SMARTBEAR

Ensuring Software Success℠

Software Development / QA Tough Issues: 15 Thorny Issues and How to Resolve Them

A SmartBear White Paper

Learn how metrics can be used to deal with sticky situations that come up during software development and testing process. In this whitepaper we'll describe a number of approaches, including common sense fixes, for tough (yet common) Quality Management, Project Management & Development issues.

Contents

Headaches and Thorny Situations

If you've worked in software development and/or Quality Management (QM) for many years, you've probably found yourself in all kinds of thorny situations when it comes to the software life cycle. In 26 years, I sure have, and I have the bruises to prove it! Missed deadlines, buggy releases, upset customers and cranky bosses drilling me on why I didn't foresee and proactively plan for these issues.

The honest answer is that software development and QM is very complex, has many variables, and is not an exact science – it takes planning and oversight to reduce the risk of things going wrong. Over the years, I've identified ways to mitigate many of the thorny issues that come up. The approach I use is to start with good engineering practices, measure what I can and make adjustments when I see things going astray. Unfortunately, some developers may have the need to feel much more reckless.

Ever feel the need to jump in the car, put a blind-fold on and start driving break-neck speed? If you do, call me ahead of time so I can get the heck out of your way! Many of us take that approach in software development and QA -- we rush the designs, don't track (or ignore) important metrics, and whine when we miss deadlines, ship buggy code, and have to face the customer when they identify the issues.

This whitepaper describes a number of approaches I found as common sense fixes for tough (yet common) issues that come up during the software lifecycle. I have divided these approaches into two distinct categories. Later we will discuss Quality Management related issues, but first let's discuss Project Management and Development.

Project Management / Development - Common Thorny Situations

Below are some common problems software development teams face and ways I've found to combat them. So let's open up the hood and go to work…

Thorny Issue #1: Why do my software releases take 6 or more months?

So let's say you are working on a software project, defined the requirements upfront, and it still took 6 months or more to get into production. Now how you got here may have varied. You may have expected it to take 6 months, or you may have expected it to take just a few months and things kept getting delayed until 6 months or more elapsed and you're left wondering what the heck happened.

Whatever the circumstance, 6 months is just too long to get a software project done, I call this the Elephant syndrome. How do you eat an elephant? One bite at a time! Hopefully, you don't choke…

Now this problem may have been caused by staffing or quality issues (we will address those later), but most likely, you bit off more than you could chew. Fixing this is quite easy – instead of trying to deliver a complete software product with all the bells and whistles you can think of, break it down into manageable short releases (no more than 2 weeks in duration) with a specific and pragmatic set of features and scope.

This approach is called an "Iterative" approach to software development, and has been especially made popular by the Agile community. But the truth is that iterative development predates Agile and was first suggested in classic development process work accomplished by Winston Royce in defining the lifecycle that became known as Waterfall. You can also use an iterative approach with Waterfall development (where you fully define all requirements up front, do all the coding, and then do all the testing in a staged approach) or you can use Agile development (where you set a specific time period for your software development and testing and only do the things that fit within that window). In my personal experience, Agile development lends itself nicely to solving this problem (iterations are called Sprints) but it can certainly be done with an iterative Waterfall approach. Its common sense – undertake less work and gets done quicker.

Thorny Issue #2: What things should I attack first when planning an Iteration?

So you're on board with limiting the work but let's say you are worried about delivering something that is useful. Well let's take an example. Let's say that you are building Contact Management software that is used to track customer and sales contacts. When you start planning out the software, you can quickly get overwhelmed with all the features and functions needed to pull this off.

I can just imagine your wheels turning. I've got to build a security system, contact listing, contact maintenance screens, reports, dashboards, email functionality, import and export features – the list goes on forever. Stop stressing! In the first iteration (or sprint), just build the contact listing and the ability to add or edit a contact. Forget about security, auditing, ability to delete contacts, yada yada -- just focus on the bare necessities. The juicy stuff can come later.

In fact, you may even be inclined to start gold plating the contact listing. You might dream up ways to allow in-place editing, print preview, file attachment capabilities, and lots of other bells and whistles. Those are all cool ideas, but let the features materialize in future iterations. If you are developing this for internal use, your team could actually start using the bare-bones model after the first iteration – allowing you to start getting some immediate return on investment and, more importantly, feedback from your customers.

Thorny Issue #3: Why are my programmers always running behind on their deliverables?

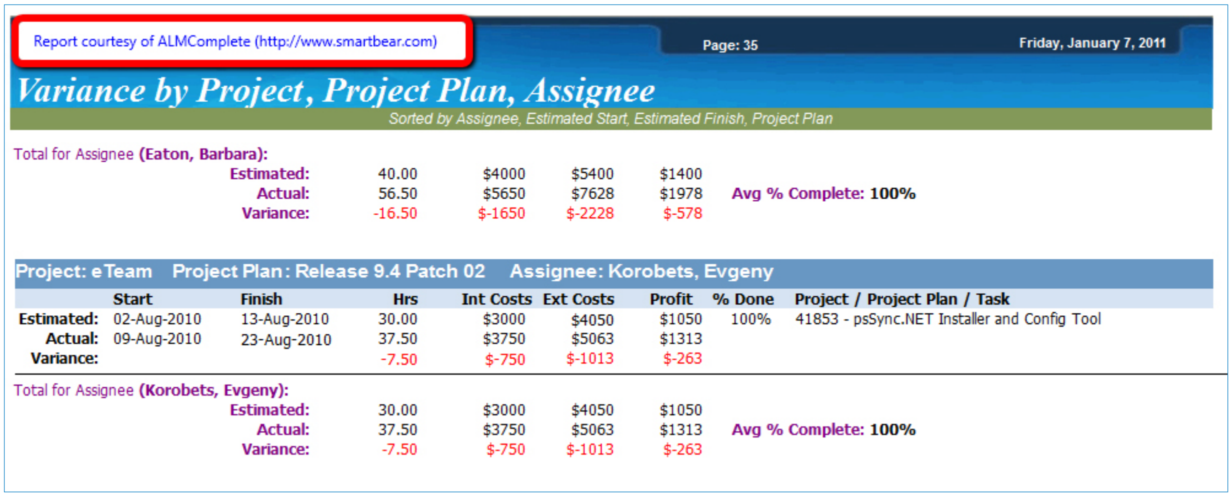
This can be caused by many issues; I will talk about the common ones. First off, who is estimating the work, the project manager or programmer? If it is the project manager, that is your first issue. You should always allow the programmer to estimate their work because they are more intimate with the essential things that have to get done for each programming task.

OK, so you say you're programmers are estimating and they are still running behind. At this point, we need to dig a little deeper. Are the requirements properly defined and did the programmer take the time to put together a design when doing the estimate? If not, they are probably under-estimating because they did not decompose the requirement into low level design tasks needed to fulfill the requirement. The more you know about design, the better you can estimate.

Another common issue is scope creep. You start off with a design, start coding and someone comes up with some cool enhancements to the feature you are working with and you try to fit that coolness into the same schedule. If that happens, you need to re-estimate the work based on the new features to ensure that it will still fit within your time constraints, and if it won't, you will have to let the cool feature wait.

Finally, some people are not very good at estimating tasks. You have optimists that like to think they can get more work done than they really can (I fall into that category), and some pessimists that sand bag by putting these really huge estimates out there to cover their butts. Both of these issues are detrimental to a project, so you need a way of determining who estimates well and who doesn't, and find ways to improve their estimating skills.

Luckily, this is easy to solve. Have each team member estimate their tasks and track their time towards it. Then after the iteration ends, do a variance report (estimate minus actual). You will find that on the first go-around most of us will underestimate the tasks. Here is an example:



Report courtesy of ALMComplete (http://www.smartbear.com)

Page: 35

Friday, January 7, 2011

Variance by Project, Project Plan, Assignee

Sorted by Assignee, Estimated Start, Estimated Finish, Project Plan

Total for Assignee **(Eaton, Barbara):**

Estimated:	40.00	$4000	$5400	$1400	
Actual:	56.50	$5650	$7628	$1978	**Avg % Complete: 100%**
Variance:	-16.50	$-1650	$-2228	$-578	

Project: eTeam Project Plan: Release 9.4 Patch 02 Assignee: Korobets, Evgeny

	Start	Finish	Hrs	Int Costs	Ext Costs	Profit	% Done	Project / Project Plan / Task
Estimated:	02-Aug-2010	13-Aug-2010	30.00	$3000	$4050	$1050	100%	41853 - psSync.NET Installer and Config Tool
Actual:	09-Aug-2010	23-Aug-2010	37.50	$3750	$5063	$1313		
Variance:			-7.50	$-750	$-1013	$-263		

Total for Assignee **(Korobets, Evgeny):**

Estimated:	30.00	$3000	$4050	$1050	
Actual:	37.50	$3750	$5063	$1313	**Avg % Complete: 100%**
Variance:	-7.50	$-750	$-1013	$-263	

Then on your next iteration, collect estimates from each team member, but buffer them by the variance (based on the individual team member) from the last sprint. So if John was 25% under-estimated his aggregated tasks last iteration and he estimates 100 hours for his tasks in the upcoming iteration, make his estimates 125 hours.

Then in your retrospective, take the same approach (look at the variances in the new sprint and make buffer adjustments accordingly). Over time, you build a much better estimator and more reliable estimates.

Thorny Issue #4: How can we estimate our efforts better?

