A Contribution to Software Development Quality Management

Vidan Marković, Zora Konjović

Faculty of Organizational Sciences, Chair of Information Systems, University of Belgrade, 154 Jove Ilica, 11000 Belgrade, Serbia
University Singidunum, 32 Danielova, 11000 Belgrade, Serbia
vidan.markovic@fon.bg.ac.rs, zkonjovic@singidunum.ac.rs

Abstract: Disruptive digital technologies have increased the potential for international businesses to access clients' data in traditionally closed local markets (e.g. insurance, banking, etc.) without the need for a physical presence. In order to stay competitive in the long run, local companies have to invest more in both technology and sourcing strategies. In this paper, we propose an enhancement to the software development suppliers management process based on a continuous services comparison that can lead to proactive improvements in suppliers' quality of services. This paper explains this process and demonstrates a company case study experiment.

Keywords: Software Quality; Software Processes; Outsourcing

1 Introduction

The ability to react fast to a recognized need for change of the functionality or performance of an existing software service, or to an emerging business need for a new software service running in production, is becoming increasingly important [1]. However, obtaining and keeping the right resources to do this job is increasingly difficult and complex. Thus, outsourcing in software development has become inevitable in order to keep pace with these needs [2]. This means that the in-house software development process must adapt to this new reality as well. Handling both new software projects development and the maintenance of existing software services through a mix of internal and external resources is not a trivial task. Companies need to update/improve their software development strategies, and practice SW development results interpretations from different perspectives: from an immediate goal achievement to wider strategic alignments with business needs.

What really matters at the end of any software development project is the perceived value of a production and the quality of IT services. These basic

objectives must be the top priority during the whole software development supplier engagement life cycle: from the supplier selection process, to on-going operations, to the supplier's contract closure.

The internet of things, social networking, big data analysis, cloud technologies, and high mobility all require that user interfaces be available on various platforms and in various environments at any time. These developments have only increased the need for better management of simultaneous projects' developments, technical implementations, and operations' teams [3]. Agility, internal and external collaboration (with users, customers, partners, competitors, regulatory bodies, etc.), and the need for frequent deployments have also increased demand for better security procedures, policies, and tools.

In order to keep pace with these exponentially increased expectations of businesses, CIOs have to rethink and continuously improve IT human resources sourcing models and try to find the best fit solution for their particular companies and eco-system needs. This means creating a desirable balance between functionality, quality, costs, and risk targets for the company [4].

Even though the number of standardized software solutions for different business industries and the business processes within them are growing [5] (many are already available as SaaS Cloud solutions [6]), companies still strive to get differentiation in the market and gain competitive advantage from the specifics of their own solutions.

In this paper, we discuss the question of how to measure and manage the quality of tailored business software solutions when software is increasingly being developed outside of company control and the gap between knowledge inside the company and the knowledge of the code is growing. Finally, how do you mitigate that kind of risk for the business?

In order to answer this, we propose utilization of the Six Step Service Improvement (3SI) method [7] in managing quality, costs, risks, and general relationships with programming outsourcers. This method enforces continuous communication with outsourcers, which is a particularly important factor that influences the quality of software services.

The 3SI method can be used in supporting transparent, performance based relationships with suppliers. However, in order to gain the most benefit from using this method, it is very important that the IT organization is mature and capable enough to professionally plan, execute, and control products and services procurement in its best interest (based on predefined procedures, polices, and expectations fulfillment criteria).

We verified this method with a case experiment on a selected instance of the use cases' class. This case company was characterized by a relatively stable business environment, by the internal processes, organization, and culture of an experienced team, and by a relatively stable system architecture (company was at

CMMI level 4). However, this method could be applied in the various implementation scenarios (i.e. different business domains/eco systems, different companies' CMMI levels, different programming languages, etc.).

This paper begins in this introduction section by defining the problem and providing a brief summary of our work, then continues through an overview of related work and the hypothesis of the potential solution. After that, it gives a detailed explanation of the proposed solution based on the utilization of the 3SI methodology. It continues by describing the results tested and verified in the case study experiment, along with a description of the benefits of the proposed method utilization. Finally, the paper finishes with a conclusion, further work proposal, and the references list.

2 Related Work and Hypothesis

Managing the requirements in iterations and building solutions based on the integration of developed and tested components and packages represent, the core of modern software development processes [8, 9]. The complexity of software solutions is growing because of an increased need to dynamically exchange information with a high number of open systems and databases.

The complexity, is also reflected in a need for higher operating capacity, performances, and highly efficient development tools and components to cover an increasingly larger area of the application domain by integrated systems. Often without proper methodological management of all possibilities and risks of new technologies.

The pressure to shorten delivery times (agile solutions to software projects) leads to a quick release of a valid version, but can lead to problems in the future maintenance if the critical knowledge about the current version of system is not systematically storied in suitable, understandable and precise form.

In order to manage the increased expectations for software services it is important to manage the associated costs and risks (that could be again mapped to the costs). COCOMO II, and other similar methods, can give a good approximation of the software cost estimation [10, 11, 12, 13, 14].

The main issue with using these methods in today's business environments is the fact that business needs fast answers that ideally could be automated and ran regularly for decision making support. Typical scientific software economic measurements and costs estimate tools (e.g. KSLOC (thousands of Source Line of Code), function points, etc.) unfortunately, do not get enough attention from business to be used in a real-time environment. Another issue with the traditional scientific approach is that the data calculated and analysed in most scenarios *do not consider the specifics of the organization business and its eco-system*

dynamics and because of that are conceived as micromanagement tools rather than strategic decision making support tools.

That is the reason why, in business practice, most estimations are based on a *rule of thumb* principle. These estimates are more accurate when applied on similar projects (i.e. the same architecture, same platform to be developed and to run on, similar teams, etc.). Previous experience and lessons learned play an important role in these calculations.

