Xaraya
Request for Comments: 0025
Category: Informational

M.R. van der Boom
C. Dudley
G.J. Rothfuss
Xaraya Development Group
October 2002

Bug tracker requirements and selection

Status of this Memo

This memo provides information for the Xaraya community. It does not specify an Xaraya standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Abstract

This RFC documents the requirements for the implementation of a bugtracking application used by the Xaraya group. Focus will be on ease of use and integration with our SCM. It is expected that implementation of the bugtracker will happen in stages.

The second part of this RFC describes the preselected solutions which are thought to qualify, and a comparison is made between suggested solutions.

Table of Contents

1. Introduction

This document defines the requirements of the envisioned bugtracker application. The purpose of the document is to record and maintain our current best idea of what we want from a bugtracker, so we can justify project decisions in terms of those requirements, review project decisions, and achieve quality by meeting those requirements.

The requirements were gathered by open discussion on our IRC channels, by theoretical analyses and by practical needs.

The requirements are, as far as we can make them, an objective measure of what we want from a bugtracker, and what would best contribute to the projects goals. They aren't a to-do list for the project. Meeting some of them may not be feasible, but we should still use them to measure the quality of our solutions.

2. How to read the requirements

We give the requirements a unique number so we can refer to them in our documents and in our decisions. In the online representation of this document the requirements will be given anchors so we can explicitly link to them.

The requirements are listed in a table with the following columns:

1. the requirement identifier
2. the measure for the requirement
3. critical,essential,optional and nice flags
4. remarks

For function requirements the measure is just a statement which is true or false. For attribute requirements it's typically a scale of some sort and a unit.

The terms "critical", "essential", "optional" and "nice" have the following meanings:

1. critical: The project would fail to reach its goals if the requirement were not met at this level.
2. essential: The project should meet the requirement at this level, but could still reach its goals without it.
3. optional: The project will try to meet this level, but product quality will not suffer directly if the requirement is not met at this level.
4. nice: Meeting the requirement will help the project, but in general it should only be met if it can be done cheaply (mostly in the time resource domain)

The entry in each cell is a level for the measure; either a minimum level (for measures like likeability where higher levels are better) or a maximum level (for measures like cost where lower levels are better). Then entries for an attribute requirement look like this:

```
Measure                          Critical Essential Optional  Nice
-------------------------------------------------------------------
Cost of the product, in dollars.  < 1000   < 500     < 100     < 50
-------------------------------------------------------------------
```

The above entries mean that it is critical that the cost of the product be less than 1000 dollars, essential that it stays below 500, optional that it be less than 100 dollars and really nice if it would drop below 50 dollars.

The entries for a functional requirement look like this:

```
Measure                          Critical Essential Optional  Nice
-------------------------------------------------------------------
Product is blue                  maybe    yes       yes       yes
-------------------------------------------------------------------
```

The above entries mean that it is essential that the product be blue (and therefore it is also optional and nice that it be blue) but it is not critical that it is blue (or that it is not blue).

3. Summary

For the people who don't want to go through the whole list of requirments this is the list of 5 most critical requirements:

1. Bugtracker state is consistent with state of product sources
2. Bugtracker makes the life of developers, users and managers easier (i.e. it helps them to make a quality product more easy and more quickly
3. Bugtracker is optimized to discover why the sources are the way they are and why they have changed
4. Bugtracker interfaces under our control are public, documented and maintained
5. Bugtracker allows to ask questions involving both bugtracker and SCM system

The most compelling functional requirement is that users must be able to carry out routine tasks using either the SCM systems interface, the native bugtracker interface or any other interface which has access to bugtracking information. That is, users must not be required to switch interfaces to carry out routine development tasks like registering new bugs or resolving issues. Meeting this requirement makes it much easier for developers to keep bug information up to date.

List other compelling requirements here

4. Requirements

ID	Measure	Critical	Essential	Optional	Nice
1	state is consistent with state of product sources	yes	yes	yes	yes
2	makes the life of developers, users and managers easier (helps them to make a quality product more easy and more quickly)	yes	yes	yes	yes
3	optimized to discover why the sources are the way they are and why they have changed.	yes	yes	yes	yes
4	interfaces under our control are public, documented and maintained.	yes	yes	yes	yes
5	Bugtracker allows to ask questions involving both bugtracker and SCM system.	yes	yes	yes	yes
6	cmdline interface	no	yes	yes	yes
7	web interface	yes	yes	yes	yes
8	mail interface	no	yes	yes	yes
9	gui client interface	no	no	yes	yes
10	Fields: id, state severity, type, description, comments(N), owner, submitter,	yes	yes	yes	yes
11	Fields: notify list, related items	no	no	yes	yes
12	quick filtering	yes	yes	yes	yes
13	quick queries	yes	yes	yes	yes
14	assign to components	no	yes	yes	yes
15	assign to developers	yes	yes	yes	yes
16	configurable notifications	no	yes	yes	yes
17	xaraya module	no	no	yes	yes
18	reference points in SCM on state changes of bugs	no	yes	yes	yes
19	ability to hold patches against a bug	no	yes	yes	yes
20	register duplicates	yes	yes	yes	yes
21	register dependencies	yes	yes	yes	yes
22	register meta bugs (grouping)	no	no	yes	yes
23	statistics (specify what?)	no	no	yes	yes
24	custom attributes	no	no	yes	yes
25	configurable submission template	no	no	yes	yes
26	mark custom attributes as required	no	no	yes	yes
27	ACL	no	no	yes	yes
28	as simple as possible, but not simpler	yes	yes	yes	yes
29	standards compliant	yes	yes	yes	yes
30	open source	no	no	yes	

```
yes
31  support multiple projects                          no      yes       yes
yes
32  configurable auth system                           no       no       yes
yes
33  flow control                                       no      yes       yes
yes
34  anyone can add comments to a bug                   no      yes       yes
yes
```

