Journal of Computer Science 6 (10): 1208-1211, 2010
ISSN 1549-3636

A Comparative Analysis of Software Engineering with Knowledge Engineering

[1]J. Frank Vijay and [2]C. Manoharan
[1]Department of Information Technology, Pannimalar Engineering College, Chennai, India
[2]VSA Group of Institutions, Salem, Tamil Nadu, India

Abstract: Problem statement: Software engineering is not only a technical discipline of its own. It is also a problem domain where technologies coming from other disciplines are relevant and can play an important role. One important example is knowledge engineering, a term that we use in the broad sense to encompass artificial intelligence, computational intelligence, knowledge bases, data mining and machine learning. We see a number of typical software development issues that can benefit from these disciplines and, for the sake of clarifying the discussion, we have divided them into four categories: (1) planning, monitoring and quality control of projects, (2) The quality and process improvement of software organizations, (3) decision making support, (4) automation. **Approach:** First, the planning, monitoring and quality control of software development was typically based unless it is entirely ad-hoc on past project data and/or expert opinion. **Results:** Several techniques coming from machine learning, computational intelligence and knowledge-based systems had shown to be useful in this context. Second, software organizations are inherently learning organizations, that need to improve, based on experience and project feedback, the way they develop software in changing and volatile environments. Large amounts of data, numerous documents and other forms of information are typically gathered on projects. The question then became how to enable the intelligent storage and use of such information in future projects. Third, during the course of a project, software engineers and managers have to face important, complex decisions. They need decision models to support them, especially when project pressure is intense. Techniques originally developed for building risk models based on expert elicitation or optimization heuristics can play a key role in such a context. The last category of applications concerns automation. Many automation problems, such as test data generation, can be formulated as constraint solving problems. A number of metaheuristic algorithms can be adapted for that purpose and have shown to be practically usable and flexible to adjust to numerous situations. **Conclusion:** This study discussed all the points above, identify open issues and future research directions and provide some carefully selected, key pointers for further reading.

Key words: Knowledge engineering, software engineering, hybrid method

INTRODUCTION

Planning, monitoring and quality control: Like any human-intensive production/engineering activity, software development needs reliable techniques to plan resource expenditures and monitor, assess and control product quality. More precisely, project expenditures need to be predicted and significant deviations need to be monitored. This requires the construction of accurate prediction models and heuristics to detect significant deviations and take remedial actions. With respect to prediction, a number of techniques coming from machine learning have shown to be useful. Examples are decision trees (Briand and Wuest, 2002) and rough sets (Harman and Jones, 2001).

The main advantages of these techniques can be described as follows:

- They can easily handle qualitative, categorical data, which are common in software engineering
- They produce models that are easier to interpret, which is important in our case as we would like to understand what factors affect software development productivity and quality
- They enable the discovery of certain structures in data sets, e.g., variable interactions in decision/regression trees

Computational intelligence, with techniques such as neural networks (Keung *et al*., 2004), can also play a role. Neural networks are good at building complex, non-linear prediction models. They do not require any assumption regarding the functional form of the relationships between predictors and the variable to be

Corresponding Author: J. Frank Vijay, Department of Information Technology, Pannimalar Engineering College, Chennai, India

predicted. However, their usage may be tedious (i.e., the training phase) and the interpretation of the resulting models difficult. This stems from the fact that it is difficult to deduce the type and form of relationships between variables from a neural network.

Fuzzy set theory has also been used to help with software engineering prediction models. The main motivation is that, as mentioned above, the data that prediction models rely on can be of qualitative and subjective nature (e.g., Team Cohesion cost driver in COCOMO II (Chulani *et al*., 1999). Fuzzy sets have been designed to deal with linguistic uncertainty and can help model the uncertainty associated with some of the subjective model parameters and input data which are elicited from expert opinion. In other words, when the user of prediction models have to provide qualitative values (e.g., categories) in input, fuzzy set theory can allow them to grant different levels of memberships to various categories, thus reflecting their uncertainty about the model inputs. Such uncertainty has, however, to be accounted for in the prediction model outputs.

