

---

# **Silver Bullets Toolkit**

Reference book  
of Successful Project Solutions  
in the Process of Software Development

**Constantine Berlinsky**

---

To my family,  
in whom I always felt  
the support and aid

Text version: 1.37e  
28/08/2004

## TABLE OF CONTENTS

1.	Foreword for the English Edition .....	5
2.	Annotation .....	6
3.	Introduction .....	7
4.	The reasons why this book was written..	10
5.	Original Idea .....	15
6.	Gratitude .....	18
7.	Software Development Methodologies ...	20
7.1.	RUP .....	21
7.2.	XP .....	23
7.3.	SADT .....	27
7.4.	MSF & MOF .....	28
7.5.	Iconix .....	29
8.	General Solutions Area .....	30
8.1.	Lifecycle Stage –Management .....	31
8.1.1.	Team Selection .....	32
8.1.2.	Division of Responsibilities .....	33
8.1.3.	Atmosphere within the Project ....	34
8.1.4.	Career Growth .....	35
8.1.5.	Labour Productivity .....	36
8.1.6.	Communication.....	37
8.1.7.	Planning .....	38
8.1.8.	Process Organization .....	39
8.1.9.	Developers Functions.....	40
8.1.10.	Personnel Training.....	41
8.1.11.	Task Orientation.....	42
8.1.12.	General Project Environment ....	43
8.1.13.	Intensity of Work .....	44
8.1.14.	Priority System .....	45
8.1.15.	Documentation .....	46
8.2.	Lifecycle Stage – Analysis.....	47
8.2.1.	Presentation of Information.....	48
8.2.2.	Strategy of Progress.....	49

8.2.3.	Two Points of View .....	50
8.2.4.	Glossary .....	51
8.2.5.	Diagrams .....	52
8.2.6.	CASE-Tools .....	53
8.2.7.	Use Cases .....	54
8.2.8.	Re-engineering of business processes	55
8.3.	Lifecycle Stage –Design .....	56
8.3.1.	Object design .....	57
8.3.2.	Design Patterns .....	58
8.3.3.	Component Development.....	59
8.3.4.	Conceptual Integrity.....	60
8.3.5.	Distribution of Errors .....	61
8.3.6.	“Wrong” Solutions.....	62
8.3.7.	Invention of Wheel.....	63
8.3.8.	Algorithm.....	64
8.3.9.	Decomposition of System.....	65
8.3.10.	OOP.....	66
8.4.	Lifecycle Stage – Coding .....	67
8.4.1.	Coding Standard .....	68
8.4.2.	Collective Code Ownership .....	69
8.4.3.	Pilot-project .....	70
8.4.4.	Smart Instrument .....	71
8.4.5.	Data structure .....	72
8.4.6.	Test Projects .....	73
8.4.7.	Pair Programming .....	74
8.4.8.	Code Refactoring .....	75
8.4.9.	Incremental Development .....	76
8.5.	Lifecycle Stage –Testing .....	77
8.5.1.	Continual Testing .....	78
8.5.2.	Tests Automation.....	79
8.5.3.	“Narrow” Tests .....	80
8.5.4.	Data Set .....	81
8.5.5.	Program Environment.....	82

8.5.6.	Defect Tracking.....	83
8.5.7.	Usability .....	84
9.	Conclusion .....	85
10.	Bibliography .....	89
11.	Copyright.....	98

# 1. Foreword for the English Edition

Previously this material was published in Russian.

- Text:  
[www.vbstreets.ru/Projects/NSP\\_Book/default.aspx](http://www.vbstreets.ru/Projects/NSP_Book/default.aspx)
- Discussion:  
<http://bbs.vbstreets.ru/viewtopic.php?t=8337>

After Russian edition I decided to release the English version with the help of a translator. Thus, the two tasks will be solved.

On the one hand, much more people speaking English will be involved in discussing the text.

On the other hand, in last 10 years I have read so many books with terrible translation (from English into Russian), that historical justice requires to be obtained ;-)

That is why I beg your pardon for some possible inconvenience connected with uncorrected translation of the text. As much as it depended on me, I tried to use the original English names in the links on the books and articles. In any case I hope that Cathlin Melamewka (See the article [10]) will forgive me (her name will be certainly spelled wrongly).

## 2. Annotation

The wars between IT-methodologies do not calm down. Every several years we receive absolutely new, quick, simple, and effective methodology (or new version of an old one). And it finely should solve the main problem – the creation of the qualitative software on term.

I think that the single truth concerning methodologies is that they do not exist at all.

There are only successful project solutions – SPS – that can work (or not) in a particular situation or project. The goal of this reference book is to collect them, give them a brief description and make IT-community to find them and classify.

### 3. Introduction

'Well, - you will say after reading the headline of this book. - There is one more new profit appeared, one more impostor who tries to teach us how to live, how we should implement the software projects! We have already found the methodology that copes with all the problems. We adapted it for our needs and there is likely to be less problems.'

And you will be right... but not completely. In no way I consider myself some kind of a new Messiah who will finely tell everybody how to achieve a success.

But I am really interested in a method of effective software development (and as a result of this knowledge I am interested in extending the level of my professional skills as an informational system developer).

This book is not a written justification against a certain methodology. But before, on the forums of the special sites I permitted myself to express my rough point of view about different up-to-date development methodologies that were supported by their staunch defenders with a religious fanatic ardour.

You will not find here any arguments for a certain methodology, as can be supposed, taking into account that for a long period of time I was a faithful supporter of the RUP methodology, I learned it and took an active part in its introducing in activity of the company where I was working.

Moreover by that time I was in the firm belief that a strict divisions of the team members working over the project into separate roles and appropriate functions could make the process of development

easily controlled and successful while their participants – satisfied with the work.

In this connection with my book I have the risk to incur just anger of defenders and supporters of the existing methodologies of software development or those that will appear in the future.

However, everybody has the right to express his own opinion and I will use this right.

**I am sure that the truth about different methodologies is that they do not exist at all.**

But now let help those who has fainted away to come around and confess to themselves what is the newest methodology of software development about which you have got to know from the latest marketing statement of a company.

Or what is the methodology that you use now in your day-to-day activity, where you have put many resources (learning materials, courses for key specialists with business trips in other city/country and finally the most important thing - time)?

You think that the methodology plays an organization role, that clearly prompts how you should work in order to achieve success in the field of informational technologies. And finally you hope to take a competitive priority “throwing dust in eyes” of the potential customers by the phrases difficult for understanding, such as “We are on the 6<sup>th</sup> level of CMM”, “Re-engineering of the business processes”, “Automation of the chaos by means of function separation in the alternative activities”, etc.

However running the same methodology in different organizations and projects (frequently in

development stages of the same project) shows for some reasons absolutely different results. Those things that perfectly work in particular cases and serve as motivation, in other cases, on the contrary, are obstacles.

What is the main reason, you could ask. I think that the project success depends on the following:

1. Resources available (first of all it is the developers quality and the second – the time)
2. Way of their interactions

If there are resources available and they collaborate with the maximum efficiency I think the project stands a better chance to result in something valuable.

As for methodologies, it seems to me that all of them describe the final collection of different efficient use methods of scarce resources.

My point of view consists in the fact that number of these methods (successful project solutions SPS) is infinite and you should not restrict yourself to the subset in the context of the methodology X. if we wish to advance in the field of successful program development we should gather these solutions (in a similar way as pattern) and learn to use them in a necessary situation.

This book is an effort to collect all the methods of successful development clear for me in a single place.

I wish you pleasant reading and good luck!

## 4. The reasons why this book was written

This book was written in order to collect in a single whole those things that I call "gold crumbs of knowledge", dispersed in the great number of sources such as Internet, literature and some kind of folk arts.

Phrases like "Two heads are better than one" or principle "divide et impera", known as early as in the Rome Empire, contributed in the development of program engineering more than both Microsoft and IBM.

After reading any book, article or message on the forum I was always interested in what useful I could get from this source exactly. Does it contain any useful thought unknown for me before? There were luckily to be new ideas in the most cases, but unfortunately they were diluted with secondary information that confused the issue.

Therefore after reading the next work I tried as far as possible to make up a summary with a list of the main ideas (unknown or not very obvious by that moment for me) that the author tried to express.

For example, here is the following result I have got after reading the book [3]:

1. The business-process description in text is much smaller than graphic format (it's really truth – the greatest working problem with diagram was that the printer did not support A3 and A2 formats).
2. Customers will read the text, understand and sign it (admit or express their claims) faster than learn UML (for example, I

spent a lot of time for explaining the include/extended link on usecase diagrams.

3. A new employee will easily learn how to write the text in the format of Cockburn's usecases than make him use UML and the supporting product correctly (for example, Rational Rose – graphics editor that leaves much to be desired).
4. Good classification of aims – you should always work on the same goal level. The level comes up if you ask the question “Why?”, and comes down if you ask “How?” (this is the great art to be on the needed goal level – the diagrams become smaller and more general or too big and detailed).
5. An excellent, intuitive comprehensible pattern of use case description (main scenario from 10 steps without "if" + main scenario extending + additional information).
6. An excellent idea concerning methods of multivariate analysis of the requirements collected (for example, with the help of electronic worksheet)- use of sorting, grouping in different attributes (importance, term, module, role)
7. And finally, I consider that the most important thing is that the lower the goal standard is, the less useful and obvious diagrams are. It follows from this the idea of dividing the task between different CASE-tools: for creating diagrams of

higher level (business process scenario, schemes of traffic, diagram of main document mode, collaboration between program module) to use tools effectively applied for drawing diagrams, connections between them (Rational Rose, MS Visio), but for more detailed description to use Cockburn's use cases.

I must confess that writing this book was a quite risky business: every time I can get a stab in the back from someone with 20 years of experience in development who devoted much time of his life for promoting the X methodology and can hear the phrase like "you should make as many project as I did and then you would offer the solutions of yours".

My general age does not exceed 24 (including only 3 years of commercial software development experience) and I have 5 embodied projects behind myself. According to "common standard" I should wait for my 40s and then express my thoughts in written.

But to hell with all these rules! You should live here and now and if you get the chance to do more than you can, do it immediately. There is little probability that you will lift a big iron case (see the movie «One Flew over the Cuckoo's Nest»). But you will not ever succeed if you do not make at least an effort.

I am really interested in my profession and every day long I try to make my life (and professional activity) better and happier as more as possible. Of course, it is foolish to think that writing something like book, article or something like "advertisement to myself" will increase my or somebody else's education

(more details see in the article [2]). But nevertheless it has the sense.

Release of the book will enlarge my “virtual learning” (in other words what attitude the potential employers will have towards me). All of these things (meeting new persons, receiving new information) extend fair chances to increase my real education i.e. real benefits that I bring to the projects which I take part in. Enlarging real education incites me to publish new materials. And so on.

By this book I would like to prove (most of all to myself) that our world is much easier than it may seem for the first glance. It is possible to succeed in spite of those obstacles that can be encountered in our way.

