

The Art of Building Great User Experience in Software



Free Sampler

Effective UI

O'REILLY®

*Jonathan Anderson,
John McRee, Robb Wilson
& the EffectiveUI Team*

O'Reilly Ebooks—Your bookshelf on your devices!



When you buy an ebook through oreilly.com, you get lifetime access to the book, and whenever possible we provide it to you in four, DRM-free file formats—PDF, .epub, Kindle-compatible .mobi, and Android .apk ebook—that you can use on the devices of your choice. Our ebook files are fully searchable and you can cut-and-paste and print them. We also alert you when we've updated the files with corrections and additions.

Learn more at <http://oreilly.com/ebooks/>

You can also purchase O'Reilly ebooks through [iTunes](#),
the [Android Marketplace](#), and [Amazon.com](#).

Effective UI

*Jonathan Anderson, John McRee, Robb Wilson,
and the EffectiveUI Team*

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Taipei • Tokyo

Effective UI

by Jonathan Anderson, John McRee, Robb Wilson, and the EffectiveUI Team

Copyright © 2010 EffectiveUI. All rights reserved.

Printed in Canada.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://my.safaribooksonline.com>). For more information, contact our corporate/institutional sales department: (800) 998-9938 or corporate@oreilly.com.

Editor: Steve Weiss

Development Editor: Jeff Riley

Production Editor: Rachel Monaghan

Copyeditor: Genevieve d'Entremont

Proofreader: Nancy Kotary

Indexer: Julie Hawks

Cover Designer: Karen Montgomery

Illustration and Interior Design:

The EffectiveUI Team

Printing History:

February 2010: First Edition.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Effective UI*, the image of a rainbow lorikeet, and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN: 978-0-596-15478-3

[F]

Contents

Preface	ix	3 Effective Planning and Requirements ..	75
1 Building an Effective UI	1	Uncertainty and the Unknown	77
Understanding UX	4	The Humility of Unknowing	78
What Good UX Accomplishes	6	The Weakness of Foresight and Planning	79
Why Engagement and Good UX Matter	10	Friction in a Complex and Peculiar System	81
The Elements of Engaging UX	11	Subjectivity and Change	87
Redefining Two Fundamental Terms	32	Lessons from Uncertainty and the Unknown	89
Design	32	The Further You Are in the Project, the Wiser You Are	89
Development	34	Start Development As Soon As Possible	90
2 Building the Case for Better UX	37	Written Functional Requirements and Specifications Are Inherently Flawed	90
Why Now Is the Moment for UX	40	Commitments to Scope Are Untenable	92
Motive	40	Relish and Respect the Unexpected	92
Means	48	Intolerance of Uncertainty Is Intolerable	93
Opportunity	50	Effective Requirements	94
Winning Support for Better UX	53	How Framework Requirements Are Built	97
Stakeholders	53	Reexamining the Three-Legged Stool	99
Education	57	Commitments You Can Live Up To	101
Quantifying the Business Value	67	Effective Process	102
Materializing and Proving the Concept	67	Development Methodology	103
Other Strategies for Building Support	73		

4 Bringing Together a Team.....	113		
The Project Leader	116	Who Should Be Involved in the	Research
Relationship to the Product	116		182
Relationship to the Stakeholders	117	Finding Research Participants	184
Relationship to the Project Team	119	Determining the Research Sample Size	185
Who Should Be the Project Leader	119	Making Recordings	188
The Stakeholders	121	Research Through Speaking with Users	190
Securing Authority	121	User Interviews	190
Collaboration and Decision Making	124	Structured Interview Techniques	191
The Characteristics of a Successful		Research Through Direct Observation	193
Project Team	125	Analyzing the Research Observations	196
Getting Professional Help	127	Discovering Personas	196
Insourcing Versus Outsourcing	130	Weaving User Stories	198
		Discovering User Priorities	199
		Guerilla User Research	200
5 Getting the Business Perspective.....	139	Stakeholder Buy-in Through	User Research
Defining Success	141		202
Creating a Project Mission Statement	142	7 Initial Product Architecture.....	205
Determining Project Success Criteria	144	The Initial Product Architecture Team	208
Exercising Restraint	145	Contextual Scenarios	210
Applying the Pareto Principle	148	Mapping High-Level Workflows	213
What Not to Restrain	148	Sketching Low-Fi Visual	Representations of Requirements
Refocusing Product Objectives	149		215
Omissions Aren't Permanent	150	Examining Key Features and	Interactions
Describing the Product's Users	151		216
User Attributes	152	Setting a Style Vision	217
Exercises to Identify Key User	Attributes	Developing Nomenclature	221
	153	Technical Architecture	222
Creating Business Requirements	160	Getting a Lay of the Land	223
Defining "Requirement"	161	Making Platform and Framework	Choices
Exercises to Develop Business	Requirements		223
	163	Understanding Data Requirements	224
Maintaining Stakeholder Buy-in	169	Mapping Interactions with	Other Systems
			225
6 Getting to Know the User.....	171	Finding Shortcuts Through Third-Party	and Open Source Components
Valuing User Research	173		228
Combating Pressure to Skip	User Research	Discovering Business Logic	229
	175	Software Architecture in Big Design	Up Front (BDUF)
Key Concepts in User Research	177		230
Empathy	177	Project Infrastructure Needs	232
User Goals Versus Product Features and	Tasks	Code Source Control	232
	178	Graphic Asset Management	233
Qualitative Versus Quantitative	Research Methods	Testing Infrastructure and	Environments
	180		234

