the team**.
- **Coding Standards**: Everyone codes to the same standards.

Can you see the difference between PRINCE2 and XP? Clearly, XP is focused on simplicity, responsibility and avoiding formalized paper documentation.

I want to state clearly that XP is a very good idea, but following strictly all the 12 core practices is a very bad idea, because here we would again value the methodology over the project purpose which is to make everyone involved in the project happy (see introduction) with a minimal effort. The 12 practices are a guideline for typical situations only. If the project is non-standard, nothing will replace a common sense in deciding how far it can be built upon these 12 practices:

- **Continuous Testing.** Unit tests should be used only when it makes sense or rather when the developer thinks it make sense (blogs.msdn.com/b/cashto/archive/2009/03/31/it-s-ok-not-to-write-unit-tests.aspx). Especially the kind of software we create in our company used for optimization of technological processes or for discovery of knowledge in data is aimed to discover the optimal output of a function. No unit test can be written, because the correct output is unknown at the time of developing the software. It is the purpose of the software to find out the unknown outputs. Moreover, unit test can prevent only very simple bugs. Bugs that appear in practice are usually caused by the program entering some sequence of events with some input values the developer hasn't overseen. The bugs are not contained within a single function but are spread among several places in the code. For that two reasons unit test are useless in a lot of situations.
- **Refactoring.** Sometimes if something works it is really better not to touch it.
- **Pair programming** is OK, but it is not the only way to improve the code. For example: once I was written software in VBA to create a regression model. My friend implemented exactly the same ideas in java. He used not only a different programming language, but also different algorithm implementations to achieve the same purpose. After each step we compared the results generated by my and by his program. If the results were the same on a plenty of different data, we supposed the software is probably bug-free and can go to the production tests (only the VBA version, the java version was created only to help eliminate bugs in the VBA version). Standard programming in pairs can't do the job in such cases. You must be more flexible and still have two or more developers to work on the same code but not always in the way XP prescribes it.
- **One room**. (Not included directly in the 12 practices, but frequently thought to be a part of XP.) There is no need for the team to work in one room. I took part in several software projects, where the teams were spread over different countries and we communicated by Skype, email and other means. It really worked.

A typical XP approach usually works well up to about 12 (optimal) to 18 (maximum) programmers. Above that, the project should be split among two or more XP teams. Various ways of communication between the teams are possible. One team may be the customer for other team, but that is not always possible to arrange. Another option is to use Critical Chain Method

for the whole project and XP within particular tasks. This option will be discussed in the last chapter.

Fortunately, there is currently no XP certification authority. That allows replacing some of XP practices with common sense, if the project requires that and that allows not wasting time for paperwork. Should an XP certification authority be created, the XP probably will have to be replaced with something “unauthorized” to keep high team productivity and high software quality.

4.4. Scrum and Other Agile Methods

Another agile software development method, gradually gaining on popularity in recent years is Scrum. A Scrum project consists of iterations called sprints. The sprint cycle typically lasts between two and four weeks. At the beginning of each sprint a sprint planning meeting is held, where the developer team together with the customer select what work is to be done in the sprint. The customer can change his mind during the project, but only at a print startup. During the sprint the requirements are frozen. In addition to the meetings that begin and end the sprint, there is an everyday 15-minutes meeting of the team. During the meeting, each team member answers three questions:

- What have you done since yesterday?
- What are you planning to do today?
- Do you have any problems preventing you from accomplishing your goal? (if a problem occurs it is resolved after the meeting only with the involved persons, to keep the meeting under 15 minutes)

Scrum is a more formalized method than XP and therefore some people don’t like it. People also don’t like the strict Scrum time framing.

There is a bunch of other more or less formalized and more or less agile software development methodologies, but they are seldom used (RUP) or very seldom used (the other listed methods) in practice and therefore there will be only listed here without any discussion:

- Rational Unified Process (RUP)
- Agile Unified Process
- Rapid Application Development
- Lean Software Development
- Dynamic System Development Method (DSDM)
- Feature Driven Development

5. Conclusions

Since a good company strategy is based on system thinking, the project management should be obviously based on the same principles. However, project management needs some ways of implementing the system thinking ideas into the real world.

Critical Chain Project Management (CCPM) seems quite a reasonable implementation method. It has two main advantages: only one buffer at the end of the project and maximization of the final objective, not of local maxima. But, it has also one important drawback: it is not an agile method; there is no place for the iterative process.

Agile methodologies, as Extreme Programming (XP) use iterations, what makes the cooperation between team members and between the team and the customer closer and ensures that the project is moving in the right direction. The drawback of XP is that it is difficult to set deadlines and to control very big projects this way.

A big advantage of XP (or its modifications) is that the emphasis is here not on methodology and paper documentation, but on responsibility and cooperation. This approach requires good behavior from the manager and from team members. If people are to work efficiently they must identify the project goals with their personal goals, they must do what they like to do in an atmosphere they feel good in. That makes them responsible and effective and makes the methodology madness and tons of paper unnecessary.

A good solution seems to be using together CCPM with Agile Methods/XP together. CCPM for the high level picture (a long view of the project, to schedule releases, to control resource utilization, to control pace and allocate staff, a way of setting priorities between projects, to reduce multitasking) and Agile Methods/XP for managing work well, for developer motivation and for flexibility (incorporating customer into the team, ensure progress simplicity, good communication, efficient re-factoring and keeping track of the direction the software

development moves to). In this way the developer team can feel good, working under the umbrella of the agile approach with intrinsic motivation, responsibility and cooperation. While the care of keeping the project milestones is taken with CCPM.

