AutomatedQA
7935 East Prentice Avenue, Suite 105
Greenwood Village, CO 80111
303.768.7480 (Main) 303.768.7481 (Fax)
www.AutomatedQA.com   www.SoftwarePlanner.com

Software Planner
Testing Best Practices

Contents

Test Case Creation

Once requirements have been created and approved, while coding is in process, it is time to create test cases.   The idea is to have a complete list of test cases completed before coding is complete, that way you do not waste time during the QA process.

Important Custom Fields for Test Cases

To ensure accurate test case tracking, create the following custom fields for test cases:

- **Smoke Test** – Yes/No checkbox that identifies if the test case is used for smoke tests.

Additionally, use the Group field (choice list) to identify if the test case is a Regression or QA Test Case, the choice list should have 2 values:

- **Regression Test Cases**
- **Standard Test Cases**

Use the Subgroup field (choice list) to identify if the type of Test Case, the choice list should have 2 values:

- **Positive Test Cases**
- **Negative Test Cases**

Smoke Tests

If you want to ensure that your testing process will be fruitful, it is common practice to create smoke tests. Smoke tests are simply a list of about 20 to 30 test cases that address the major areas being tested. For example, if you are creating a new Project Management system, a smoke test will include basic test cases to ensure that you can create a new project plan, add tasks to a project plan, rename a project plan, make changes to a project plan, and delete a project plan. So these are not "over the top" test cases that really tax the functionality, it is simply tests that should work because they are very basic.

If the Smoke Tests fail, it may be that the QA process should stop and wait until those are fixed, or it may be that testing can progress with requirements that did not fail the test case (as to not impede progress). The idea is that if the requirement cannot pass the basic tests, it is not ready for a vigorous QA testing process, so all smoke tests for a specific requirement must pass before the QA team spends valuable time testing it.

When creating the smoke test, use these guidelines:

- Put the smoke test cases in a status of **Awaiting Run (Smoke Test)**.
- Assign the smoke test cases to the person merging the code.
- Ensure that the **Smoke Test Flag** is turned **ON** for each of these.

Review the smoke test set as a team, here are the key things you are looking for:

- Is every Requirement for the release covered by at least one smoke test item?
- Are the smoke test items clear enough to run?
- Are there any missing smoke test cases to ensure a good start of testing?
- Are there any smoke test items that are too in-depth and should not really be a smoke test (should be a standard test case)?
- Do we have a manageable set of smoke test cases (20 to 30 test cases)?

Positive Testing

Positive testing is testing the software in the exact way it was designed. To create these test cases, you will want to follow the requirements document to ensure that all features of the software are tested. In positive testing, you are not trying to "trick" the system; you are simply testing all the features per the design.

All of these positive test cases should be in the **folder for the release**, should have a **Group** of **Standard Test Cases**, and should have a **Status** of **Awaiting Run**.

Negative Testing

Negative testing is used to try to break the software in ways that a typical user might, by entering erroneous data. Here are some examples of negative testing:

- **Date formats** – Try entering invalid dates (like 02/30/2006). Try entering alphabetic dates (like Jan 1,2006), try entering in totally bogus information (xxxxxxx).
- **Numeric formats** – If you know a field must allow only numeric entry, try entering character data (abcdefg). Try entering commas, decimals, hyphens, dollar signs.
- **Field Sizes** – If you know the size of a field (let's say up to 20 characters), try entering a large amount of data (like 100 X's), it should handle that gracefully. Many times this will cause a crash. I like to go to every field and just enter tons of xxxxxxxxxxxxxxxxxx and see what happens.

All of these negative test cases should be in the **folder for the release**, should have a **Group** of **Standard Test Cases**, and should have a **Status** of **Awaiting Run**.

