

第 1 讲 介绍

1.1 关于 6.170

本课程实际上由三部分内容组成：

- 面向对象编程的速成
- 媒体的软件设计
- 软件的团队开发

设计是重点，编程是必要条件，只有当你试着去使用项目时你才会真正理解一个思想。因此，这三部分都包含在本课程中。

你将会学到：

• 如何设计软件：有力的抽象化机制；在实践中被发现工作得很好的模式；如何描述设计以使你能够沟通和批评它们。

- 如何使用 Java 来实现。
- 如何使它正确工作：获得可靠的、灵活的软件。

还包括：

- 如何当一名架构师，而不仅仅只是一个低级的编码者。
- 如何避免在调试时浪费时间。

1.2 管理与政策

课程人员介绍：

- 讲师：Daniel Jackson 和 Rob Miller
- 教学助手：将在下周复习时与大家见面
- 实验助手：将成组地与大家见面
- 时间：参考网站。讲师没有固定的办公时间，但他们很乐于跟学生谈论——请发送电子邮件或直接访问。

材料：

- Liskoy 编写的课本：根据课程进度安排进行阅读。
- 讲稿：通常在讲座当天发布。
- “Gang of Four” 的设计模式书籍：推荐
- Bloch 的 “Effective Java”：推荐
- Java 教程：细节请参考一般的印刷品

推荐的这些资料真的很棒：将会作为参考，并且将帮你快速成为一个优秀的程序员。成包购买将会有优惠。

课程组织：

- 上半学期：讲座，每周练习，复习，小测验
- 下半学期：团队项目。这在以后详细介绍

学期初的变化：现在不必担心你们组将会有谁，可以在学期中交换教学助手。

复习：

- 每周教学助手将对学生的工作进行评价。
- 最初，教学助手将会把精力集中在你的一部分工作。
- 整个阶段将会以建设性和协同合作的方式讨论。
- 完全必要的部分课程：有机会去看看课堂中谈到的思想如何在实际中运用。

学习 Java：

- 这由你决定，但我们会试着帮助你。
- 使用 Sun 公司的 Java 教程并作些练习
- 大型团队的实验助理将会帮助你。

合作和 IP 规则：

- 浏览一般的信息。
- 简而言之：你们可以讨论，但书面作业必须你自己做，包括说明、设计、代码、测试和解释。
- 可以使用公共领域的代码。
- 团队开发中每件事都能合作。

测验：

- 两次随堂测验，主要考核讲义材料。

成绩：

- 70% 的个人工作=25% 的测验+45% 的问题回答
- 30% 的最终项目，团队中所有人获得相同的成绩
- 10% 的额外参与成绩
- 工作要按时

1.3 为什么软件工程很重要

软件对美国经济的贡献（1996 年的数据）：

- 出口的最大贸易盈余。
- \$240 亿的软件出口，\$40 亿的进口，\$200 亿的盈余。
- 比较：农业是 26-14-12，航空是 11-3-8，化学药品是 26-19-7，交通工具是 21-43-(22)，制造业是 200-265-(64)（来自 *Software Conspiracy*, Mark Minasi, McGraw Hill, 2000)

整个社会基础中各部分角色：

- 不只是 Internet，
- 包括运输、能源、药、财政。

软件在嵌入式设备中变得越来越普遍。例如，新汽车有 10 到 100 个处理器用来处理从音乐到刹车的各种功能。

软件的价格：

- 软硬件采购费的比率接近零
- 花费的总费用：五倍于硬件花费。Gartner 团体评估：一台个人计算机使用五年需要花费\$7-14K。

我们的软件有多好？

- 失败的开发
- 意外事件
- 质量差的软件

1.3.1 开发失败

IBM 调查（1994 年）

- 55% 的系统花费超过预期
- 68% 不能按时完成
- 88% 不得不重新设计

高级自动化系统 (FAA, 1982-1994 年)

- 工业平均是 \$100 / 工作线，预期支付 \$500 / 工作线
- 最终支付是 \$700-900 / 工作线
- \$60 亿 的工作丢弃

劳动署统计（1997 年）

- 每 6 个新的系统投入运营，就会有 2 个被取消。
- 最大的系统取消率的概率是 50%。
- 平均每个项目超过计划时间的 50%。
- 3/4 的系统被认为是“操作失败”。

1.3.2 意外事件

“大部分专家认为最有可能破坏这个世界的可能是意外事件，也就是我们进来的地方。我们是计算机专业人士，是我们引起了意外事件。”

Nathaniel Borenstein, MIME 的发明者，出自： *Programming as if People Mattered: Friendly Programs, Software Engineering and Other Noble Delusions*, Princeton University Press, Princeton, NJ, 1991.

