Serdica J. Computing **1** (2007), 1–12

Serdica
Journal of Computing

Bulgarian Academy of Sciences
Institute of Mathematics and Informatics

COMPARATIVE ANALYSIS: A FEASIBLE SOFTWARE ENGINEERING METHOD

Nelly Maneva

Abstract. The reasonable choice is a critical success factor for decision-making in the field of software engineering (SE). A case-driven comparative analysis has been introduced and a procedure for its systematic application has been suggested. The paper describes how the proposed method can be built in a general framework for SE activities. Some examples of experimental versions of the framework are briefly presented.

1. Introduction. A great number of promising methods and techniques have been created by scientists in the field of software engineering (SE). Unfortunately, most of them remain only elegant intellectual exercises. Such methods are with small practical value and are not being used (at all or properly) due to their sophisticated formal descriptions, the lack of defined transition procedures and supporting tools [4], [5], [9], [15]. Common software practitioners, under the pressure of the everyday tasks have no resources to study innovations so as to realize their benefits and to dare to apply them.

Bearing in mind the above mentioned, we try to develop a method satisfying the following requirements:

ACM Computing Classification System (1998): D.2.9.
Key words: multiple criteria, decision making, quality models, software metrics.

- to be general enough so as to be applied into different circumstances;
- to have a sound mathematical basis;
- to be technically and economically feasible;
- to be flexible so as its proper use should be achieved through tuning to concrete situations;
- to be independent from any particular style of work or environment.

There are three underlining principles for the method:

a) Interpretation

The proposed method should follow the modern paradigm of scientific knowledge – to move from factological description of the objective world to dialogical interpretations, made by a valuable subject [20]. In the area of software engineering this principle means to recognize the role of "peopleware" and to consider the human factor as a significant and crucial for the success of each SE activity.

b) Reasonable choice

All software development can be characterized as a problem solving loop [17] in which we want to build in the principle of the "reasonable choice". Such approach will be systematic and efficient because it will accomplish a local optimization at each critical decision point.

c) Measurement

Method should include the systematic use of software metrics. Measurement enables software people to gain insight into their work and products developed. Further the obtained measures can be analyzed to provide assistance in management and technical actions.

The paper is organized as follows. Section 2 describes briefly the proposed method and the procedure for its application. Section 3 presents a general framework, through which the comparative analysis can be adopted for practical use. The feasibility of the framework is examined for a number of software engineering activities with different significance, scope and complexity. Conclusions in Section 4 summarize the results and give some directions for future work.

2. What is Comparative Analysis (CA).

Before starting the explanation of the method in greater detail, it is necessary to introduce some terms and notions.

2.1. Definitions.

Object – any item under consideration. According to the Fenton's classification [8], any object in SE field belongs to one of the following classes – products, processes or resources.

Local state of the object – the condition of an object at a given point in time.

View – an object description or judgement belonging to a person with a special role in software development, for example a designer view, a user view, etc.

Quality – the totality of features and characteristics of an object that bear on its ability to meet stated or implied needs.

Quality content – object quality considered at a given point in time, under a specific view.

Comparative analysis is a study of the quality content of a set of homogeneous SE objects and their mutual comparison so as to select the best, to rank them (establishing a preference order) or to classify each object to one of the predefined quality categories.

The comparative analysis (CA) shares the main objectives and methods of the broad theory of Multiple Criteria Decision Making [3], [21], trying to specify and apply them systematically in the field of software engineering.

Two main participants have been involved: the ***Analyst***, responsible for all aspects of CA implementation, and a CA customer. ***CA customer*** is a single person or a group of persons, tasked with making a decision in a given situation. Depending on the customer's role (a programmer, a project or a quality manager, a user, etc.) the defined problem and the software life cycle moment, a ***case*** should be opened to determine the context of the desired comparative analysis.

Each case can be described by the following elements:

$$\text{case} = \{\text{View, Goal, Object, Competitors, Task, Level}\}.$$

The ***View*** describes the customer's role and the perspective from which the comparative analysis will be performed.

