Search-based resource scheduling for bug fixing tasks

Junchao Xiao
Laboratory for Internet Software Technologies,
Chinese Academy of Sciences, Beijing 100190, China.
xiaojunchao@itechs.iscas.ac.cn

Wasif Afzal
Blekinge Institute of Technology,
PO Box 520, SE-372 25 Ronneby, Sweden.
wasif.afzal@bth.se

***Abstract*—The software testing phase usually results in a large number of bugs to be fixed. The fixing of these bugs require executing certain activities (potentially concurrent) that demand resources having different competencies and workloads. Appropriate resource allocation to these bug-fixing activities can help a project manager to schedule capable resources to these activities, taking into account their availability and skill requirements for fixing different bugs. This paper presents a multi-objective search-based resource scheduling method for bug-fixing tasks. The inputs to our proposed method include *i)* a bug model, *ii)* a human resource model, *iii)* a capability matching method between bug-fixing activities and human resources and *iv)* objectives of bug-fixing. A genetic algorithm (GA) is used as a search algorithm and the output is a bug-fixing schedule, satisfying different constraints and value objectives. We have evaluated our proposed scheduling method on an industrial data set and have discussed three different scenarios. The results indicate that GA is able to effectively schedule resources by balancing different objectives. We have also compared the effectiveness of using GA with a simple hill-climbing algorithm. The comparison shows that GA is able to achieve statistically better fitness values than hill-climbing.**

I. Introduction

Software bugs correspond to mistakes by the programmers due to an incorrect step, process, or data definition. One estimate is that a professional programmer is responsible for 5 bugs per 1000 lines of code (LoC) written on average [1]. This might not be the case with every software application but there are always a certain number of bugs in almost every software application that causes incorrect results.

Software testing is one major bug finding activity that improves software quality to a certain extent before the software application is released to the end-users. As bugs are reported, they must be triaged in a cost-effective manner, considering the resources and the requirements of bugs. Triage of bugs in a cost-effective manner is an important decision-making task whereby competing objectives of technical, resource and budget constraints need to be balanced to provide maximum business value for the organization.

Anvik et al. [2] report that 3426 reports were submitted to the bug database of Eclipse open source project between Jan. 1, 2005 to Apr. 30, 2005, averaging 29 reports per day. Assuming that it takes 5 minutes to triage a report, this activity costs 2 person hours per day. This indicates that we have a need to support efficient and effective bug-assignment policies that can schedule different bug-fixing tasks by taking into account the available resource constraints and bug requirements. Laplante and Ahmad [3] further emphasize the value of having an efficient and effective bug assignment policy: "Bug assignment policies can affect customer satisfaction, employee behavior and morale, resource use, and, ultimately, a software product's success". But triaging of bugs for repairing is fraught with challenges since the number of problem variables is diverse, e.g. the severity and priority of a bug has to be balanced with resource skills and availability for finding a reasonable bug-fix schedule.

Optimal resource scheduling for bug fixing is an example of resource constrained scheduling problem [4]. The scheduling problem in general is NP-hard, i.e., finding optimal solutions in polynomial time is hard [4], [5]. This is because the search-space becomes vast as problem size increases or more constraints are added. These properties naturally make scheduling problems a suitable problem domain for evolutionary computation approaches like genetic algorithms.

In this paper we attempt to (at least approximately) formalize the problem of appropriately scheduling developers and testers to bug-fixing activities, keeping in view the capabilities of resources and requirements of bugs. We therefore seek an answer to the following research question:

RQ: How to schedule developers and testers to bug-fixing activities taking into account both human properties (skill set, skill level and availability) and bug characteristics (severity and priority) that satisfies different value objectives by using a search-based method such as GA?

The contributions of this paper are: *i)* It presents models for bugs and human resources that form the basis for scheduling resources for bug-fixing. *ii)* It presents a method of capability matching between the bug-fixing activities and human resources as well as how to evaluate an organization's value of a bug-fixing process. *iii)* It provides a bug-fixing scheduling method by using a GA and discusses several scenarios of practical use.

The paper is organized as follows. Section II describes some related work. Section III presents the basis of scheduling resources for bug-fixing. It specifies models of bugs and human resources. Section IV discusses the design of

978-0-7695-4195-2/10 $26.00 © 2010 IEEE
DOI 10.1109/SSBSE.2010.24

the scheduling method using a GA. Section V presents the industrial data used while results of applying the proposed method are given in Section VI. The GA approach is compared with hill-climbing search in Section VII. Section VIII presents a discussion while the Section X presents conclusions and suggests future work.

II. Related work

Search techniques have been successfully used to solve different scheduling related software project management problems [6], such as software project planning [7], [8], [9], [10], software project effort prediction [11] and software fault prediction [12]. Hart et al. [13] have written a review on evolutionary scheduling. However, the application of search techniques for implementing an efficient bug repair policy is very much unexplored.

A study by Mockus et al. [14] predicted defect effort schedule based on observed new feature changes. They fitted a probability model to the observed data from 11 releases of a large real-time high availability software system and found the predicted effort to be close to reality. Cubranic and Murphy [15] applied naive Bayes classifier to automatically assign the bug reports to developers and achieved a 30% classification accuracy for reports entered into Eclipse's bug tracking system between Jan. 1, 2002 and Sep. 1, 2002. Zeng and Rine [16] used a self-organizing neural network approach to estimate defects fix effort. A feature map having different clusters was created after training the weights of the self-organizing neural networks. They computed the probability distributions of effort from the clusters and then compared them with those from the test set. For projects having similar development environments the approach gave acceptable performance with average mean relative error (MRE) values between 7% to 23%. Canfora and Cerulo [17] used a probabilistic text similarity approach to assign change requests to developers. Song et al. [18] presented association rule mining based methods to predict defect correction effort. Using data from more than 200 projects, their approach was found to be better than partial regression trees (PART), C4.5 and naive Bayes. Anvik et al. [2] applied support vector machine algorithm as a text categorization technique to suggest assignment of a new bug report to a small number of developers. Precision levels of 57% and 64% were obtained for the Eclipse and Firefox development projects. Recently, Weiss et al. [19] used a text similarity technique to predict bug-fixing effort based on title and description of bug. Their approach beat the naive approach using the defect data from the JBoss project.

This paper differs from related work in some important ways. Since software development is human-dependent, this work incorporates human factors such as competencies and available time-slots to schedule resources for bug-fixing. This is done by using models for the bugs and human-resources; moreover the use of a search-based technique such as GA is presented that can bring a near-optimal value in scheduling by balancing multiple competing objectives.

