Report on proposed tools

Akash Dhasade
Department of Computer Science and Engineering
Indian Institute of Technology Tirupati
Roll No: TCS15B031

Kurma Sai Sree Bhargav
Department of Computer Science and Engineering
Indian Institue Of Technology Tiruapti
Roll No: TCS15B015

Vanka Sai Sumanth
Department of Computer Science and Engineering
Indian Institute Of Technology Tirupati
Roll No: TCS15B029

***Abstract*—This write-up presents proposed tools for Software Engineering Lab along with extensive study of existing tools in two areas: estimating release dates of open source software projects by applying machine learning techniques and tool for testing and debugging assignments in haskell. For the first bucket we've chosen to develop a tool which predicts the next release date of a software using machine learning techniques. The tool proposed for testing/debugging bucket is an improvement over the existing tool QuickEval wherein we plan to add Direct Automated Random Testing (DART) to the tool.**

***Keywords*—Software, Machine learning, predicting next release, Haskell, unit testing, random testing, coverage based testing.**

I. Introduction

Tool 1: Software cost/effort estimation and scheduling are critical in software life cycle model. They tell us about the feasibility of the project and answer whether it can be done within the budget and in time. A lot of research work has already been done on the software cost/effort estimation. Starting from Algorithmic models like COCOMO, SDC, TRW Wolverton, IBM-FSD to using Machine learning models like KNN, Artificial Neural Nets(ANN), back propagation ,Fuzzy Logic, Analogy Estimation etc... The models have been successful with each having its own pros. But not much work has been done on scheduling. Software cost/effort estimation mainly focuses on the boolean question whether the project is feasible or not where as scheduling focuses on the 'how' part, if feasible. In this paper we try to give an approach using machine learning to predict the next release date of the software project.

Tool 2: The process of evaluating assignments of students is much tedious process than perceived. One always wants the process to be (a) consistent for all solutions submitted (b) be less time consuming. It's also preferable that the teacher should try to evaluate all expressions/paths of the program including corner cases. Lack of testing tools and good debuggers is one of the many reasons why few people use Haskell [1]. The existing tool QuickEval serves the purpose by allowing teachers to generate a test suite that can achieve full coverage of the programs submitted by the students and enable assignment evaluation in less time. Though this being the case, the teacher has to manually generate test cases for uncovered portions of program once the coverage information has been generated. We plan to integrate DART technique, with which we can achieve **automatic** generation of new test input values to direct the execution of program to unexecuted paths, thereby making the tool completely automatic with no or very less user interaction.

The remainder of the write-up is structured as follows: Section II presents existing tools and research in the areas of predicting release dates of Open Source Software Projects. In Section III we propose our idea for release date prediction using machine learning techniques and explain the tentative plan. We then present the past work in the area of testing and debugging Haskell programs in Section IV and provide a comprehensive list of all tools developed till date. Section V discusses how we plan to extend QuickEval and Section VI concludes.

II. Tools survey: Predicting costs/release time of open source software projects

A. Empirical data modeling in software engineering using radial basis functions

In this paper, Shin and Goel [7] noticed that many empirical methods in the area of software engineering linear regression analysis. Hence they proposed the use of Radial Basis Function neural networks to provide a flexible way to generalize linear regression function and applied the same on NASA database for software effort estimation.

B. Can genetic programming improve software effort estimation?

The use of genetic programming in software effort estimation was proposed by Burgess and Lefley [6]. They compared their method applied to linear Least Square Regression, KNN and MLP neural network. When applied to Desharnias dataset, they showed that GP yielded satisfactory results but required more time in the experiments.

C. Estimation of software projects effort with support vector regression

In this paper[5], Oliveira proposed to apply Support Vector Regression (SVR) to software effort estimation. The simulations showed that SVR outperforms previous results reported in literature obtained with other models when applied to NASA database.

D. Software Effort Estimation using Machine Learning Techniques with Robust Confidence Intervals

This paper[8] introduced a method based on machine learning which gives the estimation of the effort together with a confidence interval for it. Simulations were made on two datasets: NASA dataset and Desharnias dataset. The simulation results showed that bagging was able to improve performance of the following regression methods in the effort estimation task: (1) SVR (2) MLP .

