Volume 5, Issue 6
June 2002—The Software Quality Advisor

The Software Quality Advisor Online

How to Start a Bug Collection

Randall W. Rice, CSQA, CSTE

How to Start a Bug Collection

This article looks at how to start studying the defect trends in your organization for the purpose of process improvement.

- Defects are elusive, but can be understood.
- Studying defect trends is not expensive.
- Defect analysis requires a process, tools, and a person or team to own the process.

When teaching people how to improve their testing and development processes, I often advise them to "start a bug collection." This reminds me of when I was in grade school and I would spend hours looking for insects around our home for science assignments. It seemed that the common bugs were so easy to find they were unimpressive – both to my friends and my teachers. However, when I did manage to find those elusive unusual specimens, usually big with strange shapes and colors, that was cause for celebration.

In those days, I wasn't trying to improve a process. I was just trying to get a good grade. In testing, I still celebrate the discovery of the unusual and elusive bugs but for a different reason. Each new type of bug or defect I find as a tester indicates a new area of process improvement. If I view testing as a net that catches defects, each new type of defect indicates where the net needs to be strengthened or mended.

The Nature of Defects

Defects are serious. They can cause everything from inconvenience (MS Word crashing in the middle of an article) to death, as in lethal doses of radiation. I prefer to use the word "defect" as opposed to "bug" because "bug" may convey a more causal meaning than is appropriate. Plus, I try not to propagate loose terminology (like using the term "QA" when "QC" is the actual function performed). In this article, I use the term "bug" to keep with the idea of a "bug collection," but I also use the term defect when discussing the analysis aspect of the bug collection.

Defects can take a wide variety of forms, and there is an established taxonomy of defects (Table 1—Page 8). Not every organization will experience every type of defect, but it helps to know about

them as potential risks.

Like insects, defects can cluster. Where you find one type of defect, keep looking – there may likely be other similar defects nearby in the application under test. This happens because people tend to work in patterns that can propagate defects.

Boris Beizer has written about "the pesticide effect" of defects [1].

(Continued on Page 2)

Inside this issue:

Book Review—Lessons Learned in Software Testing by Cem Kaner, James Bach and Bret Pettichord

This book is written by three credible authors with real-world testing experience. I found myself agreeing wholeheartedly with many of the lessons presented, disagreeing strongly with others, and confused on others.

Overall, I think that testers will find some good ideas in the book, but I would advise caution in going from reading the ideas to taking them directly into practice. One example is lesson 146, "Don't use the IEEE Standard 829" (By the way, lesson 145 was "Use the IEEE Standard 829 for Test Documentation"). After stating the useful aspects of the standard, the authors go on to downplay the standard because it can result in large volumes of test documentation. The authors dismiss quickly how to use the standard responsibly and keep the scope in control but go into great detail about how people have gone overboard in documenting tests. After urging the reader to ask "How compelling are those benefits [of using the standard] under your circumstances?", the concluding statement of this lesson is "In many contexts, the added benefits are not compelling. In such cases, given the added costs and risks associated with developing large test documentation sets, we submit that creating Standard 829-style documentation would be irresponsible." My concern is that people may take this comment to

★★★☆☆

heart too quickly and suffer losses due to a lack of test documentation. For example, try telling a government regulator that the test standard just didn't work for you.

(Continued on Page 2)

Book Review (Cont'd.) - Lessons Learned in Software Testing by Cem Kaner, James Bach and Bret Pettichord

In addition, such an extreme position should be backed up with hard research, which is not done in this case.

The authors are clear at the outset that the lessons will be controversial, and some of them are. The book is biased toward the exploratory testing view, which some people will really like and others will find less than thrilling. I expect people will be talking about the book, which is a good thing. We need to debate some of these topics as a profession.

I found parts of the book to be somewhat arrogant in tone, which took away from my impression of the book. The authors seem to have an ax to grind which comes through at many points in the book. I was also put-off by the following comment on page 205: "Hiring, training, pay, and promotion preference that result in groups that are more dominated by white males than they need to be are particularly counter-productive in testing." I was puzzled that the reviewers and editors let a racial/sexist comment into the book. The authors' point could have been made well without pinpointing a particular race or sex.

