O'REILLY

Broadview
www.broadview.com.cn

传世经典书丛
Eternal Classics

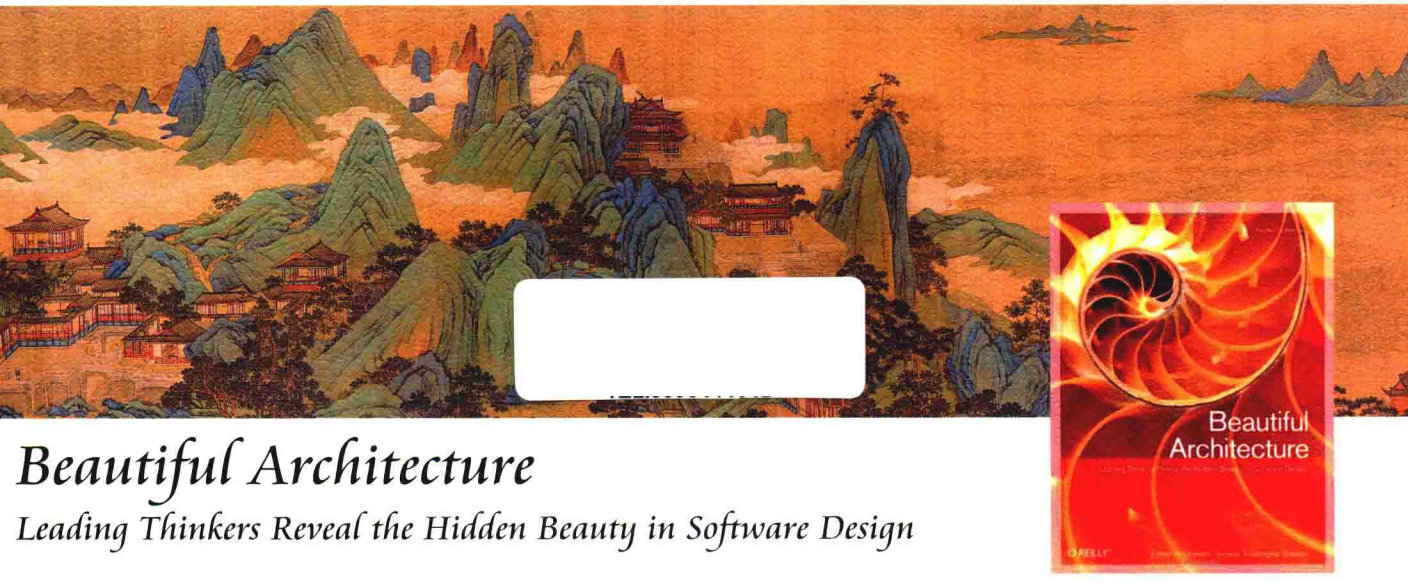
架构之美

行业思想领袖揭秘软件设计之美

（评注版）

【美】Diomidis Spinellis
Georgios Gousios 编

张逸 评注



Beautiful Architecture

Leading Thinkers Reveal the Hidden Beauty in Software Design

中国工信出版集团

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

传世经典书丛
Eternal Classics

O'REILLY®

架构之美

行业思想领袖揭秘软件设计之美（评注版）

Beautiful Architecture

Leading Thinkers Reveal the Hidden Beauty in Software Design

【美】Diomidis Spinellis
Georgios Gousios 编

张逸 评注

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书是荟萃了软件架构领域各位思想领袖真知灼见的经典之作，内容覆盖了软件架构的方方面面，包括架构理论、企业架构、系统架构、应用架构等。这些架构大师们用简洁的文本、真实的案例向读者勾勒出美丽架构的模样，并由此提出设计美丽架构的原则、实践与演进过程。全书传递的架构知识既有高屋建瓴的系统描述，又有深入系统的全面剖析，全面体现了架构设计中的简洁之美、清晰之美、风格之美、灵活之美和演进之美。

