Chapter 3
Author: Rick Lukowski and Others
Compiler and Editor: Thomas W. de Boer

Freedom Post 3: Stimuli and Stimulus Sets

Prerequisite Example Application

Starting with this post, we shall be using Adobe Acrobat reader version 5.08 as an example application to illustrate certain points. This is the latest version of Acrobat reader as of this writing. If you have an older (or no) version of Acrobat reader installed on your system, you may download version 5.08 for a wide variety of platforms, free of charge, from http://www.adobe.com/products/acrobat/readstep2.html

Concept and Definition Recap

So far, we have seen that Freedom defines requirements as the ‘black box view of the software system,’ a definition that leverages the box-based view of software proposed by Harlan Mills. The practical implication of this definition is that a requirements specification is identical to a specification of the software system external interface because the black box view of a system is identical to the system external interface. Thus, Freedom specifies, and develops, the external interface first, rather than later or even last as in some other methodologies.

'Interface-first’ follows directly from the the black box definition of requirements and the box-based concept model upon which it is founded. Some have said that the essence of agile is its values and principles. Similarly, the essence of Freedom is its models and definitions. All other artifacts of Freedom either derive from the concepts and definitions, or at least are compatible with them.

External Interface Specification

With Freedom's fundamentals firmly in mind, we know that the first thing to do when developing software with Freedom is to specify the external interface. (Actually, this is strictly not true, but is a close enough approximation for our purposes, which is to understand Freedom's approach to requirements encapsulation.)

How does one specify an external interface in Freedom? For the answer, we again turn to Harlan Mills.

Mills suggested a two-step process: (1) specify the system stimuli, and (2) specify the responses to the stimuli. To this process, Freedom adds one more step: (3) specify the protocol. In keeping with Parnas' idea of a rational process, these steps may be performed in any order, except causality requires that stimuli be identified before their responses. Thus, we shall start with identification of stimuli.

Stimuli

What is a stimulus? Isn't it the same as an input?

Almost.

There are two kinds of stimuli: Command Stimuli and Data Stimuli.

Data stimuli are values with semantic meaning to the problem being solved by the program. Examples include (but are not limited to) data typed into data entry fields or onto command lines by users; data read from files or data bases; and data input from a network connection. The data may be saved, transmitted, used in a computation, or used in any way or purpose necessary and consistent with its semantic value to the problem being solved.

Command stimuli are trigger values that have no semantic meaning to the problem being solved by the program. The simplest and most common example is the value created by a button when pressed by a user. The value has no intrinsic semantics to the problem being solved; it merely acts as a signal to initiate some response.

Another example is an XML tag value, such as 'name' in the XML fragment

<name>Rick</name>

'name' has no intrinsic value to the problem being solved. We could just as easily change 'name' to 'xxx' in the DTD and it would make no effective difference. Thus, 'name' is a command stimulus. However, the value 'Rick' is a data stimulus; it cannot arbitrary be changed to, say, 'Pete' without some profound change in the results of the program, e.g., Pete (erroneously) goes to jail instead of Rick if, say, the XML file lists convicted felons.

This example also demonstrates a common relationship among the types of stimuli: command stimuli often act as delimiters for data stimuli. This is particularly common in streaming input from sources such as formatted files (of which XML is but a special case).

Which takes us back to input. When most people use the term 'input', they are usually thinking in terms of data, i.e., 'input' and 'input data' are essentially synonymous in most people's minds. Conversely, most people do not usually associate the term 'input' with a button press (except, perhaps, developers who write lots of GUI code.) Hence, 'input' differs from 'stimuli' by 'command stimuli':

input == data stimulus == command | data

Thus, Freedom uses the term 'stimuli' in lieu of 'input' to ensure that command stimuli are not forgotten or otherwise given short-shrift.

Stimulus Sources

Stimuli can originate from any or all of three sources:

1. Human users
2. External system users
3. The environment

The most common mechanisms for entering human user stimuli are command lines and GUI displays. However, other sources such as voice and tactile (e.g., touch-sensitive screens) may be used in some applications. Experiments with monkeys have successfully demonstrated computer control directly via brain waves, so who knows what programming challenges human user interfaces of the future may pose! For now, let's just be content with the ‘simplicity’ of GUIs and command lines.

External system stimuli are stimuli that originate in some system black box external to our black box. The external black box may or may not be another software system; we don't know or care (that's what we mean by saying it's 'black.'). What we must know and do care about is the exact format with which to communicate with that other black box. The stream from the other system is expressed as a sequence of command and data stimuli in some particular format, or protocol. It may arrive via transmission carrier mechanisms such as wire (network, TV cable) or wireless (satellite, RF, microwave). These are all examples of stimuli from external system users.

Stimuli from the environment are used to measure natural data such as wind speed, temperature, pressure, stress, presence of physical objects, or the simple passage of time, when such stimuli are of significance to the problem being solved. Usually, the mechanisms for producing environment stimuli are sensors of various kinds, with appropriate A/D converters when necessary.

Stimulus Organization and Cohesion

A non-trivial program will typically have many stimuli. Whenever there are ‘a lot’ of anything, human cognitive limits require we employ some organizational strategy to avoid mental overload.

Freedom uses multiple organizational strategies to help people more effectively deal with large numbers of stimuli. The first such strategy is to organize the stimuli into groups or sets based on cohesion. The second strategy is to organize the sets of stimuli into a tree-like structure called a functionality tree. Functionality trees will be covered in the next post. Here, we will consider grouping of stimuli into sets based on their cohesion.

While there are at least seven commonly recognized types of cohesion, it is generally sufficient to consider only three types when grouping stimuli into sets:

1. Functional -- all the stimuli in the set relate to the same task, operation, or object
2. Physical -- all the stimuli in the set are physically co-located
3. Temporal -- all the stimuli in the set are or can be active at the same time

Generally, the strength of cohesion of some information is determined by the kind of cohesion exhibited, with functional cohesion being among the strongest and temporal among the

weakest. In most cases, developers are encouraged to achieve any type of cohesion they can, with the stronger types being preferred.

In the case of stimuli, all the stimuli in a given set will usually exhibit ALL of the above three types of cohesion, not just one. This is much stronger cohesion than is typically found in many situations when only one kind of cohesion is in play. Yet, achieving triple-high levels of cohesion is easy in external interfaces because such cohesion occurs naturally in that context.

