Whitepaper

Be Expressive

Improve the Clarity of your Requirements

A requirement is how two (or more) people have chosen to express their agreement regarding what is needed from a software application. The assumption is that all parties have the same understanding or interpretation of these written words. Quite often this assumption proves to be false - different stakeholders in fact had different ideas in their minds regarding these same words, and confidently thought that others shared their vision as they warmly smiled and shook hands over the agreement.

BP189171 3-Jan-2014

INTRODUCTION

If you were somehow able to hold a requirement in your hand and ask, “why do I need this? What is its purpose?”, I would offer that you’re holding an “agreement”. A requirement is how two (or more) people have chosen to express their agreement regarding what is needed from a software application. The assumption is that all parties have the same understanding or interpretation of these written words. Quite often this assumption proves to be false - different stakeholders in fact had different ideas in their minds regarding these same words, and confidently thought that others shared their vision as they warmly smiled and shook hands over the agreement. If only we could have a “vulcan mind-meld” a la star trek, a lot of our requirements issues would be gone.

These issues are possible with each and every written requirement. When you multiply this by hundreds, or sometimes thousands of requirements and then consider how the requirements must “combine” to paint a picture of the whole solution, it’s quite understandable why there can be so many “surprises” in the latter parts of a software project.

The fundamental problem is that traditional legal-type textual requirements are simply not very expressive, which tends to leave a lot open to interpretation. Very often, requirements authors attempt to compensate for this lack of clarity by adding even more legal-style text to elaborate and constrain what they’re trying to communicate. Sometimes this can lessen the range of interpretation as desired; but just as often, it can actually do the opposite, as these new words themselves may contain ambiguities, inconsistencies, or redundancies. These additional words are added to areas where misinterpretation has been detected. But what about all the hidden areas of misinterpretation that have yet to be detected? They simply remain hidden, lying in wait to rear their head later on in the development cycle.

Stakeholders directly involved in defining the requirements through elicitation, authoring, or review, have the benefit of sometimes hours of discussion and debate that go into formulating the requirements. This rudimentary form of “vulcan mind-meld” is really quite valuable in avoiding misinterpretation of the written requirement. But what of the requirements consumers who did not have a seat at the “definition table”, and didn’t have the benefit of all this undocumented communication? It is crucial that developers, testers, outsourcers, sub-contractors, and others who consume requirements understand what they are asked to build, test and manage. The risk of requirements miscommunication among these key individuals is magnified, since they’re even more likely to misinterpret the original intent.

So in addition to serving as an “agreement” between parties, the requirements have another key purpose: to communicate what is needed completely, precisely, and unambiguously to others who may not have been intimately involved in their definition.

Have you ever noticed what someone does when they’re having difficulty communicating a thought or concept? They become animated. They start using their hands. They make sketches on the whiteboard. They start pointing at the computer screen and say things like “so imagine this”. In short, they reach out for other, more expressive forms of communication. As they do this, they often expose other areas of misconception or misunderstanding. One by one they tackle these new areas, again with various forms of expression.

The same is true with requirements. Increasing the expressiveness of your requirements actually exposes additional areas of miscommunication you never knew were there – in other words, areas where you need to be even more expressive. This effect can be very powerful, and can help to raise the quality of requirements dramatically. Some go so far as to replace textual statements altogether in favor of other more expressive forms of specification. Personally, I've found a combination of the two to offers the best solution, and this is the approach I'll describe.

SETTING CONTEXT

Software requirements express what is "required" of the application in order to fulfill some higher-level business need. Requirements could describe how to provide customers with an online flight booking capability to fulfill the business need of increased sales for the airline, or how to withdraw cash to fulfill my business need of going to a movie.

Regardless, this business need must be expressed along with the associated business process so that this "world" in which the software application will function is:

i) understood by all team members, and
ii) used as a reference for the project to ensure it stays focused on addressing the business need.

