Introducing Resilient Agile – A Better Agile Methodology

5 Easy Steps to Make Agile Development Work Better for You

Doug Rosenberg
ICONIX

Overview

Your organization is committed to Agile, Scrum and TDD. That's not going to change. But somehow things aren't going as smoothly as you'd like. Your project is on the "underplanned" end of the spectrum and you feel like everything could be working more smoothly. If I've just described your current situation, then this article might be for you.

The Resilient Agile (RA) process combines Test Driven Development with Design Driven Testing, resulting in improved coverage for both unit and acceptance testing, and helps you plan your sprints better by introducing visual modeling of user stories, epics, and tasks. RA is agile on the project management side, and scenario driven on the technical side.

Figure 1 - Test Driven Development and Design Driven Testing can not only co-exist, but complement each other

Speaking of the technical side, Resilient Agile also gets everyone on the team thinking "model-view-controller" very early on, so your team's designs share a common pattern. RA uses a conceptual decomposition of scenarios into models, views and controllers – these conceptual elements don't always wind up as the structure of the code so implementation with MVC frameworks like Angular and Backbone is not required, but if you do happen to be developing with an MVC framework, you'll be in great shape.

If you're thinking that this is too much planning for an agile project, I encourage you to read the discussion on this topic at the end of the article first (see Fig. 9), and then consider reading the article all the way through from the front.

Room for Improvement?

If you're still reading, there's a decent chance that your agile development efforts aren't going as smoothly as you might like. A few possible areas for improvement are shown in Figure 2.

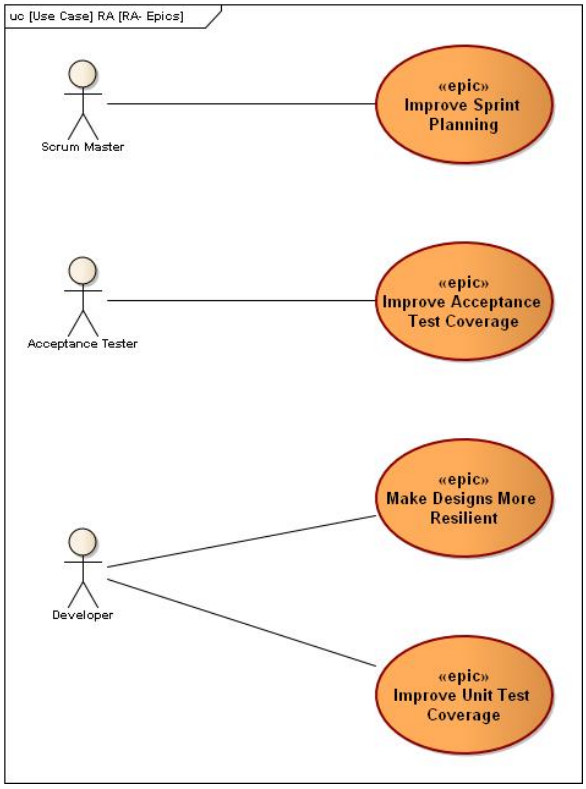

Figure 2 - Can your project benefit from better sprint plans, better test coverage, and more resilient designs?

If agile isn't working so well for your team, there's a good chance that you're suffering from one or more of the following symptoms:

- *Your sprints could be organized better. The stuff in your backlog doesn't really feel like it fits cohesively together into a single system. It feels like there wasn't very much upfront planning before coding started. Probably because there wasn't very much upfront planning before coding started.*

- *Your test coverage could use some improvement. Both unit tests (you have some but coverage isn't great) and acceptance tests (you wish you had more of them).*

- *Your team suffers from "sunny-day syndrome" wherein they neglect to consider failure modes and rainy day scenarios. Don't worry…be happy…until the bug reports start coming in. That's when you wish your designs were more resilient to variations in behavior by the users, and you especially wish that you'd thought through all of the exception behavior carefully.*

5 Steps to Better Results with Agile Development

If these ailments are plaguing your project, our 5-step approach might help. So without further ado, here it is:

1) **Use visual modeling to organize your stories, tasks, and epics**
2) **Model requirements and sunny/rainy day scenarios as well as user stories**
3) **Drive acceptance testing from requirements and scenarios**
4) **Do a "conceptual-MVC" decomposition for each scenario**
5) **Add the MVC controllers and controller tests to the backlog**

Step 6 is BAU (Business As Usual) agile development. Just follow the normal TDD process for each controller.