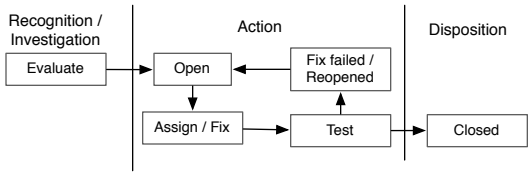

Figure 1. The bug-fixing process.

III. The bug fixing process

A typical bug-tracking system such as Bugzilla [20] keeps track of a reported bug through assigned status. So the bug is marked "new" when reported, "assigned" when assigned to a developer for fixing, "verified" when testing of the bug fix is done and "resolved" when the bug is closed. This is in line with the anomaly (bug) classification process proposed by the IEEE standard classification for software anomalies (IEEE Std 1044-1993) [21], whereby the bug life cycle is divided in to four steps: *i)* Recognition, *ii)* Investigation, *iii)* Action and *iv)* Disposition. If we assume that a bug is valid (i.e. it is not a duplicate, not incomplete/needing more information and therefore is required to be fixed), the following events describe one instance of the above four steps in more detail (also shown in Figure 1):

1) A new bug is reported in the bug database which has been evaluated as a valid bug.
2) The bug is assigned to a developer for fixing.
3) The developer fixes the bug.
4) The bug is assigned to a tester for verification.
5) The bug is verified and closed or alternatively is reopened due to an incorrect fix.

We have restricted the scope of this paper to schedule resources for a single round of these five events. So if a bug-fix from a developer fails at testing and is re-opened, a second round of events need to be taken, however, this is not dealt with in this paper since it is not known in advance how many of these round of activities would be required.

It is clear from the different bug life cycle events that given a number of bugs reported in the bug data base, there are two resource consuming activities taking place: development activity for fixing bugs and testing activity for verifying these bug-fixes. The criticality and resource demands for various bugs require resources with desired competencies and skills; moreover this has to be balanced with availability/ workload of resources for getting the job done. Due to all that common constraint of limited resources to play with and engagement of resources in multiple projects concurrently, the bug-fixing events are in a contention for finding resources that have the availability and competence to fix/ verify reported bugs. To schedule the capable and available

resources, to balance the competing objectives and to bring near-optimal value by using scheduling, we need a degree of formalism to describe the reported bugs and required human resources. This is done by describing a bug and a human resource model.

A. The bug model

This paper only focuses on the bugs found during system testing. This is however more of a constraint rather than a rule and our proposed methodology should be equally applicable to bugs found at other testing levels.

We define a bug data repository, BR, as a collection of reported bugs, $BR = \{B_1, B_2, \ldots, B_n\}$, where each bug B_i in this repository has the following attributes:

1) *Bug ID:* A unique identification of the bug.
2) *Bug description:* A short description of the bug.
3) *Bug severity:* The perceived impact of the bug, having possible values of High, Medium and Low.
4) *Bug priority:* A classification indicating the order in which the bugs are addressed, having possible values of High, Medium and Low.
5) *Required skills:* The skills required for bug-fixing, which are used to select the candidate resources. Each required skill is described as a triplet (SKT, SKN, SKL), where
   a) SKT: The type of required skills, which in our context are two, namely development skills and testing skills.
   b) SKN: The name of a specific required skill, e.g. , programming language skills for the skill type: development and test design skills for the skill type: testing.
   c) SKL: The minimum required competency in a particular skill for fixing/verifying the bug, having possible values of Low, Medium and High.

   It is to be noted that the definition of skills required to fix/verify a bug would be different across software companies, however the skill structure defined above is flexible to incorporate different skill types.
6) *Estimated effort for fixing the bug:* The estimated required effort, in number of person-hours, to be invested in development and testing of the bug-fix. Note that this estimated required effort is for one round of events, as described in Section III.
7) *Assigned time:* The date when the bug-fixing activity can be started.
8) *Deadline for bug-fixing:* The date by which the bug has to be fixed and verified. This attribute is used as a constraint for scheduling.
9) *Actual bug fixing time:* The actual date when the bug fixing is finished. This includes both the actual development and the actual testing time taken by a bug for resolution. This is given by the scheduling results.
10) *Coefficient of schedule benefit (CSB):* If the actual bug fixing time is before the deadline for bug-fixing, there is an incurred benefit given by CSB which is described by the benefit for each day before the deadline.
11) *Coefficient of schedule penalty (CSP):* If the actual bug fixing time is later than the deadline, it is expected that some other work activity would get affected. Thus there is a penalty, CSP, involved in this case for each day used up later than the deadline.

    The values for the coefficients CSB and CSP are configurable parameters chosen accordingly by the stakeholders. For instance, the penalty in missing a deadline might have higher impact than the benefit in fixing the bug before deadline; so CSP might get a higher stakeholder value than CSB.

From the above attributes we can determine the value of each bug based on the following value function:

$$Value(B_i) = f(B_i.priority, B_i.severity, B_i.deadline, B_i.actual\_fixing\_time, B_i.CSB, B_i.CSP)$$

Among these parameters, actual bug-fixing time is decided by the scheduling results and the others are determined by the value objectives of stakeholders before the scheduling. The summation of the values of all the bugs gives us the overall value of the bug-fixing process:

$$Value(BS) = \sum_{i}^{n} Value(B_i)$$

B. The human resource model

The human resource model captures the competencies and availability of development and testing personnel to undertake bug-fixing/verification. We make use of the human resource model proposed by Xiao et al. [22] to describe different attributes of human resources in the bug-fixing process. A human resource, HR, is defined in terms of four attributes:

1) HR ID: A unique identification of the human resource.
2) SKLS: The set of skills possessed by a human resource, $SKLS = \{skl_1, skl_2, \ldots, skl_n\}$. Each skl_i $(1 \leq i \leq n)$ is defined by a triplet as $skl_i = (SKT, SKN, SKL)$. The elements in this triplet are the skill type (SKT), skill name (SKN) and skill level (SKL). For example, an experienced testing resource (SKT) might be highly competent (SKL) in a certain test design technique (SKN).
3) EXPD: The work experience figure, in number of years, for the human resource. This figure can given an indication of the skill level of a particular resource.
4) STMW: The time and the workload that can be scheduled for a human resource. STMW consists of

all free time periods and the workload per day in each of these time periods:

$$STMW = \{([T_{s1}, T_{e1}], W_1), ([T_{s2}, T_{e2}], W_2), \ldots, ([T_{sk}, T_{ek}], W_k), \}$$

where T_{si} and T_{ei} represent the start and end date of the i^{th} free time period respectively, W_i represent the workable hours per day that can be scheduled in the i^{th} free time period. The unit for W_i is person hour. For example, $([25-Mar-2010, 07-Apr-2010], 6)$ indicates that the resource is available between $25-Mar-2010$ and $07-Apr-2010$ for 6 hours per day, excluding the weekends.

