# A Bug You Like: A Framework for Automated Assignment of Bugs

Olga Baysal Michael W. Godfrey Robin Cohen
School of Computer Science
University of Waterloo
Waterloo, ON, Canada
{obaysal, migod, rcohen}@uwaterloo.ca

## Abstract

*Assigning bug reports to individual developers is typically a manual, time-consuming, and tedious task. In this paper, we present a framework for automated assignment of bug-fixing tasks. Our approach employs preference elicitation to learn the developers' predilections in fixing bugs within a given system. This approach infers knowledge about a developer's expertise by analyzing the history of bugs previously resolved by the developer. We apply a vector space model to recommend experts for resolving bugs. When a new bug report arrives, the system automatically assigns it to the appropriate developer considering his or her expertise, current workload, and preferences. We address the task allocation problem by proposing a set of heuristics that support accurate assignment of bug reports to the developers. Finally, we discuss the practical challenges in developing and validating the proposed model.*

## 1 Introduction

A software development project usually maintains a bug repository that stores and tracks submission and fixes of the bugs. A bug repository is essentially a database that contains problem reports for a software project. When a new report is submitted to the repository, it needs to be assigned to an appropriate developer for its resolution.

Large development projects incur a sizable number of bug reports every day [28]. For example, in the Eclipse open source project there are on average 29 reports submitted each day [2]. Assignment of bugs is typically a manual, time-consuming and tedious task. The person in charge of assigning bugs, who, in most cases, is the team lead of the project, spends an hour or more a day on deciding which developer is most suitable for the task at hand. When making such a decision, the team lead should consider not only the expertise of the developer [3, 22] but also his preferences in fixing bugs, his current workload, as well as the priority and the difficulty of the bug. In the Eclipse project, 24% of bugs are currently re-assigned to other developers before they are resolved [2]. Automating the process of task allocation to the most suitable developers can improve the accuracy of the task assignment.

The problem of automated assignments of bugs has been addressed in the area of software engineering and data mining [1, 2]. Our work was mainly motivated by the work of Anvik et al. [2] who proposed a semi-automated approach for the bug assignment using machine learning techniques. They empirically evaluated their model on two software projects and were able to correctly assign appropriate developers with the tasks with a precision of 57% and 64% for each project. Although their results were not very promising, we believe that the problem of automated bug assignment deserves another look. We believe that employing efficient preference elicitation methods will allow us to build a reliable theoretical framework for solving automated bug assignment problem.

Currently, preference elicitation is one of the hot topics in the area of artificial intelligence (AI). Preference elicitation is concerned with acquiring the user's preferences (e.g., interests, tastes, goals) and using them to make decisions on behalf of the user (e.g., recommending what news to read or which digital camera to buy, helping the user to plan a family vacation). Although, preference elicitation is rigorously discussed among the AI research community and the deployment of preference elicitation methods has been demonstrated to bring benefits to various AI areas [6, 11], in the field of software engineering it remains mostly a topic for future exploration. Therefore, our goal is to explore how preference elicitation techniques can be applied to the field of software engineering and what challenges may arise doing so.

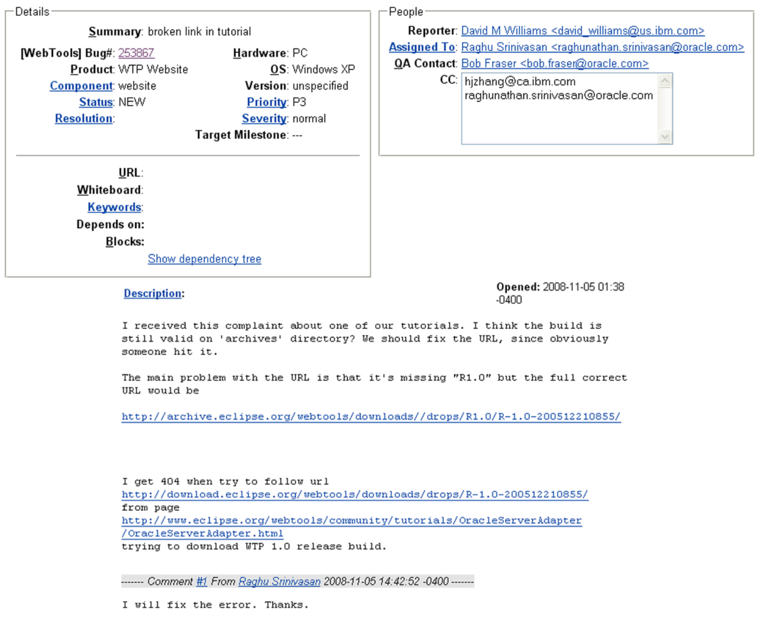
In this paper, we present a framework for automated assignment of bug fixing tasks. The proposed framework is able to infer a developer's level of expertise by tracking the history of the bugs previously resolved by this developer. Our approach also employs explicit preference elicitation methods [11, 25] to determine the developer's preferences on fixing certain types of bugs. Assigning "favorite" bugs to developers can increase their self-motivation. When given a preferred task to complete, a developer would spend less time procrastinating since he would be familiar with the task at hand. Therefore, each developer is more productive when fixing preferred bugs. Our framework addresses the weaknesses of the previous attempts to automate the bug assignment and provides a possible solution to the task allocation problem, a problem of determining the right developer for the right task. We combine knowledge about the expertise and preferences of the developers together with the information on their current workloads and the properties of the bug (priority and difficulty) under one unified framework and then we perform automated assignment of bugs.

