

DESIGN GUIDELINES

Successful product design is about maintaining balance. At an organizational level, it is a balance between business needs, user desires, and technology constraints. At the product level, it is a balance between vision, form and function. In both cases, the key to real success lies in thoroughly addressing each of those things without allowing any one to overshadow the rest. The process of evaluating and iterating a product is thus an exercise in understanding and identifying these elements and then applying the fundamental rules of good design.

These Design Guidelines are intended to provide insight into strong interface design practices and techniques, using the current Agile product as a baseline. In evaluating the PLM suite, meeting with stakeholders, and delving into user feedback and response, our team has identified eight persistent interface issues. These issues are illustrated here, along with related explanations of their design principles. Also provided are examples of actual solutions can serve as a guideline for the Agile team's ongoing evaluation of PLM development and the hands-on work of the design team.

Paul Rand famously said, "Simplicity is not the goal. It is the by-product of a good idea and modest expectations." This approach is a hallmark of great design and represents an aesthetic that we strive to help the Agile team create.

- I. Mental Model
2. Structure
3. Emphasis
4. Typography
5. Color
6. Interface noise
7. Direct manipulation
8. Terminology
9. Flattening

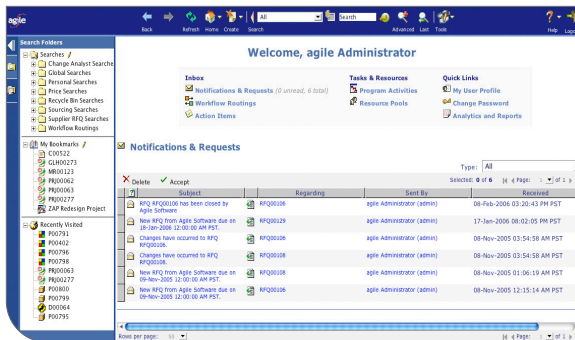
Prepared for **Agile Software**
Confidential & Proprietary
March 22, 2006

Comments to:
andrei.herasimchuk@agile.com
donna.driscoll@agile.com

I

MENTAL MODEL

If you don't know where you are, you cannot get where you want to go.



When the user logs in to the Agile Client, it is unclear where they should look to begin their work. Agile is, at its core, a large document management system; a hub for managing a range of products. Yet this metaphor is not apparent in the current application. How best would this be represented?

Establishing the proper mental model is fundamental to driving the product design process.

Mental models are typically expressed in the form of metaphor; for example, the desktop metaphor used by modern operating systems represents a very specific - and intentional - mental model.

Simplicity is the hallmark of a strong mental model: users should intuitively understand the model they are presented

Software design begins with a deep understanding of the purpose that the product fills for users. While this understanding is important in order to design successful interfaces, it is perhaps more important in framing the product properly in the user's mind.

This is called a mental model.

People's understanding of the world is based on a very complicated, integrated and contextually-dependent worldview. Even though each person has a unique understanding, there are clear patterns emerge based on social, cultural and environmental factors. Each product that a person uses or interaction they experience fits into their view and understanding of the

world. Indeed, each of these things conforms to a specific mental model in the mind of each person. Based on their past experiences and perspective, that mental model will define how they think and feel about this thing, as well as how they use it.

It is thus imperative that a software product is intentionally designed to fit a specific mental model with its

users. If it is successfully designed within that structure, the product will be increasingly usable and desirable. More, this mental model will also inform the initial and ongoing interface design, serving as a baseline for all of the design decisions that are eventually made. It is key to user understanding and fundamental to the product design road map.

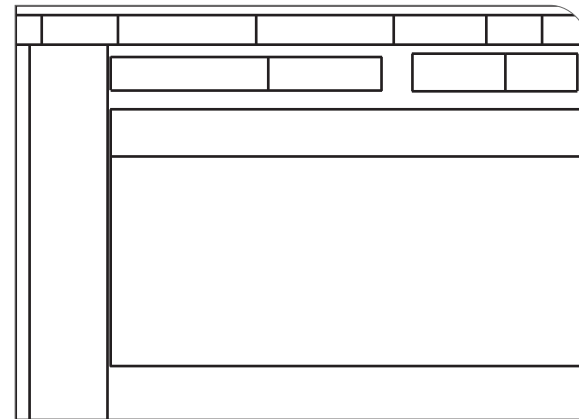
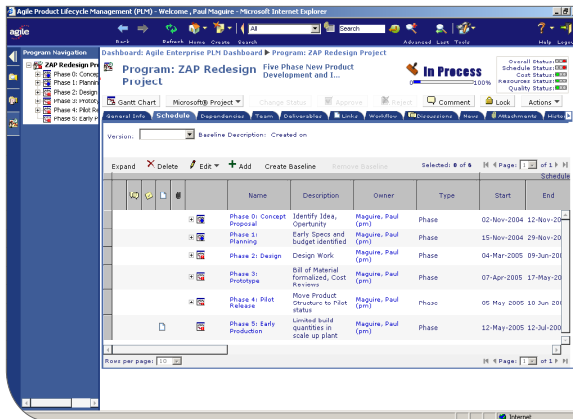
Prepared for **Agile Software**
Confidential & Proprietary
March 22, 2006

