

User Interface Design: Tips and Techniques

Scott W. Ambler
President, Ronin International



Material for this paper has been excerpted from:
The Object Primer 2nd Edition,
Building Object Applications That Work,
and
Process Patterns

All written by Scott W. Ambler and published by
Cambridge University Press

<http://www.ambysoft.com/userInterfaceDesign.pdf>

Finalized: October 26, 2000

Copyright 1998-2000 Scott W. Ambler

Table Of Contents

1. USER INTERFACE DESIGN TIPS AND TECHNIQUES	1
2. PROTOTYPING.....	4
2.1 PROTOTYPING TIPS AND TECHNIQUES.....	6
3. INTERFACE-FLOW DIAGRAMS	7
4. WHERE TO GO FROM HERE.....	9
5. SUMMARY.....	10
5.1 GENERAL GUIDELINES.....	10
5.2 SCREEN DESIGN	10
5.3 PROTOTYPING	11
6. REFERENCES AND RECOMMENDED READING.....	12
7. ABOUT THE AUTHOR.....	13

A fundamental reality of application development is that the user interface is the system to the users. What users want is for developers to build applications that meet their needs and that are easy to use. Too many developers think that they are artistic geniuses – they do not bother to follow user interface design standards or invest the effort to make their applications usable, instead they mistakenly believe that the important thing is to make the code clever or to use a really interesting color scheme. Constantine (1995) points out that the reality is that a good user interface allows people who understand the problem domain to work with the application without having to read the manuals or receive training.

For most people the user interface is the software.

Interface design is important for several reasons. First of all the more intuitive the user interface the easier it is to use, and the easier it is to use the cheaper it is. The better the user interface the easier it is to train people to use it, reducing your training costs. The better your user interface the less help people will need to use it, reducing your support costs. The better your user interface the more your users will like to use it, increasing their satisfaction with the work that you have done.

The point to be made is that the user interface of an application will often make or break it. Although the functionality that an application provides to users is important, the way in which it provides that functionality is just as important. An application that is difficult to use won't be used. Period. It won't matter how technically superior your software is or what functionality it provides, if your users don't like it they simply won't use it. Don't underestimate the value of user interface design.

1. User Interface Design Tips and Techniques

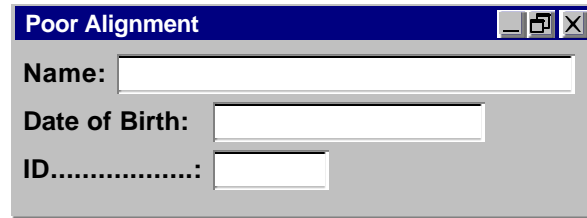
In this section we will cover a series of user interface design tips that will help you to improve the object-oriented interfaces that you create.

1. **Consistency, consistency, consistency.** The most important thing that you can possibly do is make sure that your user interface works consistently. If you can double-click on items in one list and have something happen then you should be able to double-click on items in any other list and have the same sort of thing happen. Put your buttons in consistent places on all of your windows, use the same wording in labels and messages, and use a consistent color scheme throughout. Consistency in your user interface allows your users to build an accurate mental model of the way that it works, and accurate mental models lead to lower training and support costs.
2. **Set standards and stick to them.** The only way that you'll be able to ensure consistency within your application is to set design standards and then stick to them. The best approach is to adopt an industry standard and then fill any missing guidelines that are specific to your needs. Industry standards, such as the ones set by IBM (1993) and Microsoft (1995), will often define 95%-99% of what you need. By adopting industry standards you not only take advantage of the work of others you also increase the chance that your application will look and feel like other applications that your users purchase or have built. User interface design standards should be set during the Define Infrastructure Stage (Ambler, 1998b).
3. **Explain the rules.** Your users need to know how to work with the application that you built for them. When an application works consistently it means you only have to explain the rules once. This is a lot easier than explaining in detail exactly how to use each and every feature in an application step by step.
4. **Support both novices and experts.** Although a library-catalog metaphor might be appropriate for casual users of a library system, library patrons, it probably is not all that effective for expert users, librarians. Librarians are highly trained people who are able to use complex search systems to find information in a library, therefore you should consider building a set of search screens to support their unique needs.

