Christian-Albrechts-Universität zu Kiel
Institut für Informatik und Praktische Mathematik
Prof. Dr. W.-P. de Roever
Marcel Kyas, Gunnar Schaefer Martin Steffen

P-I-T-M

WS 2004/05

Handout 3

9. November 2005

Handout3: Bugzilla

Ausgabetermin: 9. November 2005

The purpose of this handout is to explain the purpose and how to use bugzilla. Bugzilla is a web-based bug-tracking system. In the following, we first explain the purpose of bug-tracking, the anatomy and the life-cycle of a bug in bugzilla, how bugzilla is used, and finally the policies of using bugzilla in this project.

1 Bugtracking

The main functionality of bugzilla is maintaining a data base of bug reports and track the state of bugs. The main purpose of bugzilla is to help in *collaborating* in correcting bugs. It helps to keep track of the status of an error, for instance:

- who reported the error?
- what kind of error, short description?
- who is (probably) responsible?
- was it acknowledged?
- has it been repaired?

Bugzilla can generate statistics and reports on open bugs, who is responsible for a bug, and in what state it is. It can tell you how many bugs and which ones have to be corrected to reach a mile stone. But similar to *Coma* it does not take decisions and it does not correct bugs.

Bugtracking is a necessity in software development. It helps in classifying what has to be done when. It keeps people informed on what needs to be done. It simplifies the process of figuring out who is responsible for correcting a bug.

Bugzilla is a tool to make life in software development simpler.

2 Anatomy of a bug

Bugs are modeled by the following fields.

Product and Component Bugs are divided up by Product and Component, with a Product comprising many Components. In our case the product is "*Coma* conference manager". Currently, the components of coma are

Data Model This component represents the *data model* of *Coma*. The data model describes which data is represented within the tool and how it is related to each other.

Specification This component represents the *specification* document. Bugs filed against this component may include enhancements to the functionality, inconsistencies in the document, or suggestions on how to change the specification to meet the overall goal within the tight time constraints.

Further components will be added once the architecture of the tool has been settled. Each component has an initial owner and an initial quality assurance (QA) contact. The *initial owner* is the person who is automatically responsible for analysing and correcting this bug. The *initial QA contact* is responsible for verifying that the resolution of a bug is indeed a correct once. More details later.

Status and Resolution These define exactly what state the bug is in — from not even being confirmed as a bug, through being corrected and the correction being confirmed by QA.

Assigned To The person responsible for fixing the bug.

URL A URL associated with the bug, if any. This can be used to associate external documents to a bug without attaching it to them. It can, e.g., point to a section of the requirements or specification document which explains why this is a bug.

Summary A one-sentence summary of the problem. This summary should be meaningful. "It does not work" is not meaningful. "Uploading an article does not send a confirmation" is.

Status Whiteboard A free-form text area for adding short notes and tags to a bug.

Keywords This is currently not used.

Platform and OS These indicate the computing environment where the bug was found.

Version This field usually contains the particular version of the product in which the bug was found. In our case it is the same as milestones.

Priority The bug assignee uses this field to prioritise his or her bugs. It's a good idea not to change this on other people's bugs. In *Coma* priorities range from P1 (the highest value) to P5 (the least value). Bugzilla defaults to P2 for each new bug.

Severity This indicates how severe the problem is. Values are

blocker This error makes the application unusable.

- **critical** The error is a security error or an error which does not yet make the application unusable but it misbehaves in a way which violates the specification. E.g., if an author can see the discussion on his paper, this would be a critical bug.
- **major** The error does not pose any danger to the integrity to the application, but it seriously impedes usability. If uploading a paper silently fails, this would be a major bug.
- **normal** It is an error that needs to be corrected, but there is a work-around to achieve this functionality.
- **minor** It is an error which does not affect the main functionality of the application. Most errors in error-paths (exceptions handled in an unfortunate way) are of this kind.
- **trivial** These are cosmetic issues. Spelling errors or errors in comments are trivial.
- **enhancement** These are not bugs but enhancement requests. A valid enhancement request would be to ask another group to export (define an interface) some functionality, which otherwise has to be implemented on your own. Also the definition of new views can be an enhancement.
- **tracking** These are not bugs, but meta-bugs. The idea of a tracking bug is that it collects all bugs which need to be corrected for a certain feature, or even tracks the discussion on the implementation of a certain feature. The use of this severity is strictly optional.

Target A future version by which the bug is to be corrected. Values are:

- **M0**
- **M1**
- **M2**
- **M3**
- **Presentation** The final day of the lab course where the product is to be presented.

Reporter The person who filed the bug.

CC list A list of people who get mail when the bug changes.

Attachments You can attach files (e.g. test cases or patches) to bugs. If there are any attachments, they are listed in this section.

