
# Google公司的软件工程之道（Software Engineering at Google）

洪玫 教授

四川大学软件学院

E-Mail：hongmei@scu.edu.cn

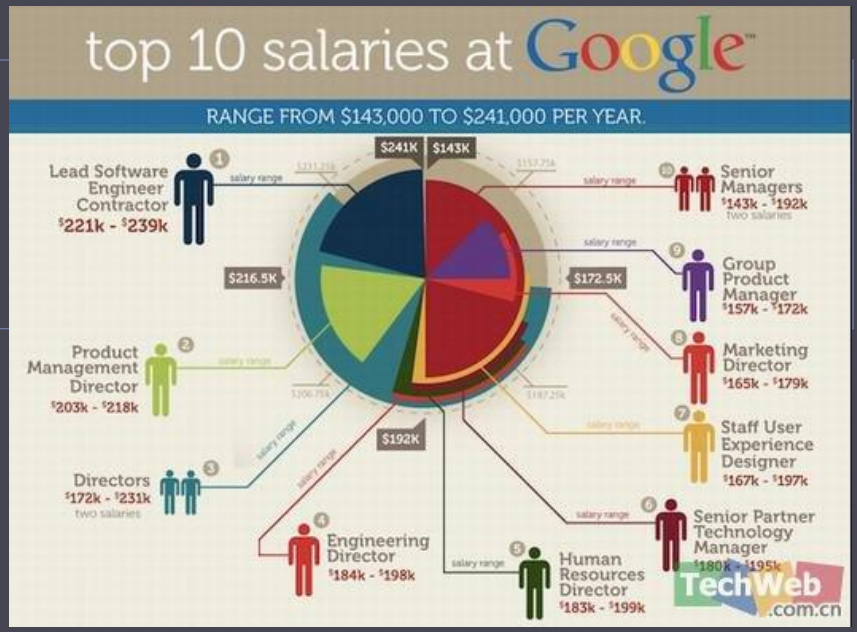
# Google公司的成就

- 美国Google（谷歌）公司是一个非常成功的互联网企业，向我们提供许多优秀的产品:
  - 谷歌搜索；
  - AdWords；
  - 谷歌地图、翻译、语音识别；
  - Chrome；
  - Android
  - 阿尔法狗；
  - YouTube；
  - 自动驾驶汽车。


- 谷歌的成功有很多原因：开明的领导力、优秀的人才、招人高门槛、以及良好的财务实力。
- 另外一个不可忽视的原因是**谷歌不断积累优秀的软件工程实践经验**。

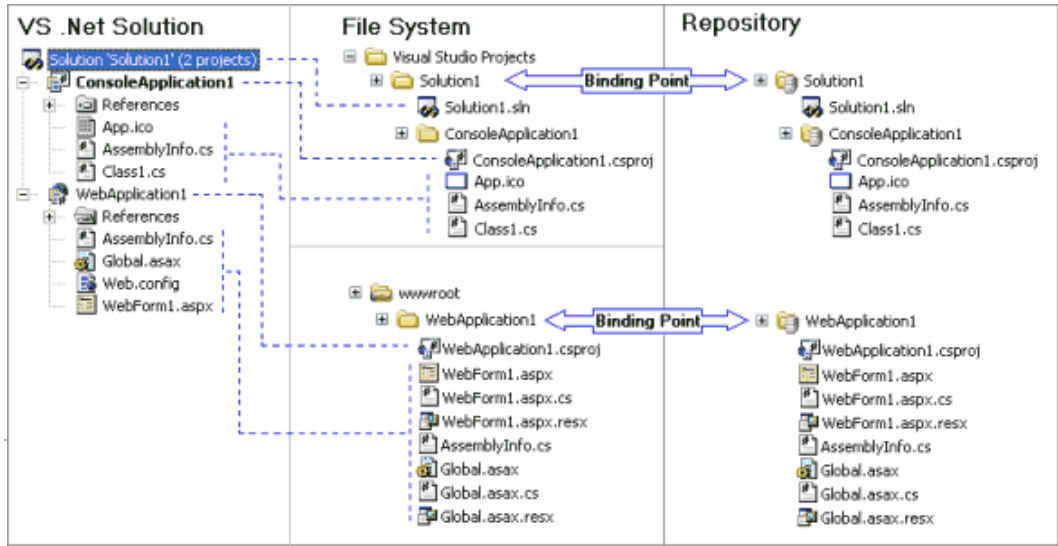
# Google软件开发之道

1. 源代码管理
2. 构建系统
3. 代码评审
4. 软件测试
5. 缺陷跟踪
6. 编程语言
7. 调试和分析工具
8. 发布工程
9. 产品启动审批
10. 事后分析报告
11. 频繁重写



