blueprint

Whitepaper

Optimized Use Cases

Use cases have what seems to be an endless stream of textbooks and articles written on the topic. In some situations use cases have excelled, producing remarkable results. In others, they have not fared as well. This paper describes some best practices for use cases, accumulated through the use of Blueprint on several projects.

BP189173 3-Jan-2014

Introduction

Use cases have been around for some time now, bursting onto the public spotlight at the 1987 OOPSLA (object oriented programming, systems, languages and applications) conference in Orlando. Since that time they've been used in countless projects. Use cases have what seems to be an endless stream of textbooks and articles written on the topic. In some situations use cases have excelled, producing remarkable results. In others, they have not fared as well.

One missing piece to this equation is the fact that requirements definition as a professional discipline has been under served until recently by the industry and by software tool vendors. Communities, professional organizations, and modern software toolsets are now finally available for requirements authors and to support the requirements lifecycle. Some of these tools bring new and innovative capabilities not imagined before, and as they get applied to existing approaches like use cases, new sets of best practices emerge.

This paper describes some best practices for use cases, accumulated through the use of Blueprint on several projects.

Fundamentals

Less is more

A pervasive, but mistaken belief in the world of requirements gathering is that quantity is always a good thing. The reality as that the goal of requirements is to precisely communicate what is needed, and for the consumer of the information to completely and accurately understand what has been communicated.

Example: Pretend for a moment that you just got the job running the county. Tourists arrive with the goal of discovering the area – its culture, character, history, attractions, and so on. Your job is to help them achieve this goal. In your enthusiasm you start creating brochures detailing the various aspects of the county. And you keep on making brochures. So many in fact, that you end up with hundreds that cover every mundane little thing in the area.

After a month of operation you discover there's no pattern to the brochures taken – one from here, one from there. Something like 90% of the brochures haven't even been touched. Surveys of tourists show no consistency in their perceptions of the county – it is almost as if different tourists have visited different counties entirely.

To summarize the results of this:

- The goal of communicating the unique culture and character of the area wasn't met;
- Visitors all left with very different impressions and understandings of the county;
- You spent a lot of time, effort, and money providing information and 90% of it wasn't even used.

This example is much like the requirements situation on many projects and organizations. "Information overload" is a huge problem. So often the answer to every issue and misinterpretation in requirements documents is to add more content in order to 'elaborate' and 'clarify'. Simply adding more content is generally counterproductive, often doing more harm than good by introducing conflicting information, inconsistencies, and redundancies.

When authoring requirements you should always put yourself in the position of the consumer. You should strive to communicate what's needed using the smallest volume of content possible. Since even this can be considerable in size, you also should strive to make that content navigable. This means you should structure the content in order to best understand what's being communicated. While this takes skill and effort on behalf of the requirements author, the positive effects on the software project can be dramatic.

Know your boundaries

If we had to pick one aspect of use case models that people should ensure they do right, it would be to have a good understanding among all stakeholders of what the system boundary is. A system boundary identifies where the role of the system ends.

Functionality outside the boundary does not need to be defined. Functionality within the boundary must be defined. For some applications it's obvious and apparent, while for others it can be quite the opposite. Since a use case documents a dialog that spans this boundary, not having a good understanding of it can severely reduce the clarity of your requirements. For those who rely on the use case model to do their work, people like designers and testers, their work-products will similarly suffer.

Fantasy vs. Reality. Try bottom-up

For those new to the use case approach, it's easy to get lost in use cases, includes, extends, actors, associations, models, and so on. With so much focus on learning to navigate this new and imaginary world, it's easy to lose sight of the real world it's supposed to represent.

Developing work-habits that regularly move you back and forth between the two can help keep your modeling work grounded in reality. As an example of this, Figure 1 shows all the steps of all the use cases in a workflow diagram that covers the 'end to end' behavior of the system.