Dependencies If this bug cannot be corrected until other bugs are corrected (depends on), or this bug stops other bugs being fixed (blocks), their numbers are recorded here.

Votes Whether this bug has any votes. This is not used in our lab course.

Additional Comments If you have something worthwhile to say, you can add it to this section. In this lab course, each change of a bug requires the changer to add a *meaningful* comment on why he changed the bug.

The most important part on understanding a bug is its state and how a bug changes its state. It is graphically depicted in Figure **??**. A state may have a *resolution* as its substate.

Figure 1: A bug's life cycle

Open Every new bug starts its life in the open state.

Assigned If a bug has been analyzed and found to be a valid bug report, then the state is changed to assigned. This step entails confirming that the bug is filed against the right component, that it is reproducable, and naming the person responsible for correcting the bug.

Resolved Once a bug is corrected or the report is inappropriate, the state is changed to resolved. This state also allows one to set the resolution. This is a value from:

- **Fixed** The error has been corrected.
- **Invalid** The bug report is considered to be invalid. This is used if the reporter uses a function of another component which is not exported in the interface in a way which is wrong, expects functionality of an exported function that is not specified or which contradicts the specification, or if the report is not meaningful at all. A bug with the summary "The tool is crashing when it is raining" is definitely invalid, as is "x*x does not return negative values".
- **Wontfix** The responsible person will not correct the bug. You can use this resolution, if you think that the behavior described in the report is indeed a bug, but there is either a workaround, or the reporter used a functionality in an unintended way. Another appropriate situation is, when a client calls a function of your code that is not part of the exported interface of your component.
- **Remind** This resolution can be used for bugs with a low severity or a low priority. It is used for errors which can be shipped in a final product and essentially means: "I'll correct it at the end of the project, if I have time".
- **Worksforme** This resolution is used for bug reports which cannot be reproduced.

Verified Once an error is resolved, the QA responsible has to check whether the resolution is correct. If so, he changes the state of the bug to verified. Note that the resoltion set in resolved also is part of the verified state.

Closed If everybody is happy with the resolution, the state of a bug is changed to closed. This means that the bug is not present in the software anymore.

Reopened If a resolution is found to be invalid or a bug which has been resolved appears again, a bug can be reopened. For all practical purposes this state is the same as open.

3 Bugzilla

Bugzilla is some server which helps to maintain a common database of bugs during the software devolopment. The relevant information on using bugzilla is explained in Chapter 5 of bugzilla's user manual.

In order to use bugzilla, it is necessary to create a user account *with a working email address*.

4 Policies

Bugzilla can be a helpful tool when used with care, but obviously it does not guarantee solid development. It is necessary to understand that bugzilla is here to help developing and not replace developing software. Everybody has to understand that the purpose of bugzilla is not to report bugs and track bugs,[1] but to *arrive with a working product.* To achieve this end, we propose the following policies.

If you want to report a bug, first think about whether it is really a bug and in which components. Not all bugs are in the code. Some may be in the documentation or even the specification. The first step when writing a bug report is to decide what the nature of the bug is and in which component you suspect the error.

Then think about the text of the report. Give a description of the bug which allows anybody else to reproduce the bug. Write down what you did, what you expected as a response, and what you actually got. If such a description is missing or erroneous, the assignee is free to resolve the bug as invalid or worksforme. Bug reports with patches are even better.

Remember, writing a good bug report takes time. Writing reports with patches is even better. Writing bad ones will just undermine the purpose of bugzilla and propably kill the project.

Keep the bug data base up to date. More importantly, *collaborate* in keeping the bug data base up to date. If you see a bug report which appears to be solved, ask whether it has been resolved or resolve it yourself with worksforme and a comment that, when you tested the code, it worked. If you see duplicate reports, feel free to mark them as duplicates. If you know something helpful, comment on the bug.

Whenever you change the state of the bug, write a meaningful comment on why you changed it. It needn't be a long comment.

Nobody should change the priority or severity of a bug without very good reasons!

If a bug is considered trivial or minor, delay its resolution. To do this, set its state to resolved with resolution remind. This allows us to see that you do not feel that it is necessary to correct the bug right now.

Use bugzilla only for discussing details of bugs. Do not put everything you do into bugzilla. We have introduced tracking bugs into the data base, but this is only useful for certain dangerous changes or very complex tasks which need much communication.

Do not push for a resolution. Often it is useful to delay the resolution in order to have something useful for a milestone. Often, the right solution is to reassign a bug. But we should agree on these steps.

And finally: If you like to go bug hunting in other peoples code instead of working on your own, we assume that you have too much time and expect you to help that group correct these bugs (just kidding). On the other hand, a bug report may be assigned to the reporter in certain circumstances.

[1]Well, this actually is, what it does