This business context can be expressed as a combination of textual statements with a set of business process and business domain diagrams. The textual statements can be categorized according to an appropriate taxonomy (e.g. Business need, business goal, business objective, business rule, business requirement, etc.). There are various taxonomies in use, and I won't use this forum to debate their merits. Business processes are best expressed using business process diagrams. There are numerous notations in use for this, from fill-blown business process modeling notation (BPMN), with its plethora of symbols, to very light-weight notations suitable for sketching processes. The major information elements from the business that are involved and the relationships between them can be illustrated using a domain diagram. The purpose of these diagrams is to simply provide context for the software requirements, not to re-engineer the business, so it's important not to go too far and spend time capturing information that doesn't impact the application. While you can get more specific, and BPMN notation will support this additional detail, the most basic information being captured for a business process would include:

i) Roles – who or what is performing the tasks and making decisions. This is represented using "swimlanes".
ii) Tasks – the discrete work being performed by the role.
iii) Decisions – when decisions are made, who makes them, associated conditions, and the result
iv) Start & end – where the process begins and ends, with ability to specify any pre or post conditions
v) Nesting – the ability to abstract or "nest" portions of a process to help analyze and communicate
vi) Free form notes – the ability to add unrestricted notes (I.e. Text of any size and position).

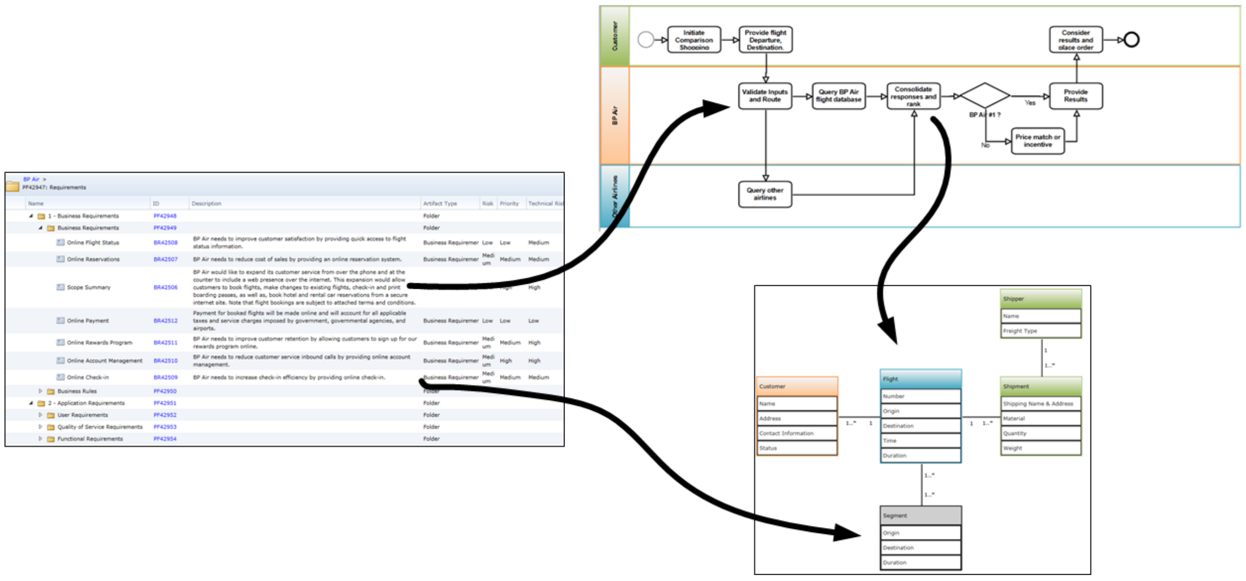
The most basic information being captured for a domain model would include:

i) Entity – a major information element from the business domain (e.g. for a vehicle registration system you may have “car”, “owner”, “trailer”, etc.)
ii) Element – an attribute of an entity that provides (e.g. for car you may have “make”, “model”, “color”, etc.)
iii) Relationships – how two elements are related to one another with additional details like ‘navigability’, ‘multiplicity’, roles, and more.

I’ve found with these essentials I’m able to capture most business contexts, and in the odd case where I can’t, I usually augment with notes.

The business need, the process, and the domain as you can imagine are related. In areas where the relationship between them is noteworthy or illustrative it makes sense to somehow capture and maintain it using a traceability relationship in whichever requirements tool/platform you’re using. Conceptually, the context may therefore appear as in figure 1.

Generally, I ensure the business needs are exhaustive in terms of breadth - meaning if there’s a need, I want it to be specified. On the other hand, I selectively apply business process diagrams and domain diagrams in areas where the process is business-critical, high-risk, or complex. This is admittedly a fairly simple approach, but I find it adequate in a surprisingly large number of situations. Of course, it can be extended where necessary.


