

# Standardized Defect Statuses

David L. Brown, Ph.D., CSQA, CSQE  
DLB Software Quality Consulting  
Glastonbury, Connecticut, USA

## Abstract

Many companies struggle with defect statuses, the stages that a defect goes through on its path to final resolution. Ambiguous and overlapping categories cause arguments, inefficiency, inaccuracy, and failures to handle defects properly. Based on measurement principles and best practices observed while consulting, the author has derived a solution to this problem. This paper provides a complete set of easy to understand, non-overlapping statuses and a full explanation of the reasoning behind them. It covers the valid reasons for withdrawing a defect and provides a state transition diagram of normal status changes. Don't keep reinventing this wheel. Adopt these standardized defect statuses and spend your time more productively fixing defects and finding more of them.

## Introduction

Tracking defects requires a good set of statuses to tell us exactly where each defect stands in terms of being fixed. This sounds easy enough, but as a consultant I have encountered many project teams struggling with conflicting and overlapping sets of statuses that made it anything but clear what was happening. I have witnessed great amounts of time being wasted inventing and reinventing status schemes that were bound to fail because they do not reflect what actually happens to a defect or do not clearly and distinctly describe each status.

As with other measurement situations, the following fundamental measurement principles need to be applied to defect statuses:

1. Defect statuses must be mutually exclusive (no overlaps). This could be called the principle of exclusivity. It ensures that a defect could never correctly be described as belonging to two different statuses.

2. Status changes occur at points in time before which one status applies and after which a different status applies. This could be termed the principle of timing. Like the principle of exclusivity, it ensures that at any single point in time, all defects belong to one and only one status category.

3. Statuses must reflect the reality of what actually happens. The categories must accurately describe the changes that a defect goes through and be recognizable as such. People should be able to map actual defects to the statuses without confusion and without the need for lots of training on special terminology. The names of the categories should be obvious and easy for everyone to interpret the same way.

4. There must be enough statuses to reflect the changes that people need to use in order to do their work. This principle of adequacy means that additional subcategories will not emerge and undermine the credibility or accuracy of the statuses.

5. There must not be too many statuses that would cause confusion about fine distinctions among the categories. This principle of difference requires that changes will reflect major distinctions that are meaningfully distinct, different enough to warrant a status change, and about which people will agree. Adequacy plus difference together assure the right number of statuses.

6. The defect statuses should not combine two or more types of information into one set of categories (e.g., priority, severity, and statuses). This principle of simplicity requires that separate variables be kept separate so that combinations of factors will not cause confusion. People often confuse what they want to do with a defect with what has been done so far.

Based on these measurement principles and the best practices I have observed being used to manage defects, I have derived a set of seven standardized defect statuses that I now recommend to everyone. This set is complete, easy to understand, and non-overlapping. A full explanation of the reasoning behind each of them is included below. These statuses meet the list of measurement principles explained above. They should be readily understood and accepted by testers, developers, and project managers. They should also be capable of being understood by customers who report defects after the product goes into production.

This paper also includes descriptions of the six valid reasons for withdrawing a defect from a list of defects for a product: “Unable to be Replicated,” “Bad Test Case,” “Tester Mistake,” “Duplicate of Another Defect,” “Requirement Was Cancelled,” and “Requirement Was Postponed to a Future Release.” Please note that a reason for withdrawal is a separate variable from defect status and can only apply while the status is “Withdrawn.” This paper goes on to describe the eighteen standard status changes and ends with a state transition diagram of normal status changes that should help everyone visualize the model.

I have not been able to find another work that addresses this problem satisfactorily. What I have seen either does not go into detail about statuses or makes measurement mistakes as discussed above. Some defect tracking tools come “out of the box” set up incorrectly (violating one or more of those principles). To their credit, they allow users to customize defect statuses to suit company standards. Unfortunately, from what I have witnessed, most users invent other problematic sets of statuses, leading to confusion and arguments. This is what has led me to this work. I prefer not to cite herein all the mistakes I have witnessed, but rather to concentrate on the proposed solution.

Defect counts need to be accurate for our decisions about the effectiveness of our defect prevention and defect removal methods to be correct. Decisions about which defects to fix and when to fix them also depend on the accuracy of defect information. Adopting these standards will promote the needed accuracy and avoid confusion and wasted time. Don't keep reinventing this wheel. Adopt these standardized defect statuses and spend your time more productively fixing defects and finding more of them.

### **Seven Standard Defect Statuses**

The following are the seven standard statuses that a defect normally can go through. Note that not all defects go through all seven statuses.