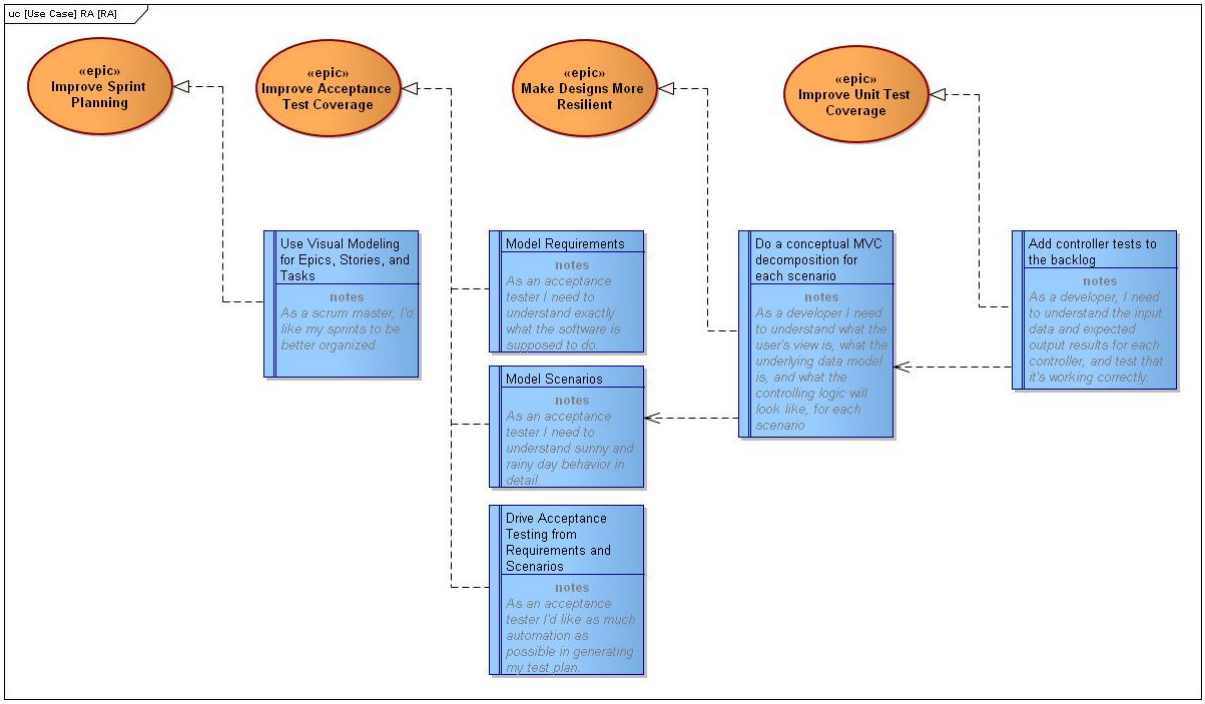

Figure 3 - The path to better test coverage and more resilient designs is straightforward

Let's take a closer look at each of these steps one at a time to see what's required.

Step 1: Use visual modeling to organize your stories, tasks, and epics

What's the goal? Or perhaps I should ask, what's the story? How about this one:

"As a Scrum Master, I'd like my sprints to be better organized."

It isn't difficult to apply some organization to your sprint plans, helping you to have better, more effective, agile project management. You can model most projects with a simple hierarchy of epics, user stories, and tasks. Figure 4 illustrates how this works (and so do the rest of the diagrams in the article). I've used Epics in this article to describe the goal of what we're trying to improve. Each Epic is realized by one or more User Stories, and each User Story is accomplished by a set of tasks.