Comments to:
andrei.herasimchuk@agile.com
donna.driscol@agile.com

2

STRUCTURE

You'd never build your house before you had the right blueprint.



Wireframes are based on real ratios; if type is used inside wireframes, it also must respect the ratio.

Wireframes must follow a well-defined grid.

If the wireframe is overly complex at a visual level, the product will naturally become too difficult to construct or design.

Before any software application can be built, a design blueprint must be constructed and agreed on by the product team. This blueprint serves as a high level model of how the product will be built.

Often times, people will use detailed wireframes for these blueprints. While acceptable, these wireframes must be reduced to the most basic level in

order to serve their primary purpose: defining the overall structure of an application. The goal of the wireframe is to create a foundation for the layout of the application, not serve as a low level detail design specification.

To follow the analogy, one is only concerned with knowing where the bedroom is, how big it is, and how it connects to the rest of the house. What

color the walls are, or where the bed will be placed inside of the room are decisions that can only be explored once these higher level decisions have been made.

The reason for this is simple: if wireframes, acting as blueprint, are overly complex themselves, it stands to

reason that once the depth of content and data is placed into it, the end result will be sufficiently difficult to manage.

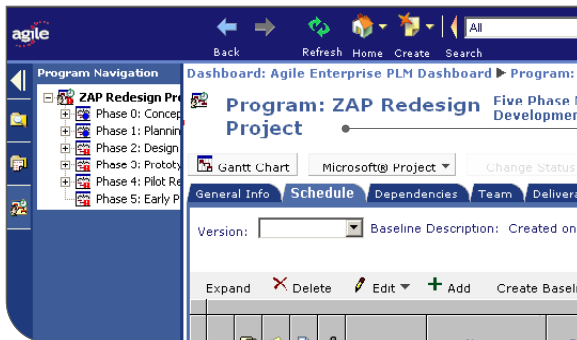
Prepared for **Agile Software**
Confidential & Proprietary
March 22, 2006

Comments to:
andrei.herasimchuk@agile.com
donna.driscoll@agile.com

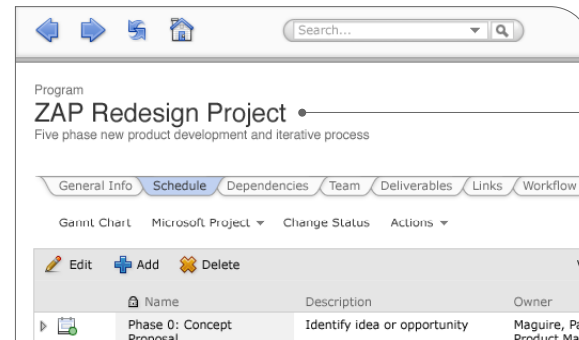
3

EMPHASIS

When everything is important, nothing is important.



The name of the main object that defines the data on the screen is lost on a screen that lacks proper emphasis.



By applying visual consistency to the layout, typography and color of the content, in conjunction with appropriate use of ratios and balance, the name of the main object gains proper emphasis.

There should always be a single focus on the screen acting as a visual anchor, even if that point is not the primary concern of the user.

Create boundaries and scope. Limit emphasized options or controls up to three and no more than five.

Pick an agreed upon measurement to act as an acid test. Do not deviate from it. For example, determine that for any control or feature to be considered high priority, it must be used by 75% of the user base at least 25% of the time.

Quite simply, if one treats every feature, every control or every piece of data in a software application as the most important thing that must be addressed or promoted, the end result will essentially make the overall product less useful across the board.

Without contrast, nothing can be ascertained. Without proper emphasis, a product is reduced to a morass of

features that require a user to traverse it like a mouse searching desperately for that piece of cheese inside a maze.