Another interesting strategy that has been used in the context of quality and cost prediction models is Case Based Reasoning (CBR) (Vijay and Manoharan, 2009; Khoshgoftaar *et al*., 1995). The basic principle of CBR is to define a similarity function or measure and use it to retrieve similar projects to reuse their cost or quality data as a basis for prediction. However, it requires that a similarity function be defined beforehand. But in software engineering we are very often in a situation where we attempt to uncover trends from data and we are not a position to define such a similarity function. With respect to cost estimation, results have so far been rather disappointing (Briand and Wieczorek, 2001) and this result very likely stems from the difficulty to define an appropriate similarity function.

We have seen that many models (e.g., cost models) cannot, due to practical constraints, be built solely based on data (Briand *et al*., 1998). Therefore, eliciting expert opinion and modeling expert knowledge is sometimes key to developing prediction systems. Ideally, software engineering prediction models should combine expert opinion and project data. For example, the COCOMO II (Chulani *et al*., 1999) model is based in part of expert opinion. One important question is then how to integrate expert opinion and project data into common models. Techniques such as Bayesian analysis (Chulani *et al*., 1999) and expert opinion elicitation techniques combined with Monte Carlo simulation (Briand *et al*., 1998) have been used in the area of cost estimation. The latter technique has also been used for technology evaluation in the context of inspections (Briand *et al*., 2000a). We are in the process of developing a hybrid techniques which uses the concept of both software engineering and knowledge engineering (Vijay and Manoharan, 2009; Keung *et al*., 2004).

Software learning organizations: Within an organization, experience and knowledge acquired on past software projects can be used to improve practices on future projects. For example, it may be important to know whether a requirements engineering technique has worked well on past projects, what were the benefits and challenges, what the project participants felt should be done to improve the way it was used or automated. The main reason is that, in software engineering, it is difficult to know a priori whether a given technique or method will fit well with the problems at hand and existing practices. Corporate learning, based on experience, then becomes key to the effective adoption of new practices and productivity/quality improvement.

However, to achieve such an objective, best practices, lessons learned, models and data need to be made accessible and reusable across an organization. Different issues have to be addressed to make this possible:

- Technical issues: Data and documents need to be stored and retrieved in an efficient manner. Knowledge bases need to be designed and maintained and connected to the company intranet for corporate-wide accessibility. Security issues then arise as a result as some of the information may be confidential
- Organizational issues: Such knowledge bases need to be fed by projects. Data, information and documents need to be provided in a consistent form, based on agreed-upon structure and content. The information provided must be precise, accurate and complete. This requires a certain organizational discipline with procedures that are defined and enforced
- Cognitive issues: Users accessing such knowledge bases may be faced with tremendous amounts of information, most of it being irrelevant to the problem at hand. It is therefore important to reduce the cognitive load of the user by allowing him to retrieve, in an efficient and precise manner, relevant information

In this context, the design and maintenance of corporate wide knowledge bases then become a key issue to address.

Well-known and mature technologies exist to address the technical issues related to the design and maintenance of knowledge bases. The organizational issue has been addressed by the Quality Improvement Paradigm (Basili and Caldiera, 1995) and the Experience Factory Model. The Quality Improvement Paradigm (QIP) provides steps and guidelines about how an organization can go about improving itself based on project experience. The Experience Factory Model provides a model of corporate infrastructure that needs to be put in place to support the QIP.

Cognitive issues can be addressed by using techniques such as Case-Based Reasoning (CBR) (Vijay and Manoharan, 2009; Gresse *et al*., 2001) to retrieve relevant pieces of information in a knowledge base. For example, similar past projects can be retrieved based on a description of a new project and relevant lessons learned on various technologies and process issues can be retrieved, e.g., the usage of inspections. In this case a similarity measure between projects would need to be defined and, in practice, it would probably require the use of expert opinion. Furthermore, Incomplete data (e.g., project descriptions), the use of categorical variables and taxonomies (e.g., project types) and the use of various measurement scales are additional issues to address in defining similarity.