The paper makes two contributions:

1. we describe how preference elicitation methods can be employed to learn the developer's preferences and expertise in fixing bugs, and

2. we present a theoretic framework for automated assignment of bug fixing tasks.

The remainder of the paper is organized as follows. Section 2.1 provides the background information on the bug repository and what data is stored in the bug report. Section 2.2 describes previous efforts to automate the process of bug assignment. Section 2.3 presents the analysis of the related work on preference elicitation and recommender systems. Section 3 presents the framework for automated assignment of bugs and describes each component and technique used. Section 4 discusses the challenges that we face in developing and validating the proposed framework, while Section 5 provides possible directions to address in the future. And finally, Section 6 concludes the paper by summarizing its contributions.



Details

**Summary**: broken link in tutorial

| | | | |
|---|---|---|---|
| **[WebTools] Bug#**: | 253867 | **Hardware**: | PC |
| **Product**: | WTP Website | **OS**: | Windows XP |
| **Component**: | website | **Version**: | unspecified |
| **Status**: | NEW | **Priority**: | P3 |
| **Resolution**: | | **Severity**: | normal |
| | | **Target Milestone**: | --- |

**URL**:
**Whiteboard**:
**Keywords**:
**Depends on:**
**Blocks:**
Show dependency tree

People

**Reporter**: David M Williams <david_williams@us.ibm.com>
**Assigned To**: Raghu Srinivasan <raghunathan.srinivasan@oracle.com>
**QA Contact**: Bob Fraser <bob.fraser@oracle.com>
CC: hjzhang@ca.ibm.com
raghunathan.srinivasan@oracle.com

**Description**:

**Opened:** 2008-11-05 01:38 -0400

```
I received this complaint about one of our tutorials. I think the build is
still valid on 'archives' directory? We should fix the URL, since obviously
someone hit it.

The main problem with the URL is that it's missing "R1.0" but the full correct
URL would be

http://archive.eclipse.org/webtools/downloads//drops/R1.0/R-1.0-200512210855/



I get 404 when try to follow url
http://download.eclipse.org/webtools/downloads/drops/R-1.0-200512210855/
from page
http://www.eclipse.org/webtools/community/tutorials/OracleServerAdapter
/OracleServerAdapter.html
trying to download WTP 1.0 release build.
```

------- *Comment #1 From Raghu Srinivasan 2008-11-05 14:42:52 -0400* -------

```
I will fix the error. Thanks.
```

**Figure 1. A sample bug report of the Eclipse project.**

## 2 Related Work

To understand the process of bug assignment and how our approach can work, we need to explore what a typical bug report is and what information it stores.

### 2.1 The Anatomy of a Bug Report

Figure 1 shows an example of a bug report for the Eclipse project retrieved from the Bugzilla bug repository [16]. A bug report is essentially a textual document that consists of several pre-defined fields, plus free-form text, and optional attachments.

Some values of the pre-defined fields, such as report ID number, date when the report is created and the name of the reporter who submitted the bug, are present. Other values, such as the component, operating system, priority and severity are assigned by the submitter and may be subsequently changed.

The free-form text includes the title of the report, a full description of the bug and optional comments. The full description typically explains the problem and its effect on the system concisely, while comments include discussions about possible solutions on how the bug can be fixed.

A bug submitter may also provide attachments to the report, including non-textual information such as an image or a binary file. A bug repository tracks the activity of each bug report. A single developer is usually assigned for fixing the bug; however, other developers may also contribute their ideas on how to solve the bug and lead to the resolution of the report.

### 2.2 Automating Bug Assignment

The problem of automated assignment of bugs has been previously addressed by the data mining research community [1, 2, 7]. Canfora and Cerulo [7] used an information retrieval approach to automate bug assignment process. They used textual descriptions of fixed change requests stored in both bug tracking and source code change repositories to index developers and source files as documents in an information retrieval system. They reported recall levels of 20% for Mozilla projects. The work of Mockus and Herbsleb [22] also addressed the problem of recommending experts in certain parts of the system. However, they used source code change data from a version control repository to determine experts for given elements of software project. Unlike this previous work, in our approach we determine expertise of the developer based on the history of past bug solving tasks extracted from the bug repository data.