Relational Testing

Relational testing is where you try to break the software in a relational way. Here are some examples:

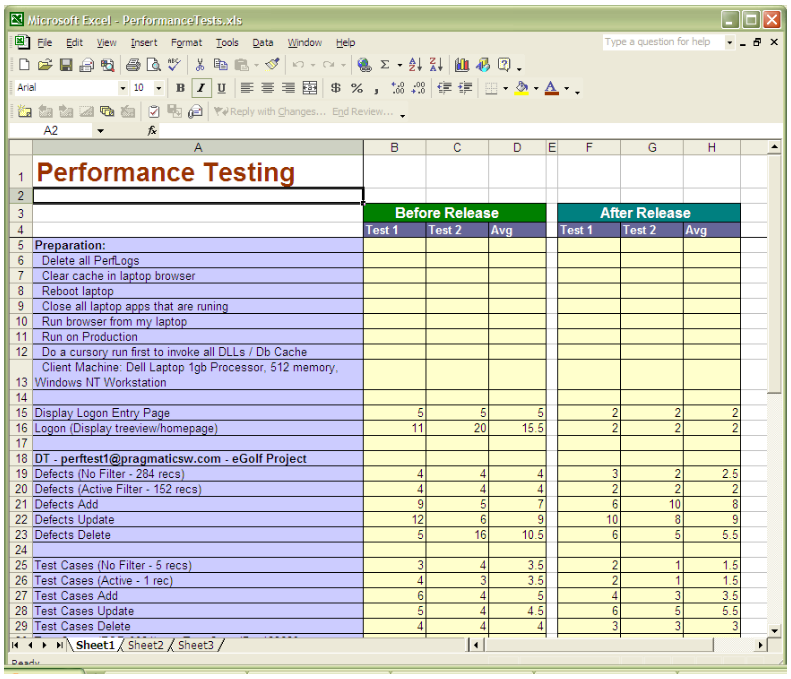
- **Duplicate records** – Try to create duplicate records. For example, if we are testing the setting up of a Project Plan; create a new project plan by the name of Widgets 1.0. Then add another project plan by the same name. That may be a business flaw; maybe you should never have 2 project plans by the same name. This is going to be dependent on your actual requirements, but should always be considered.
- **Renaming items to cause Duplicate records** – In the prior example, we tried creating 2 project plans with a name of Widgets 1.0. In this case, we will create a project plan named Widgets 1.0 and another named Widgets 1.1. Then try to rename the Widgets 1.1 to Widgets 1.0, it should recognize that this creates a duplicate.
- **Deleting records should delete dependent records** – If you delete data that has dependent records, it should delete both the parent and the child records. For example, if you delete a Project Plan that has 100 tasks, it should delete all 100 tasks, and the Project Plan itself.
- **Deleting records with dependencies** – In other scenarios, you may want to prevent a person from deleting a parent item if it has child records. For example, let's assume you have a test case that is related to a defect. However, if you delete the defect, it will lose the relation to the test case, so you would not allow the deletion of the defect unless the test case was deleted first. This is just an example, and it will depend on your business rules.
- **Auditing** – Always test to ensure that all actions done are audited (if designed to do so). Run audit reports to see if that is the case.

All of these relational test cases should be in the **folder for the release**, should have a **Group** of **Standard Test Cases**, and should have a **Status** of **Awaiting Run**.

Performance Testing

It is a good idea to have some performance test cases designed for each release to ensure that each release maintains adequate performance. Create a spreadsheet for Performance Testing, similar to the one below. It contains the following sections:

- **Before Release / After Release** – Notice that you will record timings before the code is merged and after, that way you can determine if the release is faster or slower than before.
- **Preparation** – To correctly achieve timings, you must clear your cache, reboot your machine, etc before doing the timings.
- **Area Testing** – Test each critical area of your software, logged in with different accounts (accounts with small amounts of data, accounts with large amounts of data, etc).


Microsoft Excel - PerformanceTests.xls

Performance Testing

	Before Release			After Release		
	Test 1	Test 2	Avg	Test 1	Test 2	Avg
Preparation:						
Delete all PerfLogs						
Clear cache in laptop browser						
Reboot laptop						
Close all laptop apps that are runing						
Run browser from my laptop						
Run on Production						
Do a cursory run first to invoke all DLLs / Db Cache						
Client Machine: Dell Laptop 1gb Processor, 512 memory, Windows NT Workstation						
Display Logon Entry Page	5	5	5	2	2	2
Logon (Display treeview/homepage)	11	20	15.5	2	2	2
DT - perftest1@pragmaticsw.com - eGolf Project						
Defects (No Filter - 284 recs)	4	4	4	3	2	2.5
Defects (Active Filter - 152 recs)	4	4	4	2	2	2
Defects Add	9	5	7	6	10	8
Defects Update	12	6	9	10	8	9
Defects Delete	5	16	10.5	6	5	5.5
Test Cases (No Filter - 5 recs)	3	4	3.5	2	1	1.5
Test Cases (Active - 1 rec)	4	3	3.5	2	1	1.5
Test Cases Add	6	4	5	4	3	3.5
Test Cases Update	5	4	4.5	6	5	5.5
Test Cases Delete	4	4	4	3	3	3

Use this to record benchmarks before the code is merged. Then run these benchmark tests against the merged code to determine if response time has degraded with the new release. This performance test case(s) should be in the **folder for the release**, should have a **Group** of **Standard Test Cases**, and should have a **Status** of **Awaiting Run**.

