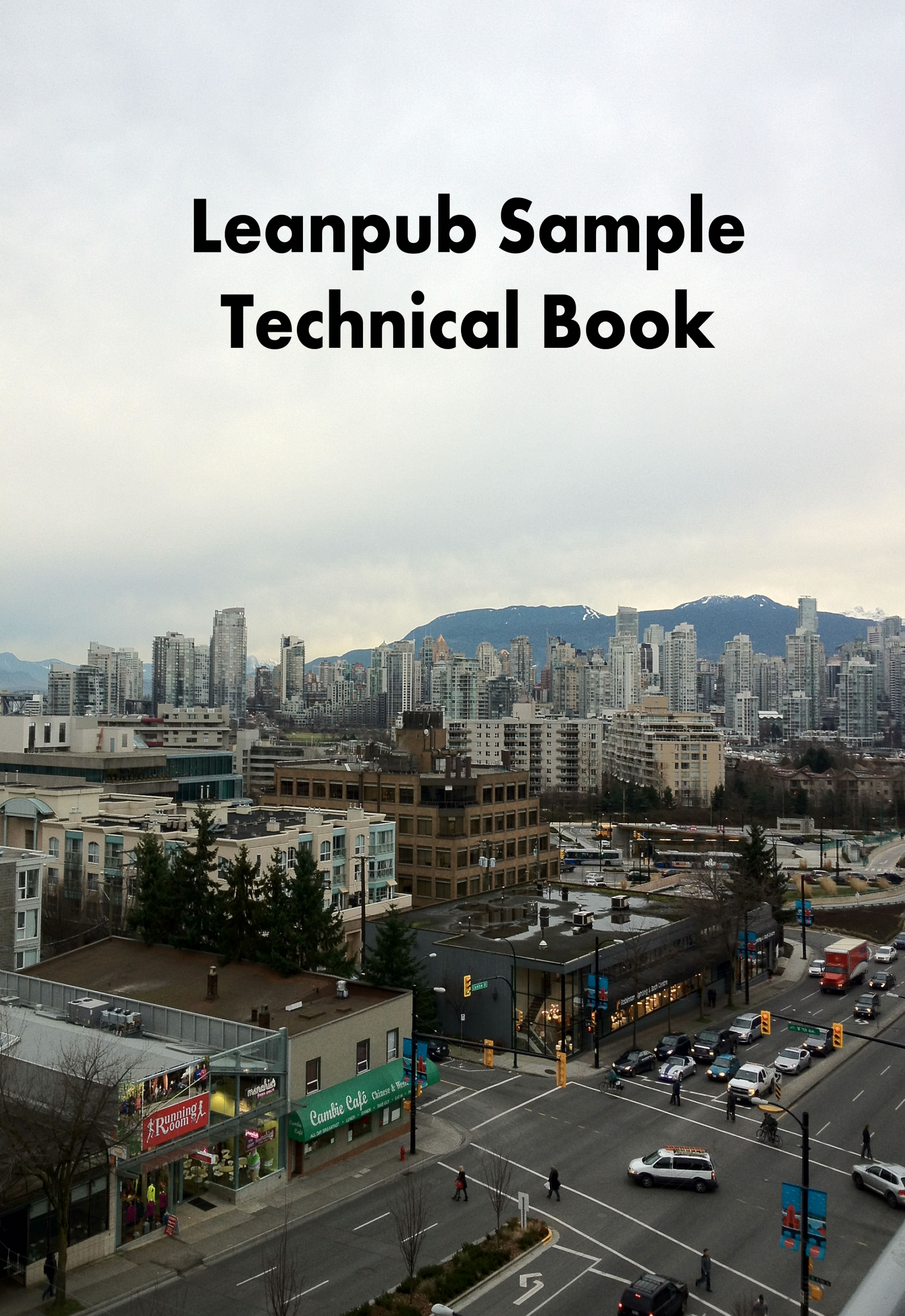
Leanpub Sample Technical Book



API Solutions

Last published on 2012-03-29

Contents

Acknowledgments

We'd like to thank all Leanpub authors for making the product better!

Preface

In this sample Leanpub book we will show you how to write a Leanpub book.