Stimulus Sets

A single cohesive set of stimuli is called a ‘stimulus set.’ Stimulus sets occur in a natural and obvious way in human user interfaces. They also occur in communication streams to external system black boxes and the environment, although this is not nearly as obvious due to the 'invisible' nature of programmatic streams. Thus, it is much easier to learn about stimulus sets using a human user interface as an example.

Identifying Stimulus Sets

At this time, if you have not done so already, please download and install Adobe Acrobat Reader (AAR) 5.08 from the URL specified at the top of this post. Then run the program, using it to display any convenient pdf file. We will not be looking at the file displayed, but rather at the AAR program itself, so the pdf file you choose to display is arbitrary.

* * *

OK, I assume AAR 5.08 is now up and running. Let's examine AAR and identify some of its stimulus sets.

Recall a stimulus set is a collection of stimuli that is functionally, physically, and temporally cohesive. Stimuli are command and data inputs, but only inputs; never outputs. So we are looking for command and data inputs that relate to the same function, are active at the same time, and are physically co-located. The last attribute is especially amenable to visual searching -- look for stimuli (command and/or data inputs) that physically group together. Several are apparent.

The First Stimulus Set

The first is at the very top of the AAR window -- the row of six menu items labeled File, Edit, Document, View, Window, and Help. These are physically cohesive by virtue of being grouped together in the same top row and being separated from the remainder of the program by a horizontal line. It can also be argued they appear on the same panel, and the horizontal line is merely the bottom of the panel. Either way, these six stimuli are physically grouped.

Quick quiz 1: What kind of stimuli are these?

The six stimuli are also temporally cohesive because they all are, or can be, active at the same time. Mouse-pressing each one is sufficient to demonstrate they are all active at the same time.

Finally, the six stimuli are (or should be) functionally cohesive.

Quick quiz 2: What is the operation, task, or object that binds the six stimuli together functionally?

More Stimulus Sets

Having identified the first stimulus set, the term starts making a little more sense, and it now becomes more obvious what some of the other stimulus sets are. The second row of 17 icons looks like another stimulus set. So does the third row of eight icons and one text field. At the very bottom is a row of eight icons and one text field. All these are stimulus sets by virtue of being stimuli that are grouped physically, temporally, and functionally (although the task or object that gives them functional cohesion may require some thought.)

Right Display Region Stimulus Sets

How about that region to the left with the thumbnails? How about the region to the right with the content of the pdf file? Are these stimulus sets?

To answer these questions, ask two other questions:

1. Are any command or data inputs present? If so,

2. How do they group based on physical, functional, and temporal cohesion?

Analyzing the pdf display region to the right in this manner:

1. Are any command or data inputs present?

Yes, there is a vertical scrollbar as part of the region. Under some circumstance, a horizontal scrollbar can also appear.

2. How do they group based on physical, functional, and temporal cohesion?

The scrollbars are physically in the same region panel, have the same function, and are (or can be) active at the same time. Thus, the right region consists of only one stimulus set of two scrollbars, either of of which may or may not be active depending on the relative dimensions of the pdf content being displayed.

Left Region Stimulus Sets

Similarly analyzing the thumbnail region to the left:

1. Are any command or data inputs present?

Yes, a scrollbar and two tabs labeled Thumbnail and Bookmarks. Also, in the upper right corner of the region is a choice menu labeled Thumbnail.

2. How do they group based on physical, functional, and temporal cohesion?

Experiment to explore the nature of the cohesion attributes. Clicking a tab displays a separate scrollbar for each tab, and also a different choice menu. The Bookmarks tab has a choice menu labeled Bookmark. So there are more stimuli than were first apparent, with temporal cohesion coming into play to mask some of them based on the tab selected. This temporal cohesion, along with associated physical cohesion, imply two stimulus sets (SS) in the left region:

a. A Thumbnail SS consisting of a scrollbar and a choice menu. b. A Bookmarks SS consisting of a scrollbar and a choice menu.

When the Thumbnail tab is clicked, a list of page icons appears in addition to the scrollbar and a choice menu. Are these part of the stimulus set?

Answer, apply the two questions:

1. Are they stimuli (command or data inputs)?

Yes, we can click on them and a response happens (the displayed pdf page changes). They _are_ stimuli.

2. How do they group based on physical, functional, and temporal cohesion?

All the page icons are physically together, active at the same time, and do the same thing (jump to a new page). They are in the same stimulus set.

Are they in the same SS as the Thumbnail scrollbar and choice menu, or in their own SS?

Again, experiment and evaluate against the cohesion criteria. What does the choice menu do? It causes the page icons to change size, which in turn has an influence on the scrollbar size. So the choice menu and scrollbar are functionally related to not only each other but the page icons. Also, the choice menu, scrollbar, and page icons are all temporally cohesive (active when the Thumbnail tab is selected, disabled when the Bookmarks tab is selected) and physically cohesive (all on the same Thumbnail panel).

Conclusion: The page icons are all in the Thumbnail stimulus set along with the scrollbar and a choice menu. They are not in their own SS since they do not differ wrt functional. physical, or temporal cohesion. Hence, a more complete stimulus set description of the left region would be:

a. A Thumbnail SS consisting of a scrollbar, choice menu, and page icons,

b. A Bookmarks SS consisting of a scrollbar, choice menu, and bookmark icons (by analogy to the Thumbnail case; the file I am displaying does not have bookmarks in it to experiment with.)

Oops! One more detail before leaving the left region. Didn't we forgot to list the Thumbnail tab in the Thumbnail SS, and the Bookmarks tab in the Bookmarks SS?

Actually, no. The two tabs activate the stimuli in their respective stimulus sets, but are not themselves in those stimulus sets. Logically, clicking them when their respective SS is already displayed should have no effect or, better, the tabs should be disabled when its SS is already displayed. (Note: this is not the actual case however. Clicking a tab when its SS is already displayed makes the entire left region disappear! Adobe may claim this a feature, but SS analysis plus the principle of Least User Surprise all say it is a bug.) Thus, the tabs are not (or should not be) temporally cohesive with the stimuli in their regions. Also, they are not physically cohesive as they lay outside the panels they activate. Finally, they have a different function. While the function of the panels is to control page navigation, the function of the tabs is to control selection of the page navigation panels. As a result, the tabs are not part of the two left region stimulus sets, but are in fact their own SS:

c. A page navigation panel selection SS consisting of two tabs labeled Bookmarks and Thumbnail.