However, with an open enterprise architecture, the increasing size of externally programmed code, and without sufficient and sustainable internal knowledge, setting a reference point for the quality, costs, and risk assessments of new software scheduled for deployment becomes extremely cumbersome.

There are principles like the Quality Improvement Paradigm (QIP), that define software discipline as evolutionary and experimental [15, 16]. This means that there is very little repetition in software development, which using statistical control in software quality control, like that used in manufacturing sciences, makes it extremely difficult and dubious [11]. The developers of QIP take a different approach than, for example, the authors of the CMMI and a number of other models that are based on the idea of statistical control of processes [16].

However, meeting the quality needs of software services includes the principles written by Deming [17, 18] and found in Total Quality Management (TQM) practice. Regardless of the particular flavour of TQM implemented, process definition, control, and improvement are always included as core TQM principles [19]. The main idea behind process control is that organizations are sets of interlinked processes, and improvement of these processes is the foundation of performance improvement [20, 21].

The oldest model that can be seen as an improvement action life-cycle model is the PDCA (Plan-Do-Check-Act) model developed by Shewhart and Deming [18]. It was originally devised for improving quality in manufacturing, and has its foundation in statistical quality control, i.e. controlling quality by applying metrics to the process.

All the basic principles of quality improvements still exist, but they now must work on higher dynamics and higher complexity. The time to market criteria has a higher weight coefficient within dynamic business environments (e.g. fast changes in products portfolio's, new organizations and processes due to mergers and acquisitions, economic crises, regulation requirements, etc.).

Squale quality models are concentrated on visualization of metrics where distribution maps, tree views, tree rings, etc. help in better understanding the quality of software, specifically by adding *practice* as an intermediate level between metrics and criteria as defined by ISO 9126 (that promotes a three-level model of quality: factors, criteria, and metrics) to support improvements actions [22]. However, even though with the Squale the visualization helps in

understanding the software quality issues from different a perspective (i.e. from software development level to CIO level), it still requires significant preparation and high-level of technical expertise in understanding the potential quality issues because of its strong bottom-up approach in software quality assessment.

We believe that in operational utilization of software quality improvement technics, where decision about improvements have to be made quickly (due to business ecosystems factors), it is critical to have a balanced mix of internal complexity (white box) consideration and external results, visible to end users (black box). It means that bottom–up and top-down approaches need to be in line with specifics of the business and software development environment.

Emerging DevOps methodology integrates development and operation activities in order to frequently generate new software deployments (i.e. in matter of minutes in some cases). New tools are arising in the market to support continuous deployment with high-level of automation.

Although all the above methods and software processes emphasize the need for continuous quality improvements, there is no clear and systematic approach on how to successfully manage and achieve continuous quality improvements in continuously changing business and technical environments.

We limit our analysis for the domain of business software solutions in services oriented industries such as banking, insurance, education, government, tourism, etc., where their core processes are heavily supported and influenced by various software utilization (made in-house, bought from third parties, and mixed solutions).

Hypothesis definition: There is *a holistic solution* that supports building higher quality software (for the business domain defined above, within specific industry, for the specific business function: i.e. claims management software for claims management department in an insurance company, or child care management software for social services department in a municipal government) by means of continuous improvements based on regular/periodical (with a particular sense of the business's *pulse*) utilization of the given methodology on defined sets of empirical, real-time data generated during software development and operations processes.

3 The Solution Proposal

"An incident is an unplanned disruption or degradation of service. A problem is a cause of one or more incidents. Quite often, in operations, these two terms are used interchangeably" [23, 24].

Minor outstanding incidents in some parts of the software product will be naturally discovered in production system utilization because with the higher complexity of software systems, there is a higher probability that some bugs will pass through all predefined levels of testing and quality assurance. That is the first reason why continuous improvements are necessary.

The second reason for software changes is a result of changes in the operating environment (i.e. infrastructure changes, operating system changes, other interconnected systems' changes, interfaces' changes, etc.) that require software adaptation to the new production environment.

Performance-related reasons also influence software changes. There are the "*Would like to have* type" of end-users' requests (MoSCoW rule) that are often purposely left as non-critical and dealt with in future software releases.

However, the most significant change requests come from a business's new/changed functional needs for a software service. More than 15% of software defects are related to requirement's errors [10]. Errors that were not detected in earlier phases of software development contribute to a higher cost in fixing defects later on [8, 14].

If, besides the above reasons for software services changes, a software service suffers unexpected degradation of quality (e.g. more bugs, more production fixes, non-compliant SLA, etc.), higher development and/or maintenance costs and risks, and if similar incidents repeat in the same or other parts of the system, then this would normally need to be escalated as an issue [24, 25]. Moreover, if the software program code has been signed by the same supplier of the programming services, and there is a recognizable pattern (rather than just a normal variation of errors within predefined statistical control boundaries [17, 23, 26]), then the company has a *problem* that needs to be resolved.

The normal engineering tendency is to technically rationalize increasing numbers of incident occurrences as (for example): too many changes on initial requirements, no ability or stability to define firm scope, poor business analysis, not properly done design (e.g. no UML diagrams, just a simple user-story or some kind of specification with a fractions of pseudo-code), no standards, etc. However, despite this rationalizing tendency, there are reasons to believe that in many cases the root of the problem was in communications' procedures and policies pitfalls.

Communication issues cannot only enlarge a small incident, but if dealt with properly, can also effectively solve a big one. Thus, treating communication as a main tool for supplier management will help in solving problems with outsourcers, especially in the area of supplier's quality of service expectations.

Agile methodologies (e.g. SCRUM, Dynamic Systems Development Method (DSDM), Extreme Programming (XP), etc.) emphasise utilization of suitable prototypes [27, 28] and effective mapping of user stories [27, 29] to improve

communications of centralized or distributed development teams and users, and to lower development risks.

