Kickoff, A Tool For Software Companies To Control Their Labor Cost Safely

Zhu Yue Cai

ENCS 282 YA/ Assignment 5 Research Proposal

Instructor: Dr. Brandiff Caron

December 1, 2014

Abstract

This document gives some ideas about how to write a project proposal, and provides a template for a proposal. You should discuss your proposal with your supervisor.

Table Of Contens

Letter of Transmittal

1. Body of Proposal

1.1 project definition

Laying off developers and recruiting new joiners are two common software company activities. But Previous research has shown that high turnover rates leads to a decrease in productivity [4]. Like other jobs that involve knowledge workers, when a software developer leaves, there is a knowledge gap. This gap is manifested as abandoned files on the software system that can be diffcult to maintain.

Generally, these abandoned files are taken over by new joiners or remain developers. There is a large literature on mentoring and integrating new developers into software projects. For example, Zhou and Mockus examined the impact of development environment on new developers[8] . Bird et al. [1] looked at the survival rate of new developers. Zhou and Mockus examined the amount of time until a developer becomes productive. Mockus [5] suggested mentors for developers based on past work and Canfora et al. [2] suggested mentors based on the email communication network. The experienced developers who have the knowledge to the abandoned files will help reduce the damage. They can take over the abandoned files themselves or mentor the new joiner to do so.

Therefore We believe that a software which can calculate the number of abandoned files and produce a list of experienced developers who have the knowledge to those files for a given layoff list is very helpful. Moreover, If this software could tell the same information for all the possible layoff lists based on input critical variables, the decision process would be simplified dramatically. This is the motivation of Kickoff. It is a useful management tool for software company to manage its knowledge storage.

1.2 Methodology

In order to minimize the cost, we will use only opensource tool for the development. And what we need is only "PostgreSQL"as our database.

In computer software engineering, revision control, or version, is any kind of practice that tracks and provides control over changes to source code. Software companies sometimes use version control software to maintain documentation and configuration files as well as source code. The first step of building Kickoff is to develop a mechanism to extract the source code change history from their version control software. Nowadays, a large number of software companies use "Git" as their version control system which is also opensoucre. Therefore "Kickoff" should be able to extract information from "Git" at the beginning. Since "Git" has a function called blame can identify changes and who made the changes in your project, we only need to input the result in the database. And this can be done by string processing technique and the psql command. We only need to integrate this into the software.

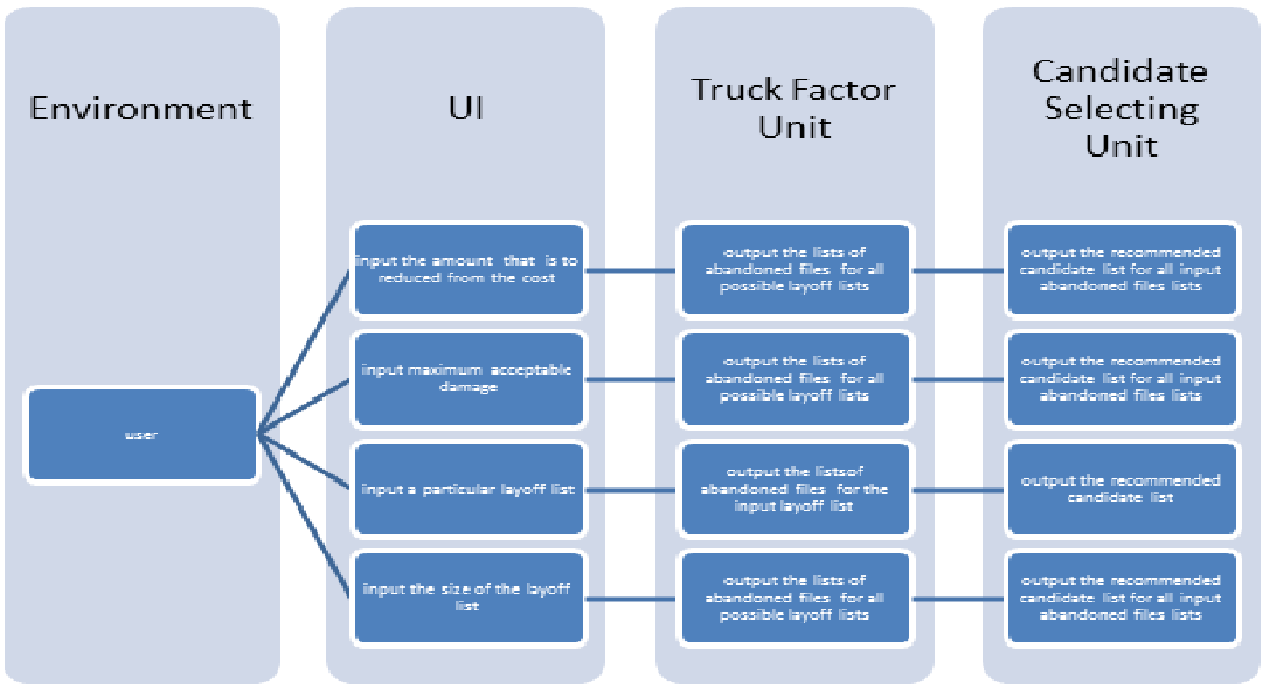
With the database ready, It is easy to calculate the number of abandoned files for a single given layoff list or for all possible lists of a given size. But it is very difficult to calculate the number for all potential layoff lists. We can use the "truck factor" algorithm proposed by