This does not complete the stimulus set analysis of the AAR program by any means. It is just the beginning. However, it should be enough to get a basic feel for what stimulus sets are in a practical sense rather than just a theoretical sense. Readers are encouraged to find other stimulus sets on their own using the definitions and techniques described above, and to ask questions about any new wrinkles they encounter.

Quick Quiz Answers

Post your answers to the two quick quizzes and let's see if we have a consensus on the answers.

Rick Lutowski

From: Jason Gorman
Rick Lutowski wrote:
>Stimuli are command and data inputs, but only inputs; never outputs.

I'm afraid I completely disagree. Data is not a stimulus. If you're specifying stimulus-response pairs you can't specify that the system responds to ‘Mr J Gorman’, for example. Some event has to trigger it to do something with that data. The stimulus is the kick, not the cat. A stimulus may *have* associated input data (parameters), but they are not stimuli in themselves.

In event-driven language processing/transformations, the stimulus is not the data, the stimulus is the event of hitting the data :-)

Jason

From: Philippe Back
What do you think of JAXP then ?

Philippe

From: Rick Lutowski
Jason Gorman wrote:
>The stimulus is the kick, not the cat. A stimulus may *have* associated input data (parameters), but they are not stimuli in themselves.

One definition of stimulus is ‘anything that elicits a response.’

The receipt of a datum can certainly elicit a response. The response may be complex, or it may be as simple as echoing the data entered (visible simple response) or just tucking the data away for future use (non-visible simple response). The null response (to do nothing and ignore the data) is also a possible response.

No doubt there are methods and techniques in which data are not considered stimuli. However, in Freedom they are. I cannot speak for other approaches you may have used, but Freedom's definition of stimulus works well in the context of Freedom.

Rick

From: Jason Gorman

Then is it the receipt or the datum that is the stimulus? Could you give me an example where a system reacts/responds without some event taking place (eg, a button click, a key press, time elapsing, a condition arising etc)? My last email only solicited a response because I sent it (for better or for worse) and you read it, for example. The mere existence of my email was not enough to illicit a response :-)

Jason

From: Rick Lutowski
Jason Gorman wrote:
> Then is it the receipt or the datum that is the stimulus? Could you give me an example where a system reacts/responds without some event taking place (eg, a button click, a key press, time elapsing, a condition arising etc)?

A pure datum in isolation is not a stimulus. It becomes a stimulus when it appears, or is received, at the interface.

The process of 'being received' is a discrete process. At some time, the first bit of the datum appears. It is not yet a stimulus. Later, the last bit of the datum appears, usually along with some indication it is the last bit (handshake signal, termination code, etc). The complete datum has now been received. It is now a stimulus. The process of receipt is not the stimulus; the presence of the complete datum at the interface is the stimulus.

(Note: One can draw a smaller black box around the part of the system that receives stimuli for the 'big' black box. At this level, each bit or byte may be a stimulus, along with associated handshake signals, whatever -- depends on the protocol details. The same S-R thinking can now be applied, but at a lower or finer-grained level. Conveniently, at the level of the 'big' system black box, this level of detail is abstracted away and can be ignored.)

> My last email only solicited a response because I sent it (for better or for worse) and you read it, for example. The mere existence of my email was not enough to illicit a response :-)

Funny; I considered using that very example in my previous reply. I was going to phrase it something like this:

Your message is a stimulus. I am responding to it. However, I am not responding simply because it was received, but because of the data in it. Moreover, the exact nature of my response also depends on the specific data in it. Hence, the data is the stimulus, not the mere fact of receipt.

Rick Lutowski

From: "Ashley McNeile"
Rick Lutowski wrote:
> (although the task or object that gives them functional cohesion may require some thought.)

The implication of this statement is that, in order to define functional cohection, you have to think about tasks and/or objects. IOW, if you do not know what the tasks and/or objects are, you are working in the dark trying to identify ‘functional cohesion’. At what stage in Freedom do you identify ‘tasks’ and ‘objects’?

Ashley

From: Rick Lutowski
Ashley McNeile wrote:
>> (although the task or object that gives them functional cohesion may require some thought.)
>The implication of this statement is that, in order to define functional cohection, you have to think about tasks and/or objects. IOW, if you do not know what the tasks and/or objects are,

you are working in the dark trying to identify ‘functional cohesion’. At what stage in Freedom do you identify ‘tasks’ and ‘objects’?

The specification of the external interface is the very first major task of developing a new application in Freedom. Because the interface is the part of the app that 'touches' the users (both human and programmatic) it must be specified with input, and ideally direct daily assistance, from individuals directly knowledgeable about the customer's needs. Usually these will be individuals on the customer's staff.

Determining 'objects' and/or 'tasks' is just one artifact of specifying the external interface. The real job is to determine the interface needed to accomplish the user's job -- the stimuli, their organization, responses, and protocols. Whether the grouping and organization of stimuli will be along the lines of task steps, objects, operations, or whatever, are among the things that are determined with the help of knowledgeable customer reps.

In my experience, it is common for a customer to have a good idea of what they want the external interface to be like at the very outset. In fact, I cannot recall any job I have been on where this was not true. In most cases, the final interface did not exactly match the customer's initial conception, but was not too far from it either.

When reverse engineering an interface, as we did with AAR, one usually must employ intense scrutiny to discover the artifacts of functional cohesion underlying the stimulus sets. When 'forward engineering' an interface for a project, the functional cohesion artifacts are often not thought about on a conscious level. The customer reps and developers just start laying out the interface the way the customer had in mind, and the cohesive elements just sort of fall out in the wash. Granted, if one learns to think about the functional, physical, and temporal cohesion more explicitly, the interface may improve as a result. However, identifying the function artifacts -- be they tasks, operations, and/or objects -- is not usually a problem. Rather, the problem is perceived in the larger sense as the 'design of the external interface.'

