International Journal of Modern Achievement in Science, Engineering and Technology (IJSET)
ISSN 3023-459X
DOI : 10.63053/ijset.1

A Study on Identifying, Finding and Classifying Software Bugs Using Data Mining Methods

Mohammad Beheshti[1]

1.Bachelor of Computer Science, Computer Software Technology Engineering, Faculty of Electrical Engineering and Computer Engineering, Islamic Azad University, Germi Branch, Iran

ARTICLE INFO

Keywords: *Software Bugs Tracking, Data Mining Techniques, Software Quality Assurance, Bug Tracking, Bug Classification*

ABSTRACT

A software bug is an error, flaw or fault in the design, development, or operation of computer software that causes it to produce an incorrect or unexpected result, or to behave in unintended ways. The process of finding and correcting bugs is termed "debugging" and often uses formal techniques or tools to pinpoint bugs. Since the 1950s, some computer systems have been designed to deter, detect or auto-correct various computer bugs during operations. Bugs in software can arise from mistakes and errors made in interpreting and extracting users' requirements, planning a program's design, writing its source code, and interaction with humans, hardware and programs, such as operating systems or libraries. A program with many, or serious, bugs is often described as buggy. Bugs can trigger errors that may have ripple effects. The effects of bugs may be subtle, such as unintended text formatting, through to more obvious effects such as causing a program to crash, freezing the computer, or causing damage to hardware. Other bugs qualify as security bugs and might, for example, enable a malicious user to bypass access controls in order to obtain unauthorized privileges. Some software bugs have been linked to disasters. Bugs in code that controlled the Therac-25 radiation therapy machine were directly responsible for patient deaths in the 1980s. In 1996, the European Space Agency's US$1 billion prototype Ariane 5 rocket was destroyed less than a minute after launch due to a bug in the on-board guidance computer program. In 1994, an RAF Chinook helicopter crashed, killing 29; this was initially blamed on pilot error, but was later thought to have been caused by a software bug in the engine-control computer. Buggy software caused the early 21st century British Post Office scandal, the most widespread miscarriage of justice in British legal history. In 2002, a study commissioned by the US Department of Commerce's National Institute of Standards and Technology concluded that "software bugs, or errors, are so prevalent and so detrimental that they cost the US economy an estimated $59 billion annually, or about 0.6 percent of the gross domestic product". In this paper, software defect detection and classification method is proposed and data mining techniques are integrated to identify, classify the defects from large software repository.

Introduction

In computer technology, a bug is a coding error in a computer program. (We consider a program to also include the microcode that is manufactured into a microprocessor.) The process of finding bugs -- before users do -- is called debugging. Debugging starts after the code is written and continues in stages as code is combined with other units of programming to form a software product, such as an operating system or an application.

Bugs are often discovered after a product is released or during public beta testing. When this occurs, users have to find a way to avoid using the buggy code or get a patch from the software developers.

The Middle English word bugge is the basis for the terms "bugbear" and "bugaboo" as terms used for a monster.

The term "bug" to describe defects has been a part of engineering jargon since the 1870s and predates electronics and computers; it may have originally been used in hardware engineering to describe mechanical malfunctions. For instance, Thomas Edison wrote in a letter to an associate in 1878:

... difficulties arise—this thing gives out and [it is] then that "Bugs"—as such little faults and difficulties are called—show themselves

Baffle Ball, the first mechanical pinball game, was advertised as being "free of bugs" in 1931. Problems with military gear during World War II were referred to as bugs (or glitches). In a book published in 1942, Louise Dickinson Rich, speaking of a powered ice cutting machine, said, "Ice sawing was suspended until the creator could be brought in to take the bugs out of his darling."

Isaac Asimov used the term "bug" to relate to issues with a robot in his short story "Catch That Rabbit", published in 1944.

A page from the Harvard Mark II electromechanical computer's log, featuring a dead moth that was removed from the device

The term "bug" was used in an account by computer pioneer Grace Hopper, who publicized the cause of a malfunction in an early electromechanical computer. A typical version of the story is:

In 1946, when Hopper was released from active duty, she joined the Harvard Faculty at the Computation Laboratory where she continued her work on the Mark II and Mark III. Operators traced an error in the Mark II to a moth trapped in a relay, coining the term bug. This bug was carefully removed and taped to the log book. Stemming from the first bug, today we call errors or glitches in a program a bug.