Don't you love it when your manager or customer comes to you and says "I need you to create this [insert product feature here], how long do you think it will take?" As a journeyman, you might say,

"Well, let's see… I think it will be a week of work." Then you start developing the software feature, the complexity of tasks start to become clear and that week of work becomes a month of work. Once a month passes, your manager and/or customer gets frustrated, you are working your fingers to the bone and everyone is pointing the finger at you because you told them a week was plenty of time. Sound familiar? Shoot me now, put me out of my misery!

This is another thorny issue that can certainly be resolved. The real issue is that when someone asks for estimates without a clear vision of what they want, there is no way to provide them (unless you are clairvoyant and if you are you should be playing the lottery and not writing code anyways)! So let's fix this little issue. When someone asks that question, simply respond with "Not sure, let's learn how you want it to work and I can give you a much better estimate."

Then sit with them and let them explain their vision for the feature. Write down the individual components of the feature (screens, reports, fields of data, important behaviors, fields of information to collect, etc.) Then ask them for a few days to come up with a prototype because you want to be sure that you are both thinking alike in what is going to be delivered. Once you deliver a prototype, lots of things will come out. Oh, we forgot this and that, what you are showing me missed the mark, I really like the way you decided to handle this and that, etc.

Then make changes to the prototypes to ensure that it accurately reflects what they are expecting. Drill into what they are expecting in terms of field validations (e.g. can we enter a date from the past, can the numbers on the screen contain commas and decimals, etc.). Ask if they want all data changes to be audited and if there is anything else they are not telling us that is important to them. Then finalize the prototype with all the assumptions and ask for their approval to create an estimate.

At this point you have a good set of requirements. If you are using Agile, this might be a compilation of user stories, but no matter what you call it, you have defined enough detail to provide a much better estimate. The next step is the most important, roll up your sleeves and figure out every task needed to complete this requirement.

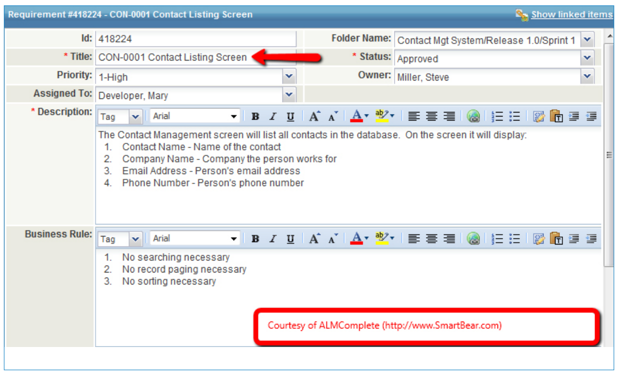
Let's take an example. Let's imagine that cool new feature the customer is asking for is a "bare bones" ability to track contacts (customers and sales prospects). During your discussion, you would ask "what does bare bones mean to you?" Does it mean that we don't need (at this time) to secure the data in any way? Does it mean that we just need the ability to view a list of existing contacts, add/edit/delete contacts? If the customer says yes, then dig deeper. Ask if you need auditing (keeping track of each field that changes, etc). Ask specifically what fields of information you want to collect for a contact (name, email, phone, etc) and if we need to be able to also send emails to each contact during this phase of the project. Ask what types of validations they need on a phone number, email address, how wide the fields need to be, are they required for data entry, etc. Ask if the list of contacts needs sorting or searching capabilities, etc.

Once done, let's imaging your clients says "No security and auditing in this phase, a standard HTML listing of contacts without searching and sorting is OK, a standard HTML data entry screen that allows entry of these fields (contact name, email, phone, etc) with these field validations (contact name is required and can be no more than 50 characters, etc), and the ability to edit or delete an existing contact.

In this process, let's imagine you prototyped 3 screens:

- Contact Listing – Shows a list of contacts
- Contact Add – Allows adding of a contact
- Contact Edit – Allows editing or deleting a contact

From here, you can create 3 requirements from this, noticed how we described each requirement and how we numbered them (CON-0001, CON-0002 and CON-0003):



Requirement #418224 - CON-0001 Contact Listing Screen

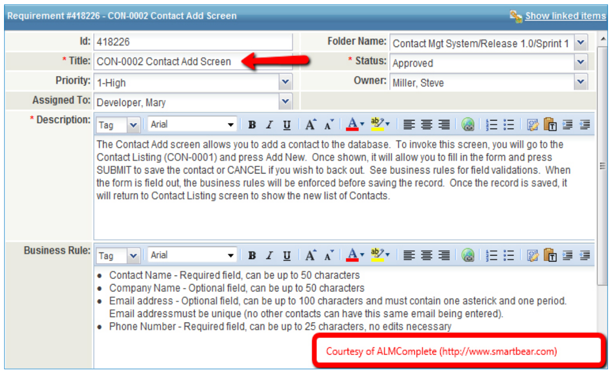
Id:	418224	Folder Name:	Contact Mgt System/Release 1.0/Sprint 1
* Title:	CON-0001 Contact Listing Screen	* Status:	Approved
Priority:	1-High	Owner:	Miller, Steve
Assigned To:	Developer, Mary		

* Description:

The Contact Management screen will list all contacts in the database. On the screen it will display:
1. Contact Name - Name of the contact
2. Company Name - Company the person works for
3. Email Address - Person's email address
4. Phone Number - Person's phone number

Business Rule:
1. No searching necessary
2. No record paging necessary
3. No sorting necessary

Courtesy of ALMComplete (http://www.SmartBear.com)



Requirement #418226 - CON-0002 Contact Add Screen

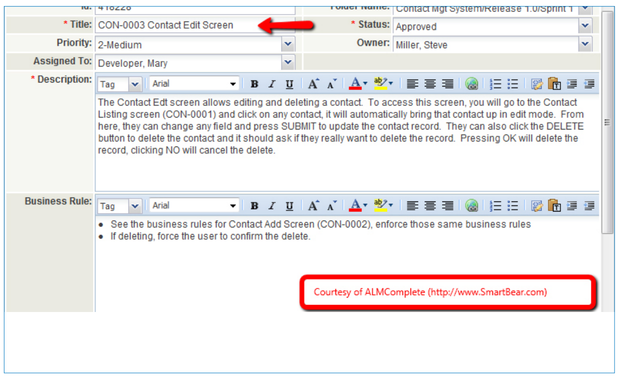
Id:	418226	Folder Name:	Contact Mgt System/Release 1.0/Sprint 1
* Title:	CON-0002 Contact Add Screen	* Status:	Approved
Priority:	1-High	Owner:	Miller, Steve
Assigned To:	Developer, Mary		

* Description:

The Contact Add screen allows you to add a contact to the database. To invoke this screen, you will go to the Contact Listing (CON-0001) and press Add New. Once shown, it will allow you to fill in the form and press SUBMIT to save the contact or CANCEL if you wish to back out. See business rules for field validations. When the form is field out, the business rules will be enforced before saving the record. Once the record is saved, it will return to Contact Listing screen to show the new list of Contacts.

Business Rule:
- Contact Name - Required field, can be up to 50 characters
- Company Name - Optional field, can be up to 50 characters
- Email address - Optional field, can be up to 100 characters and must contain one asterick and one period. Email addressmust be unique (no other contacts can have this same email being entered).
- Phone Number - Required field, can be up to 25 characters, no edits necessary

Courtesy of ALMComplete (http://www.smartbear.com)



Id:	418228	Folder Name:	Contact Mgt System/Release 1.0/Sprint 1
* Title:	CON-0003 Contact Edit Screen	* Status:	Approved
Priority:	2-Medium	Owner:	Miller, Steve
Assigned To:	Developer, Mary		

* Description:

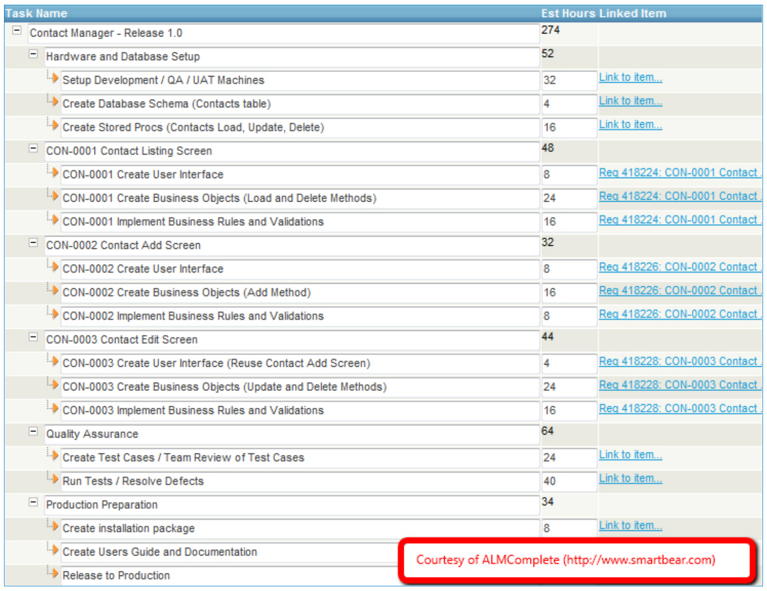
The Contact Edt screen allows editing and deleting a contact. To access this screen, you will go to the Contact Listing screen (CON-0001) and click on any contact, it will automatically bring that contact up in edit mode. From here, they can change any field and press SUBMIT to update the contact record. They can also click the DELETE button to delete the contact and it should ask if they really want to delete the record. Pressing OK will delete the record, clicking NO will cancel the delete.

Business Rule:
- See the business rules for Contact Add Screen (CON-0002), enforce those same business rules
- If deleting, force the user to confirm the delete.