Rick Lutowski

From: Ashley McNeile
Rick wrote
> In my experience, it is common for a customer to have a good idea of what they want the external interface to be like at the very outset.

If the users' idea of what the user interface should look like is used to determine the stimulus-sets, does the quality of the model derived using Freedom depend on the quality of the work done by the users to determine what the user interface should look like? IOW, if the user has made a lousy job of designing the UI, does this compromise the result of using Freedom? Suppose the user simply grouped the menu items alphabetically in their design?

I often work with senior users on applications architecture and strategy. I build models at this level, some of them executable, but seldom have any kind of UI design to work from. Could I use Freedom in these circumstances?

‘No’ would be a perfectly acceptable answer to the question above. All approaches I have seen have a domain of applicability, and I am sure Freedom has too.

Ashley

From: Rick Lutowski
Ashley McNeile wrote:
> If the users' idea of what the user interface should look like is used to determine the stimulus-sets, does the quality of the model derived using Freedom depend on the quality of the work done by the users to determine what the user interface should look like? IOW, if the user has made a lousy job of designing the UI, does this compromise the result of using Freedom? Suppose the user simply grouped the menu items alphabetically in their design?

To quote what I (= Rick) wrote:

‘it [the external interface] must be specified with input, and ideally direct daily assistance, from individuals directly knowledgeable about the customer's needs.’

‘input and ... assistance from’ means the developers are involved as well. In fact, the developers have the responsibility for the system, including the external interface; that is what they are being paid for Customer-knowledgeable individuals assist, but do not unilaterally ‘determine’ the interface, as your statement implies. The development team should be savvy enough about software interface design to preclude a ‘lousy’ interface from being specified.

> I often work with senior users on applications architecture and strategy. I build models at this level, some of them executable, but seldom have any kind of UI design to work from. Could I use Freedom in these circumstances?

Before answering I first must ask you a question: At what point do you specify the external interface?

Rick Lutowski

From: "Ashley McNeile"
Rick wrote
> ‘input and ... assistance from’ means the developers are involved as well.

My question was not really about who does it, but about how it is done.

> The development team should be savvy enough about software interface design to preclude a ‘lousy’ interface from being specified.

What, in the view of Freedom or yourself, is a lousy interface? How do you distinguish between a good interface and a lousy one? If you answer that a good interface is one that

places items that have functional cohesion under the same menu, then your argument is circular.

> At what point do you specify the external interface?

In the context of providing access to underlying business capabilities via a particular channel. My interest and work is in modelling the required business capabilities and services, independently of how they are accessed.

I have to say that I agree with Steven G on this when he said: 'Everything I have learned about software development pushes me to make the user interface the thinnest possible layer around the system, ...'.

In my view, good modeling comes from examination of the underlying business domain. By contrast, the user interface is ephemeral.

I suspect that my perspective is just different from that of Freedom. It may be that both have validity, so I suggest you just press ahead with your exposition rather than getting bogged down dealing with people like me.

Ashley

From: Steven Gordon

To carry Ashley's argument one step further: if the stimuli and how they are aggregated into sets is completely determined by the structure of the applications's user interface, then don't we need to redesign the solution whenever the user interface changes (even if the customer did provide a good initial user interface structure)?

Everything I have learned about software development pushes me to make the user interface the thinnest possible layer around the system, and to think of the system as a self-contained entity like an API, web service, or facade pattern, so as to insulated the business logic from the whims and fads of the user interface world.

Steven Gordon

From: Rick Lutowski
Steven Gordon wrote:
> To carry Ashley's argument one step further: if the stimuli and how they are aggregated into sets is completely determined by the structure of the applications's user interface, then don't we need to redesign the solution whenever the user interface changes (even if the customer did provide a good initial user interface structure)?

Requirements in Freedom are not equivalent to the _user_ interface. They are equivalent to the _external_ interface. Big difference.

Bearing that in mind, allow me to rephrase your question into more-traditional nomenclature:.

‘...don't we need to redesign the solution whenever the requirements change...?’

I'll bet you know the answer to that question. The answer with Freedom is the same.

> Everything I have learned about software development pushes me to make the user interface the thinnest possible layer around the system, and to think of the system as a self-contained entity like an API, web service, or facade pattern, so as to insulated the business logic from the whims and fads of the user interface world.

Requirements in Freedom are not equivalent to the _user_ interface. They are equivalent to the _external_ interface. Big difference.

It has been my experience after using interface-centric development for perhaps a decade that the interface is, in fact, 'thin' when measured in terms of development effort involved. I find that on the order of days-to-weeks is spent specifying and building the external-interface, while weeks-to-months (or even years) is involved in specifying and building the gray and white box aspects of the system. Thus, requirements are specified and implemented very quickly in Freedom precisely because the interface is 'thin' effort-wise.

Rick Lutowski

From: Steven Gordon
Your Adobe Acrobat Reader example seemed to rely on the organization of the existing user interface to identify the stimuli and decide which should be group together. Is this is just a short-cut for this example (i.e., Freedom actually groups the stimuli in a GUI-independent manner)?

If so, it would be more instructive to:
- start from a representative collection of the stimuli you would start with for this program,
- group the stimuli as guided by whatever GUI-independent technique or process that Freedom utilizes, and
- compare those groupings to how they were grouped in actual software as evidenced by the existing user interface.

My answer to whether my approach needs to redesign whenever the requirements change, I would answer that it depends on the changes, but if only the the user interface changes (i.e., functionality stays the same) then I would generally not need to redesign the business logic or data persistence. Was that your answer for Freedom?

Steven

From: Rick Lutowski
Steven Gordon wrote:
> Your Adobe Acrobat Reader example seemed to rely on the organization of the existing user interface to identify the stimuli and decide which should be group together. Is this is just a short-cut for this example (i.e., Freedom actually groups the stimuli in a GUI-independent manner)?

It would be more accurate to say the AAR example 'identifies the stimuli and stimulus sets that are already present in an existing user interface'. Freedom does not 'rely on' an existing interface implementation to identify and organize stimuli. Nor is the AAR example taking any shortcuts. It is a pure case of reverse engineering, i.e., derive a requirements model from an existing implementation. I chose this approach to explaining stimuli in response to requests from the group for concrete examples. The goal of Post 3 was simply to explain what stimulus sets were.

