ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
UDC/UDK 004.413/.414

CRITICAL MEASURES OF SUCCESS FOR A SOFTWARE PROJECT

Ninoslav Slavek, Damir Blažević, Krešimir Nenadić

Professional paper
The work presented here attempts to identify a set of software project performance measures and influence factors used by software development projects so that a valid comparison of performance can be made between completed projects. The performance measures identified in this document are core measures that are identified as a part of the set of critical measures of success since they address important attributes of a software development project. For forty small and medium sized software projects the performance measures and the influence factors were measured.

Keywords: *influence factors, performance measurement, software engineering standard, software project quality*

Kritična mjerila uspješnosti za softverski projekt

Stručni članak
Prikazani rad pokušava utvrditi skup mjerila performanci i faktora utjecaja za razvojne softverske projekte, tako da se mogu učiniti valjane usporedbe performanci između dovršenih projekata. Mjerila performanci identificirana u ovom radu su ključna mjerila koja su identificirana kao dio skupine kritičnih mjerila uspješnosti, s obzirom da ona adresiraju važne atribute razvojnog softverskog projekta. Za četrdeset malih i srednje velikih softverskih projekata izmjerena su mjerila performanci i faktori utjecaja.

Ključne riječi: *faktori utjecaja, kvaliteta softverskog projekta, mjerenje performance, norme softverskog inženjerstva*

1 Introduction

The Competence Centre for Software Engineering (CCSE) was established in Osijek, Croatia in 2006, intended for development of logistics and information technology in the wider region promoting software quality, reliability and diagnosis. Objectives of the CCSE are improving cooperation between universities and the economy through public-private partnerships, joint research and development, training, practical work in firms, and conducting other activities useful for partners.

The CCSE publishes periodic reports to show what is measured, analyzed and decided in order to improve software performance. An initiative from the CCSE is the implementation of a program for software project performance measurement.

The performance measurement is a process of assessing the results of a company, organization, project, or individual: a) to determine how effective the operations are, and b) to make change by addressing performance gaps, shortfalls, and other issues.

Important purpose of implementing the program of performance measurement is to support the improvement of engineering.

Other reasons for implementing the program are:

- Planning and estimation – the historical measurement data can be used as a basis to forecast or estimate future performance. The performance measurement is a long term planning tool that can justify resource allocation for software projects.
- Goal achievement – the performance measurement provides feedback about whether or not organization is meeting its business or project goals. This feedback improves the likelihood of achieving these goals efficiently.
- Communication – the reporting of well-defined performance measurement can enhance staff, stakeholders, and partner understanding and support of strategies and decisions.
- Improvement – the performance data can be compared within and outside an enterprise to identify weak areas that can be addressed to improve overall performance.

Companies and organizations measure their performance in a variety of areas using different methods and criteria for different purposes. The measures to be compared must be commonly defined. We used a list of performance measures according to Software Engineering Institute's Technical report [7] in order to: (1) define a small set of key performance measures that should be used by all software projects, and (2) define the factors for these measures. The list of performance measures is presented in Tab. 1, and the list of influence factors is presented in Tab. 2.

Theoretical considerations have been applied to a set of software projects that have been assessed and reported in this paper. All these projects have been developed in accordance with the ESA (European Space Agency) software engineering standard, PSS 05 [9]. Each project team included: project manager, QA manager, requirement analyst (I, II, III), project developer (I, II, III, IV, V), project developer-tester (I, II, III), and project librarian. The number of analysts, developers, and testers varies in different projects. All projects were under the supervising control of the CCSE team which collected data of the implemented project performance measures and influence factors. Only small and medium sized software projects are assessed in this survey. According to ESA PSS 05 standard, two-man-years or less is defined as a small project, twenty-man-years or more is a large project. For each project all software life cycle documentation was produced until the coding phase.

Eighteen software development companies developed forty various types software projects for twenty-five various companies (customers) in the region in two years.

All projects are developed using the software life cycle model that includes the basic phases that are presented (level 1 documents):

- UR phase - Definition of the user requirements
- SR phase - Definition of the software requirements
- AD phase - Definition of the architectural design
- DD phase - Detailed design and production of the code
- TR phase - Transfer of the software to operations
- OM phase - Operations and maintenance.