If a human resource has all the skills required for fixing a bug, then depending upon the available time periods, this human resource can be scheduled for the bug-fixing task. Thus according to the human resource descriptions and skill requirements of bugs, the capable resources for each bug-fixing event/activity can be scheduled.

However the organizations might lack the required competencies for fixing certain bugs within the stipulated deadline. For example, if the verification of a bug-fix requires a high skill level of domain knowledge on part of the testing resource but the one available has medium or low skill levels. In such a scenario, we setup certain rules aimed at relaxing the skill requirements in order to provide additional candidate capable resources for bug-fixing. Thus the organization can take a risk of lowering the skill requirements in an attempt to close a bug on deadline. Following are the rules to relax the skill requirements and provide additional candidate capable resources for bug-fixing if:

- the skill level gap between what is required and what is available is less than a specific number such as '1'. This number indicates the scale of gap, so e.g. the gap is '1' if there is a requirement of high level of a certain skill but only a medium one is available.
- the number of skills possessed by resources, having levels lower than the requirement, is less than a given value, e.g., 3, that is, at most a resource is lacking in 3 skill levels than what is the requirement.

IV. Scheduling with a genetic algorithm (GA)

Scheduling resources for bug-fixing activities represent a problem with different competing constraints and even with a moderate number of bugs, the search space can become vast as the number of combinations grow. To deal with the complexity of such a combinatorial optimization problem we apply a genetic algorithm (GA) to achieve maximum possible value out of the bug-fixing process.

GA is an evolutionary algorithm that uses simulated evolution as a search strategy to evolve potential solutions and uses operators inspired by genetics and natural selection [23]. A GA encodes the candidate solutions to the search problem as finite length strings called chromosomes. The chromosomes are made up of components called genes while the values of these genes are called alleles. A fitness measure discriminates good candidate solutions from bad ones and guides the search towards feasible areas in the search space. A genetic algorithm maintains a population of solutions which is iteratively recombined and mutated to evolve successive generations of candidate solutions.

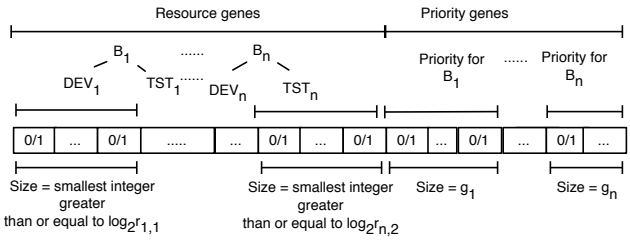

Figure 2. The bug-fixing chromosome structure.

A. Encoding and decoding of chromosome

Before encoding the scheduling problem as a GA chromosome, the following assumptions are made:

- Only one development resource and one testing resource can be allocated to each bug.
- One developer can only fix one bug at a time. Similarly one tester can verify one bug-fix at a time.

We use a binary representation of integers to encode the bug-fixing problem as a chromosome, an approach similar to the one in [8]. We establish a set of resource genes and a set of priority genes. For each bug B_i in bug repository $BR = \{B_1, B_2, \ldots, B_n\}$, the fixing of bug B_i includes two activities, development (DEV) and testing (TST). For each of these two activities, there are number of r_i capable (or additional candidate capable) resources that can be allocated to it. We encode these capable resources as a set of binary genes, where the size of the set is smallest integer greater than or equal to $log_2 r_{i,j}$ where i represents bug i and j represents activity type (DEV or TST). The binary values of these genes are used to represent the decimal number that identifies a scheduled resource as shown on the left part of Figure 2.

When two or more activities described by resource genes contend for the same resource, which activity can first acquire resources should be determined. Thus a group of genes named as priority genes are set, describing the activity priority (shown on the right part of Figure 2, where g is the priority gene size for each bug). The activity with higher priority is assigned to the resource first while if two activities have the same priority, the one placed to the left in the chromosome is assigned to the resource first.

Decoding is the reverse process of encoding. First, resource genes of each activity are decoded to a real number,

giving us scheduled resource for the activity. Second, the priority genes for each activity are decoded to a real number, giving us the priority of each activity. Third, the start time and end time for each activity is calculated. This calculation satisfies the following constraints:

1) If two activities require the same resource, the one with higher priority will be scheduled first.
2) The availability constraints of the human resources should be satisfied.

B. Multi-objective fitness evaluation of candidate solutions

Each scheduling result decoded by a chromosome is evaluated by means of a fitness function. The fitness function is designed to keep in view the scheduling objectives. Two generic and two specific objectives are taken in to consideration for scheduling. The generic objectives are:

- Objective 1: Bugs with higher priority and severity bring higher value on fixing.
- Objective 2: From a scheduling perspective, the maximum total value of fixing all the bugs should be obtained.

Besides these generic objectives, two specific objectives are used, each bringing a different value return for getting a bug fixed. One specific objective is the strict deadline objective:

- Objective 3: The deadline for each bug is strict. If a bug cannot be fixed before a deadline, the value for fixing this bug is minimum, i.e., 0. If it can be fixed before deadline, the value for fixing it is computed by its priority, severity and preference weight.

By using objectives 1, 2 and 3, the value of a bug B is described as follows:

$$Value(B) = (\alpha * priority + \beta * severity) * HasFinished(B)$$

where α and β are the preference weights for priority and severity respectively; $HasFinished(B)$ is an operator:

$$HasFinished(B) = \begin{cases} 1 & B\,is\,fixed\,before\,deadline \\ 0 & B\,cannot\,be\,fixed\,before\,deadline \end{cases}$$

The other specific objective is the relaxed deadline objective:

- Objective 4: If bug-fixing is finished before the deadline, there is an incurred benefit. If bug-fixing is finished later than deadline, a penalty is applied.

