Chapter 4 and follow up
Author: Rick Lukowski and Others
Compiler and Editor: Thomas W. de Boer

Freedom Post 4.0 -- Functionality Trees

Stimuli Recap

Freedom organizes stimuli in two ways: (1) by grouping stimuli into stimulus sets based on functional, physical, and temporal cohesion, and (2) arranging the stimulus sets into a tree-like structure called a functionality tree. In this post we will look at the format of a functionality tree and the criteria used to arrange the stimulus sets into a tree-like structure.

Arrangement Criteria

One of the possible responses to a stimulus is to change the activation status of one or more stimuli. For example, entering a correct password (a stimulus) may cause the username and password entry fields (two stimuli) to be deactivated, and may also cause a menu (a set of additional stimuli) to be activated. Freedom calls this type of response a "New Stimulus Response" (NSR) The term is a bit of a misnomer because NSR also encompasses deactivation of existing stimuli as well as activation of new stimuli.

I suspect the name "New Stimulus Response" was coined when the functionality tree notation was being developed because it is in that context that the term is most descriptive. When stimulus sets are arranged into a tree-like structure, the criteria used to arrange them is the New Stimulus Response, specifically, the activation of additional stimulus sets. Stimulus set activation is the only aspect of the NSR depicted in a functionality tree; stimulus deactivation is specified in behavior tables.

Functionality Tree Syntax and Semantics

The format of a functionality tree is basically a horizontal hierarchy with the levels arranged in columns. The "root" is in the leftmost column and the "leaves" are in the locally deepest rightmost columns. The depth of the hierarchy may vary.

Functionality trees may be created using simple text editors that use fixed-pitch fonts (e.g., tty or courier), or spreadsheets, as such tools maintain the column alignments while also making it easy to insert new rows. The following is a sample generic functionality tree. Note that your reader must use fixed-pitch fonts for the example to appear properly; proportional pitch fonts will destroy the column alignment. If this happens, try viewing the html version of this post at

Sample Functionality Tree

Level 0	Level 1	Level 2
SS 0-A	SS 1-A	
--------	--------	
stim 0-1	stim 1-1	
		SS 2-A
		---------
	stim 1-2	stim 2-1
		stim 2-2
		stim 2-3
		stim 2-4
	SS 1-B	
	--------	
stim 0-2	stim 1-3	
	stim 1-4	
	stim 1-5	
stim 0-3		

Note: "SS" means "stimulus set"
"stim" means "stimulus"

In the above simple example, the external interface consists of four stimulus sets, labeled generically as SS 0-A, SS 1-A, SS 1-B, and SS 2-A. (Note that in a real example, the stimulus sets and stimuli would have names that are semantically meaningful to the application. However, a functionality tree does not care what the names are, so generic names are used here for illustrative purposes only.)

The functionality tree shows that the four stimulus sets consist of the following stimuli:

SS 0-A	SS 1-A	SS 1-B	SS 2-A
------	------	------	------
stim 0-1	stim 1-1	stim 1-3	stim 2-1
stim 0-2	stim 1-2	stim 1-4	stim 2-2
stim 0-3		stim 1-5	stim 2-3
			stim 2-4

The tree structure of the functionality tree depicts the activation of the four stimulus sets, as follows.

1. When the application is first launched, only the stimuli shown at Level 0 are active. Hence, SS 0-A is the only stimulus set active when the program is launched.

2. If stim 0-1 is received, part of its externally-visible response is to activate SS 1-A.

3. If stim 0-2 is received, part of its externally-visible response is to activate SS 1-B.

4. If stim 0-3 is received, it does NOT activate any additional stimuli.

5. If stim 1-2 is received, part of its externally-visible response is to activate SS 2-A. Note that stim 1-2 cannot be received until stim-0-1 is received. (Why?)

Thus, the semantics of a functionality tree defines:

1. All stimulus sets
2. The stimuli that comprise each stimulus set
3. Which stimuli activate which other stimulus sets, i.e., partial New Stimulus Set response.

The semantics of a functionality tree does NOT specify:

a. Stimulus deactivatation
b. Stimulus activation in addition to the indicated stimulus set(s)
b. Any response other than stimulus set activation.

All of these semantics, and more, are the responsibility of the behavior table notation.

Functionality Tree Genericity

The format of a functionality tree is independent of the source of the stimuli. That is, the stimuli may originate with human users, external system users, and/or the environment. A functionality tree looks the same regardless of where the stimuli originate.

Likewise, a functionality tree is independent of the technology used to detect stimuli -- GUI event handling, network signal detection, environment sensors, etc make no difference to a functionality tree.

As a result, a functionality tree is a notation for specifying stimuli and their organization in a implementation-independent manner. In this respect, a functionality tree is analogous to an electrical schematic. Both depict essential characteristics of a design without specifying the details of the implementation. Hence, a functionality tree serves much like an "external interface schematic."

Example AAR Functionality Tree

In case the generic example above created more questions than it answered, we will not look at part of a functionality tree for a real program. To do this, we will use our sample application, Adobe Acrobat Reader (AAR), and reverse engineer a small part of its functionality tree for explanatory purposes.

Recall from Post 3 that the uppermost menu bar containing the stimuli File, Edit, Document, View, Window, and Help was a stimulus set, For lack of a better name, let's call this stimulus set "Main Opts". Since it is active when AAR is first launched, Main Opts appears in the leftmost (Level 0) column of the AAR functionality tree:

AAR Functionality Tree

Level 0

Main Opts

File
Edit
Document
View
Window
Help

Note that other stimulus sets are also active at start up. These would also appear in the Level 0 column in the full AAR functionality tree. In order to keep the example simpler (and this post shorter), we will ignore these other stimulus sets and focus only on Main Opts.

When the File stimulus is activated by clicking it with the mouse, a popup menu appears. This popup menu is a new stimulus set, and its menu items are new stimuli. Let's call this new stimulus set "File SS" (short for File Stimulus Set) since it is a stimulus set activated by the File stimulus. File SS contains the stimuli Open, Close, Save A Copy, Document Properties, Document Security, Page Set Up, Print, history list, and Exit. Since File SS is activated by a Level 0 stimulus, File SS appears in the functionality tree (FT) at the next level down (Level 1):

AAR Functionality Tree

Level 0	Level 1
Main Opts	File SS
---	---
File	Open
	Close
	Save A Copy
	Document Properties

Document Security
Page Set Up
Print
history list
Exit
Edit
Document
View
Window
Help

The fact that File SS is activated by the File stimulus is denoted by placing the first stimulus of File SS (Open) immediately to the right of the activating stimulus (File).

Examining the stimuli in File SS reveals that Document Properties activates yet another stimulus set consisting of three stimuli. This stimulus set, which we will call Document Properties SS, appears at the next level down (Level 2) from its activating stimulus (Document Properties):

AAR Functionality Tree

Level 0	Level 1	Level 2
Main Opts	File SS	
---	---	---
File	Open	
	Close	
	Save A Copy	
		Document Properties SS

	Document Properties	Summary
		Fonts
		Embedded Data Objects
	Document Security	
	Page Set Up	
	Print	
	history list	
	Exit	
Edit		
Document		
View		
Window		
Help		

Experimenting with Document Properties SS stimuli reveals that Summary and Fonts both cause a popup window to be displayed. These windows are primarily for information display. If all they contained was displayed information with which the user could not interact, they would not be stimulus sets because they would not contain any sources of stimuli. However, each of these windows contains a button to hide the window, and the Fonts window contains a second button to control the information displayed. The existence of these stimuli make these windows stimulus sets also.