本书适合期待提高架构能力，学会欣赏架构之美的开发人员与架构师。

版权贸易合同登记号　图字：01-2013-4027

图书在版编目（CIP）数据

架构之美：行业思想领袖揭秘软件设计之美：评注版 /（美）迪奥米德斯 • 斯宾耐立思（Diomidis Spinellis），（美）乔治斯 • 郭西奥斯（Georgios Gousios）编；张逸评注. —北京：电子工业出版社，2018.6
（传世经典书丛）
书名原文：Beautiful Architecture: Leading Thinkers Reveal the Hidden Beauty in Software Design
ISBN 978-7-121-33807-6

Ⅰ. ①架… Ⅱ. ①迪… ②乔… ③张… Ⅲ. ①软件设计 Ⅳ. ①TP311.5

中国版本图书馆 CIP 数据核字（2018）第 042441 号

策划编辑：符隆美
责任编辑：付　睿
印　　刷：三河市良远印务有限公司
装　　订：三河市良远印务有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编：100036
开　　本：850×1168　1/16　印张：26.25　字数：550 千字
版　　次：2018 年 6 月第 1 版
印　　次：2018 年 6 月第 1 次印刷
定　　价：89.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：（010）51260888-819，faq@phei.com.cn。

悦读上品 得乎益友

孔子云："取乎其上，得乎其中；取乎其中，得乎其下；取乎其下，则无所得矣"。

对于读书求知而言，这句古训教我们去读好书，最好是好书中的上品——经典书。其中，科技人员要读的技术书，因为直接关乎客观是非与生产效率，阅读选材本更应慎重。然而，随着技术图书品种的日益丰富，发现经典书越来越难，尤其对于涉世尚浅的新读者，更为不易，而他们又往往是最需要阅读、提升的重要群体。

所谓经典书，或说上品，是指选材精良、内容精练、讲述生动、外延丰盈、表现手法体贴入微的读品，它们会成为读者的知识和经验库中的重要组成部分，并且拥有从不断重读中汲取养分的空间。因此，选择阅读上品的问题便成了有效阅读的首要问题。当然，这不只是效率问题，上品促成的既是对某一种技术、思想的真正理解和掌握，同时又是一种感悟或享受，是一种愉悦。

与技术本身类似，经典 IT 技术书多来自国外。深厚的积累、良好的写作氛围，使一批大师为全球技术学习者留下了璀璨的智慧瑰宝。就在那个年代即将远去之时，无须回眸，也能感受到这一部部厚重而深邃的经典著作，在造福无数读者后发出从未蒙尘的熠熠光辉。而这些凝结众多当今国内技术中坚美妙记忆与绝佳体验的技术图书，虽然尚在国外图书市场上大放异彩，却已逐渐淡出国人的视线。最为遗憾的是，迟迟未有可以填补空缺的新书问世。而无可替代，不正是经典书被奉为圭臬的原因？

为了不让国内读者，尤其是即将步入技术生涯的新一代读者，就此错失这些滋养过先行者们的好书，以出版 IT 精品图书，满足技术人群需求为己任的我们，愿意承担这一使命。本次机遇惠顾了我们，让我们有机会精心推出"传世经典书丛"。

在我们眼中，"传世经典"的价值首先在于——既适合喜爱科技图书的读者，也符合专家们挑剔的标准。幸运的是，我们的确找到了这些堪称上品的佳作。丛书带给我们的幸运颇多，细数一下吧。

得以引荐大师著作

有恐思虑不周，我们大量参考了国外权威机构和网站的评选结果，又进一步对符合标准之图书的国内

外口碑与销售情况进行细致分析，也听取了国内技术专家的宝贵建议，才有幸选出对国内读者最富有技术养分的大师上品。

向深邃的技术内涵致敬

中外技术环境存在差异，很多享誉国外的好书未必适用于国内读者；且技术与应用瞬息万变，很容易让人心生迷惘或疲于奔命。本丛书的图书遴选，注重打好思考方法与技术理念的根基，旨在帮助读者修炼内功，提升境界，将技术真正融入个人知识体系，从而可以一通百通，从容面对随时涌现的技术变化。

