METAQUEST
SOFTWARE

A Guide To Evaluating a Bug Tracking System

White Paper

By Stephen Blair, MetaQuest Software
Published: October, 2004

Abstract

Evaluating a bug tracking system requires that you understand how specific features, such as configurable workflow and customizable fields, relate to your requirements and your current bug tracking process. This article provides tips and guidelines for evaluating features, and explains how these features fit into a defect tracking process.

Stephen Blair is a programmer-writer at MetaQuest Software, developers of the Census bug tracking system as well as solutions for e-support, application self-healing, and desktop management.

Web site: *www.metaquest.com*
Telephone: 514 341-9113
Fax: 514 341-4757
E-mail: mailto:info@metaquest.com

Contents

Introduction

Before you start evaluating bug tracking systems, make sure you identify your requirements for the system. Understanding these requirements will help you build a list of features that you can use to guide your evaluations.

To identify your bug tracking requirements, take a look at your current bug tracking process.

- What are the different roles and responsibilities of the people who will use the system?
- What is your workflow for managing and resolving bugs? Identify the steps in the process, and determine who is responsible for each step.
- What information do you need to track for each bug?
- What reports and metrics do you need?
- Do you need to provide different levels of access to different users?

Once you identify your requirements for the system, you can translate the requirements into a "feature list". Table 1 provides an example of a feature list that could be used to evaluate a bug tracking solution

Table 1. List of features to evaluate

Adaptability
Change history
Version control integration
Customizable fields
Ease of use
E-mail notifications
Reports
Security
Web-based client
Workflow

The rest of this document provides tips and guidelines for evaluating these features. Hopefully these guidelines will help you choose a bug tracking system that meets your requirements. Remember, the purpose of a bug tracking system is to support your process, not to impose its own.

Adaptability

If you want to track more than just bugs, make sure the bug tracking system can be adapted to track other types of issues (such as support calls, test cases, or purchase orders). A system that is designed specifically for bug tracking can be hard to adapt, so look for a system that provides pre-built templates for tracking different issue types. Also, as you evaluate the various features of the tracking system, look for any hardwired "bug-tracking" terminology or functionality. Fields, queries, reports, notifications, workflow: everything should be adaptable. Otherwise you'll end up trying to fit a round peg into a square hole (for example, trying to fit trouble tickets into bug reports).

For example, if a tool provides a default bug-tracking workflow that you can't replace completely, you won't be able to implement workflows for other issue types.

Evaluation Checklist:

- ❑ Pre-built templates for different issue types (support calls, test cases, …)
- ❑ No hardwired bug-tracking terminology or functionality

Bug Change Histories

Consider whether you want to track changes to bugs. Some tools maintain change histories (audit trails), which allow you to trace who did what to an issue and when (for example, who raised the priority of a bug).

If the tool provides a change history, look at what information it records. For example, if you change the priority, does the change history simply say "Defect modified" (not so good) or "Priority modified" (better), or does it say "Priority changed to High" (best)? Some tools offer a sort of compromise, where the tool adds "Defect modified" to the revision history and allows the user to add a comment such as "Changed priority to High after speaking to customer".

A related feature is something called *timestamping*. In tools that support timestamping, information entered into text fields such as Description, Resolution, and Workaround are time and date stamped with the User ID of the individual who entered the text. This time stamping provides a history of the changes to important text fields.

Evaluation Checklist:

- ❑ Changes to issues are automatically logged to revision history
- ❑ Change history indicates exactly what was changed (for example, Priority changed from Medium to High)
- ❑ Change history can be enabled/disabled on a field-by-field basis
- ❑ Changes to text fields be timestamped

Customizable Fields

You need to be able to customize the fields used to collect defect information. Otherwise, you'll end up collecting only the information the tool thinks you should track, and using the tool's terminology instead of your own.

Fields

Most bug tracking tools come with a predefined set of fields. You probably can use some of these fields as is, and others by simply changing field labels and drop-down list values to match your terminology. For example, if you already classify defects by severity and priority, and the tool includes predefined Severity and Priority fields, you just need to change the severity and priority levels to match those you already use. If you cannot reuse a predefined field, you should be able to delete it, or a least hide it so that users never see it.

To collect all the information you want to track, you may also need to add new fields. The tool should be flexible enough to allow you to put new fields anywhere you want. If you cannot rearrange fields, or if all new fields are put at the bottom of a page or on a separate tab, the usability of even the simplest "bug form" will suffer.

Field relationships

Consider also whether you want to define field relationships. Field relationships can help you keep drop-down lists short and uncluttered. For example, by creating a relationship between the Product and Version fields, you can display only the version numbers of the selected product (instead of displaying all version numbers of all products). Similarly, for software bugs, you can display only the software components and filter out documentation and hardware components.

Some tools also allow you to select a value from a drop-down list based on the values selected from other lists. For example, you could automatically assign bugs based on the selected product and component.