Expanding the Document Properties SS to include the stimulus sets it activates grows the FT further:

AAR Functionality Tree

Level 0	Level 1	Level 2	Level 3
Main Opts	File SS		
---	---	---	
File	Open		
	Close		
	Save A Copy		
		Document Properties SS	Summary SS
		---	---
	Document Properties	Summary	OK
			Fonts SS

		Fonts	OK
			List All Fonts
		Embedded Data Objects	
	Document Security		
	Page Set Up		
	Print		
	history list		
	Exit		
Edit			
Document			
View			
Window			
Help			

We may now wish to consider that, if Summary and Font activated popup window stimulus sets, some of the stimuli at Level 1 may also. Further investigation reveals this is true. In fact, some of these Level 1 triggered popup windows are heavily laden with stimuli. For example, the window triggered by the Open stimulus has three buttons, two lists, and two text fields. Adding the stimulus sets activated by the other Level 1 stimuli fills the tree out much more:

AAR Functionality Tree

Level 0	Level 1	Level 2	Level 3
Main Opts	File SS	Open SS	
---	---	---	
File	Open	Directories list	
		Files list	
		Filter	
		Selection	
		Open	
		Apply	
		Cancel	
	Close		
		Save A Copy SS	

	Save A Copy	Directories list	
		Files list	
		Filter	
		Save As	
		Optimize	
		OK	
		Apply	
		Cancel	
		Document Properties SS	Summary SS
		---	---
	Document Properties	Summary	OK
			Fonts SS

		Fonts	OK
			List All Fonts
		Embedded Data Objects	
		Document Security SS	

	Document Security	OK	
		Page Set Up SS	Paper SS
		---	---
	Page Set Up	Paper	Letter
			Tabloid
			Ledger
			Legal
			Executive

		A3
		A4
		A5
		B4
		B5
		Other
	Scale	
	Width	
	Height	
	Portrait	
	Landscape	
	OK	
	Cancel	
	Print SS	

Print	Printer	
	Command	
	File	
	File Name	
		Browse SS

	Browse	Directories list
		Files list
		Filter
		Selection
		OK
		Apply
		Cancel
	- - - -	
	All	
	From	
	From page	
	To page	
	Odd	
	Even	
	Selected Thumbnails	
	Selected Graphic	
	Reverse Order	
	Comments	
	- - - -	
	Shrink...	
	Expand...	
	Optimize for Speed	
	Download Asian Fonts	
	Save Printer Memory	
	Language Level 1	

Language Level 2
Language Level 3
- - - -
OK
Cancel

history list ***EXERCISE***
Exit
Edit
Document
View
Window
Help

Functionality Tree Observations and Guidelines

If we were given the above AAR functionality tree without ever having seen it before or knowing how it was obtained, would we be able to answer questions such as:

1. Which stimuli are text fields?

2. Which stimuli are buttons?

The answer is -- we cannot answer such questions using the above functionality tree (try it). In the process of creating the functionality tree, we lost implementation-specific information. It is true that we could have captured some. For example, instead of listing a stimulus as

OK

we could have listed it as

OK Button

We did not do this because to do so would have violated a functionality tree guideline:

Make functionality trees implementation-independent

When creating a functionality tree, try to avoid using names that imply an implementation mechanism such as Button, Text Field, Choice List, etc. In the AAR FT we do use the term 'list', but only in a generic sense to mean a 'variable list of stimuli'. This use of 'list' does not imply a GUI List component or any other specific implementation mechanism.

By keeping functionality trees generic, they are better able to fulfill their role as a schematic of the external interface that identifies stimuli and their organization without specifying any

particular implementation. To press the point a bit further, the following two questions should be just as legitimate (and just as unanswerable) as the first two:

3. Which stimuli are XML tags?

4. Which stimuli are hardware toggle switches?

Nothing in the functionality tree should rule out these possibilities, nor be biased toward them. When functionality trees are created to be generic, the requirements of the system become more generic as well. They may describe a hardware system or software system. They may be implemented using archaic or bleeding edge interface technology. It is the job of notations such as functionality screens (sketches of the UI) and interface protocol descriptions to specify the implementation-specific aspects of the requirements, i.e., the "form" of the interface. The purpose of the functionality tree and associated behavior tables is to specify only the raw functionality, or "function," of the system without prejudice to either the source of the stimuli or implementation technology.

One advantage of making functionality trees generic is that it promotes focusing of the team's mental energy on the functionality needed by the customer without diverting excessive amounts of thought to detailed implementation issues. Of course, some thought must be given to the implementability of the functionality, e.g., we do not want to specify requirements that are excessively expensive, or even impossible, to implement. This means it is usually a good idea to start developing functionality screens (sketches) and protocol format descriptions in parallel with the functionality tree. The result of such parallel development can be a more robust and generic functionality tree. Any thoughts about how to implement the functionality can be recorded immediately in the proper notation, thus reducing any urge to place such information in the FT. The primary focus of the team early-on when the functionality tree is first being developed should be on the raw functionality required, not on specific protocols. The implementation protocols can be refined after the requirements architecture and functionality specified by of the FT become less fluid.

Another observation regarding functionality trees is that the notation is very amenable to change, including expansion. Note that in the AAR example we started small and easily grew the FT in increments by adding one or a few stimulus sets at a time. This approach works well in actual development as well. It is not necessary, or even possible in many cases, to create the "complete" FT at once. This is never a problem since functionality trees can easily be expanded to incorporate new requirements as they are identified, even if that turns out to be years later during maintenance (or evolution) of the system. Because it is so easy to change, a functionality tree is a convenient notation for recording requirements on projects that employ incremental and/or evolutionary development approaches.

Functionality Tree Exercise

I hope it is now apparent what a functionality tree is, and that a functionality tree can be derived by inspection of an existing external interface (although this is not the most common way of creating one).

As a self-check, I have left the 'history list' stimuli unexpanded so that you can give the technique a try for yourself. In order to expand the 'history list' part of the tree, you should first view several pdf files with AAR in order to build a list of names in the history list to experiment with. To do a more complete job of experimentation, try deleting or moving one of the pdf files (using operating system commands) after viewing it so AAR cannot find it when you click on its name in the history list.

If you think of any questions while expanding the 'history list' part of the tree, please ask. It is very possible that others in the group will have similar questions.

This exercise serves an additional purpose beyond a self-check. It is a lead-in to the the next post, Post 5. However, why this is so will not be apparent until you try it. Have fun!

Rick Lutowski

From Rick Lutowski
Any questions or comments on Post 4 on Functionality Trees? Has anyone tried the exercise at the end of the post?

Since the group has been silent since Friday, I do not know if those who have been following these posts have tried the Post 4 exercise, and are ready for Post 5 or not.

Please advise.
Thanks.

Rick Lutowski

From Philipe Back
Rick Lutowski wrote:
> Since the group has been silent since Friday, I do not know if those who have been following these posts have tried the Post 4 exercise, and are ready for Post 5 or not

Feel free to post it.

I started the thing but I am busy working on other proposals (trying to use what you mentioned on them, nothing is worth a live example !).

Philippe Back

From Rick Lutowski
Philippe Back wrote:
> Feel free to post it.

OK. Probably tonight, unless others want a day or so more to assimilate #4.

> I started the thing but I am busy working on other proposals (trying to use what you mentioned on them, nothing is worth a live example !).

Does the above mean you are trying to use some of these Freedom ideas on real work?

Rick Lutowski

From Philippe Back
Rick Lutowski wrote:
> Does the above mean you are trying to use some of these Freedom ideas on real work?

Yes it does. Shouldn't I ? :-p

Phil

From Rick Lutowski
Philippe Back wrote:
> Yes it does. Shouldn't I ? :-p

It depends on the type or degree of use you are contemplating.