Courtesy of ALMComplete (http://www.SmartBear.com)

Once you have defined your requirements, you can then decompose each one into more specific tasks – allowing you to provide a much better estimate:

Now that we have figured out all the detailed tasks and estimated each one, we now have a much better estimate than an off-the-cuff estimate, so we will come much closer to our estimate.



Task Name	Est Hours	Linked Item
Contact Manager - Release 1.0	274	
Hardware and Database Setup	52	
Setup Development / QA / UAT Machines	32	Link to item...
Create Database Schema (Contacts table)	4	Link to item...
Create Stored Procs (Contacts Load, Update, Delete)	16	Link to item...
CON-0001 Contact Listing Screen	48	
CON-0001 Create User Interface	8	Req 418224: CON-0001 Contact ...
CON-0001 Create Business Objects (Load and Delete Methods)	24	Req 418224: CON-0001 Contact ...
CON-0001 Implement Business Rules and Validations	16	Req 418224: CON-0001 Contact ...
CON-0002 Contact Add Screen	32	
CON-0002 Create User Interface	8	Req 418226: CON-0002 Contact ...
CON-0002 Create Business Objects (Add Method)	16	Req 418226: CON-0002 Contact ...
CON-0002 Implement Business Rules and Validations	8	Req 418226: CON-0002 Contact ...
CON-0003 Contact Edit Screen	44	
CON-0003 Create User Interface (Reuse Contact Add Screen)	4	Req 418228: CON-0003 Contact ...
CON-0003 Create Business Objects (Update and Delete Methods)	24	Req 418228: CON-0003 Contact ...
CON-0003 Implement Business Rules and Validations	16	Req 418228: CON-0003 Contact ...
Quality Assurance	64	
Create Test Cases / Team Review of Test Cases	24	Link to item...
Run Tests / Resolve Defects	40	Link to item...
Production Preparation	34	
Create installation package	8	Link to item...
Create Users Guide and Documentation		
Release to Production		

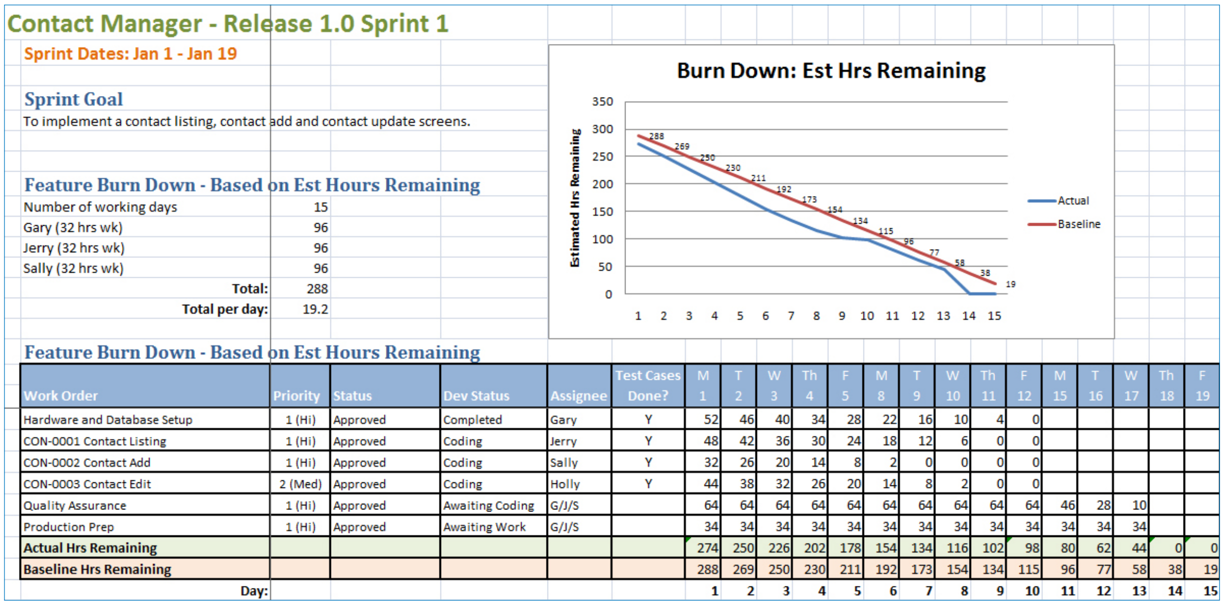
Courtesy of ALMComplete (http://www.smartbear.com)

Thorny Issue #5: Is my iteration going to complete on time?

So you've been a good student, you've collected adequate requirements and/or user stories, decomposed your requirements into lower level tasks so that you could provide a much better estimate. You've also decided to use a "time box" approach to development where you have set the begin and end dates of the project based on the number of people and hours committed to the project and built in a little buffer for things that go wrong. Well done my friend!

So the project starts and each day that passes you get more nervous that things are not progressing as they should. You start to freak out. How can you know whether you are going to get done on time? Piece of cake! It's called project burn down, and here is how it works.

First, make sure your time box dates are reasonable. Let's imagine that we are developing the mythical "Contact Management Software" and we figured out that we have estimated the effort at 274 hours of work. You know that each team member can contribute 32 hours of work a week (80 % of the work week), so you set your time box to 9 person weeks (274 / 32 = 8.56 weeks with a single person so you round up to 9 weeks). But you have 3 team members that are going to work on this, so you have time boxed it at 3 weeks (9 weeks / 3 people). So let's imagine your time box begins on 1-Jan and ends on 19-Jan. The way to determine if you will get done on time is to have your team log time towards each assigned task daily and then re-estimate the effort remaining daily. By re-estimating remaining effort daily, you can quickly see if you are on or off track. Here is how it might look if you are collecting this type of information via a spreadsheet:



Contact Manager - Release 1.0 Sprint 1

Sprint Dates: Jan 1 - Jan 19

Sprint Goal

To implement a contact listing, contact add and contact update screens.

Feature Burn Down - Based on Est Hours Remaining

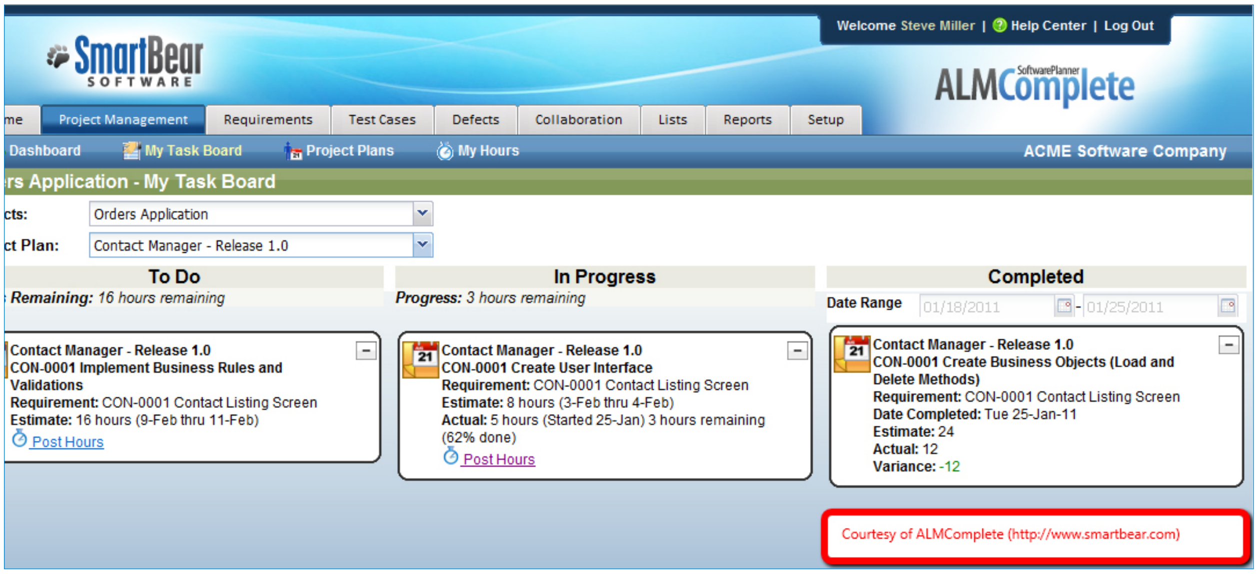
Number of working days	15
Gary (32 hrs wk)	96
Jerry (32 hrs wk)	96
Sally (32 hrs wk)	96
Total:	288
Total per day:	19.2

Feature Burn Down - Based on Est Hours Remaining

Work Order	Priority	Status	Dev Status	Assignee	Test Cases Done?	M 1	T 2	W 3	Th 4	F 5	M 8	T 9	W 10	Th 11	F 12	M 15	T 16	W 17	Th 18	F 19
Hardware and Database Setup	1 (Hi)	Approved	Completed	Gary	Y	52	46	40	34	28	22	16	10	4	0					
CON-0001 Contact Listing	1 (Hi)	Approved	Coding	Jerry	Y	48	42	36	30	24	18	12	6	0	0					
CON-0002 Contact Add	1 (Hi)	Approved	Coding	Sally	Y	32	26	20	14	8	2	0	0	0	0					
CON-0003 Contact Edit	2 (Med)	Approved	Coding	Holly	Y	44	38	32	26	20	14	8	2	0	0					
Quality Assurance	1 (Hi)	Approved	Awaiting Coding	G/J/S		64	64	64	64	64	64	64	64	64	64	46	28	10		
Production Prep	1 (Hi)	Approved	Awaiting Work	G/J/S		34	34	34	34	34	34	34	34	34	34	34	34	34		
Actual Hrs Remaining						274	250	226	202	178	154	134	116	102	98	80	62	44	0	0
Baseline Hrs Remaining						288	269	250	230	211	192	173	154	134	115	96	77	58	38	19
Day:						**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**

So each day you are recording how much effort the team members are saying they have remaining to do for each task. We also created a burn down graph that shows daily progress against plan, so in the graph, if the actual line (blue line) starts to creep above the red line (baseline), you are slipping and you should either have your team work more hours, pull back on any non-critical requirements, or get more team members if you want to stay on time with the delivery. Using a spreadsheet for this is not really optimal because it means that someone on the team (normally the scrum master or project manager) has to keep this spreadsheet updated daily.