There is a story about a millionaire who said: “I can tell you how I earned every million of mine, except the first one”. Indeed if you perceive the life success as a goal for climbing up the endless stairs, it is really difficult to make the first step. I tried to make this book as an exam for the right to climb up the first step.

And finally, it is a groove to write a book like to compose music, draw a picture or sculpture.

Actually the beauty will save the world, software projects and me.

When you see how fragments of thoughts and phrases after patient processing (with painful search of synonyms and making up participles) turn into a coherent text with a logical plot, you derive great incomparable enjoyment.

Programming gives this kind of pleasure as well. It is connected with the moment when disembodied data, requirements, orders, instructions,

rumours and fantasies related to the system are like huge heavy stones suddenly become to form in your head a single sculpture. Every particle of a puzzle finds its own place and it remains only to bring them back to life. The stone statue wakes up and hand-made creature begins to take its first steps. This is the greatest pleasure of our profession.

## 5. Original Idea

Initially I had the following idea concerning this book (I had intention to release only paper variant of the book): firstly, to make an overview of up-to-date methodologies of software development, and secondary, to express the main idea that all of the methodologies use one of the successful project solutions. (See the fig. 1).

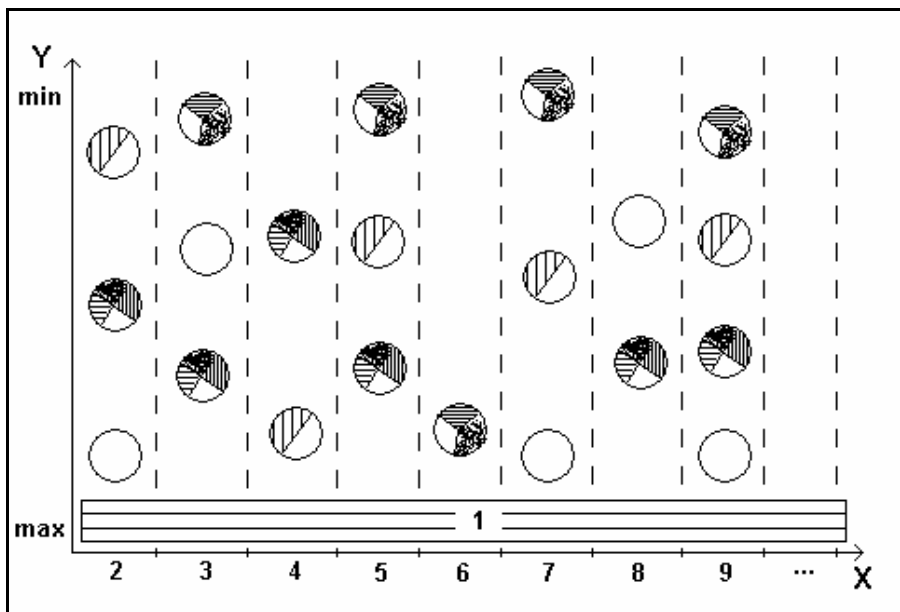


Figure 1. Successful project solutions area

**X** – axis of software lifecycle stages;

**Y** – importance level of SPS for a concrete stage

**1..9** - Lifecycle ( 1-project base «Management»);

**Circles** – SPS known by this time;

**Shadings** – different methodologies of software development including the solutions in their postulate list.

In the middle of the book I planned to leave five blank sheets. Every sheet would be a form for entering a successful solution discovered by a reader (rather a writer) in the process of developing the next solution. The form would be standard: name, code of lifecycle stage, effectiveness evaluation, description and additional information sources for solving the problem.

The concluding chapters would be "Table of Contents" and "Bibliography". Of course, these chapters would be filled in manually. Everybody has his own "golden set" of books, WEB resources and phone numbers of neighbouring pizzerias.

The idea lay in the fact that everybody would write his own book and even insert his initials in the cover.

But later I looked through my home library and found that the idea concerning book addition by a reader had been already made a reality. These are so-called "Notes". I did not have notes at all. That is why I decided to refer to the creative work more professionally, and thus, the book gained the present form.

Also my plans included (in the case of positive evaluation of my book by qualified developers and free time available) further development of the theory about "general successful project solutions areas" and its release in the form of the Internet web-site. This site would be some kind of a portal for information exchange between software developers and supplement in databases of successful solutions. The main components of the site would be the following: developer forum, divided into different themes (Lifecycle stages, Methodologies, Products, and etc.),

Guest book, User Profiles and the most important thing – Database of successful project solutions, available for publicity. The site user would value every new solution (or changes in description of old one), and in case of the positive result it would be added to the database.

However I've got the thing that I've got. As for book continuation, I think that it would be written in the moment when my professional experience would be brought to the qualitative new level. May be it would take 20 years (as in the case with F. Brooks) or less. By the way, as for Brooks: did you noticed differences between dedications of 1975 and 1995? In the first one the author mentioned his direct boss, while in the second – «Nancy, God's gift to me».

At last the family and all the things connected appear in the list of life values. But does "the core idea of the program engineering" lie in this?

## 6. Gratitude

Thus, here are the persons who influenced my growth much and whom I bless for this:

- My parents, they gave me birth ;- ) I prefer not to live in the times when the world terrorism is victorious, but there's nothing to be done.
- My sister for support.
- Technical University of Moldova – for higher technical education not to be very good, but better among others in the Republic. In the book [15] the author says that you should set yourself an object, get the education as good as it is possible, and then for God's sake do something!
- Stratan Victor for excellent explanation of theory and help in practice of design patterns, OOP and etc. Thanks to him for the first version of Rational Rose 98.
- Serdtsev Vitaly – after watching his "magic" with assembly programs, C++, low-level programming, I took a great interest in my own profession and have not been regretting still. I recall the words of our department manager V. Beshliu: "You came here in the University with different aims, but in the case of graduation you will be fervent patriots of your profession".
- Smolov Alexander – when the balance between joys and sorrows of a profession moves in favour of the last (idea from the book [6]), I sit down to my computer and struggle against aliens in my old kind XCOM (see website [15]), whom Alexander brought me on 3 diskettes with little elephants. Thanks to it all sad thoughts

disappear, but you keep dreaming about little green men for a long time. ;-)

- Dragoner V.V. – my thesis tutor. I set a pride aim to create an analogue of Rational Rose. I wrote a compiler of incoming C++ texts, Serdtsev – graphics for browser of objects and diagrams, Smolov – Russian sounds for insonification analysis results of texts through MSTextToSpeech. This project happily failed (by the thesis presentation only 50% of requirements were fulfilled), but sometimes misfortunes are more fruitful than success. I gained an experience in working with Rational Rose, read a lot of materials concerning Rose and s. o.
- Paprotskii I.B. and Starashuk A.I. – my work in government enterprise Registru (see the article [1]) under their direction had a great impact on my professional activity and career growth.
- Zolotuhina E.B. for brilliant course of system analysis and information system development.
- Edward Durlleshteanu (BitGenerator Company) – it was difficult but quite interesting to work together. I regret that difficulty outweighed interest.
- And finally, Igor Toporets, by the present my direct boss and a real professional in his field. Many of my aims I could achieve faster thanks to him. I also hope that I helped him to achieve his aims.

## 7. Software Methodologies

## Development

## 7.1. RUP

RUP - Rational Unified Process.

RUP was created in 1996 by the Rational Corporation with the assistance of Grady Booch, Jim Rumbaugh, Ivar Jacobson. It is truly fundamental work describing successful methodologies of software development. Software lifecycles, workflows, roles, activities, artifacts, and products supporting most of the lifecycle stages. This methodology is called "heavy". It supports the UML. According to the general volume and significance, RUP as an independent knowledge base can be compared to MSDN.

The main idea of RUP is to assign the work of each team member. To my opinion, the greatest problem in this case is that it is quite difficult or even absolutely impossible to find people who will do only things they are asked to do. How much time they need to be tired of monotonous working? The sense of responsibility (as well as satisfaction with results) for the work completed in the framework of fixed working conditions is lost, isn't it? Does the availability of coordinated interfaces serve as a quality assurance and effectiveness of work?

First of all, the products supporting this methodology are the products of the Rational Company: the basic product Rose (used almost in all development stages), SoDA (Software Documentation Automation), RequisitePro (Requirements Management), ClearQuest (Change Request), ClearCase (Configuration Management), Administrator (Project Repository Management), WorkBench (Project Management), Quantify (Profiling Function Performance), Purify (Detecting Memory Errors and

Leaks), PureCoverage (Monitor code coverage), Robot (Executing Test Suites), SiteLoad (Load Testing), SiteCheck (Dead Links Check). The RUP product called Rational XDE integrated in the Microfost.NET environment is now available.

## 7.2. XP

XP - Extreme Programming.

As regarded, this discipline of software development appeared, in other words its official registration took place in 2001, when in USA, Utah State, 17 supporters of agile methodologies worked out a manifesto with the main postulates. Kent Beck, Ward Cunningham and Ron Jeffries can be considered to be the ideologists of this method.

The main principles are the following: close communication, continuous testing, minimal documentation, maximal flexibility. Nevertheless it is better to present a text of this manifesto (see the book [4] ):

«We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more».

These are brief and clear explanations that make the proposed ideas available for broad masses.

At first XP suggested some revolutionary new principles of development. "It will not be necessary for you", "You should find the simplest solution that may work", "The Customer can change his requirements at any time", "Either of the developers sitting next to each other can change whatever they like in the system", and so on.

However I have just expressed my criticism concerning XP before. Many claims raised to XP will be withdrawn after more detailed acquaintance with it. The last projects in which I used to take part, was regarded to develop in the manner of XP.

There is the following criticism left:

- The idea of presence a customer and programmers side by side in one room is a fantastic wish, as to my opinion. Nobody agrees that collaboration with a customer is a vitally important matter. But solving the problem lies as far in the sphere of the documents standard forms development on the collaborating level, and, on the other hand, in the faster integration (it is necessary to help a customer to define the requirements and correct his system view).
- Denial of "big prior design" stage is accepted only in the process of trivial system development (easy sites with emphasis on design, not programming, elementary one-user programs with minimum business-logic, etc.). At this point I'd like to quote from one of the authors: "Here I consider the decisions useful for development of complicate systems. If the application is found to be easy, why I should spend my time on it? " In a complex

and interesting project it would be a criminal negligence not to create a framework of the system and the core principles of its functioning from the very beginning. Of course, real XP supporters would perceive with courage the information that in the center of the project development there is the use case appeared that makes to change the most part of code (or even worse to remove it). But I'm inclined to think that it is not acceptable.

- Dozens of userstories (sheets of paper with a few sentences characterized the use case), remained after the project completion can not be regarded as a reliable documentation. It turns out that XP focuses only on the agile software development, but not on its maintenance. In the book [4] I came across the example of a failed project 3C implemented with the help of XP (salary accounting for Chrysler, I confirm that salary, in spite of its seeming simplicity, is one of the most complex branches of book-keeping where no a single team of programmer has been lost). The author says that when quite many collaborators left the service, unwritten project data and team memory were lost. As far as I can judge, apologists were over-diligent with minimization of documentation. Serious developers cannot trust the things that are considered by students attractive.