BP Air >
PF42947: Requirements

Name	ID	Description	Artifact Type	Risk	Priority	Technical Ris
1 - Business Requirements	PF42948		Folder			
Business Requirements	PF42949		Folder			
Online Flight Status	BR42508	BP Air needs to improve customer satisfaction by providing quick access to flight status information.	Business Requiremer	Low	Low	Medium
Online Reservations	BR42507	BP Air needs to reduce cost of sales by providing an online reservation system.	Business Requiremer	Medi um	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 over the internet. This expansion would allow customers to book flights, make changes to existing flights, check-in and print boarding passes, as well as, book hotel and rental car reservations from a secure internet site. Note that flight bookings are subject to attached terms and conditions.	[illegible]	[illegible]	High	High
Online Payment	BR42512	Payment for booked flights will be made online and will account for all applicable taxes and service charges imposed by government, governmental agencies, and airports.	Business Requiremer	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 Requiremer	Medi um	Medium	High
Online Account Management	BR42510	BP Air needs to reduce customer service inbound calls by providing online account management.	Business Requiremer	Medi um	High	High
Online Check-in	BR42509	BP Air needs to increase check-in efficiency by providing online check-in.	Business Requiremer	Medi um	Medium	Medium
Business Rules	PF42950		Folder			
2 - Application Requirements	PF42951		Folder			
User Requirements	PF42952		Folder			
Quality of Service Requirements	PF42953		Folder			
Functional Requirements	PF42954		Folder			

Figure 1 Business Context - Textual Business Needs, Business Process, Business Domain and Traceability

DEFINING EXPRESSIVE REQUIRMENTS

Textual Statements

As with the business context, I typically begin with textual statements describing application requirements. I mostly use a categorization scheme and traceability espoused by rational unified process (RUP),which originated at HP, grouping requirements into five main categories: functional, usability, reliability, performance, and supportability. Each of these has numerous sub-categories which some people use and others don't.

Another popular approach is that promoted by Karl Weigers in his book software requirements. This approach arranges user requirements, business rules, quality attributes, system requirements, functional requirements, external interfaces, and constraints into a hierarchical traceability strategy. As with categorization of business needs, there are numerous taxonomies for categorizing software requirements, along with many strategies for tracing the relationships among them, and you need to decide on one that's appropriate for you. Regardless, this is simply a way to organize your textual statements, and does little to improve their expressiveness.

Example legal-style text requirements list:

- The system shall allow the user to book a flight
- The system shall allow the user choose the departure and destination airport
- The system shall allow the user to specify multiple passengers
- The system shall allow the user to specify for each passenger whether they are an adult, senior, or child.
- The system shall allow the user to select either a one-way flight or a round-trip.

Think Like a User

One of the most popular ways to improve expressiveness is to "think like a user" and imagine how they'd use application. User stories are one way to express this. Another is use cases, which I'll discuss briefly here.

Instead of a "list" of capabilities or features to be provided by the application, use cases describe the various scenarios of how the application will be used to accomplish some goal. As an example, imagine an online airline application for booking flights and other common traveler functions. Consider a couple of scenarios:

1) Book a round-trip from Dallas to Chicago for two adults
2) Book a one-way flight from Toronto to Atlanta for one adult, and one senior

These are different scenarios and will likely result in some different requirements. For example, in the first scenario, we actually have to handle two flights instead of one, because it's a round-trip. In the second, there may be special requirements for seniors and perhaps senior discounts involved. Furthermore, the second example is an international flight, so there's likely some special requirements around that as well. However you can look at these two scenarios, and probably think up several more, which are all variants of "booking a flight", the goal of the user. To describe these various scenarios, we would create a use case called "book a flight", and within it stipulate in detail how flights get booked.. We do this in a dialog style such as ...

1) Chooses to book a flight
2) System allows user to choose one-way or round-trip
3) User chooses round-trip
4) System allows user to specify departure location and destination
5) User chooses departure location and destination
6) System allows user to specify number and type of travelers as adult, child, or senior
7) Etc.