The CA **Goal** can be:

- to characterize (to gain understanding);
- to evaluate (to determine status with respect to expectations);
- to predict so that it will be possible to plan what to do;
- to improve the situation;
- other – defined by the customer.

The **Object** represents the SE object under consideration.

The set of characteristics selected to represent the quality content and the set of their relationships constitute the ***object quality model.*** It creates a description of the object, to which the proposed CA will be applied.

According to the Goal, the instances of the objects to be compared have been selected, forming the set **C** of **Competitors C = ($\mathbf{C}_1$, $\mathbf{C}_2$, ..., $\mathbf{C}_n$).**

The size and the content of the set **C** depends on the defined Goal of the desired CA. This can be illustrated by the following examples:

Example 1. If the Goal is only to create the quality model of a given object, then the set **C** is empty.

Example 2. If the Goal is to evaluate the quality of one object, the set **C** comprises only one element – an instance of the object.

Example 3. If the Goal is to perform the Comparative analysis so as to obtain the ranking of a number of objects, the set **C** comprises these objects.

Example 4. If the Goal is benchmarking, the set **C** has two elements – the analyzed object and the model of the "ideal" object.

Example 5. If the Goal is to study the tendencies in quality achievements, the set **C** should comprise the instances of the consecutive local states of the object.

Example 6. If the Object is a project and the Goal is to assess the progress of the current project, the set **C** will comprise the project under development and some previously evaluated projects, for which there are available historical data.

The CA **Task** can be Selection (to find the best), Ranking (to obtain a completely ordered list), Classification (to define the appropriate quality category for each object) or any combination of them.

The depth **Level** defines the overall complexity of the CA and depends on the importance of the problem under consideration and on the resources needed for CA implementation. The key idea is to adjust the performed comparison to the intentions of the Customer. Sometimes only a quick insight into the quality of a small set of objects is needed, contrary, for example, to a sophisticated CA to support crucial decision about long-term strategic program.

Object quality model. The case-driven CA begins with creating the hierarchical object quality model. The first level comprises **m** quality factors $\mathbf{F}_1, \mathbf{F}_2, \ldots, \mathbf{F}_\mathbf{m}$. They characterize the local state of the object as regards the defined case. Each factor $\mathbf{F}_j$ must be weighed in accordance with its relative importance. So, the following vector:

$$\mathbf{W}(\mathbf{m}) = (\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m)$$

is associated with the set of factors.

Depending on the defined depth level and the cognitive complexity of the factors, the latter can be further decomposed. The obtained hierarchical structure

describes which quality characteristics will be considered at different levels and what the relationships among them will be.

One of the most challenging problems in CA implementation is the creation of a quality model for a given object. Our suggestion is to apply an incremental approach. When a new object appears, the Analyst has to create its first model, using different sources of information. The Customer's requirements to the object should be analyzed and reformulated in terms of quality characteristics. Our experience till now shows that any Internet Search engine is a perfect tool for finding and thoughtfully studying some existing models. Then the created model is saved as a generic (basic) model for the object under consideration. When a request for CA of the same object arises, the generic model should be modified. First, there should be an attempt to find some new quality characteristics to be added at some levels of the hierarchy so as to reflect a new view or goal of the current case. If the try is successful, the expanded hierarchical structure will be stored as a generic model. The second step in modification is to decide whether some characteristics can be ignored as irrelevant to the case under consideration. So a derivative of the model is obtained. A special pattern, describing the deviation from the basic model is created and saved with reference to the case for further re-use.

On the basis of some published research results a number of generic models have been created – for software products, for representations, for SE processes, for software standards, for developers, etc.

As an example we can mention the generic model, created for the object "SE model". This model comprises three quality factors (Utility, Applicability, Validity) at the first level of hierarchy and a number of criteria at the second level. The definitions of the quality characteristics and the corresponding hierarchical model can be found in [14].

Software metrics. According to the third underlining principle for the CA, the use of software metrics is obligatory. Unfortunately, the flexibility of the case-driven CA with arbitrary objects and dynamically defined derivatives of their quality models makes the measurement problem quite difficult. Theoretically, each characteristic, represented as a node in a generic quality model can be a leaf in some derivative model and for this characteristic an appropriate metric should be assigned.