5. **Navigation between screens is important.** If it is difficult to get from one screen to another then your users will quickly become frustrated and give up. When the flow between screens matches the flow of the work that the user is trying to accomplish, then your application will make sense to your users. Because different users work in different ways, your system will need to be flexible enough to support their various approaches. Interface-flow diagrams can be used during the Model Stage (Ambler, 1998b) to model the flow between screens.
6. **Navigation within a screen is important.** In Western societies people read left to right and top to bottom. Because people are used to this should you design screens that are also organized left to right and top to bottom. You want to organize navigation between widgets on your screen in a manner that users will find familiar to them.
7. **Word your messages and labels appropriately.** The text that you display on your screens is a primary source of information for your users. If your text is worded poorly then your interface will be perceived poorly by your users. Using full words and sentences, as opposed to abbreviations and codes makes your text easier to understand. Your messages should be worded positively, imply that the user is in control, and provide insight into how to use the application properly. For example, which message do you find more appealing “You have input the wrong information” or “An account number should be 8 digits in length.”? Furthermore, your messages should be worded consistently and displayed in a consistent place on the screen. Although the messages “The person’s first name must be input.” and “An account number should be input.” are separately worded well, together they are inconsistent. In light of the first message, a better wording of the second message would be “The account number must be input” to make the two messages consistent.
8. **Understand your widgets.** You should use the right widget for the right task, helping to increase the consistency in your application and probably making it easier to build the application in the first place. The only way that you can learn how to use widgets properly is to read and understand the user-interface standards and guidelines that your organization has adopted.
9. **Look at other applications with a grain of salt.** Unless you know that another application follows the user-interface standards and guidelines of your organization, you must not assume that the application is doing things right. Although it is always a good idea to look at the work of others to get ideas, until you know how to distinguish between good user-interface design and bad user-interface design you have to be careful. Too many developers make the mistake of imitating the user interface of another application that was poorly designed.
10. **Use color appropriately.** Color should be used sparingly in your applications, and if you do use it you must also use a secondary indicator. The problem is that some of your users may be color blind – if you are using color to highlight something on a screen then you need to do something else to make it stand out if you want these people to notice it, such as display a symbol beside it. You also want to use colors in your application consistently so that you have a common look and feel throughout your application. Also, color generally does not port well between platform – what looks good on one system often looks poor on another system. We have all been to presentations where the presenter said “it looks good on my machine at home.”
11. **Follow the contrast rule.** If you are going to use color in your application you need to ensure that your screens are still readable. The best way to do this is to follow the contrast rule: Use dark text on light backgrounds and light text on dark backgrounds. It is very easy to read blue text on a white background but very difficult to read blue text on a red background. The problem is that there is not enough contrast between blue and red to make it easy to read, whereas there is a lot of contrast between blue and white.
12. **Use fonts appropriately** – Old English fonts might look good on the covers of William Shakespeare’s plays, but they are really hard to read on a screen. Use fonts that are easy to read, such as serif fonts

like Times Roman. Furthermore, use your fonts consistently and sparingly. A screen using two or three fonts effectively looks a lot better than a screen that uses five or six. Never forget that you are using a different font every time you change the size, style (bold, italics, underlining, ...), typeface, or color.