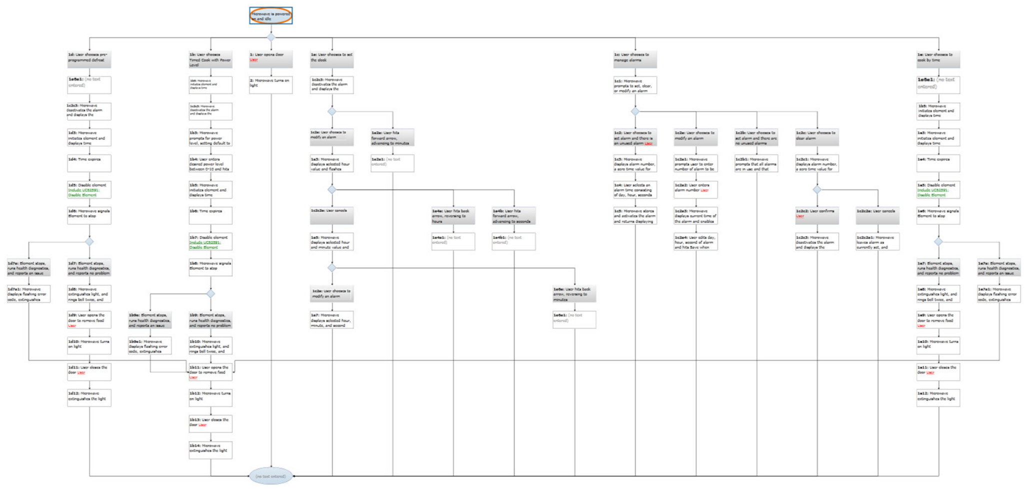


Figure 1 Workflow Diagram showing activities & decisions

This diagram very simply shows the end-user's actions and decisions along with the system's responses for the application being specified. This is something everyone will understand, without training. It doesn't concern itself with how these steps are 'packaged' into imaginary containers called "use cases" and the various types of relationships between them.

Use cases do of course provide many benefits, not the least of which is to clearly identify a higher purpose for a collection of steps & decisions, to answer the question “why are we doing these actions”.

So, where a group of steps and decisions together fulfill some goal of an external user, we group them and that becomes a use case. Where a flow crosses from one group to another, this becomes a relationship. If the flow came from a decision (diamond) it’s an “extends” relationship. Otherwise, it’s an “include” relationship.

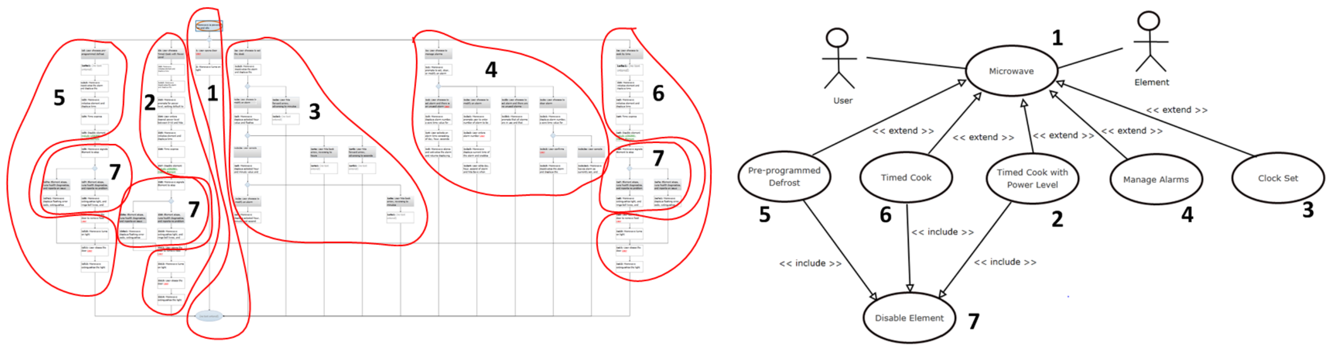


Figure 2 Use Cases from Workflow

Figure 2 shows the workflow with hand-drawn groupings along with it’s associated use case diagram.

There is typically little debate over what the actions and decisions are as this is simply the real-world work that needs to get done.

On the contrary, there is often debate over what the use cases are, since these are abstract groupings of the real-world steps and decisions, and one person could come up with a different grouping than the next person.