Recall the 7 posts were identified as the minimum necessary to understand Freedom's approach to requirements encapsulation. Other topics not relevant to a basic understanding RE are not being covered in the posts, but should be considered when using Freedom in practice. Some of these include:

Additional requirements topics:
* Quality Requirements
* Functionality screens
* Protocol formats
* Requirements prioritization
* Interface mockups
* Sw Requirements Spec (if a formal document is desired)

Object design topics:
* Extended behavior tables
* Object model tables

If you can be more specific about how you are using Freedom, I could provide additional info, if needed. We can discuss offline if you feel that is more appropriate.

--
Rick Lutowski

From Philippe Back
Rick Lutowski wrote:

> If you can be more specific about how you are using Freedom, I could provide additional info, if needed. We can discuss offline if you feel that is more appropriate.

No problem discussing this online. Being able to work using good methods is a reason why customers want to work with me. They even outsource parts of their own project management at times.

So, to be clear:

What I use are the principles, work products and the process flow 'a la Parnas'.

I like the use of tables like you propose since they scale.

One missing part for all of this requirement biz is : moving from a "concept" for a product down to the "functionality tree". Usability factors have to be factored in. I don't even talk about shaping the product so that it really delivers a solution.

Of course, what I do is not labelled as 'Freedom' but takes inspiration from it.

BTW I like the fact that Freedom concerns itself with solution delivery only and let me work out the project management part (currently, I am leveraging concepts form PRINCE2 a lot - in an agile way of course, otherwise it would way too much).

I know that mixing and matching methods may look like "betraying their spirit" but I guess that I've got enough under my belt to find a satisfying way to handle the thing.

Phil

From: Rick Lutowski
Philippe Back wrote:
> One missing part for all of this requirement biz is : moving from a ‘concept’ for a product down to the ‘functionality tree’. Usability factors have to be factored in. I don't even talk about shaping the product so that it really delivers a solution.

I think there have been questions from others regarding this same issue. Now that post 4 on functionality trees has been sent, I can attempt a better answer.

As stated previously, there is no 'turn the crank' process for going from concept to requirements. This is true not only for requirements recorded using Freedom's functionality trees and behavior tables, but for requirements recorded using prose, use cases, or any other approach. However, getting the process started is usually easier with Freedom than with other approaches due to the nature of Freedom's definition of requirements.

Here is how one can get started with the Freedom approach:

1. Freedom assumes that the developers are not doing this in isolation, but are in dialog with a knowledgeable customer rep or domain expert.

2. Freedom organizes the stimulus sets (and hence the requirements) into the hierarchy of a functionality tree (FT). The 'root' of this hierarchy is the Level 0 column of the FT. That is where one should start -- at Level 0 of the FT.

Level 0 of the FT consists of the stimulus sets that are active on program start up. This leads naturally to a particular type of 'requirements kickoff' question that can be posed to the customer rep (who is available by assumption 1):

Requirements Kickoff Question 1 (to be answered by customer rep):

'What do you want the first screen to look like when the program first starts up?'

Some customers will have a very good idea of the answer to this question and will immediately draw a useful sketch (I have seen it happen). This sketch (which Freedom calls a 'functionality screen') can immediately be analyzed for stimuli and stimulus sets, as we saw with the AAR example, to begin the FT.

Other customers may not have thought about the question before, but will certainly understand the question and relate to it. Dialog between the developers and the customer, and/or between the customer rep and the users, can help define that first sketch to kickstart the process.

Kickoff Question 1 presupposes the app has a non-trivial human user interface. What about embedded apps that may have no human interface at all? What then?

There is a variation to the question that can be used in cases where most stimuli originate with external systems and/or the environment:

Requirements Kickoff Question 2 (to be answered by customer rep):

'What is the first thing the program should listen for when it first starts up?'

This is really the same question as Question 1, just posed in slightly different terms so the customer can better relate to it, and therefore better answer it.

If your customer just happens to be another developer who is conversant in the language of functionality trees and stimulus- response, the question could be phrased more generically as:

Requirements Kickoff Question (for FT-knowledgable customer rep):

'What are the first stimulus sets active on program start up?'

OK, assume one of the above questions prompts the customer to cough up the first stimulus set(s). We now have a start on Level 0 of the FT. What next?

The next step is to take each stimulus one at a time, and ask the customer:

'What should happen when this input is detected or provided?'

If the customer knows just what they want and starts rattling off a lot of detail, start capturing the response info in a behavior table immediately. However, pay special attention to statements like 'and this new window should appear' that indicate an NSR response. At this stage, what counts most is the NSR responses as these drive the FT down to the next deeper stimulus set activation level and thereby identify additional stimuli.

In most cases, it won't take long before the customer concludes you are trying to design the UI. Actually, you are trying to identify and organize stimuli, but no point in confusing them with subtle distinctions (Note 1). Soon you and they will likely be neck deep in a big UI design session. Go for it! :-)

Note 1: I have found that it is perfectly safe to tell customers that defining the interface is equivalent to defining the requirements. They never bat an eye. It's mainly developers and methodologists that get bent out of shape over such statements. (Have seen it happen.)

> Of course, what I do is not labelled as 'Freedom' but takes inspiration from it.

Nice to know it has inspirational value. Thanks!

> BTW I like the fact that Freedom concerns itself with solution delivery only and let me work out the project management part (currently, I am leveraging concepts form PRINCE2 a lot - in an agile way of course, otherwise it would way too much).

> I know that mixing and matching methods may look like 'betraying their spirit' but I guess that I've got enough under my belt to find a satisfying way to handle the thing.

One of the nice things about consulting is the freedom to 'roll your own' methodology based on evaluation of best practices plus personal preferences. I'd say more like methodology 'mining' rather than 'betrayal.'

Once you have tried some of Freedom's concepts and techniques, I'd be interested to know which work best for you, which don't, and why.

Thanks for the feedback.

-- Rick Lutowski

From: Philippe Back
Rick Lutowski wrote

> One missing part for all of this requirement biz is : moving from a ‘concept’ for a product down to the ‘functionality tree’. Usability factors have to be factored in. I don't even talk about shaping the product so that it really delivers a solution.

This note is coming right on! Today, I facilitated a ‘product shaping’ session, with marketeers, IT people and management types. For sure, they don't care about the terminology.

But we have baselined the features, got a roadmap and identified the product families (because sometimes, people mix multiple ongoing projects together and then the fun begins when it's time to invoice work and get paid).

Now, I will be able to apply the functionality trees a bit, since we have a ‘stabilized feature set’.

I also agree that for users, the interface *is* the application. If the system was a clock, they want to be able to read the hour and not knowing about all of the gears inside... There definitely are two communication flows from my perspective: one between the customer and the technical team manager and another between the team manager and the team itself (including subcontractors)... They have different modeling needs. ‘Modeling with a purpose’ is then very useful. I would definitely avoid mixing the two... otherwise, some people will end up being spaced out and other roll their eyes up in the same meeting :-)

BTW, what do you use to manage the functionality trees ? An outliner ? Maybe I should give it a shot with Tcl/Tk. Or... what about using Freedom to design a ‘Freedom support tool’ ? :-)

Thanks again for your advice, the technique appears to work. Let's apply it for a while and see if it stands the test of fire. (BTW, would you mind putting all of these ‘long’ posts to the list of core posts ? Maybe as ‘extras’ to be reached with a link to the bottom of the page ?)