This is a quite different style than the legal-style text list. This dialog describes the same functionality, but from an alternative perspective and style. Just having this alternative vantage point can itself shed new light on the requirements, help communicate what is truly needed, and help expose hidden issues. Most find that this form is more "expressive" than the legal-style text list. First, it is expressed from the perspective of the user which will allow existing and future end-users to consume it more readily. Further, notice that each statement makes sense only in context of those around it. If you move step 4 to the end of the list, it will make no sense. In this style, it is quite easy to spot when things are missing or out of place, helping ensure completeness and integrity of the requirements.

Visualize the User Interface

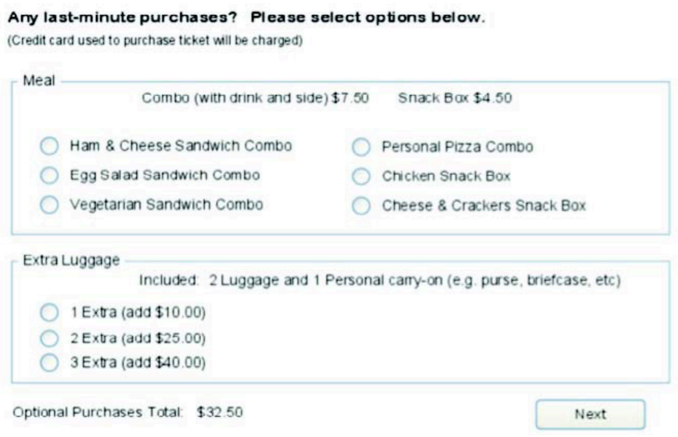
Take a look at the requirements for an application screen below:

- The screen must allow users to select meals for optional purchase
- The screen must allow the user to purchase up to three extra luggage pieces
- The screen must allow for the display of six different meals
- The screen must display the totals of any and all optional purchases
- The user must be able to select and re-select any options on the screen as many times as desired before submitting
- The screen must prominently display the standard luggage allowance
- The screen will inform the user at all times that their credit card used for ticket purchase will be billed for optional meal purchases
- The screen must prominently display the price for meals

These are just a few of the requirements that could be specified for a particular screen. These requirements don't even get into the various constraints on layouts and appearance that are commonplace when specifying screens.

In the use steps shown earlier, notice that the system steps describe many things that the screen will need to support. Instead of relying exclusively on interpretation of this written text, what tends to be far more effective is to provide a screen mockup as shown in Figure 2.

While some requirements authors dispense with the written requirement entirely, most generally favor maintaining some combination of textual statements and UI mockups.


Any last-minute purchases? Please select options below.
(Credit card used to purchase ticket will be charged)

Meal
Combo (with drink and side) $7.50 Snack Box $4.50

- Ham & Cheese Sandwich Combo
- Egg Salad Sandwich Combo
- Vegetarian Sandwich Combo
- Personal Pizza Combo
- Chicken Snack Box
- Cheese & Crackers Snack Box

Extra Luggage
Included: 2 Luggage and 1 Personal carry-on (e.g. purse, briefcase, etc)

- 1 Extra (add $10.00)
- 2 Extra (add $25.00)
- 3 Extra (add $40.00)

Optional Purchases Total: $32.50

Figure 2 UI Mockup

A picture definitely is worth a thousand words, and this approach to "visualizing" what's needed is indeed quite powerful. Supported by various visual requirements tools available today, there are some very effective approaches in practice. Just a sampling:

Wireframes: simple wireframe mockups of user interfaces, like the one above, are fast and easy to create. Their simplicity means that they often can be updated in real-time during a review session. Another good thing about them is that they don't look like real screens, so people don't get confused and think that the wireframes represent a finished product. Keeping the fidelity low, at the level of wireframe, also provides silent guidance to the requirements author to stay at the level of screen concept, and not stray into designing the actual screen.

High fidelity screens: more richly defined screens that actually resemble the end product are at times warranted. Sometimes this happens in high-risk or complex areas of the application and often in collaboration with a user interface designer.

Screen markups: very common in situations where you're enhancing an existing application, taking a screen-shot and applying markups and annotations is an effective way to quickly communicate what's needed, and to highlight subtle points.

Screen overlays: similarly powerful in situations where you're enhancing an existing application, screen overlays take a screen shot of the current application, and add screen controls that represent the enhancement. Modern requirements tools will provide an editor allowing you to do this, and a simulation engine allowing those overlays to become live during a requirements simulation.