#### 1. New

The best practice I have observed is to start each new defect report with a status of “New.” As soon as you start recording any information about a defect, its status is automatically “New.” The status remains “New” until you decide that the defect has been verified and should become “Open” or you decide it can not be verified and therefore should be “Withdrawn.”

#### 2. Open

An “Open” defect has been verified as real and therefore it needs to be fixed. This implies a handoff from testers (or customers) to developers. There is some requirement that is unmet or will be unmet unless the defect is fixed. “Open” defects stay “Open” until the people assigned to fix it declare that a fix has been made. See the next two statuses.

### 3. Pending New Build

This status indicates that somehow a repair has been made, but it has not been implemented into a build that is available for testing yet. Developers want this status to make it clear to everyone on the project that this defect is thought to be fixed, even if it has not been made available for retesting yet. If your project is not following a build method of version control, you might want to rename this status to one that matches your development methodology, but it is a best practice to have it available. This status may sometimes be skipped if a fix is immediately installed in the testable version. Do not rename this status “Fixed” because people confuse that with “Closed.”

### 4. Ready to Retest

This status indicates that not only has the fix been made to the component(s) that needed fixing, but it is incorporated into the latest build, or whatever you call it, and the module or system is ready to be tested to make sure the fix has worked. Best testing practices require thorough retesting plus bad fix testing of related requirements, and possibly even a complete regression test of all related functions, depending on your test strategy and your risk assessment of the situation.

### 5. Reopened

This status indicates that a defect that was once opened and then changed to some other status has been opened again. It is a best practice to distinguish between “Open” and “Reopened.” You want the additional information that an attempt has already been made to fix this one. The project manager might want to assign “Reopened” defects to the same person who worked on it before, or maybe not. “Reopened” defects might get assigned to some of the best staff members or to another person for a “fresh look.” The number of defects “Reopened” provides one type of measure of your repair team’s effectiveness. Management should analyze the number of reopened defects and work on eliminating the root causes of the mistakes that led to them. Of course this should be done with a proper attitude of teamwork seeking quality improvement rather than attacking the integrity of individual workers. Only repeated mistakes of the same kind should lead to “career counseling.”

### 6. Closed

This is one of the two possible endpoint statuses (“Closed” or “Withdrawn”). All real defects that really needed to be fixed should end up “Closed.” Some defects may remain “Open” for a long time, even after the production date, but there should always be a conscious decision to do so. Defects should never be “Closed” without being fixed and retested to verify that the fix was adequate. A defect may not be “Closed” just to make things look better than they are.

## 7. Withdrawn

This is the second of two possible endpoint statuses (“Closed” or “Withdrawn”). A defect that is “Withdrawn” has been declared not to be a real defect for one of six valid reasons. Any “Withdrawn” defect should indicate which reason applies. Defects may not be “Withdrawn” just to make things look better. If a defect is real, it must be dealt with until it can be “Closed” or it must be left “Open” and acknowledged as the reason that a requirement will continue to go unmet. Truly “Withdrawn” defects do happen, but not as a matter of convenience. See the section below for valid reasons.

### **Six Valid Reasons for Withdrawing a Defect**

The following are all of the valid reasons I have observed for deciding that a defect is not really a defect. A status change to “Withdrawn” means that a defect is being removed from the list of defects that need to be fixed for this release of the product. All “Withdrawn” defects do not count as current defects any more. They are neither “Open” nor “Closed” nor anything in between. They are non-defects now, for one of these reasons, but do not erase the information about them because you probably will refer to it again. Withdrawing a defect is only a (possibly temporary) change to its status. Always retain all defect records. Never erase one. Make sure no one withdraws a real defect for the sake of appearances. Any time you withdraw a defect, make sure you identify which one of the following reasons applies and attach that information to the record. You should withdraw only those defects that have one of these reasons.

#### 1. Unable to be replicated

Sometimes defects are just imagined or can not be replicated no matter what you try. Such defects should be “Withdrawn” because there is no way to deal with them. The imaginary defects will stay “Withdrawn” forever. The real ones will eventually reappear and can easily be “Reopened” and dealt with appropriately. At least you have saved the information about what happened the first time. While they remain “Withdrawn” these defects do not count in the defect totals unless they get “Reopened.”

#### 2. Bad test case

If a test fails and you thought there was a defect, but it turns out the test itself was incorrect, the reported defect did not really exist and its status should become “Withdrawn.” You should not erase these either, because you should study how many bad test cases you are creating and why. This is valuable information for improving your testing processes.