By using objectives 1, 2 and 4, the value of a bug B is described as follows:

$$Value(B) = (\alpha * priority + \beta * severity) * ScheduleValue(B)$$

where α and β are the preference weights of priority and severity respectively, while $ScheduleValue(B)$ is computed as follows:

$$ScheduleValue(B) = \begin{cases} (B.Deadline - B.FixedTime) * B.CSB \\ \quad for\,Deadline \geq FixedTime \\ (B.Deadline - B.FixedTime) * B.CSP \\ \quad for\,Deadline < FixedTime \end{cases}$$

where CSB and CSP are configurable parameters, set by the user. The strength of these coefficients indicate the impact of benefit or otherwise on the bug-fixing process so e.g. if the impact of missing a deadline is more, the corresponding coefficient is set to a higher value.

No matter whether the deadline of a bug is strict or not, the value function for the bug-fixing process is:

$$Value(BS) = \sum_{i}^{n} Value(B_i)$$

This value function is used as a fitness function during the GA evolution process.

V. Industrial case study

Our proposed methodology for scheduling resources for bug-fixing activities is evaluated using real-world data from a large Enterprise Resource Planning (ERP) software developed by a global provider of geo-technology and information technology services. The company consists of over 600 skilled professionals and have successfully been certified as CMMi level 3 compliant. The ERP project has completed several releases while the data used in this paper comes from a batch of bugs reported by the testing team for an upcoming release. This upcoming release incorporates customized functionality for one of their telecom clients. The project team working on the upcoming release have to schedule appropriate resources to cut-down the back-log of reported bugs from the testing team. The project leader plans for fixing every bug by a set deadline (keeping in view the release date for the customer) and estimates the required effort using expert judgement. The project leader is responsible for triaging the bugs to resources having the required skills (along with required skill levels) and available times. The skill set and associated levels for every resource in the project is maintained by the human resource department and the project leader also has own qualitative assessments regarding the skill levels of resources under him. The empty time slots for every resource is available through a centralized calendar application.

Therefore, having bugs with different priority, severity, time constraints, resource constraints and having resources with varying skill sets with associated skill levels and available time slots, an automated mechanism to triage bugs with maximum possible value for the organization is required.

A. Description of bugs and human resources

We evaluated our approach on a set of 25 bugs logged in the bug repository during system testing. The ID, description, severity, priority, assigned time, deadline and estimated effort are shown in Table I. The bug descriptions have been modified to protect privacy. As discussed in Section III, there are two resource consuming activities taking place during the bug-fixing process: development (DEV) and testing (TST); each of these activities require relevant skill-set. Although there can be different ways of classifying skills required for both development and testing, we use more general skill

Table I
BUG DESCRIPTIONS.

Bug ID	Bug Description	Severity (H:High, M: Medium, L:Low)	Priority (H:High, M: Medium, L:Low)	Assigned time	Deadline	Estimated effort
1	Reservations removed.	M	H	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
2	Built-in redundancy is lost.	M	H	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 4 days; 32 Hours
3	Replication too slow.	M	H	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
4	Scheduled periodic account management job not working.	H	H	25-Mar-10	12-Apr-10	DEV: 4 days, TST: 2 days; 48 Hours
5	The scheduled job cannot perform evaluation.	H	H	25-Mar-10	12-Apr-10	DEV: 4 days, TST: 2 days; 48 Hours
6	Modifying existing schedule not allowed.	H	H	01-Apr-10	19-Apr-10	DEV: 4 days, TST: 2 days; 48 Hours
7	History of customer profile not loaded.	M	H	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 4 days; 32 Hours
8	Too low processing performance.	M	H	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
9	Req002 not fulfilled.	H	H	25-Mar-10	12-Apr-10	DEV: 4 days, TST: 2 days; 48 Hours
10	Volume input/output not working.	M	H	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
11	CustomerHandler crashed.	M	M	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
12	Fallback fails in step 2 of use case 1.	H	M	25-Mar-10	12-Apr-10	DEV: 4 days, TST: 2 days; 48 Hours
13	User data not updated in customerHandler memory.	H	M	01-Apr-10	19-Apr-10	DEV: 4 days, TST: 2 days; 48 Hours
14	Failed to generate customer request.	M	M	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
15	Configuration file corrupted.	M	M	01-Apr-10	19-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
16	The configuration log is missing latest settings.	M	M	01-Apr-10	19-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
17	Too low performance for handling batch requests.	M	M	01-Apr-10	19-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
18	Page not displayed on server authentication.	L	M	25-Mar-10	12-Apr-10	DEV: 1.5 days, TST: 0.5 day; 16 Hours
19	Notification email not send.	L	M	25-Mar-10	12-Apr-10	DEV: 1.5 days, TST: 0.5 day; 16 Hours
20	Database replication error.	M	M	01-Apr-10	19-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
21	Incorrect error code.	L	L	25-Mar-10	12-Apr-10	DEV: 1.5 days, TST: 0.5 day; 16 Hours
22	Incorrect salary shown.	L	L	25-Mar-10	12-Apr-10	DEV: 1.5 days, TST: 0.5 day; 16 Hours
23	Report taking too long to generate.	L	L	01-Apr-10	19-Apr-10	DEV: 1.5 days, TST: 0.5 day; 16 Hours
24	Message update required.	L	L	01-Apr-10	19-Apr-10	DEV: 1.5 days, TST: 0.5 day; 16 Hours
25	Usage profile not updated.	L	L	01-Apr-10	19-Apr-10	DEV: 1.5 days, TST: 0.5 day; 16 Hours

Table II
SKILL REQUIREMENTS OF EACH BUG.

	Development skills						Testing skills						
Bug ID	Analytical	Programming language	Debugging	Refactoring	Use of IDE	Configuration management	Use of libraries and frameworks	Test planning (TP)	Test design (TD)	Test execution (TE)	Test review (TR)	use of bug tracking tool	Domain knowledge (DK)
1	H	M	H	M	M	M	M	H	H	H	M	M	H
2	H	M	H	M	M	M	M	H	H	H	M	M	H
3	H	M	H	M	M	M	M	H	H	H	M	M	H
4	H	H	H	H	M	M	M	H	H	H	M	M	H
5	H	H	H	H	M	M	M	H	H	H	M	M	H
6	H	H	H	H	M	M	M	H	H	H	M	M	H
7	H	M	H	M	M	M	M	H	H	H	M	M	H
8	H	M	H	M	M	M	M	H	H	H	M	M	H
9	H	H	H	H	M	M	M	H	H	H	M	M	H
10	H	M	H	M	M	M	M	H	H	H	M	M	H
11	H	M	H	M	M	M	M	H	H	H	M	M	H
12	H	H	H	H	M	M	M	H	H	H	M	M	H
13	H	H	H	H	M	M	M	H	H	H	M	M	H
14	H	M	H	M	M	M	M	H	H	H	M	M	H
15	H	M	H	M	M	M	M	H	H	H	M	M	H
16	H	M	H	M	M	M	M	H	H	H	M	M	H
17	H	M	H	M	M	M	M	H	H	H	M	M	H
18	M	M	M	L	L	L	L	M	M	M	L	L	M
19	M	M	M	L	L	L	L	M	M	M	L	L	M
20	H	M	H	M	M	M	M	H	H	H	M	M	H
21	M	M	M	L	L	L	L	M	M	M	L	L	M
22	M	M	M	L	L	L	L	M	M	M	L	L	M
23	M	M	M	L	L	L	L	M	M	M	L	L	M
24	M	M	M	L	L	L	L	M	M	M	L	L	M
25	M	M	M	L	L	L	L	M	M	M	L	L	M