8 The Iterative Development Process . . .	235	9 Release and Post-Release	263
Regarding “Process”	239	Managing Expectations	265
Iterations and Feedback	239	The Alpha and Beta Releases	266
The Scope of Iterations	243	Receiving Orderly Feedback	268
Prioritizing the Subjects of Iterations	245	Last-Minute Housekeeping	269
Finishing Iterations with Something Complete	246	User Documentation	270
Estimating Iterations	247	And Champagne Corks Fly...	271
Basic Iterative Process	248	Adoption	272
Mapping Progress and Feedback Across Multiple Cycles	252	Post-Release	273
Increasing the Amount of Feedback	254	Review	274
Iteration in Sub-Ideal Project Approaches	256	Measurement and Tracking	277
Strict Waterfall Process	257	Afterword	281
Iteration in a Big Design Up Front (BDUF) Process	261	Index	287

Regarding “Process”

There’s a risk that all the talk of process and flowchart-like diagrams coming up in this chapter might make you fret too much about whether you’re properly adhering to prescribed procedures. Successful approaches to building software are not about rigid processes, flow charts, or strict methodologies. No project is alike, and so there’s no single approach that will work for all possible projects. There are, however, a small number of principles that we’ll describe in this chapter that serve as veritable beacons by which you can navigate through the rocky, dark, unexplored sea of your project. By steering the project as closely as you can toward these principles, even if you’re not able to fully achieve them, you will make the project less rocky and the results more successful.

Iterations and Feedback

Much of what goes on during development is akin to the process a scientist goes through to make a discovery. The scientist develops a hypothesis, and then undertakes a series of experiments to test and explore it. At the end of each experiment, the scientist looks at its results, and based on them decides how to modify the hypothesis or the course of experimentation. This process involves a lot of trial and error. Each experiment represents a single iteration that helps the scientist incrementally develop a more accurate understanding of the truth he’s investigating.

An artist creating a painting follows a similar process. He applies a few lines or a few strokes of the brush and steps back to look at the result. Responding to what he sees, he then adds or modifies lines and brush strokes to bring it closer to the goal and repeats this iteration until the painting is complete. Neither the scientist nor the artist expects to get the theory, painting, or any component of either right from the very first attempt. Neither has a perfect image or design of what the solution to their problem should be. Both are following processes of planning, experimentation, study, and trial and error that require many iterations, each representing some degree of failure, to home in on the right result. Figure 8-1 shows a very simple iterative process.

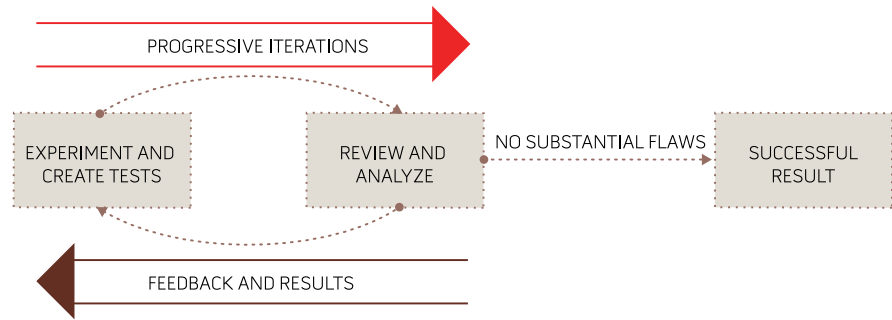


Figure 8-1. A simple iterative process

This should be a familiar process that is naturally at work in any software development project. Designers iterate on their own work to produce final designs, software engineers iterate on components to build robust results, and stakeholders propose, discuss, analyze, and revise concepts and constraints. At this point in the history of software development, it would be a surprise to discover a team that didn't use an iterative process, at least within the confines of each of the professional disciplines.