Ricca et el [6] for the job. In software development, the truck factor is the number of key developers who would need to be incapacitated to make a project unable to proceed. It is also known as bus factor, or lottery factor and is a measurement of the concentration of information in individual team members. The concept of truck factor is not related to the project, but the algorithm to calculate it is what we need. However, as Ricca et el pointed out[6], the time complexity of this algorithm is too large to implement in the industry. Therefore, optimizing this algorithm is the first problem we need to solved. We introduced a stopping condition to optimize the naive algorithm. What it does is to determine whether the uncalculated lists will have a number of abandoned files greater than the input value. If no, it will stop the program and save time. Then with the user input their maximum acceptable abandoned files amount, The program will work only for the layoff lists that satisfies the condition. For more detail about the optimzied truck factor algorithm, please refer to Appendix A.

Even with this optimization, the algorithm is only pratical to company with less than 100 developers. Further research on how to apply parallel computing technique in the optimzied truck factor algorithm is requried, such that it is also pratical large software companies.

The second problem is how to integrate previous researches to generate an algorithm for selecting the experienced developers for each abandoned file. We introduce a successor candidate selecting algorithm for this functionality. The general idea of this algorithm is to pick out those developers who have worked with the files that have co-changes with the abandoned files as candidates. The intuition behind is that if file A has changed with file B, and file A becomes abandoned, then the developer who works on file B will likely know something about file A. The idea of this algorithm comes from the matrix multiplication in Cataldo et al.'s[3]

With all these function units prepared, the conceptual structure of Kickoff is shown in figure.


Environment
UI
Truck Factor Unit
Candidate Selecting Unit
user
input the amount that is to reduced from the cost
input maximum acceptable damage
input a particular layoff list
input the size of the layoff list
output the lists of abandoned files for all possible layoff lists
output the lists of abandoned files for all possible layoff lists
output the listsof abandoned files for the input layoff list
output the lists of abandoned files for all possible layoff lists
output the recommended candidate list for all input abandoned files lists
output the recommended candidate list for all input abandoned files lists
output the recommended candidate list
output the recommended candidate list for all input abandoned files lists

1.3 Conclusion

It is harmful for a software project when there are some developers leaves. The damage is manifested as abandoned files on the software system that can be diffcult to maintain. We beleive that this damage can be reduced by finding experienced developers who have worked with the files that have co-changes with the abandoned files before to take over the abandoned files or to mentor new joiners to do so. Therefore we propose to apply the optimized truck factor algorithm to find out the abandoned files, and the candidate selecting algorithm to find out the experienced developers to build up the software Kickoff. With Kickoff, a software company can understand how harmful a particular layoff list is and how to reduce the damage. With Kickoff, a software company can also obtain information about all possible layoff solution by inputing the critical variable. Kickoff will be a very useful tool in software company management. In this proposal we've already scratched out the concepture structure of it.