Therac-25（1985-87 年）

- 带有软件控制器的放射线治疗仪器。
- 抛弃了硬件互锁，但软件没有互锁。
- 软件不能维持必要的不变量：电子束模或是强烈的光线和金属板干扰，产生 X 射线。
- 因为灼伤导致的几起死亡。
- 程序员对并发编程没有经验。
- 浏览：<http://sunnyday.mit.edu/therac-25.html>

你可能认为我们已经吸取了教训，这样的灾难将永远不会再次发生。但是…

- 2001年5月22日国际原子能机构代理在巴拿马宣布“放射线紧急状况”。
- 28位病人感光过度；8人死亡，其中有3人因为这件事情；20个幸存者中有3/4的人可能会“在某些情况下可能病情恶化而导致死亡”。
- 专家发现放射线疗法仪器“工作适当”，由于数据进入而引起紧急状况。
- 如果数据为一些屏蔽区批量进入，会引起不正确的剂量计算。
- 至少，食品药品监督管理局总结认为“软件对电子束数据的解释”是一个因素。
- 浏览 <http://www.fda.gov/cdrh/ocd/panamaradexp.html>

Ariane-5（1996年6月）

- 欧洲太空机构
- 无人火箭在升空后不久完全损坏
- 由于在Ada代码中抛出了异常
- 错误代码在火箭升空后也可能要运行
- 由于物理环境的改变：没有说明的假定被违反
- 浏览 <http://www.esa.int/htdocs/tidc/Press/Press96/ariane5rep.html>

Ariane的意外事故在大部分软件灾难中比起放射线疗法机器制造的意外事故更加典型。在代码中它几乎不可能是造成灾难的起因；通常，这个问题得追溯到需求分析，如一个连接和评估重要环境假设的失败。

伦敦救护车服务（1992年）

- 呼叫丢失，由于重复的呼叫而双倍的派遣
- 开发者的错误选择：没有足够的经验
- 浏览：<http://www.cs.ucl.ac.uk/staff/A.Finkelstein/las.html>

伦敦救护车灾难真的是管理上的一次失误。制造软件的管理者们很幼稚，并且接受了来自一家未知的公司投标，该公司的投标比许多其它有好评的公司的竞标低。而且他们犯了可怕的错误，试图突然运行，没有同时运行新的和旧的系统。

在短期内，这些问题将变得更加复杂，因为我们国内普遍使用这些软件。PITAC报告指出了这个问题，而且成功的为增加资金筹备用于软件研究进行辩护：

“对软件的需求已经变得远比我们生产的能力快。此外，国家需要的软件远比我们现在制造的要更加好用、可靠且有力。我们已经变得非常依赖于大型的软件系统，这些系统的行为并不好理解而且还常常以无可预测的方式出现问题。”

信息技术研究：对我们将来的投资

信息技术咨询委员会主席（PITAC）对总统的报告，1999年2月24日

可以在 <http://www.ccic.gov/ac/report/> 获得

关于危险的讨论会

- 比较来自媒体有关计算机事故的报告
- <http://catless.ncl.ac.uk>

1.3.3 软件质量

一种度量方法：每千行代码产生的错误（bugs/ kloc）

- 在交付后测量
- 工业上平均是 10
- 高质量：1 或者更少

Praxis CDIS 系统（1993 年）

- 为终端区域设计的英国空中交通管制系统
- 采用了精确的规范语言，与我们将要学习的对象模型非常相似
- 没有增加网络开销
- 很少的错误率：大约每千行代码 0.75 错误
- 甚至提供到客户的保证