For multiple people to independently group the steps and decision and come up with a comparable set of use cases is an indication of a well-understood problem space. It’s important when modeling to not lose sight of the real-world which is the subject of your model.

Modeling style

The use case diagram you see in Figure 2 is one ‘style’ of structuring. This style makes use of decomposition, where abstract actions or steps are decomposed into finer detail contained in included use cases. There is much debate over whether decomposition with use cases is a good idea or not. It is presented here simply as one approach.

One advantage it offers is to allow readers to “adjust” the level of detail at which they work at a high level, or follow the path of decomposition to drill into the details of a specific action. One objection to the use of decomposition is that it will influence the developers who work from the use cases toward implementations that are not object-oriented in nature.

Think like a tester

Tests are very similar to requirements. Both are descriptions of "what the system is supposed to do" so if they're talking about the same piece of software then they need to be completely consistent. Some in fact hold the point of view that tests are really just a more detailed form of the requirements.

Looking at the requirements from the perspective of a tester can be very valuable for detecting issues early. In particular I've found the tester's perspective can help a great deal in defining the system boundary. This is because testing is all about providing "stimulus" to some "thing" and then observing that "thing's" behavior in response, so the borders of where the "thing" begins and ends need to be clear, and testers are great at thinking this way.

A second area where testers have been found to be a great help in requirements definition is thinking of exception cases. Stakeholders and business analysts tend to be very good at identifying what the system should do when things go right, but experienced testers excel at thinking of all the possible ways that things can go wrong. Having this knowledge up-front means it can be accounted for and influence the requirements while they're being defined, as opposed to being an after-thought late in the cycle.

In addition to the tester as an individual, modern requirements toolsets that can automatically generate tests provide tremendous value as well. When reviewing requirements – actually even when authoring them – these tools allow the corresponding tests to be instantly generated and used to provide another litmus-test for requirements quality. Often inconsistencies and errors can be spotted in the generated tests that were missed when reviewing just the requirements.

Be expressive

Use cases are our tool for communicating what is to be built. To achieve this you need to be as expressive as possible with use cases. No matter how good you are with words, written text can only go so far. One of the easiest yet most effective ways is to mock up potential user interfaces for the steps of the use case. In other words, render the more significant aspects of the user interface as it evolves through a use case. Most often this will just be simple sketches, but where appropriate can be higher-fidelity visualizations. Together this set of mock-ups will form a storyboard for the various scenarios.

If the nature of the project is enhancing an existing application, screen-shots of the existing application can serve as the starting point, then annotate, markup, or add new screen controls as needed. Shifting focus onto storyboards as opposed to the text of the use case flows can make reviews significantly more effective. As with test generation mentioned earlier, there are tools now that support this more visual approach to defining use cases.

Another way to be more expressive is with data or information. Where the use case affects or transforms data in the application or where data influences the behavior of the use case, instead of describing this in text, modern requirements tools will actually allow you to encode these calculations and updates.

During a simulation session not only will the visualizations be shown but samples of real data can be entered, displayed and calculated similar to the real application. Together this has the effect of 'bringing the story to life' as opposed to forcing reviewers to imagine it from textual use cases.

Another example of increasing the expressiveness of the use cases again is provided with modern automation. There is typically a large amount of reference material on a typical software project related to the requirements. This can be standards, guidelines, regulations, vision documents, operational documents, and more.

Tools today can automatically open, navigate to relevant content, and highlight it when these references are called upon. This automatically brings relevant content into the discussion as opposed to leaving it buried off to the side where important aspects could be missed.

The bigger picture

Where do they come from?

Let's say I came up to you and said, "Hey, please make me three or four detailed use cases. I'll be back after lunch to pick them up, and I expect them to be correct!". Your chances of delivering what I need are pretty much zero.

You need to find out what are my goals for the application, what the application is to be used for, what major decisions will I need it to support, is this enhancing something that exists or is it new, are there any constraints, are there any special needs like security or safety, and other questions like this. In other words, you have some work to do before you get to the use cases.

