Optimized Assignment of Developers for Fixing Bugs
An Initial Evaluation for Eclipse Projects

Md. Mainur Rahman
Department of Computer Science
University of Calgary
Calgary, Canada
mmrahma@ucalgary.ca

Guenther Ruhe
Software Engineering Decision Support Laboratory
University of Calgary
Calgary, Canada
ruhe@ucalgary.ca

Thomas Zimmermann
Microsoft Research
One Microsoft Way
Redmond, WA 98052, USA
tzimmer@microsoft.com

Abstract

Decisions on "Who should fix this bug" have substantial impact on the duration of the process and its results. In this paper, optimized strategies for the assignment of the "right" developers for doing the "right" task are studied and the results are compared to manual (called ad hoc) assignment. The quality of assignment is measured by the match between requested (from bugs) and available (from developers) competence profile. Different variants of Greedy search with varying parameter of look-ahead time are studied. The quality of the results has been evaluated for nine milestones of the open source Eclipse JDT project. The optimized strategies with largest look ahead time are demonstrated to be substantially better than the ad hoc solutions in terms of the quality of the assignment and the number of bugs which can be fixed within the given time interval.

Keywords- Empirical evaluation; bug fixing; human resource management; open source data

1. Introduction & related work

Completing maximum number of jobs (a bug, feature request, task) by a limited number of developers in a quantifiable way has potential impact on the quality of the bug fixing process as well as on the number of bugs which can be fixed in a given time interval. Looking at scenarios of having hundreds (if not thousands) of bugs in a repository, a proper assignment of developers to bugs becomes a tremendous challenge. Good resource allocation, i.e. assigning the most qualified developers to the necessary tasks, becomes even more important [1].

Anvik et al. [2] addressed the problem of assigning bug fix tasks to software developers and used supervised machine learning algorithm successfully for Eclipse (*https://bugs.eclipse.org/bugs*) and Firefox projects (*http://www.bugzilla.org*). Weiss et al. [3] built a model to estimate the time required to fix a new bug. In this paper, we present an approach that enhanced the former by the capability to better allocate resources and schedule bugs based on both expertise and effort estimates. Extensive literature review on bug management is included in the technical report [4].

The proposed strategies have been empirically evaluated for a set of nine data sets from open source Eclipse JDT project (*https://bugs.eclipse.org/bugs*). Problem description and formalization is given in Section 2. The optimized solution strategies are the content of Section 3. Their evaluation for the Eclipse data sets is following in Section 4. Discussion of the results and their validity is presented in Section 5. The paper concludes in Section 6 with a summary and an outlook to future research.

2. Problem description

We consider a process of fixing a given set BUG of bugs in a given time (release interval). We assume to have N bugs in consideration, i.e. BUG = {bug(1) … bug(N)}. Our time interval is defined as [0,T], i.e., we have a planning horizon of T time units (days) in consideration for fixing the N bugs. We assume that a pool of developers DEVELOPER consisting of D individuals available during the time interval represented as DEVELOPER= {dev(1)….dev(D)}. In order to find the most suitable developer for fixing a particular bug, greedy search with varying look-ahead time has been implemented. We call this Greedy-X, where X is the look-ahead time in days.

In the context of the organization and/or the project; there is a set of competencies, which have been observed to be relevant for fixing bugs. We summarize all these competencies into a vector CompProfile having K components. We define requested Request(n) and available skills DevProfile(d) from the perspective of a bug(n) to be fixed and a developer dev(d),

respectively. Both of these vectors have the same number K of components as the CompProfile vector. The individual components k = 1…K are denoted by Request(n,k) and DevProfile(d,k). The individual skill levels [6] are formulated by numbers from interval [0,1] expressing the skill level in the respective component of the vector. As an approximation and for simplicity, both vectors have been normalized to 1:

$$\sum_{k=1..K} \text{Request}(n,k) = 1 \text{ for all } n = 1..N \quad (1)$$

$$\sum_{k=1..K} \text{DevProfile}(d,k) = 1 \text{ for all } d = 1..D \quad (2)$$

For a particular bug(n) and a given developer dev(d), we define the distance dist(n,d) as follows:

$$\text{Dist}(n,d) = \sum_{k:\text{Requestt}(n,k)\neq 0} (\text{Request}(n,k) - \text{DevProfile}(d,k)). \quad (3)$$

