



Community Experience Distilled

Mastering JavaScript Design Patterns

Discover how to use JavaScript design patterns to create powerful applications with reliable and maintainable code

Simon Timms

[PACKT] open source*
PUBLISHING community experience distilled

Mastering JavaScript Design Patterns

Discover how to use JavaScript design patterns to create powerful applications with reliable and maintainable code

Simon Timms



BIRMINGHAM - MUMBAI

Mastering JavaScript Design Patterns

Copyright © 2014 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: November 2014

Production reference: 1151114

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78398-798-6

www.packtpub.com

Credits

Author

Simon Timms

Project Coordinator

Rashi Khivansara

Reviewers

Amrita Chaturvedi

Philippe Renevier Gonin

Pedro Miguel Barros Morgado

Mani Nilchiani

Proofreaders

Simran Bhogal

Lawrence A. Herman

Elinor Perry-Smith

Commissioning Editor

Kunal Parikh

Indexer

Hemangini Bari

Acquisition Editor

Meeta Rajani

Graphics

Sheetal Aute

Ronak Dhruv

Valentina D'silva

Disha Haria

Abhinash Sahu

Content Development Editor

Sweny M. Sukumaran

Technical Editor

Siddhi Rane

Production Coordinator

Nitesh Thakur

Copy Editor

Laxmi Subramanian

Cover Work

Nitesh Thakur

About the Author

Simon Timms is a developer who loves to write code. He writes in a variety of languages using a number of tools. For the most part, he develops web applications with .NET backends. He is very interested in visualizations and cloud computing. A background in build and system administration keeps him on the straight and narrow when it comes to DevOps.

He is the author of *Social Data Visualization with HTML5 and JavaScript*, Packt Publishing. He blogs at <http://blog.simontimms.com/> and is also a frequent contributor to the Canadian Developer Connection, where his latest series explores evolving ASP.NET applications.

He is the President of the Calgary .NET User Group and a member of half a dozen others. He speaks on a variety of topics from DevOps to how the telephone system works.

He works as a web developer for Pacesetter Directional Drilling, the friendliest performance drilling company around.

I would like to thank my wonderful wife for all her support and my children who provided a welcome distraction from writing. I would also like to thank the Prime team at Pacesetter for their sense of humor and for putting up with me.

About the Reviewers

Amrita Chaturvedi is a PhD researcher in the Department of Computer Science and Engineering at Indian Institute of Technology, Kanpur, Uttar Pradesh, India (<http://www.cse.iitk.ac.in/users/amrita/index.htm>). She was schooled (kindergarten to intermediate) at City Montessori School, Aliganj, Lucknow, Uttar Pradesh, India. She received a Bachelor of Technology degree in Information Technology from Institute of Engineering and Technology, Lucknow, Uttar Pradesh, India and a Master of Technology degree in Information Technology (with a specialization in Software Engineering) from Indian Institute of Information Technology, Allahabad (Deemed University), Uttar Pradesh, India. She has worked in Nucleus Software Exports Ltd., Noida, Uttar Pradesh, India as a software engineer. She was also employed as a faculty in Institute of Engineering and Technology, Lucknow, Uttar Pradesh, India. She has worked in user interface design as a senior project associate at Indian Institute of Technology, Kanpur, Uttar Pradesh, India. She was selected as the first female PhD student from Asia under EURECA (European Research and Educational Collaboration with Asia) project 2009 to conduct research at VU University Amsterdam, the Netherlands. Her areas of specialization are software engineering, software architecture, software design patterns, and ontologies. Her research interests include software architecture and design, ontologies-based software engineering, service-oriented and model-driven architecture, semantic web, Internet technologies, and mobile agents.

She has given several talks and seminars as well as conference welcome/key notes at international conferences. She has also earned various awards such as best paper award for her research paper in an international conference, ACM SIGAPP award, and has also been a Physics Olympiad topper. She has traveled several European, North American, African, and Asian countries for educational/conference/research purposes. She has teaching as well as research experience and has worked on several implementation-based projects both jointly in a team as well as independently. She has acted as a session chair and program committee member, as well as research paper reviewer for various international conferences.

I would like to thank my incredible and beloved husband, Bhartendu Chaturvedi, for his constant support.