Typically this results in textual, hierarchical lists of goals, rules, and often sketches of the business processes and possibly domain diagrams in which the application is to support (see figure 3a, 3b, and 3c).

So I've made some use cases – now what ?

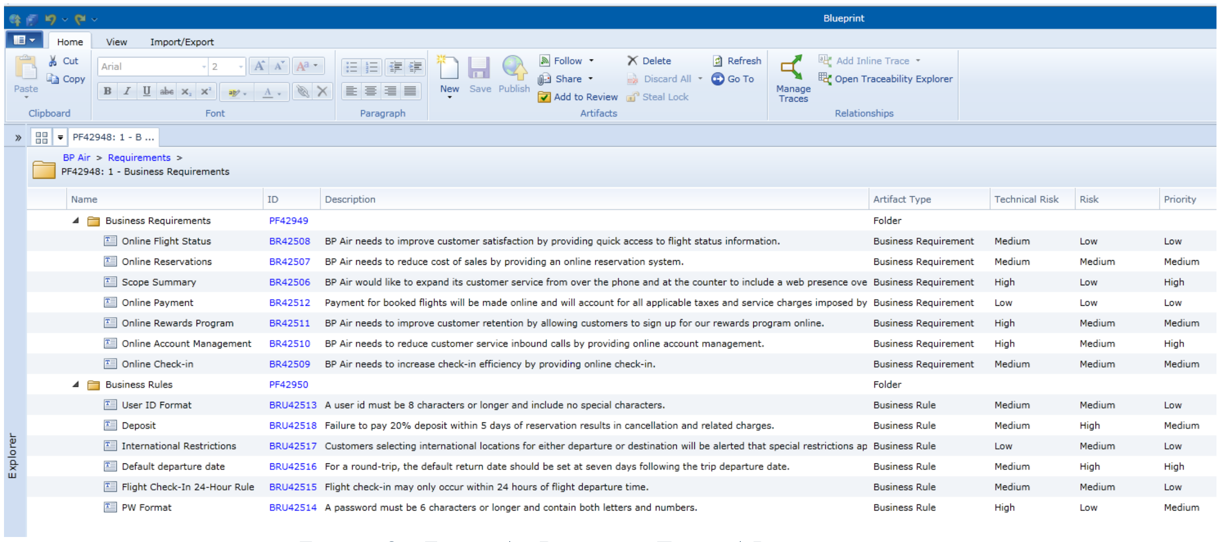
After creating use cases, you'll need them to be reviewed with the client for whom the application is for, and also with the people that are to build and test the application. Reviews of requirements are one of the most crucial control points in the software lifecycle. It's an opportunity to point the project in the right direction, and to do so early. Errors missed in reviews are simply errors whose discovery has been delayed – they will eventually be found, just later when they're more expensive to fix.

The effectiveness of the review depends on how well the requirements can be communicated. The more expressive the requirements, the more likely they'll be communicated clearly. Another major way to be more expressive, is to use simulation during reviews.

Modern requirements definition toolsets support simulation of requirements where the requirements can be "brought to life" to give an impression of how the future application, if built to these requirements, will look, feel, and behave. After all, most people when reading requirements are in fact performing a simulation in their minds trying to visualize the future application to help decide if this requirement acceptable.

The problem is that in their minds' simulation things are missed, all the interactions cannot be accounted for, and perhaps worst of all everyone has a somewhat different vision depending on how they interpret the written text. Automated simulation, projected for all to see, has none of these issues and provides all the benefits – literally a common vision (see Figure 4).

Not only is communication vital during reviews to get the requirements right, but it's also vital for those who will