Let's start from small example: Update a function (in our product) that returns list of word occurrences in the given list of words be thread safe.

Easy, you fire up search engine or your favorite book. Look up for relevant APIs; not that big deal. All in all 5 minutes work, right? I lay (past of lie), let's examine it from real world perspective: This piece of code lives in some application and uses frameworks. For this it has to init and to call some APIs such as file reading or sockets (need to learn the APIs, some APIs are proprietary so no Google help here) who said that they are thread safe. Cool can use sync utilities (hmm have to check that some threads might be sleeping ...), and let's make it even harder, our lovely function performs that well so more and more data is sent to it (aha yes the system is growing from 1,000 words to 10,000 words) and latency and deadlocks begin… and yes it should be cross platform. Now it looks like a real life!

Something is being missed here. From something clear and visible such as feature request to the piece of code (and it is perfectly fine to get non trivial code). There is a link missing.

There are some hidden artifacts here in between that are not clearly visible but do have a clear impact. These artifacts include platform and framework complexity and APIs, threading issues, platform and the list goes on. That we don't see theses artifacts doesn't mean they don't exist, they exist and kicking! The hard part is the understanding is that if you (or your manager) don't see those skills and activities it doesn't mean they have no impact, indeed they have and a big one!

So seems we are lacking skills not necessary coding ones (we even find the relevant API relatively easy on Google) but something more fundamental complex, perplex and invisible in order to solve our beginning maintenance problem effectively. This book is set of effective design and coding practices and conceptual framework to produce and sense good design and code. Let's start with working definitions (which will be extended by the books flow):

- Functional Requirements - for now simply said are user's problems (Problem Domain) for our software system to implement such as update word counting program definition at the beginning.
- Coding Craft - skills and practices needed to implement particular functionality in piece of code include testing.
- Design Skills – for the lack of better definitions I define Design Skills as skills to perform all activities and produce all artifacts between functional requirements and coding. These tricky artifacts are not visual and this book is all about them. I see them as art of software development (compare to coding and documentation which is relatively craft based and manageable process).

In all un-successful projects I have seen the coding skills were not the scarcest resource. The scarcest and the tricky ones are the design skills and knowledge. Good design somehow similar to Black Swan, hard to judge before seeing one. Also rules to benefit for Black Swan and design are the same, care for worst (use practices described in book) and let the best take care of his own. Software is complex and multivariable and any myopia looking on software as simple code (or small subset of problem) to implement functional requirements misses the whole point and gets the wrong picture. Looking at software developer as coder (similar to the shoe cobbler), we miss the important part by only looking at visible and easy. Coding skills are important do have their respected place (and a chapter in book), but they don't count for everything. Even in my teaching practice the demand for delivering coding courses (Android, Java, C++) is higher than the demand for the design courses.

Every capable software student, beginning or medium engineer can write (or take from Google) piece of code be it a function or a class with methods that accept X and return Y. It takes experience, learning and most important mistakes to design and to sense a good design and them as a derivative to write good code. You wrote function excellent, now what if 100 (and then 1000) threads access it. You wrote function that draws triangle, great what about drawing 1000 triangles in a second, and yes it should be easily portable to other platforms (some with GPUs some without).You fixed some code here, how to make sure that your change doesn't introduce new problems.

The Software Industry is facing with following challenges: 1.Larger programs – rule programs become larger and larger and dependent of more and more components.

2.Larger quantities of data that the programs handle

3.Concurrency (Multi Cores/Threads/Processes ...) is a norm rather than exception

4.Human Factor is the same; we are bounded by same brain activities to comprehend the above challenges.

As you see from above the each of the challenges above is basically design issues. In order to be better than competitors or survive in market we need to apply effective design and coding practices to set them right. I collected a few of them and present them now in this book. Each sentence or rule reflects mine and others mistakes I have done and decided to write down in order not to repeat them and have a professional compass. Override these rules only with a good reason.

Chapters Flow

I ordered the chapters according to importance and time spent on various aspects, the most important come first:

1.Requirements Engineering– the fundamentals of understanding conceptual elements i.e. problem and solutions domains. Good design covers both functional and non-functional requirements.