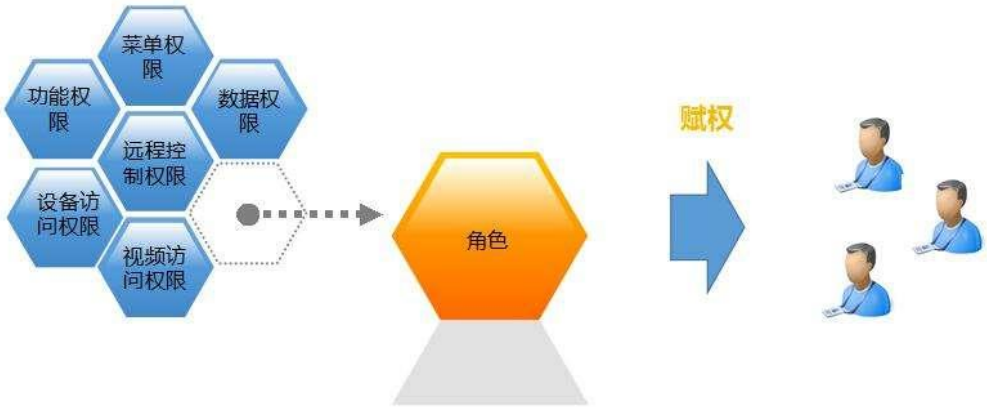
# 1. 源代码管理（The Source Repository）

- 用统一的源代码库（repository）管理代码，允许Google的软件工程师访问：
  - 大型开源项目Chrome和Android使用了独立的开源代码库；
  - 一些高价值的或高安全性的代码，设定了严格的读取权限；
  - 大多数的程序共享同一个代码库。
- Google代码库容量：
  - 86TB的代码库，存有10亿个文件，900多万个源代码文件；
  - 代码行数高达20亿；
  - 3500万代码提交（commits / check in）记录；
  - 每天4万次提交。



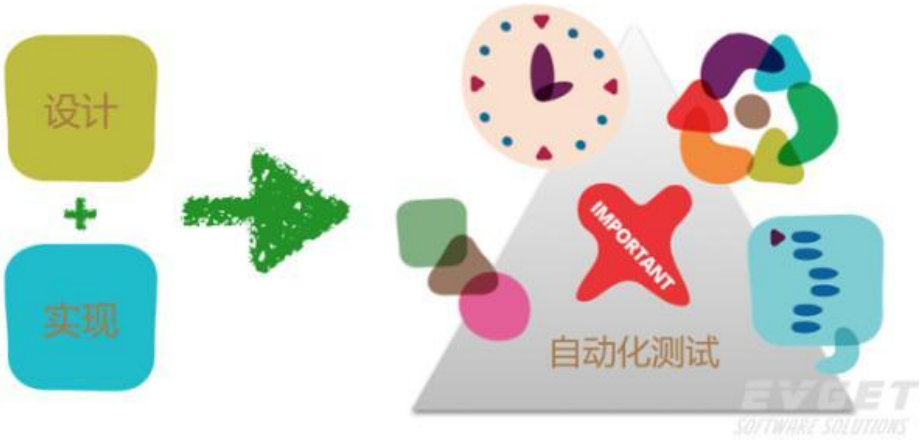
# 对代码库的读写权限限定

- 只有所有者有权批准对subtree的修改；
- 任何工程师都可以访问任何代码:
  - 可以检出（check out）所需代码，进行构建;
  - 可以在本地对其修改、测试，经代码的所有者复审 / 评审（review）后，提交（check in）修改的代码；
- 所有的开发均在代码库的“head”上, 而不是在subtree上
  - 有助于尽早识别出集成问题，并最小化所需的合并工作量；
  - 使得安全补丁能够更容易和更快的上线。