The crucial step in the scientific process is the moment the scientist takes to analyze the results of the previous experiment and make thoughtful, directed adjustments to the hypothesis or next experiment. For the artist, that crucial step is when he steps back to the canvas to take in what he's done and decides what needs to be done next. This element of building feedback—of surveying and learning from the current state—into the design process is what allows positive progress to be made. Without feedback, the scientist would be lost in an endless series of random, aimless experimentation, and the painter would toil endlessly on a painting that's always changing but never improving. Feedback in iteration is the heart of what makes an iterative process useful; it's what makes it productive and purposeful. This concept applies just as much in software development as in art or science. The goal is to make purposeful, intentional steps in an ever-improving direction.

Feedback in iteration is the heart of what makes an iterative process useful or even purposeful.

At the end of each iteration in software development, the project team should have a better understanding of the overall problem and of the solution they're attempting to craft. Each iteration represents an investigation, the findings of which are analyzed and used to determine the course of the next iteration. Each iteration exposes unknowns and advances the team's understanding of the problem and the solution. More iterations mean more opportunities for the team members to refine their knowledge. As their understanding develops and gains accuracy, they become better at designing solutions and directing the course of development.

Frequent iterations mean that those moments of feedback—of stepping back to assess what's been done and determine what needs to happen next—happen more frequently. And the shorter the iterations are, the less time it takes to arrive at a point of receiving feedback and making a course adjustment. As well, shorter iterations mean that less is invested in each round of trial and error. Frequent, small errors are much easier to learn and progress from than a few enormous, infrequent ones. The longer an iteration goes without feedback, the wider and deeper the opportunity for the effort to go significantly and irreversibly off-course becomes.

We briefly examined this concept in Chapter 3 in the section entitled “Efficiency and the unknown.” A waterfall process is weak in its siloed and limited opportunities for feedback. This tends to mean a lot of work is done and most of the budget expended before anyone realizes how far off course they've gone. This frequently results in total catastrophe and requires expensive, major course corrections. The cost of this approach is further compounded if the team fails to switch from a waterfall process to something more iterative after the first delivery, such as an agile process. Though an iterative project might head off in the same wrong direction as a waterfall process, the discovery of the error and the resulting course correction occurs much earlier and benefits from much more valuable feedback. Figure 8-2 demonstrates this clearly.