The lower the distance value, the better is the match between the two profiles. For the formulation of the solution approach, we also consider the priority prio(n) assigned to bug(n). Priorities are defined on a five point scale with prio(n) = 1 being the highest and prio(n) = 5 being the lowest. In addition, the size of a bug is expressed by the total number of source (new, revised or deleted) lines of code (SLoC) impacted by fixing the bug. The fitness between profiles goes beyond similarity and takes into account also the priority of the bug and its size.

$$\text{Fitness}(n,d)=(1-\text{Dist}(n,d))*\text{size}(n)*\text{priority}(n) \quad (4)$$

In this paper, we answer the following questions:

1) How does the quality of the assignment of developers to bugs change when comparing the baseline manual assignment with the results from Greedy-X?
2) How does the total fitness of the assignment of developers to bugs change when comparing the baseline manual assignment with the results from Greedy-X?
3) Our final analysis question is devoted to the total time saving gained from optimized allocation of developers to bugs.

3. Solution approach

The problem formulated in Section 2 is a special case of the resource-constraint project scheduling problem, known to be NP-complete [5]. Our proposed heuristic solution method is based on Greedy search. Greedy search takes some given order of the objects in consideration. In our case, this order is defined by the opening dates of the bugs, e.g., we consider the bugs in the same order as they were opened in case of the manual assignment. To overcome the limitations of making local decisions, we have introduced additional look-ahead time for the search. That means, for a given point in time, we not only look at the idle developers at this time, but consider all developers becoming available within the given look-ahead-time. Our solution method relies on a number of assumptions:

1) Each developer fixes exactly one bug at the time.
2) Each bug is fixed by exactly one developer.
3) Once assigned to a bug, each developer is working continuously on this bug.
4) The productivity of a developer is inverse proportional to the effort required to fix a bug (where the time is known).
5) The duration for fixing a bug is defined by the ratio between closing and opening date as defined from the manual assignment divided by the ratio in expertise between the new versus manual assignment.

The status of validity of the assumptions is discussed later in Section 5. A description of Greedy-X, the application of the greedy search with a look-ahead time X, is presented below.

3.1 Procedure Greedy-X

For each of the bugs, look into the developers set to find the idle developers at point in time open(t). If an individual developer would be available with the specified (X) look-ahead time, mark this as special. For the bug in consideration find the distance for all the available developers including special ones according to equation (3). Assign the bug to the developer with the lowest distance. The starting date is defined by the opening date of the bug if the developer is in idle state. Otherwise (look-ahead/special), fixing the bug would start from the date when the assigned developer would be available. The end date is dependent on the expertise improvement.

The baseline expertise level and the requested competencies suggested by Greedy-X is formulated as

$$\text{Expertise}_{\text{Adhoc}}(n) = \sum_{k:\text{Request}(n,k)\neq 0} \text{DevProfile}(d_{\text{Adhoc}}(n),k) \quad (5)$$

$$\text{Expertise}_{X}(n) = \sum_{k:\text{Request}(n,k)\neq 0} \text{DevProfile}(d_{X}(n),k). \quad (6)$$

The relative expertise improvement REI (in %) gained from the Greedy-X assignments is defined as:

$$\text{REI}(X) = \sum_{n=1..N} (\text{Expertise}_{X}(n) - \text{Expertise}_{\text{Adhoc}}(n))/ \sum_{n=1..N} \text{Expertise}_{\text{Adhoc}}(n) \quad (7)$$

The end date for fixing the bug is determined by deducting the product of total duration of the bug and equation (7) from the total duration of the bug. The assigned developer is assumed to work on that particular bug for the time period between start and end date of fixing the bug.

4. Case study for eclipse JDT

4.1. Description of data sets

A brief description of different components of the Eclipse JDT (http://www.eclipse.org/jdt) products along with characteristics of nine different milestones for the 3.1 release and developer profile are provided in the technical report [4].