The foundation of the software measurement theory has been built [8], [10], [22] and a significant number of software metrics have been proposed (an Internet search engine shows more than 45000 entries!). So our suggestion is to classify the existing metrics according to the measured characteristic and when

a metric is needed to look for a similar one. As for the object quality models, the same approach for incremental construction of a collection of re-used metrics should be applied. To facilitate the saving into repository and searching the appropriate metric, a unified scheme for metric description has been proposed, which comprises:

- the conceptual and operational definition of the measured characteristic;
- the measurement method and the procedure for its application (in case it is possible – with two different complexity levels);
- general information about the background and validity of the metric.

In order to show the feasibility of our approach we select the checklists as a universal technique for obtaining the measure of a given quality characteristic. The reasons for our choice are:

- this technique is experimentally validated;
- the technique is flexible enough, because it poses the right questions at the right moment – very convenient for the case-driven comparative analysis;
- all questions are formulated in a natural language and the so called "intellectual barrier" [22] can be overcome more easily;
- there is a great number of existing and available checklists, which can be used with minor modifications;
- it is easy to define different complexity levels in assessment, controlling the number of questions and the applied scale for the answers – binary or k-degree.

The experiments till now confirm the applicability of checklists in various cases.

Evaluation and exploitation of the results. When the object model is created and the set of metrics for leaves are defined, the next step is to perform a bottom-up evaluation in order to obtain a measure for each factor for each of the compared objects. Thus we fill the objects-factors matrix $\mathbf{E}(\mathbf{n} \times \mathbf{m})$, where $\mathbf{n}$ is the number of the objects and $\mathbf{m}$ is the number of the quality factors. Each element $\mathbf{E_{i,j}}$ is the measure of the $\mathbf{i}$-th object with respect to the $\mathbf{j}$-th quality factor.

Further we have to perform the selection, ranking or classification task. So we can find the object with the best quality or identify some objects as the most promising candidates for further study, or we have a ranking or a classification to support the customer's decision.

There are a number of methods, well known from the field of MCDM. Till now two methods for selection and two methods for ranking have been adapted and implemented. There is an available prototype of a classification tool [7],

which will be re-engineered and added to the collection of methods.

Some heuristics have been proposed so as to select that method for the CA task, whose complexity matches the defined depth level.

The last stage of the comparative analysis is the study and interpretation of the results according to the posed goal.

Considering the same crucial for the decision making situation, we can analyze it from distinctly different perspectives (from the point of view of participant, who can be a developer, a project manager, a tester or a user). Thus different cases should be opened and a systematic and thoughtful approach to the problem solving will be accomplished.

Some formal techniques have been introduced for constructing the Directed Acyclic Graph, which represents the hierarchical quality model, for defining the software metrics and their statistical validation and for describing the available methods of selection, ranking and classification.

2.2 A process model for the comparative analysis. Within the comparative analysis a number of steps can be identified, such as pre-analysis, preparation, construction, execution and completion. Each step comprises a number of activities. For each activity its aim, the results to be obtained and how the activity must be performed have been described. This model clarifies who has to do what and when thus making the comparative analysis manageable and properly applicable.

The detailed model can be described as follows:

Step 1. Pre-Analysis. For the initialization, a Customer may identify a problem for which the Comparative Analysis seems to be useful. Recognizing the need for CA, the Customer creates a CA Request, comprising an informal description of the problem situation, moment and area of consideration.

The CA Request is examined by the Analyst, who performs an analysis of its conceptual and technical feasibility and decides how to proceed. The request can be declined, postponed or accepted. The Customer receives a notification together with a reasoning behind the decision.

Step 2. Preparation. When the CA Request is approved a new case is opened and the second step should be accomplished. The purpose of this step is to define the CA context and to plan the CA implementation.

First, the components of the opened case must be defined, namely the View, Goal, Object, Competitors, Task and Level. The relevant sources of information, needed for the CA performance should be determined and made available.