System interfaces: in areas where there's no user interface, there's still room for visualizations. Event or sequence diagrams are excellent visuals for illustrating behavior of a system interface.

Leverage UI Storyboards

Any or all of these types of user interface mockups can then be strung together in a sequence to emulate a scenario through the future user interface. Storyboards are a technique borrowed from the motion picture industry and are very powerful as they help people 'experience' how the future application will behave - in other words, how it will respond to user input. In doing so, they are a tremendous aid to help expose hidden issues, errors, and omissions that would otherwise have gone undetected.

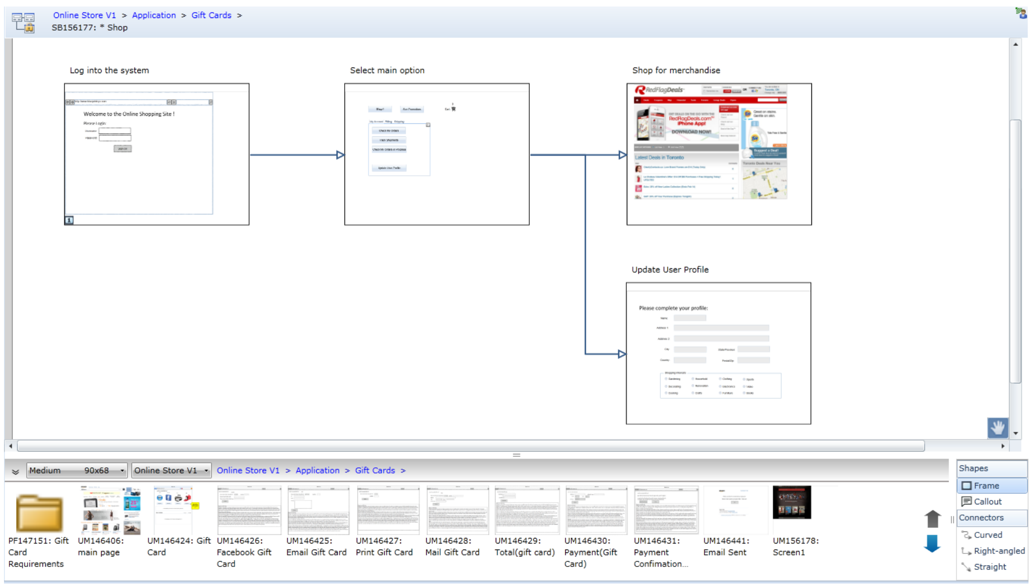

Online Store V1 > Application > Gift Cards >
SB156177: * Shop
Log into the system
Select main option
Shop for merchandise
Update User Profile
Medium
90x68
Online Store V1
Online Store V1 > Application > Gift Cards >
PF147151: Gift Card Requirements
UM146406: main page
UM146424: Gift Card
UM146426: Facebook Gift Card
UM146425: Email Gift Card
UM146427: Print Gift Card
UM146428: Mail Gift Card
UM146429: Total(gift card)
UM146430: Payment(Gift Card)
UM146431: Payment Confimation...
UM146441: Email Sent
UM156178: Screen1
Shapes
Frame
Callout
Connectors
Curved
Right-angled
Straight

Figure 3 UI Mockups in a Storyboard

Leverage UI Mockups with Use Cases

Whereas a storyboard tends to focus purely on the user interface behavior and generally covers one or maybe two scenarios, a use case focuses more on the goal that's trying to be accomplished and covers most if not all the possible scenarios. So use cases are more complete, more detailed, and more exhaustive. Storyboards are often done first as a stepping-stone to full use cases.

The UI mockups created, and possibly tuned using some quick storyboards, can be integrated with the use cases to provide an even deeper more comprehensive picture of the future system and its behavior.

While this could be accomplished manually by perhaps creating links between your use case steps and the visuals with office-automation applications, a far more effective solution is to use a modern requirements definition tool that supports this association, and provides other related benefits. The most powerful of these tools allows you to have a mixture of visualization approaches associated with the use cases, as illustrated in Figure 4.