Anvik et al. [2] proposed an approach to automate bug assignment using machine learning techniques. They recommended potential resolvers for the bug by mining bug reports that the developers have been previously assigned to and resolved for the system. Their approach is semi-automated because the triager, the team lead for example, uses the recommended list to select a potential developer who can fix the bug and makes the final decision on assigning this bug to the appropriate developer considering the team's current workloads and schedules. They empirically evaluated their model on two open source software systems, Firefox and Eclipse. They were able to correctly assign appropriate developers with the bug fixing tasks with a precision of 57% and 64% for each project respectively. We believe that employing both the efficient preference elicitation and task allocation methods will allow us to overcome limitations of the existing models and build a reliable theoretical framework for solving automated bug assignment problem.

Thus, existing research provides at best only a semi-automated solution for the bug assignment by recommending potential developers who are able to solve the task. While the developer's workloads and schedules were handled manually, the developer's preferences were not considered at all. Our proposed framework not only makes recommendations on who is capable of fixing the bug, it considers the preferences of the developers, their workloads and schedules, the properties of the report such as difficulty and priority when allocating problem reports to the team members.

## 2.3 Preference Elicitation

Recent AI research has focused on the role of preference elicitation in making optimal decisions in a variety of situations [5, 10, 13, 15, 26]. Preference elicitation deals with eliciting information from the user on his or her preferences, interests, goals, etc. and using collected information for later reasoning or decision making [25].

The main goal of an automated bug assignment system is to make an optimal decision on who should be fixing the bug at hand. Thus, the system must be aware of the developer's preferences as the optimal solution will differ from one developer to another. Our proposed framework would act on behalf of every developer, yet focus on the main goal of the team, i.e., the accurate and effective resolution of software defects.

Preference elicitation methods are currently used in many personalized recommender systems. A recommender system filters information items according to the user's interests. For example, news recommender systems provide recommendations on most interesting and relevant news articles to the users according to their interests and preferences. AI research literature provides various solutions on news recommender systems [4, 8, 12, 14, 19, 20, 23, 29]. Some recommender systems build profiles for each user that contain his or her preferences in the news content, and can recommend relevant articles according to their textual similarity to the users' profiles; this is know as content-based filtering [4, 12, 23, 29]. Other systems use ratings from early readers of an article to predict later readers' ratings; this is known as collaborating filtering [14, 19]. Hybrid recommendation models combine content-based and collaborative filtering strategies under a single framework, solving limitations of either approach [8, 20].

As new bug reports are submitted, each one is assigned to a single developer. Other developers can submit their comments for any bug to be resolved and discuss the possible resolution of the bug. However, there is typically no collaboration between developers on the actual implementation of the solution: once a bug report is assigned to a developer, it is then up to that developer alone to resolve it. Therefore, our problem is similar to the content-based news recommender system. A typical content-based news recommender system [4, 9, 29] periodically gathers news articles through RSS feeds, passing them through an indexing module and builds an index from the title, description and content of the article. The indexing module creates and stores TF-IDF term vectors of the articles by using vector space model, where TF-IDF (term frequency - inverse document frequency) weight represents the importance of a word, also called a term, in the collection of news articles. A vector space model (VSM) is a model used in information retrieval to represent documents as vectors in order to rank them. Each dimension corresponds to a single term. When recommending relevant documents such as news articles, the vector representing a user profile is compared to the vectors representing news items. All news articles are ranked according to their similarity to the user profile. In recommender system, the user profile is extracted from the user's news reading history.

Since bug reports are essentially textual documents, we could also represent them as vectors. Thus, we employ the techniques, both the TF-IDF weighting scheme and VSM, used in the context-based news recommender systems when determining the expertise of the developers in fixing certain bugs. By mining the bug tracking repository we can learn what bug reports each developer has previously resolved and decide whether he is an expert in fixing the given bug report.

Although a number of bug assignment models have been developed, they failed to account the developers' interests and preferences in fixing bug reports. For example, Alice likes to resolve documentation-related bugs, and Bob prefers to work on security bugs (e.g., authentication errors). When Alice and Bob are given their favorite tasks to complete, it is very likely they will do a better job in fixing bugs. Motivating developer performance can make developers more efficient and effective. As a result, the overall productivity of the team will likely be increased.

In order to elicit preferences from the developers, i.e., the information on what bug reports they like to work on, we employ explicit preference elicitation methods. Explicit elicitation asks the user to provide a specific value for each of his preferences [18]. Related work offers several techniques on how to learn user preferences. NewsWeeder [20] system asks users to rate each article with a value from 0 to 5 in order to get access to the next relevant news item. The system uses the collected rating information to learn the user's interests. Maes [21] suggested that computer agents should work in collaboration with the user. She stated that the agent becomes more efficient as it learns the user's preferences, interests and habits. In order to train an agent, the user can provide positive or negative feedback on the recommended news articles. Following recommendations from Lang [20] and Maes [21], we elicit preferences from the developers by asking them to rate each bug report they solve. These ratings provide us with the accurate feedback on the developer's preferences.