While many of the lessons were presented with adequate detail, other lessons are very brief and beg for more information. For example, lesson 172 is "Be prepared for the build." This lesson is three sentences long and concludes "In a fast-paced project, a test group without a well-managed test environment is useless." Agreed. Now, what are some ways to manage the environment? Who should manage it? I would have liked to have seen some of these shorter lessons either dropped or more well-developed.

Summary

My overall review of the book is to read it with caution, apply what makes sense in your environment, and discuss the ideas among your peers. Keep in mind that there is a lot of opinion in the book. Like with many things you read, pilot the ideas that make sense before you try to implement them fully. I can't give this book a glowing endorsement because there are so many situations I can think of that would be ill-served by some of the lessons in the book. However, to not discard the good with everything else, I think that everyone will be able to take some good things away from the book.

How to Start a Bug Collection (Continued from Page 1)

Just like insects, a certain pesticide may eradicate many insects in a population, but not all of them. Some insects will be resistant to the pesticide. The resistant bugs will produce other resistant bugs and before long the pesticide that used to work fine no longer is effective. Similarly, some tests are very effective until people start to perform work in new ways and their defects evade detection. Then the challenge is to develop new tests.

Defects are Good

In a past article, I explored the concept of defects as learning experiences and a cause for celebration as long as the lessons are learned and our customers and users do not experience avoidable problems (see "Defects are Good" at http://www.riceconsulting.com/defects_are_good.htm).

Defects, however, do cost money as evidenced by the resulting rework. The sooner you find them the better.

Dr. Deming had it right. For each defect in a product, there is a cause. And…someone was paid to inject the defect into the product, even if they had good intentions. The quality practitioner's challenge is to understand the cause of the defect and find ways to prevent future occurrences.

When defects are seen only as problems, people can get defensive and become more concerned with avoiding guilt than taking positive action. Unfortunately, too many organizations use defects as an occasion to punish people instead of opportunities for improvement.

What's Needed

To build a good "bug collection" you need three key components: people to own the process and to provide accurate information, a process that is not overwhelming but provides an effective framework to capture defect information, and tools to help capture and report defect information.

This framework requires an environment of management support and understanding that people can trust not to use the defect information against them. If defect information is ever used as a performance evaluation, people will quit providing accurate data. The objective of the bug collection is to understand the origin of defects in an organization, not by person, but by project activities.

With this objective and environment in mind, here are some considerations for the key elements of the framework:

The People

There needs to be an owner of the process. For this role, I often recommend a person in the position of defect administrator. The defect administrator is not a clerical position, but rather a role that requires attention to detail, people skills and organization skills. The defect administrator should know at any point in time the status of defects on a project.

Others will have input and receive information from the defect management process:

(Continued on Page 3)

"When defects are seen only as problems, people can get defensive and become more concerned with avoiding guilt than taking positive action. "

Look for Results, Not Time

© 2002 Johanna Rothman

Who's working hard in your organization? One senior manager, Cyril, noted the cars in the parking lot on the weekend, as his measure of who was truly committed to the project. Cyril also noticed when people arrived at work and when they left. Cyril thought that measuring his staff's office-time would help him get more out of his staff and release the product earlier.

In reality, his office-time measurement was only a check on his hiring techniques, to see if he'd hired people who thought that being at work was important to succeed at work. Cyril wanted to make sure he'd hired people with "fire in their bellies", and that they wanted the company to succeed as much as he wanted it to succeed. Office-time isn't a measure of performance, and certainly doesn't guarantee that your staff are as interested in the company's success as you are.

If office-time is your only measure of performance, then people think that they're successful if they show up and stay. And, if you measure office-time as a surrogate for productivity, your staff will put in their office-time, but may not necessarily get any productive work done.

One of my colleagues told me this story:

"We were late on a project for a Very Important Customer. Senior management decreed we would have dinner on the company every night at 7pm. As a Director, I organized the dinners. Do you know what it's like to plan dinner for 60 people every night? A wedding would have been easier.

"It worked for about a week, maybe two, and then people started coming in later in the morning, and leaving right after dinner every night. I suggested we stop having dinners at work, and just send people home to relax and sleep. Senior management was shocked —we were making progress, weren't we? I don't think it took any less time to finish the project. In fact, it took longer because we were all so tired."