If you are using an Application Lifecycle Management (ALM) tool, it will allow the individual team members to log time towards their tasks and re-estimate the effort using a task board. The task board shows what they have to-do, what is in progress and what has been completed. They can also log hours to each assigned task very easily:



rs Application - My Task Board

cts: Orders Application

ct Plan: Contact Manager - Release 1.0

To Do
Remaining: *16 hours remaining*

Contact Manager - Release 1.0
CON-0001 Implement Business Rules and Validations
Requirement: CON-0001 Contact Listing Screen
Estimate: 16 hours (9-Feb thru 11-Feb)
Post Hours

In Progress
Progress: *3 hours remaining*

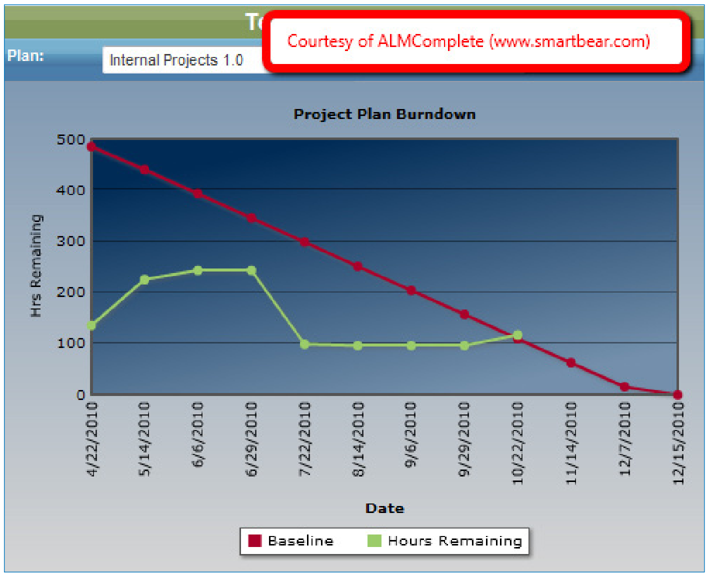
Contact Manager - Release 1.0
CON-0001 Create User Interface
Requirement: CON-0001 Contact Listing Screen
Estimate: 8 hours (3-Feb thru 4-Feb)
Actual: 5 hours (Started 25-Jan) 3 hours remaining (62% done)
Post Hours

Completed
Date Range 01/18/2011 - 01/25/2011

Contact Manager - Release 1.0
CON-0001 Create Business Objects (Load and Delete Methods)
Requirement: CON-0001 Contact Listing Screen
Date Completed: Tue 25-Jan-11
Estimate: 24
Actual: 12
Variance: -12

Courtesy of ALMComplete (http://www.smartbear.com)

And a good ALM tool can create a burn down chart for you automatically, here is an example (based on a different project with different deadlines):



To summarize, you will want your team to log time daily, re-estimate remaining effort daily and you will want to view this information on a burn down chart to determine if you are on track.

Thorny Issue #6: What do you mean everything cannot be Top Priority?

Ever work stints of really long days and feel like you are getting a lot of things done but just not making real meaningful progress? Ever shipped a software product with a huge number of new features but not all of them were fully tested and you spent the next month fighting fires to solve production issues? Well, if you have, you're not alone. Most of us have faced these issues and some of us are still struggling with them.

I've found that this is normally an issue of poor prioritization. Not everything can be a priority 1 item so it's important to realize this and more objectively set priorities when working on software development projects. The opportunity to solve this problem manifests itself in a lot of areas, including requirements, test, and defect planning.

Requirements Planning

When planning out an iteration or sprint, objectively prioritize each requirement. Not all requirements should be placed as priority 1; you have to have some that are lower priority. By doing this, it allows you to work on the highest priority items first.

During the development phase of an iteration or sprint, you can rest assured that surprises will come up. It may be that some things took longer than expected, someone gets pulled off the project for a few days to fight an emergency issue, or if you are lucky, things moved faster than expected.

Without proper priorities, you may have the team working on some things that are less critical than others and you may end up not getting the critical requirements completed while finishing requirements that are not so critical to have.

To fix this, you must first define objective (as opposed to subjective) priorities. Some teams simply use P1, P2, P3, and P4 for their priorities. These are very subjective because when assigning a priority, you really don't take the time to really consider the choices that you are making. Here are better priorities:

- 1-Critical: Anything with this priority has to be done in this release, or you will not publish the release.
- 2-Nice to Have: Anything with this priority would add value to the release but it is not absolutely critical.
- 3-If Time: Anything with this priority is pure gravy.

Test Case Planning

When you are planning out your test approach, the goal is always to fully test each requirement in the iteration or sprint. The depth of testing can vary, but it is key that each feature works per the design and you want it to be as bug free as possible while understanding that not every esoteric condition can be tested in a normal testing cycle so we want to really focus on the issues that are most likely to show up in production.

To objectively decide which test cases will bring the most value and reduce the largest number of bugs in production, I like to prioritize test cases this way:

- 1-Smoke Test: These are a small number of test cases that test the basics of the features and should be concentrated on first. For example, if you were building a contact management system and the requirement you are testing is to "add a contact", a smoke test would include a test case that ensures that a test case can be added with all fields of data filled out with valid data (do not try anything fancy to try to trick it). If adding a contact should also write an audit record, you would include a test case to ensure that the audit record was also written upon add.

- 2-Basic Tests: Using the "add a contact" feature example, these would be test cases that would include field validations:
  1. Required fields - Ensure that all fields that require data entry force you to enter data on the screen.
  2. Field widths – Ensure that each field on the screen allows you to enter no more than the allowed char acters. For example, if the database only allows 100 characters for an email address, ensure that the person cannot enter 101 characters. Do this for all fields.
  3. Value Validations – If you have any fields that require only a select set of values for data entry, ensure that the user interface only allows those valid values. For example, if you have a contact status that can only be new, assigned, and closed, test that no other values are allowed.
- 3-Negative Tests: These test cases are for safeguarding against the user making a mistake when entering data. For example, your "contact add" screen may have a "call back date" on the form. Ensure that only a valid date is allowed. If it has a field called "email", ensure that the email address contains a single ampersand and at least one period.
- 4-Deeper Tests: If you are able to get through all of the tests above, try to test more deeply. Deeper tests could include performance testing (to ensure it works well under a large load) and referential integrity tests (for example, delete a contact record; ensure that it deletes all associated contact notes, etc.).

Defect Planning

During your testing cycle, you are bound to find defects and plenty of them. However, not every defect is created equally. Some are really serious and need immediate attention while others are minor annoyances, or can only be repeated in very rare cases. It is important that you are not spending precious cycles on less critical defects if you are in any type of time pressure to finish up the release – you need to stay focused on the stuff that can really bite you in production. The best way to ensure you are spending your time wisely is to categorize the defects by how severe they are and work on them in that order. Here are some suggested severity/priorities:

- 1-Crash: This defect crashes the system and should be fixed immediately!
- 2-No Workaround: This is a major defect with no workaround. Fix these next.
- 3-Workaround: This is a defect that can easily be resolved via a workaround.
- 4-Trivial: This defect should be fixed only if we have time.

Thorny Issue #7: How can I speed up the QA Process?

When you get into the QA phase of software development and you begin to fall behind on your testing, a typical question is how you can make this process faster. Most people treat this issue by strapping a rocket to the butts of the testers (throwing more resources on the testing or by having people work 12 to 15 hour days). While this might work to get past your immediate issue, it really is not a scalable and reliable way to speed up the QA process in the long term. This approach leads to team burnout.

The better way to attack this issue is upfront planning. To attack this problem, you need to understand the key things that cost you the most time during QA. They are:

- Poor Test Design and Priorities
- Too many defect re-work cycles
- Excessively Buggy code

Let's discuss each of these issues and how to resolve them.

Poor Test Design and Priorities

One time drain is creating test cases that don't adequately test the feature(s) or are not prioritized in a way that ensures that you test the most critical ones first. Luckily, this is a snap to fix. The best way to overcome this issue is to have your QA team get started on the test cases as soon as the requirement has been developed, very early in the coding phase, and tie the test cases back to the requirements to ensure you have enough test cases to fully test each requirement.