当然，质量不仅仅涉及到错误。你可以测试软件并且除去大部分引起失败的错误，但是你不可能使用大量的时间来做你期望做到的，因为有很多特殊的情况。为了解决这个问题，你必须从开始就把握好质量。

1.4 为什么设计很重要

“在我们获得好软件之前需要什么？汽车变得更好了因为日本显示汽车可以架构得更好。有人将不得不显示软件能够架构得更好。”

John Murray, FDA 的软件质量领袖

引用自 Software Conspiracy, Mark Minasi, McGraw Hill, 2000

6.170 这门课将向你显示堆砌代码（hacking code）不是构建软件的全部。实际上，它仅仅是其中的一部分。不要把代码想象成解决问题的一部分，一般的，它仅仅是问题的一部分。我们需要更好的方式来讨论软件而不是编码，这不会显得太繁琐、太直接、技术太容易落后。

设计和设计者的角色

- 提前思考总会有所帮助（这是件便宜的事情）。
- 不要在结束时才考虑质量：与测试的可靠性进行对比，更加有效，更加便宜。
- 使得代理和团队工作变得可能。
- 设计的缺陷影响用户：不连贯、不灵活、难于使用。
- 设计的缺陷影响开发者：接口不好，错误增多，难于增加新特点。

有趣的是，计算机科学专业的学生常常反对这个观点——软件开发就是工程事业。也许他们认为工程技术缺乏神秘性，或不适合他们潜在的黑客才能。相反，你在 6.170 课程中所学的技术将会让你更佳有效的利用你的才能和潜力。

即使专业程序员也会迷惑他们自己。在某次实验中，32 个 NASA 的程序员使用 3 个不同的测试技术来测试一些小程序。他们要求评估采用每个方法所获得的错误的比例。他们的直觉是错误的。他们认为基于规范的黑盒测试是最有效的，但是实际上阅读代码更有效（尽管代码没有注释）。通过阅读代码，他们可以提高 50% 的速度来发现错误。

Victor R.Basili and Richard W.Selby.

Comparing the Effectiveness of Software Testing Strategies.

IEEE Transactions on Software Engineering. Vol. SE-13, No.12, December 1987, pp. 1278-1296.

对于基础软件（如太空交通控制系统），设计是非常重要的。即使这样，很多工业管理者没有意识到 6.170 课程中我们教的思想所产生的影响。请看 John Chapin（以前的 6.170 的演讲者）和我所写的关于如何重新设计一个 CTAS 组件，一个新的空中交通控制系统，它使用 6.170 中的思想。

Daniel Jackson and John Chapin. *Redesigning Air-Traffic Control: An Exercise in Software Design*. IEEE Software, May/June 2000. 可以浏览 <http://sdg.lcs.mit.edu/~dnj/publications>.

1.4.1 Netscape 公司的故事

对于 PC 软件，有一个神话讲：设计不是很重要，因为市场是随着时间改变的。在这方面 Netscape 公司的倒闭是一个值得深思的故事。

最初 Illinois 大学的 NCSA Mosaic 小组开发了第一个广泛使用的浏览器，但是他们的工作太快而且不利落。他们创建了 Netscape 公司，在 1994 年 4 月到 12 月之间发行了 Netscape1.0，它可以在 3 个平台上运行，很快就成为了 Windows、Unix 和 Mac 上的占有统治地位的浏览器。Microsoft 公司于 1994 年 10 月开始开发 Internet Explorer1.0，并在 1995 年绑定在 Windows 95 上。

从 1995 年到 1997 年，在 Netscape 公司的快速发展期间，开发者努力工作来产生新的特点，而很少花时间进行设计。在软件商业收缩期间大部分公司相信设计是滞后的：即一旦你有的市场和引人注目的特性，你就可以重构代码从而获得纯设计的优势。Netscape 也不例外，它的工程师可能比很多其它公司的都有才能。

同时，Microsoft 公司意识到了需要建立可靠的设计，它从起草、重建 Office 套装到使用共享组件，建立了 NT 操作系统。并忙于通过 IE 来赶上 Netscape 的市场，但是它花费了时间来重构 IE3.0。这个 IE 的重构就是现在 Microsoft 中所看到的，它作为一个关键步骤，拉近了与 Netscape 的差距。