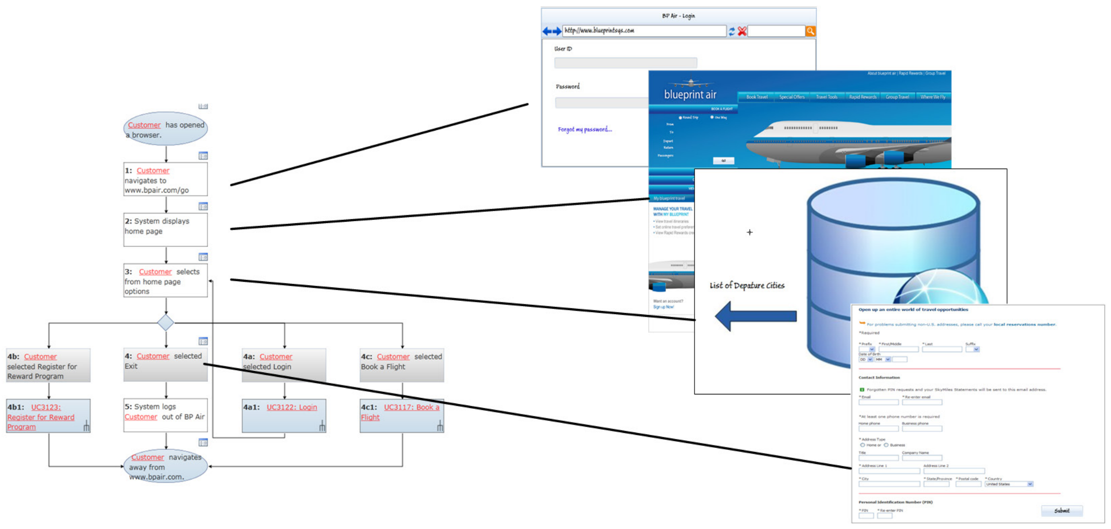

Customer has opened a browser.
1: Customer navigates to www.bpair.com/go
2: System displays home page
3: Customer selects from home page options
4b: Customer selected Register for Reward Program
4: Customer selected Exit
4a: Customer selected Login
4c: Customer selected Book a Flight
4b1: UC3123: Register for Reward Program
5: System logs Customer out of BP Air
4a1: UC3122: Login
4c1: UC3117: Book a Flight
Customer navigates away from www.bpair.com.
BP Air - Login
User ID
Password
Forgot my password...
blueprint air
List of Departure Cities
Submit

Figure 4 Use Cases integrated with UI Mockups

BRINGING IT TOGETHER

So there are multiple ways you can express requirements. Of these, textual lists are typically the least expressive. As discussed earlier, I tend to use these alternate forms and techniques to augment rather than replace text lists. However, I typically don't use them on all parts of the application.

One criterion that tends to guide how I express requirements is requirements complexity; successfully communicating highly complex requirements demands these more expressive forms. Another criterion is risk. If the risk associated with getting the requirements wrong in a certain area of the application is particularly severe, I want to be as expressive as possible with the requirements, thereby reducing the possibility of miscommunication.

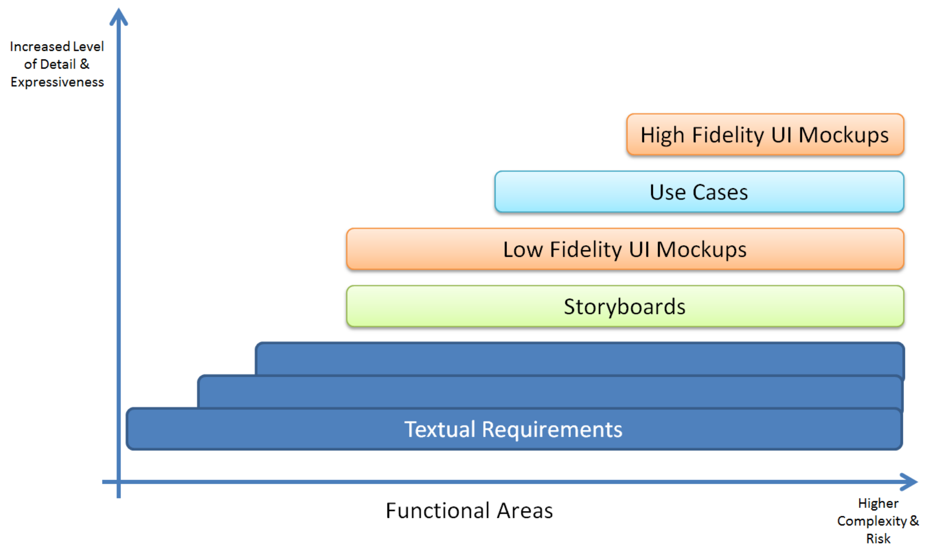


Figure 5 Be More Detailed and Expressive in areas of increased Risk and Complexity