The related (level 2) documents are:

- ESA PSS-05-08 Guide to Software Project Management
- ESA PSS-05-09 Guide to Software Configuration Management
- ESA PSS-05-10 Guide to Software Verification and Validation
- ESA PSS-05-11 Guide to Software Quality Assurance.

There are six major milestones that mark progress in the software life cycle. These milestones are the:

- approval of the User Requirements Document (URD)
- approval of the Software Requirements Document (SRD)
- approval of the Architectural Design Document (ADD)
- approval of the Detailed Design Document (DDD), the Software User Manual (SUM), the code, and the statement of readiness for provisional acceptance testing;
- statement of provisional acceptance and the delivery of the Software Transfer Document (STD)
- statement of final acceptance and the delivery of the Project History Document (PHD).

Table 1 Timetable

Project team	Project start							Project end	
	December	November	January	Feb.	March	April	May	June	m/m
Manager Škrlin	1	0,8	0,3	0	0	0	0	0,5	2,6
QA manager Milčić	0,8	1	1	0	0	0	0	0,5	3,3
Project developer Vlahović	0,8	1	1	0	0	0	0	0,5	3,3
Project developer Devčić	0,8	1	1	0	0	0	0	0,5	3,3
Project prog. Cindrić	0	0	0	1	1	1	0,5	0	3,5
Project prog. Stipanović	0	0	0	1	1	1	0,5	0	3,5
Project prog. Fabijanić	0	0	0	1	1	1	0,5	0	3,5
Project dev.-tester Karakaš	0	0	0	1	1	1	0,5	0,5	4
Project dev.-tester Eškerica	0	0	0	1	1	1	0,5	0,5	4
Project dev.-test. Marković	0	0	0	1	1	1	0,5	0,5	4
Project librarian Karajić	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	2,4

Table 2 Costs

Project team	Asset / HRK
Manager Škrlin	104 000,00
QA manager Milčić	99 000,00
Project developer Vlahović	99 000,00
Project developer Devčić	99 000,00
Project prog. Cindrić	70 000,00
Project prog. Stipanović	70 000,00
Project prog. Fabijanić	70 000,00
Project dev.-tester Karakaš	80 000,00
Project dev.-tester Eškerica	80 000,00
Project dev.-test. Marković	80 000,00
Project librarian Karajić	36 000,00
Total:	887 000,00

Table 3 Material expenses

Material expense	HRK
New equipment	138 000,00
Licence software	100 000,00
Settings 8 months	100 000,00
Rental 8 months	100 000,00
Another expenses	50 000,00
Total:	488 000,00

The last milestone does not fall at the end of a phase, but at the end of the warranty period.

Each project was developed according to the 'waterfall' approach, as the interpretation of the model. The phases are executed sequentially, each phase is executed once, although iteration of part of a phase is allowed for error correction. Delivery of the complete system occurs at a single milestone at the end of the TR phase. The approach allows the contractual relationship to be simple. A life cycle approach, based upon this model was defined, for each project, in the Software Project Management Plan. Project management software covering many types of software, including scheduling, cost control and budget management, resource allocation, collaboration software, communication, quality management and documentation or administration systems, which are used to deal with the complexity of projects.

Table 4 Traveling ant total costs

Traveling costs	HRK
Manager Škrlin	10 520,00
QA manager Devčić	2 630,00
Project developer Vlahović	2 630,00
Project developer Milčić	2 630,00
Total:	18 410,00

Margin	606 590,00

Total:	2 000 000,00
VAT (PDV 23 %):	460 000,00
Total:	2 460 000,00

Projects have been of type "Software product new development project". It is a project to create a new software product. A software product is always developed to be used by more than one customer.

The duration of each project was round 8 (eight) months. An example of costs calculation is given in tables bellow.