requirements that could easily be mapped to more specific skill-set at our subject company. The skill requirements for each bug are described in Table II where H: High, M: Medium and L: Low. Human resource attributes for available development and testing personnel (as discussed in Section III-B) are shown in Table III.

VI. THE SCHEDULING RESULTS

We applied the GA proposed in Section IV to schedule capable resources for bug-fixing activities, based on the bug model and human resource model data given in Section III. The GA used the following parameters: Population size: 100, total number of generations: 500, cross-over rate: 0.8, mutation rate: 0.01, selection method: ratio. These parameters were obtained after some experimentation, however, in future we need a more systematic mechanism of tuning them.

We assume that delaying the bug-fixing after the deadline has greater impact than fixing it before, therefore, CSB is set as 10 and CSP as 30 for every bug, i.e., one day delay in bug-fixing has three times effect on the value than completing the bug-fixing one day before. Based on the configuration of coefficients and weights in balancing objectives (Section IV-B) and resources (Section V-A), different scenarios suggest strategies for managing resources for the bug-fixing tasks. We then discuss the scheduling results out of these scenarios.

A. Scenario 1: Priority preference weight, α = 20; Severity preference weight, β = 5

With priority weight, α, set to 20 and severity preference weight, β, set to 5, we first use objectives 1, 2 and 3 from Section IV-B. That is, we use the strict deadline as an objective and find that, using data from Section V-A, only 11 out of 25 bugs can be scheduled for fixing. These bugs are listed in Table IV and the corresponding Gantt chart plan for fixing these bugs is shown in Figure 3. Gantt chart is an easy way to illustrate a project schedule and provides an intuitive interface for the project leader to monitor scheduling elements. As is clear that using a strict deadline objective, only a limited number of bugs can be fixed.

We now use the relaxed deadline objective to schedule more bugs by relaxing the deadline constraint. We assume that all the resources are available after 20-Apr-2010 and

Table III
HUMAN RESOURCE DESCRIPTIONS.

HR ID	(SKT, SKN, SKL)	EXPD (Years)	STMW
HR1	(Developer, Analytical, H)	6	([25-Mar-2010, 07-Apr-2010], 6)
	(Developer, Programming Lang., H)		
	(Developer, Debugging, H)		
	(Developer, Refactoring , H)		([12-Apr-2010, 15-Apr-2010], 6)
	(Developer, IDE, M)		
	(Developer, CM, M)		
	(Developer, Lib. and Frameworks, M)		
HR2	(Developer, Analytical, H)	6	([20-Mar-2010, 04-Apr-2010], 8)
	(Developer, Programming Lang., H)		
	(Developer, Debugging, H)		
	(Developer, Refactoring , H)		([12-Apr-2010, 14-Apr-2010], 8)
	(Developer, IDE, M)		
	(Developer, CM, M)		
	(Developer, Lib. and Frameworks, M)		
HR3	(Developer, Analytical, M)	2	([23-Mar-2010, 12-Apr-2010], 8)
	(Developer, Programming Lang., M)		
	(Developer, Debugging, M)		
	(Developer, Refactoring , M)		([15-Apr-2010, 22-Apr-2010], 8)
	(Developer, IDE, M)		
	(Developer, CM, L)		
	(Developer, Lib. and Frameworks, L)		
HR4	(Developer, Analytical, M)	2	([23-Mar-2010, 05-Apr-2010], 6)
	(Developer, Programming Lang., M)		
	(Developer, Debugging, M)		
	(Developer, Refactoring , M)		([12-Apr-2010, 17-Apr-2010], 6)
	(Developer, IDE, M)		
	(Developer, CM, L)		
	(Developer, Lib. and Frameworks, L)		
HR5	(Tester, TP, H)	4	([25-Mar-2010, 08-Apr-2010], 4)
	(Tester, TD, H)		
	(Tester, TE, H)		
	(Tester, TR, M)		([12-Apr-2010, 14-Apr-2010], 4)
	(Tester, Bug Tracking Tool, M)		
	(Tester, DK, M)		
HR6	(Tester, TP, M)	2	([25-Mar-2010, 06-Apr-2010], 3)
	(Tester, TD, M)		
	(Tester, TE, M)		
	(Tester, TR, L)		([09-Apr-2010, 16-Apr-2010], 3)
	(Tester, Bug Tracking Tool, L)		
	(Tester, DK, M)		

Table IV
BUGS THAT CAN BE FIXED UNDER A STRICT DEADLINE (α=20, β=5).

Bug ID	Value	DEV	TST
2	70	HR2: ([25-Mar-2010, 26-Mar-2010], 8) ([29-Mar-2010, 29-Mar-2010], 8)	HR5: ([30-Mar-2010, 31-Mar-2010], 4)
3	70	HR2: ([30-Mar-2010, 01-Apr-2010], 8)	HR5: ([06-Apr-2010, 07-Apr-2010], 4)
9	75	HR1: ([29-Mar-2010, 01-Apr-2010], 6)	HR5: ([02-Apr-2010, 02-Apr-2010], 4) ([05-Apr-2010, 05-Apr-2010], 4)
10	75	HR1: ([02-Apr-2010, 02-Apr-2010], 6) ([05-Apr-2010, 07-Apr-2010], 6)	HR5: ([08-Apr-2010, 08-Apr-2010], 4) ([12-Apr-2010, 12-Apr-2010], 4)
18	45	HR4: ([25-Mar-2010, 26-Mar-2010], 6)	HR5: ([29-Mar-2010, 29-Mar-2010], 4)
19	45	HR3: ([25-Mar-2010, 26-Mar-2010], 8)	HR6: ([29-Mar-2010, 30-Mar-2010], 3)
21	25	HR1: ([25-Mar-2010, 26-Mar-2010], 6)	HR6: ([31-Mar-2010, 01-Apr-2010], 3)
22	25	HR3: ([29-Mar-2010, 30-Mar-2010], 8)	HR5: ([01-Apr-2010, 01-Apr-2010], 4)
23	25	HR3: ([01-Apr-2010, 02-Apr-2010], 8)	HR6: ([05-Apr-2010, 06-Apr-2010], 3)
24	25	HR3: ([05-Apr-2010, 06-Apr-2010], 8)	HR5: ([13-Apr-2010, 13-Apr-2010], 4)
25	25	HR3: ([07-Apr-2010, 08-Apr-2010], 8)	HR6: ([09-Apr-2010, 09-Apr-2010], 3) ([12-Apr-2010, 12-Apr-2010], 3)

