SILESIAN UNIVERSITY OF TECHNOLOGY PUBLISHING HOUSE

SCIENTIFIC PAPERS OF SILESIAN UNIVERSITY OF TECHNOLOGY 2019
ORGANIZATION AND MANAGEMENT SERIES NO. 139

SOFTWARE DEVELOPMENT PROJECTS AS AN ENGINE OF GROWTH FOR ORGANIZATION'S BUSINESS VALUES

Tomasz KOWALCZYK

Warsaw University of Technology, Warsaw; tomasz.kowalczyk@pw.edu.pl, ORCID: 0000-0001-5469-2848

Abstract: According to the latest research of The Standish Group only 14% of software development projects provide high satisfaction with high return on value to their users, sponsors and other stakeholders. Also surprising are the results of the correlation study between projects alignment to the organization's strategy and achieved business values. Projects that are less suited to the strategy allow for achieving twice as large business values as projects with high alignment to the strategy. The reason for this phenomenon could be greater scale of innovation of the projects vague and distant from the strategy. However, this is in contradiction with the principles of the project portfolio management which are focused on the alignment between projects and strategies. The paper provides recommendations of how to achieve and deliver business values and activate an engine of growth, which – in opposite to the high level's strategies – answer to the real and up to date stakeholders' needs. It also shows the change in the way of thinking, propagated by Tom Gilb in the Competitive Engineering and the change from focusing on functions and features to the quantitatively described value improvements.

Keywords: planguage, impact estimation, value improvements, competitive engineering.

1. Introduction

In the second decade of the 21st century The Standish Group (known as research advisory organization that focuses on software project performance) created a new definition of projects' success called "pure success" that establishes the whole new meaning of successful projects.

As the traditional measurement considers projects' success basing on golden triangle (on time, on budget and on target) and the modern measurement defines it as: on time, on budget and with satisfied customers (regardless to the original scope), the "pure success" measurement doesn't, in fact, consider any of those set of factors. The new approach allows to call the project as 'successful' only when it delivers a high or very high customer satisfaction and high or very high return on value to the organization at the same time (Johnson, 2018). These are the only things that matter here.

http://dx.doi.org/10.29119/1641-3466.2019.138.10 https://www.polsl.pl/Wydzialy/ROZ/Strony/Zeszytynaukowe.aspx

The results obtained from the CHAOS database containing 25000 records from Fiscal Year 2017 shows in the Table 1 that only 14% of examined projects were resolved as successful. Another 67% of projects were resolved as challenged, that means they didn't meet the highest customer satisfaction nor the highest return of value. 19% of projects were classified as failed that means they have been canceled before being resolved or haven't been used at all (Johnson, 2018).

Table 1.
Resolution by Pure Measurement

Year/Resolution	2013	2014	2015	2016	2017
Successful	16%	15%	17%	17%	14%
Challenged	65%	68%	64%	66%	67%
Failed	19%	17%	19%	17%	19%

Table shows "Pure" resolution of all software projects, 2013-2017, within the CHAOS database. Pure resolution measures only "high value" and "customer satisfaction". Adapted from: "Decision Latency Theory: It Is All About the Interval" by J. Johnson. Copyright 2018 by The Standish Group International, Incorporated.

What is worth noting, is that even the most advanced IT systems will not always meet actual customers needs. The CHAOS Report shows that many of functions and features of software are not being used at all. Moreover, it's worth to remember, that the more functions and features there are, the more cost and risk projects generate. That's why customers satisfaction doesn't necessary mean delivering projects in the original scope.

Another metric related to the "pure measurement" is strategic goals defined by the project's position in relation to the organizational goals. Table 2 shows that 26% of considered projects were successful. It means they are precise or close to the organizational strategy. 56% of projects were challenged, that means they are loose, vague or distant from the organizational strategy. 18% of projects were failed. These results are not surprising until we tried to pair strategic goals with values returned to the organization. Only 20% of "precise" projects delivered high or very high values, while the same levels of values had been delivered by 44% of "distant" projects (Johnson, 2018).

Table 2.
Resolution by Strategic Goal Versus Value Measurement

Goal	%	Very High Value	High Value	Average Value	Low Value	Very Low Value
Precise	11%	7%	13%	53%	21%	6%
Close	15%	8%	16%	52%	19%	5%
Loose	21%	12%	22%	47%	15%	4%
Vague	18%	17%	32%	39%	8%	4%
Distant	17%	15%	29%	21%	19%	16%
Failed	18%					

Table shows percent of projects in relationship to the strategic goals as measure of success (column 2). Columns 3 to 7 shows the return on value to the strategic goals. Adapted from: "Decision Latency Theory: It Is All About the Interval" by J. Johnson. Copyright 2018 by The Standish Group International, Incorporated.