You are correct in assuming that Freedom records stimuli and stimulus sets in a GUI-independent manner. This hopefully will become clear when we get to the next post covering functionality trees, which is an implementation-independent notation for specifying the architecture of the external interface.

> If so, it would be more instructive to:
- start from a representative collection of the stimuli you would start with for this program,
- group the stimuli as guided by whatever GUI-independent technique or process that Freedom utilizes, and
- compare those groupings to how they were grouped in actual software as evidenced by the existing user interface.

That might be an interesting approach, but it presumes that there is some cut-and-dry algorithm for grouping stimuli into stimulus sets. There is not. Rather it is a process called 'design.' While things like human factors can provide useful guidance, nothing hands us the stimulus groupings on a platter. Different developers are likely to design different interfaces given the same input from customer domain personnel, just as they would likely produce different prose requirements hierarchies, use case breakdowns, or whatever. That is of little consequence provided that customer needs and expectations for the software are met.

> My answer to whether my approach needs to redesign whenever the requirements change, I would answer that it depends on the changes, but if only the the user interface changes (i.e., functionality stays the same) then I would generally not need to redesign the business logic or data persistence. Was that your answer for Freedom?

Exactly -- it depends on the change. As you say, if only some aspect of the external interface changed (say the color of a button) then clearly the internal design would not need to change.

It is also possible that some externally visible functionality changes would have little or no impact on the internal design, but a significant impact on the external interface. An example

might be displaying values that are already available internally but were not displayed previously, but the users now want to see.

Of course, it is easy to postulate changes to the external interface that also have dramatic effects on the internal design.

Rick Lutowski

From: Rick Lutowski
Ashley McNeile wrote:
> My question was not really about who does it, but about how it is done.

Covered this in a post just submitted in response to Steven G. To summarize, Freedom provides definitions and notations, but there is no cut-and-dry process for doing design, be it design of the external interface or design of the modules. If there were a cut-and-dry process, design could be automated and we would not have so much to do!

What, in the view of Freedom or yourself, is a lousy interface? How do you distinguish between a good interface and a lousy one? If you answer that a good interface is one that places items that have functional cohesion under the same menu, then your argument is circular.

Can you define a lousy module architecture? A lousy external interface design would likely have analogous characteristics.

Actually, I do have some ideas on this, but would like to hear yours first since you raised the issue of a 'lousy interface.'

> In the context of providing access to underlying business capabilities via a particular channel. My interest and work is in modelling the required business capabilities and services, independently of how they are accessed. [...] In my view, good modelling comes from examination of the underlying business domain. By contrast, the user interface is ephemeral. I suspect that my perspective is just different from that of Freedom.

Well, I and others believe you are in the majority. Interface- first development is not common. Freedom's perspective *is* different from the majority view. I think that was pretty obvious to all from the start, but the group indicated a desire to learn more about it anyway (or, perhaps, just because it is so different). Given that understanding, it seems rather incongruous to challenge it simply on the grounds that it is different from what you currently do. That is pretty much a given.

> It may be that both have validity, so I suggest you just press ahead with your exposition rather than getting bogged down dealing with people like me.

Having used both interface-first and internal-model-first approaches, I have no doubt both do have validity, but not necessarily in the 'either-or' way that might be expected.

Nuclear fusion may be a good analogy for explaining what I mean.

The solution to the nuclear fusion problem is ‘internal-model-first’ all the way. Specifying the external interface to a fusion reactor is not a viable first step to solving the fusion problem. The problem of a workable internal fusion process must be solved first, even tho it is ‘inside the reactor black box.’ This is the Big Risk that must be addressed first.

However, this does not invalidate Freedom. It simply changes the order that the process steps should be performed. Performing the steps in whatever order is necessary to meet the needs of the project *IS* what Freedom says to do, per Parnas' idea of a ‘rational process’. Hence, in the case of fusion, Freedom would concur with mitigating the internal fusion process Big Risk first.

Once that problem is solved and we know how to initiate and run a fusion process for net energy gain, then we can actually contemplate the development of a commercial fusion reactor. At that point, the ‘rational’ Freedom process kicks back in, starting with specification of the external interfaces of the reactor -- to its human controllers, to the external power grid, etc. Domain experts on the new fusion process would assist in specification of these interfaces, which constitute the requirements for the plant, by definition. They would tell the instrumentation human factors specialists what values need to be monitored, what the controls should be like, etc. They would similarly advise the electricians specifying the interface to the power grid, and the chemical and other engineers defining the interface to the environment, such as thermal abatement and other pollution control factors. All these things would be requirements due to being the black box view of the fusion plant. Once the black box requirements are specified -- amounts of deuterium (say) in, amount of power out, amount of pollution out, control inputs in, warning indicator values out, etc then the requirements are specified.

Meanwhile, the internal (design) view of the plant can also be proceeding, particularly critical items such as design of the fusion reactor and its containment vessel, cooling systems, etc. Initial values from the external view (requirements) would be used to size and design these systems. Problems and insight encountered in the subsystem design would be used to refine the plant requirements, and the external and internal views cycle back and forth until an engineering balance is achieved. At that point a viable overall system emerges.

This is a pretty standard engineering design process. The main thing that Freedom brings to the table is a clear differentiation between requirements (black box) information and design (gray/white box) information. Freedom is not so much a process for developing systems as it is a set of concepts and definitions for characterizing system information. I have said it before, but will say it again, because it warrants repeating:

The concepts and definitions are the essence of Freedom.

All other aspects of the methodology derive directly from them, or at least must be compatible with them. If you can accept its concepts and definitions, then you have a fighting chance of accepting Freedom. If you do not believe its concepts and definitions, it will likely be difficult accepting much else about it.

Also, as the above example hinted, the requirements side of Freedom is just as applicable to hardware systems engineering as to software systems engineering. That follows directly from the definition of a black box, i.e, 'implementation unknown.' Inside the black box could be software, hardware, and any combination thereof. The requirements don't know, and don't care, at least in principle.

Rick Lutowski

From: Brad Appleton
Rick Lutowski wrote:
> Freedom does have a mechanism to help assess quality, but it is at a rather course-grained level. This is the notion of 'quality requirements.' One of the quality requirements is Usability, which does relate to the external interface, primarily the human user facet.