I would like to share my impressions of the work in the manner of XP. In comparison with RUP (or any other methodology with fixing stage results with

the help of a requirements specification or design project, etc.), XP gives you the sense of uncertainty and anarchy *in the beginning* of the project. Business requirements and architecture change so quickly that after sitting a couple of days at home you may not recognize the structure of the basic classes (even if you have created them by yourself).

At once you begin to be aware of the XP postulate concerning 40-hour workweek without overtime work. What for you should toil at a module up to 10 p. m. that you will not need tomorrow?

*In the middle* the architecture becomes stabilize and *the end* of the project is characterized by the confidence. No requirements change of a customer seems to be terrible. During continuous modifying architecture *has to* be processed accordingly all the changes to be implemented with the maximum simplicity.

Programmers take a role of users working with a program design created by them. If they detect a flaw it is difficult to make changes in the system in the process of development and he may suffer some troubles. As a result, the design improves *forcedly*, and the system undergoes any change in business requirements.

There is no a product supporting this methodology. And it is likely to be the most important benefit. Indeed, you will not consider a text editor or source safe tool as a XP-product.

### 7.3. SADT

SADT - Structured Analysis and Design Technique.

It is known as a creation of the company SofTech or as a functional variant in a government version (IDEF0). It has been employing since 1973 in different branches of activity such as business, industry, defense, communication and design planning.

Diagrams within the IDEF0 standard take undoubted benefits for functional system modeling. However, the approach to provide system modules as boxes with input, output and control data reveals only high-level structure peculiarities of the system. In this point SADT successfully competes with activity diagrams in UML.

I didn't find in SADT mechanisms for further system development: from gathering requirements to realization, testing and deployment. The ideologists were likely not to set this as an object. The methodology loses a lot in comparison with others according to coverage of software lifecycle stages. It focuses only on gathering requirements and business modeling.

It brings to life the idea of big prior design at the start of the project in order to minimize bugs creeping in the later stages of lifecycle, when the correction is more expensive (see the book [24]).

Owing to hard decomposition of the process this methodology can be considered as a RUP «ancestor».

## 7.4. MSF & MOF

MSF & MOF - Microsoft Solutions Framework, Microsoft Operations Framework.

The creators of this methodology intended to make the practice combining the best principles of cascade and spiral development model, in other words "best from two worlds" (see the site [23]). Naturally for best results you are proposed to employ solutions and products directly from Microsoft.

However, you can't help admitting a significant contribution of Microsoft in the IT domain. It is worth to examine MSF only because "the leader does it".

For example, I was mainly interested in research work devoted to combination of roles in small-scale projects. This is especially acceptable for CIS, because most of IT-projects declared here as big and complicated, as compared with western companies, are small and medium.

This practice gives an excellent description of software testing and maintenance. Who didn't use bad language for RC-versions, full of errors, and a great number of different patches, service-packs, hot and bug-fixes pleasantly offered on the company website?

It is intelligible documentation concerning practice of risk assessment during the software development. And of course, excellent SPS in the sphere of product selling.

## 7.5. Iconix

It lies in the middle between heavy RUP and rather compact XP. (See the book [1]). I believe that the most important distinction from the other methodologies is the introduction of diagrams for validity analysis. The book is perfect from TOP10 errors for every software development step to continuous reminding, something like this: «Don't work hard on this; you don't have full information, in order to spend a lot of time».

## 8. General Solutions Area

## 8.1. Lifecycle Stage –Management

### **8.1.1. Team Selection**

According to my point of view this theme is widely illustrated in the book [10]. It is very important that the process of recruiting specialists in the sphere of IT-development is considered here more precisely, but many of these ideas can be applied in other fields of activity.

Also the works on the site [14] seemed to me rather informative, though not without some kind of self-advertisement.

The core of a successful solution is to choose the team members who are really worthy. These people are ready to take responsibilities for making decisions, play diverse roles in the development process, and have a wide experience in close fields of software development. Joel Spolsky calls these people as "superstars".

Controversial statement. Someone is not born a superstar, he becomes it. Sometime ago all of us used to be beginning programmers, that is why I think that gifted beginners also have the right to take part in a team of large-scale project.

If we talk about necessity of training beginners and not only them, as well as the problem of team integrity within group of people with different level of proficiency, I'm inclined to think that here the method of "small victorious war" takes place. This method lies in the fact that basic project solutions, architecture, development tools are firstly run and customized for the pilot-project that does not have a critical value for business. And after its successful completion the team starts working with the main project.

### **8.1.2. Division of Responsibilities**

Frequently in large companies the control over decision-making becomes more toughened, the work is divided into narrow sections with sharply defined authorities and responsibilities. As a rule, in beginning companies the same person fulfils several duties that do not correspond as far to his real professional skills.

At this point I'd like to cite Alexander II: "It's easy to rule Russia... but it makes no sense." Large-scale systems require large-scale management. However, I consider that inner self-organization is much effective than strict external control. In the book [6] there is an example with the company IBM, where the wider delegation of authority from the center to structural departments allowed extending controllability of the whole company.

I think that without trusting in people taking part in the project it is useless to begin this project at all. Therefore, it's better to delegate people authority as much as they can take. Finally if someone does not manage his task he can always be fired.

When everyone is charged for his own narrow section of work, I consider, it may cause serious troubles with product quality and senseless struggle between functional departments for transferring responsibilities.

### **8.1.3. Atmosphere within the Project**

Healthy human relationships between team members are very important for successful completion of a current project and for the further work in the future. In the book [4] there is the following idea: if you want to know whether a successful methodology employed in the project, you should ask developers a question if they want to take part in a similar project one more time.

The same matter is with the atmosphere within the project. If the working day is full of settling endless problems, if employees are strictly divided into "castes" and do not want to share the information with opposites, if there are no favourable conditions of work such as separate phone, sound insulation, convenient work place, worthy payment, etc. you should hardly expect that an employee would be ready to work in such a way one more time. (See the book [5]).

Unfortunately in most cases (great fluctuation of personnel, deficient financial resources, etc.) an employer does not pay attention to a stress situation in a project considering it as a norm, and tries to maximize profits by means of cutting expenses connected with establishing a favourable atmosphere.

I believe this is a mistake. One should use every possibility to increase labour productivity; from a free pizza for overtime work to personal subscription in a fitness club for going in for sport.

#### **8.1.4. Career Growth**

A human being is not a robot, mechanism that is able to turn the same screw on the same machine all his life long, being interrupted for thorough repairs (leaves) and periodical lubrication with engine oil (salary in the end of a week). Most of us are willing to achieve something more than we possess at a certain moment.

Therefore I consider that we should put real objectives and every minute work over the achievement. Unfortunately, the lack of free time results in the fact that the main place (and often the single place) for raising the level of our skill and development is our work place.

The employer must (in spite of seeming unprofitability of investment for fast Internet, new books, courses and even business conference trips such as PDC – see the article [2]) provide professional and career growth of his employees. At the end all are the winners: an employee becomes satisfied with his work while organization gets more qualified staff.

Of course, the risk to lose a person that is more qualified than he is expected increases. See the book [10] for understanding how to hold talented employees.

I agree the thought expressed in the article [2]: «DO NOT WORK WITH A CHIEF WHO PREVENTS YOU ACTIVELY FROM CONTINUAL TRAINING. IN NO CIRCUMSTANCES».

### **8.1.5. Labour Productivity**

A lot of factors influence the effectiveness of work. Physical factors are the following: size of personal space, illumination, sound insulation, size and quality of a monitor. There are also virtual factors: atmosphere within a project, collective relationships, language environment and cultural differences. However that may be, I think that it is needed to reveal all the sources that cause discomfort and interfere with your work, and to obviate all the obstacles.

It is much profitable to settle the problem from the very beginning, when it has not led to an evident conflict yet. People are too valuable resources for being lost in the process of awkward actions trying to solve problems.

### **8.1.6. Communication**

Many things in the process of development depend on speed and ways of information transfer. Of course, it would be better if there is a single room (deathmatch room) available, where all the developers sit side by side together with a customer's representative and nobody else. However in most cases it appears to be impossible. (Indeed, whether all the developers gather in a company assembly hall).

That is why it is necessary to find the most effective way of communication available.

For large bureaucratic companies (often government enterprises) it would be better to use paper documents, which can be transferred from one person responsible to other. For small-scale Independent Software Vendors the collaborative work of developers at a short distance from each other was invented. Companies that have geographically dispersed teams use videoconferences and diverse ways of communication via the Web.

Besides the effective communication, it is needed to apply also its absence, i.e. to be resistant to so called "informational draughts" (see the book [4]). It is worst of all during the development process to hear or see extraneous information not related to the current project. (For example, to sit in a room with employees from departments of support, sales or advertisement, etc.)

### **8.1.7. Planning**

The first rule of planning – use it whatever senseless it may seem. Most developers have a pessimistic treatment to any kind of plans (and even slight people who are planning). Programming is not a constructing process where it is known precisely that a builder constructs a 5m<sup>2</sup> brick wall in an hour, while a solution X hardens in 10 minutes.

There are too many factors that indirectly influence the general result. For instance, I have the following formula: the task takes 2-3 times more than you are planning to spend on it in the case if everything goes right.

In other words, you will find all the components needed, quickly devise an algorithm, come across no bugs in user modules and at last, the foremost – you will be capable to focus on the task without giving up.

The goal of planning is to be aware how much the project is behindhand and try to redistribute the task in time, in order to work according to schedule though with less functionality. (See the book [16]).

Large, thoroughly developed plans are insufficient for success, as a rule – the inaccuracy in every step of a schedule is occurred to be significant. Therefore, I consider the practice of “small victorious wars” (interactive development) efficient when prior objectives are set and the tasks are performed within 1-4 weeks. The team schedules and tries to solve the challenging problem in stated short amounts of time.

### **8.1.8. Process Organization**

Of course, for a software development project to become standard, it is appropriate to invent a methodology, a set of rules stated basic moments of the program product lifecycle. The fact that the organization employs this method or tries to create it indicates a high level of IT-company maturity.

But in most cases the methodology is appeared to be no more than “an advertisement trick” of a company, spreading good rational ideas in written, but actually far from reality. This way the company attempts to earn competitive priority in a struggle for profitable orders. When a company occurs to be helpless in solving the organization matters, it becomes too late.

It is even worse, when too rough management over the process is harmful for effective development project that is beyond the methodology framework. As a rule, every project is unique, that is why I think it rational to employ the set of SPS and use them in this or that situation in a flexible way.

As it is derived from the book [4], «the main thing is to cut the enemy’s hand off, but not to use a certain martial art».

As far as I can judge, the most important feature of a successful IT-company that practices commercial software development is its flexibility in matters concerning customization of projects, data domains, etc. for different types of customers, but not certificates of N level conformance in CMM (See the article [8]).

### **8.1.9. Developers Functions**

A good army needs diverse types of forces: infantry, aviation, reconnaissance party, navy and much more. Besides availability of different arms, the main objective is to turn them to effective use and advantageous interaction.

