Measure What Counts: An Evaluation Pattern for Software Data Analysis

Emmanuel Letier
Dept. of Computer Science
University College London
London, United Kingdom
e.letier@cs.ucl.ac.uk

Camilo Fitzgerald
Dept. of Computer Science
University College London
London, United Kingdom
c.fitzgerald@cs.ucl.ac.uk

***Abstract*—The 'Measure what counts' pattern consists in evaluating software data analysis techniques against problem-specific measures related to cost and other stakeholders' goals instead of relying solely on generic metrics such as recall, precision, F-measure, and Receiver Operating Characteristic area.**

I. Introduction

How we evaluate our software data analysis techniques drives what techniques we develop, how we optimise these techniques, and even what problems we dare to work on. Currently, the most common evaluation method consists in reporting generic information retrieval metrics such as precision, recall, accuracy, F-measures and the area under the Receiver Operating Characteristic (ROC) curve. This results in research that attempts to optimise these metrics. But are these the right metrics to optimise? And do they lead us to work on the right problems? We believe not. The perils of using these metrics as evaluation criteria has notably been expressed by Martin Shepperd at a recent workshop [1]. The purpose of this pattern is to show why and how we should replace such generic metrics by evaluation criteria that are specific to the software engineering activities we intend to support. It is a call to arms for the application of good requirements engineering techniques to discover, model, and analyse the stakeholders' goals, application context, and true requirements of software data analysis tools.

II. Pattern: Measure What Counts

A. Problem

The pattern is to be used when we want to evaluate how well a software data analysis method contributes to solving a specific software engineering problem, or when we want to compare how different analysis methods contribute to solving this problem. It should also be used when we need a sound method for identifying cut-off values for selecting an optimal trade-off between precision and recall.

The pattern is relevant when the data analysis method is considered in the context of providing decision support for a specifc software engineering problem. Examples are to provide support for prioritising testing activities based on which files are most likely to be error prone [8], or for prioritising requirements engineering activities based on which requirements have the highest risks of failures during and after development [2]. It is not applicable when the sole purpose of a software data analysis is to perform an empirical study without explicit intention to support software engineering decisions.

B. Solution

The solution is to develop a *goal model* [3] [4] that relates characteristics of the data analysis methods (such their precision and recall) to the goals of decision makers and other stakeholders of the data analysis tool. A frequent goal is to minimise cost where cost is some measure related to the costs incurred due to misclassification made by the data analysis tool [5]. Other goals may be about maximising value that the predictions bring to different stakeholders. In some cases, the goal model may lead to a multi-objective decision problem where data analysis methods are evaluated against a set of conflicting goals.

Let's illustrate this first with an example from another domain that is analogous to many software data analysis problems. Credit card fraud detection systems analyse credit card transactions to generate fraud alerts for cards they suspect have been compromised by fraudsters. These alerts are then investigated by human fraud inspectors who may decide to contact the cardholder and block the card. To design such system, it is essential to understand its stakeholders' goals and the context in which the system is to be used. Here, stakeholders' goals include minimising financial loss due to credit card fraud, minimising costs of the fraud investigation process, and minimizing inconveniences to cardholders [6], [7]. These goals are conflicting, yielding a multi-objective decision problem. Most importantly, the data analysis algorithm generating the fraud alerts is only one of the components that impact on these goals; other components include the bank procedures for handling potential frauds and the judgements made by human fraud investigators. A good credit card fraud detection system is one where all these components work together to optimise the goals. Optimising the fraud detection software with respect to rates of false alerts and missed frauds only, without understanding the stakeholders' real goals and the context in which the software is to be used is unlikely to yield an adequate system. A goal model for such a system defines equations that relates the stakeholders' goals to characteristics

Accepted for publication by IEEE. © 2013 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/ republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

of the fraud deteciton algorithm (its false alert rate, missed fraud rate, and timing of the alerts) and characteristics of its context of use (the proportion of fraud, the transaction amounts, the cost associated with investigating an alert, etc.). This model can then be used to evaluate and optimise fraud detection systems against these goals.

Software data analysis tools are similar. Optimising a software data analysis tool for precision and recall without understanding the context in which the tool is use and the stakeholders' actual goals for such system is unlikely to yield an optimal, or even adequate system. For examples, Arisholm et al. [8] have shown that optimising the recall and precision of fault-prediction methods is inadequate if we do not take into account the costs involved in inspecting and testing files predicted to be faulty . A goal model for a software data analysis tool will define a set of equations relating stakeholders goals to characteristics of the data analysis method and of its context of use. This model will then allow us to evaluate software data analysis tools against stakeholders' real goals rather generic measures such as precision and recall.

