SOFTWARE ENGINEERING DECISION SUPPORT – A NEW PARADIGM FOR LEARNING SOFTWARE ORGANIZATIONS

Günther Ruhe
University of Calgary
ruhe@ucalgary.ca
http://sern.ucalgary.ca/~ruhe/

Abstract: Software development and evolution is characterized by multiple objectives and constraints, by a huge amount of uncertainty, incomplete information and changing problem parameters. Success of software development very much depends on providing the right knowledge at the right time, at the right place, and for the appropriate person. Experience factory and organizational learning approaches are increasingly used to improve software development practices.

The paradigm of Software Engineering Decision Support (SEDS) goes beyond the concept of reusing models, knowledge or experience. For more focused problem domain, emphasis is on providing methodology for generation, evaluation, prioritization and selection of solution alternatives. Typically, modelling, measurement, empirical and simulation-type investigations are combined with intelligent methods of analysis and reasoning to predict the impact of decisions on future life-cycle performance.

This paper describes fundamental principles and expectations on SEDS. A comparison with knowledge management-based approaches is performed for the areas of requirements negotiation and COTS selection. The initial hypothesis on the expected benefits of SEDS are discussed for the two case study examples in the area of requirements negotiations.

1. Introduction

The need for further development of software engineering practices within companies adds to the demand for systematic knowledge and skill management in combination with active usage of this knowledge to support decision-making at all stages of the software lifecycle. With continuous technological change, globalization, business reorganizations, e-migration, etc. there is a continuous shortage of the right knowledge at the right place at the right time. Subsequently, strategic and operational decisions concerning products, processes, technologies or tools and other resources, are far from being mature. Reactive management is the rule, and pro-active analytical performance is more the exceptional case.

Experience factory and organizational learning approaches are increasingly used to improve software development practices [15], [18]. The main idea of experience based learning and improvements are to accumulate, structure, organize and provide any useful piece of information being reused in forthcoming problem situations [2]. Reuse of know-how is essentially supported by the case-based reasoning methodology [1]. However, software development and evolution typically is large in size, of huge complexity, with a large set of dynamically changing problem parameters. In this situation, reuse of experience alone is a useful, but non-sufficient approach to enable proactive decision analysis. Diversity of project and problem situations on the one hand, and costs and availability of knowledge and information organized in a non-trivial experience (or case) base on the other hand, are further arguments to qualify decision-making.

The idea of offering decision support always arises when decisions have to be made in complex, uncertain and/or dynamic environments. The process of software development and evolution is an ambitious undertaking. In software development and evolution, many decisions have to be made concerning processes, products, tools, methods and techniques. From a decision-making perspective, all these questions are confronted by different objectives and constraints, a huge number of variables under dynamically changing requirements, processes, actors, stakeholders, tools and techniques. Very often, this is combined with incomplete, fuzzy or inconsistent information about all the involved artefacts, as well as with difficulties regarding the decision space and environment.

Typically, a concrete decision support system is focused on a relatively narrow problem domain. There are two kinds of existing contributions to Software Engineering Decision Support. Firstly, an explicitly mentioned effort to provide decision support in a focused area of the software life cycle. Examples are decision support for reliability planning [17] or decision support for conducting inspections [9]. Secondly, this encompasses research results that indirectly contribute to decision support, although not explicitly stated as such. Basically, most results from empirical software engineering, software measurement or software process simulation can be seen to belong to this category.

The main purpose of this paper is to position SEDS as both complementary and supplementary to experience factory or learning software organization approaches. The concrete relationship is problem and context dependent.

The paper is subdivided into five parts. Following this introduction is a characterization of Software Engineering decision-making. Software Engineering decision support systems (SE-DSS) couple the intellectual resources of individuals and organizations with the capabilities of the computer to improve the quality of solutions. They are described in more detail in part 3. This is followed in part 4 by an analysis of the concrete examples for offering support for crucial decisions. One example concerns requirements selection. The other is related to support in software release planning. Finally, the summary and an outlook are presented in part 5.