## 3 A Framework for Automated Assignment of Bugs

Our approach uses preference elicitation methods to learn what type of bugs developers prefer to solve, as well as techniques to automatically infer their expertise levels. Our framework consists of three components.

1. Expertise recommendation

   When a new bug report arrives, the system creates a ranked list of the developers who have the most appropriate expertise to resolve the bug in question. This ranking is done by mining the bug repository to infer expertise profiles of the individual developers, based on the bug reports that they have resolved in the past.

2. Preference elicitation

   Developers submit their preferences in fixing bugs by providing feedback for every bug they fix. The feedback represents a developer's rating of the bug, for example, if a developer did not struggle with fixing the bug, he would provide a positive feedback on this bug by labeling it as "preferred". The feedback is stored in the bug report itself and can be retrieved for reasoning any time.

3. Task allocation

   Knowing preferences and expertise levels of all the developers, the system then considers their schedules, the priority, and the difficulty of the report when making a decision on who should be assigned with this report.

We next describe each component in detail.

### 3.1 Expertise Recommendation

Why do we need to know the developer's expertise? We assume that developers prefer to handle bug reports in their area of expertise. Looking at the bug solving experience of the developer we can learn his or her "implicit" preferences in fixing bugs. Our system recommends the best candidates for fixing each bug by observing developers' previous work in resolving bug reports. We employ the vector space model to infer information about the developer's expertise from the history of the previously fixed bugs.

For each bug report we first build a profile using its title (the one-line summary), the full textual description and comments. Like other term-based recommender systems, we first extract keywords from the textual content of the summary, description and comments in order to represent each report as a term vector. We remove non-alphabetic tokens and stop words—words that do not carry any meaning, such as articles, prepositions, etc. The term vector is then built from the remaining words by assigning weights to each of the word. We apply a standard TF-IDF weighting scheme to assign high weights to the most important words in the report and low weights to less informative ones.

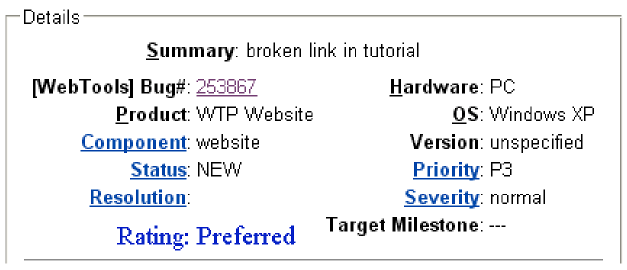
| term | $\text{tf} \cdot \text{idf}$ weight* |
|---|---|
| archive | 14 |
| complaint | 30 |
| error | 5 |
| fix | 14 |
| tutorial | 74 |
| url | 42 |

**Table 1. A partial weighted term vector for the bug report depicted in Figure 1; tf · idf weights are arbitrary due to the unknown document frequency, the number of the documents the term occurs in.**

The developer's profile is then represented as a weighted term vector extracted from the developer's history of previously resolved problems. Table 1 represents the term vector extracted from the sample bug report shown in Figure 1. We make recommendations of the best experts by comparing two term vectors, representing the developer's profile and the profile of the new incoming report being assigned. Similarity scores between two vectors are calculated using cosine coefficients ranging from 0 to 1. Thus, the system makes recommendation of the developers' expertise by ranking developers according to the similarity of their profile with the profile of the new bug report. The higher the ranking of the developer in the recommendation list is, the higher value of the expertise is assigned to the developer.

## 3.2 Preference Elicitation

As mentioned earlier in Section 2.3, we let developers express their preferences by rating bugs they fix. These ratings provide accurate feedback on the developer's preferences. We provide developers with the additional predefined field in the report, as illustrated in Figure 2, to leave their feedback on each bug they have fixed. Each report now includes a new field "Rating", where developers can submit their ratings on the report by labeling it as "Preferred", "Neutral" or "Non-preferred". These ratings are used to acquire developer's preferences in solving future problems.


Details

**Summary**: broken link in tutorial

| | | | |
|---|---|---|---|
| **[WebTools] Bug#**: | 253867 | **Hardware**: | PC |
| **Product**: | WTP Website | **OS**: | Windows XP |
| **Component**: | website | **Version**: | unspecified |
| **Status**: | NEW | **Priority**: | P3 |
| **Resolution**: | | **Severity**: | normal |
| **Rating**: | Preferred | **Target Milestone**: | --- |

**Figure 2. Eliciting developer's preferences through bug ratings.**

After the developer provides ratings, we elicit his preferences by creating a whitelist that contains all of the bugs that the developer likes to resolve. The whitelist is a vector that contains all the terms from the profiles of the "preferred" reports. When task assignment process considers the developer's preferences, this whitelist is compared against the newly submitted report vector. If the similarity score between two vectors is high, the developer would be happy to be assigned with this report as it satisfies his preferences, and if the score is low, i.e., the report does not match the developer's preferences and therefore, the developer would be less pleased with its assignment.