翻译与评注的双项选择

引进优秀外版著作，将其翻译为中文供国内读者阅读，较为有效与常见。但另有一些外语水平较高、喜好阅读原版的读者，苦于对技术理解不足，不能充分体会原文表述的精妙，需要有人指导与点拨。而一批本土技术精英经过长期经典熏陶及实践锤炼，已足以胜任这一工作。有鉴于此，本丛书在翻译版的同时推出融合英文原著与中文点评、注释的评注版，供不同志趣的读者自由选择。

承蒙国内一流译(注)者的扶持

优秀的英文原著最终转化为真正的上品，尚需跨越翻译鸿沟，外版图书的翻译质量一直屡遭国内读者诟病。评注版的增值与含金量，同样依赖于评注者的高卓才具。好在，本丛书得到了久经考验的权威译(注)者的认可和支持，首肯我们选用其佳作，或亲自参与评注工作。正是他们的参与保证了经典的品质，既再次为我们的选材把关，更提供了一流的中文表述。

期望带给读者良好的阅读体验

一本好书带给人的愉悦不止于知识收获，良好的阅读感受同样不可缺少，且对学业不无助益。为让读者收获与上品相称的体验，我们在图书装帧设计与选材用料上同样不敢轻率，惟愿送到读者手中的除了珠玑章句，还有舒适与熨帖的视觉感受。

所有参与丛书出版的人员，尽管能力有限，却无不心怀严谨之心与完美愿望。如果读者朋友能从潜心阅读这些上品中偶有获益，不啻为对我们工作的最佳褒奖。若有阅读感悟，敬请拨冗告知，以鼓励我们继续在这一道路上贡献绵薄之力。如有不周之处，也请不吝指教。

电子工业出版社博文视点

All royalties from this book will be donated to Doctors Without Borders.

评注者序

软件架构终归属于工程学的范畴，不能一概以“只可意会不可言传”来搪塞，因为架构知识是可以传递的，架构文档是可以共享的，最重要的是，架构自身是可以评审、验证与实现的。

Stephen J. Mellor 在“*Beautiful Architecture*”一书的序中，画龙点睛地勾勒出美丽架构的模样，即必须遵循的一些普遍原则，分别为：

- One fact In one place（一处一事实）
- Automatic propagation（自动传播）
- Architecture includes construction（架构包含构建）
- Minimize mechanisms（最小化机制）
- Constuct engines（构建引擎）
- O(G), the order of growth（O(G)，增长的阶）
- Resist entropy（抵制熵）

这些原则，其实就是架构师的智慧，没有足够深刻的理解与深入实践，是不可能给出如此言简意赅的架构建议的。按照我的理解，这些普适性原则其实就是在说明所谓美丽的架构，**就是简单、一致、适应变化并能去除重复的架构**。就如 Mellor 所言——**美丽的架构能用更少的机制做更多的工作**。这就是“*Beautiful Architecture*”一书不凡的开篇。

若是一本平庸的书，必然会惧怕这样精彩绝伦的序，因为它愈发的美，就愈发能照映出正文的丑；它愈发的言之有物，又愈发会衬托出正文的空洞无味。然而，若是内容是超乎寻常的精彩绝伦，这样的序就无异于锦上添花，珠联璧合了。通透点儿，就是齐活！这就好比一首歌曲的领唱者，倘若一开始就飙出高音，声入云霄。后续跟着唱的人要是没有点儿本事，恐怕就难以为继了；可要都是高手呢？那就真是一场音乐的盛宴了。

“*Beautiful Architecture*”荟萃了全球最顶级的架构师和意见领袖，他们在这本书中唱出了架构思想、实践与原则的最强音。全书共分为以下 5 个主题。

- On Architeture
- Enterprise Application Architecture
- Systems Architecture
- End-User Application Architectures
- Languages and Architecture

这些主题几乎覆盖了软件架构的方方面面，精选的每篇文章可谓字字珠玑，充满了写作者的真知灼见。开卷阅读，如与大师对话，聆听者必须凝神应对，稍不留神就可能遗漏那些重要而正确的意见，影响到对整篇文章的理解。整本书正文不足 400 页，然而每次阅读皆有新意，书的内容仿佛博尔赫斯笔下小径分叉的花园，花园虽小，景色却变幻多姿，路途虽短，距离却无穷无尽，咫尺天涯。