2. Why do We Need Support for Making Decisions in Software Engineering?

Software Engineering is defined as [21]

1. the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, that is, the application of engineering in software, and
2. the study of approaches as in 1.

Both parts of this definition imply a large number of detailed questions and necessary decisions on how to do that, i.e., they are concerned with SEDS. The demand for decision support covers the complete life cycle. For the analysis, design, construction, testing and evolution phase, decision makers need support to describe, evaluate, sort, rank, select or reject candidate products, processes, resources, tools and technologies. Example decisions are related to:

- ❑ **Requirements**: Which functional and non-functional requirements should be chosen according to the given budget and time constraints [16]? How to assign different requirements to releases under the assumption of an incremental development paradigm [7]?
- ❑ **Architecture and design**: How should the selection between candidate architectures be made to ensure the best fit in terms of software reliability, performance, security, and usability [22]? How to integrate available components and COTS products into a system design [14]?
- ❑ **Adaptive and corrective maintenance**: Which components need improvement during the maintenance cycle because of changing contexts and requirements [24]? Which modules are potentially low qualified during the development phase and need special emphasis during maintenance [20]?

- **Project planning and control**: What should the reaction be to shortages on budget, time or available resources? Which trade-offs are acceptable to deliver the product earlier? How should deficits in staff be compensated? How do we respond to violations of quality constraints for intermediate products?
- **Verification and validation**: Which technique is most appropriate? Which artefacts should be investigated? When to terminate testing or inspections? Is there any need for re-inspections? How to integrate components for system testing [5]?

Decision-making is a well-established discipline with origins and close interactions with many other disciplines such as economics, operations research, game theory, probability theory, control theory, psychology, and cognitive science. The emphasis of decision support is to provide as much background as possible for actually making the decision. This is a very essential input for the actual decision-maker (typically, a completely different person).

Decision support has been successfully designed, developed and applied in many areas such as logistics, manufacturing, health care, forestry or agriculture. Why do we also need decision support in software engineering? Some of the major concerns we encountered for current real-world situations in software development and evolution are summarized below:

- Decision problems are often poorly understood and/or described.
- Decisions are done at the last moment and/or under time pressure.
- Decisions are not relying on empirically evaluated models, best knowledge and experience and a sound methodology.
- Decisions are made without considering the perspectives of all the involved stakeholders.
- Decisions are not explained or made transparent to those involved.

What are the expectations and requirements for systems - offering SEDS? We define a set of “idealized” requirements on support systems that combine the intellectual resources of individuals and organizations with the capabilities of the computer to improve effectiveness, efficiency and transparency of decision-making Depending on the concrete problem topic and the usage scenario of the DSS (on-line versus off-line support, individual versus group-based decision support, routine versus tactical versus strategic support), different aspects will become more important than others.

(R1) **Knowledge, model and experience management** of the existing body of knowledge in the problem area (in the respective organization).

(R2) **Integration** into existing organizational information systems (e.g., ERP systems).

(R3) **Process orientation** of decision support, i.e., consider the process how decisions are made, and how they impact development and business processes.

(R4) **Process modeling and simulation component** to plan, describe, monitor, control and simulate ("what-if" analysis) the underlying processes and to track changes in its parameters and dependencies.

(R5) **Negotiation component** to evolutionary find and understand compromises.

(R6) **Presentation and explanation component** to present and explain generated knowledge and solution alternatives in various customized ways to increase transparency.

(R7) **Analysis and decision component** consisting of a portfolio of methods and techniques to evaluate and prioritize generated solution alternatives and to find trade-offs between the conflicting objectives and stakeholder interests.

(R8) **Intelligence component** to support knowledge retrieval, knowledge discovery and approximate reasoning.

(R9) **Group facilities** to support electronic communication, scheduling, document sharing, and access to expert opinions.