## 3.3 Task Allocation

When allocating a bug fixing task to the most appropriate developer, we need to consider both the developer-specific and bug-specific factors. The developer-specific factors include his preferences, level of expertise, current workload and availability. In Sections 3.1 and 3.2, we have already discussed how we can determine the developer's expertise and learn his preferences, respectively. In order to find out whether the developer is on vacation, business trip and/or how busy he is, we need a scheduler tool to retrieve necessary information. We assume that the development team supports either in-house or off-the-shelf task scheduler, so that we are able to infer the developer's workload information including the number of tasks he is currently working on, his travel commitments, vacation days, etc.

The bug-specific attributes indicate its priority (how soon it needs to be fixed) and difficulty (how long it takes to resolve it). The priority of the bug varies from critical (requires immediate attention and resolution), major (a main blocker), normal, minor (can be postponed for later) to trivial (no urgent resolution required).

To be able to assign the task to the person currently available, we need to know the effort or difficulty required to fix the bug. The focus of the paper is not to provide a new effort estimation approach, but rather to consider the cost of the bug fixing tasks when delegating them to the developers. Existing research on predicting the time and effort for a software problem offers several promising solutions [28, 31]. Most recent approach was proposed by Weiss et al. [28]. They suggested a model that automatically predicts the fixing effort (i.e., the person hours needed to fix the bug) by considering only the title and the description of the bug report. They used Lucene framework, a text similarity measuring engine, to search similar, earlier reports and use their average time as a prediction. We make use of their approach to estimate the effort needed to solve the bug report. We can retrieve the title and the description from the report to build a term vector which later can be used to find similar reports by mining the data in the bug repository. By looking at the time frame the relevant reports had been fixed for, we can make predictions on the estimated time the newly arrived report will require for its resolution.

Having this information, we then suggest strategies, which are based on a set of heuristics for addressing task assignment problem. These heuristics provide the system with a guideline to assign reports to the most appropriate developer. The key strategy that our approach suggests is to decide on the priority of considering different factors such as preferences, expertise and workload.

In a business environment, the ultimate goal is to resolve software problems as soon as they are reported and to minimize the cost of software bugs. Thus, automated bug assignment must assign bugs to the developers who can resolve problems fast and efficiently. Satisfying the time constraint is very important. Thus, the practical solution for a software development team is to give up some of the ideal properties such as the developer's preferences in certain situations. We need the system to not only automate the task assignment process but also to provide optimal and fair task allocations.

We consider situations where a set of bug reports, $\overline{R} = \{R_1, \ldots, R_n\}$, should be assigned as soon as possible. There is a set of developers $\overline{D} = \{D_1, \ldots, D_m\}$, who are able to perform these tasks. Expertise of the developer $\mathrm{E}_i$ defines the capabilities of the developer to fix a task $\mathrm{R}_i \in \overline{R}$. For each report $\mathrm{R}_i$, we know the developer's preference $\mathrm{P}_i \in [0,1]$. The availability of the developer is defined as $\mathrm{A}_i$, where $\mathrm{A}_i \in [0,1]$. The complexity of the task is manifested by the difficulty $\mathrm{d}_i$ and priority $\mathrm{p}_i$, where $\mathrm{d}_i$, $\mathrm{p}_i \in [0,1]$.

We provide heuristics to be used by the system when selecting the most appropriate developer for the given task. We consider three basic heuristics for task assignment problem.

***Workload heuristic:*** The first heuristic aims at maximizing the overall effectiveness of the team. It suggests that in order to allocate tasks to the developers, the system needs to know their current workloads and availability. For each unallocated report $\mathrm{R}_i \in \overline{R}$, there may be zero or more developers who are interested in fixing it or familiar with the problem. Thus, such a problem report is assigned to the developer with highest availability $\mathrm{A}_i$. The team's policy dictates that once the developer is assigned with the task, he or she must resolve it.

This heuristic allows the system to assign the task to the least busy developer. There should be no situation where some developers are overloaded with work items, while others have little or no work. This heuristic represents common sense. Selecting developers who are occupied with less work is a reasonable decision. Our model ensures that all developers are engaged in a bug fixing activity.