Hopper was not present when the bug was found, but it became one of her favorite stories. The date in the log book was September 9, 1947. The operators who found it, including William "Bill" Burke, later of the Naval Weapons Laboratory, Dahlgren, Virginia, were familiar with the engineering term and amusedly kept the insect with the notation "First actual case of bug being found." This log book, complete with attached moth, is part of the collection of the Smithsonian National Museum of American History.

The related term "debug" also appears to predate its usage in computing: the Oxford English Dictionary's etymology of the word contains an attestation from 1945, in the context of aircraft engines.

The concept that software might contain errors dates back to Ada Lovelace's 1843 notes on the analytical engine, in which she speaks of the possibility of program "cards" for Charles Babbage's analytical engine being erroneous:

... an analysing process must equally have been performed in order to furnish the Analytical Engine with the necessary operative data; and that herein may also lie a possible source of error. Granted that the actual mechanism is unerring in its processes, the cards may give it wrong orders.

A Software defects/bugs can be defined as “state in a software product that fail to meet the software requirement specification or customer expectations”. In other words defect is a failure in coding or logic error that makes a program to produce unexpected results. In software development life cycle (SDLC), defect management is the integral part of development and testing phases. Defect management process consists of defect prevention, defect identification, defect reporting, defect classification and prediction. Defect prevention activities are to find errors in software requirements and design documents, to review the algorithm implementations, defects logging and documentation, root cause analysis. Defect identification is to identity the code rule violations as the code is developed, changed and modified. Defect reporting is to clearly describe the problem associated with particular module in software product so that developer can fix it easily. Defect classification is the process of classifying the defects based on severity to show the scale of negative impact on quality of software. Defect prediction objective is to predict number of defects or bugs in software product, before the deployment of software product, also to estimate the likely delivered quality and maintenance effort. Defects can be in different phases of SDLC, table 1 shows the percentage of defect introduced in each phase.

Table 1. Division of percentage of Defects Introduced into Software in each Phase of SDLC, (source: Computer Finance Magazine)

Software Development Phases	Percent of Defects Introduced
Requirements	20 %
Design	25 %
Coding	35 %
User Manuals	12 %
Bad Fixes	8 %

Nowadays different data mining techniques are used in defect management to extract useful data from software historical data. Data mining is the process of extracting patterns from data. Data mining, or knowledge discovery, is the computer-assisted process of digging through and analyzing enormous sets of data and then extracting the meaning of the data. A Typical knowledge discovery process (KDP) is shown in Figure 2. KDP may consist of the following steps: data selection, data cleaning, data transformation, pattern searching (data mining), and finding presentation, finding interpretation and evaluation. Data collection phase is to extract the data relevant to data mining analysis. The data should be stored in a database where data analysis will be applied.