In most scenarios, in order to solve a complex problem it is a good idea to decompose the whole problem into smaller, better manageable, and understandable parts [9, 29, 30]. This means both simplifying and diversifying the views of the existing problem. We should limit our analysis of programming services outsourcers as manufacturers of these parts.

We also believe that a *holistic view of the problem* needs to be introduced in order to change the predominant focus on the technical side of each single instance of the problem class (that are normally hard to track [31, 32, 33]) to the more statistical and processes sides of the problem as a whole (i.e. a view of the problem class conducted in parallel with a view of a single problem instance).

Thus, a holistic view of the project set-up and maintenance process set-up with suppliers can be a prerequisite for further analysis. The main goal would be to build mutual trust and improve the quality of overall services. This also means moving outstanding issues (errors/bugs) back to predefined and mutually agreed limits of the statistical control.

In addition, we propose that with continuous cooperation on improving predefined KPIs (Key Performance Indexes), the agreed limits of statistical control could also become a target for improvements. This plan could become a common part of contracts, with the idea of rewarding high quality in the interest of both parties (i.e. introduction of incentive types of contracts rather than time and material or fixed term). Thus, software problems would be first transferred into the relationship and provider management domains for a solution, and then, once solved in the soft (people) problem area, transferred back to the hard (technology) problem area – the program code.

This holistic approach does not substitute inner software development process, regardless of the type: agile (SCRUM, DSDM, etc.) or procedural (e.g. Rational Unified Process) for whatever reasons they were chosen for the particular software development. It is a better tool to manage new developments based on the opportunity designed in a systematic, methodological way to learn from production behaviour of software and humans that use them in the particular environment. It is more like an attempt to prescribe a particular patient continually improved medicines based on the data collected on his reactions over time on different medicines (and their ingredients) for the particular disease.

We have selected the 6 Step Service Improvement methodology [7] as a method that could support systematic and transparent improvements in suppliers' services.

The main characteristics of the method are:

- The result of method utilization is an improvement in the overall quality of the software service.

- It does not pre-estimate services without measuring their outcomes first. This means that nobody is the best supplier by definition. Measurements taken during selected real time production period cycles (e.g.. month, quarter, half year, and year) in the given environment will give better answers than forecasts based only on past experiences in different environments.
- The method uses a ranking principle for services groupings in order to provide meaningful comparisons among similar type of services within the same rank (apples to apples).
- The method uses LSP (Logical Scoring of Preferences) for comparison purposes among predefined elementary criteria.
- The method is conducted in improvement cycles until it makes sense to continue to another cycle (cost/benefit sense) to avoid "gold plating" scenarios. Kaizen practice is good thing to do, but it has a cost side to be addressed as well [31].
- Learning about services is continuous, and the view of the services is holistic. This means that the method supports a continuous improvements paradigm and embraces uniqueness of each service, as well as their shared characteristics. Each service is described by the dimensions of soft (people interactions) and hard (technical, i.e. code that works) elements. These elements are interconnected in multidimensional services' cubes. Recognition, based on measurements and comparison, of what makes some elements of one cube better than another helps in improving the other service and vice-versa.

We propose the use of the method for multidimensional comparison of preferences in spiral cycles in repeatable time cycles that reflect the particular *business pulse* (dynamics of the business changes). In some cases where, for example, the business pulse is at the *elephant* level (25-35 bpm resting), the time cycle periods will be longer (i.e. quarterly or semi-annually); on the contrary, for a business with the *mouse* pulse level (450-750 bpm resting), weekly cycle periods might be the right choice.

Seasonality effects in each business would need to be taken in account; thus some *arrhythmic* cycles could be desirable as well (i.e. monthly cycles normally, and during the summer only one quarterly cycle).

4 Case Study Experiment and Results

The 3SI method starts with identification of the SW services that can be classified within the same rank. As mentioned above, the quality of software services is strongly influenced by service providers' characteristics. In order to compare

service providers' characteristics we first rank them based on the *flavour* of their software services domains (i.e. programming language, system architecture environment, team structure, etc.), and then drill into further hierarchical decompositions until we reach elementary criteria [34]. The formula to calculate the estimates of each defined criteria [34, 35] is given below (1):

$$\mathrm{E} = \left(\sum_{i=1}^{k} \omega_i \; e_i^r\right)^{1/r}, 0 \leq \omega \leq 1, \; \sum_{i=1}^{k} \omega = 1,$$

$$e \in < 0; 1 >, \; \mathrm{E} \in < 0; 1 >, \; k \geq 2 \qquad (1)$$

where the coefficient "ω" represents the weight coefficient associated with the comparative importance of each estimated elementary preference belonging to the same hierarchical group preference, and the "e" represents an elementary preference estimate. The "k" represents the number of features in the aggregation blocks. The "r" represents the correlation function to be applied on the specific level. The values of "r" are defined on the basis of the expectation of the combined influence of the estimated preference at the group level (e.g., synergy effects). The values for "r" vary from full conjunction (C, r = - ∞) to full disjunction (D, r = + ∞). The arithmetic mean (AM) is given at r = 1. More details about the mathematical aspects of the LSP method can be found in [36]. The strength of LSP resides in the power to model different logical relationships [34]:

- Simultaneity: when it is perceived that two or more input preferences must be present simultaneously,
- Replaceability: when it is perceived that two or more attributes can be replaced (there exist alternatives, i.e., the low quality of an input preference can always be compensated by the high quality of some other input),
- Neutrality: when it is perceived that two or more input preferences can be grouped independently (neither conjunctive nor disjunctive relationship),
- Symmetric relationships: when it is perceived that two or more input preferences affect evaluation in the same logical way (though possibly with different weights),
- Asymmetric relationships: when mandatory attributes are combined with desirable or optional ones.