因此，作为本书的评注者，真可以说是战战兢兢、如履薄冰。我的每句点评都尽力追求达到个人最大努力的完美，不求锦上添花，只求不得“狗尾续貂”之嫌。安全地说，这些评注不过是我写在这本大书边上的感悟罢了。这些感悟，或是阅读到精彩段落的击节赞叹，或是不明其义而反复研读之后的醍醐灌顶，或是触类旁通体会到架构本质因而不揣冒昧地给出自己的心得体会。在评注过程中，我恪守“扬长避短”的原则，不懂就不装懂，默不作声，当一位沉默的看客；一旦涉猎到自己擅长的部分，却也不妨洋洋洒洒高谈阔论，坦承自己的观点。

对于这些架构领袖们，我怀揣敬意，却也不愿以一种卑微的心态被动接受。我需要做一个具有自己人格和高度的评注者。至于对否，就交给读者诸君对评注再做一次“评注”吧！

张　逸

个人博客：http://zhangyi.xyz

Foreword

Stephen J. Mellor

THE CHALLENGES OF DEVELOPING HIGH-PERFORMANCE, HIGH-RELIABILITY, and high-quality software systems are too much for ad hoc and informal engineering techniques that might have worked in the past on less demanding systems. The complexity of our systems has risen to the point where we can no longer cope without developing and maintaining a single overarching architecture that ties the system into a coherent whole and avoids piecemeal implementation, which causes testing and integration failures.

But building an architecture is a complex task. Examples are hard to come by, due to either proprietary concerns or the opposite, a need to "sell" a particular architectural style into a wide range of environments, some of which are inappropriate. And architectures are big, which makes them difficult to capture and describe without overwhelming the reader.

Yet beautiful architectures exhibit a few universal principles, some of which I outline here:

One fact in one place

Duplication leads to error, so it should be avoided. Each fact must be a single, nondecomposable unit, and each fact must be independent of all other facts. When change occurs, as it inevitably does, only one place need be modified. This principle is well known to database designers, and it has been formalized under the name of *normalization*. The principle also applies less formally to behavior, under the name *factoring*, such that common functionality is factored out into separate modules.

▶ One fact in one place即"DRY (Don't Repeat Yourself)"原则。重复是糟糕架构的典型体现。针对一个问题域，提供多个大同小异的解决方案，会最终导致"解决方案蔓延"。一旦问题域发生变化，就需要多处进行修改。若某个解决方案忘记修改，则可能引入潜在的缺陷。

（未完见下页）

Beautiful architectures find ways to localize information and behavior. At runtime, this manifests as *layering*, the notion that a system may be factored into layers, each representing a *layer of abstraction* or *domain*.

Automatic propagation

One fact in one place sounds good, but for efficiency's sake, some data or behavior is often duplicated. To maintain consistency and correctness, propagation of these facts must be carried out automatically at construction time.

Beautiful architectures are supported by construction tools that effect *meta-programming*, propagating one fact in one place into many places where they may be used efficiently.

Architecture includes construction

An architecture must include not only the runtime system, but also how it is constructed. A focus solely on the runtime code is a recipe for deterioration of the architecture over time.

Beautiful architectures are *reflective*. Not only are they beautiful at runtime, but they are also beautiful at construction time, using the same data, functions, and techniques to build the system as those that are used at runtime.

Minimize mechanisms

The best way to implement a given function varies case by case, but a beautiful architecture will not strive for "the best." There are, for example, many ways of storing data and searching it, but if the system can meet its performance requirements using one mechanism, there is less code to write, verify, maintain, and occupy memory.

Beautiful architectures employ a minimal set of mechanisms that satisfy the requirements of the whole. Finding "the best" in each case leads to proliferation of error-prone mechanisms, whereas adding mechanisms parsimoniously leads to smaller, faster, and more robust systems.

Construct engines

If you wish to build brittle systems, follow Ivar Jacobson's advice and base your architecture on use cases and one function at a time (i.e., use "controller" objects). Extensible systems, on the other hand, rely on the construction of virtual machines—engines that are "programmed" by data provided by higher layers, and that implement multiple application functions at a time.