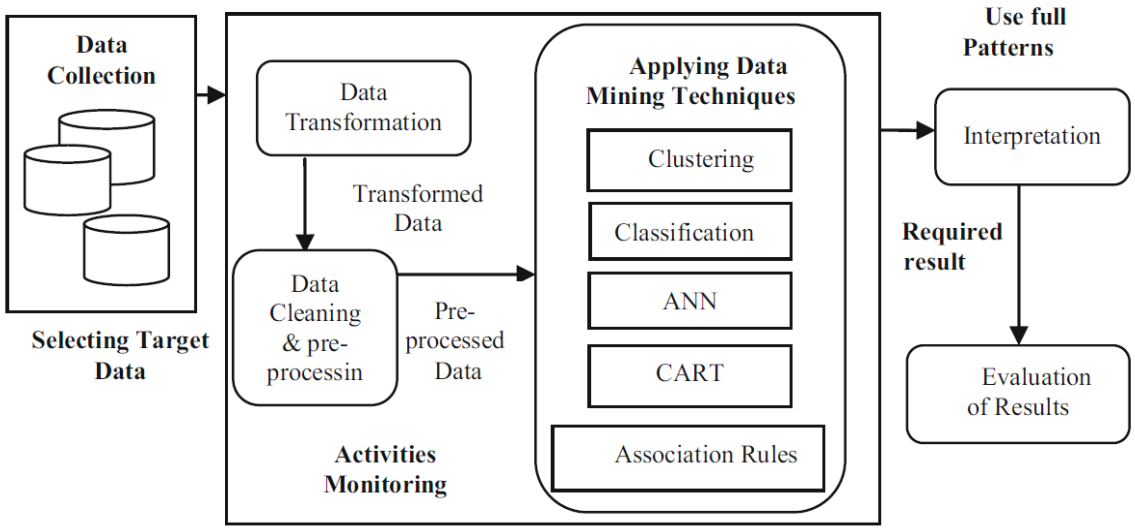

Fig. 1. Knowledge Discovery Process

Data cleaning and preprocessing and data transformation phase of KDP involves data cleansing and preparation of data, converting the data suitable for processing and obtaining valid results to achieve the desired results. Activity monitoring module deals with real time information. The purpose of data mining (DM) phase is to analyze the data using appropriate algorithms to discover meaningful patterns and rules to produce predictive models. Data mining is the most important phase of KDP cycle. After building data warehouse data mining algorithms such as clustering, classification, artificial neural networks, rule association, decision tree and classification and regression trees (CART) or chi-square automatic induction (CHAID) are applied.

Interpretation and evaluation of results is the final phase involves making useful decisions by interpreting and evaluating results obtained from applying knowledge discovery techniques. Developing a software product without defect is difficult, project managers and software engineers put more effort to minimize the defects. If the defects are detected after delivering the software product the project budget and resources cost will be more compared to cost of early detection. Software defects can be tracked and managed using software tools such as Bugzilla [15] Perforce [16], Trac [17], Fossil [18]. Software repositories contains historical data such as code bases, execution traces, historical code changes, contains a wealth of information about a software project's status, progress, and evolution [1].

Software defects data will be in formats like CSV (comma separated value), HTML (Hyper Text Markup Language) or XML (Extensible Markup Language) as software repositories. New technologies and techniques are required to reuse the existing knowledge from software repositories. This paper is organized as follows: Section 2 gives detailed survey of different knowledge discovery techniques applied to various aspects of the defect management activities while section 3 proposed method to identify and classify the defects. Section 4 presents the results we have obtained so far. Finally, in conclusion and future work, we conclude the paper and present our goals for future research.

2. Related Work

Several research works has been carried out on defect management process by many authors. Honar and Jahromi [13] introduced a new framework for call graph construction to be used for program analyses , they choose (ASM and Soot) as a byte code reader for their environment to store information about the structure of the codes such as classes, methods, files and statements. Kim et. al [2] proposed new techniques for classifying the change requests as buggy or clean. Antoniol et al. [3] highlighted that not all bug reports are associated to software problems, but also change request bug reports. Fluri et. al. [4], proposed an semi-automated approach using agglomerative hierarchical clustering to discover patterns from source code changes types.

Fenton et. al [16], presented an empirical study modeled by pareto principle on two versions of large-scale industrial software and shown the results of distribution failure and faults in software product. Jalbert and Weimer [5] have proposed method to identify he duplicate bug reports automatically from software bug repositories. Cotroneo et. al. [6] performed failure analysis of Java Virtual Machine (JVM).

Guo et al [9] explored factors which are affecting in fixing the bugs for Windows Vista, and Windows 7 systems. Davor et. al [8], have proposed an approach for automatic bug tracking using text categorization. They proposed a prototype for bug assignment to developer using supervised Bayesian learning algorithm. The evaluation shows that their prototype can correctly predict 30% of the report assignments to developers. Lutz et. al [12], analyzed the causes and impact of defects in requirements during testing phase, resulting from non documented changes and guidelines are given to respond to the situations. Kalinowski et al [7], aware that defect rates can be reduced by 50%, rework can be reduced, quality and performance is improved using Defect Causal Analysis (DCA). Then DCA is enhanced and named it as Defect Prevention Based Process Improvement (DPPI) to conduct, measure.

3. Proposed Method

The proposed method basically classified into four major steps: defects identification phase, applying data mining techniques and defects classification based on severity measures. The block diagram of proposed method is as shown in Fig 2.