3. Software Engineering Decision Support Systems o- Basic Architecture

Software Engineering Decision Support Systems (SE-DSS) can be seen as an extension and continuation of the Software Engineering experience factory and LSO approaches. In addition to collecting, retrieving and maintaining models, knowledge, and experience in the form of lessons learned, SE-DSS generates new insights from on-line investigations in a virtual (model-based) world, from offering facilities to better structure of the problem as well as in ranking and selecting alternatives. For this purpose, sound modeling and knowledge management is combined with a variety of techniques of analysis, simulation, and decision-making.

While learning the software organization approach is mainly addressing the learning aspect from an organizational perspective, the emphasis of real-world SE DSS is typically on more focused aspects of the software engineering life-cycle, e.g., resource planning, COTS selection or requirements negotiation.

The underlying hypotheses of using Software Engineering Decision Support Systems are:

Hypothesis 1: SE-DSS enables making more effective decisions (improved quality).

Hypothesis 2: SE-DSS enables - making more efficient solutions.

Hypothesis 3: SE-DSS allows more transparent decisions (to be better understood by involved individuals), reflecting trade-offs between conflicting criteria or stakeholder opinions.

Hypothesis 4: SE-DSS can be used to propose more robust decisions (stable under slightly changing environments).

Hypothesis 5: SE-DSS in combination with proper modelling, optimization and simulation facilities can be used to generate and evaluate new solution alternatives and to better react on changes in the problem parameters.

Ideally, a SE-DSS should have simulation facilities to conduct scenario-based experiments in a virtual world. Simulation models can be used to systematically develop and evaluate improvement suggestions in a virtual (laboratory-like) setting. Similar to systematic experiments in the real world, a simulation model can be used to investigate whether changes in model parameters or model structure improve model behaviour with respect to specified goals or thresholds. In order to do so, proposed changes of the real system are implemented in the model and then compared to the baseline behaviour. If several improvements are suggested, the one with the highest impact can be identified. In addition to that, the effect of combining several improvement suggestions can be analysed [11].

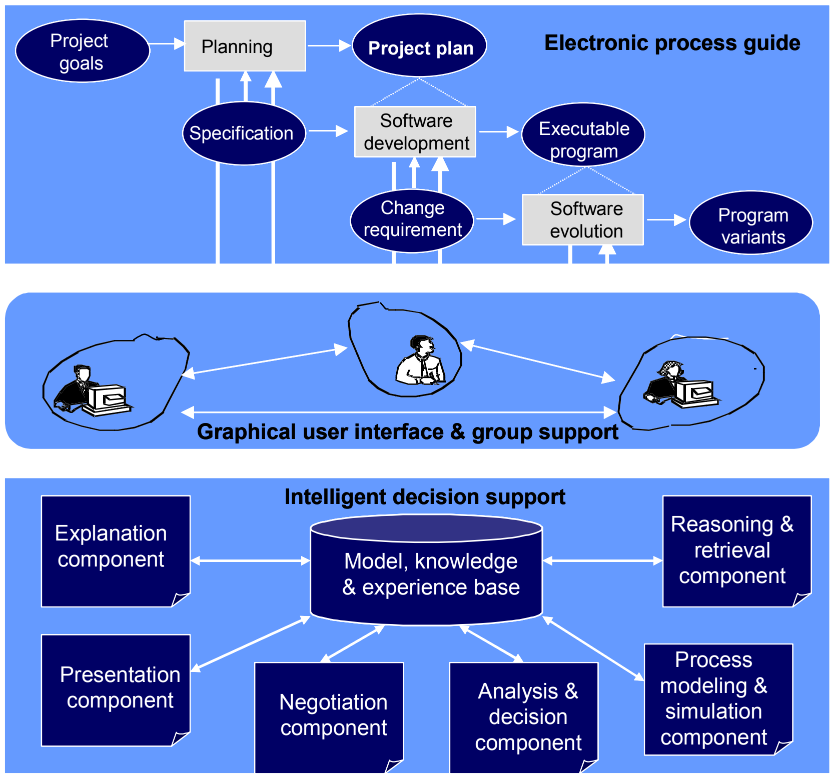

Figure 1. Principal Architecture of a Software Engineering Decision Support System.