I will not dispute. But I think that there is an important difference between conducting military operations (and their successful completing) and developing software projects. People at war (according to army headquarters' opinion) are interchangeable. I.e. for conquering the district X it is possible to send for a "cannon fodder" the division Y, and than to carry out mobilization and to form the division Y again.

The IT-projects work otherwise. Every developer is provided with unique information. Having lost one of them it would be difficult to hire the other one without quality losses.

There is a question, «How many or few people would have to be hit by a truck (or quit) before the project is incapacitated?» The worst answer is "one". The matter concerns not only the uniqueness of every person, but the team stability in case of losing one of the members.

Base on the above, I'm coming to the following conclusions. First, it is important to remember about value of every member. Second, it is the foremost to enable their interchangeability. Hitting the second target I see presently in promoting many-sided professional growth and members rotation among different functional departments.

### **8.1.10. Personnel Training**

A new person having entered the project is likely to be a star of a company. Therefore, there is no need for sparing the time for his integration in a group, for proving him all essential materials; key programmers should find the time for cooperation and training.

In the very beginning a newcomer needs favourable environment and sense that his work is in close connection with the work of others. It can be achieved by means of his participation in current projects discussions (even if they are not related to the main work that is performed).

The foremost is to obtain the impression of integrated team and its strengthening by means of using knowledge, experience and «fresh head» effect.

A matter concerned new comers in a project is connected with the question, «When it is needed and profitable to take new people in a project?». The book [6] says, «Adding manpower to a late software project makes it later», because a lot of time will be spend for training, explaining the core of a project and information around the process.

Sometimes the administration proposes the single way out of saving the project only by adding new human resources. In these cases, I believe, it is necessary to analyze the things key developers spend time on, and put only paramount objectives (project completion), while minor questions transfer for performing to other staff, including new comers.

### 8.1.11. Task Orientation

The book [4] gives us an example of testers who were greatly concerned with absence of obvious progress in the programmers' work. In order to prevent the programmers from giving up work the authorities decided to post up the list of project tasks and notes of their performance in a corridor available for anyone. The questions disappeared at once.

This example illustrates how it is important for team morale to have in the face the information on the current progress of every project element. This technique makes a wholesome impact on the developers owing to the impression that "one more day is over and X points has been crossed out". Everyone is willing to be distinguished and do more than others.

It doesn't matter whatever tools were chosen for achieving: MS Project plan, task list in MS SharePoint or simply MS Word file with numbered list and check boxes.

There is one more psychological moment to be noticed. Nothing kills the team activity and optimism so quickly, like absence of clear and real goals by the target date does.

That is why it's easy to imagine a developer who knows that he has to make the functions X, Y and Z by the weekend and who refuses to take lunch in order to be in time. And we can scarcely imagine one who works hard without clear idea about team progress and his part in this project.

### **8.1.12. General Project Environment**

In the book [6] I found the idea about “Project paper writing-book with further transfer to microfiches”. It looks funny now, but it is rationally to have a coordinated set of documents described both conceptual bases of a project and process of its implementation.

The task is not only to work out this set of documents, but also to provide all the participants the access to this information.

It was jokingly to observe in that book [6] how the author changed his point of view concerning hiding the information under the influence of David Parnas ideas.

Perhaps, I will ever change my point of view too, but presently I think that all the information on the project should be available for all developing members.

The reasons are the following:

- Brilliant design solutions in different parts of a system are the subject of wide discussion for different people and as a result their quality increases;
- Making «guru» to state his ideas on paper increases the speed of training low-qualified employees;
- Strict division into the information flows makes the groups (analytics, programmers, testers, etc.) to be more isolated from each other;
- And finally it is better to have abundance of information (with a chance to choose the most useful) than suffer the lack.

### **8.1.13. Intensity of Work**

Here is a ridiculous but effective advise, "Measure your energy, and relax more". There is a humorous case described in the book [4]. During the author's work in the Bank of Norway he discovered that the workday lasted only till 3<sup>30</sup> p.m. (everyone went home and there was no a person to communicate). At 5<sup>00</sup> p.m. all the lighting, besides emergency one, was turned off, and at 7<sup>00</sup> p.m. everything was died down. This way the authorities take care of employees not to overwork on their places.

However, the sense is that the continual overworking is harmful. It is better to be 100% productive every day, than to work up to 10<sup>00</sup> p.m. at Monday and during the week to show the activity of dead fish in the sun.

Many problems are settled easier (at any case do not seem so terrible) if before we had a good rest.

Here are my personal impressions. I am aware that I possess some qualification. I am sure that tomorrow (after reading some books, articles, Internet forums, etc.) the level of my skill will rise. Therefore, sometimes instead of insistence in willing "to catch the last bug" it would be better to postpone the problem. Next day this problem will be solved much easily.

This is regarded not as a refusal of persistence, but the firm belief that the things are easier than they may seem at first sight. If the problem is settled for a long time, hard and painful, you are likely to choose non-optimal solution.

### **8.1.14. Priority System**

Establishing a priority of the challenging problems is necessary for doing presently those things that are estimated by a customer in a maximum way.

A very unpleasant feeling may appear if after hard working over a function it is hardly used or even needless. That is why the priority of requirements and their change by a customer within the system (and business) development are very important.

A customer has the right to change significance of separate system elements. The business of developers is not to conflict with him motivating their views by the fact that «the function X of the module Y is half-developed yet and you will actually need it».

It would be better to respond to the situation changed. At this point the XP methodology is attractive. It assumes a customer in a single room sitting together with developers and responds to the customer's changeable requirements.

However, I always maintain scepticism about this style of collaboration with a customer. In most cases a customer is hardly aware of software engineering potential. Therefore according to my opinion, the top feature of development is not telephone collaboration like «well, your new idea will be tried out at first, with others to be postponed». The system possibilities more sufficient for business should be revealed as early as possible.

### **8.1.15. Documentation**

The problem at issue lies not in information accessibility in a project (this matter concerning SPS has been already discussed), but in development of a clear and qualitative documentation in general (see the article [3]).

There is a diffused opinion that “a real programmer” never uses (develops, reads, etc.) any documentation, more over diagrams (see books about XP), while the code comments smell bad (the idea was expressed otherwise, but implied in the book [14]).

I consider it is not true. No domestic equipment is sold without an operating manual, while spaceship drafts are much more expensive that its physical embodiment.

Perhaps, I am lazy or have short memory, but I really like to record the completed agreement of architecture and algorithm peculiarities. It allows to obtain further insight into the concrete details of realization without deep analysis (laziness), and rapidly recollect decision making after a hard joyful weekend (memory).

A qualitative documentation available is a guarantee of firm project (there were no places understandable for a single developer). It disciplines the development stages (in order to express the thoughts in written they should be customized in a clear form). It facilitates the process of maintenance (members of support service are not usually the first developers and without any documentation they will suffer troubles).

## 8.2. Lifecycle Stage – Analysis

### **8.2.1. Presentation of Information**

In the process of collaboration between representatives of «diverse levels of comprehension» within a system (for example, between customer and analyst, analyst and programmer, etc.), one of the main problems lies in difficulty of «speaking the same language».

Everyone thinks within the category understood by his group. A customer thinks in the terms of business process, programmer has the ideas of a class, forms and SQL-queries, testers are interested in sharply defined requirements with lists of limit values for testing process.

I conceive it very important to find the ways of information presentation clear for all parties as soon as possible. No need to keep yourself (and the opposite party) within the limits of certain document types. A colorful presentation fits a «big» conference; diagrams of business process are used for initial learning of a subject; text presentation of use cases «according to Cockburn» is intended for making requirements more detailed.

### **8.2.2. Strategy of Progress**

I believe that the most effective strategy of subject development is beginning with motion «in breadth» (after confirming that we follow the right way or after correcting the course) and then, with motion «in depth».

This method of movement protects developers against the well-known problem – «analysis paralysis» connected with impossibility to solve a local problem that is obstacle in the way of progress.

After reading the book [1] I have come to conclusion that almost every chapter gives an advice like «Do not spend much time on the development stage X, move to the stage X+1». The main argument in favour of a hurry is that on any stage there is no enough information available in order to spend big amounts of time on long-term decision-making.

It is necessary «to catch» the basic scenario of collaboration with a system without being lost in the labyrinth of involved business logic. In any way there is much more logic than it would be described. Many things will be clear only on the stage of testing or after system launching.

Any IS (information system) is created for a long amounts of time, therefore many of the customer requirements become out of date even in the process of development. Hence, it would be much better to realize an «incomplete temporary solution» as soon as possible, than give a customer «the completed solution» at the end. Fowler said the same thing concerning tests, «It is better to implement incomplete tests than not to implement complete tests».

### **8.2.3. Two Points of View**

As it was mentioned before, «Two heads are better than one». During the analysis of a subject it is effective to have two analysts working in pair, than each taken separately. As a rule this is connected with the fact that cooperation with a customer is usually effected in a form of dialog, therefore it is better to have a person who asks questions and holds a discussion, and the second one who writes down the thoughts.

Also the work in pairs contributes discussion of key moments for coming to the point, but not mechanical recording of ideas expressed by a customer.

#### **8.2.4. Glossary**

Why the construction of the Babel tower has not been completed? The difficulties appeared when the participants suddenly became speaking different languages. The problems concerning absence of a common terminology and language understood by anyone: customers, architects, workers resulted in the project failure.

The situation with software development is similar. A lot of misunderstandings that might be avoided occur because the same concepts are named otherwise. A customer uses business terms, system architects use class categories and patterns in the way of thinking, database developers – DB tables and queries.

I consider it very important:

- First, to work out a general system of terms and signs agreed by all the development members;
- Second, to transfer the concepts from subject area to system architecture.

It may seem an excellent test to show a customer the function text representing an element of the business process. If the text is full of address arithmetic, invocations of methods with names like `GetDBById( int i )`, the code is likely to be incorrect. The same function can be better called `GetDiscount( int DiscountCardId )`.

The names chosen correctly reduce comments and increase code comprehension (that directly results in easiness of the further modification and support.

### 8.2.5. Diagrams

Many people criticize the usage of diagrams because of difficulties with their producing and actualization. However, I consider any method of effective description of the system elements and way of their collaboration to be appropriate.

Presently, the following types of diagram are known:

- Class diagrams – class hierarchy;
- Object diagrams – interactions between objects in run-time, connections and relationships among them;
- Use-case diagrams – interactions among actors and use cases;
- Sequence diagrams - interactions among classes in terms of an exchange of messages;
- Collaboration diagrams – similar to Sequence type, with emphasis on interactions between objects;
- Statechart diagrams – states and ways of transition between them;
- Activity diagrams – business processes, functional system decomposition;
- Component diagrams – interactions among physical components of a system;
- Deployment diagrams – physical arrangement of modules.

The book [1] describes Sanity Check diagram for analysis of use case description logic.