Table 5 Performance measures

Number	Types of measurement	Measure
1	Project effort	team/member/hours *
2	Productivity	program item/unit/per hour *
3	Project duration	days
4	Schedule predictability	days (%)
5	Requirements completion ratio	% of specified

* as in Section 3

Table 6 Influence factors

Number	Types of measurement	Measure
1	Project size	Functional point **
2	Artefact reuse	% of reused **
3	Project type	graded **
4	Application domain	descriptive **
5	Average team size	number of engineers **
6	Maximum team size	number of engineers **
7	Team expertise	graded by experts
8	Process maturity	graded by experts
9	Functional requirements stability	% of functional requirement changes

** as in Section 4

2 European space agency standard for computer based system

The European Space Agency (ESA) standards are concerned with all software aspects of a Computer Based Systems, including its interface with computer hardware and with other components of the system. The ESA standards are given in two parts: (1) Product standards, and (2) Procedures. The software life cycle in standards involves a sequence of managerial and technical activities which can be grouped in phases. There are six phases that cover system development:
UR phase - Design of the user's requirements
SR phase - Design of the software requirements
AD phase - Design of the general architecture of the system and detailed development plan
DD phase - Detailed design and production of software
TR phase - Transfer of the system to operations
OM phase - Operation and maintenance.

The phase of definition of the user's requirements is not considered as a part of the software life cycle, but constitutes a necessary step to initiate a software project. The software life cycle is the period of time between the initial decisions to implement some functions in software and the end of its utilization in operations [5, 9]. Standard ESA PSS 05 does not include the technique how to perform a design of software but only what has to be performed.

2.1 Design of the User's Requirements

Design of the User's Requirements phase is a preliminary step which begins with decision to investigate the feasibility of developing a computer-based system. It consists of the concise definition of the system user's needs. The users may not have any previous experience with software systems so it is important to ensure the participation of the software engineers in order to develop a good understanding of the problems to be solved with the system. The final product in this phase is the User Requirement Document (URD). Its delivery task is the beginning of the software life cycle [8].

Software life cycle involves a sequence of managerial and technical activities which can be grouped in phases. These phases are considered sequentially for the sake of simplicity, although they tend to overlap to a certain context. A software development project particularly in its early stages is an iterative process where requirements are clarified and alternative designs may be considered.

2.2 Design of the Software Requirements

The approved User Requirement Document constitutes the input to the design of software requirements. The scope of the SR phase is to define the software capabilities and the human functions to be performed in order to fulfil the user's requirements. The output of this phase is the Software Requirements Document. This document is independent of any implementation particularities. The SRD should be formally reviewed by the users, by software engineers and by managers concerned during the software requirements review. Its approval constitutes one of the mayor milestones of the project.

2.3 Design of the general architecture of the system and detailed development plan

The approved Software Requirement Document is the input to this phase, the scope of which is to design the architecture of the system fulfilling the requirements laid down in SRD. The deliverable item which constitutes the output of this phase is the Architectural Design Document (ADD). The ADD should be reviewed and approved by the software engineers, by the users and by the managers concerned during the Architectural Design Review and its approval is one of the major milestones of the project.

2.4 Detailed design and production of software

The SRD and the ADD are the input of this phase, the scope of which is to define the detailed design of the software down to the lowest breakdown level, to produce the code. Concurrently with these activities the Software Detail Design Document (SDDD) and the Software Users Manual (SUM) should be produced together with the code. The SDDD and the SUM are evolving documents; they contain the sections corresponding to the top levels of the system. During this phase, unit testing, integration testing, and system testing are performed.

2.5 Transfer of the system to operations

The main purpose of this phase is to establish that the system fulfils the requirements laid dawn in SRD. The requirement tests are performed according to a test plan. When all tests have been performed successfully, the system is provisionally accepted by the users.

2.6 Operation and Maintenance

The system undergoes a final acceptance test with real data to demonstrate that it meets the requirements defined in SRD. The statement of final acceptance constitutes one of the major milestones of the project. After final acceptance the system enters into routine operation. CCSE analysis concludes before the TR phase and OM phase are implemented. The assessment finishes at the DD phase, before coding, when the design of each module is completed, reviewed and approved, and when all other activities are planed and defined.

3 Performance measures for software projects teams

Performance measures for software projects teams are given in this section, provided with definitions and illustration examples. The set of measures does not represent an exhaustive list, but they are important measures that an organization may collect and use as a basis to compare performance among projects.

3.1 Project Effort

Project effort is the total project team time that is spent on project – related activities during the life cycle of the project.

$$Project\ effort = \sum_{i=1}^{n} Team\_Member\_hours_i, \quad (1)$$

where:
$Team\_Member\_hours_i$ is the time spent on project related activities for team member i;
n is the total number of individuals that contributed time to project related activities over life cycle of the project.

Example: Project team of ten individuals recorded their time spent on project related activities based on the project plan. When project was completed, the cumulative hours for each team member were calculated and the following Tab. 7 was produced.