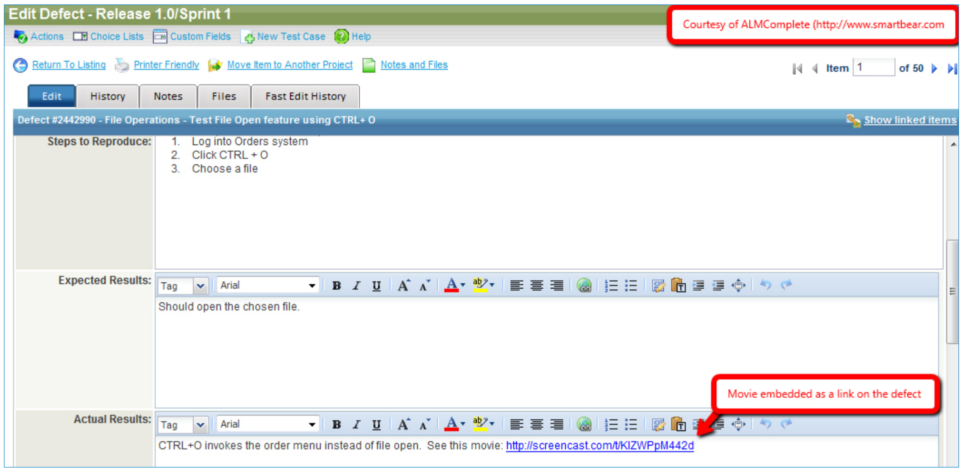
The next step is the most important, publish your test cases to the developers and ask for the team to review them. This gives powerful information – having developer feedback usually yields new test cases that were not thought about or test cases that are not quite correct (because of misunderstanding of the feature). It also gives the developer keen insight into what tests the QA engineer is going to be running and this naturally leads them to test those scenarios so that fewer issues will be found when it hits the QA team. Also talk with the developers about which of the test cases are the highest priority, so that you prioritize them so that you get through the most effective ones first.

Too many Defect Re-Work Cycles

Ever report a defect, the programmer says he or she cannot reproduce it, you add more details, they still cannot reproduce it and you go back and forth for a few days before they actually figure it out. What a time drain. The core issue is that we are not always good about specifying exactly what steps we went through to cause the issue.

A much better approach is to record a movie that shows you how you caused the issue. An easy way to do that is to use a tool like Jing (http://www.jingproject.com). Jing is a free tool that allows you to capture the area of your screen you want to record, turn recording on, and it will record all your keystrokes, screen navigation, and it allows you to narrate to the end user what is happening as you go along. Then it allows you to publish that movie into Screencast, which is then visible from a URL.

Many ALM tools will allow you then paste the URL of the movie into the defect so that the developer can click it to view the movie:



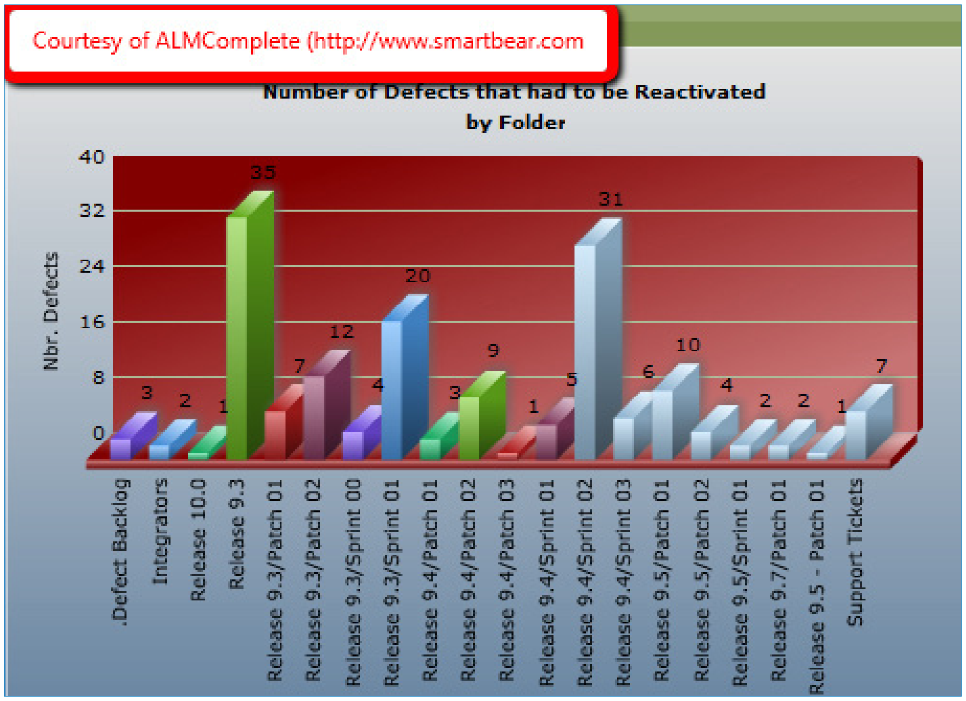
Defect #2442990 - File Operations - Test File Open feature using CTRL+ O

Steps to Reproduce:
1. Log into Orders system
2. Click CTRL + O
3. Choose a file

Expected Results: Should open the chosen file.

Actual Results: CTRL+O invokes the order menu instead of file open. See this movie: http://screencast.com/t/KIZWPpM442d

The next thing you need to do is to track how many times defects are re-worked in each release. This brings visibility to the issue and allows your team to be more conscious of writing up good defect reports and providing easy ways to see how the defect happened. Here is an example of a dashboard that shows how many defect re-works were done during recent releases, this helps you track it.



Excessively Buggy Code

There are many faces to buggy code. Buggy code could be code that does not fully implement the features of a requirement, poor error trapping that cause crashes, weak error messages, and just plain bad coding. Here are some ways to combat these issues:

- Coding Standards – Publish coding standards that outline naming conventions, error trapping standards, and other programming standards. Publish these standards to ensure everyone knows what is expected.
- Show and Tell – As the developer begins coding, have them do "show and tell" sessions every week or so that shows the progress of what they have completed thus far. These "show and tell" sessions allow them to present to the team how they have progressed on their programming. It can flesh out misunderstandings and also puts a little more pressure of the developer to have tighter code because they don't like seeing their programs crash during "show and tell"!
- Code Reviews – Many coding issues can be caught during the development phase by simply performing periodic code reviews. The earlier you can find these issues, the less impact you have on the timeline of your QA cycle. Code Reviews can be simply over-the-shoulder reviews, where you sit next to the developer, looking for poor coding, bad error trapping, etc. Or if you want to really collaborate during the code review process, consider tools like CodeCollaborator, it allows you to review and annotate code and keeps track of all team collaboration during this process.

Quality Management - Common Thorny Situations

Below are some common problems software development teams face in terms of delivering high quality software and ways I've found to combat them.

Thorny Issue #8: How do I know if I have adequate Test Coverage for each Requirement?

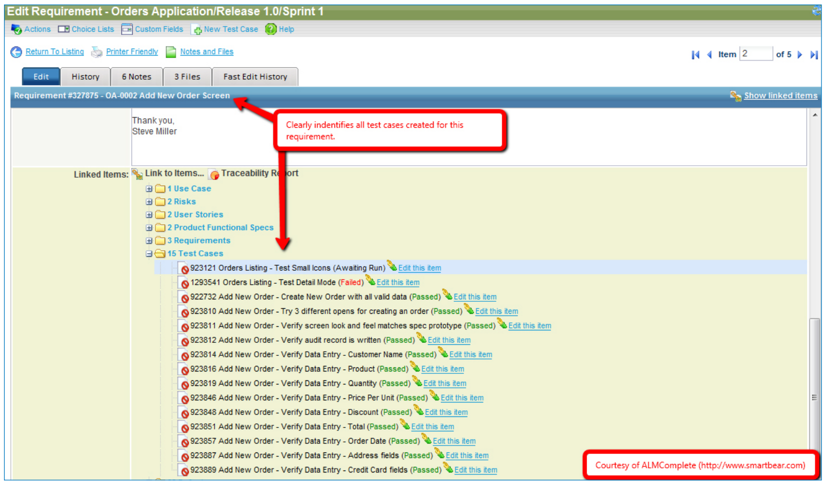
So you've been a good IT shop and started your software release by developing a solid set of requirements, began writing test cases and have launched into the coding phase. But how can you tell if you are going to fully test each requirement?

First, create a traceability matrix that identifies the different types of tests for each requirement. The idea is to have enough test cases that are going test each requirement in different scenarios:

- Smoke Tests – Create a set of test cases that test the basic functionality of the requirement and if these basic tests fail upon receiving the new build, cease testing and send it back to the development team to tighten up.
- Positive Tests – Create a set of test cases that test the requirement under normal conditions. These are things like testing that it writes the correct records, allows valid data input, has no grammar or spelling errors, etc.
- Negative Tests – These are tests to ensure that it works under irregular conditions. For examples, test date fields for valid dates, test fields on the screen to ensure you cannot add more characters on a specific field than is allowed in the database, etc.
- Regression Test – If the requirement is an enhancement to existing features, have tests that ensure that the new requirement does not break existing features.
- Performance Tests – These tests ensure that the software will work with reasonable performance.

For a more in-depth review of these techniques, see this whitepaper: http://www.softwareplanner.com/TestBest-Practices.pdf.

If your ALM tool allows you to link test cases to requirements, that is a quick way to get an overview of the tests you have allocated for each requirement:



Requirement #327875 - OA-0002 Add New Order Screen

Thank you,
Steve Miller

Clearly indentifies all test cases created for this requirement.

Linked Items:
- 1 Use Case
- 2 Risks
- 2 User Stories
- 2 Product Functional Specs
- 3 Requirements
- 15 Test Cases
  - 923121 Orders Listing - Test Small Icons (Awaiting Run) Edit this item
  - 1293541 Orders Listing - Test Detail Mode (Failed) Edit this item
  - 922732 Add New Order - Create New Order with all valid data (Passed) Edit this item
  - 923810 Add New Order - Try 3 different opens for creating an order (Passed) Edit this item
  - 923811 Add New Order - Verify screen look and feel matches spec prototype (Passed) Edit this item
  - 923812 Add New Order - Verify audit record is written (Passed) Edit this item
  - 923814 Add New Order - Verify Data Entry - Customer Name (Passed) Edit this item
  - 923816 Add New Order - Verify Data Entry - Product (Passed) Edit this item
  - 923819 Add New Order - Verify Data Entry - Quantity (Passed) Edit this item
  - 923846 Add New Order - Verify Data Entry - Price Per Unit (Passed) Edit this item
  - 923848 Add New Order - Verify Data Entry - Discount (Passed) Edit this item
  - 923851 Add New Order - Verify Data Entry - Total (Passed) Edit this item
  - 923857 Add New Order - Verify Data Entry - Order Date (Passed) Edit this item
  - 923887 Add New Order - Verify Data Entry - Address fields (Passed) Edit this item
  - 923889 Add New Order - Verify Data Entry - Credit Card fields (Passed) Edit this item

Courtesy of ALMComplete (http://www.smartbear.com)

Thorny Issue #9: How can I determine if my testing is progressing properly?