We have collected online available Promise repository NASA MDP software defect data sets [20]. Defect data sets are retrieved and stored in the file system. For pre-processing the defect data attributes are parsed and some attributes are selected for measurements using various software metrics. Based on bugs severity proposed method is divided into three layers: core, abstraction and application layer. Core layer contains the core functionality of the system, the defects in this layer affects critical functionality of the system and leads to failure of the project. An abstraction layer defect affects major functionality or major data of the system. Application layer contains user oriented functionality responsible for managing user interaction with the system defects in this layer affects minor functionality of the system.

We have used weka 3.6 software tool to apply the data mining techniques Using Naïve Bayes, J48 and Multilayer Perceptron (MLP) classification techniques, we have applied the classification for input attributes and the setting the test mode to 10-fold validation.

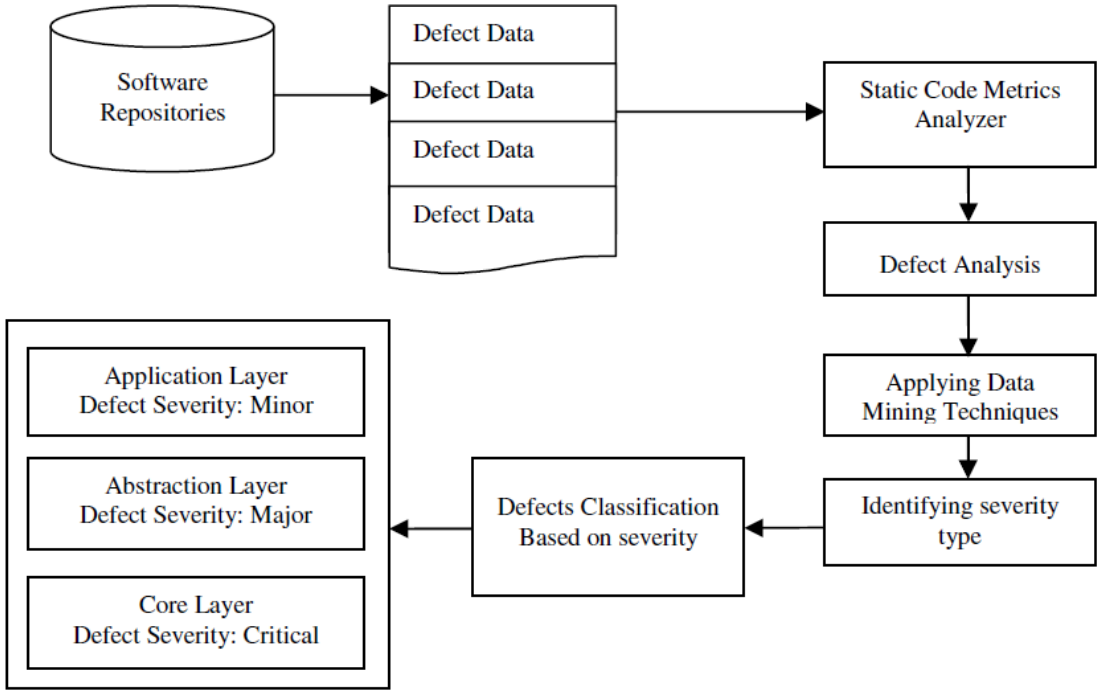

Fig. 2. Steps in defects identification and classification

4 Performance Evaluation

4.1 Accuracy

Accuracy for defect classification can be calculated as ratio of number of defects correctly classified to the total number of defects. The equations 1 and 2 are used to calculate the accuracy:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

$$Accuracy(\%) = \frac{CorrectlyClassfiedSoftwareDefects}{TotalSoftwareDefects} * 100 \quad (2)$$

Where, true positive (TP), indicates a positive value that the source code has correctly classified as positives. TN is true negatives the proposed method identified defect as negative. FP is false positive, negative values the method identified as positives and FN is false negatives, positives values that the method identified as negative.