I guess I need some help seeing how this (and Freedom) relates to a lot of the software I deal with as part of an internal IT development organization. For example:

Much of the software I deal with will have multiple user-interfaces, and often for more than one audience. One application may need to have both a command-line and a GUI. In fact it may need to have both command-line and GUI for each of the following kinds of users of the system:

- typical users or 'operators' who enact the workflow business processes
- analysts who may not be workflow 'actors' but who will define how the resulting information is viewed and reported and sliced+diced
- 'business' administrators who define and configure 'business rules' for notification, access-control, workflow,
- administrators who setup/configure/upgrade the application and it administer/maintain its repositories
- application customizers/integrators who may have source/API-level visibility to various libraries and services in order to customize or extend/adapt the application to their environment
- other applications and/or application protocols (e.g., software and network interfaces for messaging, distribution, parallel processing

And in fact there may be several such application for the 'whole system', such that there are in fact several kinds and layers of user-interface at each level of 'scale' and 'audience'. Focusing on a GUI or command-line for the outermost layers would seem to ignore or significantly defer important fundamental 'interfaces' at the lower-levels.

But I guess what gives me even more difficulty is the apparent emphasis on 'form' above 'fit' and 'function'. By that I mean the apparent focus on the external human interface MORE SO than on the essential functionality to be carried out, and the interactions that need to happen (with knowledge the interactions need to exist, and where, but without being concerned [at least in the functional requirements] about the specific user-interface (or that there might be multiple interfaces for doing it]

Its seems to me that, thus far, Freedom seems to be predicated upon a certain level of requirement having already been captured before it can begin. It seems I already have to have business-level and functional system & software requirements before I can even start to ask what the software/system interface requirements are.

It looks to me like Freedom starts at what I would consider the earliest phases of design, and assumes the initial set of business and system/software functional requirements have already been captured?

Or else Freedom is perhaps trying to 'derive' those and flesh them out by first focusing on the user-interface (in which case I still have the issue of multiple interfaces and multiple kinds of interfaces). --

Brad Appleton

From: argnosis
Brad says
> It seems to me that, thus far, Freedom seems to be predicated upon a certain level of requirement having already been captured before it can begin. '

Off I go on a tangent.

An existing system is always there. We are (nearly always) automating something which exists in another form. So the requirements usually do exist. I think, from my little knowledge so far, that Freedom is about the appropriate place to start identifying requirements. If you get them right, the rest is a lot easier.

Peter

From: Rick Lutowski
Brad Appleton wrote:
> Much of the software I deal with will have multiple user-interfaces, and often for more than one audience. One application may need to have both a command-line and a GUI. In fact it may need to have both command-line and GUI for each of the following kinds of users of the system:

Freedom defines requirements as the external interface
(NOT just the user interface), where

external interface ==
human user interface(s) + external system interface(s) + environment interfaces(s)

The interfaces can take any form, or multiple forms. Freedom does not preclude anything that you mention.

> But I guess what gives me even more difficulty is the apparent emphasis on ‘form’ above ‘fit’ and ‘function’. By that I mean the apparent focus on the external human interface MORE SO than on the essential functionality to be carried out, and the interactions

This is a misperception resulting from the fact that much remains to be explained. (Recall we are still on post 3 of 7.)

Freedom does not emphasize form over function. The functionality trees and behavior tables (yet to be explained) record function with almost no regard to form. Form is recorded by functionality screens and protocol format specs (not scheduled for explanation as they are not essential to the goal of understanding requirements encapsulation). Freedom considers form and function equally important because both are aspects of the requirements (external interface), and one aspect of the requirements is not in principle more important than another.

>... It seems I already have to have business-level and functional system & software requirements before I can even start to ask what the software/system interface requirements are. It looks to me like Freedom starts at what I would consider the earliest phases of design, and assumes the initial set of business and system/software functional requirements have already been captured?

You are correct in assuming Freedom assumes some work has been done a priori. Freedom calls this work ‘enterprise analysis’ or ‘mission analysis’. It is outside the scope of software development because it is a higher-level process in which the customer process-to-be-improved is analyzed. Some of the process improvement strategy may involve software; but parts (or all) of the strategy may have nothing at all to do with software. Hence, enterprise analysis is not part of any software development process (although a software development process may, in some cases, be triggered by the enterprise analysis process.)

Also, and perhaps more importantly, your statement above indicates you may not yet be on Freedom's requirements ‘wavelength.’ I attribute this to deficiencies in my previous explanations, so I will attempt to explain Freedom's view of requirements from a slightly different perspective:

In the above you mention --

* functional requirements
* business-level requirements
* system requirements
* software requirements
* interface requirements

and there are other categories, as you know.

It is fashionable in the software industry to play what might be called the ‘tax game’ with types of requirements.

Substitute ‘Requirement’ for ‘Tax’ in the following list and you should get the flavor of what I mean:

Accounts Receivable Tax
Building Permit Tax
Capital Gains Tax
CDL license Tax
Cigarette Tax
Corporate Income Tax
Dog License Tax
Federal Income Tax
Federal Unemployment Tax
Fishing License Tax
Food License Tax
Fuel permit tax
Gasoline Tax
Hunting License Tax
Inheritance Tax
Inventory tax
Liquor Tax
Local Income Tax
Luxury Taxes
Marriage License Tax
Medicare Tax
Property Tax
Real Estate Tax
Septic Permit Tax
Service Charge Taxes
Social Security Tax
Road Usage Taxes
Sales Taxes
Recreational Vehicle Tax
Road Toll Booth Taxes
School Tax
State Income Tax
State Unemployment Tax
Telephone federal excise tax
Telephone federal universal service fee tax
Telephone minimum usage surcharge tax
Telephone recurring and non-recurring charges tax
Telephone state and local tax
Telephone usage charge tax
Toll Bridge Taxes
Toll Tunnel Taxes
Trailer registration tax
Utility Taxes

Vehicle License Registration Tax
Vehicle Sales Tax
Water Craft registration Tax
Well Permit Tax
Workers Compensation Tax

We aren't yet as ‘comprehensive’ with types of requirements as our government is with taxes, but it sure seems like we are trying hard to get there!