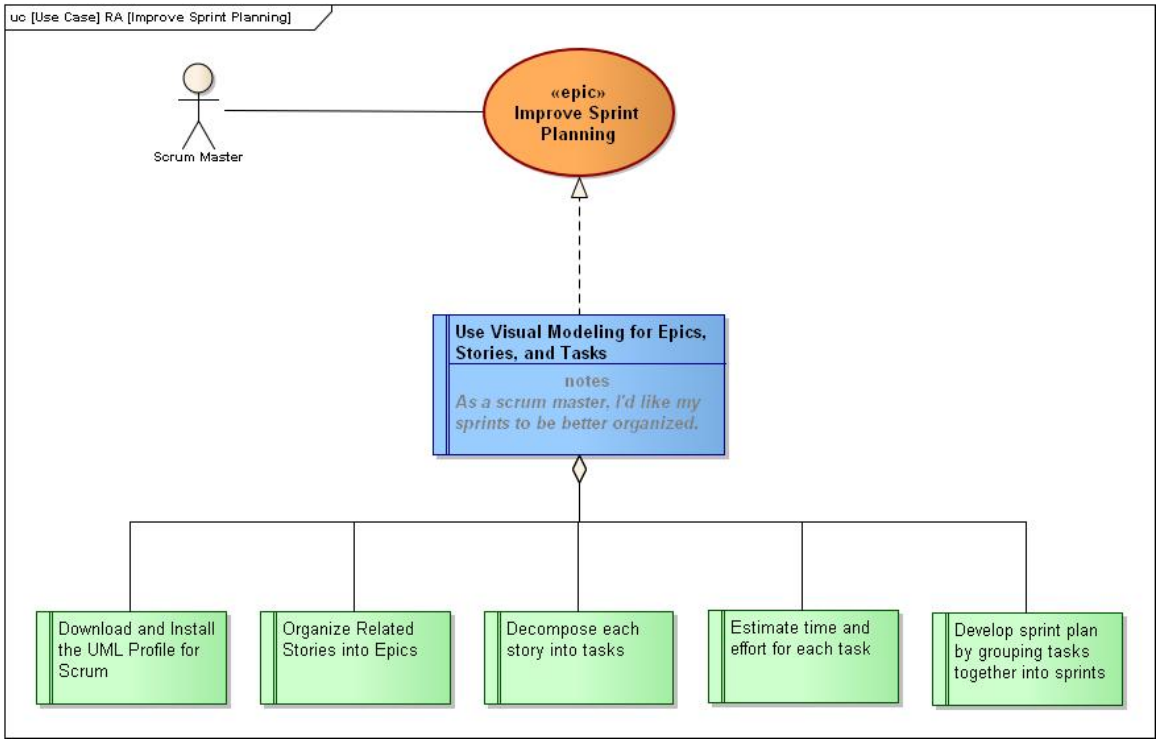

Figure 4 - It's easy to add a little organization to your sprint plans

This simple presentation is easy for everyone on your project to understand, and having all the tasks captured allows you to do bottom-up estimation on a task-by-task basis, then roll up your task estimates for each story, and roll up the story estimates for each epic.

But developing better sprint plans is really just the first step.

Step 2: Model requirements and sunny/rainy day scenarios as well as user stories

User stories are useful to help express the reason a particular software feature is needed, but the nebulous definition of a story can result in acceptance test plans that are inadequate.