2. SCHEDULE

3. BUDGET

4. QUALIFICATIONS

5. PRESS RELEASE

KICKFF HELPS REDUCE LABOR COST BY PROVIDING A SAFE DEVELOPER LAYOFF SOLUTION FOR SOFTWARE COMPANIES.

Concordia University,November 19, 2014 TODO

A group of Concordia University students have developed a software application called Kickoff to help software companies control their labor cost. Based on the desired salary cut off amount, it can give out a programmer layoff list that is the least harmful to the software project and an abandoned source code take over solution.

Laying off developer is dangerous to software companies. Since the part of project files maintained only by the layoff developers will become unknown to the company. If there was a bug reported in those files, nothing could be done until the remain developers learned how to maintain them. The learning process is time consuming, but the customer may select other product if the bug could not be fixed in a short time.

However, In order to survive in a critical financial situation, software companies sometimes have to lay off developers to reduce the labor cost. Laying off senior developers cuts off more cost but lose more project files than laying off junior developers. Therefore deciding the layoff solution is difficult for software companies.

Kickoff first generates a set of possible solutions according to the target cutoff cost, then applies the "Optimized Truck Factor" algorithm to find out the least harmful solution and uses the "candidate finding system" to generate the project remedy solution.

With Kickoff, it is easier for a software company to cut off their labor cost.

References

[1] Bird, C., Gourley, A., Devanbu, P., Swaminathan, A., and Hsu, G. Open borders? immigration in open source projects. In *MSR: Proceedings of the Fourth International Workshop on Mining Software Repositories* (2007), IEEE Computer Society, p. 8.

[2] Canfora, G., Di Penta, M., Oliveto, R., and Panichella, S. Who is going to mentor newcomers in open source projects? 44.

[3] Cataldo, Marcelo an dWagstrom, P. A., Herbsleb, J. D., and Carley, K. M. Identification of coordination requirements: implications for the design of collaboration and awareness tools. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work* (NewYork,NY,USA, 2006), CSCW'06, ACM, pp. 353–362.

[4] Guthrie, J. P. High-involvement work practices, turnover, and productivity: Evidence from new zealand. *Academy of management Journal 44*, 1 (2001), 180–190.

[5] Mockus, A. Organizational volatility and its effects on software defects. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering* (New York, NY, USA, 2010), FSE '10, ACM, pp. 117–126.

[6] Ricca, F., Marchetto, A., and Torchiano, M. On the difficulty of computing the truck factor. In *Proceedings of the 12th International Conference on Product-focused Software Process Improvement* (Berlin, Heidelberg, 2011), PROFES'11, Springer-Verlag, pp. 337–351.

[7] Torchiano, M., Ricca, F., and Marchetto, A. Is my project's truck factor low?: theoretical and empirical considerations about the truck factor threshold. In *Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics* (New York, NY, USA, 2011), WETSoM '11, ACM, pp. 12–18.

[8] Zhou, M., and Mockus, A. Does the initial environment impact the future of developers? In *Proceedings of the 33rd International Conference on Software Engineering* (New York, NY, USA, 2011), ICSE '11, ACM, pp. 271–280.

The Optimized TF Algorithm

The ‘truck factor’ is a measure in the agile community of the number of developers that must leave (EX: get hit by a truck) before the project fails [7]. Previous work has calculated the maximum number of files that are lost when a group of developers leave. Since this calculation has a $O(n!)$, they are able to examine only small projects at one single point in time. We contribute a stoping condition that allows us to calculate the maximum loss quickly, so that we can examine the truck factor on large projects. We prove that we attain the optimal solution and show how the truck factor for Chrome and Linux changes over time. We compare the maximum loss to the actual loss that Chrome and Linux have experienced suggesting how likely they are to suffer from similar turnover events.