13. **Gray things out, do not remove them.** You often find that at certain times it is not applicable to give your users access to all the functionality of an application. You need to select an object before you can delete it, so to reinforce your mental model the application should do something with the Delete button and/or menu item. Should the button be removed or grayed out? Gray it out, never remove it. By graying things out when they shouldn't be used people can start building an accurate mental model as to how your application works. If you simply remove a widget or menu item instead of graying it out then it is much more difficult for your users to build an accurate mental model because they only know what is currently available to them, and not what is not available. The old adage that out of sight is out of mind is directly applicable here.
14. **Use non destructive default buttons.** It is quite common to define a default button on every screen, the button that gets invoked if the user presses the Return/Enter key. The problem is that sometimes people will accidentally hit the Enter/Return key when they do not mean to, consequently invoking the default button. Your default button shouldn't be something that is potentially destructive, such as delete or save (perhaps your user really did not want to save the object at that moment).
15. **Alignment of fields.** When a screen has more than one editing field you want to organize the fields in a way that is both visually appealing and efficient. As shown in Figure 1 I have always found that the best way to do so is to left-justify edit fields, or in other words make the left-hand side of each edit field line up in a straight line, one over the other. The corresponding labels should be right justified and placed immediately beside the field. This is a clean and efficient way to organize the fields on a screen.
16. **Justify data appropriately.** For columns of data it is common practice to right justify integers, decimal align floating point numbers, and left justify strings.
17. **Do not create busy screens.** Crowded screens are difficult to understand and hence are difficult to use. Experimental results (Mayhew, 1992) show that the overall density of the screen should not exceed 40%, whereas local density within groupings shouldn't exceed 62%.
18. **Group things on the screen effectively.** Items that are logically connected should be grouped together on the screen to communicate that they are connected, whereas items that have nothing to do with each other should be separated. You can use whitespace between collections of items to group them and/or you can put boxes around them to accomplish the same thing.
19. **Open windows in the center of the action.** When your user double-clicks on an object to display its edit/detail screen then his or her attention is on that spot. Therefore it makes sense to open the window in that spot, not somewhere else.
20. **Pop-up menus should not be the only source of functionality.** Your users cannot learn how to use your application if you hide major functionality from them. One of the most frustrating practices of developers is to misuse pop-up, also called context-sensitive, menus. Typically there is a way to use the mouse on your computer to display a hidden pop-up menu that provides access to functionality that is specific to the area of the screen that you are currently working in.

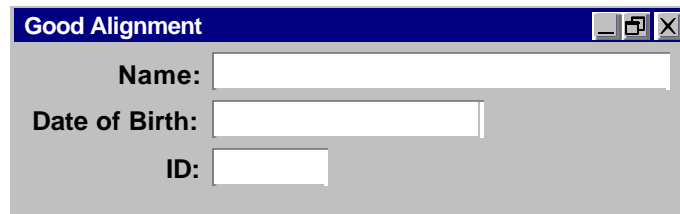


Poor Alignment

Name:

Date of Birth:

ID.....:



Good Alignment

Name:

Date of Birth:

ID:

Figure 1. Alignment of fields is critical.

2. Prototyping

Prototyping (Ambler, 2001; Ambler, 1998a) is an iterative analysis technique in which users are actively involved in the mocking-up of screens and reports. The purpose of a prototype is to show people the possible design(s) for the user interface of an application. As we see in Figure 2 there are four steps to the prototyping process:

1. **Determine the needs of your users.** The requirements of your users drive the development of your prototype as they define the business objects that your system must support. You can gather these requirements in interviews, in CRC (class responsibility collaborator) modeling sessions, in use-case modeling sessions, and in class diagramming sessions (Ambler 2001; Ambler, 1998a; Ambler, 1998b).
2. **Build the prototype.** Using a prototyping tool or high-level language you develop the screens and reports needed by your users. The best advice during this stage of the process is to not invest a lot of time in making the code “good” because chances are high that you may just scrap your coding efforts anyway after evaluating the prototype.
3. **Evaluate the prototype.** After a version of the prototype is built it needs to be evaluated. The main goal is that you need to verify that the prototype meets the needs of your users. I’ve always found that you need to address three basic issues during evaluation: What’s good about the prototype, what’s bad about the prototype, and what’s missing from the prototype. After evaluating the prototype you’ll find that you’ll need to scrap parts, modify parts, and even add brand-new parts.
4. **Determine if you’re finished yet.** You want to stop the prototyping process when you find the evaluation process is no longer generating any new requirements, or is generating a small number of not-so-important requirements.