Automated Testing

Automated Testing has many benefits including: Quicker Releases – By having your regression test cases run automatically, your software quality team can concentrate on testing new features of your software and less time regressing existing features. Higher quality releases – Your software releases will have fewer bugs and require less customer support because they will be of higher quality. Happier Customers – Your customers will be happier and more willing to serve as testimonials for future prospects.

TestComplete (**http://www.TestComplete.com**) is an award winning automated testing tool. It allows you to create automated test cases via record and playback, scripting or via a keyword driven approach. With the Software Planner bridge, you can create your automated test cases in TestComplete and organize, run, and/or schedule them to be run directly from Software Planner. You can also use Software Planner's ability to create hosts – allowing you to run (or schedule) those automated test cases on different machines. Finally, Software Planner's impressive dashboards and reports allow you to analyze both the automated and manual test effort from a single interface.

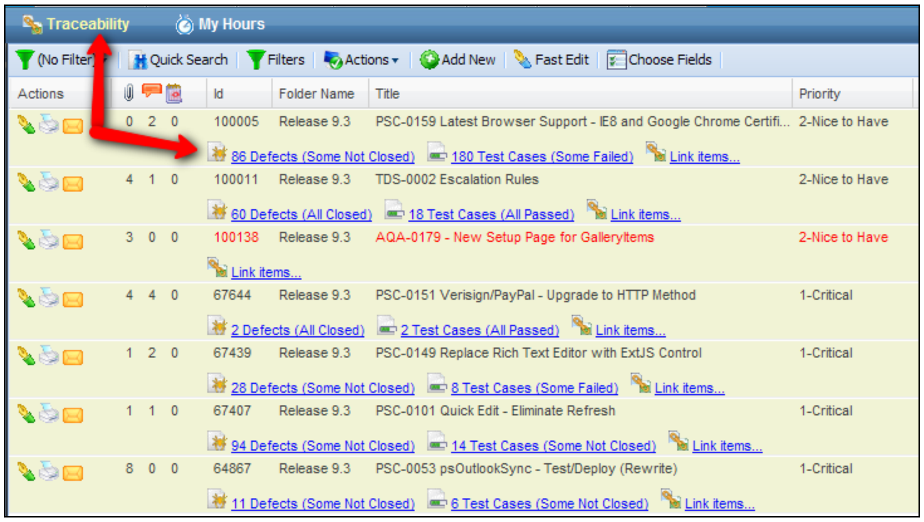
To learn more about the Automated Testing integration, see the User's Guide:
http://www.softwareplanner.com/UsersGuide_TC.pdf

Test Case Creation and Traceability

When creating test cases, make sure you create solid steps so that the person running the test case will fully understand how to run the test case:

- Always tie Test Cases to one or more Requirements to ensure traceability.
- Always name the test case Title as the Requirement Number, plus the section of the Requirement, plus a description of the test case. Here is an example:
  FOR-0026.001 Defects – Move items, confirm correct folder
- Always include Login info to let the tester know what application to run this in:
  Log into [X] Application as John Smith
- Make sure test case is assigned to the correct person
- Make sure the Group is correct (Standard Test Cases vs. Regression Test Cases).
- Make sure the status is correct.
- Make sure the Smoke Test flag is correct.
- Always number the test case steps (never leave the numbers off).
- Make sure you have great STEPS, here is an example:
  1. Click Defects/Edit
  2. Update the Status from Active to Closed, press Submit
  3. Should update the status, write an audit and stamp the Closed Date as today

Below is an example:



Actions				Id	Folder Name	Title	Priority
	0	2	0	100005	Release 9.3	PSC-0159 Latest Browser Support - IE8 and Google Chrome Certifi...	2-Nice to Have
				86 Defects (Some Not Closed) 180 Test Cases (Some Failed) Link items...			
	4	1	0	100011	Release 9.3	TDS-0002 Escalation Rules	2-Nice to Have
				60 Defects (All Closed) 18 Test Cases (All Passed) Link items...			
	3	0	0	100138	Release 9.3	AQA-0179 - New Setup Page for GalleryItems	2-Nice to Have
				Link items...			
	4	4	0	67644	Release 9.3	PSC-0151 Verisign/PayPal - Upgrade to HTTP Method	1-Critical
				2 Defects (All Closed) 2 Test Cases (All Passed) Link items...			
	1	2	0	67439	Release 9.3	PSC-0149 Replace Rich Text Editor with ExtJS Control	1-Critical
				28 Defects (Some Not Closed) 8 Test Cases (Some Failed) Link items...			
	1	1	0	67407	Release 9.3	PSC-0101 Quick Edit - Eliminate Refresh	1-Critical
				94 Defects (Some Not Closed) 14 Test Cases (Some Not Closed) Link items...			
	8	0	0	64867	Release 9.3	PSC-0053 psOutlookSync - Test/Deploy (Rewrite)	1-Critical
				11 Defects (Some Not Closed) 6 Test Cases (Some Not Closed) Link items...			

Notice that each requirement links to one or more test cases and/or defects.

Test Case Review

Once all test cases are defined for a Requirement, the tester should setup a meeting to review all the test cases for the Requirement with their team members (programmers, project manager, other testers, etc). To maximize use of the meeting, bring a projector and display the test case listing screen. If all team members review the test cases ahead of time (and make notes of missing test cases, test cases that were not clear, etc), this meeting can be done in about 30 minutes. If everyone is not prepared, it can take hours.

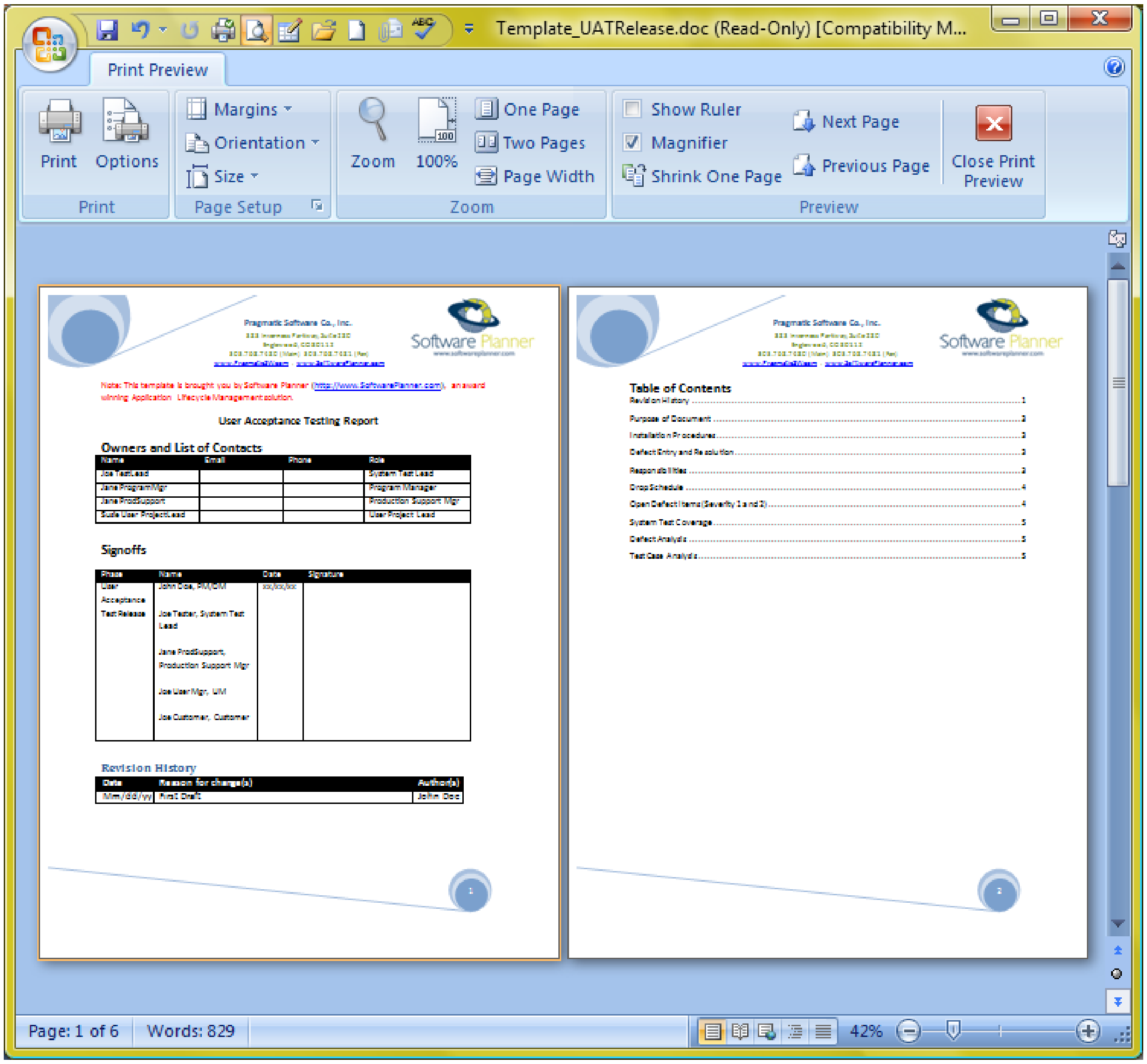
In the meeting, go to the Test Cases listing screen and filter on the specific Requirement in which test cases are being reviewed. Then go through each test case, to ensure that each test case is understood by the team members, that each test case is adequate, that there are no missing test cases for negative tests, etc.