Developing and validating a goal model for a particular data analysis technique is not trivial and can become a research problem in itself. The guidance we provide here will remain at a high level only; the topic deserves a whole set of patterns, many of which still have to be discovered and tailored to the problems of software data analysis tools.

The whole field of system requirements engineering is devoted to techniques for discovering, modelling, and analysing stakeholders' goals and their relation to the domain context and software characteristics[4]. Among these techniques, there are numerous heuristics and formal patterns for transforming stakeholders' vague concerns into measurable goals and guiding the elaboration of goal models [3]. Although useful in practice, these patterns are not specific to software data analysis tools and their guidance is still relatively general. The field of decision analysis also offers useful practical guidance on how to construct appropriate measures and deal with uncertainties [9]. Empirical research techniques are also needed to inform and validate these models [10], [11].

The equations relating variables attached to different goals in the goal hierarchy can take different forms: simple equations over random or non-random variables, or probability tables of Bayesian Belief Networks. The relation between goal models and BBN is discussed in [3].

As initial high-level guidelines, the following steps are essential to good requirements engineering and may be useful for developing evaluation criteria for data analysis tools:

1) Identify stakeholders, i.e. all the persons and groups who have an interest in the data analysis tool and its impacts on software development process (there's often many more stakeholders than you think);
2) Identify the stakeholders' goals and define appropriate measures for these goals by focussing on what *observable changes* the stakeholders would like to see in their work;
3) Scope the area of software development you intend to support by drawing a context diagram showing the different agents involved (human and software) and their interfaces between themselves and with adjacent software engineering activities;
4) Within this scope, understand the process and context in which your tool will be used;
5) Develop and validate the model relating the stakeholders' goals to characteristics of the data analysis techniques and domain assumptions.

These steps are performed iteratively rather then sequentially. Each of these steps require much further explanation than what we can provide here (see [4] for a comprehensive descriptions of these activities). The last one in particular can become quite complex. Generally the best approach is to start simple by making strong simplifying assumptions about the problem context and simplifying the goals. A simple model with explicit and clearly justified simplifying assumptions is already preferable to using precision and recall alone. This model can then be critiqued and iteratively improved.

We wish we could provide detailed goal modelling patterns specific to software data analysis. Unfortunately, we currently lack detailed examples from which to infer these patterns. We hope that by highlighting the needs for such models and providing initial high-level guidance and pointers to general literature on how to elaborate them, we will soon see a much wider use of goal modelling and other requirements engineering techniques in the development and evaluation of software data analysis tools.

C. *Examples*

Although there are no detailed examples of the systematic application of the goal modelling steps outlined in the previous section, there are examples where alternative software data analysis methods are evaluated using a goal model rather than by relying on generic measures only.

Arisholm et al. apply this pattern to evaluate fault-prediction models against a measure of cost-effectiveness that takes into account the effort taken to inspect and test an implementation class predicted to be faulty [8]. Their cost model assumes that this effort is proportional to the size of the class. Even though this cost model is a rough approximation of real costs, it is more accurate than assuming equal cost for all classes (which is what previous evaluation criteria amount to). The paper shows that using this cost-effectiveness measure instead of precision, recall, and F-measure has a significant impact on which prediction methods are considered to be most effective. Mende and Koschke later use this cost-effectiveness measure to develop and assess cost-sensitive prediction models tailored to this evaluation criteria [12].

In our own work, we have applied this pattern to evaluate the value of early failure predictions in change request management systems [2]. This work compares classification methods intended to assist decisions about the amount of requirements analysis to be performed on a change request before assigning it to implementation or rejecting it. The classifiers analyse characteristics of change requests and their online discussions (eg. the length of the discussions, the number and roles of participants, the words used in the discussions) to predict which change requests are likely to result in problems later on in their development, such as an abandoned implementation, an integration problem, bugs associated to the change, or a removal of the change from the product shortly after its integration. To evaluate these classifiers, we have developed a simple cost-value based model of upfront requirements analysis activities that has for parameters the ratio of the cost of failure to the cost of additional upfront requirements analysis and the probability that additional analysis will effectively prevent subsequent failures from happening. Estimation for these parameters is *partially* supported by empirical evidences [13], but because there remain large uncertainties (and disagreements) about their values, our tool allows stakeholders to vary these parameters to reflect their own beliefs and compute the range of values for these parameters for which the failure predictions are sufficiently accurate to justify the decision to perform or not additional requirements analysis.