I believe that it is useful to produce your own types of diagrams (or mix the existing ones) for more precise description of a system. In any case «the ideological purity» of the system elements does not assume such importance as deriving benefits from it.

### **8.2.6. CASE-Tools**

A good tool for drawing diagrams, collecting requirements, producing modules of data domains lighten much the work of system analyst. According to an existent opinion, success of the RUP methodology (Rational Unified Process) to a great extent depends on availability of qualitative tools of the Rational Company, which support all the lifecycle stages of a product.

Advantages and disadvantages of CASE-tools raise plenty of hot discussions. I think that the top question in discussions of this or that tool is its flexibility and «ideologizing». In the article [4] Rational Rose is subjected to sharp criticism for too free interpretation of UML.

Formerly I expressed my very negative treatment to this article (because of the Rose criticism). I did not sympathize the idea of Rose turning into «a graphic editor for drawing diagrams».

Presently, I have changed my point of view. I think that the wider functions are and easier the product is in comprehension, so much the better. Low requirements to knowledge of this or that model language needed for product use only minimize the amounts of time for training beginners and therefore increase the product popularity.

### **8.2.7. Use Cases**

Use Cases (or use story in the XP methodology) is a brief and clear description of a certain function of the business process valuable for a customer.

«Valuable» means that a customer is ready to pay extra for implementation of every use case. In other words, development of new class, producing a mechanism for access to data bases – this is not a use case. It makes no difference to a customer how his business requirements will be realized.

Authorization while the system starting, creation of new order, payment of purchase, receipt of accounts – these are «normal use cases». Every of them can be implemented separately and «sold» to a customer for charge. In the process of receiving use cases a customer derives more benefits from the IS developed.

A use case is like an embodiment of «divide and rule» principle. After dividing the system into smaller parts and certain task it becomes much easier to describe them, implement and evaluate the progress. The risk of appearing such situation as «90% of the module is completed...90% is left...» is mitigated.

I would like to point to the remarkable description of theory and practice of producing, classification and using user cases illustrated in the book [3]. I want to repeat one more time the main advantages of text use cases:

- a) it's easier to work out a text;
- b) it's more understandable for a customer;
- c) it's easier for training.

### **8.2.8. Re-engineering of business processes**

This term became very famous in recent years. It means, firstly, re-comprehension, redesign, improvement of business effectiveness and processes supporting. Frequently, necessity in re-engineering process rises from creation of a new IS that make the organization activity automatized.

Unfortunately, in most cases, automation especially in large organizations gives a start for “great war for capturing spheres of influence” between structural units and their managers. Just one thing wrong that in a situation when management fails, firstly, the team of developers suffers.

In most cases when there is an objective not only to create a new system from the very beginning, but to make changes in existent business processes, it would be useful to produce system models AS IS and TO BE in order to reveal differences (and to coordinate them timely).

It is always useful before “chaos automatizing” (See the article [5]) to organize the processes firstly. It prevents you from producing a great amount of needless code that significantly helps to save the time and forces of the project team.

### 8.3. Lifecycle Stage –Design

### **8.3.1. Object design**

There are the following characteristics of classes and objects qualities described in the book [8] as: coupling, cohesion, sufficiency, completeness and primitivity. Every created abstraction must fulfil a certain work deploying limited data source. If an object is lacking of something for a result, it “requests” other classes for help in solving this problem.

It is a great art to create «right» objects of data domain interacting with each other in effective way. In the project of salary accounting I was prompted a nice solution concerning method of accounting rise in salary. Every rise «knows» how to account its sum or can «request» its components to pass their sums, etc.

Hence, every object solves only the problem for that it possesses all the resources available. So in the task that was implemented in a structural style (cycles, recursion) with the help of OOP, a surprisingly pretty solution found its application.

The same person (see the last point in the Chapter «Gratitude») expressed his rather self-confident point of view, «Code of every method should be no longer than tree lines or one line in the form of process delegation to the method of other object». The meaning does not lie in the number of lines, but in code simplicity – the system good developed does not have much code, there are a lot of diverse interactions among objects.

### **8.3.2. Design Patterns**

I found excellent descriptions of design patterns in the books [7, 17 20, 9 and 19]. There is the following idea in the book [14], that presently it is impossible to pretend that you understand something in objects, if you are not capable to speak about strategies, individuals and responsibility chain.

Design patterns do not teach something fundamentally new, they rather standardize the existing SPS in the sphere of IS framework designing and «the right code» writing.

A skilled developer who perfectly understands the OOP conceptions, who is able to think in an abstract way and build beautiful and flexible architecture, employs the design patterns anyway, even sometimes without realizing it for himself. They are of no use for a beginner – the abstract factory or Memento is hardly employed in trivial projects.

Mostly likely, design patterns are used by a medium developer who raises his qualification, who is much experienced in and who is aware that it's time to raise the level of his skills.

There are two positive moments with design patterns. On the one hand, they introduce a common vocabulary of communication, it's enough to say, «Implement this object as Singleton» and everyone understands the matter of fact. On the other hand, the process of system modification is lightened if certain design patterns are known to be applied to the objects. (Otherwise design patterns and other solutions are «dissolved in the code» and maintenance becomes more difficult).

### **8.3.3. Component Development**

The concept of system development with the help of isolated modules implementing the required functions of a certain interface has gained popularity long time ago. The components extended much a circle of software developers and reduced the requirements to their qualification.

There is an opinion that nowadays success of a programmer is defined significantly by his ability to use components. Unfortunately, the components have a disadvantage resulting from their advantages. They do not make you to think in the terms of objects and data domain model. There is a joke that when the problem has occurred, the Delphi programmer becomes to attack Web-conferences with questions about «magic component» getting over the difficulties, but in the case if there is no one found, he declares that the problem is unsolvable. This joke turns into the truth more and more.

The matter is not in criticism of Delphi and RAD systems of visual programming. (I use the .NET environment that belongs to this sphere). It is a widespread stereotype: a beginning programmer, who has learnt «to throw» the components on a form, names himself a master of OOP (whatever you may say, the buttons are as class instances).

A great achievement of components is a possibility of parallel working over separate parts of a system. Purchased components are able to accelerate the development process greatly. Someone scarcely likes every time to add to Grid function of sorting, calculating sum meanings and other «standard» functions.

### 8.3.4. Conceptual Integrity

According to my opinion, the key factors of success of an informational system creation, as well as maintenance and development are the following: system integrity, mechanisms of program behavior within changes of requirements considered in advance and realized. The worst of it is when you discover the new customer's requirement completely destroys the system «ideology» and you are not ready for these kinds of changes.

These are not arguments in defense of "large prior designing and requirements fixation" of IS with the help of requirements specifications. On the contrary, I come to a conclusion that it is more rationally to apply short and quick iterations through use cases with initial *IS-framework* to be built *correctly*. However the architecture should be *flexible* and steady for modifications. The exact answer to my question, *how* to obtain this solution, currently I do not know.

Nevertheless I am sure that operation of the object conceptions taken directly from the data domain makes a project to stand a better chance of success.

As a rule, a system architect is responsible for producing the main principles of IS functioning and for «ideological purity» of the solutions applied. The main difference between project architect and manager (they are often confused) is that an architect is technical while manager is administrative heads of the development process. Lose of anyone from both may result in disastrous effects for the IS developed.

### **8.3.5. Distribution of Errors**

The book [14] illustrates that it is needed to obtain «a shift» of errors from the run-time to application compile-time. This is the idea that the program should be developed in a way when the errors occur in the period they can be relatively easy corrected, but not in the moment of program running.

Using address arithmetic, operations like `sizeof( ObjectType )`, where assumptions of object size are made, are occurred to be impressive examples how flexibility of a language (for example, C++) increases the possibility of appearing hardly observable errors, «visible» only in run-time.

The article illustrating faster access of `DataReader` to the table of data bases via `Index`, not via `Name`, as I think, contributed its negative share in the process of software development.

So the theoretical thesis of destroying encapsulation (internal object organization) encourages occurring rather practical errors.

Implicit transformation of types (for example, from `int` to `bool`), suppositions about meanings of some constants (`true=1`, `false=0...`, or `-1?`), «evident» managing of memory allocation and release in contrast to automatic garbage collection – all these are practices of wrong errors distribution.

On the contrary, strict typing of models, interface maintenance (a class must realize strictly defined subset of methods), inheritance hierarchy simplifying (any dynamic cast from super to derived class may be potentially dangerous) will help to distribute the errors correctly.

### 8.3.6. “Wrong” Solutions

One my acquaintance became indignant after having bought his child a meccano that occurred to be «wrong» – there were no needed pieces, in some places holes were not drilled and other defects. I supposed that this way the producer put into a child’s mind the thought about our world imperfection. In order to make it better, it should be processed with a file.

I would like to cite that acquaintance, «If the system is «wrong» in some places, it means that actually it is working, but was not created in the academic interests».

Thus, you should not be frightened that your data base is not in the 5 Backus normal form

Calculated fields are saved in a table, and in sale saved good «Name», but not their «Id’s».

Frequently, a lot of interactions among objects, «good-looking» on the class diagram (because of it, the first object can receive the data of the second, and though it the data of the third one, etc.) lead to great troubles with productivity. The lazy collections (load data on request), pool of object delivered, caching, etc. «rescue» in this case. But these are only variants of problem *solving*, but not *preventing*.

Anyway you should calmly take the thought that sometimes it is necessary to do things that contradict fundamental principles of IS development. In the end the important thing is customer’s satisfaction, but not keeping the basic principles in details.

### **8.3.7. Invention of Wheel**

It is bitter to understand that programming turns more and more into a skill but not into an art. In order to create software products it is enough to realize the existing conceptions, without inventing something new.

Therefore it is important to deploy resources in a project efficiently, not to make the system complex, working with things that have already been created by someone else. Do not lose an opportunity to turn a «big» project into a «small» one with application architecture to be simplified and number of realized functions to be restricted (See the book [19]).

Of course, experience of creating a sub-system that duplicates the existing operational system functionality, DBMS or graphic editor during the development period of time is much tempting. But there is a question – is a customer ready to pay for it?

Creative activity should be applied in weakly investigated fields that are reduced progressively. In most cases you should employ standard solutions that presently exist and were tested long time ago.

I consider it much effective to reapply ideas, experience, code, components, applications, etc. Frequently, creating components (but not purchasing), producing business-transaction mechanism (against system ones in DBMS) and s. o. – all these result in project to become more complex and terms of product realization to be extended.

### **8.3.8. Algorithm**

Initially a «quick» algorithm used in applications significantly reduces both demand in productivity increase in the future, and cost of this optimization. Many of us felt a «magic» effect of training example from the Borland Delphi/Builder environment, where advantages of one from three methods of sorting were shown graphically.

It is said in the book [19] that with raising the level of operations concurrency in corporate applications, developers take less care of attendant problems. There is similar situation with algorithms. With appearing rich function databases, most of programmers forgot that bubble sorting differed from insertion sorting.

Perhaps, it's all for the best. You can save the time on reading such books as [21, 22 and 23]. Using the functions from libraries mitigates risk of appearing errors, while the code becomes «purer» and clearer.