The service grouping, as the first step of identification, was done based on the identified service class's group attributes [7]:

- Technology group (TDi) - represents the technical attributes that better describe the influence of applied technology tools on service development and operations.
- Complexity group (Ci) - represents the observed level of complexity in creating a solution. More tiers in the solution implementation in most cases represent more complexity in operating that service.

- Development process group (DPi) - represents the possibility to lever influence on the service by an applied development process. Some development processes could create a very stable service, but have a problem with the low level of flexibility to change.
- Development team group (DTi) - team experiences, skills, and cohesion, and in-house and outsourced options affect the ability for quality maintenance on a specific service.
- Business support domain group (BDi) - related to the end user profile, the number, location, and type of application being used (e.g. OLTP, reports, etc.).

Based on the above definitions of group attributes, each instance of service class S_i from the catalogue was assigned values as following (2):

$$S_i = (TD_i, C_i, DP_i, DT_i, BD_i), i \geq 0 \quad (2)$$

We analysed the use case in which there are more teams (in-house and outsourced teams) working on the same software and hardware platform, but on different projects. Thus, these projects are *environmentally* similar, but size and complexity vary depending on the needs of the end users. The end user stakeholders are employed with the same company. Internal variations within the internal teams were minimal.

The observed company uses services from four Java outsourcing providers. The decision to use four outsourcing companies for the same area of expertise came from the need to better manage the risks of outsourcing. The company has gradually moved its strategy towards Java outsourcing from pure in-house development. The reason for that strategic switch was an increased need for faster and better solutions from one side, and the limited resources in highly competitive Java programmers market from the other (unfavourable supply/demand ratios).

Managing different outsourcing companies in supporting the same or similar business area is not trivial. Differences could be in (but not limited to):

- internal development processes specifics (e.g. agile, procedural, mixture),
- design and coding standards,
- IDE (Integrated Development Environments) tools utilised,
- Version Control standards,
- project management standards,
- culture.

The company has a defined set of internal coding standards to be applied with all outsourcers, and it has also adopted standards for version control and reporting on the work assigned progress. The price of each man-day for the same type of work

(Java programming) has been negotiated to the same level for all service providers. These decisions have made it possible to concentrate on more objective KPI measurements for each code supplier.

To have a better understanding of the company's working environment, some historic facts need to be addressed as well. The company's IT department has gradually moved from pure insourcing to outsourcing of its Java programming activities (Figure 1). The main reason was an increased need for Java programming jobs without a favourable supply of Java programmers in the local market for prospective employers. On average, the retention rate was less than 3 years.

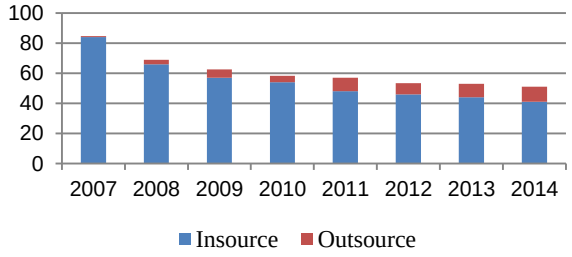

Figure 1
An example of the historical trend on insourcing vs. outsourcing FTE consumptions

However, the main issue with selecting the right balance on insourcing vs. outsourcing strategy was the tactics for managing the quality of outsourcing activities. If there are more code suppliers, some kind of multi-dimensional comparison needs to be applied that takes into account the following criteria:

- static criteria (project/activity quality of scope delivery, SPI, CPI, etc.),
- dynamic criteria (service life cycle experiences with outsourcers) including:
  - hard (number of bugs reported in period, architecture issues reported in period, costs in period, etc.),
  - soft (team relationships) criteria.

In some cases cutting off one code supplier is not feasible for different reasons (e.g. legal, no good replacements on the market, etc.). However, it is possible to learn from good experiences with different outsourcers and different services. Ideally, new knowledge on how to improve certain service parameters should be passed to that service in real time. Services should be improved continuously, ideally without any time breaks.

The suppliers reported the consumption of their Java development resources continuously for a period of one year as shown below [Table 1].

Table 1

SW suppliers efforts (in man-days) invested during observed year

Supplier	Effort (m-d)
A	458
B	724
C	307
D	263

Our primary goal was to compare the quality of services within the same service class. C_i attribute values could have significant variation depending on the type of users' requirements (e.g. functional scope complexities, non-functional technical complexities, etc.), but the other environment dependent parameter variations could be kept stable for the observed time period for the preferences calculations (i.e. $\pm$ 5%, or within the predefined result threshold).

Thus, these constants are calculated (3, 5, 7, 9, and 11):

$TD_i = (td_1, td_2, td_3)$, where $td_1 \epsilon \{2T, 3T, 4T\}$,

$td_2 \epsilon \{WO, WP, DC\}$, and $td_3 \epsilon \{J, VB, C, D\}$ (3)

Here, 2T stands for Two-Tier, 3T for Three-Tier, and 4T for Four-Tier application architecture; WO for Web/Open Source, WP for Web/Proprietary, and DC for Client (Fat Client) presentation layer; J for Java, VB for Visual Basic, C for C++, and D for DotNet programming language.

In this use case the values of TD_i domain are kept as: three-tier, Open Web platform, and Java programing language (expressions 3 and 4).

$TD_i = (3T, WO, J)$ (4)

$DP_i = (dp)$, where $dp \epsilon \{S, R, A, H\}$ (5)

Here, S stands for SSA (Structural System Analysis), R for RUP, A for Agility, and H for Hybrid.

In this use case the value of DP_i domain is hybrid 5 and 6).

