Tamas Sztanka-Toth
Homerton College
ts579

Computer Science Part II Project Proposal

Cached Bug Prediction for Python repositories on GitHub

23/10/2015

Project Originator: Professor A. Blackwell and Mr A. Sarkar

Resources required: See relevant section.

Project supervisor: Mr A. Sarkar

Signature:

Director of studies: Dr B. Roman

Signature:

Overseers: Professor J. Crowcroft and Dr. T. Sauerwald

Signature:

Introduction and Description of the Work

Testing and debugging has always been one of the most important parts of Computer Science, especially in Software Development. As projects and repositories grow bigger and bigger every day, more and more time is spent on writing unit-tests, debugging and code review. However, even with the most advanced testing tools, review techniques there still will be bugs and errors, which will only be found after deployment, hence bug prediction algorithms have gained importance. The aim of this project is to implement and investigate one of such algorithms, FixCache [1], and use it to predict bugs in Python repositories on GitHub. Previous implementations were used on C, C++, Java and Javascript projects, rather than Python, and they were mostly mining softwares repositories with different version control system: Subversion and CVS [1][2]

There are several approaches to bug-prediction to date, which rely on different information, and they can be put into two main categories: Change-log approaches and Single-version approaches [3]. Ultimately, all of them try to achieve the same results: to identify files, classes, functions which more contain a bug more likely than others. The Single-version approaches use the current state of the repository and calculate variety of metrics to predict faulty behavior. The most commonly used metric is the so-called Chidamber and Kremmer metrics suite [4]. In contrast, the Change-log approaches look at how the repositories evolved over time, and try to predict where the future defects will be. Hassan's algorithm calculates the complexity of each change and file to calculate the entropy of changes which then is used for bug-prediction [5]. The FixCache algorithm, on the other hand, approaches this problem by looking at the history of a repository and by identifying which commits have introduced a bug, and which of them have fixed it. Then, using the history of the repository, the algorithm can infer a set of bug-prone files.

Although this algorithm meant to work on every repository independently of language, I could not find any research which specifically focused on Python project (the vast majority of implementations was run on C, C++ and Java projects). As a result, my project will primarily focus on running FixCache on open-source, public, Python projects found on GitHub.

Resources Required

No special resources required. This project will entirely require the use of my personal laptop and the resources available on the MCS machines. The project will be implemented in Python.

Starting Point

- Part IA Object Oriented Programming course, Part IA Algorithms course.
- Summer internships at NRICH (after Part IA) and RealVNC (after Part IB), during both worked on projects which required advanced knowledge of Python.

Success Criteria

For the core of the project:

- To implement the FixCache algorithm in Python.

- To apply the FixCache algorithm to open-source, public, Python repositories found on GitHub and compare the results found with other implementations and repositories (eg., Java, C, C++) and deduce any language specific differences identified.

Possible extensions:

- Modify the algorithm using ideas from other bug-prediction approaches, in order to increase the correctness and performance.

Substance and Structure of the Project

1. Preparation: Background reading on bug-prediction techniques, approaches, especially technical details about FixCache algorithm.

2. Implementation: Start off by implementing the the bug-commit module which can be thought of as a parsing module, in a sense that it will prepare the data for FixCache. It will responsible for: (i) identifying when did bug-fixes occur, and which files were associated with them, (ii) during which commits were these bugs introduced, (iii) finally to sort the bug-introducing commits. Once this module is done, the core algorithm can be implemented, using data which was created by the bug-commit module.

3. Evaluation: This can be broken into two parts.

    1. Firstly, it is essential to evaluate whether the algorithm is working correctly. That is whether the files found by it, are really those which are more bug-prone than others. To evaluate this, a simple evaluation-module will be written, which will work as follows: given two points in the time, T (some point in the past) and N (present), the module will run the algorithm up until T, and then it will check whether the files in the list returned are present in any of bug-fixing commits between T and N.

    2. Secondly, the results found can be compared with previous research, and also with implementations ran on other languages' (C, C++ and Java) repositories.Testing and Improvement: Test the algorithm on several repositories, and try improving it.

4. Testing and Improvement: Test the algorithm on several repositories, and try improving it. The algorithm will be tested on the repositories listed above, in chronological order. The first repository was chosen as it has below 500 commits, so it will suffice when testing the commit parsing and bug-identifying parts of the algorithm. The latter three were picked randomly from the top thirty most starred Python repositories found on GitHub.

    1. https://github.com/pythonforfacebook/facebook-sdk

    2. https://github.com/boto/boto3

    3. https://github.com/boto/boto

    4. https://github.com/django/django

5. Write up the dissertation.

Timetable and Milestones

Week 1-2 (October 10 - October 23)

Contact project supervisor and arrange a meeting. Discuss the project ideas and start reading about bug-prediction, FixCache algorithm in particular. Write the Draft Proposal, and send it to the Overseers. Write the Project Proposal and submit it.