Table 7 Time spent

Nr	Team member	Time spent cumulative (hours)
1	Project manager	580
2	Requirements analyst	320
3	Requirements analyst	320
4	Project developer	360
5	Project developer	360
6	Project developer	360
7	Project developer	360
8	Project developer	360
9	QA manager	620
10	Documentation person	420

Total: 4060 hours

Therefore,

$$Project\ effort = \sum_{i=1}^{n} Team\_Member\_hours_i = 4060 \text{ hours}. \quad (2)$$

3.2 Team Productivity

Team productivity of a software project is calculated as follows:

$$Team\_Productivity = \frac{Project\_Size}{Project\_Effort}, \quad (3)$$

where:
Project_Size is defined as described in Section 4 of this document, and
Project_effort is defined as described in Section 3 of this document.

Team Productivity is expressed as project size per project hours, where "project size" depends on how the size is measured by an organization (e.g., lines of code, functional points).

Example: Functional points (FP) [1, 5]: A project developed acquired 130 FPs. The project effort to accomplish this was 5300 hours. Therefore,

$$Prod = \frac{Project\_Size}{Project\_Effort} = \frac{130}{5300} = 0{,}025 \text{ FP/hour}. \quad (4)$$

3.3 Project Duration

Project duration is a measure of the length of a project in working days, excluding the times periods when the project is not active due to work stoppages. Project duration includes non-working days such as weekend days and holidays.

Project start date is the date when user requirements were base-lined.

Project end date is the date of the start of the software coding.

Project duration is calculated as follows:

$$Project\_duration = num\_days - stoppage\_days, \quad (5)$$

where:
num_days is the total # of calendar days between the project start date and project end date,
stoppage_days is the number of days when project work was not executed due to work stoppage.

Example: User requirements were baselined on July 15, 2009. Software coding started February 1, 2010. The project was suspended for 10 days during January 2010, as shown in Tab. 8.

Table 8 Project duration

Month	# Calendar days	Stoppage days
July 15	15	
August	31	
September	30	
October	31	
November	30	
December	31	
January	31	10
Total:	199	10

Therefore:
Project_duration = 199 – 10 = 189 days. (6)

3.4 Schedule Unpredictability

Schedule unpredictability is a measure of how much the original project duration estimate differs from the actual project duration that was achieved. Schedule unpredictability percentage is defined as [7]:

$$SP = \frac{(Proj\_duration) - (Est\_Proj\_duration)}{Est\_Proj\_duration} * 100, \quad (7)$$

where:
SP is schedule unpredictability,
Proj_ duration is as defined in Section 3 of this paper.
Est_ proj_ duration is the original estimate of project duration as documented in the baselined version of the project plan.

Example: The estimated duration was documented as 316 days of the project plan. The actual duration realized was 325 days. Therefore Schedule unpredictability is

$$SP = \frac{325 - 316}{316} * 100 = 2{,}8\,\%. \quad (8)$$

3.5 Requirements Completion Ratio

Functional requirements describe what the system, process, product, or service must do in order to fulfil the user requirements. The Requirements Completion Ratio (RCR) measures the extent to which the planned functional requirements were satisfied in the final product implementation. RCR expressed as a percentage is

$$RCR = \frac{Satisfied\_reqs}{Planned\_reqs} * 100\,\%, \quad (9)$$

where:
Planned_reqs is the number of requirements that were originally baselined at the beginning of the project and those that have been added or modified through negotiation with the user,
Satisfied_reqs is the number of functional requirements that were satisfied in the delivered software product.

Example: The original baselined functional requirements specification contained 34 requirements, and 28 of those were satisfied, thus:

$$RCR = \frac{Satisfied\_reqs}{Planned\_reqs} * 100\,\% = \frac{28}{34} * 100\,\% = 82{,}35\,\%. \quad (10)$$

4 Influence factors for software projects – an overview

Influence factors [7] are aspects of the development environment that can impact the outcome of the software project. Some influence factors are controllable by the management, while others are not. When making comparisons between software projects, influence factors can be used to facilitate the comparison of projects that are similar to each other. Influence factors can be considered as independent variables whereas the performance measures act as the dependent variables. Influence factors are: size, artefact reuse, average team size, and process maturity.