$DP_i = (H)$ (6)

$$DT_i = (dt), \text{ where } dt \in \{IH, OH, MX\} \quad (7)$$

Here, IH stands for In-House, OH for Out-House, MX for Mixed.

In this use case the value of DT_i domain is mixed (7 and 8).

$$DT_i = (MX) \quad (8)$$

$$BD_i = (bd_1, bd_2, bd_3), \text{ where } bd_1 \in \{FE, BE\},\ bd_2 \in \{OL, RE\},\ bd_3 \in \{IN, EX\} \quad (9)$$

Here, FE stands for the Front-End and BE for the Back-End parts of the system; OL for OLTP and RE for Reports; IN for Internal users and EX for External users made service.

In this use case the values of BD_i domain are kept as follows: front end development (FE), OLTP support programs (OL), and internal users of the core Java-based system (IN) (9 and 10).

$$BD_i = (FE, OL, IN) \quad (10)$$

The complexity group attribute has had significant variation during the observed time period of one year. We need to stress that longer time periods for comparison would increase the risk of other fixed attributes varying. However, a shorter observation time period might not give proper results because some suppliers might have a better learning curve, but could later show a lack of service quality. We propose continuous measurement and immediate internal reporting of poor performance. However, we also stress that actions be taken wisely – only after having a *proper* amount of data during a *proper* amount of time (i.e. once per week/month/quarter, regarding the type of the company, the kind of ecosystems, the amount of concrete work, the acceptable pace of development, etc.).

We classify the complexity C_i of each a_i activity in three levels:

$$C_i = (l), \text{ where } l \in \{1,2,3\} \quad (11)$$

The semantics of our classification is as follows:

1) Level 1 (C_i = 1) – low range: from minor changes to the existing code without changes to the data model or component architecture, to medium changes to the existing code that may include data model modifications, but not architecture changes.
2) Level 2 (C_i = 2) – medium range: from major changes to the existing code that may include data model modifications and architecture changes, to new application development without significant changes to the surrounding system interconnections.
3) Level 3 (C_i = 3) – high range: new system development that may include a number of interconnected applications and significant changes in surrounding system interconnections and replacements.

Since $TD_i, DP_i, DT_i,$ and BD_i values are fixed for this use case, the services activities outcomes were categorized into three service class ranks depending on the complexity of these activities (12).

$$S_i = ((3T, WO, J), C_i, H, MX, (FE, OL, IN)),\ i = 1, 2, 3 \quad (12)$$

The service class attributes value assignments are given for each service class instance, and grouped into predefined services' class ranks (12). All the services in the same service group were then compared.

The selected comparison criteria are based on the hierarchical decomposition of preferences, which operates until the elementary criteria have been reached. In order to compare the quality of the services (the programming services) within a specific rank, we have used the following first, second, and third levels of the hierarchical decomposition of the preferences (starting from global preferences that were recognized at the first level of hierarchical decomposition [Table 2]):

Table 2

Software service preferences hierarchical decomposition

P1 QoS (Quality of Solution/Service)
P11 Maintainability
P121 Changeability
P122 Stability
P123 Testability
P12 Documentation
P13 Performance
P131 Processing time
P132 Throughput
P133 Resource consumption
P14 Reliability
P141 Maturity
P142 Fault tolerance
P143 Recoverability
P15 Usability
P151 Understandability
P152 "Learnability"
P153 Operability
P16 Capability
P17 Installability
P18 Availability
P2 CoS (Cost of Solution/Service)
P21 Fixed Costs
P211 Programming
P212 Licenses

P213 Infrastructure
P22 Variable Costs
P221 Unplanned Development
P222 Unplanned Resources Availability
P223 Travel/Accommodation
P224 Interest Rates
P3 Risks
P31 Strategic
P311 Regulation
P312 Market Position
P313 Shareholders
P32 Operational
P321 Time to market
P322 Unplanned costs
P323 Stakeholders
P323 Retention
P4 Likeability

The group of experts, consisting of IT personnel, PMO, and user representatives (with our support), had been given the task of assigning all weight coefficients and each level correlation logical functions, and providing estimates for each preference during this study in each monthly cycle.

During the first, kick-off, workshop meeting, we defined together (as a team) all the weight coefficients and related logical functions for each hierarchical group level based on our understanding of the importance of each preference estimate and correlation with other preferences within the same group. Each of us gave his/her opinion and we discussed all individual views to come up with the common framework to be used for the measurements and comparisons. We also set a time table for meetings on a monthly basis to discuss the collected production data, to make another 3SI run, and to create an action plan for improvements.

The weight coefficients for the first hierarchical level in this use case are:

$$\omega_1 = 0.4,\ \omega_2 = \omega_3 = 0.25,\ \omega_4 = 0.1 \tag{13}$$

We found that the main features (criteria) of this level of estimation are strongly dependent on each other because higher QoS will, in most cases, produce lower CoS [11], and lower the risks. That is the reason the team gave 40% weight to QoS. We also found that soft features are normally higher with better QoS. This is the reason the r function used in calculation at this level (r_0) is a type of weak conjunction (C-+) (*medium week conjunction* as defined in [34]).

We introduced *Likeability* criteria without purposefully drilling in further (this could be left for further research) to stress the importance that soft criteria be

considered at the highest level of hierarchical decomposition. Based on the data provided, we believe that the core of each successful project or change activity was good communication management. Better communication management gave better results on "Likability" criteria.