(Continued on Page 5)

"If office-time is your only measure of performance, then people think that they're successful if they show up and stay."

How to Start a Bug Collection (Continued from Page 2)

Testers will generate defect reports, verify defects have been resolved, and provide input as to the possible origins of the defects.

Developers will resolve defects and provide detailed insight as to the origins of the defects.

Help desk personnel will generate defect reports, especially those that are reported by customers and users after a release.

QA analysts will facilitate the defect management process and the process improvement efforts.

Management will receive the defect information, make informed decisions (hopefully) and drive process improvement efforts.

It is vitally important that the people in the above roles buy in to the defect management process. A good way to achieve this is to involve them in designing the process. It is also important to realize that defect management is not a one person or single team effort. Unless people from multiple areas of the organization are involved, improvement efforts will look like a few people telling other people how to "do it right."

The Process

Ah, the process! In a world where we struggle with defining and following processes, if we fail to have a defined and repeatable process for gathering and reporting defect data, our numbers and resulting conclusions will be incorrect. The degree of accuracy deviation may be major or minor, depending on how well the process is defined and followed. Of course, you could have a great process and follow it, feeding it faulty data and still have flawed conclusions. However, let's start with defining a good process and we'll deal with the source data issues next.

This discussion of process will be the vehicle I will use to explain how to obtain, process, display and understand the defect trends (our "bug collection").

Our process will have the following components:

Input – This is the defect information gathered from the people identified above. For the task at hand we need to know:

- Defect type/category (such as requirement error, design error, coding error, etc.)
- Defect origin (such as project activity – requirements definition, construction, testing, etc.)
- Defect reason (why the defect occurred)
- Defect severity (cosmetic, minor, workaround, critical)
- Defect owner (who is responsible to fix?)
- When identified (date and time)
- Who identified (tester, user, etc.)
- Cost of the defect impact
- Cost to fix the defect

To gather this information, we need a tool that people can easily access and use. Specific tools and tool strategies will be discussed a little later in this article.

Procedure – These are the steps to perform in gathering and processing defect information:

1. Report defect using specified tool or method
2. Categorize defect
3. Determine defect severity
4. Determine defect origin
5. Identify/assign defect owner
6. Resolve defect
7. Determine defect costs
8. Re-test defect
9. Determine defect resolution/status

(Continued on Page 7)

Links

A gold mine of all kinds of interesting and useful project and testing information:

http://www.geocities.com/mtarrani/artifacts.html

Defect tracking tools:

http://www.incose.org/tools/tooltax/defecttrack_tools.html

Defect Prevention Techniques for High Quality and Reduced Cycle Time:

http://www.iscn.at/select_newspaper/measurement/motorola2.html

The Security of Applications – Not all Applications are Created Equal:

http://www.atstake.com/research/reports/atstake_app_unequal.pdf

Using Defect Analysis to Initiate the Improvement Process:

http://www.iscn.at/select_newspaper/measurement/bruelkjaer.html

Software QA Slide Show:

http://www2.umassd.edu/CISW3/coursepages/pages/cis480/lectures/sqa.ppt

Preventing Requirements Defects:

http://wwwbruegge.in.tum.de/teaching/ws01/RE/presentations/koch.pdf

Current Status in QA:

http://www.swt.tuwien.ac.at/actions/download/files/qs/Presentation%20QA%20Research.pdf

Paper on the Basics of Software Testing:

http://www-2.cs.cmu.edu/~koopman/des_s99/sw_testing/#taxonomy

Quotes

“Fools rush in where fools have been before.”

Unknown

“No matter how cynical you get, it is impossible to keep up.”

Lily Tomlin

“Failure is only the opportunity to begin again more intelligently.”

Henry Ford

“A good plan, violently executed now, is better than a perfect plan next week. “

Gen. George S. Patton

“I do not know anyone who has gotten to the top without hard work. That is the recipe. It will not always get you to the top, but it will get you pretty near.”

Margaret Thatcher

"Hold yourself responsible for a higher standard than anybody expects of you. Never excuse yourself."

Henry Ward Beecher

"Pick battles big enough to matter, small enough to win."

Jonathan Kozol

“He who works his land will have abundant food,
but he who chases fantasies lacks judgment.”

