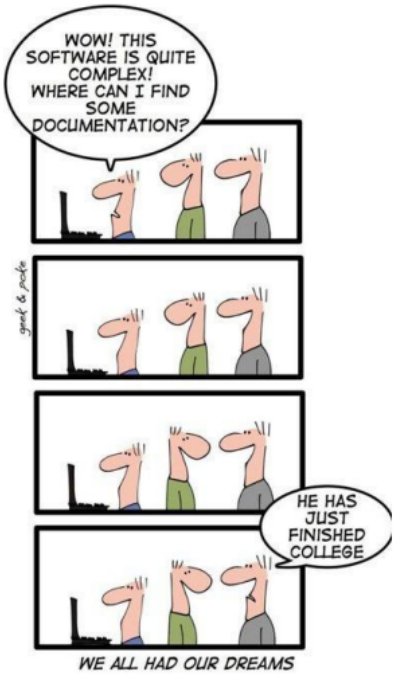
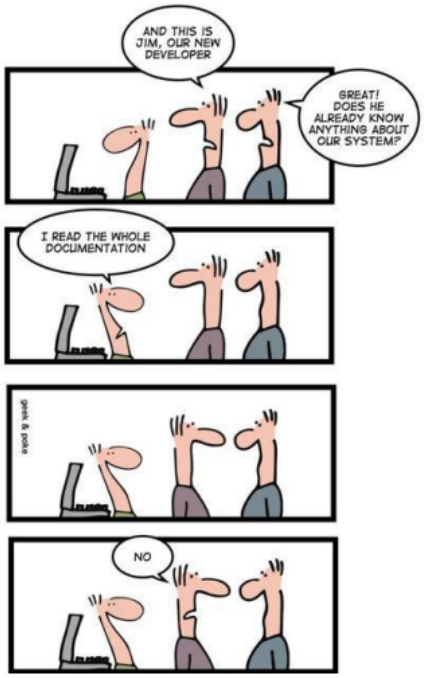
Lecture7 Reverse Engineering

1. 介绍

> 很长一段时间以来，大多数关于软件开发过程的书籍都在讨论当你从编辑器屏幕的空白页开始时应该做什么。这让我感到困惑，因为这不是人们编写代码的最常见的情况。大多数人必须对现有的代码库进行更改，即使是他们自己的代码库。在一个理想的世界里，这个代码库是精心设计和分解的，但是我们都知道理想的世界有多大的概率出现在我们的职业生涯中





逆向工程定义

> “Reverse Engineering is the process of analyzing a subject system to identify the system’s components and their interrelationships and create representations of the system in another form or at a higher level of abstraction."
>
> “逆向工程是分析一个主题系统的过程，以识别系统的组件和它们的相互关系，并以另一种形式或在更高抽象层次上创建系统的表示。”

什么是逆向工程

- 探索工件的设计
  - 从低级到高级
    - 给定二进制，探索源代码
    - 给定代码，探索规范并合理设计
- 外行人：试着了解系统是如何工作的
- 什么时候完成了逆向工程

    - 当你学的足够
        - 改变它
        - 替换它
        - 为它写一本书

为什么要做逆向工程

明确你的目的

- 替换整个系统
- 重新设计，或修改系统
- 使用系统
- 为系统写文档

什么时候要做逆向工程

- 大的方面
    - 很大的改动
    - 技术改变：桌面 -> 网页端 -> 手机端
- 小的方面
    - 添加功能，修改 bug，改善效果
- 在你的课程 Project 中
    - 理解并修改一些大型代码的部分内容

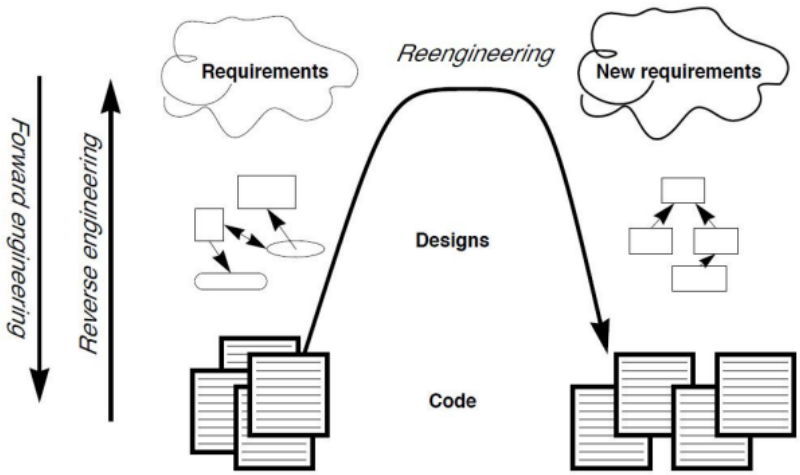
如何进行逆向工程

利用各种/任何资源，逐步构建软件的显式或概念性模型

- 阅读现有的文档
- 阅读源代码
- 运行软件
- 与用户和开发人员沟通
- 执行现有的或编写新的测试用例
- 生成并分析跟踪
- 使用工具生成代码/跟踪的高级视图
- 分析版本历史