Philippe Back

From: Steven Gordon
Philippe Back wrote
> I also agree that for users, the interface *is* the application. If the system was a clock, they want to be able to read the hour and not knowing about all of the gears inside... There definitely are two communication flows from my perspective: one between the customer and the technical team manager and another between the team manager and the team itself (including subcontractors)... They have different modeling needs. ‘Modeling with a purpose’ is then very useful. I would definitely avoid mixing the two... otherwise, some people will end up being spaced out and other roll their eyes up in the same meeting :-)

I like the clock example. I totally disagree with the conclusion.

The user wants more than to just be able to read the time. The user wants to read the correct time, but if we only look at the user interface, this fact might be lost. You may say the user never has to specify what time they want to see on their clock because we can assume he

wants the correct time, but it is not as obvious as it appears. One user might want us to ask a web site or a satelite for the exact time every second. Another might be quite happy to be able to set the clock and let it run a little fast or slow. Two different users who accepted the same interface might have completely different expectations.

Then after using the first version of the clock, they might conclude they need alarms, the ability to switch between time zones, and/or other functionalities that add additional interfaces. Those new requirements change the meaning of the interface they already have (possibly without changing the formal interface itself at all - in which case we have a prime example that the interface is not identical with the requirements).

Of course, in a system that is much larger and complex system than a clock, the semantic misinterpretations created by defining the interface and not digging deep enough to make sure that everyone is interpretting the functionalities underlying the interface the same way becomes a large source of failure.

So, the developers need to know what the users really means to build the right product. Purposely making a manager or anyone else a bottle-neck in the communication between users and developers is creating a large single point of failure. The developers need to work directly with the users.

Steve

From: Philippe Back
Steven Gordon wrote
> I like the clock example. I totally disagree with the conclusion.

Okay.

Having been a developer (some may say geek) for quite a while (believe me, my first wife can testify), I guess that I can convey information quite effectively to developers. I still code on several projects (not C++ as before but in Java/PHP/TCL and VB Script) and this helps staying realistic.

On the other hand, I love marketing and sales and can relate quite well to customers and ‘USP’s and things like ‘values’, ‘tone of voice’, ‘campains’...

So, in that light, I guess that I can shape & baseline a solution that will be well received by the ‘stakeholders’. Having banged my head against some heavy walls in the past may also have contributed to being more ‘flexible’ and ‘soft’ than in the first years of cours. Reselling products is also a good step in seeing what works and what does not.

So, I agree that you are right to disagree but I would have shown a ‘vision of *the* watch they would love to have for their current budget’ instead of ‘just a watch’.

Phil

From: Rick Lutowski <rick@jreality.com>
Philippe Back wrote:
> This note is coming right on! Today, I facilitated a 'product shaping' session, with marketeers, IT people and management types. For sure, they don't care about the terminology. But we have baselined the features, got a roadmap and identified the product families (because sometimes, people mix multiple ongoing projects together and then the fun begins when it's time to invoice work and get paid).

One more reason Freedom is just as happy leaving management to some other methodology! ;-)

> Now, I will be able to apply the functionality trees a bit, since we have a 'stabilized feature set'. I also agree that for users, the interface *is* the application. If the system was a clock, they want to be able to read the hour and not knowing about all of the gears inside...

That's it exactly.

> There are definitely are two communication flows from my perspective: one between the customer and the technical team manager and another between the team manager and the team itself (including subcontractors)... They have different modeling needs. 'Modeling with a purpose' is then very useful. I would definitely avoid mixing the two... otherwise, some people will end up being spaced out and other roll their eyes up in the same meeting :-)

Right. I have also been on a project with just that structure, where the team lead acted as liaison. That project didn't use Freedom (I was just a coding grunt called in to fight the usual crisis-mode fire), but I know the scenario.

Speaking strictly to Freedom's notations, I would suggest using functionality screens, file format descriptions, and similar depictions of the interface protocol to communicate with customers and users. Since those are the formats they will actually see and deal with, they should relate to those notations best.

For comm within the team, FTs and BTs are the notations of choice. They record the user view protocols in a format which maps directly to implementation of functionality modules (as we will see in post 7) and kickstarts the design for common service modules.

Unfortunately, functionality screens and other protocol descriptions only cover stimuli. For responses, BTs (behavior tables) are the best solution I know. Not everyone on the customer side may relate to them well, but not everyone has to. If there is one person on the customer side who can read comment-like or PDL-like phrases in a table, that is adequate (if not a bonus). In a larger meeting with users, managers, etc, responses can be verbally talked out, and the developers (and savvy customer rep, if available) can record the discussion results in the table. This is not much different effort-wise than capturing the same info in prose lists or paragraphs, except the context and formatting of the BT has quite a bit of value-added

compared to equivalent prose lists or paragraphs. Hopefully, why this is so will become clearer in post 6.

> BTW, what do you use to manage the functionality trees ? An outliner ? Maybe I should give it a shot with Tcl/Tk. Or... what about using Freedom to design a 'Freedom support tool' ? :-)

I personally have always used a simple text editor, and my FTs appear just as they did in post 4. They are surpassingly easy to manage with a text editor; one almost has to try it to realize how well-suited the format is to a plain old text editor. However, I do have a distinct bias in that direction due to being a die-hard Linux/vi user.

Most everyone else who has used FTs and BTs seem to prefer using a spreadsheet. That works too; it's just a matter of which type of tool you are most comfortable with. Any tool that can maintain table column alignment should work.

The idea of specialized tools to create and maintain FTs, BTs, OMTs, etc has been on my mind for years. At one point, one of the Freedom Ship software team members started work on a tool to create functionality trees. Of course, the first thing he did was create the functionality tree for the tool. The result was a functionality tree for a functionality tree tool! His FT was really something. (I use it, with his permission, as an example in the Freedom requirements course.) However, that is as far as he got.

The single biggest hole in Freedom is 100% lack of specialized tool support. Fortunately, most of the notations are very amenable to simple commodity tools like editors and spreadsheets, so lack of specialized tools is not a show stopper. They sure would be nice to have, tho. Their development would also provide some interesting examples of Freedom 'eating its own dog food.'

> Thanks again for your advice, the technique appears to work. Let's apply it for a while and see if it stands the test of fire.

Very much look forward to your report of how it works out.

> (BTW, would you mind putting all of these 'long' posts to the list of core posts ? Maybe as 'extras' to be reached with a link to the bottom of the page ?)

Actually, Thomas de Boer of this list has volunteered to do something like that. He is working on consolidating the discussion for the first couple of posts now. The results of his efforts will be on the web site eventually. He and I appreciate your patience.

Thanks.

Rick Lutowski

From: Rick Lutowski
Steven Gordon wrote:
> I like the clock example. I totally disagree with the conclusion.
> The user wants more than to just be able to read the time. The user wants to read the correct time, but if we only look at the user interface, this fact might be lost. You may say the user never has to specify what time they want to see on their clock because we can assume he wants the correct time, but it is not as obvious as it appears. One user might want us to ask a web site or a satelite for the exact time every second. Another might be quite happy to be able to set the clock and let it run a little fast or slow. Two different users who accepted the same interface might have completely different expectations.

These are all info examples at a greater level of detail. Freedom's definition breaks such info into two piles -- external (requirements) and internal (design/implementation, or D&I):

read the time	-- external (human UI)
ask a web site or a satellite	-- external (interface to external system)
set the clock	-- external (human UI)
let it run a little fast or slow	-- external (accuracy is externally detectable)

So all the above are valid requirements by black box definition. The fact they are not all part of the UI is not definitive -- the external interface is much more than just the human UI (did I ever mention that before..?) The external interface includes human, external system, and environment components, which correspond to different sources of stimuli.