This principle appears in many guises. "Layering" of virtual machines goes back to Edsger Dijkstra. "Data-driven systems" provide engines that rely on coding invariants in the system, letting the data define the specific functionality in a particular case. These engines are highly reusable—and beautiful.

（接上页）
要避免重复，其手段可以选择抽象（Abstraction）或分解（Partition）。利用抽象可寻求其共性特征，进行共性与可变性分析，既精简了模型，又可重用共性部分。利用分解，可缩小逻辑单元，使其变得更加可重用。然而，此两种方式都是有代价的。抽象会导致间接层次增多，从而影响性能；分解则会带来大量细粒度单元，使得系统的实体（模块、类、方法）数量增加，增加了系统的复杂性。
因此，面对重复，我们需要权衡。设计的重点就在于权衡。

O(G), *the order of growth*

Back in the day, we thought about the "order" of algorithms, analyzing the performance of sorting, say, in terms of the time it takes to sort a set of a certain number of elements. Whole books have been written on the subject.

The same applies for architecture. Polling, for example, works well for a small number of elements, but is a response-time disaster as the number of items increases. Organizing everything around interrupts or events works well until they all go off at once. Beautiful architectures consider the direction of likely growth and account for it.

Resist entropy

Beautiful architectures establish a path of least resistance for maintenance that preserves the architecture over time and so slows the effects of the Law of System Entropy, which states that systems become more disorganized over time. Maintainers must internalize the architecture so that changes will be consistent with it and not increase system entropy.

One approach is the Agile concept of the *Metaphor*, which is a simple way to represent what the architecture is "like." Another is extensive documentation and threats of unemployment, though that seldom works for long. Usually, however, it generally means tools, especially for generating the system. A beautiful architecture must remain beautiful.

These principles are highly interrelated. One fact in one place can work only if you have automatic propagation, which in turn is effective when the architecture takes construction into account. Similarly, constructing engines and minimizing mechanisms support one fact in one place. Resisting entropy is a requirement for maintaining an architecture over time, and it relies on the architecture including construction and support for propagation. Moreover, a failure to consider the way in which a system will likely grow will cause the architecture to become unstable, and eventually fail under extreme but predictable circumstances. And combining minimal mechanisms with the notion of constructing engines means that beautiful architectures usually feature a limited set of patterns that enable construction of arbitrary system extensions, a kind of "expansion by pattern."

In short, beautiful architectures do more with less.

As you read this book, ably assembled and introduced by Diomidis Spinellis and Georgios Gousios, you might look for these principles and consider their implications, using the specific examples presented in each chapter. You might also look for violations of these principles and ask whether the architecture is thus ugly or whether some higher principle is involved.

During the development of this Foreword, your authors asked me if I might say a few words about how someone becomes a good architect. I laughed. If we only knew that.... But then I recalled from my own experience that there is a powerful, if nonanalytic, way of becoming a

▶当今的软件系统，规模越来越大，数据越来越海量，空间复杂度与时间复杂度会成为系统设计的主要关注点。这带来的影响是，无论架构还是编码，都需要提前考虑系统可能面对的极限环境。例如内存是否会溢出，分布式环境下网络连接是否中断，请求数过多是否会造成应用服务的阻塞……

这些皆可视为“风险”。在进行软件架构时，需要提前识别这些风险，并对风险进行优先级排列；然后从这些风险出发去寻找合理的解决方案。应该对解决方案进行为期较短的技术预研（Spike），并搭建与真实环境相当的虚拟测试环境。

beautiful architect. That way* is never to believe that the last system you built is the only way to build systems, and to seek out many examples of different ways of solving the same type of problem. The example beautiful architectures presented in this book are a step forward in helping you meet that goal.

* Or exercise more and eat less.

Preface

THE IDEA FOR THE BOOK YOU'RE READING WAS CONCEIVED IN 2007 as a successor to the award-winning, best-selling *Beautiful Code*: a collection of essays about innovative and sometimes surprising solutions to programming problems. In *Beautiful Architecture*, the scope and purpose is different, but similarly focused: to get leading software designers and architects to describe a software architecture of their choice, peeling back the layers of their creations to show how they developed software that is functional, reliable, usable, efficient, maintainable, portable, and, yes, elegant.