One must constantly question and ask themselves just how important one feature is in comparison to every other feature in the design of any product.

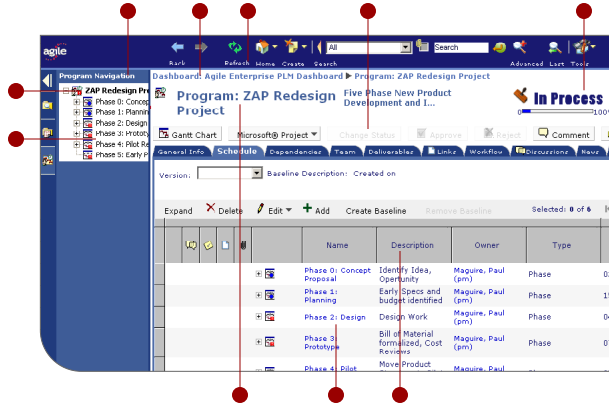
Prepared for Agile Software
Confidential & Proprietary
March 22, 2006

Comments to:
andrei.herasimchuk@agile.com
donna.driscoll@agile.com

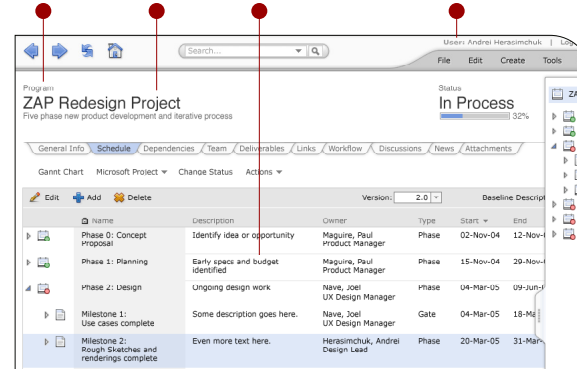
4

TYPOGRAPHY

When in doubt, follow the rule of three.



In the current Agile PLM product, there are at least 10 different font variations used.



An exploratory redesign shows the difference with reducing the font variations to 4.

As a starting point, use no more than three fonts in the design of the product.

A “font” is defined as the font face, the font color and the font variant. Font size is often considered as well. For example, 11pt. Helvetica colored as #333 and 11pt. Helvetica Italic colored as #333 is considered two different fonts.

Before deviating from the rule of no more than three fonts, first force oneself to solve the design problem within the constraints.

One of the easiest problems to fix in almost any design is poor application of basic typographical rules.

Words are the functional equivalent of icons. When you read a body of text, you are not parsing individual letters, like “d - e - s - i - g - n.” You are actually seeing a small symbol that is parsed as a single object, “design.”

One would never draw icons in an interface in radically different visual styles, so one should avoid doing so with type as well.

When type is set with excessive unevenness, it’s the equivalent of stuttering. So when in doubt, follow the rule of limiting yourself to no more than three fonts.

Only once you have type under control can you pick the optimal spots to break this rule.

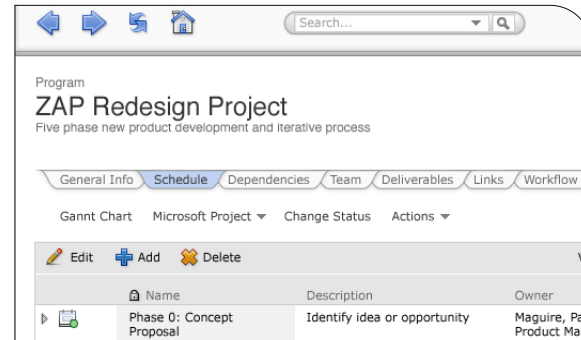
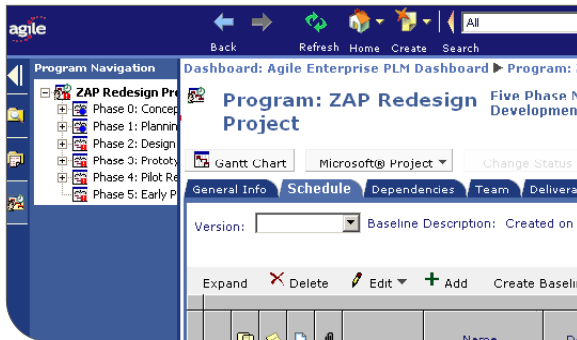
Prepared for Agile Software
Confidential & Proprietary
March 22, 2006

Comments to:
andrei.herasimchuk@agile.com
donna.driscoll@agile.com

5

COLOR

When used effectively, color elucidates as well as calms the user.