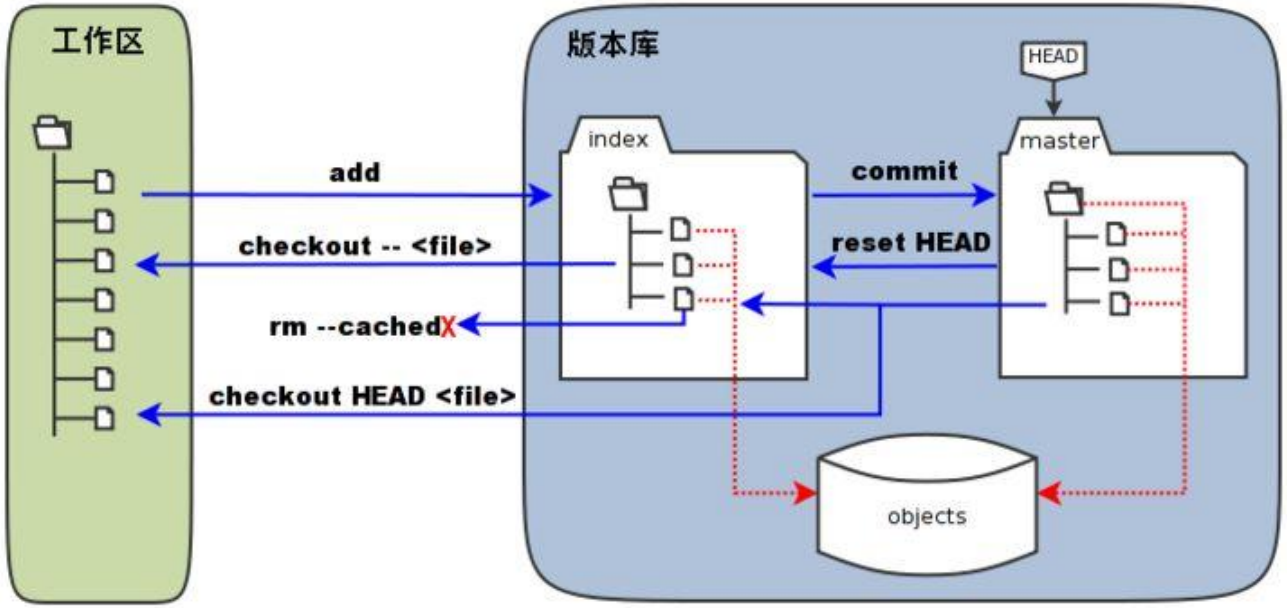
# 自动化系统频繁地运行测试

- 当代码有改动时，进行传递依赖项的测试（回归测试）
  - 如果测试没通过，系统在几分钟内自动通知作者及其评审人员；
  - 通过可视化的构建软件，直观呈现系统构建的状态；
  - “构建警察“可快速回滚违规的修改，以确保测试在head上的持续通过；
- 对于规模非常大的团队，关注持续成功构建，对“head”的开发具有实践价值。



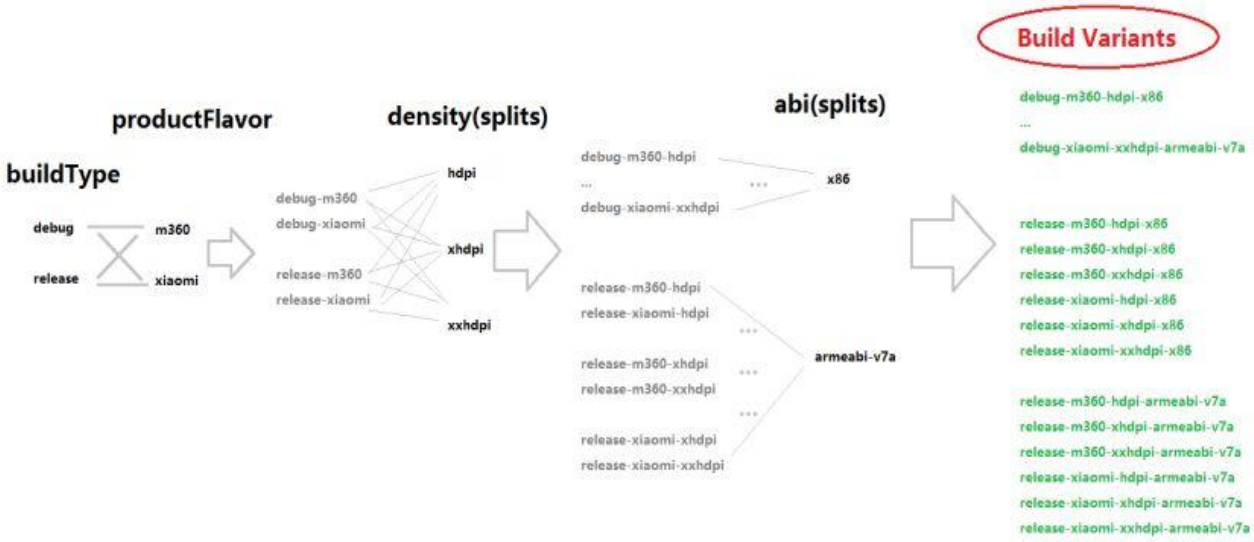
# 代码所有权

- 每个subtree都可以有一个列出subtree “所有者”用户id的文件；
- 子目录可以有选择性地从父目录继承所有者；
- 每个subtree至少需要两个以上的所有者,尤其是针对分布在不同地方的团队；
- 有时也可以在所有者文件中列出整个团队成员。