BP Air > Requirements >
PF42948: 1 - Business Requirements

Name	ID	Description	Artifact Type	Technical Risk	Risk	Priority
Business Requirements	PF42949		Folder			
Online Flight Status	BR42508	BP Air needs to improve customer satisfaction by providing quick access to flight status information.	Business Requirement	Medium	Low	Low
Online Reservations	BR42507	BP Air needs to reduce cost of sales by providing an online reservation system.	Business Requirement	Medium	Medium	Medium
Scope Summary	BR42506	BP Air would like to expand its customer service from over the phone and at the counter to include a web presence ove	Business Requirement	High	Low	High
Online Payment	BR42512	Payment for booked flights will be made online and will account for all applicable taxes and service charges imposed by	Business Requirement	Low	Low	Low
Online Rewards Program	BR42511	BP Air needs to improve customer retention by allowing customers to sign up for our rewards program online.	Business Requirement	High	Medium	Medium
Online Account Management	BR42510	BP Air needs to reduce customer service inbound calls by providing online account management.	Business Requirement	High	Medium	High
Online Check-in	BR42509	BP Air needs to increase check-in efficiency by providing online check-in.	Business Requirement	Medium	Medium	Medium
Business Rules	PF42950		Folder			
User ID Format	BRU42513	A user id must be 8 characters or longer and include no special characters.	Business Rule	Medium	Medium	Low
Deposit	BRU42518	Failure to pay 20% deposit within 5 days of reservation results in cancellation and related charges.	Business Rule	Medium	High	Medium
International Restrictions	BRU42517	Customers selecting international locations for either departure or destination will be alerted that special restrictions ap	Business Rule	Low	Medium	Low
Default departure date	BRU42516	For a round-trip, the default return date should be set at seven days following the trip departure date.	Business Rule	Medium	High	High
Flight Check-In 24-Hour Rule	BRU42515	Flight check-in may only occur within 24 hours of flight departure time.	Business Rule	Medium	Medium	Low
PW Format	BRU42514	A password must be 6 characters or longer and contain both letters and numbers.	Business Rule	High	Low	Medium

Figure 3a Example Business Textual Requirements

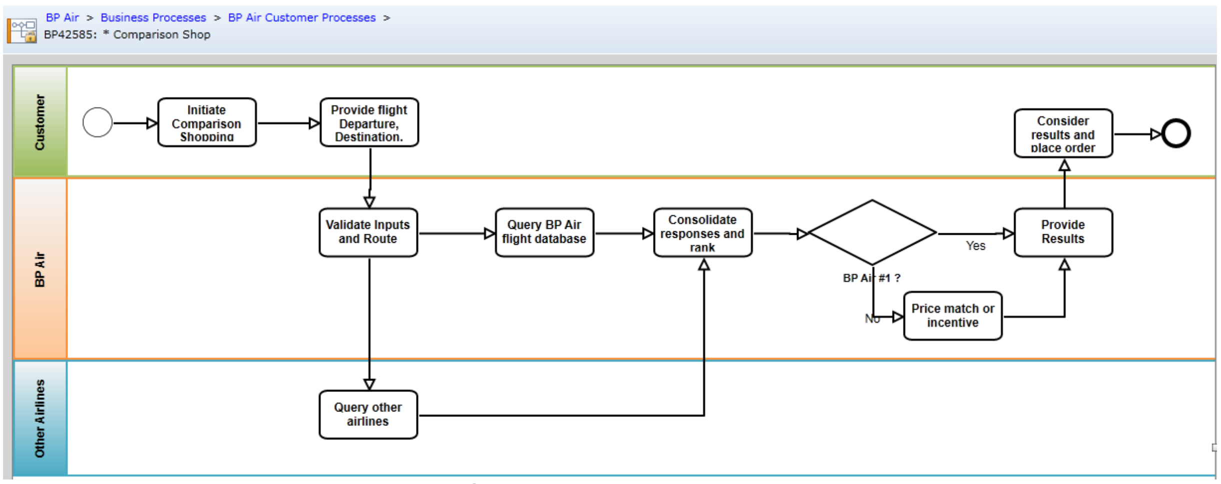

BP Air > Business Processes > BP Air Customer Processes >
BP42585: * Comparison Shop
Customer
BP Air
Other Airlines
Initiate Comparison Shopping
Provide flight Departure, Destination.
Consider results and place order
Validate Inputs and Route
Query BP Air flight database
Consolidate responses and rank
BP Air #1 ?
Yes
No
Provide Results
Price match or incentive
Query other airlines