Second, a CA plan is created. It describes the overall organization, personnel involved, tasks and responsibilities allocation and CA schedule. A rough indication of the cost is also given.

Step 3. Construction. This step is made up of four substeps:

Substep 1. Create a new or modify an existing generic model so as to obtain a derivative object model for the CA, driven by this case.

Substep 2. Create or select and modify a metric for each leaf in the derivative hierarchical object model.

Substep 3. Select and apply a method for determining the weights for the quality factors under consideration. Apply the MECCA (Multi-Element Component Comparison and Analysis) approach for definition of the weights within the whole hierarchical structure.

Substep 4. According to the defined CA Task and depth Level, select the most appropriate method for selection, ranking or classification.

Step 4. Execution. The purpose of this step is to obtain the desired results.

Substep 1. Perform the evaluation of each object from the set of Competitors and fill the objects-factors table.

Substep 2. Analyze the results of evaluation and transform the table into a normalized form.

Substep 3. Apply the selected method, accomplishing the CA task and document the results obtained.

Step 5. Completion.

Substep 1. Analyze the CA results, prepare the final report and plan some follow-up activities.

Substep 2. Describe and register some new re-use components, i.e. a case, a basic model or a pattern for the derivative model, metrics, checklists, etc.

Substep 3. Assess the performed comparative analysis. The created concluding report should summarize the identified problems in CA implementation, cost and time planning versus realization and some recommendations for improvement. The parameters of the CA should be added to historical data to be available for further processing.

A tool supporting the main steps of the comparative analysis has been implemented.

A number of factors are identified to be of great importance for the success of the Comparative analysis, namely:

- qualification, motivation and commitment of all people, involved in CA implementation;

- solid theoretical basis and appropriate level of validity of all re-used components, especially for the basic object models, metrics and applied methods;
- degree of the automation of the activities performed during the CA – i.e. the quality of the supporting tool.

3. INSPIRE: a framework for SE activities. Next we try to put together the comparative analysis and some well-known best practices in the field so as a general SE framework to be defined. The assumption is that there are many useful methods and techniques and the aim should be to develop an approach, which will encourage software developers to use it in their practical work.

The proposed methodology has been described in [16]. Here we are going only to present the key ideas, encoded in the name of the approach. Follows their brief explanation.

The approach should be:

- **I**ncremental – to start with some regulations and their systematic use for only a few selected SE activities and after their successful adoption to join other ones;
- **N**eat: to be precise, systematic and straight, based on the use of the same agile method, supported by an automated tool and with a well-defined implementation cycle;
- **S**calable: to be flexible so as to reflect the particular features of a software company, project, controlled SE activities, etc. To allow a few different complexity levels of implementation according to the stated goal and planned resources;
- **P**ermanent: to be persistent in the approach realization and to achieve continuous improvement through a long-term strategy;
- **I**ntegrated: to construct and support a set of mutually interrelated activities, whose joint implementation will bring some benefits;
- **R**ight: to be in accordance with what is proven to be just, good or proper, comprising a number of already validated methods and best practices;
- **E**stimable: to include the software measurement as an obligatory "umbrella" activity because it is widely recognized that "we cannot control what we cannot measure".

We believe that it is possible to create such a "perfect" approach, satisfying so many requirements through the Comparative Analysis – a method, applicable to the software engineering work at any moment of the software life cycle. It can be used at any situation, where a reasonable choice should be made with changed

view, goal and level of resolution. Once the framework is adopted in a given software organization, it is invariant and serves as the basis for the systematic application of the comparative analysis.

We develop the framework together with some well-defined rules for its transfer into practice – the framework introduction into a given software company should comprise well defined steps [16].

In order to examine the feasibility of our approach, we try to apply it to several SE activities with different significance, scope and complexity. Next follows the activities, selected for the experiments: software quality assurance [11], usability assurance [12], outsourcing development [13], and teamwork building [16]. More details and the lessons learned can be found in the published papers, mentioned above.