等等，你可以使用任何有帮助的东西

一些术语



- 前向工程 Forward Engineering
  - 从需求到设计再到代码
- 逆向工程 Reverse Engineering
  - 从代码到设计，可能到需求
- 再造工程 Reengineering
  - 通过某些设计，从旧的代码到新的代码

你可以分析源代码，运行系统，并访问用户和开发人员以构建遗留系统的模型。然后你必须确定阻碍你进一步进步的障碍是什么，并解决它们，这是再造工程的本质，它寻求将遗留系统转换为你可能已经构建的系统，如果你有足够的后见之明，并且知道你今天知道的所有新需求的话

逆向工程的活动

目的

- 从系统中学习系统

阅读文档

- 可能不准确
- 通常是不够的
- 可能会漏掉什么东西

与其他人交流

- 告诉我它是如何做的?
- 告诉我这是什么意思?
- 谁用它，如何用?
- 什么是输入，什么是输出?

阅读代码

- 找出主要的模块并画出它们的图
- 花不超过几个小时
- 运行测试样例

做出改变

- 解决一个小的问题
- 花不超过几个小时

使用系统

撰写文档

2. 逆向工程模式

模式的格式介绍

标题：如果没坏，就不要修 If it ain't broke, don't fix it.

- 节省你的重构的工作，将你的付出给到系统中那些将产生影响的部分

问题 Problem

- 系统的遗留的哪些部分应该重新设计？
- 这个问题比较难说明，因为
    - 遗留软件系统可能非常庞大和复杂
    - 重写一切既昂贵又有风险
- 然而，解决这个问题是可行的，因为
    - 逆向工程总是由一些具体的目标驱动

解决方案 Solution

只修复“坏了的”那部分，那些不能再适应计划中的变更

权衡 Tradeoffs

- 好处：你不必浪费时间去修复那些不只是你的关键路径的事情。
- 坏处：从长远来看，推迟看似不重要的维修可能会让你付出更多的代价
- 困难：很难确定什么是 “broken” 的部分

