敏捷测试最佳实践

Bill Liu

@billliu_seattle

大纲

- 敏捷测试基础
- 敏捷测试实践
- 敏捷测试误区

课堂讨论

- 列举几个在测试中面临的困难和挑战

我们的目标

- 不是：
  - 预测变化
  - 阻挡变化
- 而是：
  - 适应变化

案例分析

- 使用传统方式测试的一个项目
  - 产品背景
  - 介入时间
  - 测试方式
  - 遇到的困难
  - 最终结果

传统软件测试方法

- 软件开发周期中的最后一阶段
- 计划的测试时间短
- 开发阶段延期，进一步压缩测试时间
- 软件本身难以测试或漏测

传统软件测试人员

- 被动接受需求分析
- 被动等待产品
- 不了解产品如何设计的
- 不了解代码如何写的
- 不了解用户如何使用产品

测试工程师

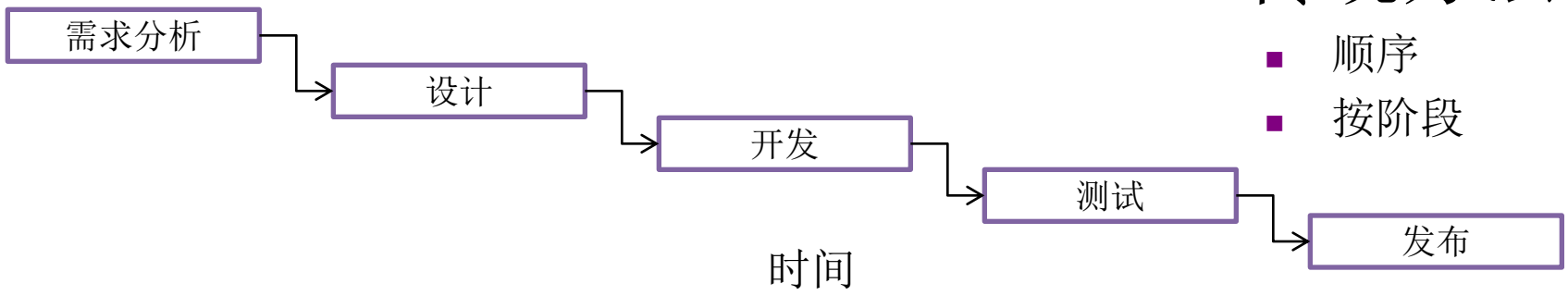
- 为什么受伤的总是我？
- 受伤的不仅仅是测试..
  - 项目延期
  - 超出预算
  - 项目取消

敏捷宣言

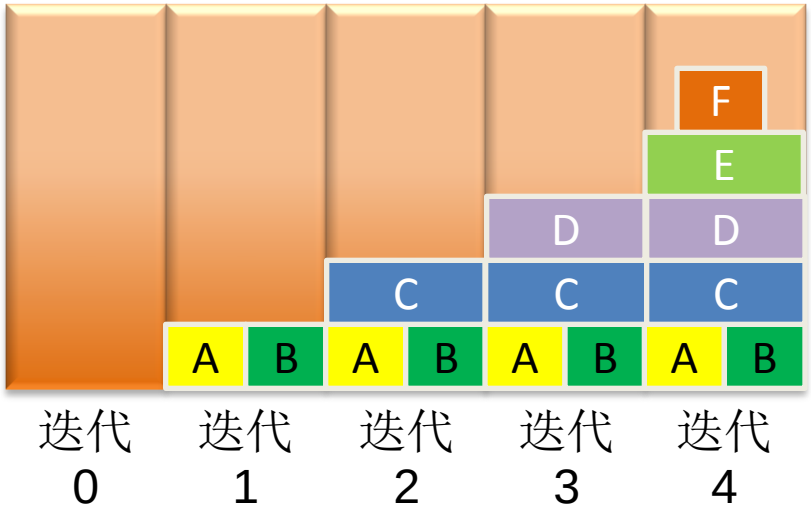
Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right,
we value the items on the left more.