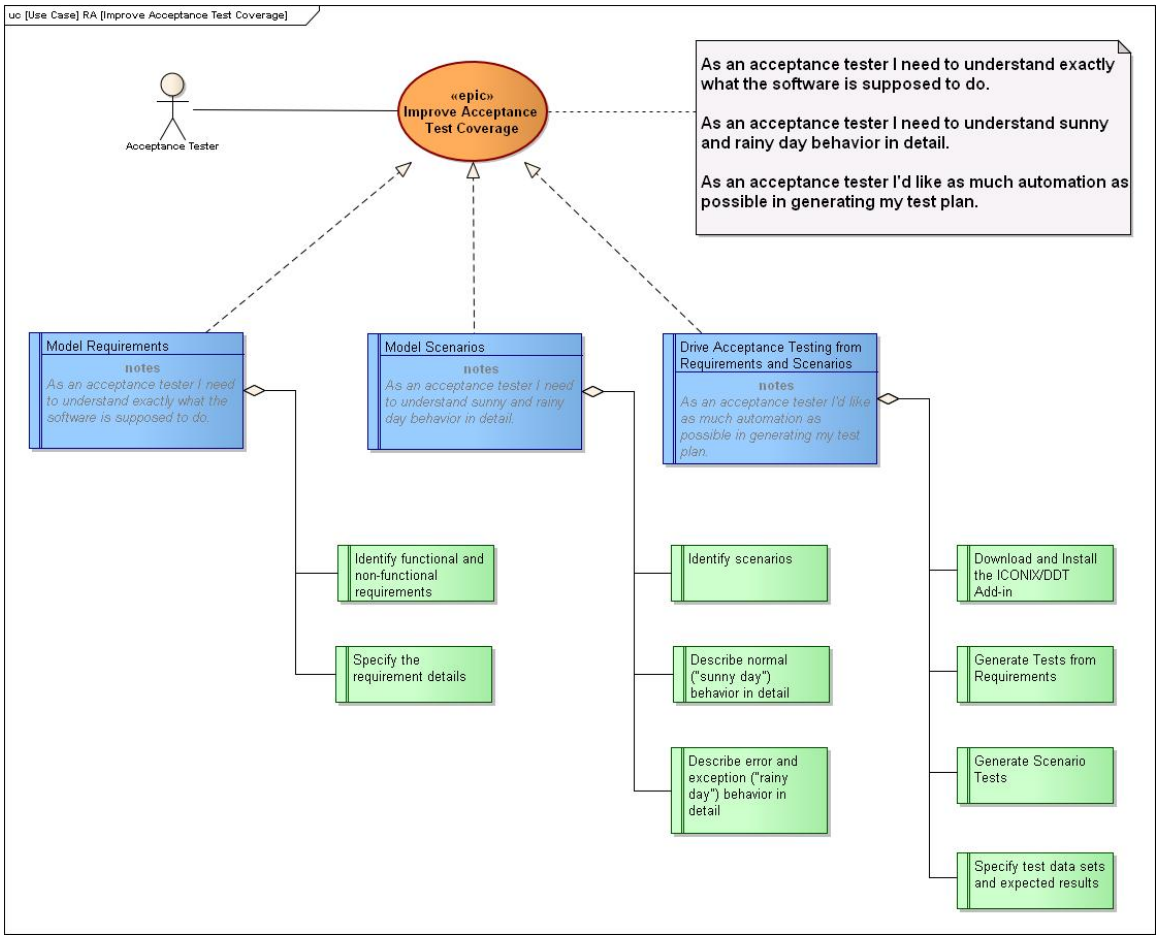

Figure 5 - Acceptance testing requires a detailed understanding of requirements and sunny/rainy day scenarios

There are multiple things that acceptance testers must consider.

Here are two important ones:

- ***Are you testing against a complete list of requirements?***
- ***Are you exercising sunny/rainy day paths through the operational scenarios?***

Step 3: Drive acceptance testing from requirements and scenarios

Improving your acceptance test coverage can actually be a straightforward process, once you realize that the keys to acceptance testing are to test against a complete list of requirements and to fully exercise sunny-day/rainy-day behavior paths.