I believe a long litany list of types of requirements is symptomatic of the industry not understanding what requirements really are (as admitted by Wiegers in his book ‘Software Requirements’). Since we don't know exactly what requirements really are, we shotgun the field with terminology in the hopes that some of the terms will hit the target for our particular project.

Freedom cuts through all that by greatly simplifying requirements nomenclature and thinking. It does so by invoking a simple underlying model -- the Mills box-based model. According to this model, requirements are NOT a long litany of desires tied to every conceivable aspect of the customer, his process, systems, software, components, architectures, yadda yadda. Rather, the box-based model leads to a very succinct definition of requirements: the ‘black box view of the software system’, which is equivalent to the external interface, or the system's externally-visible information set. ONLY the system's external information comprises requirements, and nothing else -- by definition.

One can now safely forget the litany of requirements types and the need to shotgun the field with requirements terminology. Externally-visible information about the system is requirements, and everything else is not. According to this view, there are only two kinds of requirements (externally-visible information). Freedom calls them:

* functionality (or capability) requirements
* quality requirements.

(Note: the 7 posts are covering only functionality requirements. Quality requirements are being excluded, even tho they are important, because they are not critical to our goal of understanding requirements encapsulation.)

That's it. No long litany list proclaiming requirements for every possible aspect of the system. Most of these things are no longer requirements, by definition. The remainder that do meet the definition of requirements fit cleanly into one of the two above categories. The two-category list is comprehensive and complete; no other categories of requirements are needed when the box-based model is employed.

The linguistics folks have demonstrated that how we talk DOES influence how we think, and how we act. By greatly simplifying requirements nomenclature, Freedom greatly simplifies the mental, and the physical, effort involved with specifying requirements. It's really that simple.

Freedom seems to be predicated upon a certain level of requirement having already been captured before it can begin.

Freedom assumes ‘enterprise analysis’ or ‘mission analysis’ has been done prior to developing the software. This is a reasonable assumption given that such analysis is the customer process by which they decided some new software was needed in the first place! During the enterprise analysis process, the customer also gains some very specific ideas as to what the desired software should do. Most other methodologies couch all this under the banner of ‘requirements’. Freedom does not, as its definition is very specific. The customer will have definite ‘ideas’ or ‘goals’ or whatever you want to re-label them, but they have yet to be captured in the form of ‘requirements,’ i.e., a stimulus-response-protocol specification of human interfaces, external system interfaces, and environment interfaces of the system.

Freedom's approach works (and usually very well) for at least two reasons:

1. Everything ‘the software can do’ must be reflected in the external interface, otherwise the software can't do it! This makes Freedom's definition of requirements fundamentally consistent with our intuitive notion of requirements. (I.e., the black box-based definition is not such a radical departure as it may first seem.)

2. Even a customer who is highly development-illiterate is likely to be conversant in the ‘language’ of external interfaces (at least the human user facet) due to being exposed to interfaces every time they turn their computer on. Thus, the external interface is the one _technical_ communication vehicle that customers and developers will nearly always have in common. For much the same reason, it is the ONLY aspect of the system that many users really give a darn about, as it is the only part they will ever see. The external interface is thus both a natural starting point, and an effective requirements capture language, for developers who place high value on communication with the customer, and meeting user needs and expectations.

Freedom invites developers to learn to capture requirements, in concert with the customer, directly in the language of the external interface.

Rick Lutowski

From: Rick Lutowski
argnosis (Peter) wrote:
> An existing system is always there. We are (nearly always) automating something which exists in another form. So the requirements usually do exist. I think, from my little knowledge so far, that Freedom is about the appropriate place to start identifying requirements. If you get them right, the rest is a lot easier.

I agree with Peter. The requirements, or some precursor to them, always exist prior to initiation of development. Often they are in the form of ideas about the software in the mind of the customer. Sometimes they are in the form of legacy software, and need to be reverse engineered and updated. Never have I seen a customer hire a development contractor without

first having a pretty good idea of what they want, at least in their own mind. Extracting, refining, and transforming those ideas from the customer's mind, or legacy sw, or where ever they may be, into a technical format that is clear and useful to the development team is the issue.

When looked at from this perspective, Freedom simply offers an alternate way for performing the transform that has some unusual characteristics.

Rick Lutowski

From: Steven Gordon
Rick Lutowski wrote
> I agree with Peter. The requirements, or some precursor to them, always exist prior to initiation of development. [...] Never have I seen a customer hire a development contractor without first having a pretty good idea of what they want, at least in their own mind. Extracting, refining, and transforming those ideas from the customer's mind, or legacy sw, or where ever they may be, into a technical format that is clear and useful to the development team is the issue.

In my experience ‘extracting, refining, and transforming those ideas from the customer's mind ... into a technical format’ is not the critical success factor that it would seem to be.

The requirements success factors I have discovered over the years are: - discovering what the actual root problems are that motivate the solution the customer already has in mind (and then addressing those problems instead of blindly accepting the customer's solution). - figuring out what the customer might be mis-communicating as quickly as possible. - identifying what the developers are misunderstanding as quickly as possible. - helping the customer understand the full ramifications of various combinations of requirements in order to help find the most appropriate tradeoffs. - maximizing the impact of what the customers and developers learn during the project by committing to as many specific requirements as possible as late as possible.

I have found that iterative development done in customer priority order of requirements provides a practical way to maximize the appropriate combination of these success factors for the majority of projects.

I will find it critical to be able to apply Freedom iteratively as requirements are considered, reconsidered, and changed during the course of a project. I will reserve judgement as to whether that is possible until I have digested all the installments.

Steven Gordon

From: Brad Appleton
Rick Lutowski wrote:

> Freedom defines requirements as the external interface (NOT just the user interface), where
external interface ==
human user interface(s) + external system interface(s) + environment interfaces(s)
The interfaces can take any form, or multiple forms. Freedom does not preclude anything that you mention.

I was hoping that. And I was reassured somewhat by the earlier mention of system interface being more than just user interface.

The example given however (using AAR) seemed to focus on the UI rather than the system boundaries, and often even on elements of design rather than requirements. For example ...