 Base	 Highlight
 Compliment	 Base Black
 Compliment	 Base White
 Compliment	

 Base	 Highlight
 Compliment	 Base Black
 Compliment	 Base White
 Compliment	

Define a product color palette. Be aware to pick a color system that also serves the needs of a software product at a functional level.

A good place to start is to pick a base color, up to three compliments, one highlight color and a defined level of black and white.

Before deviating from the color palette, first force oneself to solve the design problem within the constraints.

Color follows much of the same constraints as type, with the exception that color gets to the core emotional impact of the product in a more direct, visceral way that type does not.

Just like it's typographical counterpart, a color palette must be applied judiciously, carefully avoiding extraneous or random additions.

Where type speaks, color punctuates. Where type provides communication, color provides context.

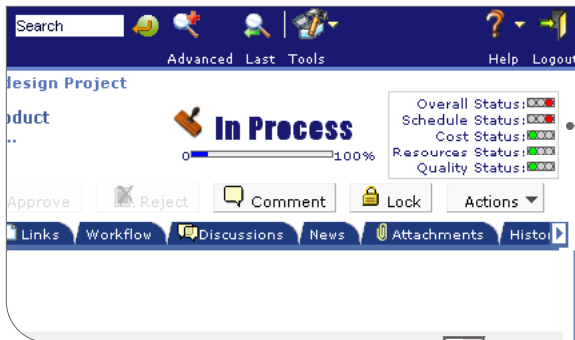
Prepared for **Agile Software**
Confidential & Proprietary
March 22, 2006

Comments to:
andrei.herasimchuk@agile.com
donna.driscoll@agile.com

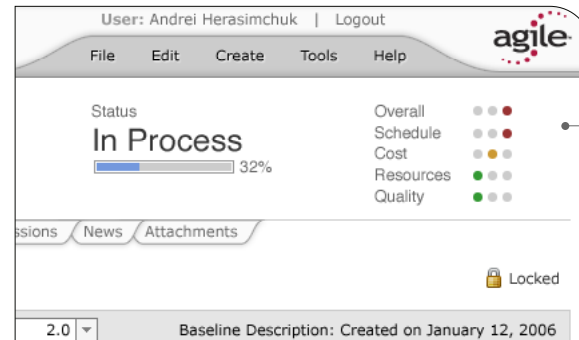
6

INTERFACE NOISE

It's rude to shout at someone through a megaphone.



Excessive line noise, ornamentation and lack of white space are preventing the Status and Overview controls from fulfilling their purpose.



Which is to clearly communicate what the current status is and quickly allow the user to see top level, red flag issues at a quick glance.

Human beings, as a general rule, are well equipped to parse large sets of data and content, but only when the interface or its presentation gets out of the way.

Remove controls and functions that fall below some agreed upon threshold level of interaction.

Always question, "what is the user trying to do here?" and measure the interface against that task.

People don't use software to use an interface. They use software to achieve a specific purpose; buying a book, writing a term paper, filling out an expense report, listening to music.

As such, one must always remember that great interface design is like great set design, it only provides the context for a production to occur. It's the story and the actors people came to see, not

to ogle at the ingeniousness of the set designer's technical prowess. If the actors are constantly tripping on the set or if the story calls for a location to be in Scotland and the set appears to be in the Amazon jungle, then the set has failed its purpose.

In short, interface noise is everything that gets in the way of the interface fulfilling its reason for being.

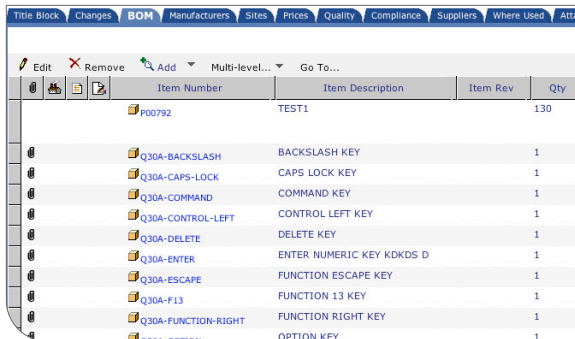
Prepared for Agile Software
Confidential & Proprietary
March 22, 2006