The principal architecture of a SE-DSS is shown in Figure 1. Real-world decisions in planning, development or evolution processes in Software Engineering are done by humans. All support is provided via a graphical user interface. Experts and their human intelligence is integrated via group support facilities. The intelligence of the support is based on a comprehensive model, knowledge and experience. The more reliable and valid the models are, the more likely we can expect good support. The accompanying suite of components interacts with the model, knowledge and experience base. The suite encompasses tools for modeling, simulation, as well as decision analysis. Furthermore, intelligent components for reasoning, retrieval and navigation are added to increase efficiency and effectiveness of the support.

4. Decision Support in Requirements Negotiations

4.1 Decision Support for Requirements Negotiations

Defining, prioritising, and selecting requirements are problems of tremendous importance. In [16], a new approach called Quantitative WinWin for decision support in requirements negotiation is studied. The difference to Boehm's [4] groupware-based negotiation support is the inclusion of quantitative methods as a backbone for better and more objective decisions. Like Boehm's original WinWin, Quantitative WinWin uses an iterative approach, with the aim to increase knowledge about the requirements at each iteration.

The overall method uses the Analytical Hierarchy Process [19] for a stepwise determination of the stakeholders' preferences in quantitative terms. These results are combined with methods for early effort estimation, in our case using the simulation prototype GENSIM [10], to evaluate the feasibility of alternative requirements subsets in terms of their related implementation efforts. As a result, quantitative WinWin offers decision support for selecting the most appropriate requirements based on the preferences of the stakeholders, the business value of requirements and a given maximum development effort.

How can the hypotheses formulated in chapter 3 work in this case? We don't have a quantitative evaluation yet, but we can briefly discuss the main arguments supporting the formulated hypotheses. The comparison is between using a DSS and subjective decision-making without any (quantitative) tool support:

Hypothesis 1: Improved quality: Quantitative Win-Win is an evolutionary approach taking into account all the information available at that moment to find the most appropriate subsets of requirements. The underlying algorithms are well established. The quality of the solutions provided mainly depends on the quality of the input data and the input model. However, as the used solution algorithms are objective, results should be better than those based on subjective selection of requirements. In addition to that, the system will provide a set of candidate solutions. Among them, the actual decision-maker can chose from.

Hypothesis 2: Efficiency: Effort to generate solutions related to their quality is improved under the assumption that models and relevant data are available. This needs an upfront investment, especially to create the effort estimation based on simulation runs.

Hypothesis 3: Transparency: From the application of Quantitative Win-Win you will get a preference structure among all the solutions generated. The preference is a result of systematic and pair-wise comparison between stakeholder and the requirements class alternatives. The final selection of requirements can be exactly linked to the chosen preference structure.

Hypothesis 4: Stability: Stability of the chosen solutions can easily be checked by computation of the stability intervals. This is relatively easy because of the power of the underlying algorithms (as opposed to subjective judgements to evaluate solutions).

Hypothesis 5: Flexibility: Quantitative Win-Win allows for following scenarios and varying problem parameters. Any changes in effort estimates, priorities or other problem parameters can be easily investigated with Quantitative WinWin. This also enables the generation of new solution alternatives.

4.2 Decision Support for Release Planning in Incremental Software Development

To achieve higher flexibility and to better satisfy actual customer requirements, there is an increasing tendency to develop and deliver software in an incremental fashion. In adopting this process, requirements are delivered in releases. Thus, a decision has to be made on which requirements should be delivered and in which release. Three main considerations that need to be taken into account are the technical precedence constraints inherent in the requirements, the typically conflicting priorities as determined by the representative stakeholders, as well as the balance between required and available effort. The technical precedence constraints relate to situations where one requirement cannot be implemented until another is completed or where one requirement is implemented in the same release as another. Similarly, certain requirements should be implemented in the same release. Stakeholder preferences may be based on the perceived utility or urgency of delivered requirements to the different stakeholders involved.