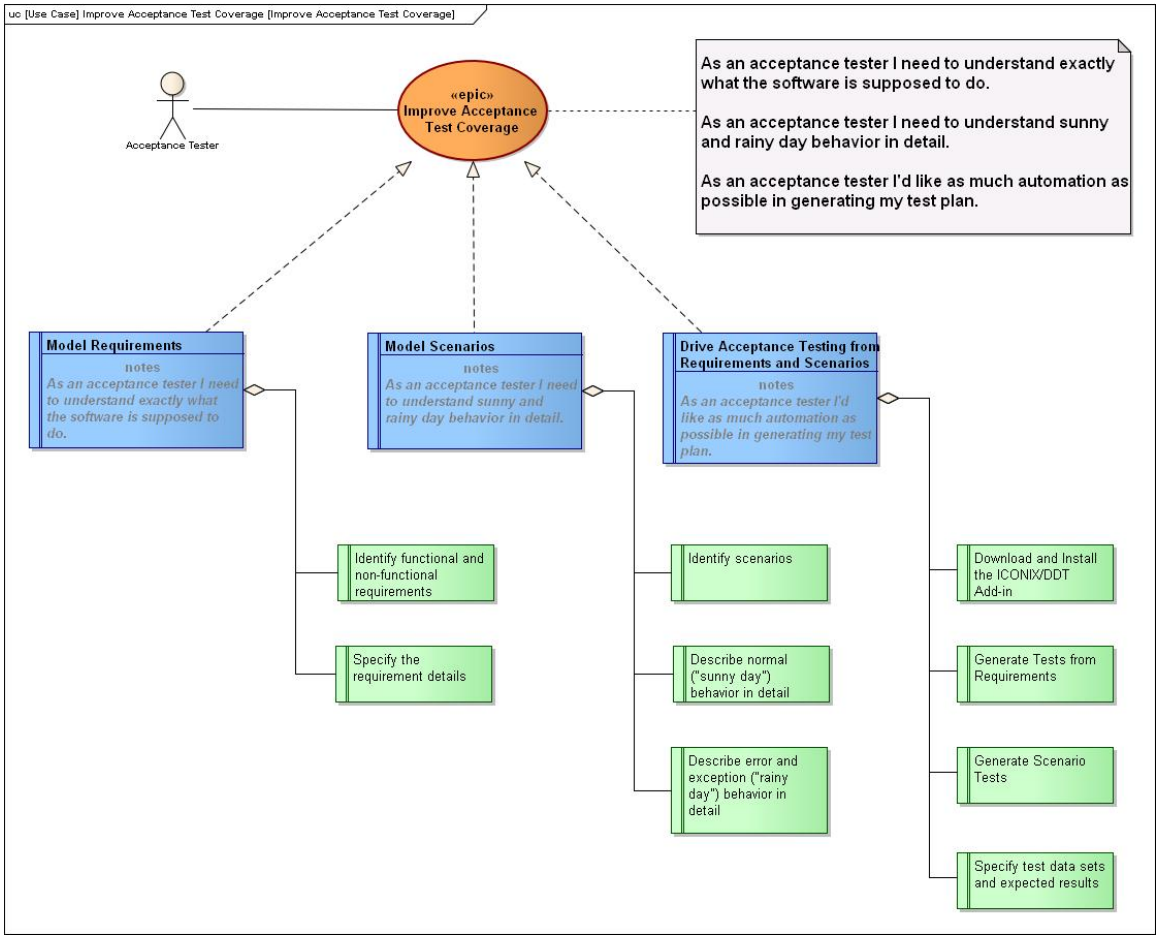

Figure 6 - The keys to acceptance testing are requirement tests and scenario tests

If you're developing with TDD, you're already used to writing unit tests before writing code. As it turns out (and this is one of the main tenets of Design Driven Testing), there are also significant advantages to writing the acceptance tests before writing code. The main benefit being that you have a much better understanding of all the details of what your code is supposed to do once the acceptance tests are written.

In addition to improving acceptance test coverage, you might want some help with improving your unit test coverage. But before that, once you've described your scenarios in detail, you're in good shape to improve the resiliency of your designs.

Step 4: Do a "conceptual-MVC" decomposition for each scenario

Model-View-Controller (MVC) is a design pattern that you probably already know, and there's a decent chance that you've been coding to this design paradigm for years. But it might have never occurred to you that you can use MVC at a conceptual level as a scenario-decomposition technique. In other words, you need to know what data your scenario is dealing with (the Model), you need to know what the user will see on the screen (the View) and you need to identify the logical functions that are needed to connect to the Model and the View (the Controllers).