Once you begin testing, it is important to know how you are progressing each day during the QA cycle. Without knowing how many need to be run, how many passed and failed each day, you will not be clear on whether your ship date is feasible.

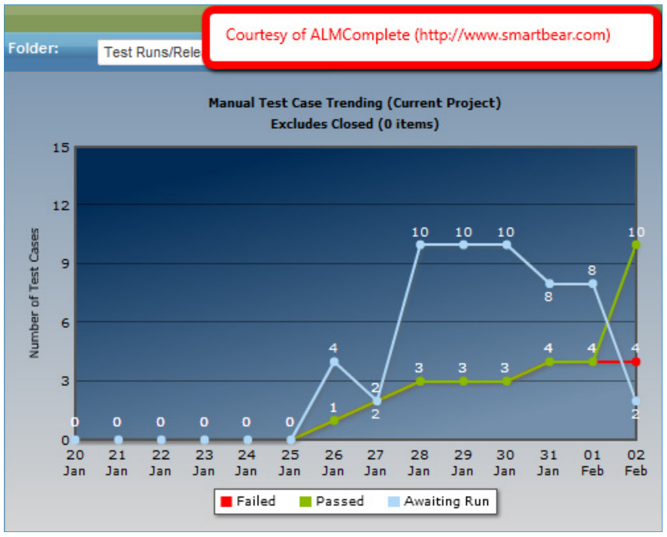
Before you start your QA cycle, you should have all your test cases defined, so you should know how many you plan to run during your QA cycle. Let's imagine that you have these requirements and these test cases (grouped by priority):

Requirement	# Smoke Tests	# Basic Tests	# Negative Tests	# Deep Tests	Total
CON-0001 Contact Listing Screen	5	20	10	5	40
CON-0002 Contact Add Screen	4	50	15	10	79
CON-0002 Contact Edit Screen	2	30	14	16	62
Total	11	100	39	31	181

In the example above, you have 181 test cases you would like to get done during your testing cycle.

However, since you have prioritized them, you will concentrate on Smoke Tests, then Basic Tests, then Negative Tests, then Deep Tests, so if you don't get through all the Deep Tests, it may be fine.

If you have 10 days of QA, you will need to get through roughly 18 test cases a day to get them all done. Now we do know that not all test cases are created equally, some take longer to run than others, so we need to keep track daily how we are trending. So each day make note as to how many have passed, how many have failed and how many are still awaiting run. Many ALM tools can provide this information in graphical format:



Thorny Issue #10: Why do I find myself rewriting the same code over and over again?

So let's say that you finish up the first iteration and things went well – you got into production in 2 months. But as you began working on your 2nd iteration, you found that you were writing very similar code as to what you had in the 1st iteration. Just because you are doing things in 1 or 2 month increments do not mean that you forget about applying solid engineering practices to the process –if you have not crossed this bridge yet, the second iteration is where you want to really start applying good architectural discipline to your work.

If you are like me, you hate doing the same work over and over again. It's all about re-use. Never code twice what you can code once. In my experience, the best way to re-use code is to separate the user interface, business, common utilities, and database logic into separate code modules. In most programming languages, this can easily be done with the use of object oriented class design. And you will also want to build a business layer than can be accessed through your application or can extend its use out to other programmers in the form of an API. This is called a Service Oriented Architecture and will save you tons of time in the future.

Confused about how to do it? Let's take an example. Let's say in our mythical Contact Management system that you took the time to write logic that would read a list of contacts (for the contact listing), added logic to validate the data entry on the screen when adding or editing a contact, and added logic to add or update the contact record to the database. If you placed all the logic in the user interface (logic for querying, validating data entry, and inserting or updating the contact record in the database), what happens when you get ready to build import routines later on (allowing you to import contacts into the database)? Well, you will be the writing code the same code again (e.g. must query the database, must validate the imported data, must insert or update the contact record).

Now let's imagine you had taken a more re-usable approach. Instead of putting that code in the user interface, you would have created a separate (business layer) class module for querying, validating, inserting and updating contact data. Then the user interface would simply call those methods and you will not have written the same code twice. This also reduces maintenance – imagine finding an error in your validation routines – make the change once in your business class module and you are done. Heck, you did so well, take the day off…

Now let's really make our brains hurt. Let's also imagine that you envision other companies wanting to write code against your Contact Management software that can query, validate, insert or update contact records because they have related data in their software product that could nicely integrate with what you have done. By simply turning that re-usable business layer into a web service (very easy to do), you can now expose those methods to other companies (in the form of a web services API) and not only can your team use the business layer, other companies can too. The coolness meter just went through the roof!

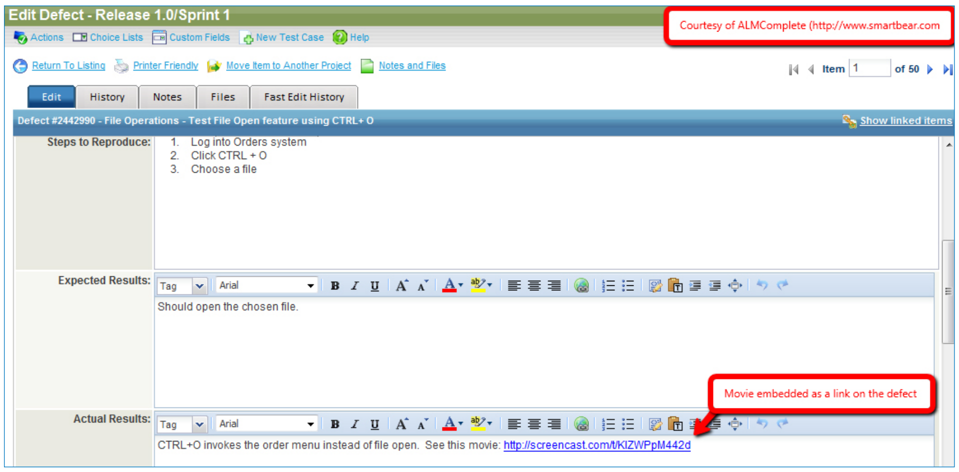
Bottom line: Separate your logic into 3 physical tiers (user interface, business layer – which would be a web service, and database layer), then you'll be a rock star. Defining a good architecture is an essential part of this effort.

Thorny Issue #11: Why are my developers taking so long to fix defects?

OK, let's pick on the developers now. Why does it take them so long to fix defects? Why do they keep pointing the finger back at the QA team, saying that defects are not reproducible? What we have here is a failure to communicate!

Most of these issues are simple to solve. Normally developers spend many cycles trying to reproduce issues which result in back and forth between tester and developer, all because they just are not quite sure how to reproduce what the tester saw. To solve this, have the tester create a movie that shows them exactly how to reproduce the issue. This also takes less time for the tester – instead of having to write out every single step to reproduce it, they can show them. A movie is worth a thousand words!

An easy way to do that is to use a tool like Jing (http://www.jingproject.com). Jing is a free tool that allows you to capture the area of your screen you want to record, turn recording on, and it will record all your keystrokes, screen navigation, and it allows you to narrate to the end user what is happeningas you go along. Then it allows you to publish that movie into Screencast, which is then visible from a URL. Many ALM tools will allow you then paste the URL of the movie into the defect so that the developer can click it to view the movie:



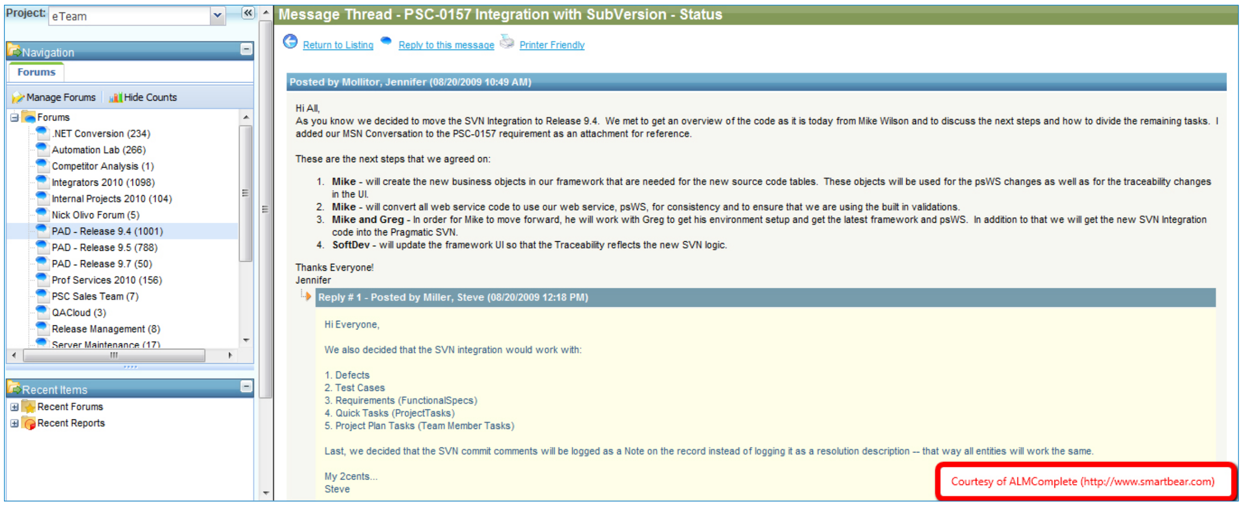
Defect #2442990 - File Operations - Test File Open feature using CTRL+ O

Steps to Reproduce:
1. Log into Orders system
2. Click CTRL + O
3. Choose a file

Expected Results: Should open the chosen file.