Consider a scenario, where there are four developers in the team: Alice, Bob, Carol and Ted. Each developer is currently working on a number of tasks. The system needs to assign a newly arrived report $R_i$ to an appropriate developer. All developers are experts on solving certain problems. In this example, Ted is really interested in working on this task $R_i$, however he is already assigned with four tasks. Both Alice and Bob have high familiarity with the report, while Carol is less familiar with $R_i$. However, both Alice and Bob have extremely busy schedules, while Carol has currently only two tasks to solve. Thus, $R_i$ should be assigned to Carol as she has the lowest workload. The main goal of the system is to have bugs reports resolved as soon as possible and at minimal cost. If we wait for the expert to become available, the resolution of the bug will be postponed for later which will result in high cost of the bug. Thus, this heuristic suggests it is less expensive to assign reports to the developers with higher availability values.

***Expert heuristic:*** The second heuristic intends to minimize the cost of the bug resolving task, by utilizing the expertise of the developer. In our model developers do not compete with each other over the same tasks. However, if a developer is familiar with the task $\mathrm{R}_i$, i.e., he has successfully resolved a similar task in the past, such a developer is considered an expert. Therefore, tasks similar to the problem report $\mathrm{R}_i$ will be assigned to the expert who is currently available. The main strategy here is that each report is allocated to the developer who can actually resolve it.

In our next scenario, Alice and Carol, both equally available, have different expertise levels in fixing task $R_i$, where Alice is the main expert on this bug. For Carol, task $R_i$ is one of tasks she would really like to work on. If the report has a high priority $p$, i.e., needs to be resolved immediately, and high difficulty $d$, Alice will be assigned with this report because being the expert she will spend less time than Carol in completing the task. Even if the report has a low priority and a high difficulty or a high priority and a low difficulty, Alice will still be more efficient than Carol in fixing the bug $R_i$. Therefore, this heuristic focuses on employing the expertise of the developers in providing fast and accurate solutions.

***Preference heuristic:*** This heuristic is defined to maximize the overall team productivity. Knowing the preferences of the developers, the system allocates them with their preferred tasks. Developers working on "desirable" problems are typically more motivated and thus, produce faster and more creative solutions. We understand that it is difficult and sometimes impossible to elicit full preferences from the developers. Developers may choose not to provide their preferences, yet by that they choose to avoid potential gains from the preference elicitation. In this situation when the preferences are missing in the bug reports, we consider these reports as "non-preferred". This assumption originates from the social studies on the analysis of online product ratings and reviews [24]. Their main finding indicates that online shoppers provide more positive feedback on their shopping experience than negative one, and positive feedback overweighs negative feedback 8 to 1. We believe that developers are more likely to give positive feedback on the tasks than negative one. Thus, if developers choose not to provide their preferences, then their $P_i$ will be set to 0.

We believe that developers motivated by solving their "favorite" tasks are more efficient. However, it is not feasible to assign only preferred tasks to the developers. As mentioned earlier, our goal is to have all tasks be assigned and resolved in time. Thus, the developers should be designated with both preferred and non-preferred tasks. In fact, since the developers are employed by the company, the majority of them have the necessary qualifications to resolve any given bug report. In our framework, we consider preferences of the developer in the situation where the developer is already assigned with some tasks, i.e., his availability is low, or the developer is familiar with the task, i.e. he has a high expertise value.

We can illustrate this heuristic in the example, where both Alice and Bob are the experts on the report $R_i$ and have similar workloads. The system then considers their preferences on the task $R_i$. If their preferences are similar, the system randomly assigns the task $R_i$ to one of them. A more interesting example is where Alice, being familiar with the bug, does not prefer to deal with it, while Ted wants to solve this bug without having sufficient experience in fixing similar bugs in the past. How the system should handle this? Assuming that both Alice and Ted are currently available, the system will look at the properties of the report $R_i$. If the report is crucial and requires immediate attention, it would be given to Alice since her expertise yields faster resolution. On the other hand, if the report does not need urgent resolution, it will be delegated to Ted. He might not have enough expertise in fixing this bug but his motivation can make him more efficient than Alice who might delay the work on this "unwanted" task. Thus, this heuristic promotes preference elicitation when allocating tasks to the developers.

Another aspect to consider is how to facilitate and enhance the skills of the developers. We need to employ a mechanism to allocate easier tasks to new team members until they become familiar with the system and its architecture. Our system will gradually increase the difficulty of the tasks assigned to the committers in order to develop their expertise. It is rational and strategic to train developers by delegating more difficult tasks to resolve and reinforcing them to learn various parts of the system. This mechanism provides the development team with several experts in each module of the system. A common problem that the companies working with obsolete software systems face, is the "brain drain" problem when the leave of the main experts (e.g., developers of the system's kernel) can compromise the whole development and production of the software system.