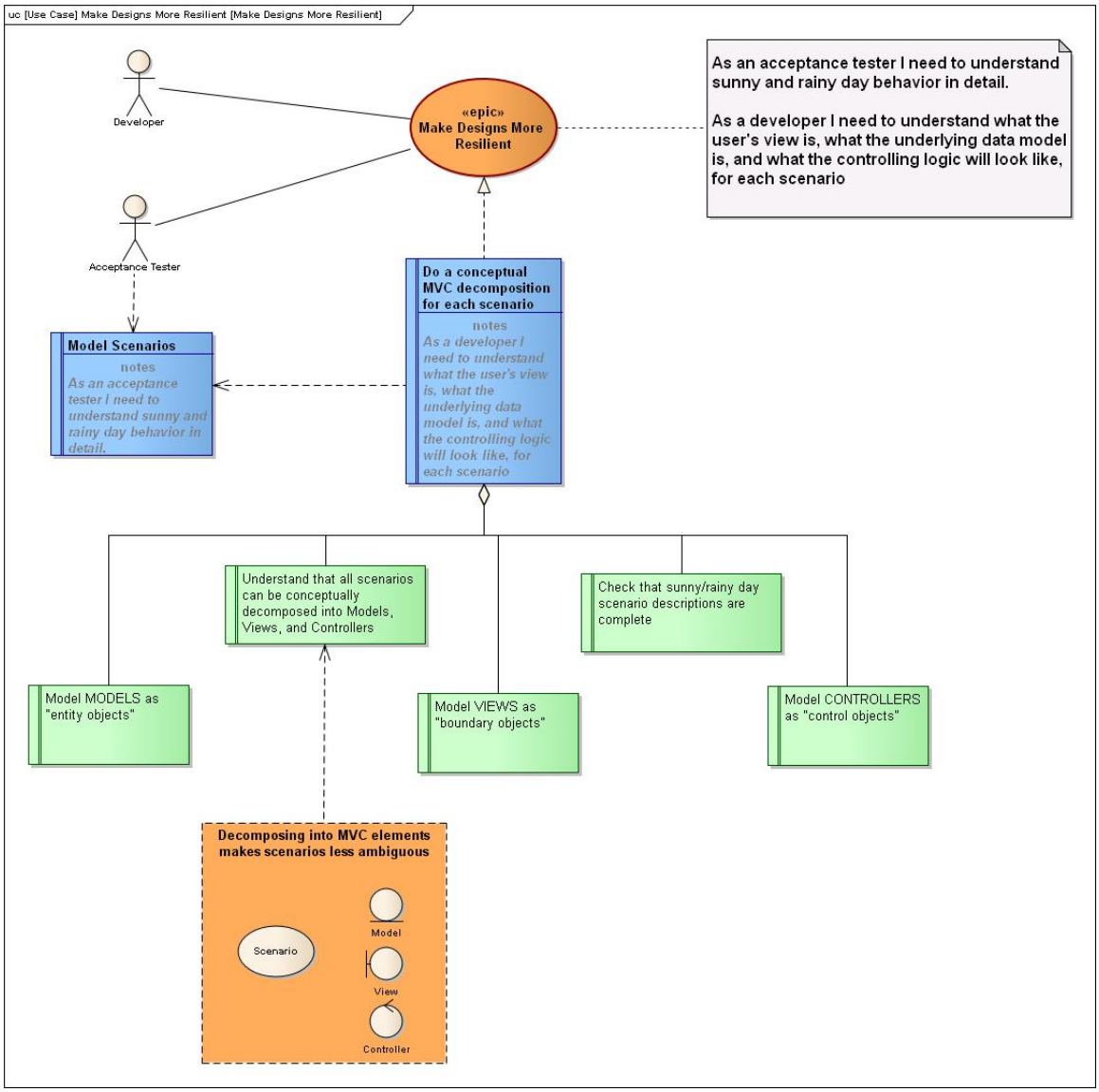

Figure 7 - Identifying controllers for all "rainy-day" scenarios makes your designs more resilient

Conceptual MVC decomposition for all of your scenarios (including both sunny-day and rainy-day parts of each scenario) is what puts the resilience in Resilient Agile. It's also a great opportunity for collaboration between developers and acceptance testers.