To put together this book, we contacted leading architects of well-known or less-well-known but highly innovative software projects. Many of them replied promptly and came back to us with thought-provoking ideas. Some of the contributors even caught us by surprise by proposing not to write about a specific system, but instead investigating the depth and the extent of architectural aspects in software engineering.

All chapter authors were glad to hear that the work they put in their chapters is also helping a good cause, as the royalties of this book are donated to *Medécins Sans Frontières* (Doctors Without Borders), an international humanitarian aid organization that provides emergency medical assistance to suffering people.

How This Book Is Organized

We have organized the contents of this book around five thematic areas: overviews, enterprise applications, systems, end-user applications, and programming languages. There is an obvious, but not deliberate, lack of chapters on desktop software architectures. Having approached more than 50 software architects, this result was another surprise for us. Are there really no shining examples of beautiful desktop software architectures? Or are talented architects shying away from an area often driven by a quest to continuously pile ever more features on an application? We are really looking forward to hearing from you on these issues.

Part I: On Architecture

Part I of this book examines the breadth and scope of software architecture and its implications for software development and evolution.

Chapter 1, *What Is Architecture?*, by John Klein and David Weiss, defines software architecture by examining the subject through the perspectives of quality concerns and architectural structures.

Chapter 2, *A Tale of Two Systems: A Modern-Day Software Fable*, by Pete Goodliffe, provides an allegory on how software architectures can affect system evolution and developer engagement with a project.

Part II: Enterprise Application Architecture

Enterprise systems, the IT backbone of many organizations, are large and often tailor-made conglomerates of software usually built from diverse components. They serve large, transactional workloads and must scale along with the enterprise they support, readily adapting to changing business realities. Scalability, correctness, stability, and extensibility are the most important concerns when architecting such systems. Part II of this book includes some exemplar cases of enterprise software architectures.

Chapter 3, *Architecting for Scale*, by Jim Waldo, demonstrates the architectural prowess required to build servers for massive multiplayer online games.

Chapter 4, *Making Memories*, by Michael Nygard, goes through the architecture of a multistage, multisite data processing system and presents the compromises that must be made to make it work.

Chapter 5, *Resource-Oriented Architectures: Being "In the Web"*, by Brian Sletten, discusses the power of resource mapping when constructing data-driven applications and provides an elegant example of a purely resource-oriented architecture.

Chapter 6, *Data Grows Up: The Architecture of the Facebook Platform*, by Dave Fetterman, advocates data-centric systems, explaining how a good architecture can create and support an application ecosystem.

Part III: Systems Architecture

Systems software is arguably the most demanding type of software to design, partly because efficient use of hardware is a black art mastered by a selected few, and partly because many consider systems software as infrastructure that is "simply there." Seldom are great systems architectures designed on a blank sheet; most systems that we use today are based on ideas first conceived in the 1960s. The chapters in Part III walk you through four innovative systems software architectures, discussing the complexities behind the architectural decisions that made them beautiful.

Chapter 7, *Xen and the Beauty of Virtualization*, by Derek Murray and Keir Fraser, gives an example of how a well-thought-out architecture can change the way operating systems evolve.

Chapter 8, *Guardian: A Fault-Tolerant Operating System Environment*, by Greg Lehey, presents a retrospective on the architectural choices and building blocks (both software and hardware) that made Tandem the platform of choice in high-availability environments for nearly two decades.

Chapter 9, *JPC: An x86 PC Emulator in Pure Java*, by Rhys Newman and Christopher Dennis, describes how carefully designed software and a good understanding of domain requirements can overcome the perceived deficiencies of a programming system.

Chapter 10, *The Strength of Metacircular Virtual Machines: Jikes RVM*, by Ian Rogers and Dave Grove, walks us through the architectural choices required for creating a self-optimizable, self-hosting runtime for a high-level language.