4. Conclusions. Some directions of our future research and practical work are the following:

- to expand the collection of generic models for different SE objects, analyzing the existing and looking for some new proposals for quality characteristics. It will be nice to be able to define a kernel set of quality factors for each class (products, processes and resources) to start with and enrich it further for a particular object.
- to apply the suggested approach to of-the-shelf software packages and see whether the CA approach can facilitate the evaluation and selection of complex application software such as ERP (see [1]), CRM and the like.
- to create a conceptual model of an integrated environment, supporting the proposed INSPIRE framework and to develop a prototype to examine its features;
- to identify some active research centers in the field of SE and to look for collaboration with their experts so as to improve the theoretical basis and the content of the re-use repository.

REFERENCES

[1] Ahituv N., S. Neumann, M. Zviran. A System Development Methodology for ERP Systems. *Journal of Computer Information Systems*, Spring (2002), 56–67.

[2] Anderson E. A Heuristic for Software Evaluation and Selection. *Software Practice and Experience* **19**, No 8 (1989), 707–717.

[3] Biro M. et al. Business Decision Problems Supported by Software Product and Process Assessment. ISCN'95 – Newsletter, `http:/www.iscn.ie/news/iscn95/doc-15.html`

[4] Brooks F. P. No silver bullets: Essence and accidents of SE. *IEEE Computer* **20**, No 2 (1987), 10–19.

[5] McConnell St. After the Gold Rush. Microsoft Press, 1999.

[6] Delvin, K. The real reason why software engineers need math. *CACM* **44**, No 10 (2001), 21–22.

[7] Eskenasi A., V. Angelova. A New Method for Software Quality Evaluation. *Journal of New Generation Computer Systems* **3**, No 1 (1990), 47–53.

[8] Fenton N. Software Metric – A Rigorous Approach. Chapman&Hall, 1991. `http://vorlon.ces.cwru.edu/~bxm4/radial.html`

[9] Hussman H. Formal Foundations for Software Engineering Methods. Springer-Verlag, 1997

[10] Jones C. Applied Software Measurement. McGraw-Hill, 1997

[11] Maneva N. Implementation of a Software Quality Improvement Program. Proc. of the Int. Conference CompSysTech'2000, II.7-1, II.7-6.

[12] Maneva N. An Approach to Usability Assurance. Proc. of the Int. Conference CompSysTech'2003, II.3-1, II.3-5.

[13] Maneva N. Software Quality Assurance and Maintenance for Outsourced Software Development. Proc. of the First Balkan Conference on Informatics, 21–23 Nov 2003, Thessaloniki, Greece, 644–649.

[14] Maneva N. Software Engineering Models and their Evaluation. Proc. of the MASSE'2003, *Mathematica Balkanika, New Series* **18**, Fasc. 1–2 (2004), 149–156.

[15] Pressman R. Software engineering – A Practitioner's Approach, Fifth edition. McGraw Hill, 2001.

[16] Maneva N., Nikolova N. Team-work training for software people. Proc. of the 34 Spring conference of the UBM, *Math. and Education in Math.* **34** (2005), 243–250.

[17] Raccoon L. B. S. The Chaos Model and the Chaos Life Cycle. *Software Engineering Notes* **20**, No 1, (1995), 55–66.

[18] Raccoon L. B. S. Fifty years of progress in SE. *Software Engineering Notes* **22**, No 1 (1997), 88–104.

[19] Sanders J., E. Curran. Software Quality. Addison–Wesley, 1994.

[20] Wilber K. A Brief History of Everything. Shambhala Publ. Inc., 1996

[21] Zelemy M. Multiple criteria decision making. McGrow-Hill Book Company, 1982

[22] Zuse H. A Framework of Software Measurement. Walter de Gruyter&Co, Hawthorne, NY, USA, 1997

Software Engineering Dept.
Institute of Mathematics and Informatics
Bulgarian Academy of Sciences
Acad. G. Bonchev Str., Bl. 8
1113 Sofia, Bulgaria
e-mail: neman@gbg.bg

Received February 15, 2006
Final Accepted February 23, 2007