If any new test cases are need, or if any test cases should be changed, update them on the spot. Once completed, mark the Requirement as **Test Case Reviewed** (custom field) in your Requirements area, this signifies that all test cases for the requirement has been fully reviewed and approved.

User Acceptance Test Documentation

Prior to launching into User Acceptance Testing (UAT), supply your client with a document. This document describes Installation Procedures, Defect Entry and Resolution, Roles and Responsibilities, Drop Schedule, and describes any open Defects.

TIP! This template is available at http://www.pragmaticsw.com/Template_UATRelease.doc



Pragmatic Software Co., Inc.
[illegible]

Note: This template is brought you by Software Planner (http://www.SoftwarePlanner.com), an award winning Application Lifecycle Management solution.

User Acceptance Testing Report

Owners and List of Contacts

Name	Email	Phone	Role
Joe TestLead			System Test Lead
Jane ProgramMgr			Program Manager
Jane ProdSupport			Production Support Mgr
Suzie User ProjectLead			User Project Lead

Signoffs

Phase	Name	Date	Signature
User Acceptance Test Release	John Doe, PM/DM Joe Tester, System Test Lead Jane ProdSupport, Production Support Mgr Joe User Mgr, UM Joe Customer, Customer	xx/xx/xx	

Revision History

Date	Reason for change(s)	Author(s)
Mm/dd/yy	First Draft	John Doe

1

Pragmatic Software Co., Inc.
[illegible]

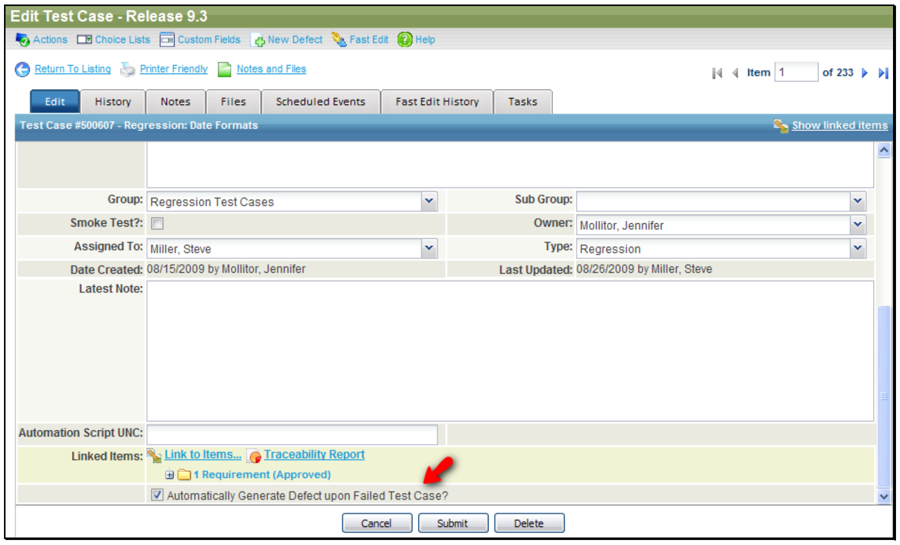
Table of Contents

2

Defect Resolution Best Practices

Defect Creation

If running a test case and it fails, use the Automatic Defect generation feature to generate the defect. This quickens your defect creation process and automatically fills in information from the test case. It also creates a linkage between the defect and the test case that failed.