Custom views and field sets

Based on the user's role and the task at hand, do you want to display different sets of fields? The ability to display custom views allows you to reduce UI clutter and present only the fields necessary for the task at hand. For example, a custom view for logging new bugs wouldn't need to include the fields used by QA and Engineering to add information during the resolution process.

Evaluation Checklist:

- ❑ Change field labels and drop-down lists to match your company's terminology
- ❑ Custom fields—add and remove fields as required
- ❑ Control over which fields are required and which are optional
- ❑ Customizable field layout
- ❑ Add, remove, and rename tabs
- ❑ Define field relationships
- ❑ Create custom views

Notifications

Notifications are really a workflow feature, because they help you track and manage bugs. For example, notifications can keep team members informed about important changes to bugs, such as changes in priority or addition of new information (such as how to reproduce the problem). Notifications can also help automate the defect management process. For example, you can notify development managers when new bugs are submitted, and developers when bugs are assigned to them.

When evaluating a bug tracking tool, look closely at what kind of notifications you can send. Most bug tracking tools support a fixed, standard set of notifications:

- Bug added
- Bug edited
- Bug status changed
- Bug assigned

Obviously there's a potential for a spam-like deluge of notifications, especially if you enable "Bug edited" notifications. So look for the ability to define notifications for specific changes to specific fields. For example, can you send a notification when someone changes the priority of a bug? Or when your most important customer reports a "must-fix" bug?

If you want to allow customers to submit bugs directly into the system, check if the tool allows you send notifications to the customer (e.g., send the customer a confirmation, notify customer when bug is fixed).

Finally, look at what information goes into a notification message, and how easy (or hard) it is to customize the contents. Some tools offer a predefined list of possible content, such as summary, full details, change history, or all of the above.

Evaluation Checklist:

- ❑ Automatic e-mail notifications for comment events (new bugs, changes to bugs, status changes, bug assignments)
- ❑ Support for custom events (send notifications for specific changes to specific fields by specific users, for example, notify the developer when a tester modifies the Repro Steps fields)
- ❑ Send notifications to users or customers (where customers don't have a user account for the tool)
- ❑ Easily customizable notification message contents

Ease-of-Use

Is the interface consistent and easy to navigate? Web-based applications can often be difficult to navigate, especially if they behave more like a sequence of HTML pages than an application. Some Web applications are constantly reloading pages because responding to user input requires a round trip between the Web server and the user's browser. This constant reloading of pages can make an application difficult to work with, because the application doesn't behave the way users expect an application to behave—like a Windows application.

When a Web application behaves like a Windows application, it's easier to adopt into your development process. There's less learning curve, and less resistance to change.

Another important consideration is the user interface text; is it user-friendly, simple, and clear?

Finally, and maybe most important, is the interface easy to use? Try evaluating how easy is to perform common tasks, such as logging a new bug or searching for keywords in common text fields. Is it obvious how to do these tasks? And how many mouse clicks does it take? Frequently-used features should never be more than a few mouse clicks away.

Evaluation Checklist:

- ❑ Interface is easy to navigate
- ❑ Easy to enter new bugs
- ❑ Easy access common features (requires only a few mouse clicks)
- ❑ User interface text and messages are easy to understand

Security

Security is typically based on user accounts, user groups, and user group permissions. Look for a bug tracking system that provides different levels of permissions:

- Field-level permissions control whether users can edit, or even view, fields.
- Bug-level permissions control access to bugs by users.
- Feature-level permissions control access to specific features, such as reporting or user account administration.
- Project-level permissions determine which users can access which projects/databases.

There are two approaches to providing field-level security. One approach is to specify on a field-by-field basis which user groups have permission to edit the field. Other tools, such as Census from MetaQuest Software, take a different approach, and allow you to define custom Web views of the bug database, where each custom view includes different fields. For each custom view, you can then specify which user groups that have permission to access the view.

For example, a customer view would include only the fields required to submit a new bug, while a developer view would include all fields. Customers would not be allowed to access the developer view, while developers would be allowed to access either view.

Bug-level security is usually accomplished by limiting the available queries.

Evaluation Checklist:

- ❑ User groups and user group permissions
- ❑ Field-level security
- ❑ Bug-level security
- ❑ Feature-level security
- ❑ Project-level security
- ❑ View-level security (define multiple views of a project and then grant access permissions)

Reports and Metrics

A bug tracking system should allow you to quickly gather the information you need for staff meetings (bug listings and printouts), as well as provide more detailed metrics to help you make decisions.

Look for tools that provide both distribution metrics (metrics that break bugs down by category or classification: for example, issue age by severity) and trend metrics (metrics that show changes over time: for example, defect arrival rate).

You may also require the ability to build your own custom reports, preferably using an industry-standard reporting tool such as Crystal Reports.

Evaluation Checklist:

- ❑ Bug listing reports, summary and detailed
- ❑ Distribution reports (cross-tab or chart)
- ❑ Trend (time-based) reports
- ❑ Web-based reporting
- ❑ Library of predefined reports
- ❑ Export reports and metrics
- ❑ Support for industry-standard reporting tools such as Crystal Reports

Workflow

A bug tracking tool should automate and enforce your process for managing and resolving bugs. The tool should provide a configurable workflow that allows you to define the steps in your process and the order of the steps.