The CHAOS Report authors tried to consider the meaning of the fact that return of value was smaller when projects were closer to the strategy. Their opinion was that challenged (especially vague and distant) projects are more innovative and are often business disruptive, with higher returns (Johnson, 2018). This phenomenon may also have other reason. In most cases organizational strategy is just a will or even a dream of their founders. Meanwhile, market demand may not match company's services and products which are driven by the strategy. That is why organizations should adapt their offer to the real customers' needs (Ries, 2012).

Projects alignment to the strategic goals (and looking wider - to the organizational strategy) is also an issue and subject of project portfolio management, for instance in accordance with The Standard for Portfolio Management issued by Project Management Institute. Basing on PMI definition, an organizational strategy is a bundle of goals and policies showing directions of development. It gives a primary input to portfolio management which acts as the vehicle through which initiatives and investments are undertaken to realize strategic goals and objectives (Project Management Institute, 2017). Referring again to the results of The Standish Group research, we can recognize a really serious problem. Distance between the project's results and strategic objectives doesn't have positive correlation with organizational strategy. Projects that precisely hits the strategy, deliver twice lower value than projects identified as vague or distant, and this contradicts the principles of project portfolio management.

Currently implemented IT projects, programs and portfolios both in the sector of companies and governmental organizations confirm phenomena described above in practice. The effectiveness and valence of IT projects has remained at the similar level for over 30 years, despite of improvements and optimizations realized through the software engineering. Such type of work environment is exposed to frequent conflicts between customers and suppliers that lead to increasing losses. The popularity and availability of software will continue to increase in next years and decades. It will be influenced, among others, by the development of the 5G cellular network and the related expansion of phenomena such as big data and artificial intelligence. Therefore, improving the results and values achieved by IT projects is a very important issue.

The main objectives of this article are to define the concept of business values and value-based approach for activating an engine of growth and managing software development projects delivering quantified benefits for Stakeholders. In order to achieve these goals, the paper is divided into two major parts. First covers basic knowledge about planning language (created by Tom Gilb), called "planguage", which focuses on quantified stakeholder's needs and the related requirements and value improvements. Second part describes key principles for successful requirements, presented in the form of checklist that can help managers in value management during the entire software development lifecycle.

2. The Concept of Business Values

Nowadays due to the rapidly changing world people cannot rely anymore on their (once a confident) knowledge of how to solve problems similar to ones they had solved before. Stable business, social and IT environments have changed forever (Gilb, 2005).

It is no different in the case of software development. Until the mid-sixties of the XX century software had been created by users themselves for their own needs (primarily for scientific purposes) or in close cooperation with end users. In such a homogeneous environment, stakeholder expectations (expressed in the form of functional and non-functional requirements) were well understood. However, the situation changed significantly in the late sixties, when development of computer hardware and programming languages enabled the modeling of much more complex information systems. People discovered the usefulness of the software in completely new fields such as information management or supporting production processes. Unfortunately, most of the implemented projects were unsuccessful. It was the beginning of the so-called "Software Crisis", which made it clear to the engineers that their methods and techniques do not keep up with user's expectations. At that time engineers and managers began to look for a new way out of the crisis. In the result software engineering has born as a set of new directions, methods and techniques for software development (Jaszkiewicz, 1997), that was coincided with the rise of the planning language (called "Planguage"), created by Tom Gilb.

2.1. Planning Language, Called "Planguage"

Planguage was created as an open and flexible communication and cooperation "platform" for interdisciplinary teams working towards well-documented common purposes. It supports the whole software development process, from requirement specification to product delivery as well as giving opportunity for tailoring specific projects, organizations and cultures in order to find out 'what works now' by means of practice, not theory. From team's perspective planning language helps thinking as engineers and managers, not only as programmers. Thanks to this Planguage allows to concentrate on stakeholder-critical values, instead of focusing only on functions, use cases, and code delivery (Gilb, and Brodie, 2011).

2.2. Values and Requirements

The main assumption for each project should be achieving Stakeholders' values (defined as the "benefit we think we get from something"), not delivering defined functionalities.