### 3. Tester mistake

Just like a bad test case, tester errors should not count as defects, but you want to “Withdraw” them instead of erasing them so that they can be studied or possibly “Reopened” as needed.

### 4. Duplicate of another defect

Each defect (each separate thing wrong that needs to be corrected) should count once and only once. Additional reports of defects already found are valid reports and may provide additional information about what is wrong, but they should not add to the defect count. Duplicates should be “Withdrawn” after the information about the circumstances under which it was found gets copied into the first report of that defect. Caution must be exercised to ensure that duplicates really are about the same defect and not just another similar defect. Even duplicates should not be erased, in case it is later discovered that it wasn’t really a duplicate, and needs to be “Reopened” as a real defect of its own. Also, whoever reported the duplicate may want to use that defect report as the key for tracking progress, so each withdrawn defect should “point” to all the other reports of that same defect. Management may want to study how pervasive the discovery of a defect was and improve testing strategies accordingly.

### 5. Requirement was cancelled

All defects must relate to requirements. If it is decided that a requirement is not correct, then any defects related to not meeting that requirement should be “Withdrawn.” There may be a lot of work to remove or undo parts of the system on account of the requirements change, but at least you don’t have to worry about fixing the related defects. “Withdraw” them and they no longer count. Watch out for potential misuse of this category and confusion over this one versus the next one.

### 6. Requirement was postponed to a future release

It is possible that a requirement may be declared postponed but not cancelled. Fixing a defect associated with that requirement gets postponed too, so it should be “Withdrawn.” It no longer counts as a defect for this release of the product. There may be a lot of work to do removing or undoing other parts of the system on account of the requirements change, but at least you don’t have to worry about this defect for now. These defects should not be erased. No defects ever should be erased. At the beginning of the project for the next release, the list of these “Withdrawn” defects becomes a list of desired enhancements to consider (requirements to add to the next release) during the next requirements analysis.

Enhancement proposals should never get reported as “defects” because they relate to requirements that do not exist for this release. They should be handled by your requirements analysis process. There can be no defects related to non-existent

requirements. However, if an enhancement request does get into your defect database, you will need to decide whether to “Withdraw” it for one of the two above reasons (based on the likelihood of the requirement being adopted or not), or decide to change your requirements now and give the “defect” legitimacy and an “Open” status.

### **Eighteen Standard Status Changes.**

Out of 43 possible status changes (one initial one plus changes from 7 statuses to each of the other 6), I conclude that only 18 of them should be ordinarily expected. For example, you would not expect to go from “Closed” to “Ready to Retest” without first reopening the defect. Of course it is always possible to make a mistake and close the wrong defect, in which case you would want to change it back to “Ready to Retest” (an undo operation as opposed to a normally expected status change).

Some illogical changes should never happen, like going from “Closed” to “Open.” The “Reopened” status precludes that change. Similarly, our rules should not allow going from “Open” directly to “Closed” without any retesting. Below are the 18 standard (normally expected) status changes with a brief description of each. Please note that changes numbered 3, 4, 6, 8, 9, and 10 are most often made by developers to signal a change in status that is under their control, while the other changes are most often made by testers or project managers. See figure 1 at the end of this paper for a helpful diagram of these changes.

#### 1. From no status to “New”

When a new defect report gets created, initially there is no status. A defect tracking tool could automatically make this change for you every time you create a new defect report. It should not be possible to return to having no status from any other status.

#### 2. From “New” to “Open”

Many companies require that a defect be verified by testers before they consider them “Open” for assignment to be repaired. I consider this to be a best practice and thus incorporated this into the model. If it were not for this best practice, you would not need a status of “New” and would go directly from no status to “Open,” but I do not recommend that.

#### 3. From “Open” to “Pending New Build”

This intermediate status is most often requested by developers who want to be able to distinguish among defects that are “Open”, or “Reopened”, and those that they know they have tackled already, even if the repairs have not yet been turned over to testers for retesting. I have seen companies try to get along without this status only to return to

using it when the developers cried foul. The typical reason is that several repairs will be released together in the next build, which may not happen for a while, and they want to have the progress of their work be apparent in the counts. The name of this status is the one status you might want to rename, based on the way your company does development and what you call the next testable copy of the software. It is a best practice to have this status, whatever you name it. I do not recommend removing it, nor do I recommend renaming any of the other statuses for the sake of standardization. See change number 6 below for times you want to bypass it.