Also, the fact some of the above are not stimuli ('let it run a little fast or slow') is not definitive -- the interface is defined not just by (1) stimuli or inputs, but by (2) responses or outputs, and (3) protocols. 'Let it run a little fast or slow' is an externally measurable response.

> Then after using the first version of the clock, they might conclude they need alarms, the ability to switch between time zones, and/or other functionalities that add additional interfaces. Those new requirements change the meaning of the interface they already have (possibly without changing the formal interface itself at all - in which case we have a prime example that the interface is not identical with the requirements).

alarms	-- externally detectable
switch between time zone	-- external (stimulus to control switching)
functionalities that add additional interfaces	-- external ("interfaces")

These are all requirements as well by the black box definition.

None of this refutes Freedom's approach, or offers a contradiction in any way as to why an external interface specification cannot serve as a requirements spec. I certainly see no 'prime' example otherwise. If anything, you have only further demonstrated the validity of Freedom's approach.

If you see it another way, then you are probably missing something important. Too narrow an interpretation of 'interface' is a likely cause. 'Interface' does not mean just human interface, or just 'visible' in in a literal 'light wave' sense. Interface includes externally visible (detectable) stimuli, responses, and protocols to humans, external systems, and the environment.

Everything mentioned in the example above is covered under this black box definition of requirements.

Going beyond the above example...

If the customer specifies something internal like 'the clock shall be constructed using gears and springs', that would NOT be requirements but D&I info. However, Freedom can still record such info as binding on the development team, to the same degree that requirements are binding, by specifying such internals as 'D&I constraints.' D&I constraints are always recorded in behavior tables (response specifications), not in the functionality tree or protocol specs (stimulus specification). Only considering stimuli and protocols results in an incomplete picture. Perhaps this is the source of your misunderstanding and consequent disbelief.

Since we have not yet reached post 6 on behavior tables, this is a plausible explanation. Please try to reserve judgment until all the posts are in, and all subsequent questions have been clarified. If you can identify any serious deficiencies in Freedom's approach after that:

a. I will attempt to rectify it (like any other software, Freedom is subject to evolution.)

If it is a fatal deficiency that defies rectification, then

b. You will have likely proved Mills' box-based model of software invalid or unworkable.

That in itself would make a noteworthy paper at some conference.

Beyond that, the deficiency may hint at a better underlying model than the box-based model. If we can work together to clarify what that new model should be, some very significant progress will have been made -- perhaps more than Freedom currently purports to make.

So any way one cuts it, this is a no-loose situation. Obviously, I don't think (a), let alone (b), is going to happen. But I welcome your objective skepticism, and do not fear the consequences of any objective skepticism being correct. The most likely result would either be a further strengthening of Freedom or, perhaps, something better than Freedom.

> Of course, in a system that is much larger and complex system than a clock, the semantic misinterpretations created by defining the interface and not digging deep enough to make sure that everyone is interpretting the functionalities underlying the interface the same way becomes a large source of failure.

Interpretation, or semantics, is the key to your point here. The semantics of the interface is captured primarily in the behavior tables, which we have yet to cover. Again, please reserve judgment until the rest of the story is told.

> So, the developers need to know what the users really means to build the right product. Purposely making a manager or anyone else a bottle-neck in the communication between users and developers is creating a large single point of failure. The developers need to work directly with the users.

With that I can fully agree. However, sometimes the users may not be available. That does not diminish the truth of the statement, but simply affects its execution. This is why Freedom advocates working with a 'customer rep.' Certainly users can play the role of customer reps. But if users are not accessible, the term covers other, hopefully more feasible, alternatives as well.

Rick Lutowski

From: Steven Gordon

I was not disagreeing with freedom (yet). I mostly disagreeing with the conclusion that the interface is the requirements means that once an analyst can reach agreement with the customer on the user interface then the developers can just develop to that user interface in a vacuum.

From: Rick Lutowski
Steven Gordon wrote:
> I was not disagreeing with freedom (yet). I mostly disagreeing with the conclusion that the interface is the requirements means that once an analyst can reach agreement with the customer on the user interface then the developers can just develop to that user interface in a vacuum.

I see. Then I did miss what you were saying, which is significant. If you don't mind, I would like to explore this a little more.

First, equating requirements to the interface does not, in and of itself, cause one to conclude that customer involvement should not occur downstream. However, Freedom currently does recommend customer involvement only during requirements development, but makes no such recommendation during development of the design and code. So Freedom is in contradiction to what you are saying above, but not for the reason you state.

I am willing to consider that you may be right and Freedom should be changed in this regard. First, tho, I would like to explain why Freedom suggests what it does. I hope you will then take the liberty of contradicting the arguments or, perhaps, at least identifying where assumptions may differ.

Freedom does not insist on customer involvement during design and implementation for several reasons:

1. The behavior tables are capable of capturing customer-mandated gray/white box D&I Constraints in addition to the black box requirements.
2. (While this has not been discussed): The customer's priorities for system-wide quality are captured by specification of the Quality Requirements. Customer priorities for the software quality attributes are recorded right up front, prior to developing Fts, BTs, etc. Quality Requirements directly affect all significant design and coding decisions, and even requirements (external interface) decisions.
3. Incremental and evolutionary development affords frequent 'update points' at which additional customer input can be obtained via updates to the requirements and D&I Constraints without ongoing customer rep involvement during design and coding.
4. Rapport and understanding developed between the team and the customer rep during requirements can be leveraged during design and implementation after the rep has 'gone home.' In most cases, the team will know what he would have said about a question. In cases were they are not sure, they now should have an open line to quickly find out.
5. Availability of customer personnel is often problematic. A customer may be able to spare a rep to work with the development team during specification of requirements, but not for the full duration.

#5 is really the driver. While it would be desirable to have full time customer involvement on a continuous basis, this is rarely achievable (especially nowadays after most companies have trimmed themselves to the bone). Freedom attempts to make recommendations that are practical and realistic rather than ones that are theoretically desirable but probably unattainable in practice. Since it is next to impossible to create viable requirements without some customer involvement, Freedom focuses its cry for customer involvement in the direction where it is most needed rather than diluting the customer involvement message across the entire development spectrum.

On the assumption the customer hears and responds to the request for requirements assistance, #1 - #3 then attempts to 'make hay while the sun shines' by capturing from the customer rep most everything that will be needed from them while they are available. This includes their inputs relative to design and implementation (quality requirements, D&I constraints) as well as requirements. #4 is the fail safe, for occasional use when needed.

If things work out better than expected, and customer management agrees to extend the participation of the rep into design and/or coding, nothing in Freedom says one should refuse this assistance! However, Freedom does not depend on this happening, but takes a more conservative approach.

Am interested in hearing your views on where Freedom goes astray on this issue.

-- Rick Lutowski

From: Philippe Back
Steven Gordon wrote
> I was not disagreeing with freedom (yet). I mostly disagreeing with the conclusion that the interface is >the requirements means that once an analyst can reach agreement with the customer on the user >interface then the developers can just develop to that user interface in a vacuum.

Of course not. Some awareness of the context of the project is in order to be able to bring the risks to an acceptable level.

My context:

- I have to subcontract most of the development work to third parties
- I have to ensure that what third parties will build is described in clear enough terms
- I have to select technology that works for the project
- I have to provide estimates of effort, calendar time & budget that allow me to pay the bills at the end
- I have to deliver the finished product and interim versions

So, letting the developers work in a vacuum is never what I will do.

I want them to produce a working first version that proves itself and then move forward on other phases of the project. Contractually speaking, if they cannot match my requirements, I want to be able to move to second source quickly.

So, I will not let an analyst gather some specs and throw them over the wall to developers.