传统 vs 敏捷



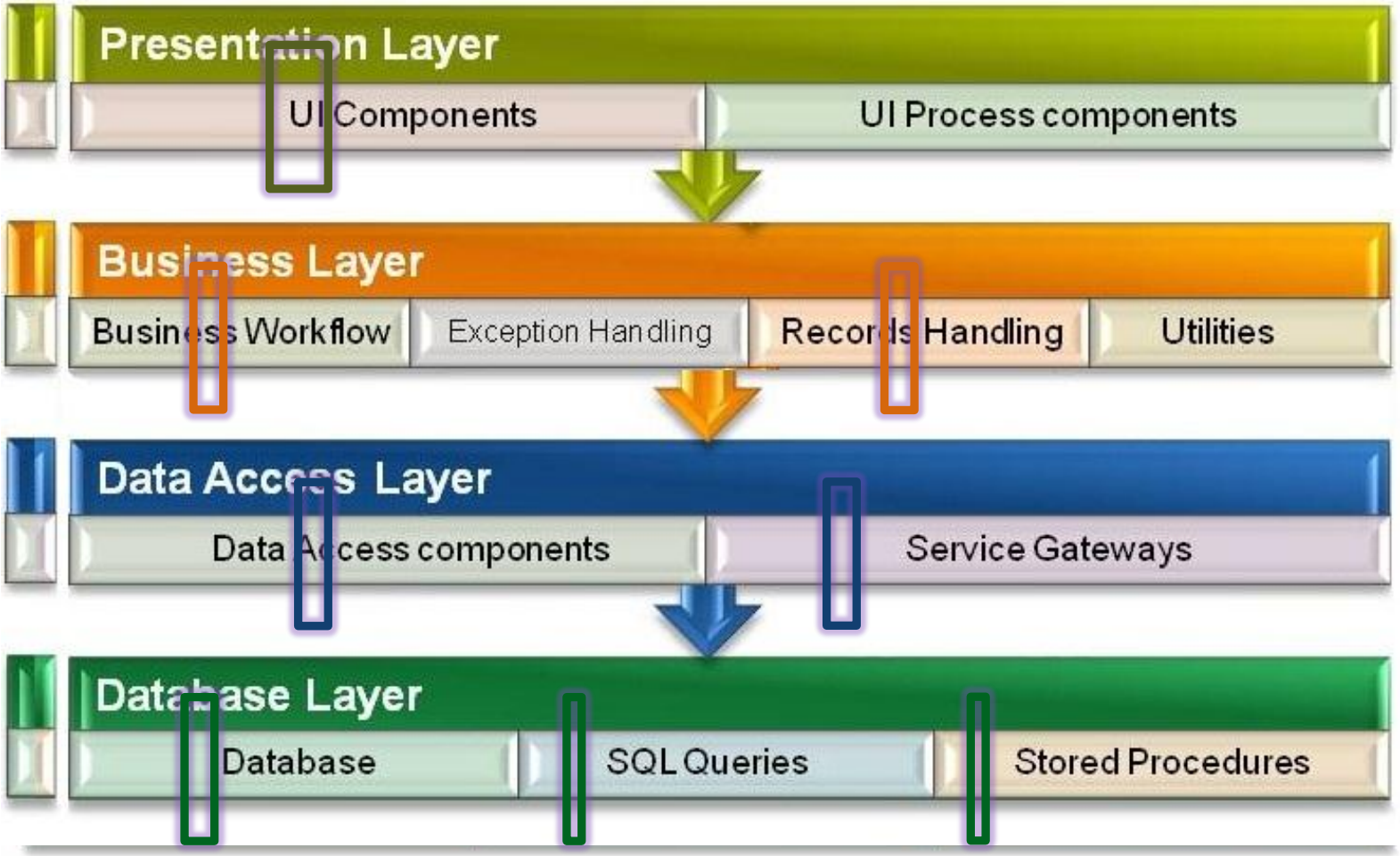
- 传统方法
  - 顺序
  - 按阶段



- 敏捷方法
  - 迭代
  - 渐进

用户需求

不仅仅是迭代....