4.1 Project size

Project size is the measure of the extent of the software product that is developed and delivered by the project team. It is measured by LLC (Logical Lines of Code) or by Function Point method (FP). The size is measured by FP in this paper.

4.2 Function Point Method

Function point method sizes software by quantifying the tasks that the software provides to the user based primarily on logical design. The objectives of FP analysis are to measure the functionality that the user requests and receives. Five function types are measured: external input, external output, external inquiry, internal logical file, and external logical file. For each project functional points are computed by completing the table shown in Fig. 1 as an example. Five information domain characteristics are determined and the items counts of the weighting factors are provided in the appropriate table locations.

Table 9 Information domain items and weighting factors

Information domain item	Weighting factor		
	Simple	Average	Complex
Number of user inputs	3	4	6
Number of user outputs	4	5	7
Number of user inquiries	3	4	6
Number of files	7	10	15
Number of external interfaces	5	7	10

For an average system the FP is calculated as:

FP=*#*Inputs* + 5*#*Outputs* + 4*#*Inqueries* + 10**Files* +
+ 7*#*Interfaces*. (11)

4.3 Artefact Reuse

Artefact reuse is the use of existing software or software knowledge to build new software or new documents for the project under consideration. Reusable software knowledge items are referred to as reusable artefacts or reusable assets and may include requirement documents, designs, test cases, code, documentation or any other work product that is part of the project's development process.

An artefact reuse value is determined based on the reuse assessment method that is employed. A proxy measure of artefact reuse is defined by [5, 6]:

$$Artefact\ reuse = \frac{PE_{Saved}}{PE_{Total}} * 100, \quad (12)$$

where:
PE_{Saved} is the project effort that was conserved or saved through the reuse of pre-existing work products,

PE_{Total} is the total project effort that is calculated as described in 3.1.

Developing an estimate of artefact reuse relies on judgements made about: (a) the percent of overall project effort required to develop the artefacts, (b) the percent of effort savings realized by artefact reuse.

4.4 Project type

Project type is a classification that characterizes a project as belonging to one of the following type and subtype categories that is shown in Tab. 10 [5, 6].

Table 10 Project types

Type	Subtype	Description
New software	n/a	Newly developed software that does not include a pre-existing base of previously developed software.
Modifications of existing software	Enhancement	Adding, changing, or deleting functionality to a pre-existing application
	Maintenance	Enhancement such as repairing defects, code restructuring, performance tuning, or other changes that are not directly related to changing the functionality of the application.
	Conversion	Conversion of source code so that application can be ported to a different platform. Functionality remains unchanged.
	Package implementation	Acquiring, modifying, configuring, and deploying a commercial off-the-shelf (COTS) software application. No changes made to delivered features or functionality.
	Package customization	Acquiring, modifying, configuring, and deploying a COTS software application. Result in changes to delivered features or functionality.
	Reengineering	Reconstructing an application based on formal design artefacts and a pre-existing software base.

4.5 Software Application Domain

The application domain describes the environment and role of a software application. The application domain of the software project is selected by choosing a category and various subcategories from the shown taxonomy in Fig. 1 [5, 10]:

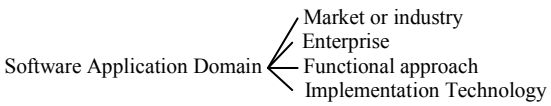

Figure 1 Software Application Domain

4.6 Average Team Size

Average Team Size is the average number of individuals allocated to the project over the course of the project life cycle. It may be calculated by (a) average headcount method, or (b) full-time equivalent (FTE) method. The Average headcount method is used for calculation in this paper given as:

$$Average\ Team\ Size = \frac{\sum_i Team\_Member\_Count}{n}, \quad (13)$$

where:
Team_Member_Count is the number of project staff members who work during month i of the project, and
n is the duration of the project in months.

4.7 Maximum team size

Maximum Team Size is the highest number of individuals that are allocated to the project over the course of the project life cycle. Maximum team size for a project of *n* months duration is calculated as follows:

$$Maximum\ Team\ Size = \mathrm{Max}(x_i.....x_n), \quad (14)$$

where:
x_i is the size of project team that worked at least 40 hours during month *i* of the project, where *i* is a positive integer in the range (1, *n*); and
n is the duration of the project in months.