That is also a workable approach in a case of big project. Particular teams work within the agile framework, while the whole project is managed with CCPM.

The main point is however that the crucial factor in software development project is peopleware: making developers feel happy and system thinking: seeing the final goal, not parts of it. A way to be happy is to pursue challenging goals. The driving forces in the pursuit are the E-factors: enthusiasm, energy, emotion, excitement and the feeling that you realize your own goals while working on the project. Thus the project must be organized so that people can identify the appropriate part of the company goals with their own goals. That is of course not necessary and many companies don't do that. But the cost they pay for this is that they use effectively only several percent of the developers' productivity and even less of their creativity. System thinking says that strict methodologies try to optimize local maxima that never lead to the global maximum. We must avoid methodology madness and build an intelligent organization where people know how to learn and how to be creative and innovative – the only way to beat the competitions. ISO, CMM, PRINCE2, measuring lines of code and whatever set of strict rules won't do the job, but if applied too vigorously will kill the creativeness and innovation. Though basing on some methods like CCPM with XP to a reasonable degree can help manage the projects, we must keep in mind that system thinking must go before any methods. We must always see the whole system and maximize its real goal: to be happy and rich.

6. References

Books:

- Peter M. Senge “The Fifth Discipline: Art and Practice of the Learning Organization”, Random House Business Books, 1993
- Tom DeMarco and Timothy Lister in their book “Peopleware: Productive Projects and Teams”, Dorset House Publishing, 1999
- David D. Burns, “Feeling Good Together: The Secret to Making Troubled Relationships Work”, Broadway, 2008
- Kazimierz Śliwa, “O organizacjach inteligentnych i rozwiązywaniu złożonych problemów zarządania nimi”, SIG, 2001
- Mariusz Flasiński, “Zarządzanie projektami informatycznymi”, PWN, 2006
- Eliyahu M. Goldratt, "Critical Chain", North River Press, 1997
- Meredith Belbin, “Team Roles at Work”, Butterworth-Heinemann, 1996
- Charles Fox “Prince2. A No Nonsense Management Guide”, Van Haren Publishing, 2007
- Marshall B. Rosenberg, “Nonviolent Communication: a Language of Life”, Puddle Dancer Press, 2003
- Joel Spolsky, “Joel on Software”, Apress, 2004
- Robert D. Austin, “Measuring and Managing Performance in Organizations”, Dorset House Publishing, 1996
- Mike Daisey, “21 Dog Years. Doing Time @ Amazon.com”, Harpen Collins, 2002
- “The U.S. Army Leadership Field Manual”, McGraw-Hill, 2004
- Bernard Girard, “The Google Way: How One Company Is Revolutionizing Management as We Know It”, No Starch Press, 2009
- Kent Beck, “Extreme Programming Explained: Embrace Change”, Addison-Wesley, 2001
- Alistair Cockburn, “Agile Software Development”, Addison-Wesley, 2001
- Thomas Storber, Uwe Hansmann, “Agile Software Dvelopment. Best Practices for Large Software Development Projects”, Springer, 2010

- Mike Cohn, “Succeeding with Agile: Software Development Using Scrum”, Addison-Wesley, 2009
- Steve McConell, “Software Project Survival Guide”, Microsoft Press, 1997

Online resources (all accessed between Sep 1, 2010 and Sep 15, 2010):

- http://agilemanifesto.org/principles.html
- http://xprogramming.com/book/whatisxp
- http://www.noop.nl/2008/05/top-five-reason.html
- http://www.siliconglen.com/news/2008/01/prince2-agile-common-sense.html
- http://myweb.tiscali.co.uk/spains
- http://www.best-management-practice.com/gempdf/DSDM_White_Paper_v2.pdf
- http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf
- http://www.ricksteves.com/about/pressroom/activism/eurodream.htm
- http://www.linkedin.com/answers/business-operations/project-management/OPS_PRJ/440462-7863114?browseIdx=0&sik=1243922341320&goback=.amq
- http://courses.cs.vt.edu/~cs4704/jss.pdf
- http://www.belbin.com
- http://www.articledashboard.com/Article/PRINCE2-vs-PMBOK/478463
- http://www.pmacademy.co.za/prince2-vs-pmbok
- http://corporategeek.info/Prince2-Certification-Nutshell
- http://www.best-management-practice.com
- http://prince2.technorealism.org/index.php?page=Prince2_Exam_Questionsnt/
- http://www.hojohnlee.com/weblog/archives/2005/10/02/ten-views-of-project-management
- http://www.joelonsoftware.com
- http://en.wikipedia.org/wiki/Systems_thinking

- http://en.wikipedia.org/wiki/Critical_Chain_Project_Management
- http://en.wikipedia.org/wiki/Belbin_Team_Inventory
- http://en.wikipedia.org/wiki/Project_management
- http://en.wikipedia.org/wiki/Critical_path_method
- http://en.wikipedia.org/wiki/Program_Evaluation_and_Review_Technique
- http://en.wikipedia.org/wiki/Student_syndrome
- http://en.wikipedia.org/wiki/Parkinson%27s_law
- http://en.wikipedia.org/wiki/Maslow%27s_hierarchy_of_needs
- http://en.wikipedia.org/wiki/Capability_Maturity_Model
- http://en.wikipedia.org/wiki/ISO_9000
- http://en.wikipedia.org/wiki/Two-factor_theory
- http://en.wikipedia.org/wiki/The_Principles_of_Scientific_Management
- http://blogs.msdn.com/b/cashto/archive/2009/03/31/it-s-ok-not-to-write-unit-tests.aspx