Numerous, complex decisions have to be made during software development and maintenance. For example, one may want to decide what should be the order of development and integration of components in a system (Briand *et al*., 2002), whether a given document needs further inspection before being approved (Briand *et al*., 2000b) and used for the next phases of development, or whether an inspection technique at a given stage of development is beneficial (Briand *et al*., 2000a). Such types of decisions are usually not trivial. They typically involve a certain level of risk and substantial resources are at stake.

Some of these decision problems can be reformulated as optimization problems. For example, the integration order problem above can be reformulated as a combinatorial optimization problem and techniques such as genetic algorithms or simulated annealing can help find near optimal solutions (Briand *et al*., 2002). The advantage of such metaheuristic techniques (Vijay and Manoharan, 2009), as they are referred to, is their flexibility. The objective function to be minimized is often to be tailored to specific situations. Such heuristics, as opposed to mathematical optimization techniques, enable such tailoring without changes to the optimization algorithms and automation. Furthermore, meta-heuristics allows us to solve complex, non-linear optimization problems that are not always addressable by conventional mathematical optimization techniques (Vijay and Manoharan, 2009). Their drawback tough is that there is no absolute guarantee such heuristics will provide near optimal solutions. Only case studies and experimentation can help us determine whether they are adequate for a problem and under which conditions.

Not all decisions can be formulated as an optimization problem. In some cases, the parameters that have a strong influence on a decision outcome are not known or can only be estimated with a certain level of uncertainty. This is the case of the inspection cost-benefit evaluation example mentioned above (Briand *et al*., 2000a). In general, to decide about using a technology, one usually needs to formulate a cost benefit model and possibly perform some simulation to account for the multiple sources of uncertainty in the model inputs and parameters (Briand *et al*., 2000a). However, in practice, even when carefully considering simplifying assumptions, such models depend on parameters that are not only unknown but specific to a particular development environment and for which we cannot collect data. Fortunately, there exists a large body of literature on expert estimation, which has been used, for example, in the nuclear industry to build risk models. Reported techniques have shown, under certain conditions, to be very useful to help estimate unknown model parameters.

Automation: Many activities in software engineering need to be automated so as to make methods and techniques economically viable. One good example is the generation of test data. In most cases, whether we refer to unit, integration, or functional testing, test strategies are defined based on coverage criteria, e.g., cover all control flow edges in a procedure. As a result, in many situations, generating appropriate test cases consists in finding test data that are compliant with a set of logical constraints, e.g., conditions determining the control of execution in a procedure. This exercise is very tedious and error-prone.

Fortunately, a number of research articles (Vijay and Manoharan, 2009; Pedrycz and Peters, 1998) have shown that metaheuristic techniques can also be used in this context. For example, based on constraints, an objective function can be defined in the context of genetic algorithms in order to ensure convergence of the algorithm towards acceptable input data. Initial results suggest this is feasible but more empirical investigations are however needed to determine the best ways to use those techniques and assess their limitations to address software engineering issues. Though many techniques are available and have been experimented with, software engineering problems provide new contexts in which to use them.

CONCLUSION

From the discussions above, we have seen that a wealth of knowledge engineering, artificial and computational intelligence techniques can be used to address a number of important software engineering issues. Though we have focused on techniques and problems on which we already have experience, it is clear that this study only scratches the surface. The potential for cost-effective applications in software engineering is enormous.

Expectedly, most of the techniques discussed here are based on heuristics. What this implies is that they can only be validated through experimentation and case studies. And they need to be investigated for each problem to be addressed and under realistic conditions. Only then we can determine whether they are applicable, economically viable and under which conditions this is the case.