Proverbs 12:11, The Bible

Questions From the e-Mail Bag

Q: I came through your site while I was surfing for faqs in Testing. I would appreciate if you could answer me the following questions which would be helpful for me,

1) What are the characteristics of a Test Plan?

A: A test plan is just a project plan for testing. Test plans can be written for various phases of testing, such as unit, integration, system and user acceptance testing. There is an IEEE standard for test plans (standard 829) that covers just about all aspects of a test. Many people take the standard and reduce it down to fit their own needs. I consider a test plan as a tool for communication, not a large document that will be hard to maintain. I will typically keep the size of the test plan to under 20 pages, except for very large or complex projects.

2) What considerations are taken to prepare the Test Plan?

A: I typically start at the highest level and define the major objective of what the test is to validate or verify. I usually relate each test objective to a project objective. Then, I identify features (functions and attributes) to be tested (or not to be tested) - this is the scope of testing. Next, I identify the test team, the schedule and the resources needed (test environ-

(Continued on Page 5)

Questions from the Mail Bag (Continued from Page 4)

ment, etc.). I pull all of this information together, have the test team review it, management approve it, then distribute the plan. After it has been published or posted, I treat the test plan like any other project document. If it changes, then people are notified of the change.

3) What factors do you consider when creating Automated Scripts?

First, I look for those functions that are highly repeatable and boring to perform. I also make my test cases and scripts very modular so they can be combined together in many different combinations. Finally, I keep specific data out of the scripts and feed it to a basic script I place into a loop.

4) What is the difference between Static and Dynamic Testing and what are the advantages and disadvantages of each?

Static testing is like a review or walkthrough where the item being inspected does not change. You can find more defects in static tests because in dynamic tests defects may be masked behind other defects. People who get really good in reducing defects are good at performing static testing.

Dynamic testing is testing performed while executing the software. It is called dynamic because the code execution is constantly changing. Dynamic testing is needed for making sure the software works according to the specs and requirements and according to user needs.

5) Which Software Development cycle is most challenging to validate and why?

In my opinion, the very first activity of defining the user needs (prerequirements) is the most difficult to validate. The validation occurs in User Acceptance Testing, which I believe should be based on user scenarios, not requirements or any other paper document. The paper document can be wrong. The challenge is to model the user's world correctly and to get an adequate coverage in testing.

If you have a question for Randy, e-mail him at rrice@riceconsulting.com.

"People who get really good in reducing defects are good at performing static testing."

Look for Results, Not Time (Continued from Page 3)

I've noticed that people who spend a lot of time at work tend to spend time on non-work things, because they're not home to arrange the rest of their lives. If you're currently measuring office-time because you want to complete a project quickly, then tell your staff that you want the project done quickly. Explain how to make other tradeoffs on the project, to meet the schedule deadline.

Think about why you are measuring office-time and what you really are trying to learn. Here are questions to help you assess the project, other than the amount of time people spend at work:

- Do you have a project schedule, with milestones? Do the people have clearly identified tasks, broken down into inch-pebble tasks (one to two-day tasks)? Are people meeting their estimated dates? Is something preventing people from getting their work done?
- What is the Fault Feedback Ratio, FFR, the ratio of bad fixes to good fixes? The FFR may be higher than people realize because it's not always obvious that newly found errors were caused by bad fixes. The higher the number, the more people are spinning their wheels, trying to fix something that just won't stay fixed. I've noticed that the longer people work and the more tired they are, the higher the FFR. (An acceptable FFR number depends on the product you're building. For commercial products, I become concerned when I see a FFR of more than 15%, because it's too hard for the project team to find, fix, and verify the defects. For mission critical or safety products, an FFR of even 5% may be barely acceptable.)
- Does the project team know what they should be working on? Do the requirements stay relatively static, or are even the core requirements changing every week? Do all the stakeholders agree about why you are building this project, who will buy the product, and who will use the product? Sometimes, instead of slogging through the work, it's worth taking more time to revisit and rewrite the requirements, and then working with confidence that you are addressing the right problem.
- How are project decisions made, and how often are those decisions revisited? Can the project team make decisions as a team, or does the project manager make all the decisions, which the project team then ignores, so that the team does what they want anyway? Are decisions revisited frequently? If the product or project decisions are continually revised, then the team can't make progress on the project. Look at project meeting minutes, and see when the decisions change.
- Are team members other than the project manager looking ahead at project risks, and deciding what to do about them? Does the project team have a way to assess risk, and to develop plans to deal with those risks?
- How well are people working together? Are they working as a team, or are they each working alone, isolating themselves from each other? In my experience, when teamwork breaks down on projects, the team members are trying to protect themselves from something painful or stupid, such as or miscommunications, unclear project roles and responsibilities, or many other causes. As a manager, you can help uncover this problem and help people fix it.
- Does the project team know what you think success means for this project, including releasing the product quickly? Sometimes we don't clearly identify and communicate the objectives that are most important to us. If you don't tell the project team you want the project completed by a certain date, they won't know.