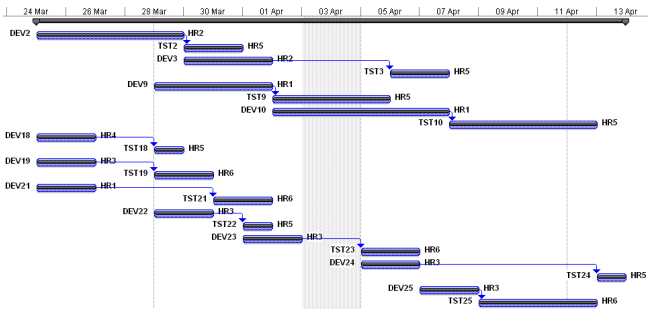

Figure 3. Strict deadline bug-fixing plan.

Table V
BUGS THAT CAN BE FIXED UNDER A RELAXED DEADLINE (α=20, β=5).

Bug ID	Value	Precedent days compared to the deadline	Bug ID	Value	Precedent days compared to the deadline
1	-18900.0	-9	14	-37500.0	-25
2	-23100.0	-11	15	-24000.0	-16
3	4900.0	7	16	-33000.0	-22
4	-4500.0	-2	17	-42000.0	-28
5	-40500.0	-18	18	4950.0	11
6	-33750.0	-15	19	4500.0	10
7	-21000.0	-10	20	-25500.0	-17
8	6300.0	9	21	2000.0	8
9	3750.0	5	22	1500.0	6
10	-16800.0	-8	23	1750.0	7
11	-31500.0	-15	24	1250.0	5
12	-52800.0	-32	25	2250.0	9
13	-51150.0	-31			

each workday comprises of 8 working hours. Using objectives 1, 2 and 4 from Section IV-B, the simulation results appear in Table V. The data in Table V indicates that relaxing the deadline enables all the bugs to be scheduled for fixing but many of them are delayed as indicated by negative integers in the third and sixth columns of Table V. The corresponding Gantt chart plan is shown in Figure 4 that can help show the project leader that negotiating a relaxation in deadline would help fix all the bugs.

B. Scenario 2: Priority preference weight, α = 5; Severity preference weight, β = 20

In this scenario, we change the priority and severity preference weights as α = 5 and β = 20 respectively; that is to say that we now consider severity as more important than the priority of a bug. Using the strict deadline as an objective, the simulation results indicate that there are still 11 out of 25 bugs that can be scheduled before deadline. Although the total number of bugs that could be fixed before deadline remains the same for both the scenarios, a comparison of two schedules indicate that the two scheduling plans differ at bug IDs 3, 5, 10 and 16. This is shown in Table VI. The data in Table VI show that increasing the severity preference weight β (scenario 2) has resulted in scheduling bug ID 5 with highest priority but at the cost of not fixing bug IDs 3 and 10 from scenario 1. Since bug IDs 3 and 10 cannot be fixed, the available resources are enough to fix bug ID 16.

Scenarios 1 and 2 indicate that by plugging different combinations of priority and severity preference weights, the project leader can balance the importance of fixing certain bugs at the cost of others (provided that the deadline is strict). This, in our view, suggest valuable strategies for resource scheduling.

C. Scenario 3: Priority preference weight, α = 20; Severity preference weight, β = 5; Simulating virtual resources

In the previous two scenarios we see that, using a strict deadline, the resources are not enough to fix all the bugs on or before the deadline. We also saw that one way to

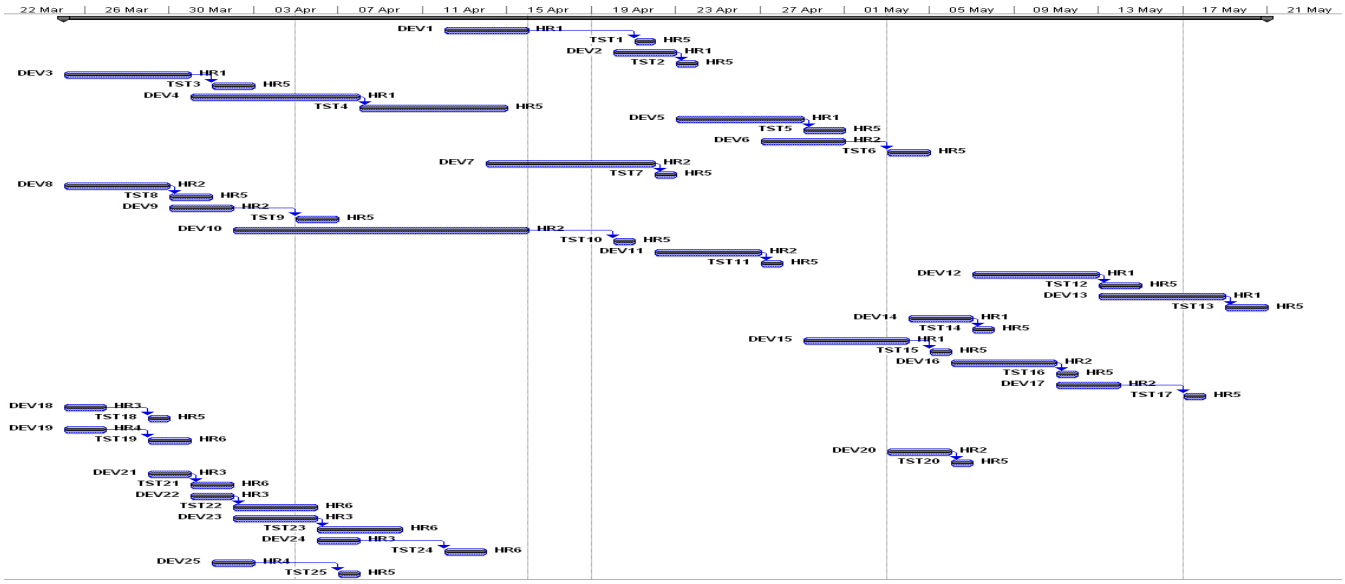

Figure 4. Relax deadline bug-fixing plan.