Presentation Layer
UI Components
UI Process components
Business Layer
Business Workflow
Exception Handling
Records Handling
Utilities
Data Access Layer
Data Access components
Service Gateways
Database Layer
Database
SQL Queries
Stored Procedures

敏捷测试核心理念

- 质量是建立起来的，而不是后期测出来的。
  - 把质量提前
  - 质量是整个项目组的责任
  - 预防而不是探测
  - 缩短质量反馈周期

敏捷测试原则

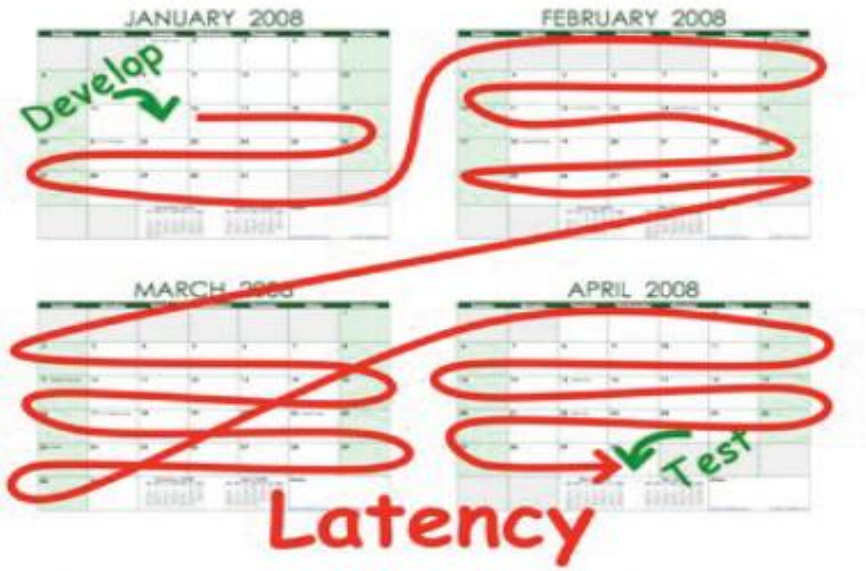
- 尽早测试

上游

下游

敏捷测试原则

- 频繁测试


JANUARY 2008
FEBRUARY 2008
Develop
APRIL 2008
Test
Latency

敏捷测试原则

- 完成 = 代码 + 测试

课堂游戏

案例分析

- 使用敏捷测试的一个项目
  - 项目背景
  - 为什么敏捷
  - 敏捷测试切入点
  - 敏捷测试具体实践

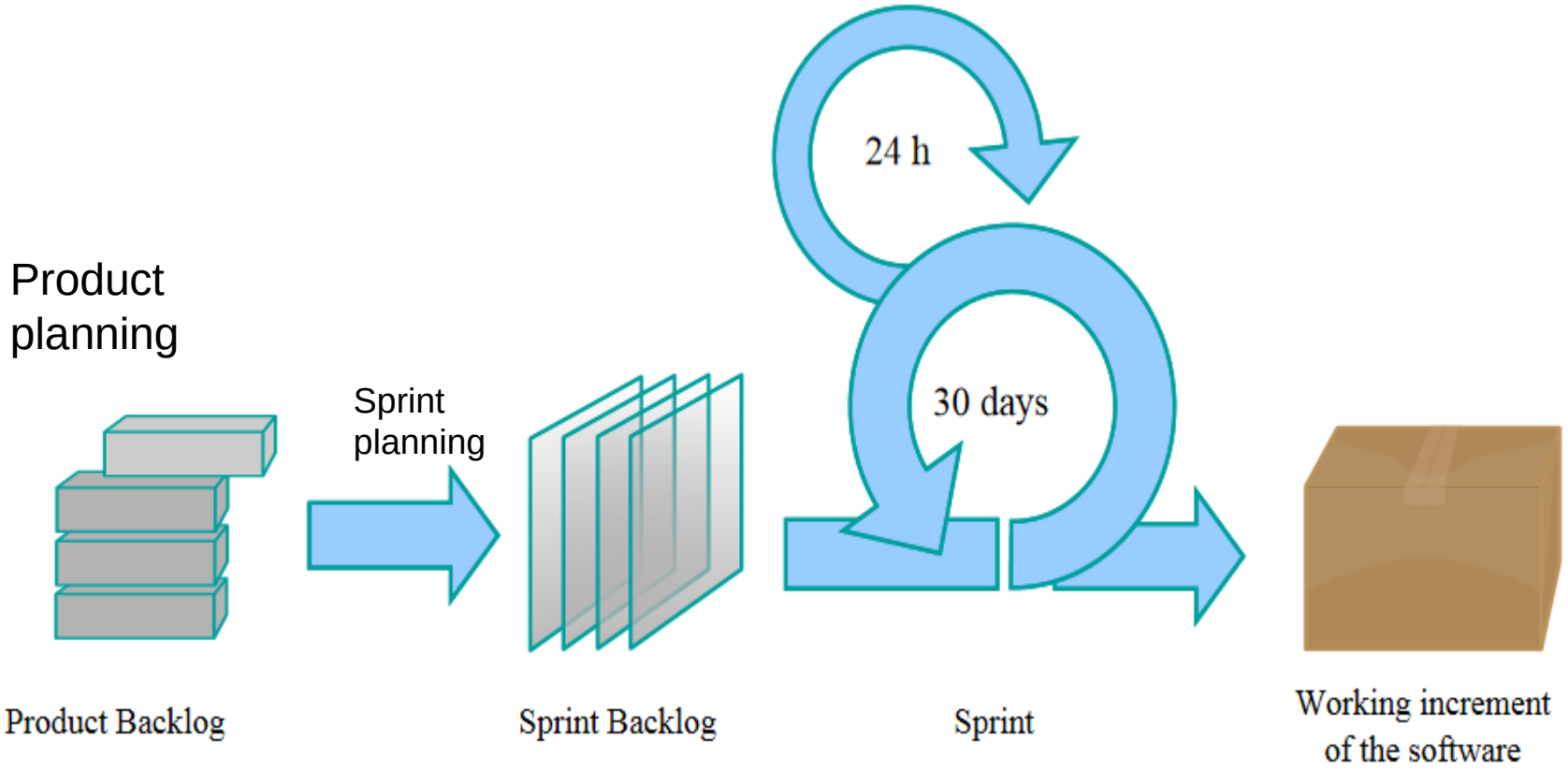
开始敏捷测试

- 学习和调研
- 创建环境
  - 研究需要的工具
  - 搭建基础设施
- 衡量指标
  - 确定目标
  - 衡量成功的指标

开始敏捷测试

- 透明进度
  - 进度报表
  - 交流和沟通
- 用敏捷来指导敏捷实践
- 动手开始

敏捷开发过程（Scrum）



迭代计划阶段

- 学习
  - 相关技术
  - 用户需求
- 讨论和反馈
  - 讨论和反馈用户需求
  - “如果..怎么办？”，“如果..该如何处理？”
- 测试策略分析

测试策略分析

3.3 TESTING BREAKDOWN

Category	Priority	Target Metric	Justification	Delivery Milestone
Unit Testing	High	➢ TBD% CC	Dev owned metric	Code Complete
Functional Testing	High	➢ 60% CC	This is desired and should be achievable especially if we get error cases fully automated with Mocks / Fault Injection	Test Code Complete
E2E Testing	High	➢ 60% CC	As above	Test Code Complete
Perf / Stress Testing	High	As per Functional Spec	Perf KPIs will measure time to call API, time to resolve E2E and usual CPU / Memory etc	Code by Test Code Complete / Runs during stabilisation
Security Testing	High		Validate the caller identity testing	Stabilisation
Integration Testing	High		Validate the NM throughput and the tenant isolation from NM.	Stabilisation
Runner Testing	High	As per Functional Spec		Stabilisation
Component Failure / Failover Testing	High	As per component design	Ensure the system is resilient enough to meet SLA.	Stabilisation
Deployment Testing	High		Ensure our automated DE scripts are utilized as soon as possible. All testing of Create role, Update Role and Update Role config to be tested. Time to deploy needs to be measured.	Test Code Complete
Monitoring Testing	High	As per Functional	Ensure that all critical alarms are triggered in testing and appear in mail. Ensure there are	Stabilisation