A method called EVOLVE is presented in [6] for optimally allocating requirements to increments; Methodologically, it is relies mainly on genetic algorithms, the principles of incremental and evolutionary software process models and aspects of greedy algorithms and the Analytic Hierarchy Process. EVOLVE typically generates a small set of most promising candidate solutions from which the actual decision-maker can choose.

We briefly discuss the main contributions of EVOLVE in light of the above five hypotheses. The comparison again is between using a DSS and subjective decision-making without any (quantitative) tool support:

Hypothesis 1: Improved quality: The proposed planning problem is highly complex (NP-complete) and cannot be expected to be solved adequately by individual judgement and trial and error type methods. Even Greedy-type heuristics are not competitive in terms of quality.

Hypothesis 2: Efficiency: Effort to generate solutions related to their quality is much lower than for any other method

Hypothesis 3: Transparency: The underlying fitness score function of EVOLVE guarantees optimal balancing between different stakeholder preferences, and this makes the proposed transparency.

Hypothesis 4: Stability: Stability of the chosen solutions can be judged from the different runs of the evolutionary algorithm (eventually, with varying crossover and mutation rates).

Hypothesis 5: Flexibility: EVOLVE allows investigation of any changes in requirements, priorities or other problem parameters. This enables the generation of new solution alternatives. Furthermore, variations of the weighting parameter in the objective function result in offering a set of most promising candidate solutions.

5. Summary and Conclusions

There are very good reasons for offering support for making decisions at the various stages of software development and evolution. Most of the related problems are very complex including different stakeholder perspectives and constraints. Mostly, decisions have to be made under uncertainty and incompleteness of information. Nevertheless, making good decisions is of tremendous importance for developing software faster, cheaper and of higher quality.

Currently, there is an increasing effort not only to measure or model certain aspects of the development processes, but to go further and integrate all available data, information, knowledge and experience with a sound methodology to provide the backbone for making good decisions. This mainly includes searching for all the objectives and constraints that influence a decision as well as elaborating on the defined solution space for possible courses of action. Typically, the different courses of action are non-comparable because of the different involved perspectives and objectives.

This is exactly the borderline between offering decision support as addressed in the article, and real-world decision making selecting among a range of alternative solutions generated by the intelligent pre-processing steps of SEDS.

As characterized by [23], decision support is most appropriate for semi-structured and unstructured problems with emphasis on managerial control. What can be expected from decision support in the area of software engineering is higher decision quality; improved communication between all involved parties, increased productivity, time savings, and improved customer satisfaction. To achieve this goal, further effort should focus on (i) advancing SEDS methodology, especially by integrating aspects of decision-making under uncertainty, (ii) developing knowledge and experience-based software engineering decision support systems offering intelligent support on demand and via the web, (iii) further implementation and industrial evaluation of SEDS methodology and SE-DSS's, and (iv) evaluation of the underlying research hypotheses one to four describing the impact of SEDS on software development and evolution.

Acknowledgement

The author would like to thank the Alberta Informatics Circle of Research Excellence (iCORE) for the financial support of this research. Special thanks to the reviewers for their valuable comments that contributed to an improved version of the paper.

References

[1] K.D. Althoff, "Case-Based Reasoning", in: Handbook of Software Engineering and Knowledge Engineering (SK Chang, ed), Vol. 1, pp 549-588.

[2] V. Basili, G. Caldiera, D. Rombach, "Experience Factory". In: J. Marciniak: Encyclopedia of Software Engineering", Volume 1, 2001, pp 511-519.

[3] B.W. Boehm, "A Spiral Model of Software Development and Enhancement", IEEE Computer, 21 (5), pp. 61-72, 1988.

[4] B.W. Boehm, P. Grünbacher, B. Briggs, "Developing Groupware for Requirements Negotiation: Lessons Learned", IEEE Software, May/June 2001, pp. 46-55.