Workflow is typically modeled as a series of states, such as New, Fixed, and To be Verified. To support your process, you'll need to be able to add and remove workflow states, as well as define the allowable transitions between states. For example, between Fixed and Closed you may want to add a required Verify Fix state, to ensure that an issue is never closed until after QA verifies the fix.

You should also be able to control which users are allowed to move bugs between states. For example, you may want only members of the QA group to be able to move bugs from Verify Fix to Closed. Granting workflow permissions allows you to enforce accountability and responsibility throughout the process.

While workflow is based primarily on states, you may also want your workflow to depend on other bug attributes, such as product or defect type, or even on user group membership. For example, you may need different workflows for software, hardware, and documentation issues.

Finally, you should also expect a bug tracking tool to support different workflows for different projects.

Evaluation Checklist:

- ❑ Default workflow
- ❑ Configurable workflow: define custom states and transitions
- ❑ Enforceable workflow: specify possible transitions and who has permission to make the changes
- ❑ Separate workflows for different projects, different issue types

Version Control Integration

The starting point for any worthwhile version control support is the ability to link bugs to source code. Linking bugs to source code means developers can document a bug by linking to the related source files (which is especially important if the bug is deferred). These links will also allow you to track the work done on a bug.

Next, verify the version control operations that can you perform from inside the bug tracking system. You should be able to perform common operations such as checking files in and out, viewing file histories, and getting different versions of a file. The ability to check files in and out is key, particularly for Web-based bug tracking systems. Not all Web-based systems support the check in and check out operation, which limits the usefulness of the version control integration. Web-based check outs/ins help to improve development workflow and save time.

If a bug tracking system has both a Windows client and a Web client, make sure you can perform all version control operations through both interfaces.

Evaluation Checklist:

- ❑ Link bugs to source code
- ❑ Remote, web-based access to Visual SourceSafe databases
- ❑ Perform basic version control operations through the bug tracking system
- ❑ Web-based check outs and check ins

Web-based Client

In addition to evaluating the usability (see Ease-of-Use on page 6) of a Web-based client, you should also evaluate the implementation of the client.

Architecture

Web-based clients can be implemented in a number of ways:

- **Internet Information Server (IIS) applications.** An IIS application is a Visual Basic application that lives on a Web server and responds to requests from the browser. An IIS application uses a combination of ASP and HTML to present its user interface and uses compiled Visual Basic code to process requests and respond to events in the browser.
  - IIS applications offer good performance. They also offer easy customization and branding, because the user interface is driven by HTML templates stored outside of the DLLs.
- **Internet Server API (ISAPI) applications.** ISAPI was developed by Microsoft to address the resource problems of CGI. ISAPI uses Dynamic Link Libraries (DLLs) that are customized to return specific HTML. The IIS Web server loads the DLL the first time a request is received and then keeps the DLL in memory to service other requests. This minimizes the overhead associated with executing such applications many times.
  - ISAPI applications offer good performance. However, they are often difficult to customize or brand, because all the HTML that makes up the user interface is emitted by a DLL.
- **Common Gateway Interface (CGI) applications.** CGI is a standard way for a Web server to pass a Web user's request to an application and then forward the results back to the user.
  - One of the problems of CGI is that every request from the browser starts a new process on the Web server. Consequently CGI applications are resource hungry, and can impact the performance of your Web server.
- **Active Server Pages (ASP) applications.** ASP is a server-side scripting environment for IIS that combines HTML pages, script commands, and COM components.
  - ASP performance can be hindered by the fact that ASP is driven by script languages that must be interpreted at run time.

Application standards

Does the Web-client behave like an application, or like a series of pages? Many Web applications are page-based, like a Web site, and are basically a series of HTML page generated on the server. Every interaction with the user requires a round-trip to the server and a page reload. Something as simple as running a predefined query, or switching to another tab, can require a page reload.

For example, suppose you log on to your bug tracking system to review the defects submitted yesterday. One looks familiar so you decide to do a keyword search for related or duplicate

issues. In a page-based tool, this loads a new page, replacing the list of new defects you were looking at. After you type the text you want to search for and click Ok, another page is loaded, this time listing the defects that matched your query. And so it goes, back and forth between different pages as you search, scan lists, and read details.

Contrast this with the behavior of a Web application whose interface follows familiar Windows standards. In this type of Web client, the keyword search, the list of issues, and the issue view/edit form are all part of a unified interface. You don't load another page to define the query, running the query updates the list in place, and the defect you're viewing doesn't disappear just because you ran a query.

Customer communications

If you want to allow customers to submit bugs or to monitor issue status through a Web-based interface, check that the tool includes a Web-based client that customers can use. The customer submit interface should allow an unlimited number of customers to submit bugs.

Evaluation Checklist:

- ❑ Web-based client implemented using best technology
- ❑ Web-based client follows familiar desktop UI standards
- ❑ Web-based client for customers