## 4 Discussion

Our work provides a novel model for automated bug assignment. However, it lacks experimental evaluation due to the number of challenges in developing and validating the proposed model. In order to validate our heuristic-based approach, we could develop a simulated environment of the framework and test the efficiency and accuracy of the automated bug assignment. The setting of the simulated experiment could consider several parameters, including the number of developers, the number of tasks to be assigned, the task properties (priority and cost) and the developer's expertise, preferences and availability. We could also evaluate each of the heuristics and find the optimal solution for task assignment problem. Unlike other research areas such as AI or distributed systems where development and validation through a simulated environment is widely acceptable, the software engineering research community often seeks an experimental validation including industrial-sized case studies, dynamic or static analysis, etc. [30].

While assessing a technique using a simulated environment may be common in some research areas, there are also several risks associated with it. First, simulation requires building a model of a real system and a real dataset, and it may be hard to build realistic models practically. A model that achieves good results in simulation might not work in a practical setting. Another limitation of using simulation is that the results may be highly tuned to a particular implementation, the better the optimization of the prototype's code the better the performance it will deliver.

Conducting a case study on a real data would require an "almost from-scratch" implementation of the prototype of the bug tracking system. In addition, the case study will require recruitment of a software development team to apply automated assignment of bugs using the expertise levels, preferences and workloads of the developers. Such a case study would be very valuable to support our work, yet at the moment is in not feasible due to the high cost of the effort required to perform it. A possible first step to validate the proposed model would be to use evaluation through survey. First we could conduct a survey with the developers asking for the feedback on their previous bug fixing experience, their satisfaction with the bug assignments and bug resolutions, whether they were successful and confident in handling bugs in the past, etc. We could also elicit information on whether considering their preferences in fixing bugs would be helpful in distributing tasks within the team. The analysis of the collected data would provide the initial estimates of the usefulness of the proposed model. Later we could implement a prototype of the framework and test it within a software team working on the maintenance activities.

## 5 Future Work

The main challenge to address in the future is strategic developers. In time, developers could learn how the system assigns bug fixing tasks and try to manipulate task assignment. Thus, we should ensure that the assignment of bugs is a fair and manipulation-free process. A possible solution to prevent any misuse of the system would be to consider only most recent bug fixing experience. For example, only $n$ randomly taken past reports can be considered to determine a developer's expertise. Also it is in the best interests of the developers to provide truthful information on their bug preferences. If a developer limits bug reports to a certain type of bugs or a certain module of the system, it is less likely that he will get this type of bug all the time and thus, in most cases he will be assigned with the least preferred bugs to resolve.

Motivating developers to rate bug reports truthfully is the key issue. If developers do not submit their ratings to the reports, they might be assigned with the tasks that are undesirable. In fact, developers are encouraged to take advantage of the preference elicitation and to benefit from the decisions the system makes on their behalf. If developers try to manipulate the system, for example by preferring only easy tasks, the chances that they will be assigned with the same task every time are low. However, our future research will focus on designing an effective incentive mechanism to support truth-telling strategy and fairness of the task allocation. We can define a set of rewards to be given to the developers for their hard work, efficiency, productivity and creativity. For example, for resolving a very difficult bug, the developer might be assigned with less workload for the next day or two.

Another challenging strategy to tackle in the future is the trustworthiness of the developers in accomplishing tasks in time [17, 27]. For example, there are two developers, Alice and Bob, where Bob has a higher expertise level than Alice for completing the task $R_i$. According to the current expertise heuristic, the task $R_i$ should be assigned to Bob. However, if Bob is not good at meeting deadlines and delays bug resolutions, a better decision would be to assign the task to Alice, who might not be the best expert but is reliable in solving tasks on time. Thus, the notion of trustworthiness can be introduce to our framework to address the problem of delivering accurate solutions at the expected time.

## 6 Conclusions

In this paper, we presented the prototype of the intelligent system that automates bug assignment. When a new bug report arrives, the system automatically assigns it to the developer who is familiar with this bug and eager to solve it and thus, most suitable to accomplish the task efficiently and in a timely manner. The proposed framework can help software companies in managing bugs during the software development and software testing processes. It also allows the team lead to save several hours a day that were previously spent on manual assignment of bug reports.

We have argued how preference eliciation techniques can be used in the area of software engineering and how development teams can benefit from eliciting information about their developers. However, answering questions on what and how to elicit preferences from the users in a specific domain can become quite challenging. We believe that eliciting preferences from the developers is beneficial for the team performance. However, further research, including the empirical validation of the proposed approach can provide better insights on the impact of preference elicitation in personalizing bug assignment process. Therefore, we would like to call for further research in adopting preference elicitation methods to the software applications and development projects. We hope that our work can inspire the rest of the research community to build better systems to facilitate more efficient and intelligent assignment of maintenance tasks.

## References

[1] J. Anvik. Automating bug report assignment. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 937–940, New York, NY, USA, 2006. ACM.

[2] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 361–370, New York, NY, USA, 2006. ACM.

[3] J. Anvik and G. C. Murphy. Determining implementation expertise from bug reports. In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 2, Washington, DC, USA, 2007. IEEE Computer Society.