Philippe Renevier Gonin has been an assistant professor at the University of Nice Sophia-Antipolis (UNS), France, since 2005. He teaches web technologies, software engineering (architecture, development), and HCI (Human Computer Interaction).

In the research area, he works on connections between user-centered design (for example, user and task models) and software engineering (for example, component architecture and UI development).

Pedro Miguel Barros Morgado holds a Master's degree in Informatics and Computing Engineering at FEUP (Faculdade de Engenharia da Universidade do Porto) and did his master thesis on Object-Oriented Patterns and Service-Oriented Patterns.

Since 2009, he has been working with several different programming languages, frameworks, and technologies, which included the main OO programming languages such as PHP, Python, C/C++, Java, and JavaScript as well as web languages such as HTML, JSON, and XML. He has worked with different database technologies such as MySQL, PostgreSQL, Oracle SQL, and SQL Server and also with different caching systems and search engines.

He has worked as an IT consultant in the banking field for a year, and built a recommendation system (data mining and text mining) as a research assistant at INESC (Technology and Science-Associated Laboratory) for a period of 1 year. Finally, he focused on web projects as a technical lead at Rocket Internet AG, for which he built scalable systems for FoodPanda, CupoNation, Camudi, and Lamudi. Due to his experience, he has specialized in project management and product development based on an e-commerce area. For more information, take a look at his LinkedIn account at <https://www.linkedin.com/in/pedrombmorgado>.

Mani Nilchiani is a developer, an artist, and a teacher based in Brooklyn, New York. He holds an MFA degree in Design and Technology from Parsons The New School for Design. He is a frontend engineer at The Daily Beast where he focuses on UI Development, API design, integration, and architecture, and works as an adjunct faculty at Parsons The New School for Design, where he teaches a graduate-level curriculum of JavaScript development and design patterns. As a digital artist, he approaches code as an expressive medium to create interactive, site-specific, and generative works of art.

www.PacktPub.com

Support files, eBooks, discount offers, and more

For support files and downloads related to your book, please visit www.PacktPub.com.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Free access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view 9 entirely free books. Simply use your login credentials for immediate access.

To my wife, Melissa, and children, Oliver and Sebastian, who have been with me every step of the way. Without their support, I would be but half a person.

Table of Contents

Preface	1
Chapter 1: Designing for Fun and Profit	7
The road to JavaScript	7
The early days	8
A pause	10
The way of Gmail	10
JavaScript everywhere	13
What is a design pattern?	16
Antipatterns	18
Summary	20
Part 1: Classical Design Patterns	
Chapter 2: Organizing Code	23
Chunks of code	23
What's the matter with global scope anyway?	25
Objects in JavaScript	27
Build me a prototype	31
Inheritance	34
Modules	36
ECMAScript 6 classes and modules	40
Best practices and troubleshooting	41
Summary	41
Chapter 3: Creational Patterns	43
Abstract Factory	44
Implementation	49
Builder	51
Implementation	52

Factory Method	55
Implementation	55
Singleton	58
Implementation	59
Disadvantages	60
Prototype	60
Implementation	61
Hints and tips	62
Summary	63
Chapter 4: Structural Patterns	65
Adapter	65
Implementation	67
Bridge	69
Implementation	71
Composite	74
An example	75
Implementation	76
Decorator	78
Implementation	79
Façade	80
Implementation	81
Flyweight	83
Implementation	83
Proxy	85
Implementation	86
Hints and tips	87
Summary	87
Chapter 5: Behavioral Patterns	89
Chain of responsibility	90
Implementation	91
Command	94
The command message	95
The invoker	97
The receiver	98
Interpreter	99
An example	99
Implementation	100
Iterator	101
Implementation	101
ECMAScript 6 iterators	103

Mediator	103
Implementation	104
Memento	105
Implementation	106
Observer	109
Implementation	110
State	112
Implementation	113
Strategy	116
Implementation	117
Template method	119
Implementation	121
Visitor	123
Hints and tips	128
Summary	128
Part 2: Other Patterns	
Chapter 6: Functional Programming	131
Functional functions are side-effect free	132
Function passing	132
Implementation	134
Filters and pipes	136
Implementation	137
Accumulators	139
Implementation	140
Memoization	141
Implementation	142
Immutability	144
Lazy instantiation	145
Implementation	145
Hints and tips	147
Summary	148
Chapter 7: Model View Patterns	149
First, some history	150
Model View Controller	150
The MVC code	155
Model View Presenter	160
The MVP code	161