We used the following values for weight coefficients at the second hierarchical level (14):

$$\omega_{11} = \omega_{12} = 0.20,\ \omega_{13} = \omega_{14} = 0.15,\ \omega_{15} = \omega_{16} = 0.10,\ \omega_{17} = \omega_{18} = 0.05;$$
$$\omega_{21} = \omega_{22} = 0.50;\ \omega_{31} = 0.70,\ \omega_{32} = 0.30 \quad (14)$$

We used the following values for weight coefficients at the third hierarchical level (15):

$$\omega_{111} = \omega_{112} = 0.35,\ \omega_{113} = 0.30;\ \omega_{131} = \omega_{133} = 0.35,\ \omega_{132} = 0.30;$$
$$\omega_{141} = \omega_{142} = 0.30,\ \omega_{143} = 0.40;\ \omega_{151} = \omega_{152} = 0.20,\ \omega_{153} = 0.60;$$
$$\omega_{211} = 45,\ \omega_{212} = 0.20,\ \omega_{213} = 0.35;$$
$$\omega_{221} = 40,\ \omega_{222} = 0.20,\ \omega_{223} = 0.20,\ \omega_{224} = 0.20;$$
$$\omega_{311} = \omega_{312} = 0.30,\ \omega_{313} = 0.40;$$
$$\omega_{321} = \omega_{323} = 0.30,\ \omega_{322} = \omega_{324} = 0.20 \quad (15)$$

All the above values are discussed among the team members, and are the results of the mutual agreement on importance of each preference within the specific hierarchical group level. The level of precision is influenced by the level of experiences and knowledge of the team. That is the reason why we selected multifunctional team with different experiences in IT projects management, software developments (both from internal and external software services providers) and users of the software systems.

The r function was calculated according to (16). Please note that the value of r is given together with the description of the type of the logical function (i.e. for r_0 we used C-+ function because we wanted to achieve significant level of good estimates for each group criteria at the first hiearchical level), and number of the grouping elements for which it was calculated (i=4 means that we have 4 elements in the group).

$r_0 = -0.235$ (C -+, i=4); $r_1 = r_2 = 1$ (A);

$r_3 = -0.148$ (C -+, i=2);

$r_{11} = r_{13} = r_{14} = r_{15} = 0.573$ (C--, i=3); $r_{21} = r_{22} = 1$ (A);

$r_{31} = -0.208$ (C -+, i=3); $r_{32} = -0.235$ (C -+, i=4). (16)

Please note that the conjunction function was used to stress the need to have good results on all estimations at the same group level.

Since code suppliers normally worked on projects and maintenance activities, we collected data concerning both types of activities. Projects were done in accordance with the predefined project methodology (in this case it was PMI) [37, 38]; all other functional, performance, and bug fixing related changes were considered as maintenance activities and conducted through regular change management procedures. In order to reduce paper work and become more agile, the company only ran as a project those code changes requiring over 15 man-days of work or over 3K Euro in development investment.

Programming activity could be related to non-value adding activities, such as bug fixes (b_i) performed on existing code and new code development (n_i). Regardless of the type of activity programming cost (C) always exists (17).

$$C = \sum_{i=1}^{k} f_n(n_i) + \sum_{i=1}^{l} f_b(b_i),$$

$$\text{where } k \geq 1,\ l \geq 1, \tag{17}$$

The consequences of increased costs due to lower code quality were considered by one or by more sides (directly or indirectly), depending on the contract type between the parties involved. In a time and material type of contract, the sponsor would usually pay for both (r_i and n_i). On the contrary, with fixed term contracts the sponsor, by definition, pays only what was calculated as the amount of work for n_i. A maintenance contract normally covers production issues and new functionalities development. In this case, the sponsor would not be fully aware of the real quality of the deployed code before running deep in the production environment.

Result: By applying 3SI methodology we found that the preserved quality of results and the quality of built mutual relationships among insource and outsource teams on software development activities were correlated (i.e. where the Likability estimate was higher other estimates tended to be higher as well). The figures below (Figure 2 and Figure 3) show the results of team estimates and calculations for the first reporting period (the first month's estimates/the first iteration).

The results show that in this use case overall estimates based on the complexity of the programming activities have not shown any significant difference. All companies gave results that are lower, by up to 5%, as the complexity level grows.

The results show that the lowest estimated difference from the best to the worst supplier in the same complexity class was around 35% (Figure 2 and Figure 3, the difference between provider B and provider D). This significant difference stresses the need for further analysis to discover the root cause of these estimates.

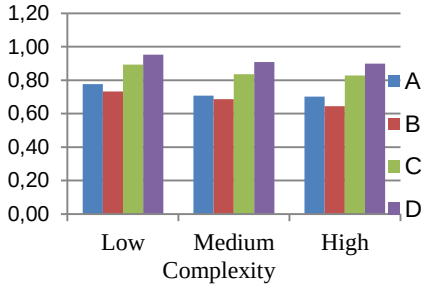

Figure 2
Maintenance activities estimates comparison results per class, per supplier

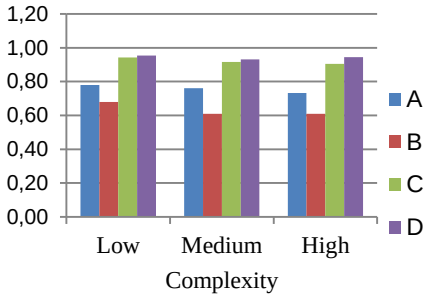

Figure 3
Projects activities estimates comparison results per class, per supplier

In the next step, we identified the major reasons for the differences in the services provided by different suppliers. We started from the most important difference contributor and continued with other major ones (we did not include all reasons in order to avoid gold plating: i.e. to find those 20% of reasons that contribute to 80% of the difference) [26]. The main driver responsible for lowering future costs based on past service performance in dynamic business environments is to have quick (on time) reactions. "What makes measurements so potent is its capacity to instigate informed action – to provide the opportunity for people to engage in the right behaviour at the right time" [39]. After a predefined cycle period of one month, another 3SI cycle was initiated (monthly cycles). For the purpose of this experiment, we repeated these cycles for the period of six months (six iterations). Figure 4 presents the result for six iterations in graphic form.