迭代执行阶段

- 设计－可测试性
- 执行－持续集成
- 反馈－质量评估

软件的可测试性

- 可测试性 (Testability)
- 是否容易测试
- 影响软件质量的关键
- S.O.C.K原则

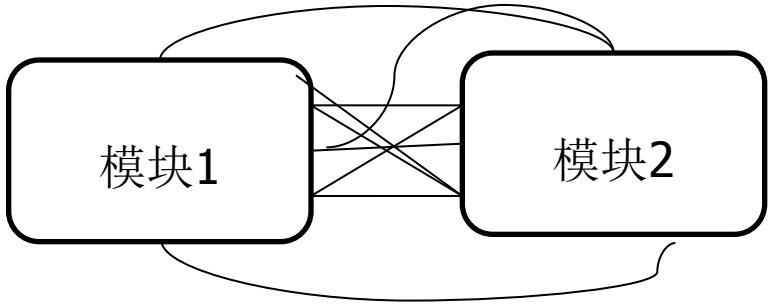
软件的可测试性

- Simplicity
  - 设计简洁
  - 功能模块化
  - 越简单，越容易测试

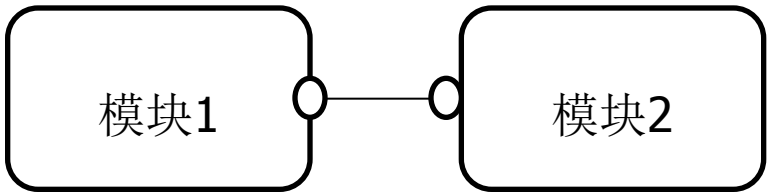
软件的可测试性

设计1

设计2



设计3