4.8 Team expertise

In software development, team-based work structures are commonly used to accomplish complex projects. Software project teams must be able to utilize the expertise and knowledge of participants without overwhelming individual members. To efficiently leverage individuals' knowledge and expertise, software project teams develop team cognition structures that facilitate their knowledge activities.

Team expertise is a 5-tuple of measures of the proficiency of the project team during each phase of the development life cycle [11]. The measure is a subjective one based on the informed expert judgment of those who perform the assessment. The time expertise measure for each phase is an integer in range (1 ÷ 5) where 1 represents novice proficiency ability, and 5 represents expert proficiency. The expression for team expertise is given as

$$TE = (TEreq, TEarch, TEdd, TEcode, TEst), \quad (15)$$

where:
TEreq is expertise rating for team members who contribute to the Concept and Requirements Analysis Phase,
TEarch is expertise rating for team members who contribute to Architectural and/or High-Level Design Phase,
TEdd is expertise rating for team members who contribute to Detailed Design Phase,
TEcode is expertise for Code Construction and Unit Testing Phase,
Test is expertise rating for team members who contribute to System Test Phase.

4.9 Process maturity

Process maturity is the extent to which a project's processes are explicitly defined, managed, measured, and controlled. Some of the approaches to process maturity rating are: ISO 9001, ISO 15504 (SPICE), and SEI Capability Maturity Model Integration (CMMI). CMMI originally issued by the Software Engineering Institute extends classical CMM and ISO 9000. These approaches use different rating schemes to indicate the degree of process maturity. A maturity level is a defined evolutionary plateau for organizational process improvement. The maturity levels are measured by the achievement of the goals associated with each predefined set of process areas. There are five maturity CMM levels, each a layer in the foundation for ongoing process improvement, designated by the numbers 1 through 5 as shown in Tab. 11.

Table 11 The five maturity levels of the CMM

CMM Maturity level	Title	What it means
5	Optimizing	Continuous process improvement on all levels. Business objectives closely linked to processes. Deterministic change management.
4	Managed	Quantitatively predictable product and process quality. Well managed, business needs drive results.
3	Defined	Standardized and tailored engineering and management process. Predictable results, good quality, focused improvements, objectives are reached.
2	Repeatable	Project management and commitment process. Increasingly predictable results, quality improvements.
1	Initial	Ad-hoc, chaotic, poor quality, delays, missed commitments.

4.10 Functional Requirements Stability

Functional requirements stability (FRS) is a measure that quantifies the cumulative degree to which the requirements changed throughout the life cycle of the project from the original requirements baseline. FRS is defined as, [7]

$$FRS = \frac{R_T - R_C}{R_T}, \tag{16}$$

where:
R_T is the total number of requirements that were originally base-lined at the beginning of the project; and
R_C is the total number of changes to the original base-lined requirements.

The maximum value of FRS is 1,0 and indicates complete stability of the functional requirements.

5 Result of Performance Measures and Influence Factors

Tab. 7 shows the Influence Factors of 40 medium and small software projects performed in the competence of the CCSE program.

6 Discussion

The companies, organizations, and software projects tried to understand their overall performance, compare it, intending to find the way to become better. The performance measures are core measures that could be identified as part of the set of critical measures of success since they address important attributes of any software development project.

Software organization seeks to find a way to gauge the performance of their software projects against other in order to enable enhanced decision making.

It may want to know what metrics and measures they should use and what reasonable targets for their measures are. Organizations that are more experienced in measurement want to compare their performance with competitors, also intended to learn about the best practice so they can adapt them for their own use through benchmarking. In each of these cases the metrics are used as a valid comparison of measurement data in an integral step toward realizing these objectives.

Before valid measurement comparison and benchmarking can be conducted, common operational definitions for measures must be in place. In this way organizations are able to effectively compare software project performance among projects within their organizations and with projects outside of their organization.

Companies can benefit from adopting a standard set of software project performance measures. By doing so a) personnel within the organization do not need to apply new definitions as they move from one to another project, and b) organization could analyse and compare performance among projects.

The main reasons why organizations measure performance include: Planning and Estimating – Historical measurement data can be used as a basis to forecast or estimate future performance.

Objectives or goals achievement – the purpose of performance measurement is to provide feedback about whether or not an enterprise is meeting its objectives or goals. This feedback improves the likelihood of achieving these objectives or goals efficiently.