Figure 3b Example Business Process

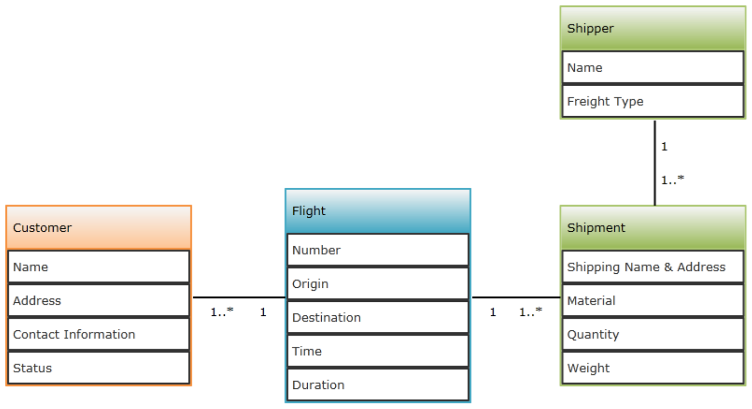
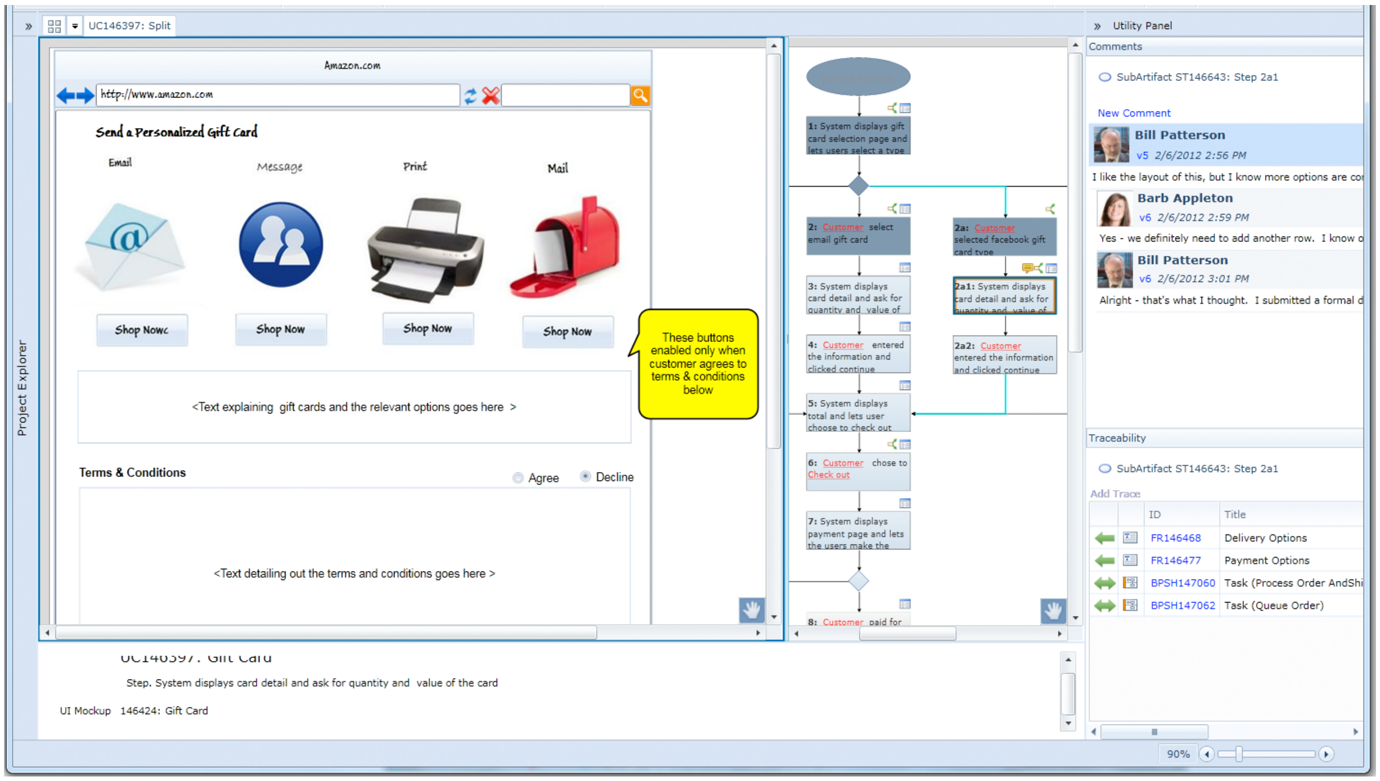