(Continued on Page 8)

NIST Study: Buggy software costs users, vendors nearly $60B annually

According to a June 25th story in Computerworld that quotes a 309 page study conducted by the National Institute of Standards and Technology (NIST), software defects are costing the U.S. economy an estimated $59.5 **billion** each year, with more than half of the cost borne by end users and the remainder by developers and vendors.

“Improvements in testing could reduce this cost by about a third, or $22.5 billion, but it won't eliminate all software errors,” the study said. Of the total $59.5 billion cost, users incurred 64% of the cost and developers 36%.

There are very few markets where "buyers are willing to accept products that they know are going to malfunction," said Gregory Tassey, the NIST senior economist who headed the study. "But software is at the extreme end, in terms of errors or bugs that are in the typical product when it is sold."

The report took 18 months of research that included extensive feedback from end users. The study also examined the impact of buggy software in several major industries, including automotive, aerospace and financial services. The results were then extrapolated for the U.S. economy.

The study was conducted for NIST by the nonprofit Research Triangle Institute in Research Triangle Park, N.C.

"No one thinks you can get all the errors out of software," said Tassey. Vendors are under pressure to get products to market quickly, and there are diminishing returns on testing: The more effort you put into it, the fewer the bugs that are found. "However, the general consensus seems to be [that] the current state of the art with respect to testing is poor and can be sufficiently improved," he said.

According to the Computerworld article, “The study didn't propose specific actions for improving testing but called for the development of testing standards, noting that today's testing tools "are still fairly primitive." “

The article also quoted the report as stating that “standardized testing tools, scripts, reference data and metrics, among other things, "that have undergone a rigorous certification process would have a large impact on the inadequacies" now found. “

You can read more about this report at www.computerworld.com.

You can download a copy of the 309 page report in PDF format at www.nist.gov/director/prog-ofc/report02-3.pdf

Note: In the next issue of the Software Quality Advisor, Randy will comment on the report.

“...the general consensus seems to be [that] the current state of the art with respect to testing is poor and can be sufficiently improved"

Rothman Consulting Group, Inc.

1. Management Practicum Workshop . . . . . *Johanna Rothman and Esther Derby*

Do you have real management problems? Want to explore real and possible solutions? Want to refresh your management skills? *Management Practicum* is an experiential workshop to help you solve your management problems. *Management Practicum* works with real situations and challenges that managers in software organizations face. Through facilitated discussion, presentations, and simulations, instructors and session attendees will create learning experiences to address the management issues that participants bring to the workshop. We present this publicly and in-house.

2. In-House Workshops . . . . . .

Software Project Management: Software projects are challenging. Although books can help, you’ll need to develop your own pragmatic approach. In this workshop, we address what software project managers really do: how to make tradeoffs between schedule, cost, defects, people, and the work environment; how to plan and monitor a project; and how to know when a project is complete.

Advanced Topics in Project Management: Now that you have some experience managing projects, what are your top problems? We’ll address attendee concerns for your organization, ranging from forming and leading teams to keeping sponsors engaged, from decision-making processes to how to build slack into the schedule. This workshop is customized for your organization.

Talking To Techies: If you’re a non-technical manager in a technical world, talking to your technical staff can be an exercise in frustration for both sides. Explore what makes technical people different, how to recognize failing interactions, and how to improve your communication skills with the talented, challenging people who work with you.