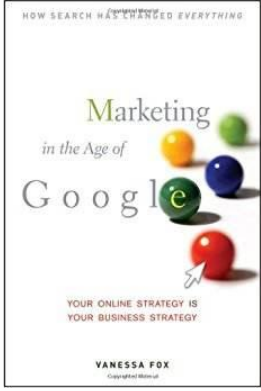


# 2. 构建系统（The Build System）

- 谷歌使用Blaze分布式构建系统, 进行软件的编译、链接和测试：
  - Blaze提供构建和测试软件的标准命令, 适用于整个代码库；
  - Google工程师构建版本和测试软件，**简单而快速**；
- 程序员为Blaze编写版本构建的“BUILD”文件；
- 构建系统使用谷歌的分布式计算设施，构建工作分布在成百上千台机器上，穿插进行，使**快速构建超大型程序**、**并行运行成千上万的测试**成为可能。



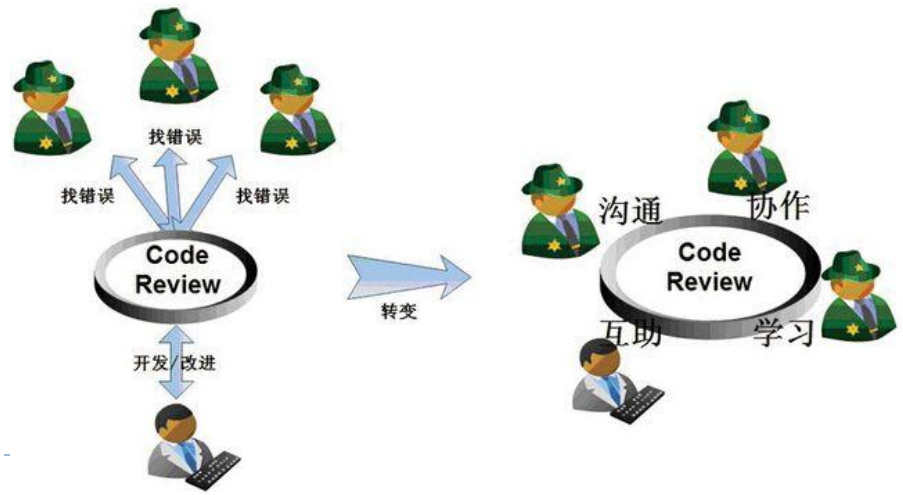
# 系统构建（Build）的策略




- 个人构建步骤是确定的，构建系统可以缓存构建结果，且可以在不同的用户之间安全地共享。
- 构建系统是可靠的，构建系统可以跟踪构建规则本身的变更
  - 即使某操作导致规则变化，依旧可以重新构建目标系统；
  - 能够妥善处理构建的中断或在构建中修改源文件等，只需要重新运行构建命令，不需要运行“make clean”；
- 构建结果缓存在“云中”，可以自动复用，而不是重建。
- 快速的增量构建，可以仅针对上次构建之后修改过的文件进行增量分析，完成新的构建。
- 预提交检查，在启动代码审查、准备提交变更时，谷歌有工具自动运行一组测试：
  - 每个subtree都可以包含一个配置文件，包括要运行的测试、是在代码评审时进行测试、还是在提交之前运行测试、还是两者都需要等信息；
  - 测试可以是同步的，例如，在发送变更给审查之前运行，或在提交变更到代码库中之前运行；测试也可以是异步的，即通过邮件将结果发送到评审讨论线程。