You spoke of the different menu items such as File, Edit, View, ... Help. But didn't talk about the items in each of those menus. IMHO the grouping of the top-level menus as 'cohesive' and of treating the selection-list of each as as additional response-stimulus is merely due to the particular interface. I would have expected the set of stimuli grouped together under the 'File' menu to be much closer to a stimulus-set (and a much more cohesive one) than the top-level menu-items

User interface design at that level of detail seemed to me to assume that the majority of functional requirements (i.e. what the system is to do, as opposed to the design of the interface that lets me tell it to do it) are already known. From what I see, a typical use-case does that sort of thing. But I got the impression that Freedom does-away with use-cases and hence should be able to capture the same information that use-cases were intended to capture (only better :-)

> You are correct in assuming Freedom assumes some work has been done a priori. Freedom calls this work 'enterprise analysis' or 'mission analysis'. It is outside the scope of software development because it is a higher-level process in which the customer process-to-be-improved is analyzed. Some of the process improvement strategy may involve software; but parts (or all) of the strategy may have nothing at all to do with software. Hence, enterprise analysis is not part of any software development process (although a software development process may, in some cases, be triggered by the enterprise analysis process.)

I agree there has to be some notion of what is wanted a priori, and probably some kind of concept/vision description, and perhaps a business case was done. I would consider that all the work that takes place before the decision is made to fund and staff the project and create a project charter (or equivalent). I would still expect a lot of requirements gathering/analysis needs to be done by development after that point, and that it will be substantially more than UI, but would include gathering all the same kinds of info that one might otherwise attempt to capture in use-cases (Development still has to do that - right?)

> Also, and perhaps more importantly, your statement above indicates you may not yet be on Freedom's requirements 'wavelength.' [...] We aren't yet as 'comprehensive' with types of requirements as our government is with taxes, but it sure seems like we are trying hard to get there!

No - you're off base there (and with the 10 or so paragraphs that followed). The essence is that there is a certain amount of knowledge of what the system is to do that has already been described at some level. I'm trying to ascertain what that amount and level is. It is the requirements knowledge level/depth that is captured a priori that I'm looking at, not the various types of requirements within one or more 'levels'.

Assume for a moment a 'system' that includes both hardware and software. Some amount and depth of 'requirements' is needed before being able to get to the point of being able to differentiate which of those things will be 'allocated' to the software versus the hardware (or both). Once we get to the software, some more knowledge must be known before being able to dive into details of GUI design.

Which of those things does Freedom assume has already been done, and to what extent? Would Freedom (like many other SOFTWARE development methods) assume that we've already gotten far enough into 'system'-level capabilities and functions to at least know which broad-level of things are expected of the software as opposed to the HW? Would it completely ignore the HW or treat it as a possible stimulus source?

Some are proponents of a so called 'Systems Engineering' approach (and indeed, there are some instances of methods such as 'RUP' that are intended for 'Systems Engineering'). The 'Systems Engineering' methods likely would not be able to assume a priori gathered knowledge of what things are to be done by the SW versus the HW.

This is important because it speaks to the amount of 'system knowledge' already attempted to be gathered for which we may not yet have sufficient feedback to determine how much of it is 'valid' and how much of it is stale/out-of-date or simply wrong. And it speaks to the amount of knowledge development has still to gather before it can obtain such 'sufficient' feedback.

It will also help me determine just how capable Freedom seems to be (or not) of the kind of short and frequent iterations that are characteristic of agile methods and/or whether it seems to be more BRUF (big requirements up front).

With several agile methods (including XP), while it is true that there may be some folks who already have a good idea of what they want the system to do, it is unlikely that much of it has been captured in any formal manner other than a request 'ticket' in a change-tracking system that has a headline and a brief prose paragraph or two that is at a very high and vague level. Yet those development methodologies are able to start at that point (rather than require much more written knowledge to be formally captured).

Thus far I'm getting the impression Freedom would require more up-front analysis and description (than say, XP, or Crystal, or FDD, or ASD) before it could start to 'do its thing'

Brad Appleton

From: "Ashley McNeile"

Rick wrote
> You are correct in assuming Freedom assumes some work has been done a priori. Freedom calls this work ‘enterprise analysis’ or ‘mission analysis’. It is outside the scope of software development because it is a higher-level process in which the customer process-to-be-improved is analyzed. Some of the process improvement strategy may involve software; but parts (or all) of the strategy may have nothing at all to do with software. Hence, enterprise analysis is not part of any software development process (although a software development process may, in some cases, be triggered by the enterprise analysis process.)

I think that what you describe here as a ‘higher-level process’ is actually a lot of what I do with executable modelling -- namely explore the relationship between processes and IT, sometimes with a view to radical process improvement.

I think I feel more comfortable with what you are saying if you regard this as out of scope for Freedom. It confirms my suspicion that were are thinking about different domains of applicability.

Ashley

From: Rick Lutowski
Ashley McNeile wrote:
> I think that what you describe here as a ‘higher-level process’ is actually a lot of what I do with executable modelling -- namely explore the relationship between processes and IT, sometimes with a view to radical process improvement.

Yes, and there are process modeling tools out there (for a price) that will 'execute' process models -- which I have always taken to mean 'simulate'. Perhaps you can provide more detail since you do this sort of thing.

> I think I feel more comfortable with what you are saying if you regard this as out of scope for Freedom. It confirms my suspicion that were are thinking about different domains of applicability.

As mentioned previously, this is a level or two (or more) above Freedom. Glad the boundaries are becoming clearer and the waters are perhaps starting to de-murk a bit.

Also, Brad has a post in my inbox which is fishing for more detail in this area. So check my reply to him for more info along these lines. Hope to get it out in a little while.

Thanks.

Rick Lutowski

From Ashley McNeile
> > Perhaps you can provide more detail since you do this sort of thing.

> Info is available at www.metamaxim.com .

Did you manage to take a look at this? Did it make any sense to you?

Ashley

From Rick Lutowski
Ashley McNeile wrote:
> Info is available at www.metamaxim.com. Did you manage to take a look at this? Did it make any sense to you?

Now I see where your are coming from! A couple questions before I venture any comments:

1. The 'products' page indicates modelscope can handle STDs. Can it currently handle any other types of models?

2. If it runs on a JDK, why does it require Windows with IE? Why can't it work with Linux/Mozilla or Mac/Safari?(If it can, you should update the page to indicate this.)

Rick Lutowski