The naive TF algorithm proposed by Ricca et al [6] has a high time complexicy. When n is the size of developer team and m is the total number of files in the project then the time complexity is given by:

$$T(n,m) = n * \sum_{i=1}^{n} \frac{n!}{i!(n-i)!} * m \tag{1}$$

For example, if a project has 30 developers then the number of developer combinations that is over 17 million. In previous work, the largest project had only 38 developers [7]. We are considering much larger projects, for example, Chrome and Linux have 1000’s of contributors. It is impractical to combute all developer combination. We introduce a stopping condition which we prove identifies, for a given group size, the set of developers who’s loss will result in the maximum file loss.

Truck factor definitions

The following is the definisiton of symbols which will be used later:
D = the set of all developers in the project.
d_i = a particula develoer in D, i is the ID of the developer.
F = the set of all files in the project.
f_j =a particula file in F, j is the ID of it.
$M(f_j)$ = the set of develoers who have modified the file f_j.
$I()$ is the logic function defined as:

$$I(condition) = \begin{cases} 1 & \text{if condition is true} \\ 0 & \text{otherwise} \end{cases}$$

We first calculate the proportion of developer d_i who have modified the file f_j by equation:

$$L(d_i, f_j) = \frac{1}{|M(f_j)|} * I(d_i \in M(f_j)) \tag{2}$$

Then the file loss (FL) function returns how many files would be abandoned if a given developer combination C left the project:

$$FL(C) = \sum_{d_i \in C} \sum_{f_i \in F} (I(M(f_i) \subseteq C) * L(d_i, f_i)) \tag{3}$$

Then the maximum file loss for a given group size is returned by the truck factor function: $TF(C) = max(FL(C))$ We can see that a naive implementation of this algorithm will have a complexity of $O(n!)$, which is impractical to compute for large projects.

Stopping Condition

The shared proportion of the number of files a developer d_i has modified on the project:

$$L(d_i) = \sum_{f_i \in F} L(d_i, f_j) \tag{4}$$

The upper bound of the file loss for a given developer combination C is:

$$UFL(C) = \sum_{d_i \in C} L(d_i) \tag{5}$$

Since

$$\forall condition : I(condition) \leq 1$$

We have:

$$FL(C) \leq \sum_{d_i \in C} \sum_{f_i \in F} [1 * L(d_i, f_i)] = UFL(C) \tag{6}$$

For each group size, we order the developer combinations by their UFL value and calculate the FL for each developer combination until the stopping condition, $UFL(C) < max(FL)$ is met. In other words, we have shown that the number of files that a given group of developers modify, UFL, will be less than or equal to the number of files that these developers own exclusively, FL. Provided that we order the developers by the number of files that they modify, UFL, we can stop when the maximum loss that we calculated from the FL function is greater than or equal to the total number of files that the subsequent group of developers modify. The algorithm is presented in Box?Algorithm

TODO:combinations

Unforetunately, we still must calculate must do $n!$ UFL calculations. So we introduce a method to order these UFL calculations. TODO $TFX = threashold, i = 1;$

```
$F$ =total number of files in the project;
For each developer $d_i$ in the team $D$, calculate $K(d_i)$;
Order $D$ by $K(d_i)$ descently;
While $i \leq |D|$ repeat the block{
$Max\_fileLoss = 0;$
$combination\_generating(D, i);$
Initialize $input\_combination = input\_flow\_generating();$
While $SF(input\_combination) \geq Max\_fileLoss$ repeat the block{
If $KL(input\_combination) > Max\_fileLoss$ is true, $Max\_fileLoss = KL(input\_combination)$
$input\_combination = input\_flow\_generating();$
}
out print: if you lost a developer combination of size $size$, your maximum file loss would be $Max\_fileLoss;$
if $Max\_fileLoss \geq (TFX * F)$ is true the Truck Factor is $size$ and stop the algorithm;
Otherwise increase $i$ by 1 and continue;
}
```