4.2. Analysis of results

The empirical results compare the baseline (manual) assignment with the results from the different greedy search algorithms, having varying look-ahead time.

Answer to the Question 1: We see a clear improvement pattern in terms of quality of assignment, and the improvement becomes the more significant, the bigger the look-ahead time is. Similar tendencies have been observed for the other milestones. The results for all milestone projects and all variations of X are summarized inFigure1.

For all nine milestones, we observe an almost monotonic improvement from increasing the look-ahead time for the respective greedy search. For example, the average improvement for the Greedy-30 application is 16%.

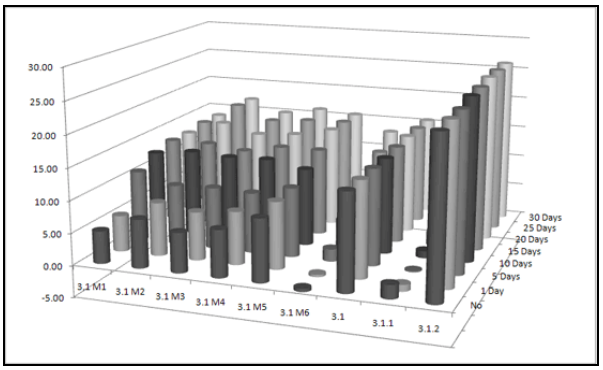

Figure 1. REI (in %) from Greedy-X

Answer to the Question 2: Total fitness here is defined as the summation of all the individual fitness values (4). Total fitness depends on the assigned gain from the chosen assignment approach. The total fitness gained from Greedy-X and ad hoc is represented in (8) and (9), respectively.

$$\text{Total\_Fitness}_X = \sum_{n=1..N} \text{Fitness}(n, d_X(n)) \quad (8)$$

$$\text{Total\_Fitness}_{Adhoc} = \sum_{n=1..N} \text{Fitness}(n, d_{Adhoc}(n)) \quad (9)$$

Based on the different degrees of fitness, we are now able to define our relative improvement in the degree of total fitness RFI:

$$\text{RFI}(X) = (\text{Fitness}_X - \text{Fitness}_{Adhoc})/\text{Fitness}_{Adhoc} \quad (10)$$

In Figure 2, the RFI results for all milestone projects under varying X values have been summarized.

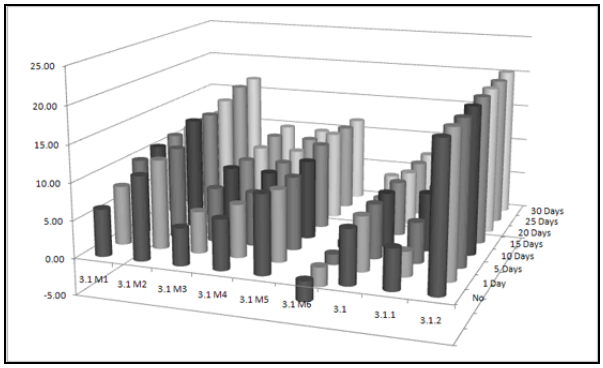

Figure 2. RFI (in %) from applying Greedy-X

We observed the similar type of repetitive improvement in fitness as well for all the nine milestones. The more the look-ahead time, the more the improvement is.

Table 1. Time savings from Greedy-X

Mile stone	Look ahead time (in days)						
	1	5	10	15	20	25	30
3.1 M1	2046	2237	2266	2281	2273	2294	2295
3.1 M2	1384	1481	1615	1597	2019	2400	2358
3.1 M3	3097	3107	3468	3349	3606	3802	3851
3.1 M4	5916	5938	6282	6377	6685	7089	7034
3.1 M5	16208	15793	15644	15828	16694	16583	16860
3.1 M6	1383	1505	1571	1605	2103	2622	3153
3.1	1176	1185	1185	1220	1222	1222	1222
3.1.1	645	487	693	1313	2073	2331	2476
3.1.2	1839	1843	1878	1880	1923	1928	1929

Answer to the Question 3: The results of the analysis are given in Table 1. For all the nine milestones, we have summarized the accumulated time savings (in hours) gained from the different solutions of different greedy search. For milestone 3.1 M1, with 5 days look-ahead we can save 2237 hours. We observe a substantial time savings with increasing look-ahead time which would allow fixing additional bugs.