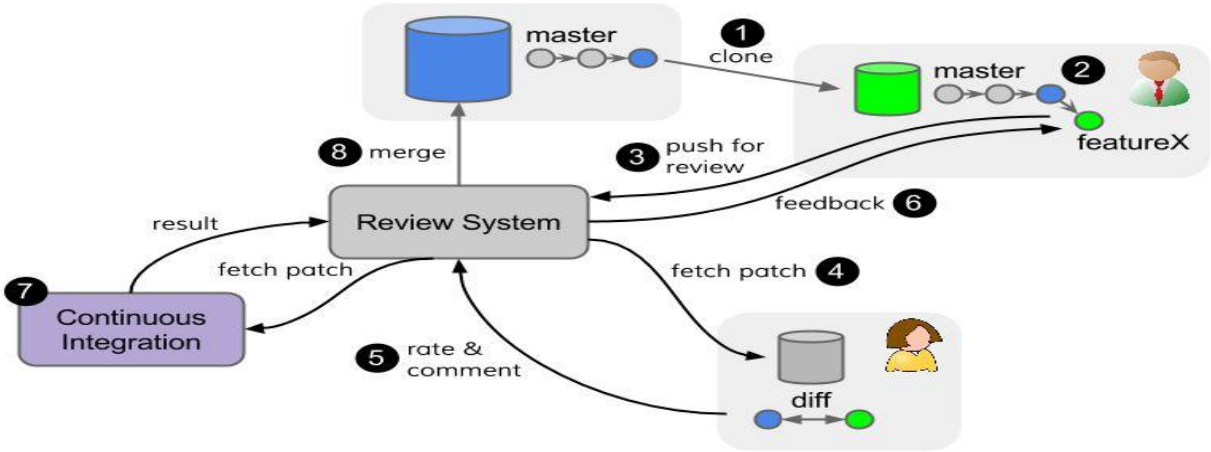
# 3.代码评审（Code Review）

- 谷歌开发了优秀的基于Web的代码评审工具，并和邮件系统集成，允许作者发起评审请求、允许评审人员浏览代码差异 ，并能直接在工具上写评审意见。
  - 当代码变更的作者发起代码评审时，评审人员会收到电子邮件通知，并附有指向web评审页面的链接；
  - 当评审人员提交他们的评审意见时，也会发送邮件通知作者；
  - 自动化工具可以发送自动化测试、静态分析工具的测试结果；
- 所有源代码主库的变更，都必须经过除作者外，至少一个以上工程师的审查。
  - 如果作者不是该修改文件的所有者，那么此项变更还需要至少一个所有者的审查和批准。
- 在特殊情况下，一个库subtree的所有者，可以在通过审查前就发布该库subtree的一项紧急的变更，但评审人员必须留名。



# 代码评审人员选择

- 对于一个给定的变更，谷歌通过工具来自动推荐出合适的评审人员。
  - 工具通过查看发生变更的代码的所有权和作者、近期评审人员的历史、每个潜在的评论人员的待审查的代码数量等信息来确定；
  - 每个库subtree都至少有一个所有者审查和批准该项变更；
  - 作者可以自由选择他们认为合适的评审人员。
- 代码评审中一个潜在的问题是：
  - 如果评审人员相应过慢或过于勉强而不批准变更，将降低开发速度；
  - 允许工程师绕开那些评审任务过重的评审人员；
  - 把较为简单的变更发送给经验不丰富的评审人员；
  - 把相对复杂的变更发送给有经验的评审人员，或同时发给多个评审人员。



# 代码评审策略

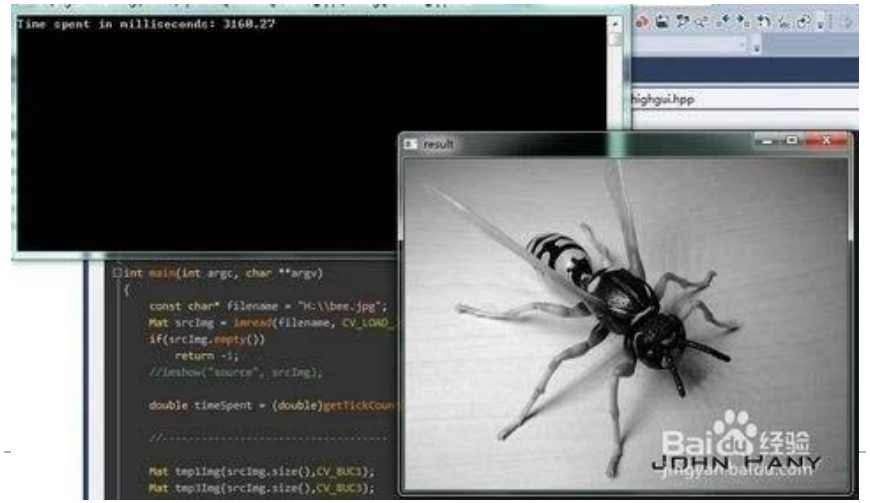
- 每个项目的代码评审讨论会被自动拷贝到指定的项目维护者邮件列表中。
  - 任何人都可以对任何变更进行自由评论，无论他们是否被提名为此变更的评审人员，同时也不管这条变更是否已经通过了审核；
  - 如果发现一个错误，通常会追踪到引入此项错误的变更和原代码的评审意见，并指出问题在哪里，以便让原始作者和评审人员都了解问题所在；
  - 也可以同时发送代码审查给多个评审人员，只要有一个评审人员批准，就提交这项变更，随后的评审意见将在其后，依次加入变更序列中。
- 鼓励工程师每次尽量保持**小规模的变更**，大的变更也最好是分解成一系列的小变更，以便评审人员很容易地一次性地完成评审。
  - 借助代码审核工具给每次变更的规模打上一个文字描述的标签：
    - 对于30～99行的添加/删除/等类型的变更，标注为 “medium-size” ；
    - 对于300行以上的变更，根据其程度进行标注，如(300～999)为 “large” ，(1000 ～1999)为“freak in huge” 。