It is therefore important not to fall into the trap of blindly using knowledge engineering techniques to arbitrary software engineering techniques. The well-known "hammer nail" dilemma should be avoided as it could lead to substantial waste of effort and negatively affect the perception that there is an important role to play for knowledge engineering in software development. The knowledge engineering community needs to make a conscious effort to understand the reality of software engineering challenges and technologies. In a similar way, software engineers need to get educated on the latest developments in computational intelligence, knowledge engineering, machine learning and hybrid techniques of estimation.

REFERENCES

Basili, V.R. and G. Caldiera, 1995. The Experience Factory: Strategy and Practice. 1st Edn., University of Maryland, United States, pp: 41.

Briand, L.C. and J. Wuest, 2002. Empirical studies of quality models in object-oriented systems. Adv. in Comput., 56: 97-166. DOI: 10.1016/S0065-2458(02)80005-5

Briand, L.C., K. El Emam and F. Bomarius, 1998. COBRA: A hybrid method for software cost estimation, benchmarking and risk assessment. Proceeding of the IEEE International Conference on Software Engineering, Apr. 19-25, IEEE Xplore Press, Tyoko, Japan, pp: 390-399. DOI: 10.1109/ICSE.1998.671392

Briand, L.C., B. Freimut and F. Vollei, 2000a. Assessing the cost-effectiveness of inspections by combining project data and expert opinion. Proceeding of the IEEE International Symposium on Softwear Reliability Engineering, Oct. 8-11, IEEE Xplore Press, San Jose, CA., USA., pp: 124-135. DOI: 10.1109/ISSRE.2000.885866

Briand, L.C., K.El Emam, B.G. Freimut and O. Laitenberger, 2000b. A comprehensive evaluation of capture-recapture models for estimating software defect content. IEEE Trans. Software Eng., 26: 518-540. DOI: 10.1109/32.852741

Briand, L.C., J. Feng and Y. Labiche, 2002. Experimenting with genetic algorithms and coupling measures to devise optimal integration test orders. Proceeding of the 14th international Conference on Software Engineering and Knowledge Engineering, July 15-19, ACM Pres, Ischia, Italy, pp: 43-50. DOI: 10.1145/568760.568769

Briand, L.C. and I. Wieczorek, 2001. Resource Modeling in Software Engineering. In: Encyclopedia of Software Engineering, Marciniak, J.J. (Ed.). Wiley-Interscience, USA., pp: 70-157.

Chulani, S., B. Boehm and B. Steece, 1999. Bayesian analysis of empirical software engineering cost models. IEEE Trans. Software Eng., 25: 573-583. DOI: 10.1109/32.799958

Gresse, C. von Wangenheim, K. Althoff and R. Garcia, 2001. Goal-Oriented and Similarity-Based Retrieval of Software Engineering Experienceware. In: Learning Software Organizations-Methodology and Applications, Ruhe, G. and F. Bomarius (Eds.). Springer, USA., pp: 80-120.

Harman, M. and B.F. Jones, 2001. Software engineering using metaheuristic innovative algorithms: workshop report. Inform. Software Technol., 43: 905-907. DOI: 10.1016/S0950-5849(01)00196-3

Keung, J., R. Jeffery and B. Kitchenham, 2004. The challenge of introducing new software cost estimation technology into a small software organization. Proceedings of the Australian Software Engineering Conference, (ASEC'04), IEEE Xplore Press, USA., pp: 52-59. DOI: 10.1109/ASWEC.2004.1290457

Khoshgoftaar, T.M., A.S. Pandya and D.L. Lanning, 1995. Application of neural networks for predicting program faults. Ann. Software Eng., 1: 141-145. DOI: 10.1007/BF02249049

Pedrycz, W. and J.F. Peters, 1998. Computational Intelligence in Software Engineering. 1st Edn., World Scientific Publishing Company, USA., pp: 485.

Vijay, J.F. and C. Manoharan, 2009. Initial hybrid method for analyzing software estimation, benchmarking and risk assessment using design of software. J. Comput. Sci., 5: 717-724. http://www.scipub.org/fulltext/jcs/jcs510717-724.pdf