4.2 Precision

Precision for a class is the ratio of the number of correctly classified software defects and the actual number of software defects which was assigned to the type. Precision is the measurement of correctness and is also defined as the ratio of the true positives (TP) to total positives (TP + FP) and is calculated using Equation (3).

$$\Pr ecision = \frac{TP}{TP + FP} \quad (3)$$

4.3 F-Measure

F-Measure is a measure which combines the harmonic mean of recall and precision it is calculated using equation 4:

$$F = 2 * \frac{\Pr ecision.recall}{\Pr ecision + recall} \quad (4)$$

From the NASA MDP, we have applied Naïve Bayes, J48 and Multilayer Perceptron (MLP) classification techniques on various data sets like CM1, JM1, KC1, KC3, MC1, MC2, MW1, PC1, PC2 and PC3. The performance results are shown in table 2. MLP achieves significantly better results compare to other two

classifies i.e. J48 and Naïve Bayes. Our proposed method divides the source code defects into three layers namely core, abstraction and application. Core layer corresponds to condition which can cause failure of the software products or harm the life of the end user these defects are termed as critical defects. Abstraction layer corresponds to condition which can cause major functionality of the software, these defects are termed as major defects. Application layer corresponds to condition which contains user oriented functionality responsible for managing user interaction with the system defects in this layer affects minor functionality of the system.

Steps to identify defects and classification

Step1: Collect the data sets from different defect data sources (Eg.: Promise repository, NASA MDP repository)

Step 2: static code metrics are used to measure the source code, various metrics used are:
McCabe Metrics; Cyclomatic Complexity and Design Complexity, Halstead Metrics; Halstead Content, Halstead Difficulty, Halstead Effort, Halstead Error Estimate, Halstead Length, Halstead Level, Halstead Programming Time and Halstead Volume, LOC Metrics; Lines of Total Code, LOC Blank, Branch Count, LOC Comments, Number of Operands, Number of Unique Operands and Number of Unique Operators, and lastly Defect Metrics; Error Count, Error Density, Number of Defects (with severity and priority information).

Step 3: Defects data are extracted from software repositories and stored in to the database for pre-processing

```
1.#defect Analyser
2.Start
3.prev = NULL
4.while (iteration < 25)
5.{
6. if (iteration!=1)
7.{
8.  cur = getcurrentchanges( );
9.sold = comparechanges(prev, cur)
10. if(sold >= accptablelimit)
11. {
12.  Buglist = findbugs(changed modules)
13. Updatewith severity(buglist)
14. }
15. }
16. prev = cur;
```

Step 4: Applying data mining techniques to identify the defects severity and classify

```
17. #defect severity identified ( )
18. Builds static module mapping
```

```
19. Hash < module name, module type>
20. Enum { module type} Core Module, Abstraction Layer, Application
    Layer
21. Get module type (module name)
Step 5: Defect Classification
22. get moduletype(module name)
23. {
24. return hashtable get(name)
25. }
```

4.4

Results

Table 2. Performance of Various Classifiers

Data Sets	Classifiers	TP Rate	FP Rate	Precision	Recall	F-Measure
	J48	0.817	0.717	0.801	0.817	0.808
CM1	Naive Bayes	0.792	0.66	0.804	0.792	0.798
	MLP	0.847	0.794	0.797	0.847	0.816
	J48	0.764	0.628	0.725	0.764	0.736
JM1	Naive Bayes	0.782	0.651	0.739	0.782	0.742
	MLP	0.79	0.711	0.75	0.79	0.727
	J48	0.748	0.504	0.725	0.748	0.729
KC1	Naive Bayes	0.727	0.538	0.699	0.727	0.706
	MLP	0.758	0.582	0.733	0.758	0.714
	J48	0.794	0.562	0.776	0.794	0.783
KC3	Naive Bayes	0.789	0.52	0.782	0.789	0.785
	MLP	0.768	0.611	0.75	0.768	0.758
	J48	0.974	0.892	0.963	0.974	0.967
MC1	Naive Bayes	0.889	0.661	0.962	0.889	0.922
	MLP	0.974	0.913	0.962	0.974	0.967
	J48	0.712	0.374	0.705	0.712	0.707
MC2	Naive Bayes	0.72	0.432	0.712	0.72	0.696
	MLP	0.68	0.423	0.669	0.68	0.672
	J48	0.87	0.76	0.838	0.87	0.85
MW1	Naive Bayes	0.814	0.414	0.874	0.814	0.837
	MLP	0.866	0.668	0.852	0.866	0.858
	J48	0.901	0.722	0.88	0.90	0.88
PC1	Naive Bayes	0.881	0.59	0.886	0.881	0.883
	MLP	0.921	0.616	0.907	0.921	0.911
	J48	0.972	0.979	0.957	0.972	0.965
PC2	Naive Bayes	0.907	0.858	0.959	0.907	0.932
	MLP	0.976	0.979	0.957	0.976	0.967
	J48	0.85	0.655	0.831	0.85	0.839
PC3	Naive Bayes	0.357	0.162	0.859	0.357	0.409
	MLP	0.863	0.762	0.821	0.863	0.834