Comments to:
andrei.herasimchuk@agile.com
donna.driscoll@agile.com

7

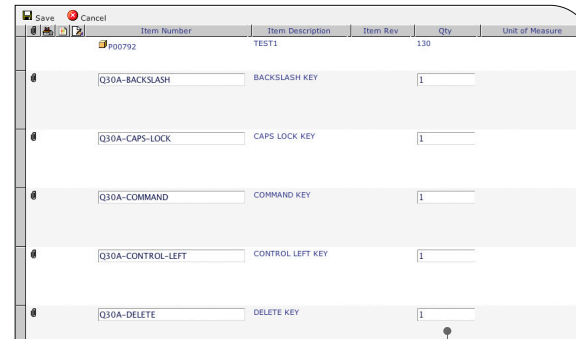
DIRECT MANIPULATION

Second hand information is never as reliable as first hand experience.



Item Number	Item Description	Item Rev	Qty
P00792	TEST1		130
Q30A-BACKSLASH	BACKSLASH KEY	1	
Q30A-CAPS-LOCK	CAPS LOCK KEY	1	
Q30A-COMMAND	COMMAND KEY	1	
Q30A-CONTROL-LEFT	CONTROL LEFT KEY	1	
Q30A-DELETE	DELETE KEY	1	
Q30A-ENTER	ENTER NUMERIC KEY KDKDS D	1	
Q30A-ESCAPE	FUNCTION ESCAPE KEY	1	
Q30A-F13	FUNCTION 13 KEY	1	
Q30A-FUNCTION-RIGHT	FUNCTION RIGHT KEY	1	
Q30A-OPTION	OPTION KEY	1	

To change the quantity of multiple parts in a BOM, the user must first select the desired parts and click edit.



Item Number	Item Description	Item Rev	Qty	Unit of Measure
P00792	TEST1		130	
Q30A-BACKSLASH	BACKSLASH KEY		1	
Q30A-CAPS-LOCK	CAPS LOCK KEY		1	
Q30A-COMMAND	COMMAND KEY		1	
Q30A-CONTROL-LEFT	CONTROL LEFT KEY		1	
Q30A-DELETE	DELETE KEY		1	

Then, the user must manually enter the same quantity for each of the parts. If direct manipulation were supported, the same operation could be applied to multiple parts simultaneously in a single step.

If the interface lags in response at any level during a direct manipulation operation, no matter how good the product's feature set may be, it will always be perceived by the general user in a negative light.

Obviousness is not the most important aspect of direct manipulation; short learning curves and usefulness over the long term is.

All direct manipulation operations require feedback, even in cases when that feedback is an explanation of an error.

A user's assessment of how usable a software product is typically is a direct response to and specific comment about how successfully that product handles direct manipulation.

By definition, software is designed to allow people to do things. Most of the things people want to do with software require direct manipulation: selecting and moving an object, tagging

or otherwise identifying an object, creating relationships between various objects: these very simple tasks that are common to many software products ultimately determine whether the product is useful or not. Therefore, it is important that ample design time and thought go into the process of how people complete direct manipulation operations.

One of the most common mistakes in interface design today is attempting to make everything *intuitive*. In products that are built for ongoing use, being intuitive is actually one of the *least important* design objectives. What is important is a relatively short learning curve; what is important is that the product be optimized for very efficient and effective use over the long haul.

The goal should never be to gear your product for usability in the first three seconds; the goal should be to make your product usable for the many hundreds of times it is used thereafter.

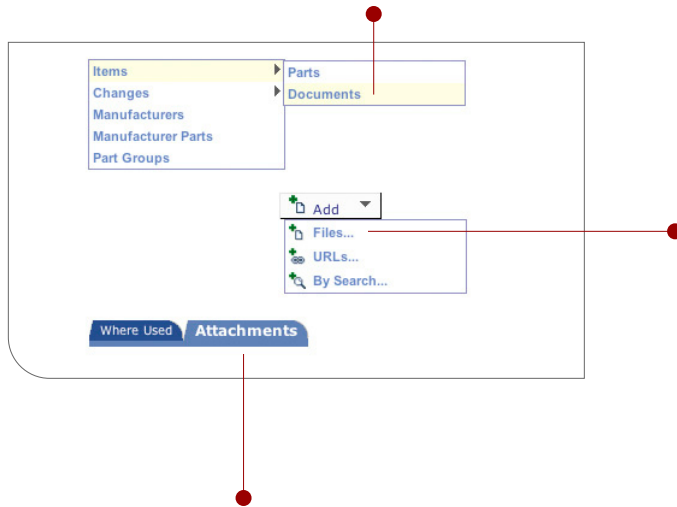
Prepared for Agile Software
Confidential & Proprietary
March 22, 2006

Comments to:
andrei.herasimchuk@agile.com
donna.driscol@agile.com

8

TERMINOLOGY

Always ask, 'What is this?' until you are given the final answer.