Step 5: Add the MVC controllers/controller tests to the backlog

Once you've gone to the trouble of identifying all of these controllers (aka higher level software functions or "logical functions") it seems like you should do something with them. So how about putting them into the Scrum backlog?

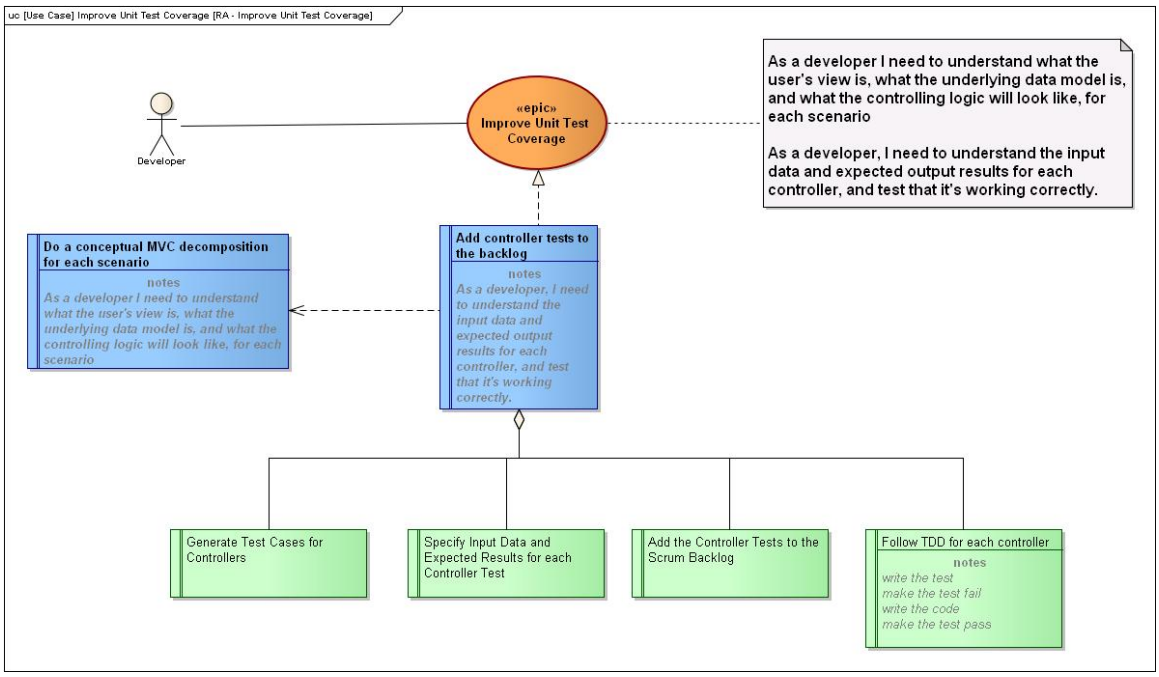

Figure 8 - Using the Design Driven Testing add-in, you can quickly generate test cases for each controller and specify input data and expected results

One controller at a time is a simple and easy way to organize your coding activities. If you're following a test-first approach, you'll want to have JUnit/NUnit test classes defined for each controller with setup and teardown methods, the input data for your tests clearly specified, along with a description of the expected results. Luckily, the ICONIX/DDT add-in for Enterprise Architect makes it simple, easy, and fast for you to generate these test cases from the model. If you choose this path, you'll find that your unit test coverage is improved (in addition to your acceptance test coverage being better).

From here you're in familiar territory and you can just follow the test-driven process for each controller. Simple enough.

Isn't this too much planning for an agile process?

While it's true that the agile manifesto devalues planning as compared to feedback, that doesn't mean that your project has to eliminate planning altogether. In fact, eliminating planning is a suicidal strategy for a software project. The key to doing "just enough planning" is to keep your planning adaptive.

One of my agilist colleagues, Prof. Chuck Suscheck, has a wonderful way of expressing this concept. In Chuck's words "***the fidelity of the plan is directly related to its time horizon***". In other words, when we're planning two years ahead, we paint the picture with broad brush strokes, but when we're planning the stuff we're about to build, we still need to plan it carefully.