Actual Results: CTRL+O invokes the order menu instead of file open. See this movie: http://screencast.com/t/KIZWPpM442d

Another way to improve communication and speed up defect resolution is to use a discussion forum. In the forum, developers can post any questions they have about requirements, testers can talk about their test approach, everyone can report their overall status, etc. You can find free discussion forums on the web (Google has some) and some ALM tools have that feature built right in:



Message Thread - PSC-0157 Integration with SubVersion - Status

Posted by Mollitor, Jennifer (08/20/2009 10:49 AM)

Hi All,
As you know we decided to move the SVN Integration to Release 9.4. We met to get an overview of the code as it is today from Mike Wilson and to discuss the next steps and how to divide the remaining tasks. I added our MSN Conversation to the PSC-0157 requirement as an attachment for reference.

These are the next steps that we agreed on:

1. **Mike** - will create the new business objects in our framework that are needed for the new source code tables. These objects will be used for the psWS changes as well as for the traceability changes in the UI.
2. **Mike** - will convert all web service code to use our web service, psWS, for consistency and to ensure that we are using the built in validations.
3. **Mike and Greg** - In order for Mike to move forward, he will work with Greg to get his environment setup and get the latest framework and psWS. In addition to that we will get the new SVN Integration code into the Pragmatic SVN.
4. **SoftDev** - will update the framework UI so that the Traceability reflects the new SVN logic.

Thanks Everyone!
Jennifer

Reply # 1 - Posted by Miller, Steve (08/20/2009 12:18 PM)

Hi Everyone,

We also decided that the SVN integration would work with:

1. Defects
2. Test Cases
3. Requirements (FunctionalSpecs)
4. Quick Tasks (ProjectTasks)
5. Project Plan Tasks (Team Member Tasks)

Last, we decided that the SVN commit comments will be logged as a Note on the record instead of logging it as a resolution description -- that way all entities will work the same.

My 2cents...
Steve

Courtesy of ALMComplete (http://www.smartbear.com)

Finally, you can improve communication and remove roadblocks by doing daily triage meetings. This is a quick meeting (no more than 30 minutes) where you get a quick status of progress and you can discuss any defects that are still non reproducible, etc. By improving communication, defect resolution gets put on the fast track!

Thorny Issue #12: How can we compare developer output and effectiveness?

You mean you want developers to be both effective and get a lot done? Geez, that's a lot to ask! Before you can determine how effective a developer is, you first have to define what effective means to you. Let's imagine that you define effective as a developer who: a great mentor for both Jerry and Mary regarding software quality – he should share his secrets with them as well.

Thorny Issue #13: How do I evaluate overall quality once in Production?

So you got through the development and release of your product, you killed a few brain cells in the process and partied like it was 1999 (or should we say 2999) after you shipped the release - good for you! Post production success is kind of like opening up a present at Christmas (or Hanukah or whatever), you are not 100% sure what you are going to get until you open it.

You may be in store for a very busy couple of weeks that follow a release or it might just be smooth sailing. It is easy to get caught up in the frenzy and make snap judgments about what the quality level was once the release has been promoted into production. But you can easily do some analysis and determine if you were able to maintain an acceptable level of quality. The best way to analyze it is to keep track of all incoming issues, track the severity, the type of issues, and the number of hours it took to repair those issues.

A good ALM tool will have a support management component to it that allows your end clients to submit issues and track the status and will provide you with the ability to analyze it. By using that system, you can easily categorize the issues by severity, type (defect, documentation issue, client misunderstanding, etc.), and you can track the number of hours it took to resolve any issues.

To make your quality evaluation more concrete, you should evaluate those statistics each week after production and continue doing so until your next release. Once the next release goes into production you can then objectively quantify if you are getting better from release-to-release or worse. If you getting better – good for you! If you are getting worse – you gotta make some changes! Here is how your quality evaluation sheet might look:

Release	# Severity 1 & 2 Defects	# Severity 3 & 4 Defects	Total Defects	Hours Impact	Notes
Release 1.0	20	80	100	80	
Release 1.1	40	200	240	160	This release had issues!
Release 2.0	10	25	35	40	Improved quality!
Release 2.1	8	30	38	40	Kept quality improvement!

As we can see above, we really were having some quality issues in Release 1.0 and 1.1 but we made improvements in Release 2.0. We also sustained those improvements in Release 2.1, so it appears that our process improvements are sticking. The analytics above is simply one way to evaluate overall quality, feel free to hook into whatever metrics work best for what you are trying to accomplish.

Thorny Issue #14: How can I reduce my workload and go home earlier every day?

What do you mean? You don't like working 18 hour days, having your manager call you after hours and on the weekends and having your significant other badger you about never seeing you? How could you not like that?

No matter if you are a project manager or scrum master, developer or tester, there are ways to reduce your workload and get out of the office earlier each day, and to keep your sanity. OK, so let's talk about some common sense ways to attack this problem for each type of employee role.

Project Manager or Scrum Master

As a project manager, you are the task master. As a scrum master, you are the person who needs to help remove roadblocks that keep people from being the most productive they can be. Both roles work in a very similar way, you are ultimately responsible for the release or iteration.

Most stress and overworking at this level materialize because of poor team communication and cohesion and not keeping your eye on the ball. This is actually pretty easy to fix. Hold daily meetings with your team (in the Agile world we call these Daily Scrum meetings). Keep the meetings between 15 to 30 minutes. Prior to the meeting, the project manager / scrum master should be well prepared. They should have already looked at their release burn down to determine if the release is behind or ahead of schedule. They should also have looked at each team members tasks to see if any are slipping, and if so, if it will affect any other team members' work.

The project manager / scrum master should open up the meeting with anything they found that seems off track. If nothing is off track, just ask if anyone has any issues they want to discuss, and then end the meeting quickly so everyone can get back to work. Let’s imagine that the project manager / scrum master has identified some slippage. They may say “I looked at the burn down today and it appears that we are off track. Joe – I noticed you have 2 tasks that are 8 hours behind schedule and your late delivery is going to impact Mary. What can we do to help resolve that?”

Then each team member can quickly talk about any roadblocks they are having the team as a whole can help them find solutions (others might pitch in or design details may change, etc.). But by having this very transparent process, it flushes out problems before they compound and cause missed deadlines. This also provides an open atmosphere where teams can communicate and build trust and cohesion that will reduce overall workload.

Developer

Developers are overworked for a number of reasons. First, many underestimate their tasks and “code like hell” to get the work done. That can be easily fixed by taking each requirement they are working on and decomposing it down into individual tasks – this always leads to a better estimate. They can also keep track, from release-to-release, how closely they come to their estimates and begin building estimate buffers when doing future estimates. Eventually their estimating skills will improve and this issue will be solved.

A second reason for overwork is developers do not reuse code regularly enough. The team should keep a reusable set of function libraries that the team can share. How many times have you seen developers rewrite code that figures out leap years, does MOD 5 calculations, formats dates and times in a specific format, etc.? By identifying code that can be commonly reused by team members, you can dramatically decrease the amount of time (and increase the quality) of code written.

A third reason for overwork is that developers often spend too little time unit testing their code before shipping it off to the QA team. By doing this, testers and developers start playing the game of: identity issue > need more steps to reproduce > fix issue > issue still not quite resolved > fix again > OK, now it is resolved.

Do yourself a favor; spend more time unit testing than you do coding. It will dramatically reduce the back and forth defect resolution marathon that just sucks time from you and gets you home late each night. If you want to deliver higher quality code, have your testers publish their test cases to you before coding begins so that you can run through those tests yourself during unit testing – that will dramatically reduce the defect marathon, and it will save you from going over and punching your testers when they are relentlessly sending you defect reports!

A final thing that can help reduce work is by simply doing code reviews. Simple concept – when you are coding, sometimes you can’t see the forest for the trees. It is so easy to get so heads down in the coding process that obvious issues just are not clear to you. By having someone sit over your shoulder and review your code, it will quickly identify logic errors, poor error trapping, maintainability problems, ineffectual re-use, and many other common issues that will materialize during QA and takes valuable time away.

Testers

As a tester, you can easily become overworked especially during the QA phase. Many testers run into this issue because of improper planning before QA starts. As a tester, you should begin developing your test cases as soon as a requirement is approved – and before (or during) coding. When you develop your test cases, be sure to tie your test cases back to a requirement and have plenty of positive, negative, regression, and performance test cases defined to ensure you get good test coverage. As you are doing this, prioritize each test case, that way when you get hot and heavy into testing, you can run the tests in priority order. So if you get squeezed on time, run the really important ones and spend less time on the less important ones.

Another thing that takes time away from testers is that they don't always re-use test cases from release to release. Instead, they keep the test cases in a spreadsheet that gets lost or does not have good requirement linkage and they can't really easily find the ones they need so they just start re-creating them. What a time drain! Use a test management tool to organize your tests (for a free tool, see http://smartbear.com/products/free-tools/qaplanner/). In your tool of choice, organize your test library by functional area so that when you are planning out your next release, you can easily find test cases you have written in the past that test some functionality, and either use the existing ones or start to enhance them to test any new behaviors.