Test Case #500607 - Regression: Date Formats

Group:	Regression Test Cases	Sub Group:	
Smoke Test?:	☐	Owner:	Mollitor, Jennifer
Assigned To:	Miller, Steve	Type:	Regression
Date Created:	08/15/2009 by Mollitor, Jennifer	Last Updated:	08/26/2009 by Miller, Steve
Latest Note:			
Automation Script UNC:			
Linked Items:	1 Requirement (Approved)		
	☑ Automatically Generate Defect upon Failed Test Case?		

Defect Submission

When submitting defects, you want to make it simple for the programmer to reproduce the error, and it is great to have screen shots or a movie of any errors or any other things that will be helpful to the programmer. For screen shots and movies, we recommend a free screen capture tool called Jing. You can download it at www.jingproject.com. It allows you to quick take a screen shot, mark it up with arrows and text, and you can even create narrated movies that show exactly how the issue was encountered. The free version of Jing allows up to 5 minutes of narrated videos to be created and you can upload it their website and can access it via a single click. Here is an example of a movie:

http://screencast.com/t/YWRiMDUyNzE

Also, when submitting defects, enter a good set of "Steps to Reproduce", here is an example:

1. Log into SP as admin
2. Click Test Cases/Automation Lab
3. Press Execute Now for the Regression Test Set
4. It gives an error
5. See this movie: http://screencast.com/t/YWRiMDUyNzE

Defect Priority

It is important to set defect priorities so that programmers will work the defects in priority order. Below are the established priorities:

- **1-Fix ASAP**
- **2-Fix Soon**
- **3-Fix if Time**

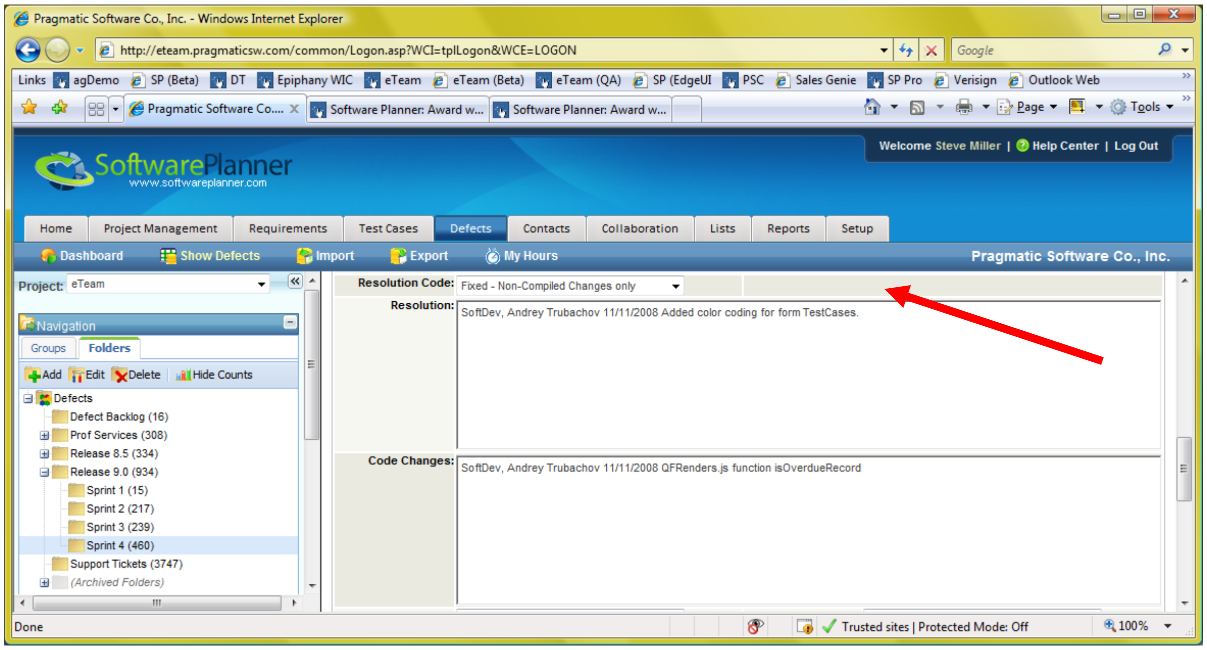
Defect Severity

It is important to set defect severities that are objective, not subjective:

- **1-Crash** – This crashes the system with no work around.
- **2-Major Bug** – This is a major bug with no work around.
- **3-Workaround** – This is a major bug that does have a work around.
- **4-Trivial Bug** – This is a trivial bug (spelling, cosmetics, etc.)

Defect Resolution

Upon resolving a defect, the programmer should choose a resolution code and enter descriptive information about what they did to fix the issue in the resolution field.



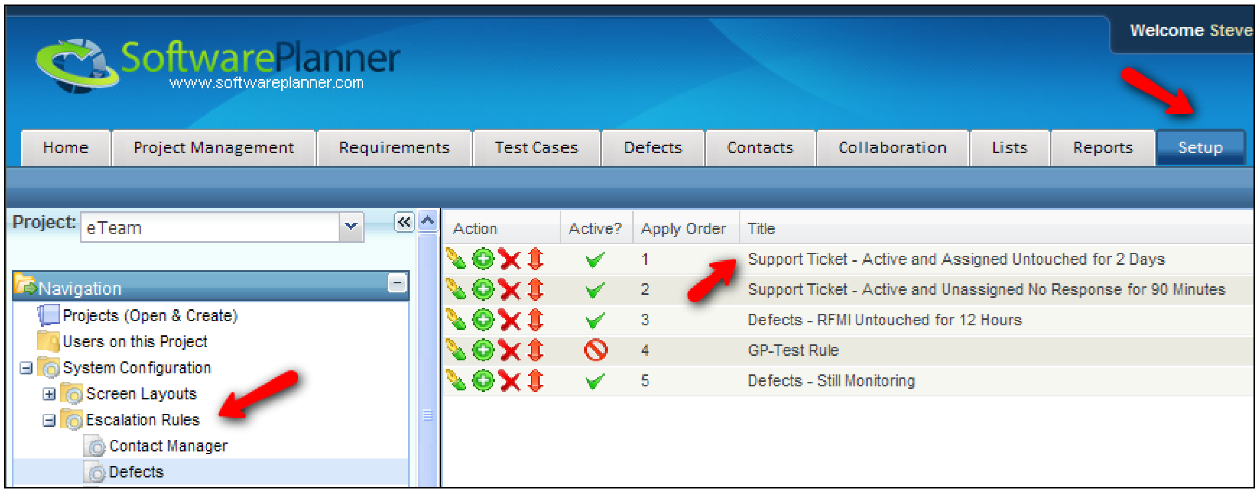
Resolution Code: Fixed - Non-Compiled Changes only

Resolution: SoftDev, Andrey Trubachov 11/11/2008 Added color coding for form TestCases.

Code Changes: SoftDev, Andrey Trubachov 11/11/2008 QFRenders.js function isOverdueRecord

Escalation Rules

Software Planner allows you to setup escalation rules. For example, you may want to have an escalation triggered for any ACTIVE defect that has not been updated in a week. When the escalation happens, you might want it to send out an email as well as update the priority to a higher level and/or even reassign the defect. To set these rules, go to Setup / Screen Configuration / Escalation Rules / Defects.



Action	Active?	Apply Order	Title
	✓	1	Support Ticket - Active and Assigned Untouched for 2 Days
	✓	2	Support Ticket - Active and Unassigned No Response for 90 Minutes
	✓	3	Defects - RFMI Untouched for 12 Hours
	⦸	4	GP-Test Rule
	✓	5	Defects - Still Monitoring

Navigate through the wizard to setup the rule.

Learn More

If you wish to learn more about Software Planner, request a free a trial, or receive a personalized demo of the product, contact **AutomatedQA** at +1 978-236-7900. You can also learn more at http://www.SoftwarePlanner.com.