# 4 软件测试（Testing）

- **单元测试**是Google公司非常提倡和广泛采用的工程实践。
  - 产品线上所有的代码都要求进行单元测试，如果新添加的源文件没有进行相应的测试，代码审核工具将会将它们突显出来。
- 代码评审者通常要求：
  - 任何增加了新功能的变更都应该**添加新的测试**来覆盖这项新功能；
  - 采用Mocking框架，允许对于重量级库函数有依赖的代码进行轻量级单元测试的构建；
- **集成测试和回归测试**也得到了广泛的应用。
- 测试被视作代码审核和提交流程中的一部分。
- Google也有自动化工具来度量测试覆盖率，其结果作为可选层（An Optional Layer），与源代码浏览器集成在一起。
- 在Google，部署之前的**压力测试**也是必备礼仪（derigueur），要求团队通过图表显示延迟和错误率等结果。

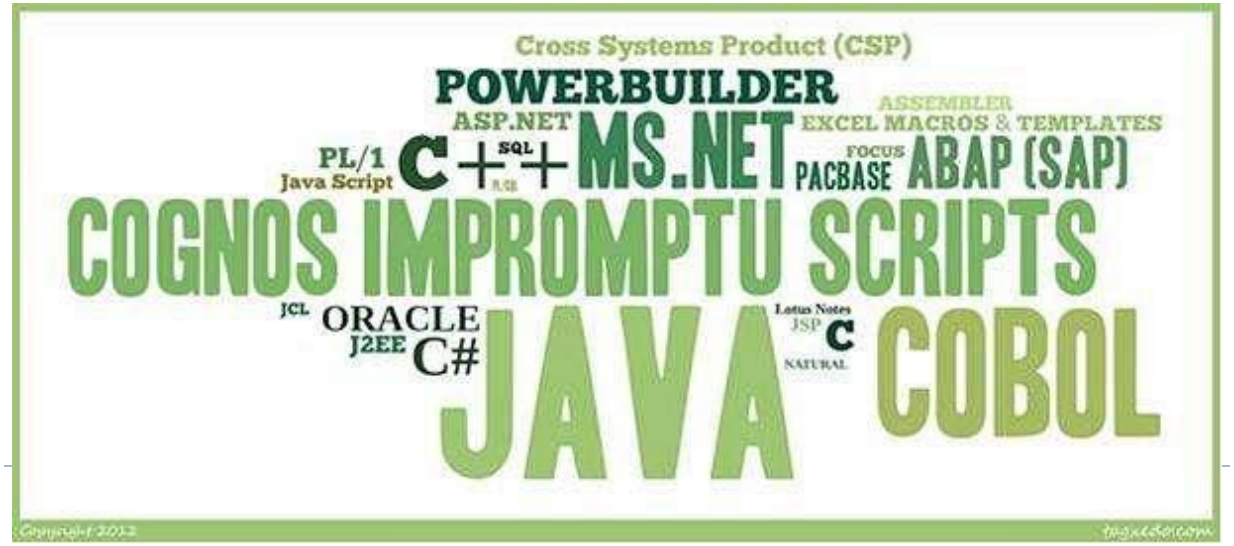
# 5 缺陷跟踪（Bug Tracking）

- 谷歌使用一个名为Buganizer的缺陷跟踪系统:
  - 跟踪Bug、特性请求、客户问题、产品发布或缺陷清理流程等；
  - Bug按层次化的组件 / 模块进行分类，每一组件会有一个默认的接受人和邮件抄送列表；
  - 当发送变更的源代码供审核，系统会自动提示工程师，将变更与特定问题联系起来；
- 定期扫描各个组件中未关闭的Bug（未修复的），优先关注这些Bug，并合理地将它们分配给相应的调试工程师。
- 有些团队会有一个特定的人员来负责Bug分配，而有些团队则在定期团队会议上进行Bug分配。
- 依据Bug上的标识来判断Bug是否已经被分配、Bug需要在哪个版本发布前被修复。