Improvement – Performance data can be compared within and outside an enterprise to identify weak or strong areas. Communication – The reporting of well-defined performance measures can enhance staff and partner mutual understanding and support strategies and decisions.

The presented examples have shown high schedule predictability and high requirement completion ratio, enabling thus expected successful completion after detailed coding.

Table 12 Influence factors of software projects

Project number	Size (FP) *	Artefact reuse (%)	Project type **	Application Domain *	Average team size (num.)	Maxim. team size (num.)	Team expertise *	Process Maturity (CMM)	Functional requirement stability *
1	150	14	New software	Enterprise	10	12	3,0	2,0	0,85
2	200	12	Modifications reengineering	Market or industry	12	14	3,0	2,0	0,80
3	160	15	New software	Market or Industry	8	12	2,4	2,5	0,80
4	130	20	New software	Market or industry	8	10	2,0	3,0	0,82
5	160	14	New software	Market or industry	9	10	2,4	2,8	0,78
6	220	11	Modification enhancement	Market or industry	12	14	2,8	2,5	0,86
7	110	21	Modification enhancement	Enterprise	10	11	2,8	2,0	0,84
8	150	15	New software	Enterprise	12	13	2,8	2,8	0,80
9	130	12	New software	Enterprise	10	12	3,2	2,6	0,75
10	150	12	Modification maintenance	Enterprise	12	14	3,2	3,2	0,80
11	140	16	New software	Enterprise	12	13	3,2	2,8	0,75
12	160	21	New software	Enterprise	11	11	2,6	2,6	0,80
13	140	13	New software	Market or Industry	10	12	2,8	2,5	0,82
14	180	12	Modification enhancement	Enterprise	8	10	2,6	2,6	0,85
15	180	12	Modification reengineering	Enterprise	10	12	2,8	2,5	0,78
16	150	10	New software	Market or industry	12	14	3,4	2,8	0,82
17	150	13	New software	Enterprise	10	12	3,0	2,8	0,85
18	130	14	New software	Enterprise	8	12	2,8	2,6	0,73
19	120	15	New software	Enterprise	10	12	2,8	2,8	0,65
20	180	20	New software	Enterprise	12	14	3,0	3,0	0,84
21	170	14	Modification reengineering	Enterprise	12	14	3,0	2,6	0,85
22	140	12	New software	Enterprise	10	12	2,8	2,8	0,85
23	150	15	New software	Enterprise	10	12	2,8	2,0	0,75
24	150	15	Modification reengineering	Enterprise	11	14	3,0	2,0	0,65
25	170	12	New software	Enterprise	9	10	2,6	2,5	0,83
26	120	12	New software	Enterprise	12	12	2,8	2,6	0,82
27	180	15	New software	Market or industry	10	12	2,6	2,6	0,80
28	130	13	New software	Market or industry	8	10	2,6	2,8	0,78
29	170	18	New software	Enterprise	12	14	3,0	2,5	0,85
30	150	13	New software	Enterprise	10	12	2,8	2,6	0,77
31	160	17	New software	Enterprise	10	12	2,8	2,6	0,78
32	150	18	New software	Enterprise	11	12	2,6	2,5	0,84
33	140	19	New software	Enterprise	9	12	2,8	2,5	0,85
34	130	13	New software	Enterprise	8	10	3,0	3,2	0,80
35	160	12	New software	Enterprise	13	14	3,2	2,5	0,75
36	160	16	Modification reengineering	Market or industry	12	14	3,4	3,0	0,82
37	170	17	New software	Enterprise	12	14	3,6	3,0	0,80
38	150	12	New software	Enterprise	10	12	2,8	2,8	0,73
39	180	13	New software	Enterprise	12	14	2,8	2,5	0,78
40	120	15	New software	Enterprise	8	10	2,4	2,5	0,72

* as in Section 4.
** as in Tab. 10.

7 Conclusion

This paper presents a set of software project performance measurement and influence factors usable by organizations for the software project development. The terms and definition presented can be used by organizations that are: a) beginning a measurement program intended for increasing professional standards of the firm, b) standardizing the way measures are defined already across the enterprise, c) conducting benchmarking between projects within and outside of organization, and d) comparing their performance with projects that have submitted their data to proprietary and public project performance repositories.

