Automated Assessment of Correctness of Recommendation Systems

Angela Lozano
Université catholique de Louvain
Louvvain-La-Neuve, Belgium
angela.lozano@uclouvain.be

Andy Kellens
Vrije Universiteit Brussel
Brussels, Belgium
akellens@vub.ac.be

Kim Mens
Université catholique de Louvain
Louvvain-La-Neuve, Belgium
kim.mens@uclouvain.be

***Abstract*—Using a concrete example, this position paper makes a case for evaluating the correctness of software recommendation systems in an *automated* way, *prior* to conducting user studies, in order to assess the validity of the results and ideal configuration of the system to be evaluated.**

I. Context

Recommendation systems have many acclaimed advantages such as promoting reuse by reducing the effort required to use third party code, increasing awareness of design decisions, providing code completion hints, and pointing out incorrect or incomplete code.

This paper relates our experience in evaluating Mendel [1][1], a recommendation system to support developers when extending or reusing object-oriented applications. In general, recommendation systems can be evaluated with respect to their usefulness, usability, or correctness. The *usefulness* of a recommendation tool is the degree to which it reduces the effort needed to perform certain development tasks, and is usually evaluated by asking developers to use the recommendation system while performing a certain task, and later discussing their opinions on the recommendations provided. *Usability* expresses how easy to use the recommendation tool is, and is usually evaluated via observational studies, sometimes accompanied by questionnaires. Finally, *correctness* evaluates how trustworthy a system is by measuring the degree to which the recommendations it proposes are correct (*precision*), as well as the amount of correct information it recommends (*recall*).

II. Issues evaluating recommendation systems

While user studies are most suited to assess the quality of recommendation systems, they are notoriously expensive to conduct. In addition to the difficulty of defining a realistic, non-biased empirical experiment, a major difficulty lies in finding a sufficiently *large and representative* set of developers that can serve as test subjects. These developers need to be willing to spend time learning to use the system, to answer questionnaires or interviews, and to participate in controlled experiments. Achieving such a level of commitment, especially in an industrial context, is challenging and usually requires prior proof of the merits of the tool.

In this paper we take the stance that providing prior insights on the correctness of a recommendation system, is required to gain sufficient confidence in the system before initiating a user study. The *correctness* of such a system can be expressed in terms of *precision*, *recall*, and *f-measure*[2]. While the success of a recommendation system depends on more than its correctness, correctness does play a crucial role. Recommendation systems with low precision present their users with a potentially large list of recommendations of which only a small subset are correct or useful. Systems with low recall might be of limited use, as they omit important recommendations.

It is our experience that assessing correctness of a recommendation system can often be done (semi-)automatically, without requiring costly user studies. By comparing the recommendations provided by the system with a kind of "gold standard", it is possible to measure precision, recall and f-measure. For example, recommendation systems that aid developers in using a certain API may consider the potentially large corpus of existing applications that use that API as the gold standard. However, such a gold standard can also be defined for other kinds of recommendation systems, such as our Mendel tool which aims at completing and improving consistency of a certain code base. In the remainder of this paper we present Mendel, our approach for evaluating correctness of Mendel automatically, and we discuss our experiences with this evaluation strategy.

III. Mendel

Mendel [1] is a recommendation system for completing object-oriented code bases during maintenance and extension tasks. Given a user's current browsing context, the tool proposes structural properties (e.g., relevant classes, messages to send, use of super calls, or particular source code templates) that may be missing from the class or method browsed. Mendel starts from the assumption that

[1] http://soft.vub.ac.be/mendel

[2] i.e., the overall balance between precision and recall

entities belonging to a same class hierarchy are likely to exhibit similar properties. It relies on a genetic metaphor: based on the class hierarchy to which a source-code entity belongs, the tool defines the "family" of a method as all implementors of that method's signature within sibling or niece classes. It then computes the *traits* (i.e. structural characteristics) of the entities of that family and reports on traits exhibited by most of the entities in the family, but not by the entity itself.

A. Evaluating Mendel

When developing Mendel we experimented with various definitions of "family" of source-code entities, different thresholds and different sets of structural properties taken into account by our approach. Initially we performed a *qualitative* evaluation of each configuration of the tool, by manually investigating the output to find interesting recommendations.

Our qualitative evaluation revealed that Mendel did not behave satisfactorily under all configurations. This discouraged us from attempting a user study right away: we were not certain which configuration of the tool would be most suitable, and wanted to avoid to have to repeat a costly user study multiple times. To overcome this problem, eventually we decided to automatically simulate the use of Mendel to test and compare various configurations of the tool. Below, we discuss how this *automated simulation* was implemented.

B. Automated simulation to evaluate Mendel

The idea behind our experimental set-up is to take the source code of a program, remove the implementation of a particular source-code entity, and then check whether or not Mendel's suggestions align with the original properties of that entity. In other words, we use the existing source-code as gold standard for the evaluation of the recommendation system, as has been proposed by various authors. Furthermore, this experimental set-up lies close to the intended usage scenario of Mendel, where a developer is performing a maintenance or extension task in which a new entity is added to the system, and where Mendel aids the developer in completing this new entity.

We implemented a small tool to fully automate this evaluation process:

1) As input, it takes the source code of a software system.
2) For each class and method in that system, it computes and stores the *traits* (i.e. structural characteristics) exhibited by those source-code entities.
3) It iterates over all source-code entities in the system:
   a) For a class, all of its methods are removed; for a method, its method body is removed.
   b) Mendel is used to suggest missing *traits*.
   c) These recommended *traits* are compared with the original *traits* of the entity.
   d) The original implementation of the entity is restored.
4) Finally, it generates a report summarizing the results of the validation. This report then serves as input for further analysis of the data.

IV. Discussion

As mentioned above, the lack of knowledge regarding which configuration of Mendel would perform best made us opt for an automated means to evaluate our approach. The main advantage of this strategy was that it allowed us to iteratively refine our approach: we were able to experiment with various configurations of Mendel, and for each configuration we were able to automatically compare and evaluate its correctness in an objective manner.

In addition to this advantage, we observed that a pure qualitative evaluation of our approach could be deceiving. Prior to applying our automated evaluation to each different configuration of Mendel, we verified manually whether the configuration yielded expected results for particular contexts. As such, we originally dismissed various configurations where an initial assessment of the recommendations produced by Mendel appeared to be disappointing. In a later stage, we automatically experimented with a broad range of configurations of the tool. We noticed that a number of configuration that were dismissed earlier by hand, actually performed rather well when applied to an entire system.

As such, only considering a small subset of the system in our manual investigations of the results skewed our qualitative-only evaluation. Note that it was precisely the fact that our evaluation could be automated which allowed us to experiment with Mendel on a larger scale than would be possible with a mere qualitative analysis or user study.

Finally, as a caveat on our automated evaluation approach, we can mention that using the latest version of the code as gold standard, may provide pessimistic results of the correctness of the tool. Recommendations that do not align with the current version of the source code could be considered incorrect, even though they might indicate an interesting missing property of the source code. However, we would need a user evaluation to assess to what extent the latest version of the code diverges from the developer's view on the usefulness of the recommendations.

Acknowledgment

This research is supported by the IAP Program of the Belgian State. Angela Lozano is funded as a post-doc researcher on a Belgian FNRS-FRFC project.

References

[1] A. Lozano, A. Kellens, and K. Mens, "Mendel: Source code recommendation based on a genetic metaphor," in *Int'l Conf. on Automated Software Engineering (ASE)*, 2011, pp. 384–387.