

Tracking Severity

Assessing and classifying the impact of issues (a.k.a. defects).

By Tim Dyes

Mar 1, 1999

Summary: Guidelines for assessing and classifying the impact of issues. How does one categorize Severity? Should you use numbers like 1, 2, 3; generic names like High, Medium, Low; or more specific names? A telephone switching system, for example, might use industry-specific categories such as “system issue,” “line issue,” or “call issue.” Other environments, as we'll see in this article, tailor classification terms to meet their own functional needs.

So, you want some advice on how to classify your bugs. Because needs and priorities vary from project to project, I won't try to tell you what's best for you. Instead, let me relate what has worked well for me with a few supporting examples, as well as what has been reported as working well elsewhere, and how these classification systems meet or don't meet key recommendations coming from standards bodies. Armed with that knowledge, you will be able to craft a classification system that works for you.

First of all, I favor the term “issue” over “bug” or “defect.” I regard an “issue” as anything that warrants investigation and that may result in a change to the product. Issues may include not only defects, but new feature requests, spec changes, new requirements, observations, suggestions for improvements—anything that may result in a change to the product. Using one mechanism to track these different flavors of issues eases the management task, i.e., what are the issues, how important are they, when do we address them, and what's their current status.

Why classify issues at all? Why not simply address them in “first in, first out” order? The answer is because one issue may be more important than another. Important to whom? Important to the customer, usually. This impact on the end user determines your ability to retain and expand your customer base, as well as being able to justify the exorbitant price of your product.

Clearly, issues should be classified so that they can be investigated and resolved in some rational order. I refer to that classification as “Priority,” which targets an issue for resolution in a specific

product release. Assessing issue Priority is a release management activity that can only be determined after other issue attributes have been assessed. Issue Priority should take into account the issue's impact on the customer when it occurs, how likely it is to occur, the effort required to resolve and validate it, and the risk to and importance of the current release schedule. An initial Priority assignment is usually made to target the issue for resolution in a specific one of the next two or three releases. As the number of issues grows, Priorities may be juggled around to accommodate release dates. And as each release gets out the door, the open issues are reviewed with Priorities juggled again to accommodate customer needs and to fit into the next release schedule. Hence the well-known bow wave effect as minor issues are deferred from release to release. With that said, the focus of this article is not on issue Priority, but rather on assessing issue impact on the customer when the issue occurs. This impact is referred to as "Severity."

Severity

Categorizing levels of Severity is important for several reasons. First, it helps determine release-targeting priority. It may also be useful in communicating the number of open issues in each of the various Severity categories. Be careful, however, about adding unnecessary complexity to your issue tracking system—its primary purpose is to speed the identification, assessment, and disposition of issues. Anything that burdens or complicates that process is best avoided.

Okay, let's get down to business. How does one categorize Severity? Should you use numbers like 1, 2, 3; generic names like High, Medium, Low; or more specific names? A telephone switching system, for example, might use industry-specific categories such as "system issue," "line issue," or "call issue." Other environments, as we'll see in this article, tailor classification terms to meet their own functional needs.

And should you break Severity into both Breadth and Depth components? Depth would capture how severely the issue impacts one user, while Breadth would capture how many users are affected by one occurrence of the issue. In the telephone switching system example above, Breadth categories would be "affects entire system," "affects a single line," or "affects a single call." Depth categories would be "loss of service," "usage problem," or "service degradation." You could likely come up with any number of meaningful Severity categories. The trick is to come up with the fewest number that give you the most value.

Guidelines

The following four guidelines on Severity categories serve to:

- Make Severity assessment easier to do.
- Get the Severity right most of the time for whoever records the issue.
- Make improper assessment easier to spot.
- Reduce debate regarding whether an issue has been assessed appropriately.

Guideline 1: Use the minimal number of categories to satisfy your release targeting and communication needs. At least three categories are generally required—corresponding to issues that must be fixed, issues that should be fixed, and issues that can probably be deferred. Why not

just use these three generic categories? See Guideline 2 for the answer.

Guideline 2: Favor meaningful names over numbers or generic names. It's far easier to assess something as a "Reliability" issue instead of "High" or "Category 2." Meaningful names provide other benefits as well. They reduce debate when trying to assign Priority to an issue, and they communicate the state of the product more effectively. For example, in the field of medicine everyone from a part-time software tester to the president of the company can relate to the meaning and importance of a "Safety" issue. Keep in mind that while using meaningful names speeds issue assessment, reduces debate, and improves communication, that doesn't mean you can get by without explicit definitions (see Guideline 3).

Guideline 3: Write down and communicate clear, unambiguous guidance as to what each Severity category means. If an issue might fall into more than one category, give guidance as to how to decide which one to use. Examples are always useful.

Guideline 4: Don't be afraid to add to or simplify your classification as the need arises. Adding a Severity category can improve the visibility and attention an issue will receive. Conversely, when you find a category does not add much value, consider merging it into an adjacent one.

From Academic to Practical

The IEEE standard 1044 contains an in-depth exploration of software anomaly classification. Severity is one of 1044's required attributes, and 1044's Severity classification provides a good starting point for our discussion (see [Table 1](#)).

This is a reasonable breakdown, with the linear scaling of Severity easing the prioritization task. The generic naming, however, makes the assessment more difficult and error-prone. For maximum clarity of meaning in my health care test projects, I have remapped the IEEE classification as follows (and shown in [Table 2](#)).