[4] D. Billsus and M. J. Pazzani. A personal news agent that talks, learns and explains. In *AGENTS '99: Proceedings of the third annual conference on Autonomous Agents*, pages 268–275, New York, NY, USA, 1999. ACM.

[5] C. Boutilier. A pomdp formulation of preference elicitation problems. In *Eighteenth national conference on Artificial intelligence*, pages 239–246, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[6] D. Braziunas. Computational approaches to preference elicitation. Technical report, Department of Computer Science, University of Toronto, 2006.

[7] G. Canfora and L. Cerulo. How software repositories can help in resolving a new change request. In *Proceedings of the Workshop on Empirical Studies in Reverse Engineering*, 2005.

[8] I. Cantador, A. Bellogín, and P. Castells. News@hand: A semantic web approach to recommending news. pages 279–283. 2008.

[9] P. Castells, M. Fernandez, and D. Vallet. An adaptation of the vector-space model for ontology-based information retrieval. *IEEE Trans. on Knowl. and Data Eng.*, 19(2):261–272, 2007.

[10] U. Chajewska, L. Getoor, J. Norman, and Y. Shahar. Utility elicitation as a classification problem. In *In Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 79–88, 1998.

[11] L. Chen and P. Pu. Survey of preference elicitation methods. Technical report, Swiss Federal Institute of Technology in Lausanne (EPFL), 2004.

[12] M. Claypool, A. Gokhale, T. Mir, P. Murnikov, D. Netes, and M. Sartin. Combining content-based and collaborative filters in an online newspaper. In *In Proceedings of ACM SIGIR Workshop on Recommender Systems*, 1999.

[13] C. C.White, A. P. Sage, and S. Dozono. A model of multiattribute decision making and trade-off weight determination under uncertainty. *IEEE Transactions on Systems, Man and Cybernetics*, 14:223–229, 1984.

[14] A. S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 271–280, New York, NY, USA, 2007. ACM.

[15] J. S. Dyer. Interactive goal programming. *Management Science*, pages 62–70, 1972.

[16] Bug reports for Eclipse projects. `https://bugs.eclipse.org/bugs/`.

[17] R. Ismail and A. Josang. The beta reputation system. In *Proceedings of the 15th Bled Conference on Electronic Commerce*, 2002.

[18] R. Kass and T. Finin. Modeling the user in natural language systems. *Computational Linguistics*, 14(3):5–22, 1988.

[19] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl. Grouplens: applying collaborative filtering to usenet news. *Commun. ACM*, 40(3):77–87, 1997.

[20] K. Lang. NewsWeeder: Learning to filter Netnews. In *in Proceedings of the 12th International Machine Learning Conference (ML95)*, 1995.

[21] P. Maes. Agents that reduce work and information overload. *Commun. ACM*, 37(7):30–40, 1994.

[22] A. Mockus and J. D. Herbsleb. Expertise browser: a quantitative approach to identifying expertise. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 503–512, New York, NY, USA, 2002. ACM.

[23] M. Nadjarbashi-Noghani, J. Zhang, H. Sadat, and A. A. Ghorbani. Pens: A personalized electronic news system. In *CNSR '05: Proceedings of the 3rd Annual Communication Networks and Services Research Conference*, pages 31–38, Washington, DC, USA, 2005. IEEE Computer Society.

[24] E. B. or Anna Jarrard. Bazaarvoice analysis reveals that positive online reviews outweigh negative reviews 8 to 1, 2006.

[25] F. Rossi. Preferences, constraints, uncertainty, and multiagent scenarios, 2008.

[26] A. Salo and R. P. Hämäläinen. Preference ratios in multiattribute evaluation (prime)-elicitation and decision procedures under incomplete information. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 31(6):533–545, 2001.

[27] W. T. Teacy, J. Patel, N. R. Jennings, and M. Luck. Travos: Trust and reputation in the context of inaccurate information sources. *Autonomous Agents and Multi-Agent Systems*, 12(2):183–198, 2006.

[28] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller. How long will it take to fix this bug? In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 1, Washington, DC, USA, 2007. IEEE Computer Society.

[29] J. wook Ahn, P. Brusilovsky, J. Grady, D. He, and S. Y. Syn. Open user profiles for adaptive news systems: help or harm? In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 11–20, New York, NY, USA, 2007. ACM.

[30] M. V. Zelkowitz and D. Wallace. Experimental validation in software engineering. *Information and Software Technology*, 39:735–743, 1997.

[31] H. Zeng and D. Rine. Estimation of software defects fix effort using neural networks. In *COMPSAC '04: Proceedings of the 28th Annual International Computer Software and Applications Conference - Workshops and Fast Abstracts - (COMPSAC'04)*, pages 20–21, Washington, DC, USA, 2004. IEEE Computer Society.