Table VI
COMPARISON OF BUG-FIXING SCHEDULES UNDER DIFFERENT PRIORITY AND SEVERITY PREFERENCE WEIGHTS.

Bug ID	Priority	Severity	Value preference weight	
			$\alpha = 20; \beta = 5$	$\alpha = 5; \beta = 20$
2	3	2	70	55
3	3	2	70	0
5	3	3	0	75
9	3	3	75	75
10	3	2	75	0
16	2	2	0	50
18	2	1	45	30
19	2	1	45	30
21	1	1	25	25
22	1	1	25	25
23	1	1	25	25
24	1	1	25	25
25	1	1	25	25

provide more candidate resources is to relax the deadline. The other way to achieve the deadline is, of course, addition of more resources. Therefore, we add virtual resources for fixing bugs in this scenario. Initially we have 6 resources and are able to schedule 11 out of 25 bugs for fixing. Increasing the resources to two more by adding one development resource and one testing resource, with high skill levels in all skills, enables scheduling over 15 bugs for fixing. Similarly increasing the resources to 10 by adding two highly-skilled development resources and two highly-skilled testing resources allow scheduling more than 20 bugs before deadline (Figure 5).

This scenario gives another option to the project leader for viewing the scheduling outcome if more resources were available than initially assigned. Therefore, the simulation

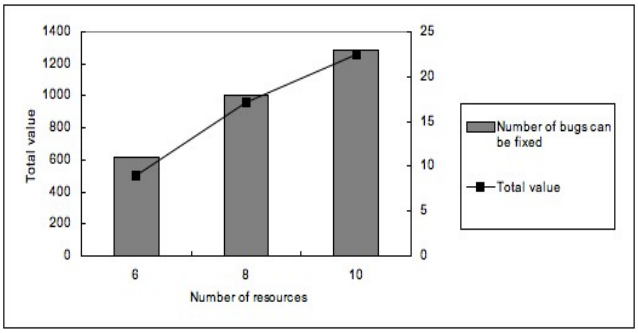

Figure 5. Effects of increasing number of resources.

of virtual resources can provide schedules under varying circumstances, keeping in view the available resource pool.

VII. COMPARISON WITH HILL-CLIMBING SEARCH

Hill-climbing (HC) is a basic local search algorithm and, likewise GA, is used to compute the value obtained in scheduling resources for bug-fixing tasks. We have used the three scenarios discussed in Section VI to compute the total bug value by using HC and have compared it with GA. The results are shown in Figure 6. The figure shows that if the number of bugs is small (i.e. 1 to 3), GA and HC obtain the same optimal value. But when the scale of problem increases with an increase in number of bugs, GA gives better results than HC. In order to test whether any significant differences exist between the bug values from two algorithms, we used Wilcoxon rank sum test. The p-value of 0.004 confirmed that the bug values from HC and GA do not have equal

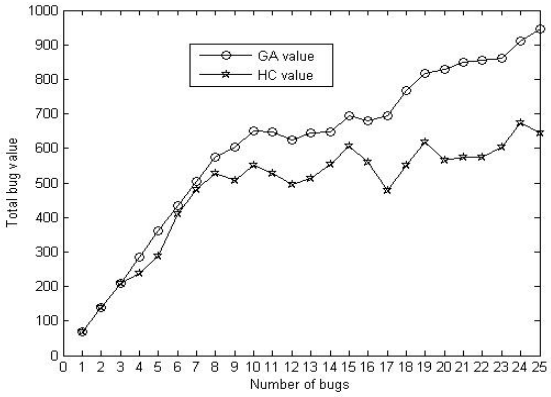

Figure 6. A comparison of total bug value obtained by using GA and HC.

medians at 0.05 significance level. Thus bug values from GA are significantly different and better than those of HC.

VIII. Discussion

Considering that we have a need to support efficient and effective bug-assignment policies, this paper has provided early results as to how a GA can help strike a balance between competing constraints to achieve near-optimal value for the organization. Due to the dynamic nature of the bug-fixing schedules, different scenarios are possible and these changing scenarios have to be modeled effectively for near-optimal solutions. The multi-objective fitness function proposed in this work attempts to model the uncertainty in the scheduling problem. The scenarios discussed here provide a way to schedule resources under different circumstances, e.g., having a strict/flexible deadline, having assigned different weights to severity and priority and last but not least, the ability to foresee the resource requirements by adding virtual resources to meet the deadline. GA is able to effectively suggest different strategies to tackle the bug-fixing process and is found to be more effective than hill-climbing. Consequently the project leaders can use these results to support their resource scheduling decisions.

We are also aware of certain limitations of our study. First we have some assumptions that might get violated, e.g., it is common that the bug-fix is verified not to be correct by the testing team and a second round of bug-fixing activities is undertaken. If this is the case, then the different elements of the bug model would require new data for the second round of activities. We, however, limit ourselves to only one round of bug-fixing activities in this paper. Second, there are some rules that are followed for the relaxed deadline objective. While these rules would differ with respect to the expectations of the project leader of her team members, we followed some intuitive ones. Any change in this rules is, however, possible. Third, there is a possibility that a resource works concurrently on more than one assignment. However we only consider the empty time slots that a particular resource has for dealing with one bug at a time. If such a division of workload is not possible then it is expected that the human resource model needs to incorporate this change. Fourth, a company might face the difficulty to quantify the skills and the associated levels. As our subject company is on the path of CMMi Level 4, such a quantification is seen as a continuing improvement opportunity for the workforce. The human resource model presented here uses a simple classification of skills which can be changed to suit specific needs. There is a possibility that the human resource model in this paper has ignored relevant human performance factors. An important point to make here is that the company using such an approach needs to continuously update the skill database of its resources since it is common for the resources to educate themselves and learn as part of the project experience.