After a tester has developed their test cases for a requirement, it is also a good idea to have a review of those test cases with the developer that is doing the coding. By simply meeting to discuss the test cases, it will flush out missing test cases or test cases that are incorrect based on the requirement. This can save many hours of bogus defects that can be reported and reworked for no reason. This also helps the developer because they will know what tests are going to be run and will be more likely to better unit test those areas, especially if they are trying to reduce their individual QA defect count!

Finally, many testers manually test the same regression test cases over and over for each release. Once the system under test becomes more complex and feature rich, it can take a lot of time to run through a full regression by manually testing it. I have seen some systems take a single tester 2 weeks to fully regression test (with manual methods). If your regression suite takes more than 4 hours to fully complete, it is time to switch from manual to automated testing for your regression suite. By automating the regression tests, you can have software run through the regression tests while you concentrate on testing the new features of the software. This also allows you to run those tests daily, giving many more regression test cycles than you could accomplish manually.

When it comes to automating your tests, you may be lured into thinking that the same person can do manual and automated test creation. While it is possible, this can really add to the issue of working too many hours. It is best to hire a good automation test engineer whose sole purpose in life is creating and maintaining your automated test library. For more information on automated testing, see this link: http://www.testcomplete.com.

Thorny Issue #15: How can my team grow professionally and learn from our mistakes?

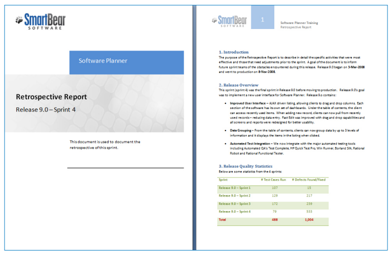
OK, so you make a new year's resolution to improve and grow professionally and decide to take the first step. As the old adage goes, you can't improve what you can't measure. If you really want to improve your software development and QA process and you want to have each of your team members grow professionally, you must evaluate each release or iteration and make adjustments in the next release or iteration.

Many releases or sprints encounter problems that must be corrected and some go smoother than planned. Regardless of how successful or disastrous a release or sprint is, it is important to review the release or sprint in detail once it is over. This allows your team to figure out what things were done well and to document the things that need improvement. It also aids in building a knowledge base that teams coming after you can review to ensure they get the most out of their upcoming projects. The key to future successful sprints is to learn from past mistakes. The process of formally reviewing your sprint is called a Retrospective.

Below are the steps for conducting successful Retrospectives:

1. Plan Your Retrospective - Upon completion of a sprint, the team should conduct a Retrospective. This is where the Scrum Master invites all the major players of the team (Product Owner, Team Members, Software Quality Engineers, etc.) to a meeting to review the successes and failures of the sprint.
2. Require Team Participation - Ask the attendees to bring a list of 2 items that were done well during the sprint and 2 things that could be improved upon.
3. Hold the Retrospective Meeting - Go around the table and have each person to discuss the 4 items they brought to the meeting. Keep track of how many duplicate items you get from each team member. At the end of the round table discussion of items, you should have a count of the most common items that were done well and the most agreed upon items that need improvement. Discuss the top success items and the top items that need improvement.
4. List Items Done Well and Things Needing Improvement - Upon listing of the success and improvement items, discuss specific things that can be done to avoid the items that need improvement upon the next release. If some items need more investigation, assign specific individuals to finding solutions.
5. Create a Retrospective Report - The best way to keep this information organized is to create a "Retrospec tive" report, where you document your findings. Send the Retrospective report to all team members. Before team members embark on their next sprint, make sure they review the Retrospective report from the prior project to gain insight from the prior project.

A good Retrospective Report might look similar to this



SmartBear SOFTWARE

Software Planner

Retrospective Report

Release 9.0 – Sprint 4

This document is used to document the retrospective of this sprint.

SmartBear SOFTWARE 1 Software Planner Training Retrospective Report

1. Introduction

The purpose of the Retrospective Report is to describe in detail the specific activities that were most effective and those that need adjustments prior to the sprint. A goal of the document is to inform future sprint teams of the obstacles encountered during this release. Release 9.0 began on 3-Mar-2008 and went to production on 8-Nov-2008.

2. Release Overview

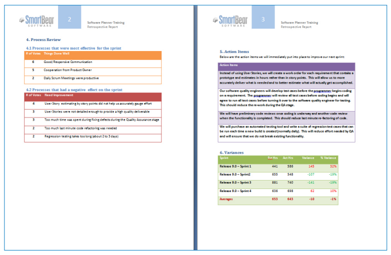
This sprint (sprint 4) was the final sprint in Release 9.0 before moving to production. Release 9.0's goal was to implement a new user interface for Software Planner. Release 9.0 contains:

- **Improved User Interface** – AJAX driven listing, allowing clients to drag and drop columns. Each section of the software has its own set of dashboards. Under the table of contents, the client can access recently used items. When adding new record, clients can now pull from recently used records – reducing data entry. Fast Edit was improved with drag and drop capabilities and all screens and reports were redesigned for better usability.
- **Data Grouping** – From the table of contents, clients can now group data by up to 3 levels of information and it displays the items in the listing when clicked.
- **Automated Test Integration** – We now integrate with the major automated testing tools including Automated QA's Test Complete, HP Quick Test Pro, Win Runner, Borland Silk, Rational Robot and Rational Functional Tester.

3. Release Quality Statistics

Below are some statistics from the 4 sprints:

Sprint	# Test Cases Run	# Defects Found/Fixed
Release 9.0 – Sprint 1	107	15
Release 9.0 – Sprint 2	129	217
Release 9.0 – Sprint 3	172	239
Release 9.0 – Sprint 4	79	533
Total	488	1,004



SmartBear SOFTWARE 2 Software Planner Training Retrospective Report

4. Process Review

4.1 Processes that were most effective for the sprint

# of Votes	Things Done Well
6	Good/Responsive Communication
5	Cooperation from Product Owner
2	Daily Scrum Meetings were productive

4.2 Processes that had a negative effect on the sprint

# of Votes	Need Improvement
4	User Story estimating by story points did not help us accurately gauge effort
3	User Stories were not detailed enough to provide a high quality deliverable
3	Too much time was spent during fixing defects during the Quality Assurance stage
2	Too much last minute code refactoring was needed
2	Regression testing takes too long (about 2 to 3 days)

SmartBear SOFTWARE 3 Software Planner Training Retrospective Report

5. Action Items

Below are the action items we will immediately put into place to improve our next sprint:

Action Items
Instead of using User Stories, we will create a work order for each requirement that contains a prototype and estimates in hours rather than in story points. This will allow us to more accurately deliver what is needed and to better estimate what will actually get accomplished.
Our software quality engineers will develop test cases before the programmer begins coding on a requirement. The programmer will review all test cases before coding begins and will agree to run all test cases before turning it over to the software quality engineer for testing. This should reduce the re-work during the QA stage.
We will have preliminary code reviews once coding is underway and another code review when the functionality is completed. This should reduce last minute re-factoring of code.
We will purchase an automated testing tool and write a suite of regression test cases that can be run each time a new build is created (normally daily). This will reduce effort needed by QA and will ensure that we do not break existing functionality.

6. Variances

Sprint	Est Hrs	Act Hrs	Variance	% Variance
Release 9.0 – Sprint 1	441	586	145	32%
Release 9.0 – Sprint2	655	548	-107	-19%
Release 9.0 – Sprint 3	881	740	-141	-19%
Release 9.0 – Sprint 4	636	698	62	10%
Averages	653	643	-10	-1%

It will contain an introduction (that explains what the goals and dates of the release or iteration were, a list of features delivered in the release, and the release quality statistics.

Then it would be followed with a process review (what you did right and wrong), and action items that you can carry into your next release or sprint to improve:

We created a template that you can use for the document, download it here:
http://www.softwareplanner.com/Template_Retrospective.doc

Learn More

SmartBear Software provides affordable tools for development teams that care about software quality and performance. Our collaboration, profiling and automated testing tools help more than 100,000 developers and testers build some of the best software applications and websites in the world.

- ALMComplete is a software solution for software development management and test management including requirements, test cases, defect tracking, tasks, and support tickets. Coupled with collaborative tools like document sharing, team calendars, interactive dashboards, burndown charts, knowledge bases and threaded discussions, teams begin communicating more effectively and begin delivering solutions quickly and with high quality.
- DevComplete is an ideal tool for software development management including project management, requirements management, and defect tracking. Create project plans, decompose requirements into project tasks and work item deliverables, establish end-to-end traceability between requirements, project tasks and defects, manage defect resolution, review resource availability, project burn downs, slipping tasks and much more through sophisticated dashboards and reports.
- QAComplete is a comprehensive application for test and QA management including requirements management, test case management, and defect tracking. Take a more strategic approach to testing by prioritizing key test functions, accounting for risk, planning for coverage, and controlling test execution. Full traceability among requirements, test cases and defects ensures complete test coverage through every stage of the software development process.

About SmartBear Software

SmartBear Software provides tools for over one million software professionals to build, test, and monitor some of the best software applications and websites anywhere – on the desktop, mobile and in the cloud. Our users can be found worldwide, in small businesses, Fortune 100 companies, and government agencies. Learn more about the SmartBear Quality Anywhere Platform, our award-winning tools, or join our active user community at www.smartbear.com, on Facebook, or follow us on Twitter @smartbear.