The cycle periods of 3SI need to be based on the specifics of the business context and content of the software service. We proposed planning time after these periodical assessments to talk to each service provider about the estimates in order to trigger further improvements. In some cases it may work well to have frequent measurements analysis (i.e. on a weekly or monthly basis), but in some cases it

would be advisable to wait for a whole quarter to pass to have a meaningful assessment of performance. In our experiment, after each cycle's data collection and comparison, we scheduled separate interviews with each of the suppliers, during which we presented the findings and asked for feedback. The main result we noticed was *significant improvement in the services of all providers* over time (Figure 4); however, we also noticed/proved that one provider (provider D, Figure 4) had to be replaced (this decision was also based on the measurements, which helped in conducting fair/objective service closure).

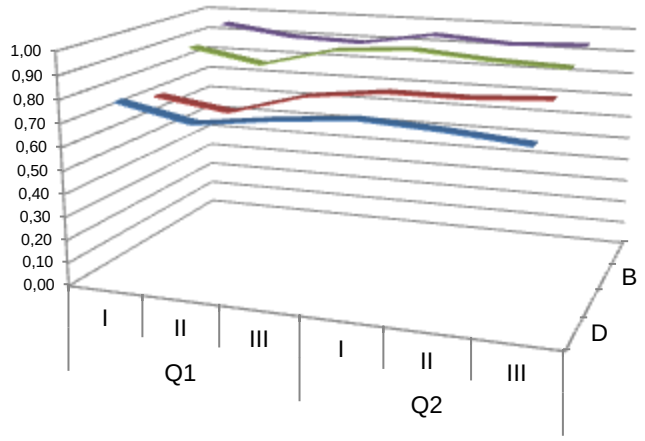

Figure 4
3SI method implementation results for the use case over the given period of time

The cycle of assessments in 3SI could also vary due to the maturity of the business partnership. We suggest starting partnership performance assessments where the learning curve offsets calculation at the beginning (i.e. tolerate up to 10% lower marks due to a learning curve's offsets in the first reporting data sample), and then to continue without offsets into regular comparative analysis and continuous cycles' improvements.

Conclusion

The pressure to give better, cheaper, and faster results in software development is getting stronger as business becomes more and more competitive. A small difference in performance can mean a significant bottom line difference for the company. The fight for knowledge resources does not recognize boundaries. Emerging digital technologies will bring even more stress to the local/domestic office building.

Nowadays, companies rely on some mixture of insourcing and outsourcing for software development activities. In order to manage the risks of using only one outsourcer many companies use two or more outsourcers for the same systems domain. This is mandatory if that domain supports the core company business.

The software services given by code suppliers are devoted to software development projects and/or to maintenance activities on existing software. All these activities represent costs for the company and for the supplier. The goal is to achieve a win-win situation and evenly share the risks regardless of the activity type. This goal, even with detailed contracts in place, is very hard to achieve in practice.

In this paper, we have shown that by using the 3SI methodology it is possible to create estimates for preferences satisfaction for each supplier and to compare them regularly to find a reason why in some areas one supplier may be better than another and vice-versa. The utilization of that knowledge in a regular, systematic way could support the continuous improvement of all services at the same rank.

This case study experiment has been conducted within the live production environment over period of six months, involving a number of people with limited abilities (i.e. with average knowledge and available time that can be taken from regular operational activities). We did our best to optimize utilization of their time, capacity, and availability to simulate a real operational situation. We concentrated our activities on running 3SI in repeating cycles and providing value in improving overall quality of the software services in this specific environment as a main priority.

The fact that comparisons are conducted in periodical cycles could be used to enforce closer communication between parties that could lead to increased quality of the overall services. In some cases, it might lead to calls for contractual expectations that were not seen before or sometimes to even end the partnership. But, this is not the primary goal; this would be an extreme consequence of the new knowledge acquired. The primary goal is to influence and change (if necessary) the parts of the development process that caused an increased number of defects and lower quality of the code.

Since making software is a creative, rather than purely technical, activity with strong human and team interaction, an example from the case study has shown the importance of soft skills and demonstrated certain correlations between Likability criteria and other more quantitative metrics. However, we have also learned that *Likability* estimates can also be improved with the right actions taken.

Further research and implementation: We recommend trying other methodologies (e.g. Squale model) within the same environment and compare the results and the feedbacks from the end-users. In this case, it is absolutely necessary to assess the ratio between costs and achieved benefits in order to reach the primary goal of this paper which is a sufficient gain for obtaining a reasonable price and all this under real constraints primarily related to skills and available time of involved experts.

The utilization of this method in industry can be supported by the creation of a parameterized software solution, which we see as a next practical step. We also envision the need for further research on data analytics based on generated

knowledge base that will support automations in the continual quality improvements (i.e. design of different 3SI *templates* for different industrials' needs).

Further research in soft skills and communications improvements specifically during procurement/selection and suppliers' quality management processes on the complex software projects realizations is also desirable.