Sometimes you cannot do without algorithm to be created. In that project dealing with salary accounting the initial algorithm of salary rise that do not depend on existing ones was changed.

The task suggested about recursion and endless cycles. It was proposed to define rises dependent on existing ones (it's quite easy), and then to obtain the required result through difference with a general set of rises. It was done and resulted in a great increase of productivity.

### 8.3.9. Decomposition of System

The principle Model-View-Controller is known for a long time, and was discussed in the book [19] and article [6]. The core of this method lies in decomposition of a system into [tree] three logical levels.

View – presentation level. It includes forms of user interface and their analogs. This is the thing that the user «sees» and interacts with, trying to get a useful feedback from a system.

Model – work with data, DBMS, queries, stored procedures and other «insides» of the data collaboration. User must not see “directly” the internal repository of information (this principle is violated so frequently that I would not agitate for it).

Controller – business logic, classes of the data domain model, application servers, services, own mechanisms of data processing (pool of used objects, events notification, lazy loading, etc.) A controller serves like «a glue» between model and presentation, proving the information among them.

According to my opinion, a sanity check diagram was developed on the base of the idea covering MVC. It is a useful supplement for UML, but unfortunately not being included in the official specification.

Decomposition of a system and relative isolation from levels lighten independent development of layers and system maintenance. Tools of automatic data synchronization between layers (.NET named it binding) occurred to be no less effective.

### **8.3.10. OOP**

Advantages and disadvantages of OOP are the following:

- + : reusing code (inheritance, components and libraries)
- + : information encapsulation
- + : natural modeling of domain data objects
- + : resistance to requirement changes
- + : code understandability (it is read as if it is written in native language)
- : reduction of methods implementation velocity
- : rise of executable code size
- : rise of system complexity
- : significant increase of requirements to developers qualification

OOP has proved to be as an advanced methodology of software development, first of all, owing to effective division in concepts of subject domain and system presentation. Perhaps, the future belongs to the programming technologies that make this division more obvious (Functional Programming, Aspect Oriented Programming, Logical Programming).

The author of the article [7] says with a pure American pragmatism, «You should prove your case by your actions, especially when others do not want to listen to you». Many of us heard about perspectives of logical languages in the development of 5-generation PC in Japan. And where can we see the realization of these PCs? Paper tiger is far from the real one. That is why presently OOP is the most viable technology of software development.

## 8.4. Lifecycle Stage – Coding

### 8.4.1. Coding Standard

Coding Standard is a document that describes agreements on structure and organization of the program code.

The more important elements are the following:

- Names of program objects (components, modules, classes, variables, constants, methods);
- Prefixes used (cb – **Combo**Box, tb – **Text**Box, btn – **Button**, Hungarian notation)
- Symbol register **camel**Style, **Pascal**Style, **c\_plus\_plus**\_style, ANOTHER\_CPP\_STYLE
- Design and structure of a program (indents, tabulation, declarations)
- Remarks format (template of classes specification, methods, algorithm explanation)
- Exception processing
- Work with memory
- Design of structures like if/then/else, switch, for/foreach/do/while and so on.

It is a difficult task to introduce the Coding Standard (Every programmer gets accustomed to write the code in his own style). But after concluding this agreement it would be hard to imagine the work without it.

The main advantages of the Standard:

- Inculcating «good» code writing style (it is easy to produce a code recognized by a compiler, much harder – a code recognized by a human);
- Cutting costs on software maintenance (the code is written in a single style);
- Disciplining development process (and its participants – if there was said to describe every method characteristic, it must be done in 100% cases but not from time to time).

### 8.4.2. Collective Code Ownership

In my opinion, this SPS (as well as a related idea of versioning support of source code, is one of the «pillars» of the XP methodology. The main idea of this principle is that any programmer in a team can change any part of a system (though in XP says about any *pair* programmers).

At first glance, this SPS follows the anarchy way and leads to the breakdown of a software project. But this is not actually true.

On the one hand, collective code ownership is hardly imagined without proper tools. Presently, there are a lot of both commercial (VSS, ClearCase), and free of charge (StarTeam, CVS) products for maintaining different versions and collective work.

On the other hand, collective code ownership raises the responsibility of each team member. If it is necessary to change something for the final result (and a developer is experienced in the required field), it can be done without continual consultations.

Versioning tools maintain such operations with files as Check-Out (taking version from storage for redesigning), Check-In (putting changed local version in storage for overall access) and the foremost Merge (merging of *local* version and *common* version in Check-In). If the system is not capable to merge automatically, it proposes a developer implementing Check-In to make it.

Versioning systems are usually integrated in the development environment (the main function is «Get latest version»). It should be noticed that systems work mainly with *text* files.

### **8.4.3. Pilot-project**

Development of user interface prototype (pilot-project) *before* coding saves big amounts of developer's and final users' time.

There are automatic tools for designing such models (for example, MS Visio), but I think that there are no better tools than pencil, eraser, and sheet of paper invented.

When a user sees the future shapes of application and its interaction (in a form of arrows) drawn in a haste, he is not afraid to change something in the document. If you present to a user screen shots designed perfectly well (that look like real ones) or a half functioning applications (with stubs instead of method invocation), a wrong impression that the system is almost completed can appear.

The main goal in creation of a counterpart is a prompt reply to the following question: whether developers' comprehension responds to the business requirements tasks; and the search of ways for quick receipt of effective feedback from the system.

#### **8.4.4. Smart Instrument**

Recently I was told about a hammer that made a constructor's job much easier. It is hollow inside, and the empty space consists a half of solid particles. In the moment of blow, particles "arrive" and reduce the rebound energy in a few instants later. Thus, the tool serviceability increases.

The similar thing is with programming. It is necessary to find and promote the most convenient and fast development environment, compiler, functions libraries, and sets of components.

A physiological comfort is no less important while working with the environment - colour spectrum of code editor, location of toolbars, and hot bottoms of access to the most applied functions.

We should admit that many IDE (Integrated Development Environments) continuously extend these nonfunctional (but very important) features. Now it is impossible to imagine the environment without the IntelliSense technology, bookmarks, highlighter of different program elements, creation of code templates and developed text editor.

### **8.4.5. Data structure**

Good knowledge of SQL and ability to understand the query code, that takes A4 sheet, undoubtedly are to a developers' credit, but what for all these difficulties? If you have to select data using a great number of Joins, it means that no optimal way of information storing has been chosen. Hence, sometimes you should change the data structure (normalize or denormalize the database).

In general, it is related not only to SQL. During developing of a report in CrystalReports, I revealed that a check stub with the same information should be added to the request blank at the bottom. I began to think about subreport, its linking with the main report, and so on. In other words it was a difficult task.

But I was suggested actually a genial solution (it may be explained only owing to great experience of work and luxuriant imagination). The solution was that it was necessary «to turn over» the report on a side and implement two parts of it in the Landscape format; later two parts of the same Details should be simply duplicated.

#### **8.4.6. Test Projects**

While software developing, you should rely on viable, tested solutions that were tried out in practice. Within the lack of time that is a characteristic of almost all projects, value of a mistake in choosing the wrong platform, development environment, DBMS, technology, programming language, etc. rises greatly.

Therefore, I consider that before using a new technology (no matter whatever revolutionary it may seem) it is important to find some resources for at least superficial study of a subject. The test projects are most relevant for this.

The goal of these projects is in short time to get information on advantages of a technology (velocity, easiness of applying and developing, productivity, steadiness, ranging ability), as well as disadvantages and methods of managing the restrictions.

After investigating and receiving a positive feedback, participants of an experiment become as coaches for other developers. This experimental works can be conducted in the periods between project processes and can be a special form of encouragement for good work. Everyone wishes to familiarize himself with new ideas of IT world, besides the routine.

### **8.4.7. Pair Programming**

Pair programming (PP) is a practice of parallel work of two developers over the one part of the system. (See the article [14]). At first glance, it is wasting of resources in vain (especially the time). Moreover, many representatives of professions are not able to work together.

I do not consider that PP «in pure form» is a good idea. There are always «routine» operations during a system development process: writing SQL code, developing user interface, methodical testing, debugging. It is better to concentrate on these kind of work and execute it independently.

On the contrary, decisions concerning system architecture (hierarchy and interactions among classes, inheritance, DB design, model modification according to requirements change) should be made together. It would be better to discuss them with more than two people (up to five).

Here is a list of PP advantages:

- Defects is found at earlier stages, when it is easier and cheaper to correct;
- Number of general errors is reduced (in average by 15%, by the way, the time of development rises by 15%);
- Improvement in system design, more effective solutions and shorter code owing to brainstorming;
- Difficulties are easily settled;
- Growth of employees learning;
- Rise of knowledge volume of a system;
- Much enjoyment derived from work.

### 8.4.8. Code Refactoring

The article [13] says about refactoring as the most powerful concept appeared in last 2000 years. I think that it is an overstatement, though the significant benefits of the process of the idea comprehension by programmers are obvious.

Refactoring is code improving, altering its internal structure without changing its external behavior. (See the book [14]). It's simple. This is that every programmer was involved when he was feeling some kind of discomfort due to the earlier produced code («code smells»).

Frequently programmers do not write an ideal code, sufficiently good code. Only later when they observed the code duplication, tangling logic and long methods, they began to establish order.

The essential elements of a Refactoring process constitute a catalogue of refactorings (according to the Pareto's rule 20% of them give 80% of effect), and principles are the following:

- To make changes in parts;
- To test before and after refactoring;
- In case if it is needed to introduce a new functionality, but is difficult (Refactoring should be used), it is better to implement refactoring, and then to write a new code.
- The rule of "tree blows": firstly you should create any code that satisfies the tests; secondly, after coming across the same code, understanding its defects you should think about, and finally after the third time begin to practice refactoring.

Also I would like to distinguish the appearance of automated refactoring tools (Refactoring browser).

### 8.4.9. Incremental Development

Presently nobody is seemed to employ the «waterfall» model. But often it is only declared concept far from reality.

In most companies (especially in large-scale ones) there is a strict functional decomposition, i.e. every department fulfils a narrow segment of work. Hence, it seems easier and more effective to implement the work dividing it into parts and make the project sufficient foe success executing every phase in turn: analysis, designing, creating DB, coding, testing and deployment.

This model is based on the suggestion that value of correcting defects exponentially rise according to the stage of work. I.e. it is much expensive to correct a defect having crept into the program during the testing process (or even worth the maintaining process), than cording. Therefore it is advisable to carry out every stage with maximum quality in order to prevent defects in the future («big design» in the beginning of the development).

However, the life makes correction in any plans. Technologies, platforms and business alter so quickly that time and resources spent on the detail requirement description, for example, with the help of a requirements specification, seem to me absolutely useless.

Of course, there is hardly a good architecture without prior system framework designing. Presently I see the solution of the problem with continual business requirements changes in gradual growing in contrast to building the product.