Of course, my context is not ‘working on super-large projects’ [but this may happen again]. Still, the added value in cash is higher than for larger projects I've seen. I have always tried to move from ‘being a cost’ to ‘being a cash generator’. This changes a lot of things. People listen to you, you build up trust and productivity is boosted.

Let's face it, I moved to my own practice due to the fact that spending hours (or days) in unfruitful meetings and slow moving departments made me feel like I was bleeding valuable hours for nothing. Time is the scarcest resource in life and wasting it is not something I want to do. Hence my interest in methods that work.

A lot of other perspectives do exist of course. As far as I am concerned, I've made my call: work as an indie that can lead multiple resources to develop useful solutions and move as close as possible to the source of money in corporations: management, sales & marketing. Being agile and being able to model helps a lot in there.

Phil

From: Kent J McDonald

I think where you need to be careful here is that you run the risk of people jumping to the wrong conclusion when you say: 'However, Freedom currently does recommend customer involvement only during requirements development, but makes no such recommendation during development of the design and code.'

The potential exists for incorrectly applied literal interpretation of the statement. Some people will read the above statement to say: Talk to the customer during requirements, but don't bother them again until you have something for them to test. I know that is not what you intended, but I have seen it happen time and again where if something is not explicitly stated in a description of a practice or methodology, or even if it is, people will more often than not jump to a literal translation of the message.

I think it is also worth noting that if you tell your customer that you will communicate with them a lot while gathering requirements but not so much during development, you are setting the expectation that they won't hear from you, and they will plan their time accordingly. I have found that the communication during development is perhaps the most rewarding and effective communication because that is when the developer is reaching a better understanding of the requirements and the system because they are actually working on the problem, and the communication often involves questions to clarify understanding, which if allowed to flow free, will make the development process go that much smoother.

Granted, it is not always possible to have a customer on site dedicated throughout the whole project, but if that is not the case, the expectation should be set that 'we will be calling you early and often throughout the process to ask questions, and if we have the ability to do that, you will be much happier with the end result.'

Kent J. McDonald

From: Philippe Back
Subject: Re: [AM] Freedom Tool for trees

http://tkoutline.sourceforge.net/wiki/25

Phil

From: Steven Gordon
Kent J McDonald wrote:
> I think where you need to be careful here is that you run the risk of people jumping to the wrong conclusion when you say: 'However, Freedom currently does recommend customer involvement only during requirements development, but makes no such recommendation during development of the design and code.'

Right!

In English ‘do not recommend’ implies ‘recommend not’. If you want to remain neutral on the issue, you either have to not mention the issue at all, or use lots of words to clearly express that you recommend it at some points in your process and are neutral about it at other points in your process.

Steven

From: Steven Gordon <sagordon@asu.edu>
Philippe Back wrote:
> Of course not. Some awareness of the context of the project is in order to be able to bring the risks to an acceptable level.
So, letting the developers work in a vacuum is never what I will do.

I was pretty sure you did not really mean to imply that developers should work to a spec in a vacuum. I just believe this point is so crucial that I will always challenge statements that suggest moving away from the ideal of totally open information flow.

If the customers cannot handle totally open information flow, then we should certainly be pragmatic. To build communication bottlenecks right into the process does not even give the customer the opportunity to decide how much communication they can handle. Pragmatism applies to how we apply the process to any particular context

Steve

From: PeteCRuth
Kent J McDonald wrote:
> I have found that the communication during development is perhaps the most rewarding and effective communication because that is when the developer is reaching a better understanding of the requirements and the system because they are actually working on the problem, and the communication often involves questions to clarify understanding, which if allowed to flow free, will make the development process go that much smoother.

Right on! And I have found that the best way to engender this relatively intimate level of continual (as opposed to continuous) communication is to be putting working software on the table as quickly and as often as possible for user review and comment.

IMHO, the objective of having a user representative ‘sitting in the cubicle’ with you throughout the process is an often unattainable goal. The demand for input from them is rarely that frequently required, and they have their own jobs to do, as well. If they're not doing those jobs, the cost of their labor goes right into the development project, and it is often difficult to justify the return on that investment to the managers and bean counters. One effective remedy is to ensure that the project sponsor has the authority to mandate the periodic collaboration of the user community whenever it might be required. If that isn't possible, it's a ‘deal breaker’ from the git-go. If they can't make that commitment, they most assuredly will not sit still for ‘full-scale’ involvement, regardless of what they agree to at the

outset of the project. The phrase, ‘We'll do whatever we have to do to make this work’ from the project sponsor, means just that, nothing more, and nothing less. The difference is in the perspective one has: even when the development project has the highest priority, management still has to get the ‘real’ work done, and that usually means some form of compromise. In the real world, business agility means serving many masters on the road to achieving organizational objectives, including the sort of conflicts posed by the need for commitment to software development projects. Having been on both sides of the bridge, I can tell you that compromise is often absolutely necessary when the need for immediate results bumps up against future payback of a development project. While this is more prevalent in smaller organizations, it is true often enough in large projects: nothing happens in a vacuum. Anyone who has not had to contend with this issue should consider themselves truly fortunate.

Regards,

Pete

From: J. B. Rainsberger
Steven Gordon wrote:
> Right!
In English ‘do not recommend’ implies ‘recommend not’.

Well, no, it doesn't; but sadly, yes it does. :)
J. B. Rainsberger

From: Brad Appleton
Rick Lutowski wrote:
> In post 7 we will see that a single stimulus set maps exactly to a single class. Good observation and insight!

Thanks for clarifying. More below ...

> The package mapping is not as straightforward because packages do not map to anything in particular conceptually beyond a collection of classes. Developers create packages based on lots of different criteria. Thus, some teams may decide to create a single package for all the classes that implement the entire FT, as you suggest. Other teams may choose to create a different package for different branches of the FT, for example, one package for human user stimulus sets and another package for external system stimulus sets. Other packaging approaches are certainly possible. This is a combination design and release issue, like any other decision related to packages.

That actually makes me think the ‘package coupling & cohesion’ principles are even more likely to be applicable here. If we consider a ‘requirements package’ to simply be a group of related stimulus sets (whether or not they're in the same FT) then those OOD principles should apply anytime we try to relate stim-sets together.

In fact, such a ‘requirements package’ could perhaps be the more formal ‘equivalent’ (in FREEDOM) of what a use-case was perhaps dreamed to be when it was first ‘thunk of’ :)

> Normally, growing a FT does not result in a need to refactor (that is, change the SS activation architecture) of the tree, except perhaps at a very local level (e.g., insert a SS between two existing SS's).

Certainly one possibility (and one that would ‘resolve’ a potential violation of the ‘Law of Demeter for Stim-Sets’ (LoD-SS :-). Code refactorings are also often very local, but with ‘emergent’ effects over time.

> The more likely cause of FT refactoring is user dissatisfaction with the stimulus set organization. For example, after a period of usage, the users get tired of having to ‘dive deep’ into, say, a menu hierarchy to reach some frequently used functionality.

Sounds like a potential metric (probably very closely related to (7 +/- 2), or at least a guideline for ‘threshold depth’ of FTs.

What does one usually do to resolve such an issue? (wonder if it might bear any resemblance to the common refactoring of replacing conditional/switch with polymorphism)

> they will decide that stimuli in two different stimulus sets should be co-located into one stimulus set (that is, should be physically co-located and active at the same time) because it would make their work more efficient.

Sounds like another potential common pattern/refactoring

Brad Appleton

From: Rick Lutowski
Brad Appleton wrote:
> That actually makes me think the ‘package coupling & cohesion’ principles are even more likely to be applicable here. If we consider a ‘requirements package’ to simply be a group of related stimulus sets (whether or not they're in the same FT) then those OOD principles should apply anytime we try to relate stim-sets together.