References

[1] Boehm B.: Value-based Software Engineering - Reinventing, ACM SIGSOFT Software Engineering Notes, 2003, Vol. 28, No. 2

[2] Gonzalez R., Gasco J., and Liopis J.: Information System Outsourcing: A Literature Analysis, Information & Management, 2006, Vol. 43, No. 7, 821-834

[3] Peppard J. and Ward J.: The Strategic Management of Information System: Building a Digital Strategy, 4th Ed., Wiley, 2016

[4] Chun M. and Mooney J.: "CIO Roles and Responsibilities: Twenty-Five Years of Evolution and Change," Information & Management, 2009, Vol. 46, No. 6, 323-334

[5] Addo-Tenkorang R. and Helo P.: Enterprise Resource Planning (ERP): A Review Literature Report, Proceedings of the World Congress on Engineering and Computer Science 2011 Vol II WCECS 2011, San Francisco, USA October 19-21, 2011

[6] Chin-Sheng C. and Wen-Yau L. M.: A Cloud Computing Platform for ERP Applications, Information & Management, 2015, Vol. 27, No. 7, 127-136

[7] Markovic V., and Maksimovic R.: A Contribution to Continuous Software Services Improvement Based on Six Step Service Improvement Method, IJESEKE, 2012, Vol. 22, No. 4, 1-21

[8] Kan S. E.: Metrics and Models in Software Quality Engineering, 2nd ed., Addison-Wesley, 2003

[9] Betz C.: Architecture & Patterns for IT, Elsevier Inc., 2011

[10] Jones C.: Critical Problems in Software Measurement, Burlington, Mass: Software Productivity Research, 1992

[11] Kuvaja P., Similä J., Krzanik L., Bicego A., Saukkonen S. and Koch G.: Software Process Assessment & Improvement - The Bootstrap Approach, Oxford: Blackwell Publishers, 1994

[12] Pulford K., Kuntzmann-Combelles A. and Shirlaw S.: A Quantitative Approach to Software Management, The AMI Handbook, Wokingham, England: Addison-Wesley, 1996

[13] Karjalainen J., Mäkäräinen M., Komi-Sirviö S., and Seppänen V.: Practical process improvement for embedded real-time software, Quality Engineering, 1996, Vol. 8, No. 4, 565-573

[14] Boehm B. W., Abts C., Brown A. W., Chulani S., Clark B. K., Horowitz E., Madachy R., Reifer D., and Steece B.: Software Costs Estimation with COCOMO II, NJ: Prentice Hall, 2000

[15] Basili V., Daskalntonakis M., and Yacobellis R.: Technology Transfer at Motorola, IEEE Software, 1994, 70-76

[16] Basili V. and F. McGarry.: The Experience Factory: How to Build and Run One, Tutorial TF01, 20th International Conference on Software Engineering (ICSE'98), Kyoto, 1998

[17] Deming E.: Out of the Crisis: Quality, Productivity and Competitive Position, Cambridge University Press, Cambridge, 1982 & 1986

[18] Shewhart W. A., and Deming E. W.: Statistical Method from the Viewpoint of Quality Control, Dover Publications, 1986

[19] Hackman J. R., and Wageman R.: Total Quality Management: Empirical, Conceptual, and Practical Issues, Administrative Science Quarterly, 1995, Vol. 40, No. 2, 309-342

[20] Dean J. W., and Bowen D. E.: Management Theory and Total Quality: Improving Research and Practice through Theory Development, Academy of Management Review, 1994, Vol. 19, No 3, 392-418

[21] Choppin J.: Total Quality Management, What isn't It?, Managing Service Quality, 1995, Vol. 5, No., 147-49

[22] Bergel A., Denier S., Ducasse S., Laval J., Bellinguard F., Vaillergues P., Balmas F., and Mordal-Manet K.,: Squale - software quality enhancement, In Proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR 2009), European Projects Track, 2009

[23] Juran J.:Total Quality Management, McGraw-Hill, 2001

[24] Boyd R.: Understanding ITIL Key Process Relationships, Computer Economics, 2007, 1-3

[25] Chrissis M. B., Konrad M., and Shrum S.: CMMI, Guidelines for Process Integration and Process Improvement, Addison-Wesley, 2004

[26] Dion R.: Process Improvement and the Corporate Balance Sheet, IEEE Software, 1993, 28-35

[29] Patton J, Economy P.,: User Story Mapping, O'Reilly Media, 2014

[27] Ciriello R., Richter A., Schwabe G.,: When prototyping meets storytelling: practices and malpractices in innovating software firms, ICSE-SEIP '17 Proceedings of the 39th International Conference on Software Engineering:

Software Engineering in Practice Track, Buenos Aires, Argentina - May 20-28, 2017

[28] Blomkvist J. K., Persson J., and Aberg J.,: Communication through Boundary Objects in Distributed Agile Teams, Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - Chicago 2015, pp. 1875-1884, 2015

[30] Wysocki R. K.: Effective Project Management, 4th ed., Wiley, 2006

[31] Imai M.: Gemba Kaizen: A Commonsense, Low-Cost Approach to Management, McGraw-Hill, 1997

[32] Zsidisin G. A., Jun M., and Adams L. L.: Relationship between Information Technology and Service Quality in the Dual - The Direction Supply Chain: A Case Study Approach, International Journal of Service Industry Management, 2000, Vol. 11, No. 4, 312-328

[33] Salle M.: IT Service Management and IT Governance: Review, Comparative Analysis and Their Impact on Utility Computing, Hewlett-Packard Company, 2004

[34] Dujmovic J. J., and Nagashima H.: LSP method and its use for evaluation of Java IDEs, International Journal of Approximate Reasoning, 2006, Vol. 41, No 1, 3-22

[35] Dujmovic J. J., and Bai H.: Evaluation and Comparison of Search Engines using the LSP Method, ComSIS, 2006, Vol. 3, No. 2, 31-56

[36] Dujmović, J. J.: Continuous Preference Logic for System Evaluation, proceedings of Eurofuse 2005, edited by B. De Baets, J. Fodor, and D. Radojević, pp. 56-80, ISBN 86-7172-022-5, Institute "MihajloPupin", Belgrade, 2005

[37] Culver-Lozo K.: Software Process Iteration on Large Projects: Challenges, Strategies and Experiences, Software Process - Improvement and Practice, 1995, No. 1, 35-45

[38] PMI: Project Management Body of Knowledge (PMBOK Guide), 5th ed., PMI, 2013

[39] Spitzer D. R.:Transforming Performance Measurement, AMACOM, 2007