Part IV: End-User Application Architectures

End-user applications are those that we interact with in our everyday computing lives, and the software that our CPUs burn the most cycles to execute. This kind of software normally does not need to carefully manage resources or serve large transaction volumes. However, it does need to be usable, secure, customizable, and extensible. These properties can lead to popularity and widespread use and, in the case of free and open source software, to an army of volunteers willing to improve it. In Part IV, the authors dissect the architectures and the community processes required to evolve two very popular desktop software packages.

Chapter 11, *GNU Emacs: Creeping Featurism Is a Strength*, by Jim Blandy, explains how a set of very simple components and an extension language can turn the humble text editor into ~~an operating system~~* the Swiss army knife of a programmer's toolchest.

* As some die-hard users say, "Emacs is my operating system; Linux just provides the device drivers."

Chapter 12, *When the Bazaar Sets Out to Build Cathedrals*, by Till Adam and Mirko Boehm, demonstrates how community processes such as sprints and peer-reviews can help software architectures evolve from rough sketches into beautiful systems.

Part V: Languages and Architecture

As many people have pointed out in their works, the programming language we use affects the way we solve a problem. But can a programming language also affect a system's architecture and, if so, how? In the architecture of buildings, new materials and the adoption of CAD systems allowed the expression of more sophisticated and sometimes strikingly beautiful designs; does the same also apply to computer programs? Part V, which contains the last two chapters, investigates the relationship between the tools we use and the designs we produce.

Chapter 13, *Software Architecture: Object-Oriented Versus Functional*, by Bertrand Meyer, compares the affordances of object-oriented and functional architectural styles.

Chapter 14, *Rereading the Classics*, by Panagiotis Louridas, surveys the architectural choices behind the building blocks of modern and classical object-oriented software languages.

Finally, in the thought-provoking Afterword, William J. Mitchell, an MIT Professor of Architecture and Media Arts and Sciences, ties the concept of beauty between the building architectures we encounter in the real world and the software architectures residing on silicon.

Principles, Properties, and Structures

Late in this book's review process, one of the reviewers asked us to provide our personal opinion, in the form of commentary, on what a reader could learn from each chapter. The idea was intriguing, but we did not like the fact that we would have to second-guess the chapter authors. Asking the authors themselves to provide a meta-analysis of their writings would lead to a Babel tower of definitions, terms, and architectural constructs guaranteed to confuse readers. What was needed was a common vocabulary of architectural terms; thankfully, we realized we already had that in our hands.

In the Foreword, Stephen Mellor discusses seven principles upon which all beautiful architectures are based. In Chapter 1, John Klein and David Weiss present four architecture building blocks and six properties that beautiful architectures exhibit. A careful reader will notice that Mellor's principles and Klein's and Weiss's properties are not independent of each other. In fact, they mostly coincide; this happens because great minds think alike. All three, being very experienced architects, have seen many times in action the importance of the concepts they describe.

We merged Mellor's architectural principles with the definitions of Klein and Weiss into two lists: one containing principles and properties (Table P-1), and one containing structures (Table P-2). We then asked the chapter authors to mark the terms they thought applied to their chapters, and produced a corresponding legend for each chapter. In these tables, you can see the definition of each principle, property, or architectural construct that appears in the chapter legend. We hope the legends will guide your reading of this book by giving you a clean overview of the contents of each chapter, but we urge you to delve into a chapter's text rather than simply stay with the legend.

TABLE P-1. Architectural principles and properties

Principle or property	**The ability of an architecture to...**
Versatility	...offer "good enough" mechanisms to address a variety of problems with an economy of expression.
Conceptual integrity	...offer a single, optimal, nonredundant way for expressing the solution of a set of similar problems.
Independently changeable	...keep its elements isolated so as to minimize the number of changes required to accommodate changes.
Automatic propagation	...maintain consistency and correctness, by propagating changes in data or behavior across modules.
Buildability	...guide the software's consistent and correct construction.
Growth accommodation	...cater for likely growth.
Entropy resistance	...maintain order by accommodating, constraining, and isolating the effects of changes.

TABLE P-2. Architectural structures

Structure	**A structure that...**
Module	...hides design or implementation decisions behind a stable interface.
Dependency	...organizes components along the way where one uses functionality of another.
Process	...encapsulates and isolates the runtime state of a module.
Data access	...compartmentalizes data, setting access rights to it.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.