#### 4. From “Pending New Build” to “Ready to Retest”

Once a build is created that incorporates fixes for a defect or group of defects, those defects can all be changed to “Ready to Retest.” This change should be done by developers. It represents a handoff to the testers.

#### 5. From “Ready to Retest” to “Closed”

Once a defect has been retested and those tests have been passed, then the tester can change the status of that defect to “Closed.” I believe the word “Closed” should be used as opposed to “fixed” because “Closed” is the logical opposite of “Open” and because developers consider defects that are “Pending New Build” or “Ready to Retest” to be “fixed” even if they have not yet been retested. I have seen way too much unwarranted confusion (and unproductive fighting) over this issue and conclude that the safest thing to do is to stick with “Closed.”

#### 6. From “Open” to “Ready to Retest”

If a fix for a defect is made without needing to wait for a new build, this is the status change to use. It is the equivalent of a number 3 plus and immediate number 4. This change should be done by developers. It represents the same handoff to the testers as a number 4 change.

#### 7. From “Ready to Retest” to “Reopened”

If retesting uncovers the fact that a defect has not really been fixed (or not fully fixed), the defect can not be “Closed” and should be “Reopened.” One could say that the status just goes back to “Open”, but it is a best practice to provide the additional information that this defect was thought to be repaired once, but it was in fact not repaired. This is, or should be, very helpful to developers. See the discussion of “Reopened” above.

#### 8. From “Reopened” to “Pending New Build”

This is just like the change from “Open” to “Pending New Build.”

#### 9. From “Reopened” to “Ready to Retest”

This is just like the change from “Open” to “Ready to Retest.”

#### 10. From “Pending New Build” to “Reopened”

It is possible for a developer to suddenly notice that a defect that was thought to be fixed really isn't fixed yet. It has not been released to testers in a new build yet, but it has been declared “fixed” by changing the status to “Pending New Build.” Developers need a means for taking a defect back for more repairs. There is no reason to subject such defects to new builds and retesting if you know it will fail again. The best practice in this case is for the developer to reopen it, signaling that it has a history of changes and may need special attention, similar to defects that have failed retesting. Developers should not be punished for using this change. It is honest and saves work.

#### 11. From “Closed” to “Reopened”

This should be rarely needed, but it is possible that even after passing retests, a defect can be found to not actually be fixed. It needs to be clear that it is this same defect that is being “Reopened”, and not a new and different one that is being opened. If a bad fix causes a new defect, that's different. Create a new defect record for those. This status change means the original defect was not repaired and retesting did not catch the fact. Instances of this change represent dual process improvement opportunities. That is, you might want to investigate both your failed repairs and your failed retesting for possible improvements.

#### 12. From “New” to “Withdrawn”

Sometimes, before you can verify a defect, you discover that it never should have been reported. See the section on the 6 valid reasons for withdrawing a defect.

#### 13. From “Open” to “Withdrawn”

Sometimes, even after you verify that a defect is real, there could be an overriding reason to withdraw it such as changing requirements. See the section on the 6 valid reasons for withdrawing a defect.

#### 14. From “Pending New Build” to “Withdrawn”

It is possible, even after a defect has been fixed, that a decision can be made that leads to it being “Withdrawn” (for example again, changing requirements). This status change implies that the “fix” needs to be undone somehow. See the section on the 6 valid reasons for withdrawing a defect.

#### 15. From “Reopened” to “Withdrawn”

A “Reopened” defect can be “Withdrawn,” especially if the requirements are changed. This status change implies that the previously attempted “fix,” even though incorrect or incomplete, needs to be undone somehow. See the section on the 6 valid reasons for withdrawing a defect.

#### 16. From “Ready to Retest” to “Withdrawn”

Like “Pending New Build” to “Withdrawn” above, it is possible, even after a defect has been fixed, that a decision can be made that leads to it being “Withdrawn.” This status change also implies that the “fix” needs to be undone somehow. See the section on the 6 valid reasons for withdrawing a defect.

#### 17. From “Closed” to “Withdrawn”

This should be very rarely needed, but once again it is possible that even after a defect has been fixed, retested, and “Closed,” a requirements change or some colossal blunder is uncovered and the defect gets “Withdrawn.” This probably means that lots of rework needs to be done to undo original work plus repairs. See the section on the 6 valid reasons for withdrawing a defect.