1. Urgent is split into Safety and Reliability with appropriate definitions. Safety issues are "must fix" for next release, and can potentially result in a field recall. System crash and data loss are Reliability issues that might be deferred depending on how often they occur and just what the impact on the user is.
2. High is moved into Reliability. To me, the breakdown or absence of a critical function with no workaround is always a Reliability issue; if a single category were too broad for your needs, then perhaps classifications such as Crash, Data Loss, and others would be more appropriate. For me such a breakdown just complicates the assessment (remember Guideline 1).
3. Medium and Low are combined into Nuisance. Again, this is a judgment call, but as long as a workaround exists, the user is not barred from the functionality. Dealing with the workaround is a nuisance, and perhaps a serious one, but it's not a reliability problem.
4. None is split into Spec Change and Observation. Spec Change implies a decision to provide a new specification or function, or to modify existing behavior based upon customer needs. The number of Spec Changes reveals the stability (or, usually, the

instability) of the requirements. This may be indicative of changing customer needs in a new or dynamic market, or it may indicate a poor understanding of what those needs really are. Either way, a large number of Spec Changes indicates product flux with a corresponding development and validation effort. An Observation, on the other hand, catches anything left worth noting. Observations include suggestions for improvement not severe enough to make it into a higher category. An Observation is very useful for capturing changes that developers or QA would like to see made—changes to improve the clarity, extensibility, or testability of the code. Observations can capture items that need to be addressed but that don't fall into other categories, and may come from testing, code reviews, or customer comments. Capturing these observations keeps them on the table and helps with the planning of subsequent releases.

(*) Note that Table 2 includes “Discomfort,” which we rank as higher Severity than “Reliability.” On the IEEE scale, however, this would be categorized as Low or Medium since it can be worked around. In our environment, we made a judgement that a Discomfort issue should be highly visible and would be more important than a Reliability issue occurring at the same frequency.

[Table 3](#) gives another example used by a leading compiler/tools vendor. Note that it more closely matches the IEEE classification.

James Bach made the observation in the **swtest-discuss** mailing list that few D and E issues may indicate a culture that doesn't have a high quality standard, or testers who are otherwise too tired or demoralized to report minor issues. An abundance of D's, on the other hand, may be a sign of either especially high quality, or especially superficial testing.

Cem Kaner in *Testing Computer Software* recommends using just three categories: Fatal, Serious, and Minor. To quote Cem, the purpose of tracking issues is to get things “that should be fixed, fixed. Anything that detracts from this prime objective should be excluded.” This is good advice. Stay focused on why you want Severity classifications at all, and then let that focus guide your decisions on what classifications you really need.

Related Attributes: Frequency and Breadth

You may have seen Severity used in a way that correlates more directly with the Priority. For example, a nuisance that happens very frequently might be deemed more severe than a crash that happens once in blue moon. Such a Severity classification that embeds Frequency is harder to assess and leaves much room for debate. I favor and use a separate Frequency attribute to capture that information. Again, a minimal and well-defined classification of Frequency is warranted. In practice, while I find the Frequency attribute actually recorded only fifty percent of the time, it definitely helps with the Priority assessment. More importantly, keeping the Frequency out of the Severity assessment makes the assessment much easier to do.

Another attribute that might be broken out from Severity is its Breadth. This Breadth is applicable for products that support multiple users or uses. For example, an issue that affects all users should probably get more Priority than if it affected just one user. A Severity classification that includes

Breadth may wind up obscuring Severity's linear scaling. However if Breadth is applicable, it will be discussed when assigning Priority anyway. If you think it useful, consider adding Breadth as a separate attribute.

Assessing What Works for You

As usual, no one solution will work best for everyone. Use your experience and judgement to decide what works best for your situation, and don't be afraid to change things as time goes by. Once a release goes out the door, it's a good time to evaluate the effectiveness of your practices and make adjustments in the lull between the storms. Remember the four guidelines:

1. Keep it minimal, i.e., simple.
2. Use meaningful names.
3. Provide explicit definitions with examples.
4. Don't be afraid to change things as your needs change.

TABLE 1
IEEE 1044 Severity
Classification

CLASSIFICATION	DEFINITION
Urgent	System crash, unrecoverable data loss, jeopardizes personnel
High	Impairment of critical system functions and no workaround exists
Medium	Impairment of critical system functions and workaround exists
Low	Inconvenience or annoyance
None	None of the above or an enhancement

TABLE 2
Severity Classification
Example from the Medical
Domain

CLASSIFICATION	DEFINITION
Safety	Serious injury or death to patient or operator
Discomfort	Discomfort to the patient*
Reliability	Failure to complete a primary function as expected—in our case failure to complete a blood component collection procedure, or failure to collect within expected volume, yield, and contamination limits
Nuisance	May annoy the user but does not result in a reliability problem—product is able to perform its primary function as expected
Spec Change	New feature that isn't addressing a higher Severity issue—if it is, then the higher Severity rating should be chosen
Observation	Everything else

TABLE 3
Severity Classification
Example from
Compiler/Tools Domain

CLASSIFICATION	DEFINITION
A	Crash or data loss
B	Problem with no workaround
C	Problem with workaround—with workaround being defined as something readily apparent to a user with basic knowledge of the product, or easily explained in the manual or over the phone
D	Minor problem—would otherwise be a B or C issue but in this case doesn't materially impair the user
E	Enhancement request

About the Author *Tim Dyes is a senior developer and lead software QA engineer at Cobe BCT, Inc. Tim likes to write code, solve problems, and play with his kids.*

Originally published in [STQE magazine](#)