While all areas of the application are covered by textual requirements at a high-level, I go to the greatest level of detail in those areas that are complex and risky. Wireframe UI mockups and Storyboards may be used to express much of the functionality. For more complicated areas Use Cases may be used to increasingly greater levels of detail, evolved from earlier storyboards if they were created. And for the areas of highest risk and most complexity high-fidelity UI mockups may be leveraged.

While it is possible to do this manually with spreadsheets, drawing tools, and other point-solutions, this is where an integrated requirements workbench really shines in its ability to not only support all these forms of expression, but also to maintain requirements integrity, ensure consistency across them, and manage traceability. The diagram below gives a high-level overview of a traceability strategy for this information, including traceability back to the business context.

Tracing back to business reference is vital to ensure that all business needs are being addressed. Conversely, a complete traceability strategy ensures that every application functionality that is being specified has an explicit business need.

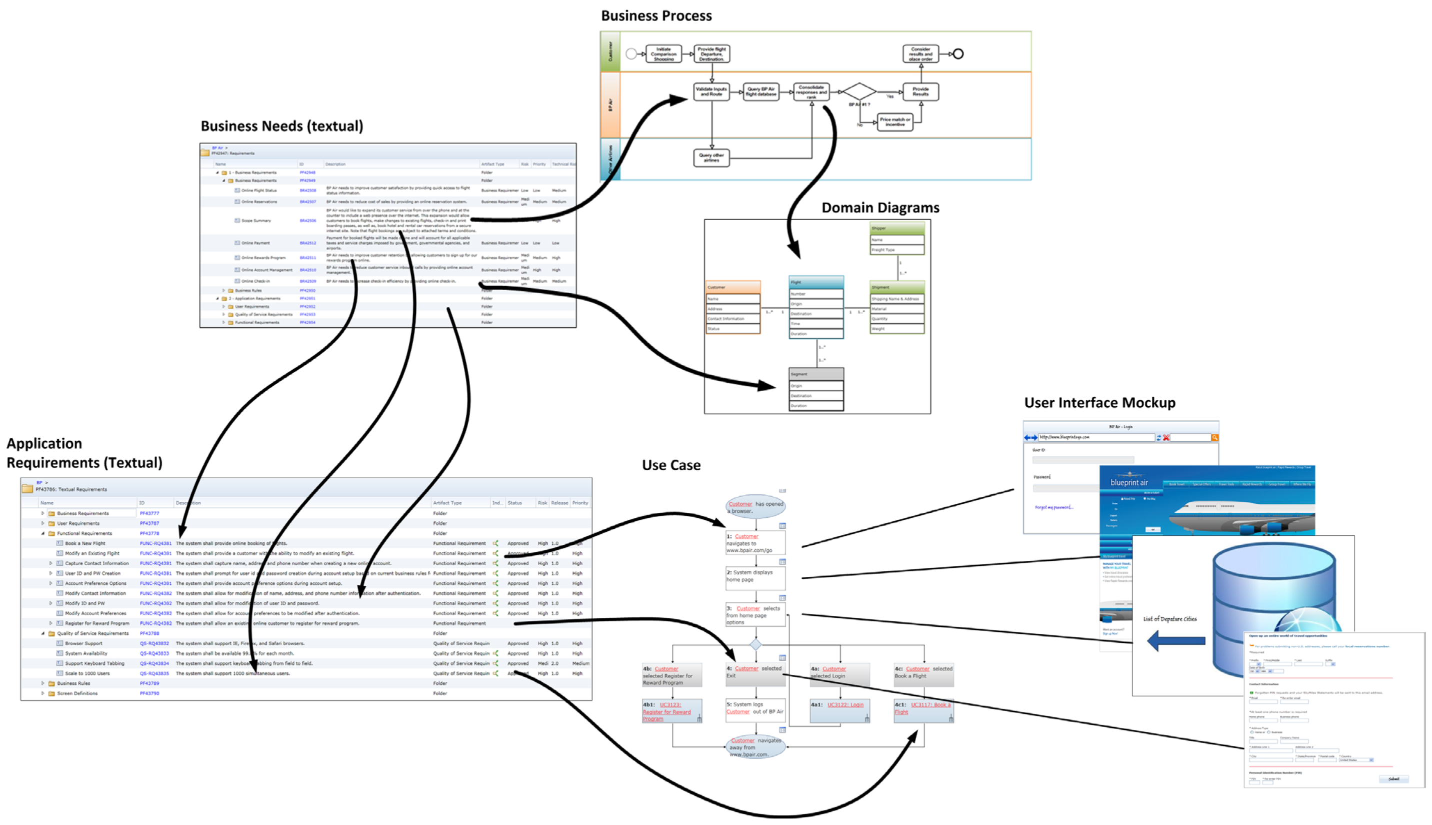


Figure 6 Business Context with Expressive Requirements

LEVERAGING EXPRESSIVE REQUIREMENTS WITH BLUEPRINT

Defining expressive requirements can dramatically improve the outcome of software projects by uncovering latent errors early and reducing miscommunication. Expressive requirements can leveraged to their fullest by using Blueprint, a modern and comprehensive requirements definition and management platform. The following are just a few examples of the benefits from using expressive requirements in Blueprint.

Automatically Generate Requirements Documents

Blueprint will allow you to generate requirements documents automatically from the requirements information defined using it. This can be a significant shift for a more traditional organization in that time is spent focused on the requirements themselves, making them more expressive, analyzing them and improving their quality as opposed to being focused on the mechanical tasks of producing and maintaining a requirements document. You still get the document, just without the huge overhead.

Automatic Simulations

Blueprint transforms requirements into an animated, live simulation. Simulations with expressive requirements make reviews far more effective and engaging. These simulations leverage all the requirements content integrating it into a single “vision” of the future application. Traceability is a vital part of the review process. When reviewing requirements, you must be able to relate them back to the original business need on-demand. Blueprint exposes this traceability not only during requirements authoring, but also during the simulations that are used for analysis and review.