Model View ViewModel	164
The MVVM code	165
A better way to transfer changes between the model and the view	167
Observing view changes	169
Hints and tips	170
Summary	170
Chapter 8: Web Patterns	171
Sending JavaScript	171
Combining files	172
Minification	175
Content delivery networks	176
Plugins	177
jQuery	177
d3	179
Doing two things at once – multithreading	182
The circuit breaker pattern	185
Back-off	186
Degraded application behavior	187
The promise pattern	188
Hints and tips	190
Summary	190
Chapter 9: Messaging Patterns	191
What's a message anyway?	192
Commands	193
Events	194
Request-reply	196
Publish-subscribe	199
Fan out and fan in	202
Dead-letter queues	205
Message replay	207
Pipes and filters	208
Versioning messages	209
Hints and tips	210
Summary	211
Chapter 10: Patterns for Testing	213
The testing pyramid	214
Test in the small with unit tests	214
Arrange-Act-Assert	216
Asserts	217

Fake objects	218
Test spies	219
Stub	220
Mock	222
Monkey patching	223
Interacting with the user interface	224
Browser testing	224
Faking the DOM	225
Wrapping the manipulation	226
Build and test tools	227
Hints and tips	227
Summary	228
Chapter 11: Advanced Patterns	229
Dependency injection	229
Live postprocessing	233
Aspect-oriented programming	234
Macros	238
Hints and tips	239
Summary	240
Chapter 12: ES6 Solutions Today	241
TypeScript	241
The class syntax	242
The module syntax	243
Arrow functions	244
Typing	246
Traceur	248
Classes	249
Default parameters	250
Template literals	251
Block bindings with let	252
Async	254
Conclusion	255
Hints and tips	255
Summary	256
Appendix: Conclusion	257
Index	261

Preface

JavaScript is starting to become one of the most popular languages in the world. However, its history as a bit of a toy language, means that developers are tempted to ignore good design. Design patterns are a great tool to suggest some well-tried solutions.

What this book covers

This book is divided into two main parts, each of which contains a number of chapters. The first part of the book, which I'm calling *Part 1*, covers the classical design patterns, which are found in the GoF book. *Part 2* looks at patterns, which are either not covered in the GoF book or the ones that are more specific to JavaScript.

Chapter 1, Designing for Fun and Profit, provides an introduction to what design patterns are and why we are interested in using design patterns. We will also talk about the history of JavaScript to give a historical context.

Chapter 2, Organizing Code, explains how to create the classical structures used to organize code: namespaces or modules and classes as JavaScript lack these constructs as first class citizens.

Chapter 3, Creational Patterns, covers the creational patterns outlined in the Gang of Four book. We'll discuss how these patterns apply to JavaScript, as opposed to the languages which were popular at the time when the Gang of Four wrote their book.

Chapter 4, Structural Patterns, examines the structural patterns from the Gang of Four book following on our look at creational patterns.

Chapter 5, Behavioral Patterns, covers the final set of patterns from the Gang of Four book that we'll examine. These patterns govern different ways to link classes together.

Chapter 6, Functional Programming, explains some of the patterns which can be found in functional programming languages. We'll look at how these patterns can be used in JavaScript to improve code.

Chapter 7, Model View Patterns, examines the confusing variety of different patterns to create single-page applications. We'll provide clarity and look at how to use libraries which use each of the existing patterns, as well as create their own lightweight framework.

Chapter 8, Web Patterns, covers a number of patterns which have specific applicability to web applications. We'll also look at some patterns around deploying code to remote runtimes such as the browser.

Chapter 9, Messaging Patterns, explains messaging which is a powerful technique to communicate inside, and even between, applications. We'll also look at some common structures around messaging and discuss why messaging is so useful.

Chapter 10, Patterns for Testing, focuses on some patterns which make for easier testing, giving you more confidence that your application is working as it should.