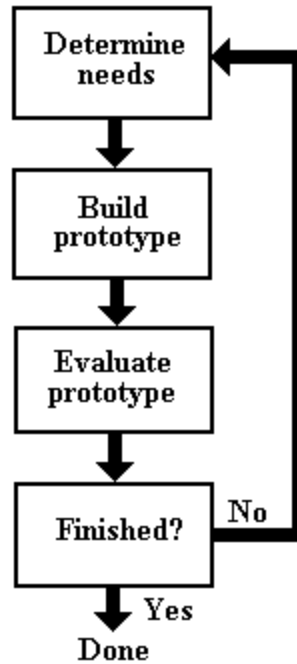


Figure 2. The iterative steps of prototyping.

Blatant Advertising – Purchase *The Object Primer, 2nd Edition*

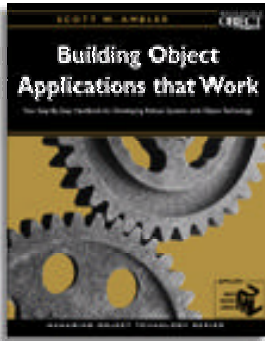


The Object Primer 2nd Edition is a straightforward, easy to understand introduction to object-oriented concepts, requirements, analysis, and design techniques applying the techniques of the Unified Modeling Language (UML). The Object Primer goes further to show you how to move from object modeling to object-oriented programming, providing Java examples, and describes the techniques of the Full Lifecycle Object-Oriented Testing (FLOOT) methodology to enable you to test all of your development artifacts. It is the first book that describes how to develop a system using object-oriented and relational database technology end-to-end, from requirements all the way to Java code and a RDB design. It also puts this material in the context of the leading software processes, including the enhanced lifecycle for the Unified Process, the process patterns of the Object-Oriented Software Process (OOSP), and the best practices Extreme Programming (XP). Visit www.ambysoft.com/theObjectPrimer.html for more details.

2.1 Prototyping Tips and Techniques

I have covered the fundamentals of the prototyping process, so now I want to share with you several tips and techniques that you can use to create truly world-class prototypes.

1. **Look for real-world objects.** Good UIs allow users to work with the real-world objects they are used to. Therefore you should start by looking for these kinds of objects and identify how people interact with them.
2. **Work with the real users.** The best people to get involved in prototyping are the ones who will actually use the application when it's done. These are the people who have the most to gain from a successful implementation, and these are the ones who know their own needs best.
3. **Set a schedule and stick to it.** By putting a schedule in place for when you will get together with your users to evaluate the prototype, you set their expectations and you force yourself to actually get the work done. A win-win situation.
4. **Use a prototyping tool.** Invest the money in a prototyping tool that allows you to put screens together quickly. Because you probably won't want to keep the prototype code that you write, code that's written quickly is rarely worth keeping, you shouldn't be too concerned if your prototyping tool generates a different type of code than what you intend to develop in.
5. **Get the users to work with the prototype.** Just like you want to take a car for a test drive before you buy it your users should be able to take an application for a test drive before it is developed. Furthermore, by working with the prototype hands-on they will quickly be able to determine whether or not the system will meet their needs. A good approach is to ask them to work through some use-case scenarios using the prototype as if it is the real system.
6. **Understand the underlying business.** You need to understand the underlying business before you can develop a prototype that will support it. Perform interviews with key users, read internal documentation of how the business runs, and read documentation about how some of your competitors operate. The more you know about the business the more likely it is that you'll be able to build a prototype that supports it.
7. **There are different levels of prototype.** I like to successively develop three different types of prototypes of a system: A hand-drawn prototype that shows its basic/rough functionality, an electronic prototype that shows the screens but not the data that will be displayed on them, and then finally the screens with data. By starting out simple in the beginning I avoid investing a lot of time in work that will most likely be thrown away. By successively increasing the complexity of the prototype as it gets closer to the final solution, my users get a better and better idea of how the application will actually work, providing the opportunity to provide greater and greater insight into improving it.
8. **Don't spend a lot of time making the code good.** At the beginning of the prototyping process you will throw away a lot of your work as you learn more about the business. Therefore it doesn't make sense to invest a lot of effort in code that you probably aren't going to keep anyway.

Blatant Advertising – Purchase *Building Object Applications That Work* today!