The issue with conventional approach for requirements elicitation and gathering is that it is not close enough to Stakeholder's values. IT business analysts usually fail to get enough information for calculation of values as well as business Stakeholders frequently fail to justify requirements using values. This shows the greatest danger – a lack of basic information

allowing to engineer and prioritize implementation tasks in order to deliver the highest value, even if the requirements are fulfilled in the meaning of stakeholder's functional and non-functional expectations. Another issue is a specificity of values, which are multi-dimensional beings. A given value can be understood on many levels, for instance on financial, environmental, architectural or competitive level. Therefore, using simple prioritization mechanisms such MoSCoW (ie. Must Have, Should Have, Could Have, and Would like to Have) is not enough (Gilb, and Brodie, 2011), because there may encounter a conflict between different levels of values.

Lack of consistent definition of the requirements is also an important issue. The most popular and simple classification based on the software engineering assumes that requirements are divided into functional and non-functional groups. However, this is not a complete classification, even without considering the value issue (Gilb, and Brodie, 2011). The concept of requirements types defined in Planguage includes several main categories and few subcategories, as follows:

- Vision Requiremets: at the highest level, the future direction for a system.
- Function Requirements: what a system has to "do": the essence of a system, its mission and fundamental functionality.
- Performance Requirements: the performance levels that the Stakeholders want – their objectives. How good? These can be further classified as:
  - Qualities: how well the system performs, for example: usability, availability and customer satisfaction.
  - Resource Savings: the required improvement in resource utilization: relative economic and other resource savings compared to defined benchmarks. These are known simply as "Savings".
  - Workload Capacities: how much the system performs. In other words, the required capacity of the system processes. For example, system peak processing volumes, speeds of execution and data storage capacity.
- Resource Requirements: the levels of resources that stakeholders plan to expend to develop and operate a system. Resources have to be balanced against the stakeholders' perceived values gained from the system functions and the system performance levels.
- Design Constraints: these are any design ideas that must be included in the system design. In order to be able to define values that can be measured, it is necessary to define quantified quality requirements.
- Condition Constraints: these are any additional constraints to those imposed by the function requirements, the performance requirements, the resource requirements and the design constraints. Condition constraints are often used to capture system-level constraints (for example, "the system must be legal in Europe") (Gilb, 2005).

From the Stakeholders' point of view the performance requirements are the most important for value delivery by far.

2.3. Scales of Measure

It is a well-known paradigm that management is possible only when the subject is measurable. That is why to achieve anything, quantification and measurement are required. Lack of measurability does not allow to specify precise criteria for judgment of failure or success (Gilb, 2005). This issue was already raised in 1998 by Simon Ramo, who wrote – "No matter how complex the situation, good systems engineering involves putting value measurements on the important parameters of desired goals and performance of pertinent data, and of the specifications of the people and equipment and other components of the system" (Ramo, and St. Clair, 1998).

A scale of measure is the heart of a scalar specification and essential to support all the project's targets and constraints. The scalar attributes (such as performance and resources) are best measured in terms of defined conditions, otherwise they lose its meaning (Gilb, 2005). Below, there are examples of the scales of measure, presented in the Table 3.

Table 3.
Examples of Scales of Measure

Performance	Effect of Change in Performance	Scale of Measure
Customer Satisfaction	Fewer letters of complaint	Number of letters complaining about a defined [Product] received within a defined [Time Period]
Customer Satisfaction	Fewer returned goods	Percentage of defined [Product] returned within defined [Time Period after Purchase] with defined [Customer Issue]
Environmentally Friendly	Improved rating as measured on international standard	Number of defined [Product Type] failing defined [Test] within a defined [Time Period]
User-friendly	Fewer errors made	Percentage of defined [Transaction Type] with defined [Error] input by defined [User Type]
User-friendly	Faster time for completion of transactions	Time in minutes for a defined [Transaction] to be carried out to <satisfactory> completion
Restful Ambience	Calming, relaxing effect	Percentage of users of defined [User Type] agreeing that defined [Room Space] was <restful>
Reliability	Fewer breakdowns	Mean Time Between Repair (MTBR)
Staff Satisfaction	Lower rate of staff turnover	Number of staff of defined [Job Description Response]
Predictability	Less variance in time to initial response	Percentage of service calls of defined [Service Type] exceeding <initial response> within defined [Time Period]

Adapted from: "Competitive Engineering – A handbook for systems engineering, requirements engineering and software engineering using planguage" by T. Gilb. Copyright 2005 by Elsevier Butterworth-Heinemann.

2.4. Epilogue of Planguage

Planning language is not only a value management tool. It is also a platform for precise communication. Thanks to predefined structure, including parameters, concepts and icons (selected examples are given in the Table 4) it allows to express thoughts in the way similar to the markup languages (like XML) but in the easier way, available even for rookie users.