Chapter 11, Advanced Patterns, includes some patterns, such as aspect-oriented programming, that are rarely applied in JavaScript. We'll look at how these patterns can be applied in JavaScript and discuss if we should apply them.

Chapter 12, ES6 Solutions Today, discusses some of the tools available to allow you to use features from future versions of JavaScript today. We'll examine Microsoft's TypeScript as well as Traceur.

Appendix, Conclusion, covers what you have learned, in general, in the book, and you will be reminded of the goal of using patterns.

What you need for this book

There is no specialized software needed for this book. JavaScript runs on all modern browsers. There are standalone JavaScript engines written in C++ (V8) and Java (Rhino), and these are used to power all sorts of tools such as Node.js, CouchDB, and even Elasticsearch. These patterns can be applied to any of these technologies.

Who this book is for

The intended audience is developers who have some experience with JavaScript, but not necessarily with entire applications written in JavaScript. Also, developers who are interested in creating easily maintainable applications that can grow and change with need.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "The next item of interest is that we need to make use of the `this` qualifier to address the greeting variable from within the `doThings` function."

A block of code is set as follows:

```
var Wall = (function () {  
    function Wall() {  
        console.log("Wall constructed");  
    }  
    return Wall;  
})();  
Structures.Wall = Wall;
```


When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:


```
var Wall = (function () {  
    function Wall() {  
        console.log("Wall constructed");  
    }  
    return Wall;  
})();  
Structures.Wall = Wall;
```