Building Object Applications That Work is an intermediate-level book about object-oriented development. It covers a wide range of topics that few other books dare to consider, including: architecting your applications so that they're maintainable and extensible; OO analysis and design techniques; how to design software for stand-alone, client/server, and distributed environments; how to use both relational and object-oriented (OO) databases to make your objects persistent; OO metrics, analysis and design patterns; OO testing; OO user interface design; and a multitude of coding techniques to make your code robust. Visit

www.ambysoft.com/buildingObjectApplications.html for more details.

3. Interface-Flow Diagrams

To your users the user interface is the system. It is as simple as that. Does it not make sense that you should have some sort of mechanism to help you design a user interface? User interface prototypes are one means of describing your user interface, although with prototypes you can often get bogged down in the details of how the interface will actually work. As a result you often miss high-level relationships and interactions between the interface objects (usually screens) of your application. Interface-flow diagrams (Page-Jones, 1995; Ambler, 1998a; Ambler, 1998b; Ambler, 2001) allow you to model these high-level relationships.

Interface-flow diagrams show the relationships between the user interface components, screens and reports, that make up your application.

In Figure 3 we see an example of an interface-flow diagram for an order-entry system. The boxes represent user interface objects (screens, reports, or forms) and the arrows represent the possible flow between screens. For example, when you are on the main menu screen you can go to either the customer search screen or to the order-entry screen. Once you are on the order-entry screen you can go to the product search screen or to the customer order list. Interface-flow diagrams allow you to easily gain a high-level overview of the interface for your application.

