架构是系统设计的一部分，它突出了某些细节，并通过抽象省略掉了另一些细节。软件系统的架构包括行为上的和结构上的。外部行为描述展示了软件如何与用户、其他设备和外部设备进行交互，也就是需求。结构描述展示了软件如何被划分为多个部分，以及这些部分的关系。

架构的设计受到许多因素的制约，架构是好是坏并没有统一的标准。这取决于人们对软件的需求、软件被构建和运行的环境，以及软件团队本身的特点等等因素。评价软件好坏有很多指标，例如性能、安全、可伸展性等等。一般来说，这些指标是很难全部满足的，试图改进其中一个往往会对其他指标产生负面影响。所以从某种意义上来说，软件架构是折中的游戏。对于一组功能需求和品质需求，没有唯一的正确架构。

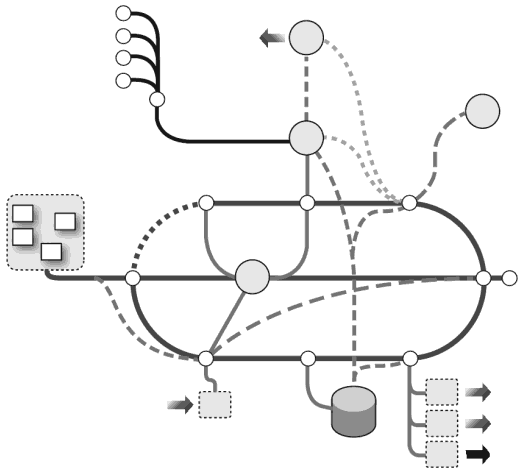
《架构之美》没有太多空洞的概念和论述，而是抛砖引玉地展示多个实际的项目。通过对它们架构利弊的分析，以及相关的思考，给读者提供了有益的启发。在这里我简单介绍几个其中的例子。

“混乱大都市”和“设计之城”

一个例子是两个类似但是命运完全不同的项目：“混乱大都市”和“设计之城”。

“混乱大都市”来自于一家成立不久的公司。软件的延期是不可容忍的，所以软件工程师们被迫尽其极限快速交付。代码是以一系列的疯狂冲刺方式堆叠在一起的，这使得“混乱大都市”从一开始就缺乏规划，导致了一系列恶果。结构混乱难以理解，没有清晰的层次，模块间的依赖关系错综复杂；组件的角色定义模糊，本该在这个组件的事情却在另一个组件里实现；整个项目完全是由胶带、Shell脚本、Perl胶水、makefile和Visual Studio项目混合在一起的。

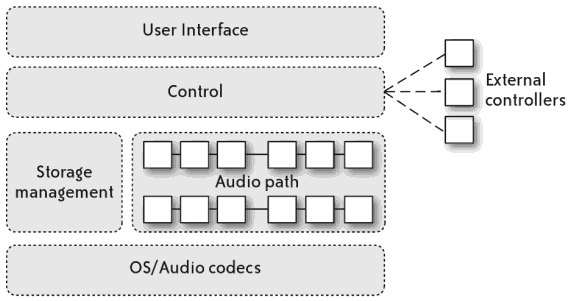
新手难以快速熟悉这个项目，就连老手也不知道整个项目的全景。所有人花费了大量精力得到了一份“架构图”，看上去像伦敦地铁图，甚至还有环线：



最终这家公司放弃了“混乱大都市”项目，并组织团队重新编写，祝他们好运……

“设计之城”与“混乱大都市”类似，都是C++编写的音频软件。从一开始，项目就有了清晰的目标：具体的首个产品和未来功能的路线图，代码必须支持这些功能和未来的变化。

团队采用了极限编程的方法，对于项目的很多关注点作了规定。例如统一的文件结构、命名规范、编码惯例等等。并且在系统核心的设计上花费的不少时间，最终得到了一个“过滤器和管道”的架构：



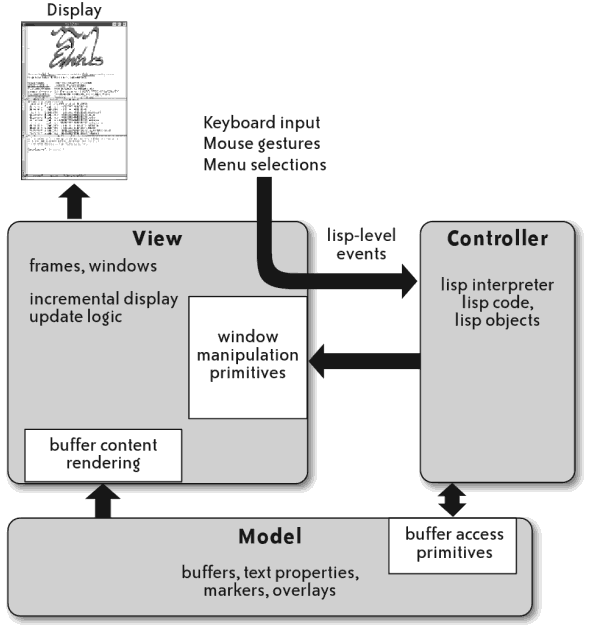
此书编写时“设计之城”仍然在使用中，并且扩展出了一些成功的产品。事实上，“设计之城”的代码并不完美，有些地方存在技术争论，但是在整洁的背景下这些问题显得特别突出，从而在将来会得到解决。