5. Validity

Even though the initial results are looking promising, there are several threats to validity of results.

1) Running nine milestones in the JDT open source environment does not give any justification to claim for external validity. More comprehensive analysis is necessary, covering both other open source data sets as well as exploiting proprietary data sets.
2) The assumption of normalized vectors is considered to be an approximation of the truth, thus ignoring individual differences in the total amount of competence.
3) Our simplified model takes lines of code as the sole indicator variable, combined with the indirect relationship to productivity when compared to the baseline assignment.
4) The whole experiment was performed as a comparison between the manually generated baseline solution and assignments generated from greedy search. This implied sticking to the precedence relation as defined by the opening dates of the bugs. However, this is not a limitation imposed by the real-world, as changing the sequence of bug fixing might have an even bigger impact on the final results.
5) Another basic assumption of the study is that the number of defects and the defects themselves are known in advance. This assumption was valid in order to compare the manual with the greedy optimized assignments.
6) Determining the competence profiles of developers is important as well as extremely difficult. Ignoring the different competence priorities would be even more dangerous. What we suggest is to make a first step into this direction, knowing that evaluating the competence profile quantitatively from what has been done in the past is again a simplification of reality.

Even though there are a number of acknowledged facts to limit validity of the results, we consider that the main value of the study is having proposed a method for which it was initially proven that manual decision-making is insufficient in the complex situation encountered for deciding "Who should fix that bug".

6. Summary and outlook

Continuing existing research related to the questions "Who should fix this bug?" and "How long does it take to fix this bug?", we have studied the question "Who is best in fixing a series of bugs?". The results are promising, at least in comparison with the baseline ad hoc assignments. Better assignments could be demonstrated for nine open source Eclipse projects. We have also applied our model successfully to other Eclipse projects such as AspectJ project (http://www.eclipse.org/aspectj). Related results are presented in the technical report [4].

For further research, numerous other heuristics are available (as surveyed in [5]) which potentially are candidates to achieve further improvements, better than Greedy-X. Our method is relying on some assumptions, which partially can be relaxed. The availability of all the necessary data is critical for the method as well. Further investigations are needed for finding how the reliability of the profile data can be improved. Finally, other search strategies, especially genetic algorithms, will be used.

Acknowledgment

This research was supported by the Canadian NSERC Discovery Grant #. 250343-07. We also acknowledge the reviewer's comments related to a former and the current submission of the paper.

References

[1] P. Kapur, A. Ngo- The, G. Ruhe, A. Smith (2008): Optimized staffing for product releases and its application at Chartwell Technology. Journal of Software Maintenance and Evolution: Research and Practice Vol. 20, pp 365-386.

[2] J. Anvik, L. Hiew, G. C Murphy (2006): Who Should Fix This Bug? Proceedings of the 28th International Conference on Software Engineering ICSE 2006, Shanghai, pp 361-370.

[3] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller (2007): How Long will it Take to Fix This Bug. Proceedings of the Fourth International Workshop on Mining Software Repositories.

[4] M.M. Rahman, G. Ruhe, T. Zimmermann (2009): Optimized Assignment of Developers for Fixing Bugs: TR 081/2009. [http://ucalgary.ca/~mmrahma]

[5] R. Kolisch and S. Hartmann: Experimental investigation of heuristics for resource-constrained project scheduling: An update, European Journal of Operational Research Vol. 174 (2006), pp. 23-37.

[6] A. Schröter, T. Zimmermann, A. Zeller: Predicting Component Failures at Design Time. In Proceedings of the 5th International Symposium on Empirical Software Engineering (ISESE 2006), Rio de Janeiro, Brazil, September 2006, pp. 18-27.