Although there may be legitimate technical reasons to refer to Attachments, Files and Documents as unique types of items, in casual language these are three labels for the same thing – documents – and the distinction between them is unlikely to be clear to users.

The purpose of marketing is to sell, which makes the method by which marketing determines salable names inappropriate for determining terminology inside of a functional interface. As much as marketing managers want to control the design of a product, their job as defined does not serve the product nor the user well.

Generally speaking, terminology should be no more complicated than common language in everyday conversation.

Create terms without any attempt at cleverness, complex word construction or by combining too many words into a single term that would prompt the use of a dictionary.

Just call things what they are.

It's really that simple.

Avoid marketing labels in the interface; call things what they are at the mundane object level.

Always use industry standard language when possible. For example, designers know that "leading" is the measurement between two baselines in body copy. To refer to "leading" as "line height" will simply confuse the target audience.

If you can't answer the question, "what is this" in a single sentence with regard to a term in the interface, it is likely you have misused the term.

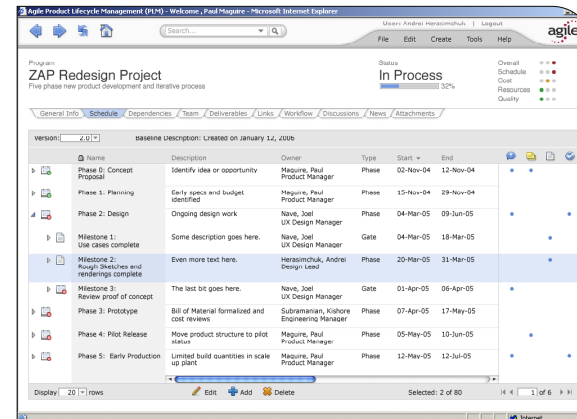
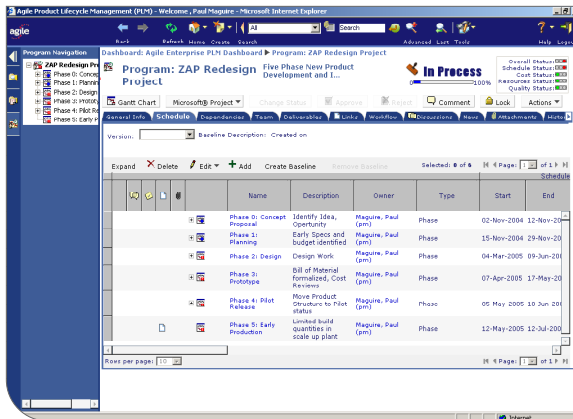
Prepared for Agile Software
Confidential & Proprietary
March 22, 2006

Comments to:
andrei.herasimchuk@agile.com
donna.driscoll@agile.com

9

FLATTENING

“Complexity increases at an exponential rate as a factor of the number of items presented.”



Visual: Remove all extraneous line noise and ornamentation. First achieve visual simplicity before decorating with flourishes.

Information: Always check the screen for words and symbols that have been repeated in proximity of each other. Then consolidate them.

Behavioral: Pick a primary method for performing an action and feature it prominently. Controls and interactions that duplicate the behavior be hidden one level deeper.

Products that suffer from complex interfaces often will find at the root of the problem a large amount of extraneous redundancy. This redundancy cuts across the board and is not limited in scope. It generally interferes at all levels of the interface: the visual, informational and behavioral aspects of the product. It's extra visual line noise, repetitive informational

symbols and too many controls ways to perform the same action all visible at the same time. Imagine if you will attempting to drive a car with three dashboards, four steering wheels, and highway signs that were consistently duplicated on both sides of the road.

A significant amount of complexity can be removed from the product through the simple task of flattening. So

much that often times true innovative design cannot be accomplished until a product's interface has been sufficiently flattened.

Prepared for Agile Software
Confidential & Proprietary
March 22, 2006

Comments to:
andrei.herasimchuk@agile.com
donna.driscoll@agile.com