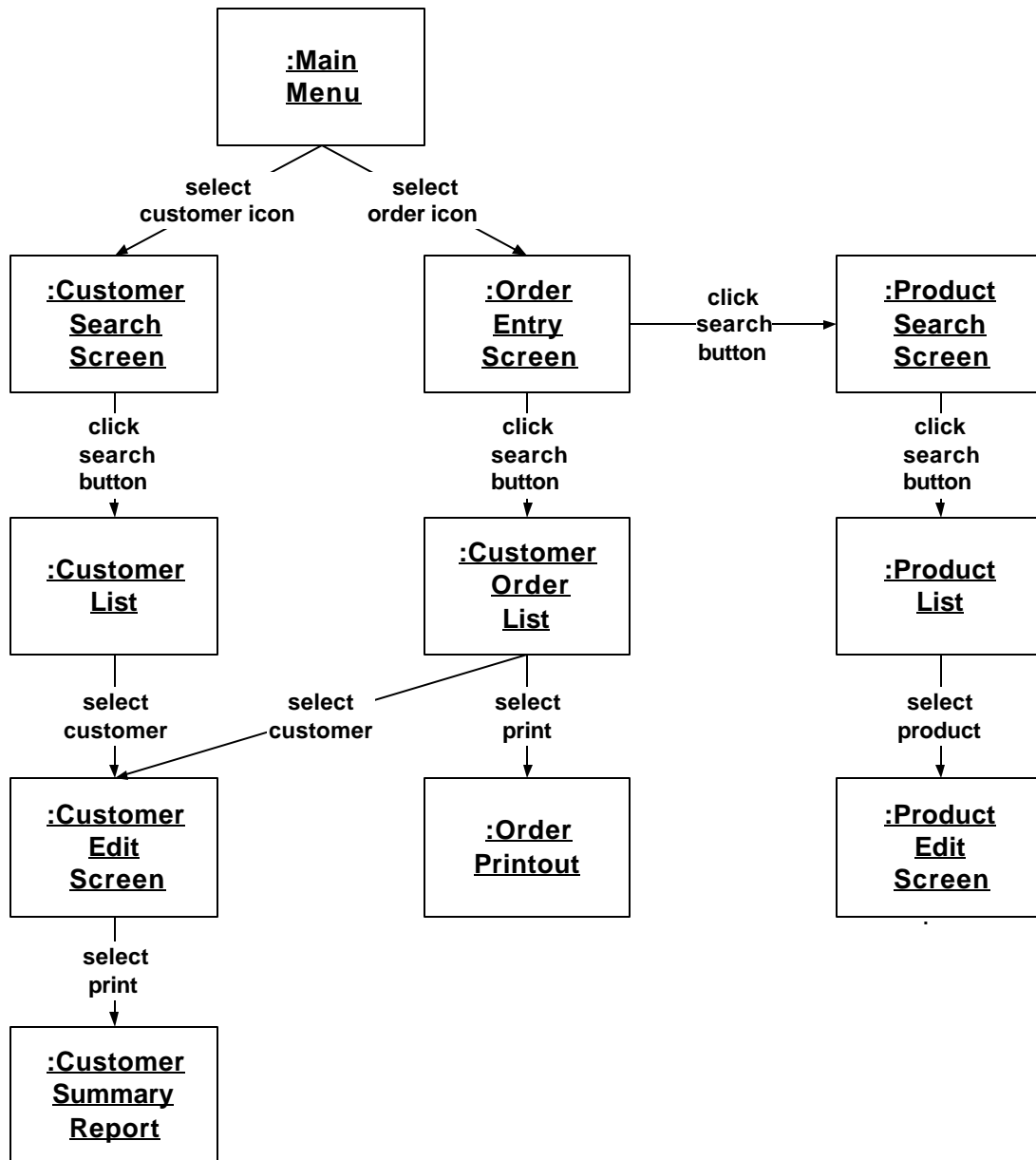


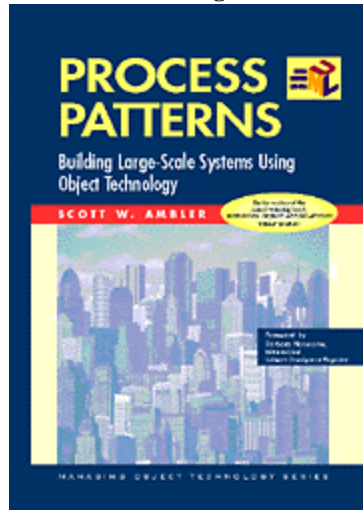
Figure 3. An interface-flow diagram for an order-entry system.

Because interface-flow diagrams offer a high-level view of the interface of a system you can quickly gain an understanding of how the system is expected to work. It puts you into a position where you can validate the overall flow of your application's user interface. For example, does the screen flow make sense? I'm not so sure. Why can I not get from the customer edit screen to the customer order list, which is a list of all the orders that a customer has ever made.

Interface-flow diagrams enable you to validate the design of your user interface.

Furthermore, why cannot I get the same sort of list from the point of view of a product? In some cases it might be interesting to find out which orders include a certain product, especially when the product is back-ordered or no longer available. Also, interface-flow diagrams can be used to determine if the user interface has been design consistently, for example in Figure 3 you see that to create the customer summary report and a printed order that you select the print command. It appears from the diagram that the user interface is consistent, at least with respect to printing.

Blatant Advertising – Purchase *Process Patterns* today!



This book presents a collection of process patterns for successfully initiating a software project and taking it through the construction phase. It provides a wealth of advice for engineering requirements, modeling, programming, and testing. It puts these topics in the context of a proven software process for the development of large-scale, mission-critical software, covering topics that you typically don't find in other books about object-oriented development such as project management, quality assurance, risk management, and deliverables management. Object-oriented development is hard, particularly if you are building systems using n-tier technology such as Enterprise JavaBeans (EJB) or even the "simple" Java platform, and you need to understand the big picture to be successful. *Process Patterns*, and its sister book, *More Process Patterns*, give you that big picture. For more information, and to order online, visit www.ambysoft.com/processPatterns.html

4. Where to Go From Here

I am a firm believer that every developer – particularly analysts, prototypers, and programmers building the user interface – should have a fundamental understanding of human factors engineering (HFE) and the industry-standard user interface guidelines for the platforms to which they are building. For example, Anybody developing to the Win32 platform should own and have read the Microsoft UI guidelines (Microsoft, 1995). Never forget that the user interface is the software to your users, not the database, not the network, not the cool Java code that you're writing. In that light, I would go so far as to say that if you don't understand user interface design then you have no business developing software.