Any command-line input or output is written as follows:

```
npm install -g traceur
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "clicking the **Next** button moves you to the next screen".

[ Warnings or important notes appear in a box like this.]

[ Tips and tricks appear like this.]

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you really get the most out of.

To send us general feedback, simply e-mail feedback@packtpub.com, and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files from your account at <http://www.packtpub.com> for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

You can also download the example code files for this book from GitHub at <https://github.com/stimms/JavaScriptPatterns>.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata, under the Errata section of that title.

To view the previously submitted errata, go to <https://www.packtpub.com/books/content/support> and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at questions@packtpub.com, and we will do our best to address the problem.

1

Designing for Fun and Profit

JavaScript is an evolving language that has come a long way from its inception. Possibly more than any other programming language, it has grown and changed with the growth of the World Wide Web. The exploration of how JavaScript can be written using good design principles is the topic of this book. The preface of this book contains a detailed explanation of the sections of the book.

In the first half of this chapter, we'll explore the history of JavaScript and how it came to be the important language that it is today. As JavaScript has evolved and grown in importance, the need to apply rigorous methods to its construction has also grown. Design patterns can be a very useful tool to assist in developing maintainable code. The second half of the chapter will be dedicated to the theory of design patterns. Finally, we'll look briefly at antipatterns.

The topics covered in this chapter are:

- The history of JavaScript
- What is a design pattern?
- Antipatterns

The road to JavaScript

We'll never know how language first came into being. Did it slowly evolve from a series of grunts and guttural sounds made during grooming rituals? Perhaps it developed to allow mothers and their offsprings to communicate. Both of these are theories, all but impossible to prove. Nobody was around to observe our ancestors during that important period. In fact, the general lack of empirical evidence lead the Linguistic Society of Paris to ban further discussions on the topic, seeing it as unsuitable for serious study.

The early days

Fortunately, programming languages have developed in recent history and we've been able to watch them grow and change. JavaScript has one of the more interesting histories in modern programming languages. During what must have been an absolutely frantic 10 days in May of 1995, a programmer at Netscape wrote the foundation for what would grow up to be modern JavaScript.

At that time, Netscape was involved in the first of the browser wars with Microsoft. The vision for Netscape was far grander than simply developing a browser. They wanted to create an entire distributed operating system making use of Sun Microsystems' recently released Java programming language. Java was a much more modern alternative to C++ that Microsoft was pushing. However, Netscape didn't have an answer to Visual Basic. Visual Basic was an easier to use programming language, which was targeted at developers with less experience. It avoided some of the difficulties around memory management which makes C and C++ notoriously difficult to program. Visual Basic also avoided strict typing and overall allowed more leeway.

Brendan Eich was tasked with developing Netscape repartee to VB. The project was initially codenamed Mocha but was renamed LiveScript before Netscape 2.0 beta was released. By the time the full release was available, Mocha/LiveScript had been renamed JavaScript to tie it into the Java applet integration. Java applets were small applications that ran on the browser. They had a different security model from the browser itself and so were limited in how they could interact with both the browser and the local system. It is quite rare to see applets these days, as much of their functionality has become part of the browser. Java was riding a popular wave at that time and any relationship to it was played up.

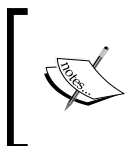
The name has caused much confusion over the years. JavaScript is a very different language from Java. JavaScript is an interpreted language with loose typing that runs primarily on the browser. Java is a language that is compiled to bytecode, which is then executed on the Java Virtual Machine. It has applicability in numerous scenarios from the browser (through the use of Java applets) to the server (Tomcat, JBoss, and so on) to full desktop applications (Eclipse, OpenOffice). In most laypeople's minds, the confusion remains.

JavaScript turned out to be really quite useful for interacting with the web browser. It was not long until Microsoft had also adopted JavaScript in their Internet Explorer to complement VBScript. The Microsoft implementation was known as JScript.

By late 1996, it was clear that JavaScript was going to be the winning web language for the near future. In order to limit the amount of language deviation between implementations, Sun and Netscape began working with the **European Computer Manufacturers Association (ECMA)** to develop a standard to which future versions of JavaScript would need to comply. The standard was released very quickly (very quickly in terms of how rapidly standard organizations move) in July of 1997. On the off chance that you have not seen enough names yet for JavaScript, the standard version was called ECMAScript, a name which still persists in some circles.

Unfortunately, the standard only specified the very core parts of JavaScript. With the browser wars raging, it was apparent that any vendor that stuck with only the basic implementation of JavaScript would quickly be left behind. At the same time, there was much work going on to establish a standard **document object model (DOM)** for browsers. The DOM was, in effect, an API for a web page that could be manipulated using JavaScript.

For many years, every JavaScript script would start by attempting to determine the browser on which it was running. This would dictate how to address elements in the DOM, as there were dramatic deviations between each browser. The spaghetti of code that was required to perform simple actions was legendary. I remember reading a year-long 20 part series on developing **Dynamic HTML (DHTML)** drop-down menus such that they would work on both Internet Explorer and Netscape Navigator. The same functionality can now be achieved with pure CSS without even having to resort to JavaScript.



DHTML was a popular term in the late 1990s and early 2000s. It really referred to any web page that had some sort of dynamic content that was executed on the client side. It has fallen out of use as the popularity of JavaScript has made almost every page a dynamic one.

Fortunately, the efforts to standardize JavaScript continued behind the scenes. Versions 2 and 3 of ECMAScript were released in 1998 and 1999. It looked like there might finally be some agreement between the various parties interested in JavaScript. Work began in early 2000 on ECMAScript 4, which was to be a major new release.

A pause

Then, disaster struck! The various groups involved in the ECMAScript effort had major disagreements about the direction JavaScript was to take. Microsoft seemed to have lost interest in the standardization effort. It was somewhat understandable as it was around that time that Netscape self-destructed and Internet Explorer became the de facto standard. Microsoft implemented parts of ECMAScript 4 but not all of it. Others implemented more fully featured support but, without the market leader on board, developers didn't bother using them.

Years passed without consensus and without a new release of ECMAScript. However, as frequently happens, the evolution of the Internet could not be stopped by a lack of agreement between major players. Libraries such as jQuery, Prototype, Dojo, and MooTools papered over the major differences in browsers, making cross-browser development far easier. At the same time, the amount of JavaScript used in applications increased dramatically.

The way of Gmail

The turning point was, perhaps, the release of Google's Gmail application in 2004. Although XMLHttpRequest, the technology behind **Asynchronous JavaScript and XML (AJAX)**, had been around for about 5 years when Gmail was released, it had not been well used. When Gmail was released, I was totally knocked off my feet by how smooth it was. We've grown used to applications that avoid full reloads, but at that time it was a revolution. To make applications like that work, a great deal of JavaScript is needed.



AJAX is a method by which small chunks of data are retrieved from the server by a client instead of refreshing the entire page. The technology allows for more interactive pages that avoid the jolt of full page reloads.