**GNU Emacs**

另一个例子是GNU Emacs

Emacs是当今流行的文本编辑器之一。Emacs于上世纪70年代中旬开始开发，至今已走过了三十多年，但其架构基本没有改变过。而且，Emacs的特性在不断地增加中。

Emacs采用了交互式应用程序中应用广泛的“模型-视图-控制器”模式。模型是程序所操作数据的底层描述；视图时向用户展示数据的方法；控制器负责实现用户与视图的交互，并对模型进行更新。



* Emacs的模型，缓冲区就是简单的字符串。每个缓冲区都有一种模式，用来指定对特定类型文本进行编辑时的行为；缓冲区还维护了一个撤销日志，用来记住各个用户命令间的边界；此外缓冲区还为文本在缓冲区中的定位提供的支持；缓冲区中的每个字符有自己的属性，包括如何显示以及如何应对鼠标操作等。
* Emacs的视图可以自动更新显示结果，并且只在用户输入时更新。例如现在的网页浏览器就借鉴了这样的方式。
* Emacs的控制器是用自己独立的Emacs Lisp语言开发的。Emacs几乎所有的功能都由Emacs Lisp实现，用户可以自己编写Emacs Lisp程序来扩充Emacs的功能。事实上，通常的Emacs发行版就包含了很多流行的Emacs Lisp包。

随着软件的特性不断增长，用户界面将变得复杂，程序本身将变得难以维护。评价一个程序的用户界面复杂度，有两个常见的维度：要维护的模型的复杂度，以及操作该模型命令集的复杂度。

Emacs的模型，即缓冲区仅仅是字符串，缓冲区的改变永远是可见的；Emacs的命令非常之多，但是一个新用户完全可以像使用其他基本的文本编辑器一样使用Emacs，Emacs还提供了许多工具用于发现和理解新功能。Emacs更像是一个包的组合体，由社区的许多人员共同维护。Lisp语言在这当中扮演了重要的抽象边界的角色。

另外还有两个类似的架构：Eclipse和Firefox

**Eclipse**

Eclipse实际上是一个开发环境的框架，它本身没有提供任何相关功能。通过Java Development Tools等类型的插件，Eclipse就可以对软件开发提供大量广泛的支持，而这正是Emacs所欠缺的。

Eclipse为每个插件的输入和显示都提供了非常底层的支持，这允许插件实现更加复杂的功能，但也带来了问题：

Eclipse的插件开发是不安全的，而这正是Emacs具备的。一个充满bug的插件会让Eclispe崩溃，而在Emacs中用户可以终止这样的插件，并且用户的数据不会被破坏。

Eclipse插件间的结构比较复杂，这对插件作者是个挑战。

插件需要足够的样板文件代码。Eclipse中有个帮助大家编写插件的插件Eclipse Plug-in Development Environment，但这并不能降低底层接口的复杂性。

这引了一些思考。在插件中究竟应该使用哪类接口？插件开发人员应该在怎样的抽象级别上开发，能否更接近问题？如果保护数据不被充满bug的插件破坏？

**Firefox**

Firefox作为现代的浏览器，支持JavaScript动态的修改页面，这一点很像Emacs中的Lisp和缓冲区。不过，Firefox更加彻底地让自己的用户界面，以及许多内置功能都使用JavaScript来实现。

这也引起一些思考。在所有使用插件的程序中，插件的开发语言是不是为程序添加功能最好的方法？如果不是，为什么？

感想

引用William J. Mitchell为此书所写的文章：

*“好的建筑师不是通过随意的、蛮力的方式来构造他们的设计，他们肯定会避免笨拙的做法。”“在漂亮的建筑作品所表现出来的不同和复杂性背后，你通常会发现一些关于功能组织和规范秩序的简单而优雅的原则。发现这些原则需要思考，而思考正是建筑的快乐和经验的关键部分。”*

*“如果你能弄明白这些原则，就可以通过某种标准编程语言中同样优雅的几行代码构造出这些作品的模型；但如果你弄不明白这些原则，那么肯定要编写更长的、不包含那么多深刻见解的代码。”*

*“建筑师会仰慕那些应用了简单优雅原则来实现许多复杂性的架构之美。类似地，软件架构师和程序员也会仰慕那些清晰而准确执行了许多复杂任务的代码之美。科学家会仰慕那些描述了各种不同现象的简单法则之美和它们的解释能力。”*

架构之美在于其简洁深刻地描述和解决现实的软件设计问题，无论是它为软件开发人员减少的麻烦，还是本身形式上的和谐和整齐，都足以让人赏心悦目。