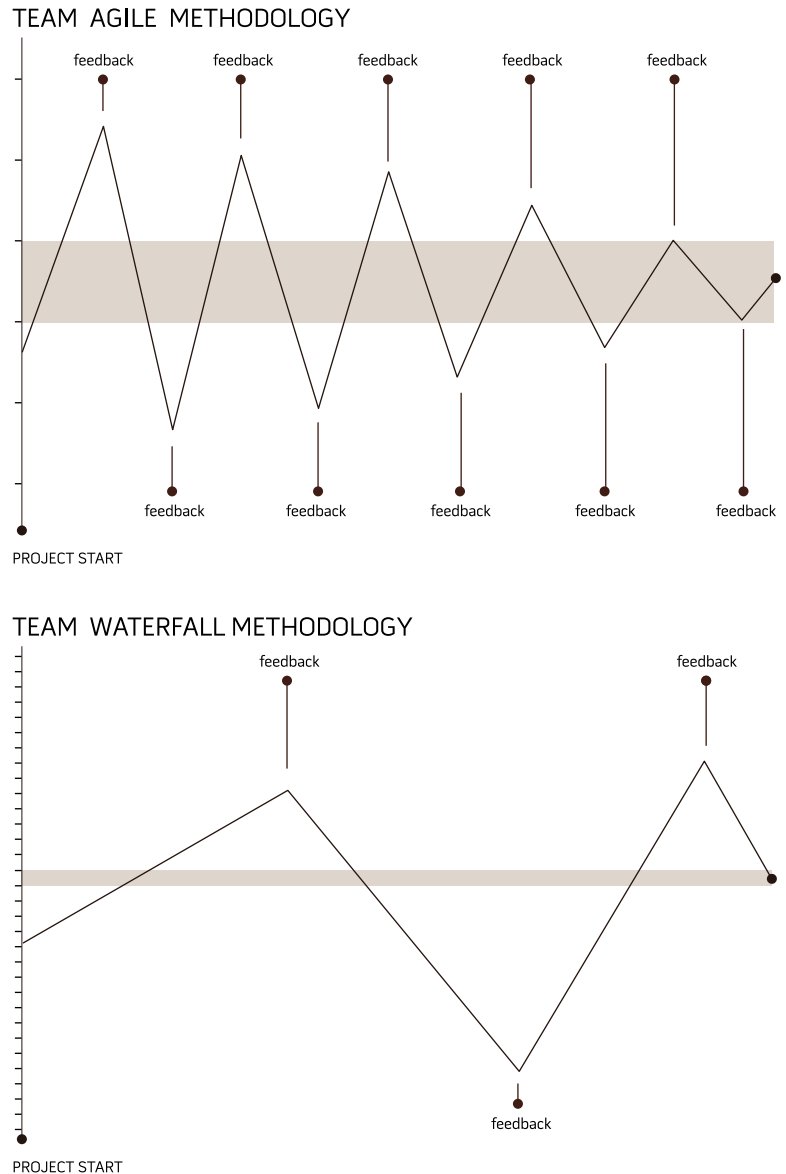


Figure 8-2. Error and course deviation in waterfall and iterative processes

The frequency and timeliness of feedback is what distinguishes healthy projects from unhealthy ones.

Generally speaking, the more feedback that is built into a project, the more depth of understanding will go into it and the more knowledgeable the project team will grow through the course of the project. It's also important that the feedback comes from all facets of the project and from all the professional disciplines involved in building it; more diverse perspectives provide more comprehensive, robust feedback.

And so we've introduced the two most important principles behind an iterative approach to development for UX-focused projects:

- *Improve the efficiency of progress by building in more opportunities for feedback.*
- *Improve the quality of the feedback by ensuring that it reflects diverse perspectives.*

The means by which you apply these principles depends on the particularities of your project. But in choosing and then applying an approach to developing your product, you can judge its likelihood to succeed based on its propensity to generate more frequent, higher-quality feedback.

The Scope of Iterations

In order to build more frequent feedback into a project, iterations must be small. As Figure 8-2 showed, the waterfall approach goes through only four major iterations (each line segment). This means there are only three opportunities for feedback. By contrast, the iterative process goes through about 40–50 smaller iterations (in the simplified diagram, anyway; a real iterative process will go through far more than 50 iterations), which means there would be 39–49 opportunities for feedback. So smaller iterations lead to greater amounts of feedback, which in turn lead to a smoother ride and better results.

This begs the question of how you decide the scope of an iteration—that is, how much progress should you expect of an iteration? Iterations need to be small enough to allow for a high frequency of feedback, but not so small as to be impracticable. The various submethodologies of Agile methodology each has its own answer to this, some more complicated than others. For example, Scrum frames iterations in terms of time, requiring “sprints” of about two to four weeks. Feedback occurs through a regiment of short daily meetings, forward-looking and retrospective post-sprint meetings, periodic planning meetings, and through team structures.

Our inclination is to scope iterations based on functionality rather than time. An iteration should ideally be concerned with the smallest meaningful unit of functionality. By “meaningful,” we mean a unit that, though it might be rather insignificant within the whole scope of the project, is whole unto itself, meaning that it can stand alone from a UX and software engineering perspective. To discover these meaningful smallest units, and also to

determine in what order they must be developed, the hierarchy of application components must be traced down to its lowest meaningful level.

One way to approach this problem in the beginning is to ask the question, “What component or feature is the heart or essence of the product?” Think, for example, of Twitter. What single feature or component truly defines Twitter? It’s not the capability for users to choose a personal style for their Twitter pages, nor is it the ability to add metatags to “tweets” (for example, #iranelection, @someuser). It isn’t even the ability to follow Twitter feeds of friends. The most basic and core feature of Twitter is the ability to post 140-character messages to a web page. Without this, you don’t have a product remotely like Twitter. This capability would therefore be the first focus of the project. It can be further broken down into two parts: a means for posting new tweets into the Twitter service, and the means for displaying those tweets on a web page. A system isn’t much good without any data, so you’d likely start with the posting capability.

This is likely the smallest meaningful unit of functionality within the Twitter example. This capability requires two components:

- *A web page with a 140-character text input box and submit button.*
- *The capability of storing the submitted tweet in a database.*

But each component cannot stand on its own; each requires the other. A focus on just the web UI or just the backend mechanisms would make the focus meaningless from the perspective of either the software engineers or the UX team, respectively. But taken together, they comprise a function that can stand on its own and is meaningful to the user, to the UX team, and to the engineers. That gives stakeholders and users something meaningful to look at and respond to.

As the project progresses, the focus of subsequent iterations will shift from the smallest meaningful component to whatever is the next most important, smallest meaningful increment of progress that can be made. Sometimes this means that the development of a new component will be undertaken, but more often this leads to a meaningful refinement of an existing component or the bringing together of smaller units of functionality to build a larger aspect of the product.

Want to read more?

You can find this book at oreilly.com
in print or ebook format.

It's also available at your favorite book retailer,
including [iTunes](#), [the Android Market](#), [Amazon](#),
and [Barnes & Noble](#).



O'REILLY[®]

Spreading the knowledge of innovators

oreilly.com