IX. Validity threats

There can be three types of validity threats to the kind of study we have conducted. *Construct validity* threats refer to the extent the experiment setting actually reflects the construct under study [24]. These threats might arise due to the assumptions we made in the study and the way we modeled the problem. However, a search-based technique such as GA is independent of the way the problem is modeled; it is the fitness function that contains the crucial information and needs to be adapted for a more complex model. The assumptions in this paper made sense for the type of case study discussed, however, as mentioned in Section VIII, the bug and human resource model might change if a different process of bug-fixing is followed. *Internal validity* threats refer to any sources of bias that might have affected the results. Since GA is a stochastic algorithm, different runs produces different solutions. The different GA parameters were obtained after careful experimentation and taking into account that further changing the parameter values do not have significant impact on the results. Moreover, the GA was run multiple times (30) to overcome randomness inherent in the GA. *External validity* threats are concerned with generalization. The results obtained in this paper as such should be applicable to the situations where our assumptions are held. Otherwise, the bug and the human resource model needs to be adapted accordingly.

X. Conclusions and future work

We have presented a search-based resource allocation method for bug-fixing tasks. We proposed models for the bug-fixing process, the human resources and the capability matching method between bug-fixing activities and human resources. On the basis of these models and our proposed method, the resources were allocated for bug-fixing activities using a GA. Depending on differing objectives, three scenarios were discussed using an industrial data set and the results

showed that GA was able to give schedules having balanced different objectives and entailing maximum value for the organization. Comparison with hill-climbing showed that GA gave statistically better results in terms of maximizing the value objective.

Based on this paper, some interesting future work can be undertaken:

- Combining the bug-fixing process with other resource consuming activities that might happen concurrently, e.g., testing of newly implemented requirements might take place in parallel with bug-fixing activities, probably needing similar resources.
- Increasing the generalizability of the proposed method by considering scheduling a larger set of bugs.
- Supplementing the scheduling with cost issues, i.e., the cost incurred in investing resources to perform different activities might impact the value objective.
- Analyzing the sensitivity of parameters in the GA and the fitness function, such as population size of the GA, priority preference weight and severity preference weight of the fitness function.

XI. Acknowledgements

Junchao Xiao is supported by the National Natural Science Foundation of China under grant Nos. 90718042, 60903051, the 863 Program of China under grant No. 2007AA010303, as well as the 973 program under grant No. 2007CB310802. Wasif Afzal is supported by the Swedish research school in verification and validation (www.swell.se).

References

[1] H. Pham, *Software reliability*. Singapore: Springer-Verlag, 2000.

[2] J. Anvik, L. Hiew, and G. Murphy, "Who should fix this bug?" in *Procs. of the Int. Conf. on Software Engineering*, 2006.

[3] P. Laplante and N. Ahmad, "Pavlov's bugs: Matching repair policies with rewards," *IT Professional*, vol. 11, no. 4, 2009.

[4] M. B. Wall, "A GA for resource-constrained scheduling," Ph.D. dissertation, Dept. of Mechanical Eng., Massachusetts Institute of Technology, USA, 1996.

[5] L. Ozdamar and G. Ulusoy, "A survey on the resource-constrained project scheduling problem," *IIE Transactions*, vol. 27, pp. 574–586, 1995.

[6] M. Harman, S. A. Mansouri, and Y. Zhang, "Search based software engineering: A comprehensive analysis and review of trends techniques and applications," Dept. of CS, King's College London, Tech. Report TR-09-03, 2009.

[7] G. Antoniol, M. Di Penta, and M. Harman, "Search-based techniques applied to optimization of project planning for a massive maintenance project," in *Procs. of the 21st IEEE Int. Conf. on SW Maintenance*. IEEE, 2005.

[8] J. Xiao, Q. Wang, M. Li, Q. Yang, L. Xie, and D. Liu, "Value-based multiple software projects scheduling with GA," in *Procs. of the 2009 Int. Conf. on SW Process*, 2009.

[9] E. Alba and J. Chicano, "SW project management with GAs," *Information Sciences*, vol. 177, no. 11, pp. 2380–2401, 2007.

[10] C. Chang, H. Jiang, Y. Di, D. Zhu, and Y. Ge, "Time-line based model for SW proj. scheduling with genetic algorithms," *IST*, vol. 50, no. 11, pp. 1142–1154, 2008.

[11] C. Kirsopp, M. J. Shepperd, and J. Hart, "Search heuristics, case-based reasoning and software project effort prediction," in *Procs. of the Genetic and Evolutionary Computation Conf.* Morgan Kaufmann Publishers Inc., 2002.

[12] W. Afzal, R. Torkar, R. Feldt, and T. Gorschek, "Genetic programming for cross-release fault count predictions in large and complex software projects," in *Evolutionary Computation and Optimization Algorithms in Software Engineering*, M. Chis, Ed. IGI Global, Hershey, USA, 2010.

[13] E. Hart, P. Ross, and D. Corne, "Evolutionary scheduling: A review," *Genetic Programming and Evolvable Machines*, vol. 6, no. 2, pp. 191–220, 2005.

[14] A. Mockus, D. M. Weiss, and P. Zhang, "Understanding and predicting effort in software projects," in *Procs. of the 25th Int. Conf. on SW Eng.* IEEE Computer Society, 2003.

[15] D. Cubranic and G. Murphy, "Automatic bug triage using text categorization," in *Procs. of the 16th Int. Conf. on SW Eng. and Knowledge Eng.*, 2004.

[16] H. Zeng and D. Rine, "Estimation of software defects fix effort using neural networks," in *Procs. of the 28th Annual Int. Computer SW and Applications Conf. - Workshops and Fast Abstracts*. IEEE Computer Society, 2004.

[17] G. Canfora and L. Cerulo, "Supporting change request assignment in open source development," in *Procs. of the 2006 ACM symp. on Applied computing*. ACM, 2006.

[18] Q. Song, M. Shepperd, M. Cartwright, and C. Mair, "SW defect association mining and defect correction effort prediction," *IEEE Trans. on SW Eng.*, vol. 32, no. 2, 2006.

[19] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *Procs. of the 4th Int. WS on Mining SW Repositories*. IEEE, 2007.

[20] "Bugzilla – A bug tracking tool." http://www.bugzilla.org/, Last checked 21 Mar 2010.

[21] *IEEE std. classification for SW anomalies, IEEE Std. 1044-1993*, IEEE, Inc., USA, 1993.

[22] J. Xiao, Q. Wang, M. Li, Y. Yang, F. Zhang, and L. Xie, "A constraint-driven human resource scheduling method in software development and maintenance process," in *24th IEEE Int. Conf. on SW Maintenance*. IEEE, 2008.

[23] J. Holland, *Adaptation in natural and artificial systems*. Cambridge, MA, USA: MIT Press, 1992.

[24] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering: An introduction*. USA: Kluwer Academic Publishers, 2000.