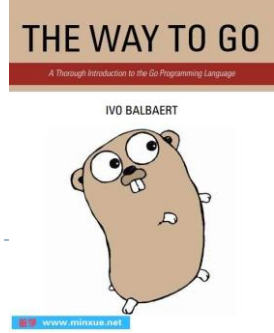


# 6 编程语言（Programming Languages）

- Google鼓励软件工程师使用官方认可的四种编程语言：**C++、Java、Python、 Go**中的一种进行编程。
- **不同的编程语言使用数量达到最低**，从而减少代码复用和程序员合作上的障碍。
- Google为每种语言制定了**编程规范**，以确保整个公司的代码编写使用相似的风格、布局、命名约定等。
- **代码可读性培训**：
  - 让在意代码可读性，且有经验的工程师培训其他的工程师；
  - 针对特定的编程语言，通过审查变更来训练编程人员写出可读性强、符合惯例的代码；
  - 在特定语言增加了任何有意义的、新代码的变更时，需要有一个该项语言可读性培训合格的人的批准。



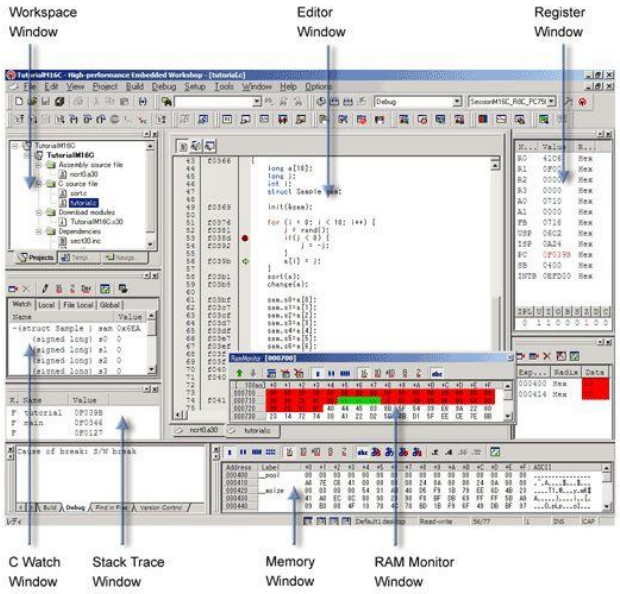
# 6 编程语言（Programming Languages）




- 除了上述四种编程语言，也有许多专业的领域特定语言（Domain-Specific Languages，DSL），用于特殊用途。
- 不同的编程语言之间的互操作，通过使用协议缓冲（Protocol Buffers）来实现：
  - 协议缓冲是对结构化数据编码的一种有效的、可扩展的方法；
  - 协议缓冲包括一种用于说明结构化数据的DSL，一个可以解析这样的描述和生成C++、Java、Python代码的编译器；
  - Google版本的协议缓冲是与Google的RPC（Remote Procedure Call）库相集成的。
- 过程（Process）的共性是在巨大的代码库和多样语言下依然能轻松进行软件开发的关键因素：
  - 无论什么项目或语言都使用一组相同的命令，来执行所有日常的软件工程任务（如代码检出、编辑、构建、测试、评审、提交代码、缺陷报告等）；
  - 不论开发人员正在编辑的代码来自不同的项目、还是用不同的语言编写的，他们都不需要重新学习一种新的开发流程。

# 7 调试和性能分析工具（Debugging and Profiling Tools）

- Google服务器与为正在运行的服务器进行调试的工具库是联系在一起的。
  - 一旦服务器崩溃，一个信号处理程序将堆栈跟踪信息自动存储到日志文件中，并保存其core文件；
  - 如果由于堆内存耗尽，导致服务器崩溃，服务器将保存一个堆对象的样本子集所在的那些站点的堆栈跟踪信息；
  - 提供Web的调试接口，允许检查传入和传出的RPC（包括时间、错误率、速率限制等）、改变命令行标志值、资源消耗、分析等；
- 这些工具大大增加整体调试的容易度，因此也很少启动像GDB那样的传统调试器。