Figure 3c Example Business Domain Diagram


UC146397: Split
Project Explorer
Amazon.com
http://www.amazon.com
Send a Personalized Gift Card
Email
Message
Print
Mail
Shop Nowc
Shop Now
Shop Now
Shop Now
These buttons enabled only when customer agrees to terms & conditions below
<Text explaining gift cards and the relevant options goes here >
Terms & Conditions
Agree
Decline
<Text detailing out the terms and conditions goes here >
1: System displays gift card selection page and lets users select a type
2: Customer select email gift card
2a: Customer selected facebook gift card type
3: System displays card detail and ask for quantity and value of
2a1: System displays card detail and ask for quantity and value of
4: Customer entered the information and clicked continue
2a2: Customer entered the information and clicked continue
5: System displays total and lets user choose to check out
6: Customer chose to Check out
7: System displays payment page and lets the users make the
8: Customer paid for
Utility Panel
Comments
New Comment
Traceability
Add Trace
90%

UC146397: Gift Card

Step. System displays card detail and ask for quantity and value of the card

UI Mockup 146424: Gift Card

SubArtifact ST146643: Step 2a1

Bill Patterson v5 2/6/2012 2:56 PM
I like the layout of this, but I know more options are co

Barb Appleton v6 2/6/2012 2:59 PM
Yes - we definitely need to add another row. I know o

Bill Patterson v6 2/6/2012 3:01 PM
Alright - that's what I thought. I submitted a formal d

SubArtifact ST146643: Step 2a1

ID	Title
FR146468	Delivery Options
FR146477	Payment Options
BPSH147060	Task (Process Order AndShi
BPSH147062	Task (Queue Order)

Figure 4 Screen from Use Case Simulation

build and test the application to understand what they need to do. Any miscommunication here means people will go off in the wrong direction with their work. Simulation is also a very good tool for this as well. Once they do understand however, they actually need access to the use cases and associated requirements information in order to do their work since their tasks depend upon this information. These are areas where modern tools can really make a difference, in a number of ways.

First, tools today can automatically generate tests directly from use cases. This is a huge time-saver. Not only is the work done automatically, but it's correct. Even more advanced tools allow you to filter the set of tests produced to focus only on those of high-risk, or the most business-critical.

Second, requirements definition tools today also can auto-populate the tools used by the designers and testers with the requirements, and tests, produced using the requirements definition tool. This avoids transcription errors and oversights that often happen when you deliver a document, and the practitioner needs to manually enter relevant information into their toolset.

Third, requirements definition tools today can automatically generate the documentation you need either because your process calls for it, or to comply with corporate standards. Document generation governed by templates allows you to define entirely the format and content of the documents. More advanced tools can even automatically produce red lined documents showing changes since some previous version like the last review session for example.

conclusion

There have been significant gains made in requirements definition tools in recent years. This perhaps shouldn't be surprising given that this area, arguably one of the most crucial for determining the success of software projects, was neglected by software tool vendors for decades. These advancements, coupled with best practices learned by applying this new technology in real and complex projects, have the potential to clear the log-jam of software project failures that has been plaguing the industry for years.

About Blueprint

Founded in 2004, Blueprint develops requirements definition and management (RDM) software. With its best-in-class RDM solution, Blueprint helps companies get complex software and IT project requirements right from the start. Blueprint solves many of the time-consuming, costly, and error-prone elements of defining requirements, ensuring that mission critical projects are completed on time and budget. Headquartered in Toronto, Ontario, Blueprint has global sales, operations and partner presence.

General Inquiries and Sales

info@blueprintsys.com
647.288.0700
1.866.979.BLUE(2583)
+44 203 051 0432

www.blueprintsys.com