Johanna Rothman
phone: 781/641-4046
Rothman Consulting Group, Inc.
e-mail: jr@jrothman.com
www.jrothman.com
fax: 781/641-2764

How to Start a Bug Collection (From Page 3)

It is important to understand that defects have a lifecycle that may be repeated multiple times before the defect is resolved (Figure 1).

Output – These are the deliverables seen as a result of performing the process:

- Defect logs
- Defect profiles
- Current defect status list
- Charts and graphs of defect information

How to Report Defect Information

I use the following guidelines when reporting defect information:

- Be as objective and fair as possible
- Be as accurate as possible
- Don't report defect information by person, only by project or other aggregate view
- Use easy to understand graphics
- Keep the information in context by providing past data if available
- Be able to answer questions about the data

QC – This is to verify defect information is gathered and reported correctly:

- Checklists to prompt for completeness of defect information and to verify the process has been followed as designed
- Verification of source defect data to ensure the data is correct and reasonable

Standards

The defect reporting process can be supported by standards that list the defect information to be reported, how the composite defect information will be reported and how to categorize and prioritize defects. The standard should contain a list of defect types such as seen in Table 1 on Page 8.

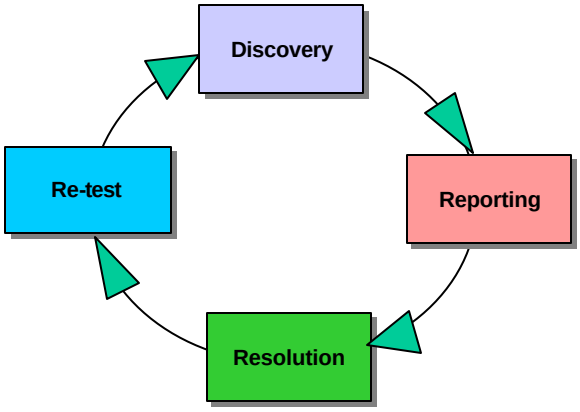

Figure 1—The Defect Life Cycle

Tools

The most obvious and applicable tools for this job are commercially available defect tracking tools. There are close to one hundred tool brands with varying levels of functionality. There are also web-based defect tracking services. Another option is to build your own tool, such as the one shown in Figure 2. However, homegrown tools may not be as robust as those you can buy. The important thing is to be able to get and report composite defect information from the tool.
(Continued on Page 9)