## 8.5. Lifecycle Stage –Testing

### **8.5.1. Continual Testing**

Perhaps, participating in last projects, which were implemented with the XP method, influenced me strongly, but now during the project development I treat «the mixed conceptions» (lifecycle stages) much tolerantly. I consider it is rather effective immediately to test every the element developed before its integrating into the general system.

For instance, after the new DALC (Data Access Logic Component) having been developed, it would be appropriate to write a test (usually a console application is enough) that verifies a component initialization, work with data base and business logic practices. It is necessary to test the correspondence between internal state and expected behavior of an object in a result with the help of a debugging program (or somehow in a different way).

This simplified testing permits to distract attention from the complex graphic interface and makes it possible to control “the essence” of an object model new element.

This method provides a particular benefits if it is necessary to be sure in correct functioning of the system element that is used with the great number of starting conditions (for example, algorithms).

For automation of this testing there were invented some tools. (Their name contains the word «Unit» - JUnit, NUnit, etc. Prefix characterizes the language/development environment of a program being tested). The application recognizes and loads the test class methods with the help of reflexion.

## 8.5.2. Tests Automation

I hope that the time passed away long ago, when the work of a tester looked like a daily monotonous entering the same data for the purpose to see the system's behavior. Presently, a tester is not armed worse than programmers are, and often even better. There is a large selection of tools for testing nowadays:

- Revealing memory leakage
- Determining tested code coverage
- Defining velocity of methods work
- Testing user interface
- Load testing

Great advantage in the work of QA department is availability of tools allowing testing process to be automated. After adding new functions in a system, it is very important to make sure that the work of the existing functions was not interrupted. The system that testing the most control examples by itself without human interference (for instance, at free time) is relevant for this purpose.

Independent testing (by a program, not by a man) together with the principal "who breaks the compiled project that finds and correct the bug", according to my opinion, raise the personal responsibility of a developer during integrating his code into the general system.

### **8.5.3. “Narrow” Tests**

While testing the system it is relevant to add the components one by one in order to focus on a «narrow section». Thus, detecting errors of a module does not influence the results of the other. Errors do not suffer interference and it is easier to find them out.

It resembles an incremental development because it is always easy to carry out a task dividing it into small parts.

Generally if there is a lot of defects appeared with a little change, it means that «bugs do not simply dwell here, there is a bugs’ nest» (See the article [33]). This situation signals about serious problems of the software developed.

#### 8.5.4. Data Set

For effecting the qualitative process of testing it is important to possess a full set of data that presents different variants of automatic process situations. More frequently, the final system user provides this set of data, but sometimes developers can enlarge the presented set with their own variants of exception cases.

For example, a thought to check the system for turning off the light, connection failure or sudden inaccessibility to the DBMS in the middle of the business transaction does not come into the user's mind.

It should be pointed out that testing is necessary to implement with the «sample» data sets, but not with the developers' sets. I.e. business-essences should be included in the system correctly through the corresponding functionality, but not be generated by a «magic bottom».

Frequently the data generated or included «manually» in a DB are not the same than that created by the system «officially». If we take into account the quality of data, first of all there are distinguishes in default values of data columns. Sometimes there can be also differences in things that the DBMS and program allow to enter.

### 8.5.5. Program Environment

The typical developer's expression is widely known as «universal reply» to an error message on the customer's computer. «My machine was working perfectly well...at least 5 minutes ago. This problem is *theirs*, not *mine*». Actually this is the problem of testers, who did not take into account all (or almost all) the situations of the environment in which the program was installed.

Environment means a medium in which an application is working. On the one hand, it includes hardware: computer (of diverse power levels), monitor (of diverse sizes and resolution), peripheral devices, components and all the things related to hardware. On the other hand, this is software: operational system and patches (service packs), drivers, JVM, office packets, DBMS and data access, tools for accounting, communicating and servicing, etc.

It is necessary to provide general system testing in various software and hardware configurations. If a program is intended for working only with a certain version of this or that component, the user's access should be facilitated as much as possible. For example, if an application requires for work the installed component X of the version Y, it would be better to include it in the installation packet than to expect that a user will correctly install it by himself.

### **8.5.6. Defect Tracking**

The systems dealing with defect-tracking are widely spread in last years and are excellent examples of how a realization of a simple idea can be of great benefit. The essence of these tools is to intensify the control and to automate the process of tracking, correcting and testing defects in the software.

A developer having discovered a defect (an owner) gives it the code name describes its symptoms and situation that was a result of this defect. A manager of the project appoints the priority, terms and a person responsible for settling the problem. The appointed person solves the problem and informs «the owner». The last makes sure that the problem has been settled and «closes» the task.

All these operations can be made in «the paper form», for example in MS Word, but the systems integrating the whole process of keeping information concerning defects, process participants notifications via e-mail, receipt of accounting, significantly increase the effectiveness of work.

Further analysis of defects, functions of their distribution in the time, classification taking into account priority, criticality and frequency, average time of correction – all these help at least to understand the reasons of appearing (and to make corresponding arrangements) if not to guard against their creeping into the program.

### **8.5.7. Usability**

Frequently, software testing is understandable as testing functionality, productivity, ranging, bug-proof and other “serious” things. At the same time the aesthetic perception of the program by users and the serviceability are forgotten.

Of course, the possibility of choosing the goods from the list of available ones, printing a receipt and paying for the purchase in the convenient for a buyer way is important for the process of making a new sale. But if the «main» functional effect is obtained you should deploy some resources in order to make the process more convenient. For example, to count and mark out the total sum of purchases, to make auto-substitution of names according to the first letters with entering a new ware (but not make an operator to choose wares from the whole list), to provide the work of «hot» keys, etc.

The book [11] describes relatively easy, but effective methods of increasing subjective satisfaction with the work.

It can occur to be useful to make a video recording of users’ work with applications in order to define on what kind of work they spend more time.

Popular ways of working with applications with the help of «Wizards» confirm the idea that «the programs should be written for users, but not for programmers». Here is a great merit of the Microsoft Company. You spend much shorter amounts of time on learning its products, even such complex as DBMS, than the products of the competitors.

## 9. Conclusion

Well, its all for now. The book has been over quickly, hasn't it? I also like short books. I always prefer «user» books, in contrast to «User Bibles» of the X technology. Why do we need to read a "thick Talmud", if we can spend 5 minutes in order to look through Getting Started and begin to use the product, while the problems can be solved in the process of working?

Of course, you can express some criticism concerning absence in the book «the whole picture» of the software product lifecycle from the birth to the death, actually I planned to include some more chapters.

In the chapter «Lifecycle – Implementation» I wanted to tell about such SPS as methodologies of «stress softening» for users in the process of transfer to a new product. Qualitative documentation, training, data converting from an old system to new one would soften the process of transfer.

Close connection with the final users, opportune reaction on discovered problems with the help of patches and hot-fixes. Keeping and controlling the versions of successful and test product builds, conducting history of implementation for every customer. Creation of install package and process of components updating.

In the chapter «Lifecycle- Maintenance» I wanted to describe the methodology of making changes in the existing software. This is a rather important matter because maintainers are often not first developers. If the system can «stand» some changes of little significance in an interface, reports

and data format, while with the critical mass to be accumulated (entropy), sometimes it is easy to give up and rewrite everything from the beginning.

An appreciable competitive advantage is the «hot support» service of customers. A user would be pleased to know that in any time of day and night the service knows about him, remembers and is ready to assist. Of course, with the business to be expanded, every separate customer contributes less (in percentage) in the company's prosperity, but it resembles an avalanche – if one would go away, all other would follow, and there is no far from a bankruptcy.

All these things could be written if not to take into account one circumstance. The matter is that I am a venturesome and easily carried away person. That is why after having begun to develop the theme of SPS I could not stop (though if you are reading this chapter I could ;-).

After two software lifecycles mentioned above I thought about one more – product advertisement. An operational system is not something in the middle between shampoo and pampers. The popularity of software can be hardly increased by the advertisement like: «There is an unknown bug in every seventh copy of text processor, find it and win a trip in Redmond at your out account». ;- ) «A message about invitation in America appears after half-hour work with the program!». The software advertisement is a separate theme that is waiting for investigators.

After advertisement I thought about the process of software selling, strategies of developing and integrating with other products, and so on.

As a result I came to a conclusion that this book can be written for ages, and my plans were different (besides the book I have the working day from 9 a.m. to 6 p.m.). Writing the draft took more than 3 months (with the two to be planned). A lot of interesting books, articles, product versions are realized every day... and all of them are needed to read, analyze and try...

Hence, I decided to focus only on the stages of *producing* the item. All the amendments, which would come into my mind, reviews received, as well as results of my further professional development will be most likely included into the second edition of the book (with the free time available).

Now it's time for the concluding remarks. When the most part of the material has been already written I discovered the article [34]. I expected to see there the familiar names – Brooks, Cockburn, Fowler, etc. But I was mistaken. No one book mentioned was familiar to me. The name McConnell aroused some dim associations, but generally connected with the coffee brand.

I understood that the more I tried to perceive the world, the wider horizon was opened in front of me and the knowledge became unlimited. Having studied the book Turbo Pascal 7.0 some time ago, I considered myself as a cool guru ;-). Now I can acknowledge that «wonderful trouble» (See the book [6]) connected with the complete impossibility to master big amounts of knowledge in a certain branch of science, make the profession of programmer absolutely perfect.

I think that the most part of SPS is unknown for the wide audience yet. Fortunately, there are people who try to make the life better. And they are not

special – everybody must have enough force in order to raise the level of development at least by a few millimeters.

It can't be that hard. That is why I believe that everybody (including me) will have enough forces to increase productivity that allows to share the surplus but not to fight for lack (See epigraph to book [8]).