Every developer should understand the fundamentals of user interface design.

So what do you need to do? First, although reading this white paper is a good start it isn't enough, you still need further education. I would start by taking a one or two-day overview course in user interface design, one that covers issues such as human factors engineering, metaphors, mental models, screen design basics, and report design basics. If you are developing object-oriented user interfaces (OOUIs), then I highly suggest Chapter 9 of my second book, *Building Object Applications That Work* (Ambler, 1998a). Second, I would take a two-day course in the user interface standards for the platform that you are developing too. Standards currently exist for many of the common operating systems as well as general UI standards for Internet development. Once again, if you can't find a course then you'll have to do some reading.

5. Summary

In this paper I presented several tips and techniques for designing effective user interfaces. The lists below summarize the key tips for UI design.

5.1 General Guidelines

- Be consistent in a user interface, it's critical.
- Set user interface standards and stick to them.
- Choose industry standards so as to increase the chance that your applications will look and feel like other applications developed externally to your organization.
- Explain the rules of how your application works to your users. If it's consistent, then the rules should be simple and few in number.
- Support both novices and experts.
- Word text consistently, positively, and in full English.
- Look at other applications with a grain of salt because not everyone understands good user interface design.
- Display the objects that your users need to do their jobs on the desktop.
- Think in terms of clusters of business objects and their corresponding interface objects, not in terms of applications.
- Interface objects should look, feel, and behave exactly like the real-world objects that they represent.

5.2 Screen Design

- Navigation between screens and on screens are both important.
- Understand your widgets so that you know how to apply them properly.
- Use color sparingly and always have a secondary indicator.
- Follow the contrast rule – put dark text on light backgrounds and light text on dark backgrounds.
- Use fonts sparingly and consistently.
- When items are unavailable gray them out, don't remove them if you want your users to form accurate mental models.
- Use non-destructive default buttons.
- Left justify edit fields and right justify their labels.
- Right justify integers, decimal-align floating point numbers, and left justify strings.
- Don't create busy/crowded screens.
- Use group boxes and whitespace to group logically related items on the screen.
- Open windows in the center of the action.
- Pop-up menus shouldn't be the only source of functionality.

5.3 Prototyping

- The requirements of your users drive the development of your prototype.
- During evaluation ask: What's good about the prototype, what's bad about the prototype, and what's missing from the prototype.
- Stop the prototyping process when you find the evaluation process is generating few or no new requirements.
- Look for real-world objects and identify how users work with them.
- Work with the people who will use the application when it's done.
- Set a prototyping schedule and stick to it.
- Use a prototyping tool.
- Get the users to work with the prototype, to take it for a test drive.
- Understand the underlying business.
- Don't invest a lot of time in something that you'll probably throw away.
- Document interface objects once they have stabilized.
- Develop an interface-flow diagram for your prototype
- For each interface object that makes up a prototype, document
 - It's purpose and usage
 - An indication of the other interface objects it interacts with
 - The purpose and usage of each of its components

6. References and Recommended Reading

Ambler, S.W. (1997). *The Realities of Mapping Objects To Relational Databases*. San Francisco: Miller-Freeman Publishing, Software Development, October 1997, pp. 71-74.

Ambler, S.W. (1998a). *Building Object Applications That Work: Your Step-By-Step Handbook for Developing Robust Systems with Object Technology*. New York: Cambridge University Press.

Ambler, S.W. (1998b). *Process Patterns: Building Large-Scale Systems Using Object Technology*. New York: Cambridge University Press. <http://www.ambysoft.com/processPatterns.html>

Ambler, S.W. (1999). *More Process Patterns: Delivering Large-Scale Systems Using Object Technology*. New York: Cambridge University Press. <http://www.ambysoft.com/moreProcessPatterns.html>

Ambler, S.W. & Constantine, L.L. (2000a). *The Unified Process Inception Phase*. Gilroy, CA: CMP Books. <http://www.ambysoft.com/inceptionPhase.html>.