Comments and discussions often result from simulation sessions. These are useful exchanges of ideas that also need to be recorded and managed so they’re not missed, and that you can always refer back to them. Blueprint records this informal input alongside the requirements during simulation and makes it accessible throughout the product so people are aware of these discussions and so authors can make changes based on them.

Automatically Generate Tests

Blueprint lets you to automatically generate tests from the requirements. These tests will cover all possible usage scenarios throughout the model and express for each step in the test any relevant screens, data, or externally referenced materials. The importance of traceability continues through to the tests as well. Each test generated has complete traceability information to identify the requirements it helps prove, which is hugely valuable to the QA group.

Integrations with Lifecycle Tools

Practitioners who “consume” the requirements such as developers and testers base their work products ultimately on the requirements. It’s therefore very important that the requirements information be available within their toolsets as well. Blueprint integrates with popular lifecycle toolsets to provide them with high-quality, expressive requirements content.

Managing requirements change

With more expressive requirements most change occurs at the beginning of the project when changes are inexpensive. To make requirements change effective and managable Blueprint supports complete traceability among all requirements artifacts, provides mechanisms to create relationships that are fast and easy, makes traceability information available as you work, and provides sophisticated and filterable views into traceability for deeper analysis. Traceability is very important for analyzing impact as part of requirements change. Also, Blueprint versions all requirements elements allowing you to go back in time to see them as they existed at points in the past. The complete version history is maintained, detailing who changed what, and when they did it for each and every change. Finally, Blueprint provides baselines that give a reliable and stable foundation for change. Baselines in Blueprint can be full, encompassing all requriements on the project, or partial, letting you select specific requirements.

Summary

The key problem for software requirements has been poor requirements. One of the main reasons for poor quality requirements is miscommunication owing to a lack of requirements expressiveness. Multiple forms of expression for requirements can improve communicatiion while integrating these multiple forms allows them to be manageable and scalable. Blueprint makes this approach for practical, expressive requirements possible.

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