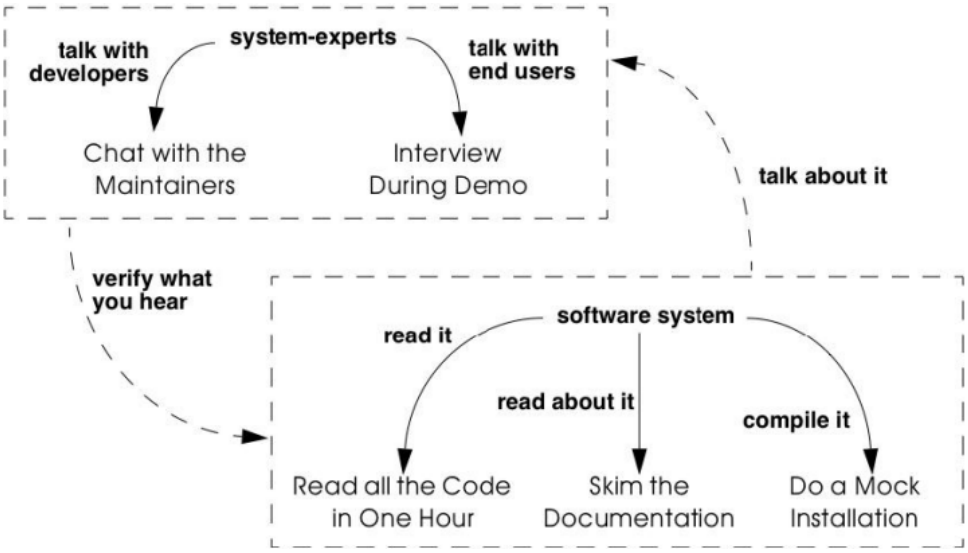
基本原理 Rationale

遗留系统的某些部分可能很难看，但可以很好的 work，那么不需要进行任何重大的维护工作

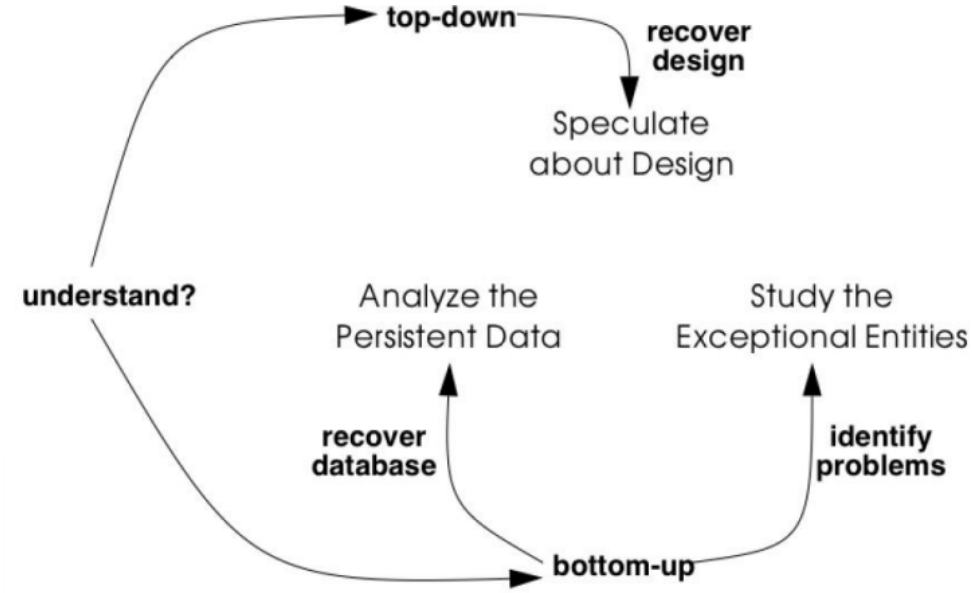
如果可以隔离和包装这些组件，则可能永远不需要替换它们。

设置方向模式 Setting Direction Pattern

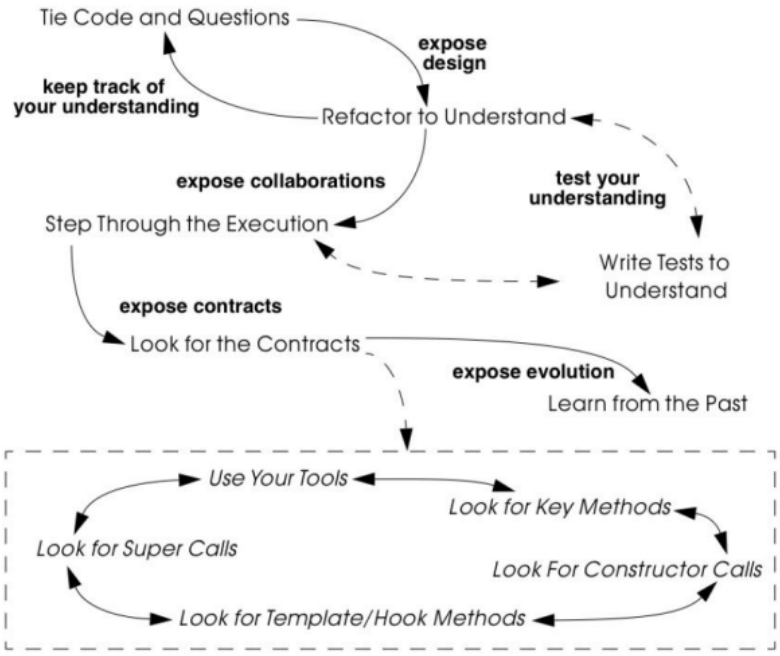
第一次联系模式 First Contact Pattern



最初的理解的模式 Initial Understanding Pattern



具体的模型捕获模式 Detailed Model Capture Pattern



3. 其它的注意

什么才是真正重要的

- 大的概念 —— 分层
- 模块存在的问题
  - 变更的部分
  - 有 Bug 的部分

逆向工程的指标

- 耦合度
  - 上层依赖于下层
  - 较低层不依赖任何东西
- 变更频率
  - 影响一个模块的 git 变更的百分比
- BUG
  - 代码覆盖度

浏览器

- 阅读代码的工具
- 交叉引用
  - 变量、过程、类的定义
  - 变量、过程、类的使用
- 互动，多个视图

调试器

- 查看代码是如何执行的
- 单一步骤
- 断点

处理大小

- 获得概述
- 分解
- 分别检查每一部分
- 通过实验来测试知识
- 正确的概述

通过改变来学习

- 选择容易解决的简单问题
    - 修 Bug
    - 增加一个简单的特性
    - 重构
- 目的是学习，而不是修改
- 在系统的不同的部分选取问题

现有的文档

- 需求
- 结构概述
- 组件的设计
- 接口说明
- 用户手册
- 代码注释

写新的文档

- 描述存在的是什么
- 描述你学到了什么
- 简单而抽象
- 描述
    - 问题
    - 整体的设计
    - 组件
    - 示例

测试

- 测试可以是文档
- 执行测试可以帮助你了解系统如何工作
- 编写测试来学习
- 编写测试来记录你所学到的东西
- 结构不好的系统是很难测试的

不要陷入困境

- 尝试了很多方法，要积极主动
- 要与其他人沟通
- 获得别人的帮助，帮助别人
- 坚持
- 记笔记，否则你可能会忘记