4.1 State is consistent with state of product sources

This requirement is critical. It means that the information in the bug tracker database is at all times consistent with the current state of the source tree, so you can trust both on what should be done to fix issues with the current tree. This means in practice that the version control system should enter information into the bugtracker database when sources are changed to which a bug is attached. This can be implemented for example by giving signals to users when changing files to which a bug is registered, giving them a chance to look into the bug database.

Without this requirement the information in the bug tracker database will not only loose quality, when the project grows it will be near to impossible to estimate the amount of work which is needed to reach a certain goal, without expert knowledge of the source tree. The use of the bugtracker and the source tree alone should be sufficient for any party competent enough to interpret the information to estimate the amount of work needed to reach a certain goal.

4.2 Makes the life of developers, users and managers easier

The bug tracker is a tool we want to use. If this tool doesn't make our life easier it's a bad tool. Three groups need to profit from this tool:

Users: need to be able to search and filter the bug database to discover what the state is of the source tree, so a decision for using a certain feature can be made.

Developers: need to be able to discover by working the source tree or querying the bug database what area needs work and what things are going wrong. The interface to the source tree and to the bug database need to be highly integrated and well connected. On discovery of an error the developer should be able to register the discovery directly from the source tree, after which the bug enters a known state in the bug database.

Managers: need to be able to analyze the bug database and from that information make a translation to the work needed on the source tree. This will allow them to make educated estimates for planning, resources and other project information needed to manage the project.

4.3 Optimized to discover why the sources are the way they are and why they have changed.

There are a lot of attributes which can be attached to the unique identification of a bug. Initially those attributes will be chosed which help optimize the bug tracker tool to be used to answer the question why sources are the way they are and why they have changed.

4.4 Interfaces under our control are public,documented and maintained.

The bugtracker should have a publically accessible API and if the interfaces need to be expanded then the source needs to be neatly laid out and well commented to allow easy expansion.

4.5 Allows to ask questions involving both bugtracker and SCM system.

A question of one system should give the same result on the other. This ensures consistency between the two systems.

4.6 Interfaces

4.6.1 Cmdline interface

Usually most of the version control system interaction occurs on the command-line. It makes sense to have an interface to the bugtracker database which can be used from the command-line as well. Although not a critical requirement this will highly influence the usage of the bug-tracker by the developers themselves. It allows them to update the bug information while working with the VCS.

4.6.2 Web interface

This requirement is listed as critical for a number of reasons. Most people are used to using a bugtracker with a web interface, so this interface has the lowest threshold for people to use it.

Also, the flexibility of a web interface probably satisfies the requirements of dealing with searching and filtering while keeping an accessible system for most users. Although the command-line interface may be more powerful and offer more options, for most users command-line usage is often arcane and counterproductive. Also, a web-interface only needs a reasonable web-browser to use where as other interfaces might have more requirements (such as an installation of the version control system)

4.6.3 Mail interface

The mail interface allows developers to reply to bug reports via mail or trigger state changes in the bugtracker via mail. It should also allow for quick, easy and common tasks to be carried out - like returning a bug as "fixed", "not a problem", etc..

4.6.4 GUI client interface

This would entail a standalone custom client. This is unlikely to be implemented and wouldn't be all that useful if it was, since it would entail users and developers downloading the custom client software.

4.7 Fields: id, state severity, type, description, comments(N),owner, submitter

These are the basic attributes of a bug and are all essential to effective bug management and categorisation.

4.8 Fields: notify list, related items

This allows interested parties to subscribe themselves to a bug so that they automatically receive notifications about any updates to it. The related items field would indicate any other bugs which may have an impact on the current one.

4.9 Quick filtering

It should be possible to quickly filter bugs by various criteria, such as state, submission date, severity.

4.10 Quick queries

It should be possible to query the bugtracker for a list of bugs by specificing various search terms. This information should then help the Project Managers organise the project and allocate resources as needed. The

bugs database will rapidly start to hold an huge amount of information and so without good filtering and queries that information is not as useful as it could be and could become rather unwieldly.

4.11 Assign to components

It should be possible to assign bugs to specific metadata, for instance components that indicate a subsystem. Each component should have a default owner who gets assigned the bug when the bug is assigned to that component (this ensures the right people are automatically notified about new bugs).

4.12 Assign to developers

Bugs should be assignable to developers, and the assignment needs to be changeable. This allows people other then the default component owner to take ownership and work on bugs.