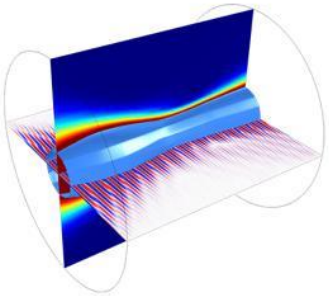


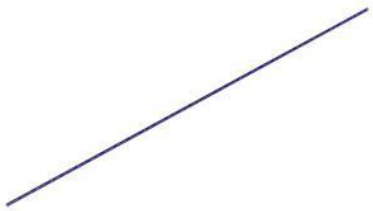
# 8 发布工程（Release Engineering）

- 在Google，只有很少的几个团队有专职的的发布工程师，大多数团队的发布工程的工作是由非专职的软件工程师来完成的。
- 大多数软件发布频繁：
  - 每周或每两周发布一个版本，甚至每日发布；
  - 大部分常规的版本发布实现了自动化；
  - 频繁发布新版本有助于保持工程师的士气；
  - 通过更快的迭代，提高整体的开发效率；
  - 在给定的时间内，有更多的机会获得用户的反馈和对反馈做出响应。

# 软件发布过程

- 版本的发布通常开始于一个崭新的workspace，通过同步最新的“绿色”版本(即最后一个通过所有自动测试的版本)的变更号，并创建一个版本分支。
- 发布工程师可以选择额外的变化作为“择优挑选（cherry-picked）”，即将主分支合并到新版本分支。
- 一旦候选版本已经完成打包，则通常部署到staging（Beta 服务器或准产品线环境）服务器上，由少量用户进一步完成集成测试。
- 最后发布就可以推广到所有数据中心的服务器上了。
- 一些非常高流量、高可靠性的服务是需要几天时间、逐步推出，以减少由于未被之前的检测步骤所发现的、新引入的缺陷而造成任何停机 / 服务中断（outages）的影响。



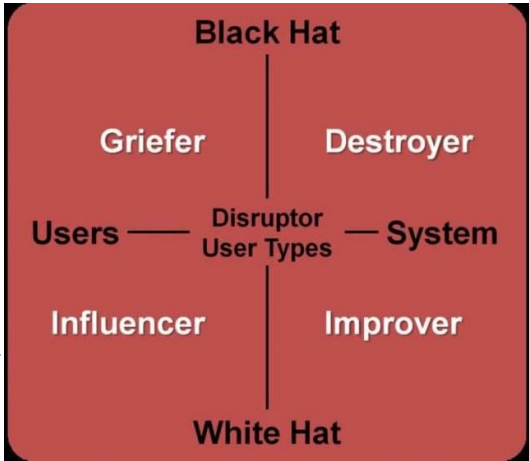


# 9 产品启动审批（Launch Approval）

- 发布任何用户可见的变更或发布重大设计变更时，需要通过实施这项变更的核心工程团队以外的成员的审批。
- 审批必须确保代码的编写符合法律要求、隐私需求、安全需求、可靠性需求、业务需求等。
- 验收过程的设定也是为了确保在发布重大新产品，或者新特性时，能通知到公司内部相关的人。
- Google有一个内部使用的验收审核工具，用于跟踪所需的评审及批准，确保符合每个产品定义的启动流程。
- 这个工具很容易操作、可定制，不同的产品或产品领域按不同的要求进行审核和批准。

# 10 事后分析报告（Post-mortems）

- 每当生产系统有重大的outage（运行中断）或类似的事故发生时，所涉及到的人员都要求写一份事后分析报告：
  - 描述事故，包括标题、摘要、影响、时间表、事故根源、做对 / 错了什么（what worked/what didn’ t）、以及行动计划；
  - **分析重点在于问题本身，以及未来该如何避免再次发生，而不是谁出的问题或问题由谁负责**；
  - 影响部分，试图量化意外事件带来的影响，如停机多少时间、丢失了多少查询和损失了多少营业额等；
  - 时间表部分，可以完整给出从停机开始到诊断、纠正问题的过程；
  - 做对 / 错了什么，总结教训，哪些做法有助于快速检测并解决问题，哪些做错了、采取了哪些具体行动，在将来减少发生此类问题的可能性和/或类似问题的严重性。



# 11 频繁重写（Frequent Rewrites）

- Google大多数的软件每隔几年就会被重写一次，这对Google敏捷性和长期成功起着关键的作用；
- 随着软件环境和其他技术的变化，产品的快速变化成为了一项典型的需求；
- 随着技术或市场的变化，也进一步影响用户的需求、渴望和期望；
- 多年前发布的软件是围绕一组旧的设计要求来设计的，通常不适合当前的需求，并积累大量的复杂性：
  - 重写代码，削减掉所有不必要的复杂性；
  - 重写代码，将知识和归属感传递给新的团队成员；
  - 把更多的精力投入到开发并修复代码中的缺陷；
  - 频繁的重写，鼓励不同项目之间的工程师交流；
  - 频繁的重写，有助于确保代码使用最新的技术和方法。