Packaging principles would be especially relevant in the case of reusable requirements components (See Post 5).

> In fact, such a ‘requirements package’ could perhaps be the more formal ‘equivalent’ (in FREEDOM) of what a use-case was perhaps dreamed to be when it was first ‘thunk of’ :)

My understanding (correct me if I'm wrong) is that Jacobson's original concept of a use case was an informal prose description of system interaction. Nowadays, UML attempts to model them as functional bubbles that decompose into lower-level functional bubbles (shades of old fashioned DFDs).

The Freedom construct that most closely approximates a use case is a stimulus set, although the two are far from equivalent. Stimulus sets are collections of stimuli in the external interface. Stimulus sets do not 'decompose' like use cases, although they can be arranged into a hierarchy based on activation.

Stimulus sets and use cases are the same only in the sense that both model a 'chunk' of functionality. But the conceptual thinking underlying the modeling is sufficiently different that comparisons can be misleading.

So regarding packaging...

A requirements package would likely consist of more than one stimulus set. However, a requirements package might consist of one use case if the use case were at a sufficiently high level. Because stimulus sets and use cases are conceptually different it is difficult to compare the two and, thus, difficult to compare packaging strategies for the two.

> Certainly one possibility (and one that would 'resolve' a potential violation of the 'Law of Demeter for Stim-Sets' (LoD-SS :-). Code refactorings are also often very local, but with 'emergent' effects over time.

The only thing wrong with the Law of Demeter is calling it a 'Law' (as in 'law of nature'!)

In any event, I would not let things like LoD and OO principles drive the design of the external interface, e.g., stimulus set grouping and organization. LoD etc are intended for code module design, esp OO design. Their applicability to external interface design is problematic.

For human user stimuli, usability principles and customer and user input should be the main drivers. For external system and environment stimuli, comm protocol design principles should be the drivers. Even tho Freedom's mapping from the FT to code modules is quite direct, the more appropriate design criteria are those that apply to interface design, not code module design. The latter can be used as secondary criteria, but should not assume primary weight.

> Sounds like a potential metric (probably very closely related to (7 +/- 2), or at least a guideline for 'threshhold depth' of FTs.

7 +/- 2 can be a useful driver for stimulus sets. However, even this well-accepted rule should be applied with care. See: http://www.guuui.com/posting.php?id=338

> What does one usually do to resolve such an issue? (wonder if it might bear any resemblance to the common refactoring of replacing conditional/switch with polymorphism)

Probably the best way to resolve such issues is to get user feedback, especially through use of a UI mockup, a prototype which can be grown into the final system rather than discarded

(unless the users say it totally misses the target!) For a discussion of depth versus breadth, see: **http://www.ddj.com/documents/s=4058/nam1012431804**

The bottom line in this reference is ease of use, not conformance to specific rules. Unfortunately, ease of use is more subjective than specific rules, hence the importance of user feedback.

>> they will decide that stimuli in two different stimulus sets should be co-located into one stimulus set (that is, should be physically co-located and active at the same time) because it would make their work more efficient.
> Sounds like another potential common pattern/refactoring

I find your observations interesting in a way that transcends the individual comments themselves.

Many of your comments apply code design principles to design of the external interface. As mentioned above, this is problematic as it applies the principles outside their originally intended scope. Indeed, many of them may still apply, but one must be careful.

Other criteria apply more directly to external interfaces, and these should be given primary weight. Some of the more well-known (for a human UI) are by Nielsen:
http://www.useit.com/papers/heuristic/heuristic_list.html

-- Rick Lutowski

From: Brad Appleton
Rick Lutowski wrote:
> Stimuli Recap

Thanks for the recap - very helpful!

> Freedom organizes stimuli in two ways: (1) by grouping stimuli into stimulus sets based on functional, physical, and temporal cohesion, and (2) arranging the stimulus sets into a tree-like structure called a functionality tree.

So if I understood this last post, a functionality tree may contain multiple stimulus-sets. the cohesiveness of stimuli in a set may be functional, physical or temporal. What about the coupling between different stimulus sets in the same functionality tree? Is it the stimulus 'set' that is the 'atomic' unit of encapsulation or is it the functionality tree?

It seems to me there must naturally be lots of opportunities for tight coupling between sets of stimuli in the same functionality tree and that a change that impacted a functionality tree might easily impact much more than just one stimulus-set in the tree.

In some ways the functionality reminds me a little of a UML collaboration diagram or sequence diagram in that it maps out and orders/sequences interactions between the things in

the diagram. Here each thing in the diagram (functionality tree) seems to be a stimulus. If it is implementation independent, then the black-box must come into play when relating two stimuli from different sets (right?). Somehow it is done so that ‘details’ of one ‘stimuli’ aren't known to details of another in the tree.

The functionality tree also is somewhat reminiscent of a ‘call graph’ or ‘call tree’ but here is it more like an activation-tree among stimuli.

These two things combined make me wonder if there isn't some analogous equivalent of ‘the Law of Demeter’ for stimulus-sets, such that a given stimuli/set is ideally ‘ignorant’ of any ‘activation ordering’, or at least to the extent that it shouldn't know anything about ‘stimuli’ that it is not 'directly' and immediately 'connected to' in the functionality tree. Yes?

I'm also beginning to see how some of the ‘Principles of OOD’ as written by Robert Martin would ‘translate’ into the context/domain of requirements encapsulation.
Principles of Class Design:

- (SRP) The Single Responsibility Principle
- (OCP) The Open-Closed Principle
- (LSP) The Liskov Substitution Principle
- (DIP) The Dependency Inversion Principle
- (ISP) The Interface Segregation Principle

Principles of Package Cohesion:

- (REP) The Reuse/Release Equivalency Principle
- (CCP) The Common Closure Principle
- (CRP) The Common Reuse Principle

Principles of Package Coupling

- (ADP) The Acyclic Dependencies Principle
- (SDP) The Stable Dependencies Principle
- (SAP) The Stable Abstraction Principle

‘classes’ might loosely translate to stimulus-sets and ‘packages’ might loosely translate to functionality-trees. I don't think all of them would ‘translate’, but I think most of them might. Not sure if the ‘package’ coupling would translate into coupling between stimulus-sets or functionality trees.

When you talk about ‘growing a functionality tree’ in an incremental fashion, that also makes me think that some kind of ‘refactoring’ should be applicable to these functionality trees (here it would be refactoring of the structure of the tree in some manner based upon known ‘undesirable’ structures within a tree (e.g., perhaps cycles in such a tree are bad, as might be redundancy, or other irregular-looking things that make the tree look more chaotic than simple/orderly)

Brad Appleton

From: Rick Lutowski
Brad Appleton wrote:
> So if I understood this last post, a functionality tree may contain multiple stimulus-sets. the cohesiveness of stimuli in a set may be functional, physical or temporal. What about

Stimuli in a stimulus set are bound by functional, physical AND temporal cohesion (AND, not OR).

> the coupling between different stimulus sets in the same functionality tree? Is it the stimulus 'set' that is the 'atomic' unit of encapsulation or is it the functionality tree?

Requirements encapsulation is per stimulus set, not per FT. Recall the RE design rule from post 1:

'Create a functionality module for each unique stimulus set of the functionality tree.'

Stimulus sets tend to be rather loosely coupled as far as their code is concerned. However, they are much more tightly coupled as far as their working together to achieve the customer's goals (but I guess that's true of everything in a program, isn't it!?)