#### 18. From “Withdrawn” to “Reopened”

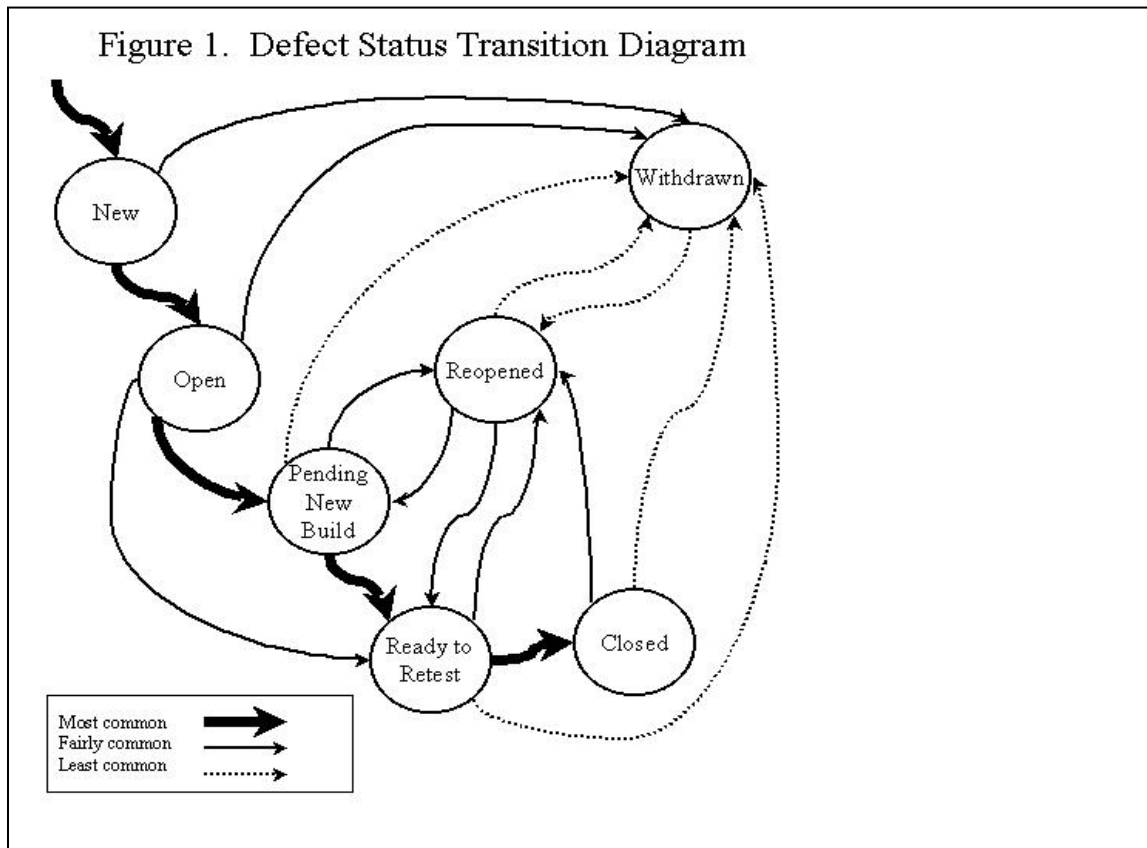
In spite of all the valid reasons you might withdraw a defect from your list, it is possible that it might get “Reopened.” The most likely reason would be that the requirement comes back to life because a customer decides they can’t live without that requirement being met. If the system does not or will not meet the requirement on account of the previously reported defect, that defect springs back to life. This is another example of why you should never erase or completely discard any defect. See the section on the 6 valid reasons for withdrawing a defect.

### **Measurement Implications.**

Accurately counting defects is one of the most basic and most important things you can do for measuring software quality. Following the standards provided by this paper should help you improve the accuracy of your defect counts. Remember that defect counts always apply to a specific version of a product. The total number of defects reported minus those withdrawn is the total number of known defects to have been found for that version thus far. Subtract from the total number of known defects those that are closed and you have the total number of defects that are in one stage or another of being repaired. Managing testing and defect removal efforts should become markedly

easier by following these standards. At least it should cut down on the unfruitful arguments that do not find or fix anything.

Companies should be interested in studying how long it takes to find and repair defects, and how many defects have been found versus how many they expected to find, along with many other aspects of defect identification and removal. All of these investigations depend on accurate counts. I encourage the use of these standards to improve that accuracy, and I would remind the reader that each defect represents an unmet requirement that customers care about. We need to be as effective and efficient as we can about improving. I hope this will help you do that.



file:///Users/joyce/Desktop/daves\_article.pdf