Ambler, S.W. & Constantine, L.L. (2000b). *The Unified Process Elaboration Phase*. Gilroy, CA: CMP Books. <http://www.ambysoft.com/elaborationPhase.html>.

Ambler, S.W. & Constantine, L.L. (2000c). *The Unified Process Construction Phase*. Gilroy, CA: CMP Books. <http://www.ambysoft.com/constructionPhase.html>.

Ambler, S.W. (2001). *The Object Primer 2nd Edition: The Application Developer's Guide to Object Orientation*. New York: Cambridge University Press. <http://www.ambysoft.com/theObjectPrimer.html>.

Constantine, L. L. (1995). *Constantine on Peopleware*. Englewood Cliffs, NJ: Yourdon Press.

IBM (1993). *Systems Application Architecture – Common User Access Guide to User Interface Design*. IBM Corporation.

Maguire, S. (1994). *Debugging the Development Process*. Redmond, WA: Microsoft Press.

Mayhew, D.J. (1992). *Principles and Guidelines in Software User Interface Design*. Englewood Cliffs NJ: Prentice Hall.

McConnell, S. (1996). *Rapid Development: Taming Wild Software Schedules*. Redmond, WA: Microsoft Press.

Microsoft (1995). *The Windows Interface Guidelines for Software Design*. Redmond, WA: Microsoft Press.

Page-Jones, M. (1995). *What Every Programmer Should Know About Object-Oriented Design*. New York: Dorset-House Publishing.

7. About the Author

Scott W. Ambler is President of Denver-based Ronin International. Scott is the author of *The Object Primer 2nd Edition* (1995, 2001), *Building Object Applications That Work* (1998), *Process Patterns* (1998) and *More Process Patterns* (1999), and co-author of *The Elements of Java Style* (2000) all published by Cambridge University Press. He is also co-editor with Larry Constantine of the Unified Process book series from CMP books, including *The Unified Process Inception Phase* (Fall 2000), *The Unified Process Elaboration Phase* (Spring 2000), and *The Unified Process Construction Phase* (Summer 2000) all of which focus on best practices to enhance the Unified Process. He has worked with OO technology since 1990 in various roles: Process Mentor, Business Architect, System Analyst, System Designer, Project Manager, Smalltalk Programmer, Java Programmer, and C++ Programmer. He has also been active in education and training as both a formal trainer and as an object mentor. Scott is a contributing editor with *Software Development* (www.sdmagazine.com) and writes columns for *Computing Canada* (www.plesman.com). He can be reached via e-mail at scott.ambler@ronin-intl.com.

Index**A**

Analysis patterns	7
Architecture	7
Author	
contacting	13

B

Book	
Building Object Applications That Work	7
Process Patterns.....	9
The Object Primer.....	5
Business rules	
and prototyping	6

C

Color.....	2
Consistency	
user interface	1
Contrast rule.....	2

D

Databases	7
Default button.....	3
Design patterns	7
Distributed design.....	7

E

Error message.....	2
Extreme Programming.....	5

F

Field alignment.....	3
FLOOT	5
Fonts	3
Full Lifecycle Object-Oriented Testing	5

I

Interface-flow diagram.....	7
example	7

M

Metrics	7
---------------	---

O

Object databases	7
Object-Oriented Software Process.....	5
Object-oriented user interface	
prototyping	4

OOSP.....	5
OOUI.....	9

P

Pop-up menu.....	3
Portability	
and color.....	2
Prototyping	
and interface-flow diagrams	7
levels of.....	6
OOUIs	4
steps of.....	4
tips and techniques	6
tools	6

R

Relational databases	7
----------------------------	---

S

Screen navigation.....	2
Standard	
user interface	1

T

Testing	
user interface flow.....	8
Training courses	9

U

Unified Modeling Language.....	5
Unified Process.....	5
User interface (UI)	
screen navigation.....	2
User interface design	1
field alignment	3
screen navigation.....	2
standards.....	1
the contrast rule	2
tips and techniques	1
use of color	2
use of fonts	3
use of pop-up menus.....	3
Users	
experts	1
novices	1

X

XP	5
----------	---