> It seems to me there must naturally be lots of opportunities for tight coupling between sets of stimuli in the same functionality tree and that a change that impacted a functionality tree might easily impact much more than just one stimulus-set in the tree.

Customers can request changes to the external interface that span a single stimulus to the entire FT. This is not so much a matter of the FT being susceptible to change or coupling as it is the customer's desires being susceptible to change across different breadths of impact. This is true not just of Freedom but all approaches to requirements. Customers can ask for all manner of changes (and often do.)

> In some ways the functionality reminds me a little of a UML collaboration diagram or sequence diagram in that it maps out and orders/sequences interactions between the things in the diagram. Here each thing in the ...

Closer to collaboration than a sequence diagram. The FT is not time domain any more than a drawing of, say, a GUI is. The FT *IS* a drawing of the external interface, it's just a schematic drawing (much like an electrical schematic is a drawing of an electrical circuit.)

> ... diagram (functionality tree) seems to be a stimulus. If it is implementation independent, then the black-box must come into play when relating two stimuli from different sets (right?). Somehow it is done so that 'details' of one 'stimuli' aren't known to details of another in the tree.

It's not so much that the black box 'comes into play' when relating stimuli as that the stimuli and their relationships ARE the black box specification (part of it, anyway).

Correct in that details of a stimulus are not know to other stimuli. Stimuli almost never directly refer to one another, although they often refer to one another's behavioral results (such as a value that was saved or computed as a response to some stimulus).

> The functionality tree also is somewhat reminiscent of a 'call graph' or 'call tree' but here is it more like an activation-tree among stimuli.

It is precisely an activation tree. At the code level, the activation is usually accomplished by method calls. So 'yes' to both of the above.

> These two things combined make me wonder if there isn't some analogous equivalent of 'the Law of Demeter' for stimulus-sets, such that a given stimuli/set is ideally 'ignorant' of any 'activation ordering', or at least to the extent that it shouldn't know anything about 'stimuli' that it is not 'directly' and immediately 'connected to' in the functionality tree. Yes?

Freedom basically extends OO to encompass requirements encapsulation in addition to data structure and hardware interface encapsulation. So current OO theory should apply to equally well to requirements encapsulation as as encapsulation of data and hw.

> 'classes' might loosely translate to stimulus-sets and

In post 7 we will see that a single stimulus set maps exactly to a single class. Good observation and insight!

> 'packages' might loosely translate to functionality-trees. I don't think all of them would 'translate', but I think most of them might. Not sure if the 'package' coupling would translate into coupling between stimulus-sets or functionality trees.

The package mapping is not as straightforward because packages do not map to anything in particular conceptually beyond a collection of classes. Developers create packages based on lots of different criteria. Thus, some teams may decide to create a single package for all the classes that implement the entire FT, as you suggest. Other teams may choose to create a different package for different branches of the FT, for example, one package for human user stimulus sets and another package for external system stimulus sets. Other packaging approaches are certainly possible. This is a combination design and release issue, like any other decision related to packages.

> When you talk about 'growing a functionality tree' in an incremental fashion, that also makes me think that some kind of 'refactoring' should be applicable to these functionality trees (here it would be refactoring of the structure of the tree in some manner based upon known 'undesirable' structures within a tree (e.g., perhaps cycles in such a tree are bad, as might be redundancy, or other irregular-looking things that make the tree look more chaotic than simple/orderly)

Normally, growing a FT does not result in a need to refactor (that is, change the SS activation architecture) of the tree, except perhaps at a very local level (e.g., insert a SS between two existing SS's).

The more likely cause of FT refactoring is user dissatisfaction with the stimulus set organization. For example, after a period of usage, the users get tired of having to 'dive deep' into, say, a menu hierarchy to reach some frequently used functionality. So they may say -- 'Hey Mr. Developer, can't you make this menu/window/field/whatever more easily accessible?' Or maybe they will decide that stimuli in two different stimulus sets should be co-located into one stimulus set (that is, should be physically co-located and active at the same time) because it would make their work more efficient. Of course, one tries to address such usability concerns up front by working with users, etc. But sometimes it just takes a period of actual usage to determine the 'right' FT organization. Usability issues are probably the most likely cause of major refactoring, not simply adding a new stimulus set.

Rick Lutowski

From: Richard Fisher

I fully agree with keeping the functionality tree implementation independent. But why not distinguish between data and command stimuli? That much is surely implementation independent but very illuminating.

Rick Fisher

From: Rick Lutowski
Richard Fisher wrote:
> I fully agree with keeping the functionality tree implementation independent. But why not distinguish between data and command stimuli? That much is surely implementation independent but very illuminating.

I agree distinguishing command from data stimuli in the FT can be helpful. Often, the names of the stimuli connote the distinction, e.g., OK is clearly a command, while Filename is a likely data stimulus. In cases where the name leaves doubt, I have always relied on the behavior table to clarify the issue. However, there is certainly value in making the FT definitive as well.

One way to do this is to annotate the stimulus name in the FT, e.g., Address (d) for a data stimulus or Address (c) for a command stimulus. Such an annotation convention is preferable to altering the stimulus name itself, e.g., Address Data or Address Cmd, because the stimulus name should be as nearly identical as possible to the way the name should appear on the actual interface (assuming it appears in a visible manner.) Thus, Address Data in the FT would imply labeling, say, a text field 'Address Data' and not just 'Address'. Address (d) would imply labeling the field 'Address'. Depending on the tool employed, other annotation conventions are also possible -- choice of fonts or colors in the FT perhaps.

Freedom's notations are not standardized, so one is free to make adaptations and improvements as necessary. I see this remaining the case until a Freedom-specific toolset

becomes available, at which point the tools would impose notational conventions on users of the tools. Until then, and even after, Freedom's notations will continue to evolve.

If anyone has other ideas on how to better distinguish command from data stimuli in the FT, please let me know.

Thanks.

Rick Lutowski

From: Scott W. Ambler
Someone wrote:
> Right!
In English 'do not recommend' implies 'recommend not'. If you want to remain neutral on the issue, you either have to not mention the issue at all, or use lots of words to clearly express that you recommend it at some points in your process and are neutral about it at other points in your process.

The sentence also reads as if you're promoting a serial approach to development, which clearly you don't.

You should discuss the importance of customer involvement and strategies for ensuring that you get it. I run into excuses all the time why the stakeholders can't get involved, but rarely do they hold up to scrutiny. Customer involvement is worth fighting for.

- Scott

From: Rick Lutowski
Scott W. Ambler' wrote:
> The sentence also reads as if you're promoting a serial approach to development, which clearly you don't.

The implication of the above statement is that requirements can be revisited and changed at any time. This is the best rationale I have heard yet for continuous customer involvement throughout the development process. However the frequency of requirements change will not be constant, nor in most cases will the time a customer rep can make available. Fortunately, the two frequency curves are usually in synch.

> You should discuss the importance of customer involvement and strategies for ensuring that you get it. I run into excuses all the time why the stakeholders can't get involved, but rarely do they hold up to scrutiny. Customer involvement is worth fighting for.

Agree.

Currently Freedom specifies customer involvement only in the context of requirements. I am convinced this needs revision relative to other tasks. Based on the discussion, I think the revision should be of the form:

a. Customer should be made aware up front of necessity of direct involvement in requirements, benefits of involvement in other tasks, and consequences of non- involvement. -- ADDED MESSAGE

b. Customer rep must directly participate with initial requirements specification -- CURRENT MESSAGE and subsequent changes. -- ADDED MESSAGE

c. Customer rep should, at a minimum, be available to respond to D&I questions. -- ADDED MESSAGE

I think this covers the main points that were raised. Please let me know if it misses anything major.

Rick Lutowski