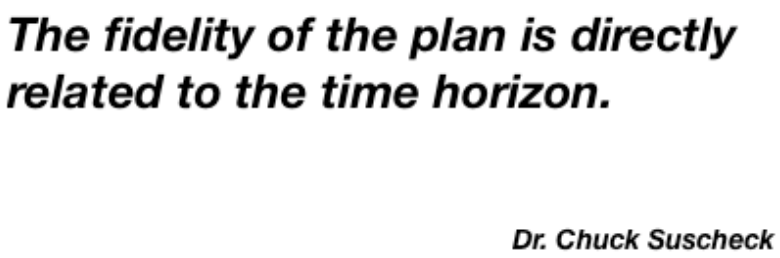

Figure 9 - Even agile projects require careful planning, but agile plans need to be adaptive

A corollary to Chuck's time horizon that we can consider is that ***the fidelity of the plan is also directly related to the scope of what we're planning***. With Resilient Agile, we plan carefully one scenario at a time. That includes both sunny and rainy day planning for the scenario, as the failure to consider "rainy day" behavior is one of the leading causes of project failures. If we release a piece of software and the desired behavior for a scenario changes, we can easily toss the existing one and re-plan it, because the scope of our scenarios is typically around two paragraphs of narrative text.

So…plan in small increments and for short time horizons. But when you plan, do the best plan you can manage to do.

I'll close here with a story about a project I was consulting on where we were trying out some of the techniques (specifically visual modeling of sprint planning) described above. This project, for a client who shall remain nameless, was originally scoped in excess of $20 Million dollars budget. We modeled the stories, tasks, and epics and did bottom-up cost estimation. The project then went through a series of de-scoping and re-scoping exercises and, because we had captured the plan in Enterprise Architect, we were able to quickly delete parts that got de-scoped, drag-and-drop others, and rapidly reorganize and re-publish the plan. We planned carefully but adapted the plan to fit schedule and budget constraints.

Summary

In this article, I've presented you with a simple 5 step process for making your agile development process work better.

With Resilient Agile you will get:

- *Better organized sprint plans*
- *Acceptance tests that cover requirements and rainy-day scenarios*
- *More resilient designs that include controllers to handle exception behavior, and*
- *Comprehensive unit test coverage for all controllers*
- *Plans that are adaptive and flexible in response to change*

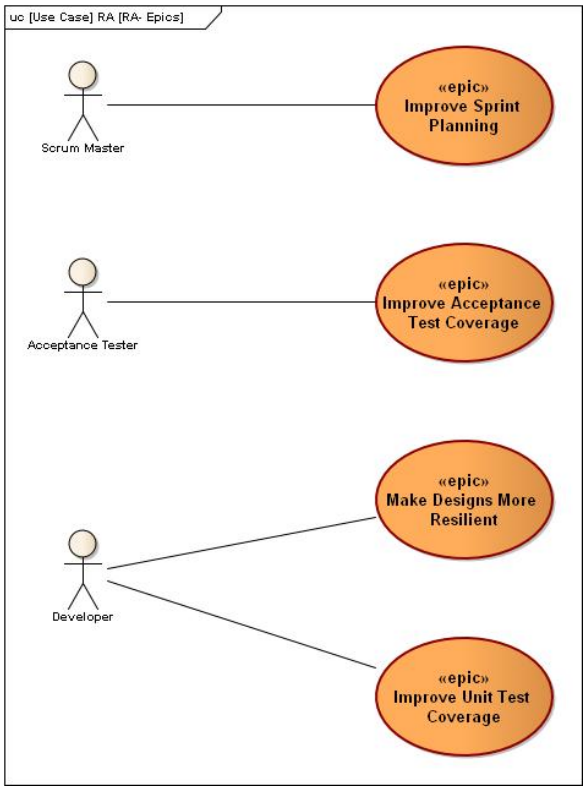

Figure 10 - If agile is already working perfectly for you, you don't need Resilient Agile. If not…

Visit us at www.iconixsw.com to learn more.