E. Predicting Release Time for Open Source Software based on the Generalized Software Reliability Model

In this work [9], Washizaki, Honda and Fukazawa proposed the use of a mathematical model named Generalized Software Reliability Model(GSRM) based on a stochastic process to predict the release time of Open Source Software Projects. In addition to this, they proposed a method to evaluate the accuracy of predicted release time. The results showed that GSRM shows 0.572% error rate which corresponds to a predicted release date of two days prior to actual release date.

The literature review reveals that no tool can be preferred over other and combinations of tools work the best in most cases. A lot of work has been done in the area of effort estimation of Open Source Software Projects using machine learning techniques. Not much work has been found in applying similar techniques for predicting release dates of Open Source Software Projects wherein the large amount of data available on github can be effectively used to train models. Existing work in this area shows only use of mathematical models for predicting release dates. Hence we propose to use Machine Learning Techniques to achieve the same as described in the next section.

III. Proposed Tool: Using Machine learning techniques for predicting release date of software

A. Feature Engineering

What are the features/inputs that dictate the next release date or more accurately the time it takes for the next release? To understand that, we need to first know what happens when a new version of a software is released. Major things that happen during a software update are bug fixes, addition of new features, performance improvements, design enhancements, support for the latest technologies, database expansions etc.. These are the broad categories of what happens in a software update. Each of which has its own impact on the time it takes for the next update. Not only this, the time required for the next update also depends upon the software life cycle model, the team follows. For agile life cycle models the next release might be in 2-3 weeks, where as in the case of systematic models like incremental models, spiral models etc.. it may take longer. Each of these issues take their own time to get addressed for example different bugs take different spans to get fixed. Different features take different effort/time to be incorporated etc. There may be several other factors like the work experience of the team etc.. We need to take into account all these features while predicting the next release date of a software.

B. Data Collection:

Data can be acquired by crawling github with python scripts. We can acquire data like number of bugs fixed, different tags describing the type of the Bugs, the number of commits, the number of contributors, the code frequency, number of lines added/deleted etc..

C. Machine learning model:

Regarding what machine learning technique to be used, there are various choices. We could use a simple model like KNN and it may work flawlessly, that's what the literature review says. We can also use complicated models like neural nets. Each model has its own pros and cons.

IV. Tools survey: Testing/debugging Haskell programs

A. QuickCheck

QuickCheck is a tool which aids the Haskell Programmer in formulating and testing properties of programs. Properties are described as Haskell functions, and can be automatically tested on random input, but it is also possible to define custom data generators. There are two drawbacks of QuickCheck. First, with random testing it is easy to catch bugs that are highly likely to occur, but it is very difficult to catch bugs that occur very infrequently. Secondly, it's not always possible to define properties that hold for functions to ensure it's correctness. It is worth noting that QuickCheck shows random test case only when the assertion fails to hold for that random test case.

B. Haskell Program Coverage (HPC)

HPC is a tool-kit to record and display Haskell Program Coverage. HPC includes tools that instrument Haskell programs to record program coverage, run instrumented programs, and display information derived. HPC presents coverage information in two forms: highlighting of sources and summary statistics. The drawback of HPC is that it does not provide the feature of random generation of input.

C. HUnit

HUnit is a unit testing framework for Haskell. HUnit is an adaptation of JUnit to Haskell, wherein you can easily create tests, name them, group them into suites, execute them, with the framework checking the results automatically. Tests in HUnit are specified compositionally and assertions are combined to make a test case, and test cases are combined into tests. As mentioned earlier, it's not always possible to specify assertions or properties that hold for functions thus restricting the use of HUnit over all applications.

D. Hat

Hat is a source level tracer for Haskell. Hat gives access to detailed, otherwise invisible information about a computation and helps locating errors in programs. Hat allows you to run your programs such that it additionally writes a trace to a file. After the program has terminated one can view the trace with a browsing tool. The trace consists of high-level information about the computation and thus can be used for debugging.