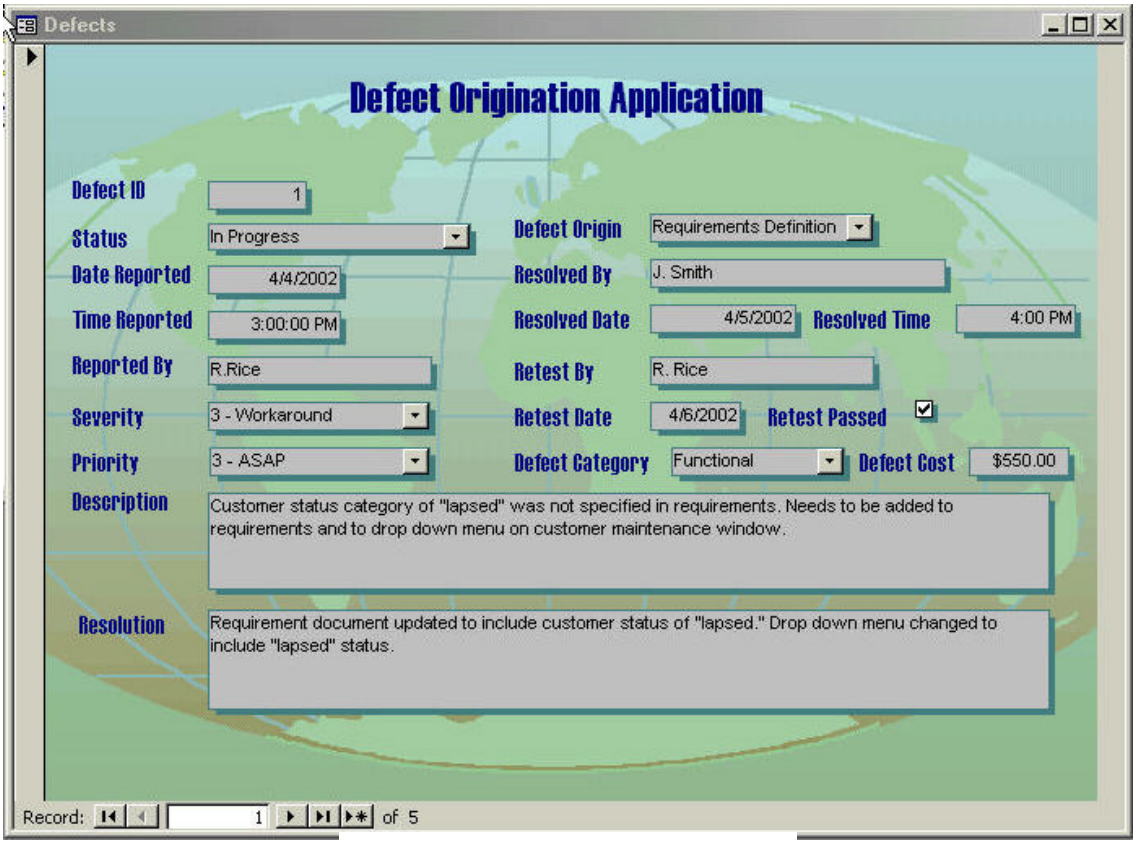

Figure 2—Homegrown Defect Tracking Tool

Table 1 - Defect Taxonomy

To understand your defects, you can view them from at least three perspectives:

1. When the defect w as injected:

- **Local:** The defect occurs in your internal development. In this case you can indicate the project phase, for example: **ds** for design or **cd** for coding.
- **Regional:** The defect is in a related software project, but not the one in the current time/defect log. In this case you can indicate the injection phase **pr** (for `project') or **op** (for `other project' of my own organization).
- **External:** The defect is in software out of your scope of control or repair. In this case you can call the injection phase **ex** (for `external').

2. Defect types:

- **IC:** Interface Capability.
  The design of an interface is wrong, so that the interface does not provide the functionality that it must provide.
- **IS:** Interface Specification.
  The specification of an interface is wrong, so that the parameters involved cannot transfer all of the information required for providing the intended functionality. This is a less fundamental variant of IC: Only parameters need be added.
- **ID:** Interface Description.
  The non-formal part of the description of an interface is incomplete, wrong, or misleading. This is typically diagnosed after an IU.
  Note that the description of a variable or class attribute or data structure invariant is also an (internal) interface.
- **II:** Interface Implementation.
  Something that I cannot influence does not work as it should.
  This defect should never be used when I am the source of the defect. (In principle, this is a special case of ID.)
- **IU:** Interface Use.
  An interface was used wrongly, i.e., in such a way as to violate the interface specification.
- **IV:** Data Invariant.
  A special case of IU. The interface violation is: not maintaining the invariant of some variable or data structure. Violating the meaning of a simple variable is a special case of this.
- **MD:** Missing Design (of required functionality).
  A certain requirement is covered nowhere in the design.
  This is stronger than IC, w here the coverage is present, but incomplete.
- **MI:** Missing Implementation (of planned functionality).
  A certain part of a design was not implemented.
  If the part is small, MC, MA or WA may be more appropriate.
- **ME:** Missed Error Handling.
  An error case was not handled in the program (or not handled properly).
- **MA:** Missing Assignment.
  A single variable was not initialized or updated. Only one statement needs to be added.
- **MC:** Missing Call.
  A single method call is missing. Only one statement needs to be added.
- **WA:** Wrong Algorithm.
  The entire logic in a method is wrong and cannot provide the desired functionality. More than one statement needs to be added or changed.
- **WE:** Wrong Expression.
  An expression (in an assignment or method call) computes the wrong value. Only one expression needs to be changed.
- **WC:** Wrong Condition.
  Special case of WE. A boolean expression was wrong. Only one expression needs to be changed.
- **WN:** Wrong Name.
  Special case of WE. Objects or their names were confused. The wrong method, attribute, or variable was used. Only one name needs to be changed.
- **WT:** Wrong Type.
  Two `similar' types were confused.