2.Frameworks – the lion's share of our time spent on using and understanding frameworks. Software designed properly lives peacefully and allows easy transfer of knowledge with underlying framework or platform.

3.Programming Paradigms – before and while coding you need to make sure you master the key programming paradigms. Paradigms are means to represent patterns (not only design patterns) and to reuse knowledge.

4.Coding as Craft – set of effective practices while coding.

5.API Design & Software Architectures – we actually spent a very limited time of actual architecture and API design, but it is important to set it right, especially the effective API Design practices and the costly design decisions. Another look for book's outline; where external circles are visible skills while the big internal are design, invisible skills.

The idea of this book is developing the design sense—not necessarily the ability to produce good design on demand, but the ability to recognize good design. While looking for design/code I look for the following items described in the upcoming chapters. There is positive correlation between good design and the subjects above. Note the division to chapters there are might be other divisions, but this order suits me the best.

Acknowledgments Avner Ben

- Chief Architect at Elisra Systems
- Skill Tree ® designer

Scott Whitmire - Enterprise Architect at T-Mobile - Vice Chair, Board of Education at International Association of Software Architects (IASA) - Author of "Object Oriented Design and Measurement" book.

Mark Poyas - Senior Quality Analyst at Redbend

1

Markdown

1.1 Why Should You Care?

A Leanpub book is composed of a bunch of Markdown files.

The order of these files is defined in a file called Book.txt, which is in the same folder as this file.

To learn about the syntax and philosophy of Markdown, see this article by John Gruber[1].

Briefly, Markdown is a nice way of writing content which is easily transformed into HTML. For example, # at the beginning of a line becomes an h1, ## becomes an h2, ### becomes an h3, etc. Lists, paragraphs and other formatting is also intuitive.

1.2 How Markdown is Used in Leanpub

Even if you know Markdown, you need to learn a few things about how we use it at Leanpub.

[1]http://daringfireball.net/projects/markdown/syntax

Heading Levels Become Chapters and Sections

First, we use # chapters, ## for sections and ### for sub-sections. (You can also use #### for sub-sub-sections, but don't get carried away! Most technical books are good with just #, ## and ###, and most business and fiction books are good with just # and possibly ##.)

So, if you look at the top of the Markdown.txt file you'll see that it has one #, meaning it is a Chapter.

Another thing this means is that one file can contain as many chapters or sections as you want: every # makes a new chapter; it has nothing to do with what file it is in. However, we strongly recommend having one file per chapter (or one file per chapter section), since it makes creating sample books easier and keeps your book directory cleaner. Since we recommend this, that's what we'll do in this example book.

Links Become Footnotes

We support Markdown syntax for links, as well as normal HTML links. Both of these are converted into functioning footnotes in the PDF. Here's an example of a link to Leanpub[2].

[2]http://leanpub.com

1.3 Markdown Extensions in Leanpub

We've made a few additions to Markdown for use in Leanpub. Two of the most important are tables and crosslinks. Furthermore, since Leanpub is so good for technical books, we also support extensions for external code samples, special directives for code syntnax highlighting, etc. If this is a technical book, you will see these discussed in the Code Samples chapter.

Tables

Creating a table in a Leanpub book is relatively simple. Here's an example:

First Name	Last Name	Email
Peter	Armstrong	peter@leanpub.com
Scott	Patten	scott@leanpub.com

Yes, that's it!

Crosslinks

A crosslink lets you refer to another element of your book. For example, you can refer to another section or figure, even if it's in a different chapter.

Creating crosslinks is a two-step process.

1. You need to set a name for the thing you want to link to. For example, you can name any chapter or section by putting {#some-name} after the chapter title.

2. You need to link to it in the text. You do this with a regular Markdown link with a target of “#some-name”; for example this is a cross-link to the Why Sample Books section in the Sample Books chapter.

Note that for standalone crosslinks, this only works when the anchor {#some-name} is at the beginning of a section.

Footnotes

To add a footnote, you insert it like this[3] and then you define the footnote content later.

That’s it. Then you can keep writing content after the footnote content definition as well.

[3]This is the footnote text.