5 Conclusion and Future Work

As the requirement becomes complex IT industries needs new techniques and approaches to reduce the development time and cost, also to satisfy the customer's needs. As a result, Data mining techniques are widely used to extract useful knowledge from large software repositories to adopt it in bug detection to improve software quality. This paper, adapted software defects detection and classification methods by integrating data mining techniques to identify, classify the defects from large software repository. Work

discussed here uses Core, abstraction and application layer based on defects severity in the software product for classification. The designed method discussed here is evaluated using the parameters precision, recall and Fmeasure. MLP achieves significantly better results compare to other two classifies i.e. J48 and Naïve Bayes. As a future work, different machine learning algorithms may be included in the experiments on different data sets.

References

1. Ajay Prakash, B.V., Ashoka, D.V., Aradhya, V.N.: Application of data mining techniques for software reuse process. Procedia Technology 4, 384–389 (2012)
2. Kim, S., Whitehead Jr., E.J., Zhang, Y.: Classifying Software Changes: Clean or Buggy? IEEE Transactions on Software Engineering 34(2), 181–196 (2008)
3. Antoniol, G., Ayari, K., Penta, M.D., Khomh, F., Guéhéneuc, Y.G.: Is It a Bug or an Enhancement? A Text-Based Approach to Classify Change Requests. In: Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research, New York, pp. 304–318 (2008)
4. Fluri, B., Giger, E., Gall, H.C.: Discovering Patterns of Change Types. In: Proceedings of the 23rd International Conference on Automated Software Engineering (ASE), L'Aquila, September 15-19, pp. 463–466 (2008)
5. Jalbert, N., Weimer, W.: Automated Duplicate Detection for Bug Tracking Systems. In: IEEE International Conference on Dependable Systems & Networks, Anchorage, June 24- 27, pp. 52–61 (2008)
6. Cotroneo, D., Orlando, S., Russo, S.: Failure Classification and Analysis of the Java Virtual Machine. In: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems, Lisboa, July 4-7, pp. 1–10 (2006)
7. Kalinowski, M., Mendes, E., Card, D.N., Travassos, G.H.: Applying DPPI: A Defect Causal Analysis Approach Using Bayesian Networks. In: Ali Babar, M., Vierimaa, M., Oivo, M. (eds.) PROFES 2010. LNCS, vol. 6156, pp. 92–106. Springer, Heidelberg (2010)
8. Čubranić, D.: Automatic bug triage using text categorization. In: SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering (2004)
9. Guo, P.J., Zimmermann, T., Nagappan, N., Murphy, B.: Characterizing and Predicting which Bugs Get Fixed: An Empirical Study of Microsoft Windows. In: ACM International Conference on Software Engineering, Cape Town, May 1-8, pp. 495–504 (2010)
10. Fenton, N.E., Neil, M.: A critique of software defect prediction models. IEEE Transactions on Software Engineering 25, 675–689 (1999)
11. Yorozu, Y., Hirano, M., Oka, K., Tagawa, Y.: Electron spectroscopy studies on magnetooptical media and plastic substrate interfaces (Translation Journals style). IEEE Transl. J. Magn. Jpn. 2, 740–741 (1982)
12. Lutz, R.R., Mikulski, C.: Requirements discovery during the testing of safety-critical software. In: Presented at the Proceedings of the 25th International Conference on Software Engineering, Portland, Oregon, pp. 578–583 (2003)
13. Honar, E., Jahromi, M.: A Framework for Call Graph Construction, Student thesis at School of Computer Science. Physics and Mathematics (2010)
14. Boetticher, G., Menzies, T., Ostrand, T.: PROMISE repository of empirical software engineering data, West Virginia University, Department of Computer Science (2007), http://promisedata.org/repository
15. Bugzilla, http://www.bugzilla.org
16. Perforce, http://www.perforce.com
17. Trac Integrated SCM & Project Management, http://trac.edgewall.org
18. Fossil, http://www.fossil-scm.org