Table 4.
Description of some of the main generic Planguage parameters, concepts and icons

Concept of Parameter	Meaning	Used for	Note also
Planguage Term	A term that is part of Planguage	Structuring specifications	Glossary contains a set of Planguage terms
User-Defined Term	A term defined by users	Identifying "local" user terms	It should be short and descriptive
Tag:	An identifier for a Planguage term or a user-defined term	Providing a unique "local" reference to a term	Hierarchical tags can be used. These can be used in full (very explanatory) or abbreviated depending on context
Gist:	A rough, informal, brief description or summary	Getting consensus initially. Summarizing finally	Usually not a precise, detailed or complete definition. For a scalar parameter, "Ambition" can be used to express the ambition level
Stakeholder:	Any person or organizational group with an interest in, or ability to affect, the system or its environment	Understanding who has to be consulted or considered when specifying requirements	Usually a set of several different stakeholders is identified
Status:	The approval level of the specification	Identifying which version of the specification is being used	For example: "Status: Draft." See glossary for additional terms to express approval level
Source: <-	Where exactly a given specification or part of it, originated	Used to enable readers to quickly and accurately check specifications at their origin	The icon for source is "<-". Usually the icon is used in specifications, rather than the term "Source"
Assumption:	Any assumption that should be checked to see if it is still applies and/or is still correct	Risk Analysis	Other more precise parameters should be used if possible, for example, Dependency, Risk
Fuzzy <...>	Identifies a term as currently defective and in need of improvement	Alerting the reader and author that the term is not trustworthy yet or lacks detail	The keyed icon for fuzzy is "<imprecise word>". The "<>" icon is always used

Adapted from: "Competitive Engineering – A handbook for systems engineering, requirements engineering and software engineering using planguage" by T. Gilb. Copyright 2005 by Elsevier Butterworth-Heinemann.

3. The Key Principles for Successful Requirements

Project management through Stakeholder's values requires an approach based on continuous improvements. Software quality and its business benefits refer not only to the testing phase. That is why the values should be considered at all stages of software development lifecycle. In order to do this, every project manager or product owner should act in accordance with the following principles:

- Understand top level critical objectives.
- Think stakeholders: not just users and customers!
- Focus on the required system quality, not just its functionality.
- Quantify quality requirements as a basis for software engineering.
- Don't mix ends and means.
- Capture explicit information about value.
- Ensure there is "rich specification": requirement specifications need more information than the requirement itself!
- Carry out specification quality control (SQC).
- Consider the total lifecycle – not just a focus on software.
- Recognize that requirements' change: use feedback and update requirements (Gilb, and Brodie, 2011).

3.1. Understand Top Level Critical Objectives

Understanding the top-level critical objectives, sometimes called as "high-level requirements" is a crucial thing for the project team, and it's unfortunately often being ignored. Those objectives, to be properly understood, have to be well-clarified and this is another case many project teams struggle with. Each of requirements must be clear, measurable and quantified (Gilb, and Brodie, 2011).

3.2. Think Stakeholders

Requirements are often being focused on user and customer needs, while many project teams don't take into consideration any other Stakeholders. It's worth to remember, that Stakeholders are not only customers and end-users; it's also anyone that has an interest in the project (e.g. Management, IT development and maintenance and etc.) (Gilb, and Brodie, 2011).

3.3. Focus on the System Quality, not just its Functionality

System quality is an important part of every IT project. It includes availability, usability, portability - basically, it's any quality that a particular Stakeholder may need. However, as the functionality of the system attracts an attention and be easier to understand, the system quality (especially quantified) seems to be often ignored, while it can be major driver for the project and business success (Gilb, and Brodie, 2011).

3.4. Quantify Quality Requirements

Every well-clarified requirement has to be quantified properly. Far too often people don't remember about the power of numbers. Nice-sounding words (f.e. "much better performance and amazing user experience") will never be a good replacement for accurate numbers that provides the opportunity to measure and track progress of the project (Gilb, and Brodie, 2011).

3.5. Don't Mix Ends and Means

People often confuse the solution with their real need, while in the most cases it's not the same thing. A particular need can be solved in many ways. Narrowing it to just one solution is a big mistake: first of all, people still struggle with defining their actual needs, so they can't possibly find the solution, when they don't even know what the actual problem is. Second – proposed solution may not necessarily answer for the real problem and may have unpredictable side effects (Gilb, and Brodie, 2011).

3.6. Capture Explicit Information About Value

Expressing values during the definition of requirements is a very difficult activity. It requires a mental journey to a higher level - above tangible things. People should ask themselves why they need particular things and the answer should be very deep, not only describe application design (Gilb, and Brodie, 2011).