Netscape 的开发在继续。通过 Communicator4.0，已有 120 个开发者（从最初的 10 个）和 3 百万行代码。Michael Toy，开发经理，说道：

“我们处于一个很坏的情形下……我们应该在一年前就停住绑定这些代码。这是死的……这就像被粗鲁地唤醒……我们为走得太快而付出了代价。”

有趣的是，1997 年 Netscape 公司内部关于模块设计的争论来自于渴望回归到以小组的形式开发。没有清晰而简单的接口，就不可能把工作分成几个独立的部分。

Netscape 用了 2 个月来重新架构浏览器，但是这还不够长。因此他们决定使用 Communicator6.0 重新开始起草。但是 6.0 还没有结束，它的开发者重新分配到 4.0。Mozilla5.0 版本作为开源发布，但是这也毫无帮助：没有人想在类似意大利细面条似的代码上进行工作。

最后，Microsoft 赢得了浏览器的战争，AOL 收购了 Netscape。当然，这不是 Microsoft

浏览器战胜 Netscape 浏览器的全部故事。Microsoft 的商业实战没有帮助 Netscape。平台的独立性是一个从一开始就显得特别重要的问题。Netscape 从 1.0 开始就运行在 Windows、Mac 和 Unix 上，Netscape 努力维护平台与代码的独立性。他们甚至计划使用纯 Java 版本（Javagator），建立一系列他们自己的 Java 工具（因为 Sun 公司当时还没有开发这些工具）。但是在 1998 年他们放弃了。而且，Communicator4.0 包含大约一百二十万行的 Java 代码。

我从一本关于 Netscape 及其商业和技术策略的优秀书籍中引用了这段文字。你们可以在这里阅读整个故事：

Michael A. Cusumano and David B.Yoffie. *Competing on Internet Time:Lessions from Netscape and its Battle with Microsoft*, Free Press, 1998. 特别是第 4 章，设计决策。

顺便注意：Netscape 花了 2 年多的时间来发现设计的重要性。如果你们没有完全被说服也不要奇怪，有些东西是通过经验来获得的。

1.5 建议

课程学习策略

- 不要落后：步调要快！
- 参加每次讲座：教材没有包含所有的资料。
- 提前思考：不要急于编码。
- 设计，不是调试。

不能把重点放在提前开始或提前思考。当然，我也不期望你们在拿出来时就解决所有的问题。但是如果你们在工作早期就开始的话，可以在长期的过程中给自己节省很多时间，你们将获得更好的结果。首先，你们将得益于逝去的时间：你们将潜意识地琢磨问题。其次，你们将知道你们所需要的额外资源是什么，你们可以在合适的时间很容易地得到它。特别的，充分利用课程人员——我们将在这里提供帮助！我们安排了实验助手聚会时间和教学助手工作时间，但只要不是问题集截至的前夜，你们就可以获得更多的帮助。

简单的说：

“我给出了强烈的警告，关于含糊不清、复杂和过大的新设计，但是我的警告容易被忽视。我总结一下就是有两种构建软件设计的方式：一种是使软件尽量简单从而显然不会有缺陷，另一种是使得软件尽量复杂从而不会有明显的缺陷。”

Tony Hoare, Turing Award Lecture, 1980

讨论 Ada 的设计，不过与程序设计有很大的关系。

如何“使它简单”（KISS）

- 当冰很薄时不要滑冰：避免小聪明式的设计、复杂的算法和数据结构。
- 不要使用很含糊的编程语言特点。
- 对复杂性持有怀疑态度。
- 不要太贪心：慢慢爬行（‘creeping featurism’&‘second system effect’）。
- 记住：事情容易复杂化，但是难于简单化。

最佳化规则

- 不要做。
- 仅对于专家：也不要做。

（选自于 Michael Jackson, *Principles of Program Design*, Academic Press, 1975）

1.6 最后的话

记住：

- 明天是一个讲座而不是复习。
- 今晚完成在线登记表。
- 从现在开始学习 Java。
- 练习 1 下周二交。

查看：

- http://www.170systems.com/about/our_name.html