In both examples, the evaluation models make strong simplifying assumptions about the application domain and the models have not been validated empirically. However, they have the benefits of making their simplifying assumptions explicit and of providing clear justification why these evaluation criteria are adequate in the context of a particular software engineering problem. The models therefore provides the basis for further validation and refinement of evaluation criteria that reflect as closely as possible the actual stakeholders' goals.

D. Discussion

- While we presented this pattern as an evaluation pattern, one of its main benefits will be to influence the *design* of software data analysis tools that are optimised to the novel problem-specific evaluation criteria. Adoption of this pattern is likely to generate interest for cost-sensitive (and goal-sensitive) data analysis techniques that exploit knowledge of these evaluation criteria for generating optimal predictions.
- Possibly one of the biggest difficulty in applying this pattern will be to dispel the myth (wrongly attributed to Einstein) that "not everything that counts can be measured". This belief is mostly caused by misconceptions about the concepts, objects, and methods of measurement: Measurement is not about obtaining precise knowledge of a quantity; it is rather about reducing uncertainty about that quantity, however small that reduction of uncertainty might be [9].
- Another potential trap is to a priori exclude from the goal model soft factors and other variables that are judged to be too difficult to measure with precision. If we suspect that a variable has a high impact on stakeholders goals, we should include it in the model even if its values have high uncertainties. Reducing uncertainties about such a variable, even slightly, may proved to have much more impact on the stakholders' goals than further reducing uncertainties about a variable for which our prediction models already work well. For example, in the context of fault predictions, it may be more important to reason about fault severity rather than continuing to reduce uncertainties about fault locations irrespective of their potential severity. (By fault severity, we mean a measure of the real impact of a defect in the application domain rather than the label of that name in bug management systems.) The systematic development and validation of stakeholders' goal model will help identifying which reduction of uncertainties matters most to the software engineering problem we are interested in.

REFERENCES

[1] M. Shepperd, "Assessing the predictive performance of machine learners in software engineering." Presented as the 24th CREST Open Workshop, London, UK, 30-31 January 2013. [Online]. Available: http://crest.cs.ucl.ac.uk/cow/24/

[2] C. Fitzgerald, E. Letier, and A. Finkelstein, "Early failure prediction in feature request management systems: an extended study," *Requirements Engineering*, pp. 1–16, 2012.

[3] E. Letier and A. Van Lamsweerde, "Reasoning about partial goal satisfaction for requirements and design engineering," in *12th International Symposium on the Foundation of Software Engineering (FSE 2004)*, vol. 29, no. 6. ACM, 2004, pp. 53–62.

[4] A. Van Lamsweerde, *Requirements engineering: from system goals to UML models to software specifications*. Wiley, 2009.

[5] I. Witten, E. Frank, and M. Hall, *Data Mining: Practical Machine Learning Tools and Techniques, 3rd Edition*. Morgan Kaufmann, 2011.

[6] D. J. Hand, C. Whitrow, N. M. Adams, P. Juszczak, and D. Weston, "Performance criteria for plastic card fraud detection tools," *Journal of the Operational Research Society*, vol. 59, no. 7, pp. 956–962, 2008.

[7] W. Heaven and E. Letier, "Simulating and optimising design decisions in quantitative goal models," in *19th IEEE International Conference on Requirements Engineering (RE 2011)*. IEEE, 2011, pp. 79–88.

[8] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *J. Syst. Softw.*, vol. 83, no. 1, pp. 2–17, Jan. 2010.

[9] D. Hubbard, *How to measure anything: Finding the value of intangibles in business*. Wiley, 2010.

[10] R. Hoyle, M. Harris, and C. Judd, *Research methods in social relations*. Wadsworth Fort Worth, TX, 2002.

[11] C. Wohlin, P. Runeson, M. Host, C. Ohlsson, B. Regnell, and A. Wesslén, "Experimentation in software engineering: an introduction," 2000.

[12] T. Mende and R. Koschke, "Effort-aware defect prediction models," *2011 15th European Conference on Software Maintenance and Reengineering*, vol. 0, pp. 107–116, 2010.

[13] B. Boehm, "Architecting: How much and when?" *Making Software: What Really Works, and Why We Believe It. OReilly, Sebastopol, CA*, pp. 161–185, 2010.