[5] L.C. Briand, J. Feng, Y. Labiche, "Experimenting with Genetic Algorithm to Devise Optimal Integration Test Orders", *Technical*

Report Department of Systems and Computer Engineering, Software Quality Engineering Laboratory Carleton University, 2002.

[6] L.C. Briand, K. El-Emam, B. Freimut, O.Laitenberger, “A comprehensive evaluation of capture-recapture models for estimating software defect content”, IEEE Transactions on Software Engineering, vol.26 (2000), pp 518-540.

[7] D. Greer, G.Ruhe, “Software Release Planning: An Evolutionary and Iterative Approach”, submitted to IST (2002).

[8] H.W. Hamacher, G. Ruhe, “On Spanning Tree Problems with Multiple Objectives”. *Annals of Operations Research 52(1994),* pp 209-230.

[9] J. Miller, F. Macdonald, J. Ferguson, “ASSISTing Management Decisions in the Software Inspection Process”, *Information Technology and Management,* vol.3 (2002), pp 67-83.

[10] D. Pfahl.: “An Integrated Approach to Simulation-Based Learning in Support of Strategic and Project Management in Software Organisation”. Ph.D. thesis, University of Kaiserslautern, Department of Computer Science, October 2001.

[11] D. Pfahl, G. Ruhe, "System Dynamics as an Enabling Technology for Learning in Software Organisations", 13th International Conference on Software Engineering and Knowledge Engineering. SEKE'2001, Skokie: Knowledge Systems Institute, 2001, pp 355-362.

[12] S. Pfleeger, “Making Goog Decisions: Software Development and Maintenance Projects*”, Tutorial at 8th IEEE Symposium on Software Metrics, 2002.*

[13] G. Ruhe, “Software Engineering Decision Support: Methodology and Applications”. Appears in: *Innovations in Decision Support Systems* (Ed. by Tonfoni and Jain), Springer 2003.

[14] G. Ruhe, "Intelligent Support for Selection of COTS Products", appears in: Proceedings of the Net.ObjectDays 2002, Erfurt, Springer 2003.

[15] G. Ruhe, “Learning Software Organisations”. In: Handbook of Software Engineering and Knowledge Engineering (S.K. Chang, ed.), World Scientific Publishing 2001, Vol 1, pp 663-678.

[16] G. Ruhe, A. Eberlein, D. Pfahl, “Quantitative WinWin - A New Method for Decision Support in Requirements Negotiation,

Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'2002), pp 159-166

[17] I. Rus, J.S. Collofello, "A Decision Support System for Software Reliability Engineering Strategy Selection", *Proceedings of the 23rd Annual International Computer Software and Applications COMPSAC 99*, Scottsdale, AZ, October 1999, pp 376-381.

[18] I. Rus, M. Lindvall, "Knowledge Management in Software Engineering, *IEEE Software* May/June 2002, pp 26-38.

[19] T.L. Saaty, "The Analytic Hierarchy Process", Wiley, New York, 1980.

[20] N.F. Schneidewind, "Software Quality Control And Prediction Model for Maintenance", *Annals of Software Engineering*, vol. 9 (2000), pp 79-101.

[21] SEWBOK. Guide to the Software Engineering Body of Knowledge. Version 0.95. IEEE Computer Society, May 2001.

[22] M. Svahnberg, C. Wohlin, L. Lundberg, M. Mattsson, "Quality Attribute Driven Selection of Software Architecture Structures", *Proceedings of the First Workshop on Software Engineering Decision Support, SEDECS'2002,* Ischia, pp 819-826.

[23] E. Turban, J.E. Aronson, "Decision Support Systems and Intelligent Systems", Prentice Hall, 2001.

[24] G. Visaggio, (2000) "Valued-Based Decision Model For Renewal Processes in Software Maintenance", *Annals of Software Engineering*, vol.9 (2000), 215-233.