3.7. Ensure There Is "Rich Specification"

Even the best-defined requirement itself is not enough. Equally important is deep specification of its background as it may contain the knowledge about things such as: who wants the requirement and when, what would be the impact of fulfilling this requirement and so on. This information allows to prioritize the requirement, judge its value and risk, etc (Gilb, and Brodie, 2011).

3.8. Carry Out Specification Quality Control

Right after specifying the requirements, the quality control should be carried out. None requirement should be released for use without it. There are three rules that have to be implemented in every requirement: "testable", "unambiguous to readers" and "no optional

designs present". The requirement that doesn't meet those rules, should be redefined, so it could really do its job (Gilb, and Brodie, 2011).

3.9. Consider the Total Lifecycle

Taking the entire software development process into consideration is crucial for avoiding problems with cost of maintenance and future development of the system. For instance, if we want to have a possibility to change a performance or capacity of the system in real time, it has to be designed into the system (Gilb, and Brodie, 2011).

3.10. Recognize That Requirements' Change

Already specified requirement is not a "being" that never can change, people cannot stick to it no matter what. It's important to use software development methods, such as agile, to collect feedback from Stakeholders and fit the requirements to Stakeholders' expectations and actual value. Because of different factors (ie. politics, economy, technology change) requirements may evolve during time and it's important to accept it instead of narrowing it down to tunnel vision (Gilb, and Brodie, 2011).

4. Conclusions

The modern IT project management landscape is dominated by agile methodologies like SCRUM, focused on the software development processes and functional requirements. Main assumptions of this philosophy are valuable, because they take into consideration not only technical issues but also the specificity of human behavior and relationships with other people. These phenomena led to increased efficiency and work's quality of the development teams. Unfortunately, it did not affect, nor took into consideration, the values that projects bring to the organizations.

The CHAOS Report shows that in 2017 only 14% of examined projects delivered high and very high values and high and very high customer's satisfaction. This is a result much below expectations. Expected values will not also be achieved by further optimization of the technology and development processes. Nowadays, the main problem is lack of a description of measurable business needs (so-called qualities), which are the basis for developing the specification of functional and non-functional requirements. Such description of top-level critical objectives as "Will provide a much more efficient user experience" or "A primary goal is to provide a much more productive system development environment then was previously the case" makes it impossible in practice to determine the functionalities that will certainly meet Stakeholder's expectations.

Performance requirements should contain at least: description, scale of measure, current (quantified) status of the need, expected value of the need and time of delivery (Figure 1).

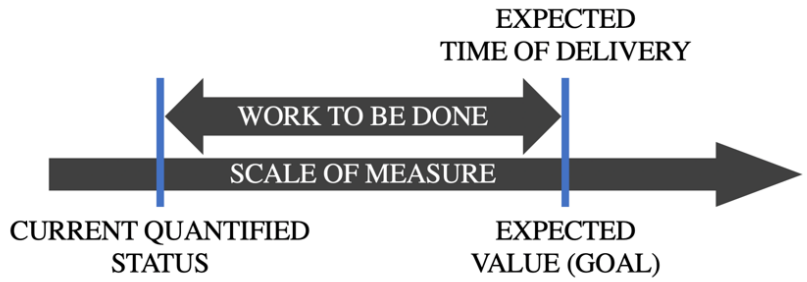

Figure 1. Goal's definition. Adapted from: own study.

Such set of parameters gives a chance to answer for the most important question in modern IT projects – "How will we understand that we have succeeded?".

References

1. Gilb, T. (2005). *Competitive Engineering – A handbook for systems engineering, requirements engineering and software engineering using planguage*. Oxford: Elsevier Butterworth-Heinemann.
2. Gilb, T., Brodie, L. (2011). What's fundamentally wrong? Improving our approach towards capturing value in requirements specification. *Core, 4,* 26-36.
3. Jaszkiewicz, A. (1997). *Inżynieria oprogramowania*. Gliwice: Wydawnictwo Helion.
4. Johnson, J. (2018). *Chaos Report, Decision Latency Theory: It's All About the Interval*. Boston: The Standish Group International, Inc.
5. Project Management Institute (2017). *The Standard for Portfolio Management*. Newtown Square: Project Management Institute, Inc.
6. Ramo, S., and St. Clair, R. (1998). *The Systems Approach: Fresh Solutions to Complex Civil Problems through Combining Science and Practical Common Sense*. Anaheim: KNI Incorporated.
7. Ries, E. (2012). *Metoda Lean Startup*. Gliwice: Wydawnictwo Helion.