Acknowledgement

The authors would like to thank prof. dr. sc. Franjo Jović who collaborated at the CCSE in Osijek for his valuable remarks.

Table 13 Indicators of software projects

Project number	Project effort / hours	Productivity / FP per hour	Project duration / days	Schedule predictability / %	Requirements completion ratio / %
1	3610	0,026	180	25	95,5
2	3220	0,027	190	22	96,0
3	3320	0,027	185	24	96,5
4	3600	0,026	178	25	90,7
5	3400	0,028	180	10	92,4
6	3210	0,034	176	15	95,0
7	3450	0,028	178	18	94,4
8	3480	0,030	190	18	96,0
9	3500	0,028	185	20	90,8
10	3450	0,027	190	22	95,4
11	3600	0,030	178	8	94,5
12	3500	0,028	180	32	94,0
13	3400	0,033	176	16	95,0
14	3400	0,027	175	17	90,8
15	3250	0,030	180	18	95,0
16	3400	0,026	180	24	90,8
17	3600	0,028	185	6	93,3
18	3610	0,027	178	20	96,6
19	3650	0,028	187	8	94,5
20	3320	0,030	186	14	92,0
21	3310	0,032	180	10	91,5
22	3400	0,028	190	10	94,4
23	3650	0,030	187	14	93,0
24	3700	0,028	180	15	92,7
25	3800	0,027	185	16	92,0
26	3610	0,031	175	20	93,8
27	3600	0,029	180	12	92,0
28	3460	0,027	190	14	93,8
29	3650	0,032	192	14	95,0
30	3800	0,026	187	15	94,5
31	3610	0,027	182	20	92,3
32	3550	0,028	175	20	92,0
33	3420	0,030	170	14	92,3
34	3280	0,028	187	15	93,3
35	3700	0,030	187	14	93,5
36	3620	0,027	180	12	94,0
37	3320	0,025	192	18	95,5
38	3600	0,024	183	8	92,0
39	3530	0,025	180	10	91,6
40	3500	0,028	178	12	92,0

8 References

[1] Albrecht, A. J. Measuring Application Development Productivity. // Proceedings of the SHARE/GUIDE IBM Application Development Symposium. Monterey, CA, 1979.

[2] Basili, V.; Caldiera, G.; Rombach, H. D. The Goal Question Metric Approach. // Encyclopedia of Software Engineering, John Wiley & Sons, Inc. 1994.

[3] Chrisis, M. B.; Conrad, M.; Shrum, S. CMMI, Guidelines for Process Integration and Product Improvement, NY, Addison-Wesley, 2006.

[4] Glass, R. L.; Vessey, I. Contemporary Application Domain Taxonomies, IEEE Software, July 1995.

[5] Ebert, C.; Dumke, R. Software Measurement, Springer Berlin, ISBN 978-3-540-71648-8, 2007.

[6] ISO 2006b, ISO/IEC 14143-6:2006 - Information technology - Software measurement - Functional size measurement - Part 6, 2006.

[7] Kasunic, M. A Data Specification for Software Project Performance Measures: Results of a Collaboration on Performance Measurement, Technical report, CMUS/SEI-2008-TR-012, 2008.

[8] Pressman, R. S. Software engineering a practitioner's approach, McGraw Hill, NY, 2005.

[9] Software Engineering Standard ESA PSS 05, ESA Publication Division, ESTEC, Netherlands, 2003.

[10] Stevenson, T.; Carrington, D.; Strooper, P.; Sharon, N. An industry/university collaboration to upgrade software engineering knowledge and skills in industry. // The Journal of Systems and Software, 75, Elsevier, 2005.

[11] William, R.; He, J.; Butler, T.; Brian, S.; King, M. Team Cognition: Development and Evolution in Software Project Teams. // Journal of Management Information Systems. 24, 2(2007).

Authors' addresses

Ninoslav Slavek, Assistant professor
Faculty of Electrical Engineering
Kneza Trpimira 2B, 31000 Osijek, Croatia
ninoslav.slavek@etfos.hr

Damir Blažević
Faculty of Electrical Engineering
Kneza Trpimira 2B, 31000 Osijek, Croatia
damir.blazevic@etfos.hr

Krešimir Nenadić, Assistant professor
Faculty of Electrical Engineering
Kneza Trpimira 2B, 31000 Osijek, Croatia
kresimir.nenadic@etfos.hr