V. Proposed Tool: Improvement/ Extension of QuickEval

The proposed usage of QuickEval, though not limited to, was to aid teachers in evaluating assignments. It's a testing tool which helps in understanding the behaviour of a program using random test generation along with user-specified tests. Current version of QuickEval works as follows -

1) The user specifies an option: 'r' - random test generation, 'i' - user specified input.
2) With random test generation, QuickEval generates specified number of random tests for the specified function, the output of evaluating the function over the inputs being shown on the screen.
3) Secondly, it shows the coverage information for the function under the tests that have been run. This is very similar to output shown by HPC.
4) Knowing the unreached or unexecuted parts of the function, the teacher is supposed to enter an appropriate input to make the program execution reach unevaluated part of the program using the option 'i'.

The makers of QuickEval have left the integration of Direct Automatic Random Test Generation (DART) as a proposed extension to the tool. DART is a testing technique used for directed automatic testing of programs. It works in three phases whereby the first two phases have been already implemented in QuickEval. The third and last phase is to achieve automatic generation of new test input values to direct the execution of the program to unexecuted paths. Improving this part will make QuickEval fully automatic and will require no or very less user interaction. This being one of the major proposed improvement in QuickEval, we further plan to automate assignment evaluation such that the evaluation report for all students is saved directly in an excel sheet. The tool would clearly report the mistakes made by students and thus function as an auto-grader.

VI. Conclusion

This write-up touched upon the existing tools and research in two areas: estimating release dates for open source software projects using machine learning techniques and testing/ debugging of Haskell programs. Various options for design and various features to be taken into consideration to make a prediction on the next release date have been described. We briefly described four tools for testing/ debugging Haskell programs namely, QuickCheck, HPC, HUnit and Hat. We then discussed their drawbacks and showed how QuickEval overcomes them by combining the features of HPC and QuickCheck. Finally, we explained how the addition of DART technique would reduce the manual work and also proposed few other additions to automate evaluation of assignments.

References

[1] Predicting Release Time for Open Source Software based on the Generalized Software Reliability Model Hironori Washizaki, Kiyoshi Honda and Yoshiaki Fukazawa

[2] Software Effort estimation using Machine Learning techniques Monika, Om Prakash Sangwan

[3] An Assessment and Comparison of Common Software Cost Estimation Modeling Techniques Lionel C. Briand, Khaled El Emam Dagmar Surmann, Isabella Wieczorek

[4] Estimation of the COCOMO Model Parameters Using Genetic Algorithms for NASA Software Projects

[5] A. L. I. Oliveira, Estimation of software projects effort with support vector regression. Neurocomputing, vol. 69, no. 13-15, pp. 1749-1753, August 2006.

[6] C. J. Burgess, M. Lefley. Can genetic programming improve software effort estimation? A comparative evaluation. Information and Software Technology, vol. 43, pp. 863-873, 2001.

[7] M. Shin, A.L. Goel, Empirical data modeling in software engineering using radial basis functions. IEEE Transactions on Software Engineering, vol. 26, no. 6, pp. 567-576, June 2000.

[8] Petrnio L. Braga and Adriano L. I. Oliveira and Silvio R. L. Meira, "Software Effort Estimation using Machine Learning Techniques with Robust Confidence Intervals" in *19th IEEE International Conference on Tools with Artificial Intelligence*

[9] Hironori Washizaki, Kiyoshi Honda and Yoshiaki Fukazawa, "Predicting Release Time for Open Source Software based on the Generalized Software Reliability Model."

[10] Philip Wadler. Why no one used functional languages. SIGPLAN Notices, 33(8): 23-27, 1998.

[11] Koen Claessen and John Hughes. Quickcheck: a lightweight tool for random testing of Haskell programs. In *ICFP*, pages 268-279, 2000.

[12] Andy Gill and Colin Runciman. Haskell program coverage. In *Haskell 2007*, pages 1-12, 2007.

[13] Patrice Godefroid, Nils Klarlund, and Koushik Sen. Dart: directed automated random testing. In *PLDI 2005*, pages 213-223, 2005.

[14] HUnit. Haskell Unit Testing. https://hackage.haskell.org/package/HUnit-1.2.2.1#readme Jan 2018 (Last visited)

[15] QuickEval: An interactive tool for coverage based testing of Haskell programs. Subhash P. Kale, supervised by Dr.Amey Karkare.