4.13 Configurable notifications

The bugtracker should include a notification system, which at least can notify one email-address on bug information changes. Possible notifications are:

1. on state change
2. notify list for users who wish to keep track on one specific bug
3. notification when a bug hasn't been touched for a while

4.14 Xaraya module

The bugtracker should integrate with Xaraya by being a module, or being wrapped into a module. Sharing authentication with Xaraya would also be a welcome feature.

4.15 Reference points in SCM on state changes of bugs

In addition to the first critical requirement that the state of the sources in the tree should be synchronized with the information int the bugtracker, this requirement formulates the most important consequence for this. On each state change of a bug, a reference point in the source tree should be linked in, created or otherwise made avialable to the bug-tracker.

The fix-process of a bug will be highly simplified by this. It allows developers to check the sources at the specific reference points, to gather information on how to fix the bug. By rolling back the source tree to a certain reference point the described behaviour in the bug can be reproduced so a check is possible which changes have occurred after that reference point which might affect the bug involved.

4.16 Ability to hold patches against a bug

Ultimately a bug needs to be resolved. Ideally this would happen by applying a neatly packaged patch to the source tree. The bugtracker should be able to hold a patch against a bug. This patch should be configured in such a way that it can be applied to the source tree without repackaging, so integration with our patch system is trivial. As an alternative to holding a patch against a bug directly into the bugtracker, is to store a reference to a patch in the patch system. If both are supported, the patch can either be applied directly from the bugtracker, thus interfacing with the patch system, or the patch can be applied from the patch system which notifies the bugtracker of success or failure.

4.17 Register dependencies

Bugs may have dependencies. The following dependencies are considered:

1. bugs block other bugs and vice versa
2. bugs are duplicates
3. bugs are meta-bugs

4.17.1 Blocking relationship

Bugs should be able to block other bugs from being resolved.

4.17.2 Duplicate relationship

Due to the nature of a complex hierarcy of folders and files in the sources it often happens that bugs get registered twice. This happens often when at first it seems that two problems are unrelated and it is discovered later that two registered bugs actually are two different descriptions of the same problem.

The bugtracker should allow modifying the information of a registered bug in such a way that either:

1. the identical bugs are merged into one and the information from both bugs is merged into one bug
2. the bugs will be registered as duplicates and information changes on the first are automatically reflected in the other

The first situation is preferred

4.17.3 Meta relationship (a.k.a grouping)

It should be possible to assign bugs to groups to indicate a relationsship between them. These assignments should be orthogonal to the rest of the system.

4.18 Statistics (specify what?)

It should be possible to generate statistics from the bugtracker. Rather than specifying the precise requirements (which are unknown), it makes sense to have a query interface where the bug data is exposed and can be queried to generate statistics on the fly. These statistics could be presented in a tabular or graphical form (or preferably both). They should also be easily printable for reference.

4.19 Custom attributes

The bugtracker should allow to add custom attributes to a bug to support special needs. Examples are target releases, keywords etc.

4.20 Configurable submission template

The bug submission form should be configurable to target it better to the different bug-entering audiences, not all of which are technically adept and may need some help from the form.

4.21 Mark custom attributes as required

It should be possible to specify which attributes have to be filled out when entering a bug.

4.22 ACL

The operations of the bugtracker should be controlled by ACL. It should also be possible to specify a regular expression so that new users can be automatically added to the appropriate access groups.

4.23 As simple as possible, but not simpler

The bugtracker should not cut corners to make it appear easier. For instance, using a forum as a bugtracker because it is "convenient" would be wrong. However, it should be simple, intuitive and easy to use. If it isn't easy to use then people will be less inclined to submit bugs, thereby impacting code quality.

4.24 Standards compliant

The bugtracker should use standards where possible. For instance: XHTML or HTML 4.01 and CSS for display, ANSI SQL for database schemas. The use of these standards should be documented.

4.25 Open source

As Xaraya itself is an open source project, we strongly prefer open source solutions. Open Source software generates a lot of advantages for us. Being able to modify a package to our needs being one of the most important ones.

4.26 Support multiple projects

The bugtracker should support multiple projects. This means it should support bugs from different source trees, and handle them seperately.

4.27 Configurable auth system

This is only an optional requirement as it is not necessary to be able to use the same ID for both the tracker and the version control system. For instance ssh public/private keys could be used to authenticate instead. If every user has the tracker id's public key in their authorized_keys file then the tracker will be able to make any updates needed to the VCS *as* the user specified. Commands running the other way should be much easier as tracker updates are just entries in a database and so do not need to run as any particular user. It would be nice to be able to use the same userid and password for both, but since not everyone who has a tracker id will have a VCS id I'm not sure how well it would work in practice. Those who want to use the same ID can simply set the passwords to be the same or maybe a cron script could do it for them but its a bit hacky.

4.28 Flow control

The ability to assign a bug from one owner to another does imply some form of flow control. However it should also be possible to specify that all bugs need to be passed by a Quality Assurance Team before being certified as closed and fixed.

5. Revision history

2002-10-31: First revision

2002-11-11: More details added

2002-11-13: Filled out the remaining sections