3. Defect Reason - *why the defect was introduced*:

- **om:** Omission.
  I forgot something that I knew I had to do.
- **ig:** Ignorance.
  I forgot something, because I did not know I had to do it.
- **cm:** Commission.
  I did something wrong, although I knew in principle how to do it right.
- **ty:** Typo.
  I did something trivial wrong, although I knew exactly how to do it right.
- **kn:** Knowledge.
  I did something wrong, because I lacked the general knowledge (i.e., the education) how to do it right.
- **in:** Information.
  I did something wrong, because I lacked the specific knowledge how to do it right or had received misleading information about how to do it. This refers to a problem with communication.
- **ex:** External.
  I did nothing wrong. The problem was somewhere else and the defect was introduced by some other person.

Source: Lutz Prechelt, prechelt@ira.uka.de, http://www.geocities.com/mtarrani/defecttypes.doc

Look for Results, Not Time

(Continued from Page 3)

- Do you own this project? Do you say things such as “my “project instead of “our” project? If is your project, then your staff will never be committed; their work will be just a job. It has to become their/our project.

You can say s omething like this to your staff: “We won't reach these goals if any of us are working at less than our peak effectiveness. Do you have ways of knowing when your effectiveness is slipping? (You can prime the pump by suggesting things like FFR.) If you can tell me what to look for, then I'll be able to notice and let up on the pressure. And here's what you can notice about me, so you can let me know when I'm slipping.”

Remember that your staff’s effort goes to what is measured. Office-time is a dysfunctional measurement. When people are evaluated based on a single measurement, they will find ways to optimize that measurement that do not necessarily contribute to the desired outcome. If you measure and reward office-time, you’ll only get office-time, not a finished product. Instead, help the team view a picture of the project and the evolving product. You’ll know if people are committed to your project and are fired up about it.

Johanna Rothman is a consultant, trainer and author based out of Arlington, MA You can contact her at jr@jrothman.com.

The Software Quality Advisor Online

June, 2002

P.O. Box 891284
Oklahoma City, OK 73189

405-793-7449
405-793-7454 Fax
rrice@riceconsulting.com

"Test everything. Hold onto the good."
I Thessalonians 5:21

We're on the Web!
www.riceconsulting.com

July 2002 Issue:

Standards are Your Friend
by Randall W. Rice

Book Review—Communication Gaps and How to Close Them

Coming to Chicago!

August 14—16, 2002

A Three-day course in User-Oriented Approaches for Delivering Quality Software

Presented by Process Management Group, Ltd. (www.pmgltd.com), the Midwest's Premier Provider of IT Quality and Software Testing Services, and Rice Consulting Services, LLC (www.riceconsulting.com), a world recognized leader in Quality and Testing Training.

See details and register at
www.riceconsulting.com/chicagoq3_2002.htm

Calendar of Events

User-oriented Practices for Delivering Quality Software

Chicago, IL, August 14—16

Sponsored by the Process Management Group, Ltd. And Rice Consulting Solutions, LLC

www.riceconsulting.com

How to Become an Effective Test Team Leader, Madison, WI, October 3 — 4, 2002

Sponsored by the Wisconsin QA Chapter.

Register at www.riceconsulting.com/madison2002.htm

A Three-day Course in Web Testing

Chicago, IL, October 30—November 1, 2002

Sponsored by the Process Management Group, Ltd. And Rice Consulting Solutions, LLC

EUROStar Conference

November 11—15, 2002

Edinburgh , Scotland

Randy will be presenting a one-day tutorial on Surviving the Top Ten Challenges of Software Testing and a keynote address on Getting the Most Value from Every Person on Your Test Team.

We hope to see you at one of these events!

If you have a group of 12 or more people in your city that would like to sponsor a training event, contact Randy Rice at rrice@riceconsulting.com to find out how to book a special presentation.

Bug Collection (Continued from Page 7)

You can find an extensive list of defect tracking tools and services at: http://www.incose.org/tools/tooltax/defecttrack_tools.html

Summary

It's easy to spend tons of money on testing, especially if tools are a part of the effort. If you want to get the greatest value for the money, spend your time and money to learn about your defects. Defects are costing your organization money in rework, whether people realize it or not. Since the cost is already being realized, it only makes sense to take the extra step to learn from them.

1. Boris Beizer, Software Testing Techniques. Second edition. 1990