## 10. Bibliography

### Books:

1. Doug Rosenberg, Kendall Scott, «Use Case Driven Object Modeling with UML : A Practical Approach»
2. Dean Leffingwell, Don Widrig, «Managing Software Requirements: A Use Case Approach, Second Edition»
3. Alistair Cockburn, «Writing Effective Use Cases»
4. Alistair Cockburn, «Agile Software Development»
5. Edward Yourdon, «Death March: The Complete Software Developer's Guide to Surviving 'Mission Impossible' Projects»
6. Frederick P. Brooks, «The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition»
7. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, «Design Patterns»
8. Grady Booch, «Object-Oriented Analysis and Design with Applications (2nd Edition)»
9. Craig Larman, «Applying UML and Patterns»
10. Ed Sullivan, «Under Pressure and On Time»
11. Vlad Golovach, «Design of User Interface», Usethics Company
12. Peter Coad, Mark Mayfield, David North, «Object Models: Strategies, Patterns, and Applications»
13. Allen I. Holub, «Enough Rope to Shoot Yourself in the Foot: Rules for C and C++ Programming»

14. Martin Fowler, «Refactoring: Improving the Design of Existing Code»
15. Lee Iacocca, «Iacocca: An Autobiography»
16. Kent Beck, Martin Fowler «Planning Extreme Programming»
17. John M. Vlissides, «Pattern Hatching: Design Patterns Applied»
18. T. Badd «Object oriented programming»
19. Martin Fowler, «Patterns of Enterprise Application Architecture»
20. Alan Shalloway, James R. Trott, James Trott, « Design Patterns Explained: A New Perspective on Object-Oriented Design»
21. Jon Louis Bentley, Jon Bentley, «Programming Pearls (2nd Edition)»
22. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, «Introduction to Algorithms»
23. Donald E. Knuth, «The Art of Computer Programming, Volumes 1-3 Boxed Set»
24. David A. Marca, Clement L. McGowan, «SADT: Structured Analysis and Design Techniques»

Sites:

1. [www.interface.ru](http://www.interface.ru) – documentation, excellent Center of Learning
2. [www.gotdotnet.ru](http://www.gotdotnet.ru) – everything about .NET environment
3. [www.microsoft.com](http://www.microsoft.com) – Microsoft products. You can treat as this firm (and its leader) as bad as you wish, but their progress in software standardization and development of the whole field seems unlimited to me

4. [www.xprogramming.ru](http://www.xprogramming.ru) – the most famous Russian site dedicated to the methodology of Extreme Programming
5. [www.maxkir.com](http://www.maxkir.com) – excellent translations of popular articles and, of course, XP
6. [www.osp.ru](http://www.osp.ru) – one of the best on-line IT-magazines in Russia
7. [www.citforum.ru](http://www.citforum.ru) – great number of diverse documentation. Inestimable achievements in searching information for essays, articles, books, etc.
8. [www.optim.ru](http://www.optim.ru) – not bad Russian on-line IT-magazine
9. [www.rational.com](http://www.rational.com) – without comments. Everything about Rational products, RUP methodology and UML
10. [www.therationaledge.com](http://www.therationaledge.com) – magazine covering the last events in the Rational world
11. [www.rsdn.ru](http://www.rsdn.ru) – excellent base of knowledge of programming and not only
12. [www.dotsite.spb.ru](http://www.dotsite.spb.ru) – everything about C# language and .NET environment
13. [www.sql.ru](http://www.sql.ru) – the most famous resource related to DBMS. You can spend the whole your life for reading forum themes
14. <http://russian.joelonsoftware.com/> – lambent, humorous and philosophic Joel Spolsky's notes on software development (though not without praising *his own* company)
15. [www.xcomx.narod.ru](http://www.xcomx.narod.ru) – everything about XCOM, aliens and Gillian Anderson

16. [www.softcraft.ru](http://www.softcraft.ru) – programming with different faces. It makes you believe in advantages of Russian programmers in comparison with Indians, Chinese and other «offshore giants»
17. [www.caseclub.ru](http://www.caseclub.ru) – everything about CASE
18. [www.testers.com.ua](http://www.testers.com.ua) – testing
19. [www.refactoring.com](http://www.refactoring.com) – the real Refactoring
20. [www.omg.org](http://www.omg.org) – standards, specifications
21. [www.1001.vdv.ru](http://www.1001.vdv.ru) (part «Solo on keyboard») – without the blind method of typing this book would appear a few months later (perhaps, it would not appear at all because my patience have given out before)
22. [www.cetus-links.com](http://www.cetus-links.com) – variety of links of useful sites of IT-world
23. <http://www.microsoft.com/rus/msdn/msf/> – Microsoft Solutions Framework
24. <http://is.twi.tudelft.nl/~hommes/toolsub.html> – the widest classification of methodologies for business modeling and products supporting them

Articles:

1. <http://www.osp.ru/os/2002/10/039.htm> – «How to succeed in hopeless projects», Magazine «Open Systems Publications», #10, 2002, Constantine Berlinsky
2. [http://software.ericssink.com/Career\\_Calculus.html](http://software.ericssink.com/Career_Calculus.html) – «Career Calculus», Eric Sink
3. [www.softcraft.ru/design/moving.shtml](http://www.softcraft.ru/design/moving.shtml) – «New Initiative in Programming. Movement

- for open project documentation», 2003, A. A. Shalito
4. <http://www.interface.ru/fset.asp?Url=/rational/014.htm>, «Rational Rose, BPwin and others – Aspect of Business Processes Analysis», Magazine «To Manager of the Information Service», #11/2000, Pavel Saharov
  5. <http://www.interface.ru/fset.asp?Url=/erp/news/m010628491.htm>, «Automation of Chaos», Andrey Akopiants
  6. <http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=A8381E9C-884D-4CB2-9DBE-255C2790634B>, «Designing Data Tier Components and Passing Data Through Tiers», Microsoft Corporation, 2002г., Angela Crocker, Andy Olsen and Edward Jezierski
  7. [http://www.osp.ru/pcworld/2004/03/102\\_print.htm](http://www.osp.ru/pcworld/2004/03/102_print.htm), «Niklaus Wirth – a Patriarchy of Reliable Programming», *PC World*, #03/2004, Ruslan Bogatyriov
  8. [http://www.osp.ru/cio/2004/04/080\\_print.htm](http://www.osp.ru/cio/2004/04/080_print.htm), «Blast wave of CMM», *IS Manager*, #04/2004, Christopher Coch
  9. [http://www.osp.ru/cio/2004/04/088\\_print.htm](http://www.osp.ru/cio/2004/04/088_print.htm), «Model of Data Domain», *IS Manager*, #04/2004, Dmitry Tsutsaev, Andrey Alexeenko
  10. [http://www.osp.ru/cw/2004/16/054\\_1\\_print.htm](http://www.osp.ru/cw/2004/16/054_1_print.htm), «Goal – to Staff the Project Team», *Computerworld*, #16/2004, Cathlin Melamewka

11. [http://www.osp.ru/os/2004/02/072\\_print.htm](http://www.osp.ru/os/2004/02/072_print.htm), «How Software Engineers Use Documentation: The State of the Practice», Open Systems Publications, #02/2004, Timothy Lethbridge, Janice Singer, Andrew Forward
12. <http://xprogramming.com.ua/advantages.php>, «Advantages of XP in Comparison with Other Famous Development Methodologies», Alexander Fedorenko
13. <http://xprogramming.com/xpmag/AchillesTortoise.htm>, «Misconceptions in XP II: Achilles and the Tortoise Discuss Design», Chaos Engineering, Bryan Dollery
14. <http://members.aol.com/humansandt/papers/pairprogrammingcostbene/pairprogrammingcostbene.htm>, «Pair programming cost bene», Alistair Cockburn
15. <http://xprogramming.ru/Articles/CodeSmells.html>, «The Code Smells Bad», Ruslan Eriomin
16. <http://www.martinfowler.com/articles/explicit.pdf>, «To Be Explicit», ThoughtWorks, Martin Fowler
17. <http://www.martinfowler.com/articles/newMethodology.html>, «The New Methodology», ThoughtWorks, Martin Fowler
18. <http://www.crystallmethodologies.org/articles/jmc/justintimemethodologyconstruction.html>, «Just in time methodology construction», Humans and Technology Technical Report, 2000.01, Alistair Cockburn
19. [http://www.adaptivesd.com/Articles/Retiring\\_LC\\_Dinosaurs.pdf](http://www.adaptivesd.com/Articles/Retiring_LC_Dinosaurs.pdf), «Retiring LC

- Dinosaurs», Software Testing & Quality Engineering, 07/08, 2000, Jim Highsmith
20. <http://www.martinfowler.com/articles/designDead.html>, «Is Design Dead?», ThoughtWorks, Martin Fowler
  21. <http://alistair.cockburn.us/crystal/articles/uctyl/usecasestenyyearsater.htm>, «Use Cases, Ten Years Later», Humans and Technology, Журнал STQE, 03/04, 2002, Alistair Cockburn
  22. <http://www.crystallmethodologies.org/articles/mpp/methodologyperproject.html>, «Methodology per project», Humans and Technology Technical Report 04.1999, Alistair Cockburn
  23. [http://www.controlchaos.com/Silver\\_Bullet\\_or\\_Bitter\\_Pill.pdf](http://www.controlchaos.com/Silver_Bullet_or_Bitter_Pill.pdf), «Silver Bullet or Bitter Pill?», Ken Schwaber
  24. [http://maxkir.com/sd/people\\_as\\_nonlinear\\_RUS.htm](http://maxkir.com/sd/people_as_nonlinear_RUS.htm), «Characterizing people as nonlinear component in software construction», Humans and Technology, 10.1999, Alistair Cockburn
  25. <http://www.xprogramming.com/xpmag/expDocumentationInXP.htm>, «Essential XP: Documentation», 11/21/2001, Ronald E Jeffries
  26. <http://www.xprogramming.com/xpmag/manualsInXp.htm>, «Manuals in Extreme Programming», 11/21/2001, Ronald E Jeffries
  27. <http://maxkir.com/sd/testing.html>, «Organizing and Naming Automatic Tests», February-March, 2003, Kirill Maximov

28. <http://xprogramming.ru/Articles/ExtremeTesting.html>, «Extreme Testing», Roman Eriomin
29. <http://xprogramming.ru/Articles/CustomerInXP.html>, «Customer. His Habits and Features», Roman Eriomin
30. <http://www.objectmentor.com/publications/xpepisode.htm>, «Engineer Notebook: An Extreme Programming Episode», Robert C. Martin and Robert S. Koss
31. <http://www.pragmaticprogrammer.com/articles/stqe-01-2002.pdf>, «Learning to love unit testing», Dave Thomas and Andy Hunt
32. <http://xprogramming.ru/Articles/TDD-PERL.html>, «Software development based on testing. Example in PERL», Denis Kosyh
33. <http://nosorog.org/cgi-bin/statyipk.pl?nm=1069802969>, «Programmers and Cosmopolitism», unknown author
34. <http://www.silicontaiga.ru/home.asp?artId=2429>, «Ten books about programming that amazed the world, but still unknown in Russia», *Computerra*, Andrey Terohov
35. <http://www.klerk.ru/soft/1c/?1880>, «Systems of Managing Knowledge and Continual Lifecycle of the Corporate Informational System», 15.01.2003, Alexey Kabanov
36. <http://www.nstda.ru/home.asp?artId=2142>, «Requirements to Informational Systems and Lifecycle Model», Carabi Solutions, 11/12/2003, Ekaterina Koltunova

37. <http://www.interface.ru/rational/teh.htm>,  
«Software Development Technology», site  
"Corporative Systems", 1/2002, Dmintry  
Bezugly
38. <http://www.vernikov.ru/material89.html>,  
«What CASE Harm an Organization the  
Less?», 2001, Sergey Rubtsov

# 11. Copyright

In respect of this material there are the following rules:

1. Only the author can make amendments in the text;
2. The on-line publication does not require the author's approval;
3. The translation of the text and/or publication requires permission of the copyright owner.

Please, contact the author of this book:  
[\*\*nsp\\_book@mail.ru\*\*](mailto:nsp_book@mail.ru)

The author accepts any fair criticism, assistance in design, literary correction of the text and offers for cooperation.

Sincerely Yours,  
Constantine Berlinsky,  
Software Developer of the IT-Department  
of the Maximum Company,  
Republic of Moldova