Milestones: Project Proposal submitted, project identified.

Week 3-4 (October 24 - November 6)

Research more deeply into the FixCache algorithm, especially the techniques to identify commits which introduce bugs into repositories. Identify any implementation differences for Python repositories.

Milestones: Theoretical background sufficient to start the implementation.

Week 5-6 (November 7 - November 20)

Implement the bug-commit identifying module of the algorithm, and test it if it works correctly on both small and large projects. Use this module to sort the bug-introducing commits in time.

Milestones: Bug-commit identifying module implemented.

Week 7-8 (November 21 - December 4)

Implement the core algorithm using the bug-commit module implementation from before. Run the algorithm on both big and small repositories and store the results in a structured manner.

Milestones: first version of FixCache algorithm implemented and working.

Week 9-10 (December 5 - December 18) - *Christmas Vacation*

Write the evaluation module for the algorithm, which will be used to empirically test how effective the algorithm is when identifying bug-prone files.

Milestones: Evaluation module implemented.

Week 11-14 (December 19 - January 15) - *Christmas Vacation*

Test the implementation using the evaluation module, and see if it works correctly. Compare the results with results found by other implementations, ran on other languages' repositories. By modifying the parameters of the algorithm, see how the results change accordingly. A short vacation is also planned in this period.

Milestones: Tested implementation, comparisons with other results made.

Week 15-16 (January 16 - January 29)

Prepare the 'Progress Report' and prepare a presentation about the work done so far.

Milestones: Progress Report submitted, presentation created.

Week 17-18 (January 30 - February 12)

Try modifying the algorithm in order to increase performance. Identify any possible language specific findings, differences (as a results of this implementation or of comparing it to other implementations).

Milestones: investigated the possibility of modifying the algorithm in order to increase performance.

Week 18-19 (February 13 - February 26)

Start writing up the project, starting with the 'Preparation' and 'Implementation' chapters of the dissertation.

Milestones: draft 'Preparation', 'Implementation' chapters written

Week 20-21 (February 27 - March 11)

Finish the 'Preparation' and 'Implementation' chapters, and start writing up the 'Evaluation' chapter.

Milestones: 'Preparation', 'Implementation' chapters finished.

Week 22-23 (March 12 - March 25) - *Easter Vacation*

Finish the 'Evaluation' chapter, write 'Conclusions' and 'Introduction'. Send over the first draft dissertation to supervisor, overseers and DoS.

Milestones: first draft dissertation completed, and sent to the supervisor.

Week 24-26 (March 26 - April 15) - *Easter Vacation*

Improve the dissertation quality relying on comments made by the supervisor, overseers and DoS. During this period revision for the exams will also be done.

Milestones: second draft dissertation completed.

Week 27-30 (April 16 - May 13)

Final four weeks. This serve as a buffer period for any unexpected difficulties. Complete the dissertation, send it over to supervisor, overseers and DoS, and submit it at least two weeks before the deadline (Friday the 13th of May). Revision for exams is continued in this period.

Milestones: Dissertation submitted.

Bibliography

1: S. Kim, T. Zimmermann, E. J. Whitehead, Jr., A. Zeller; *Predicting Faults from Cached History*; ICSE '07 Proceedings of the 29th international conference on Software Engineering, 2007

2: C. Sadowski, C. Lewis, Z. Lin, X. Zhu, E.J. Whitehead, Jr.; *An empirical analysis of the FixCache algorithm*; Proceedings of the 8th Working Conference on Mining Software Repositories, 2011

3: M. Lanza, R. Robbes; *An extensive comparison of bug prediction approaches*; Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on, 2010

4: S.R. Chidamber, C.F. Kemerer; *A metrics suite for object oriented design*; IEEE Transactions on Software Engineering, 1994

5: A. E. Hassan; *Predicting faults using the complexity of code changes*; Proceedings of the 31st International Conference on Software Engineering, 2009

Resource Declaration

Hardware

- My personal computer: Samsung Ultrabook series 5 (256 GB SSD, Intel Core i3 1.90GhZ x4, 4GB DDR3 Memory) for code development and writing dissertation. I accept full responsibility for this machine and I have made contingency plans to protect myself against hardware and/or software failure.
- MCS machines as back-up, in case my personal computer fails.
- Back-up storage: External HDD with 500GB, plus a Git repository.

Software

- Sublime Text 3 editor for development.
- PyCharm IDE as a back-up for development (with student account).
- GitHub repository for version control and back up (private repository with student account).