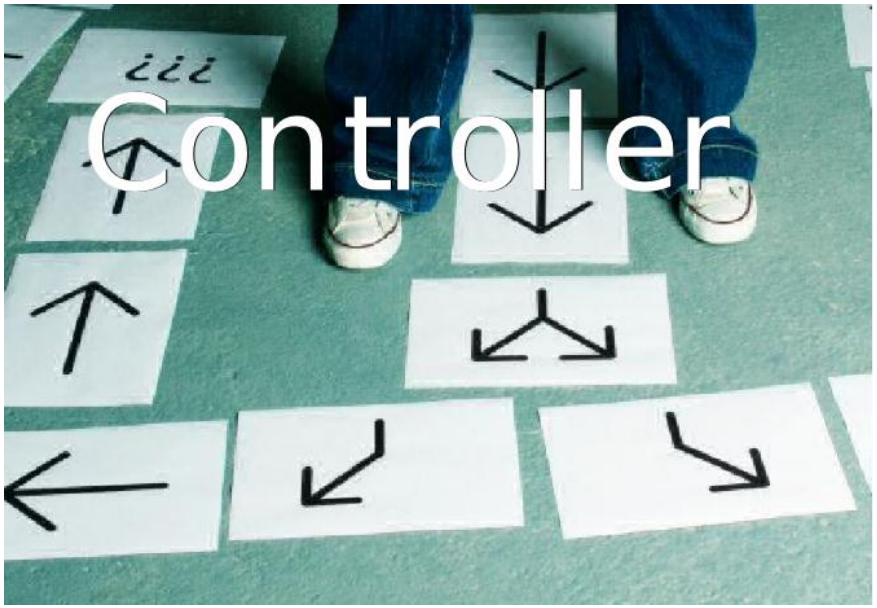


软件的可测试性

- Observability
  - 可观察性
  - 运行时的状态
  - 正确地验证
  - 更容易验证

软件的可测试性

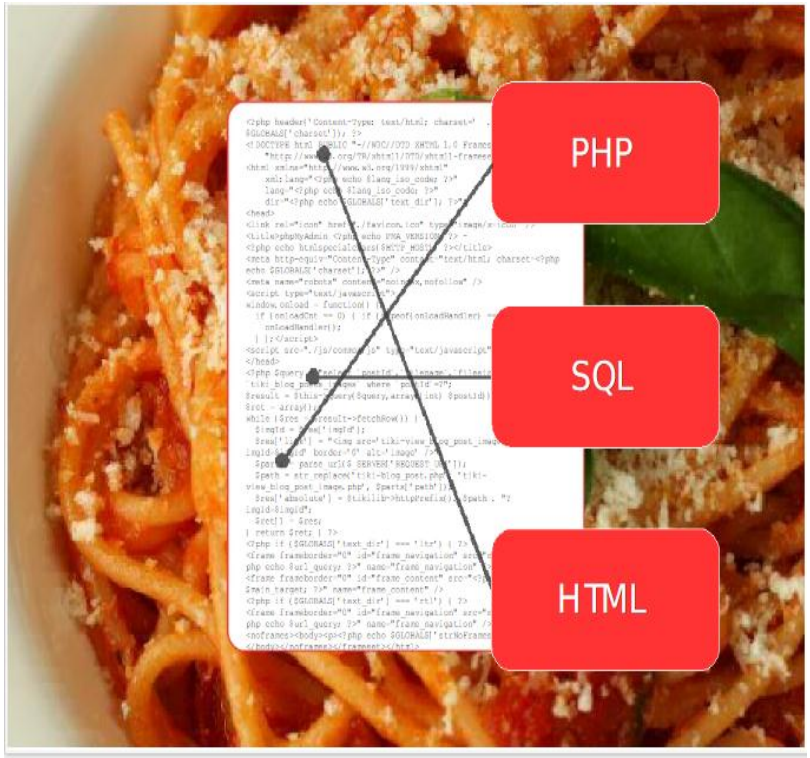
- Control
  - 控制软件的运行
  - 不同分支
  - 出错处理
  - 异常处理



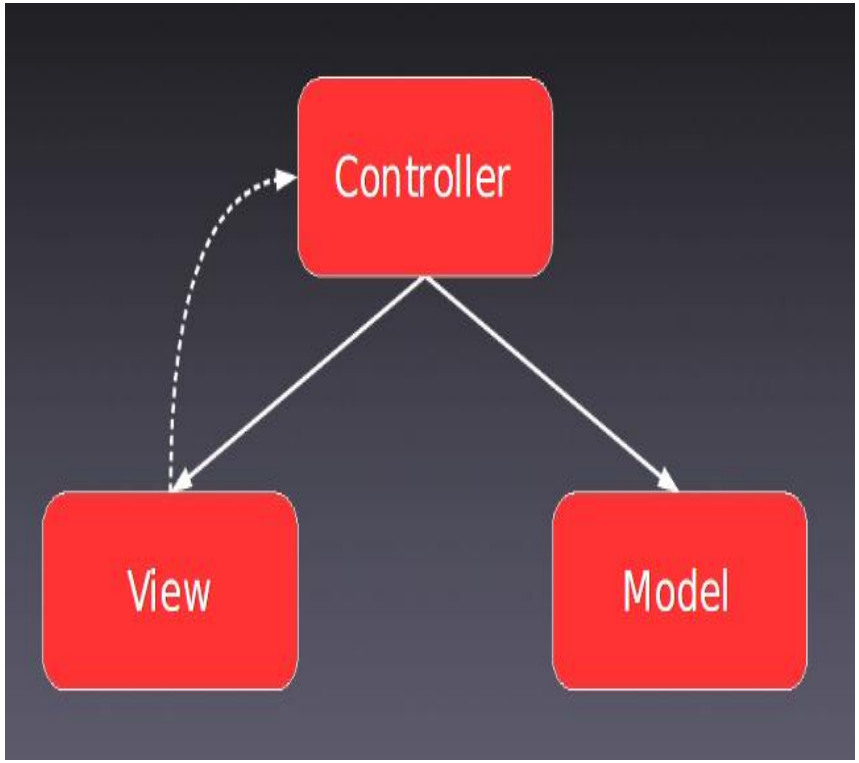
迭代执行阶段—可测试性

- Knowledge
  - 了解模块的各个状态
  - 每个状态有清晰的定义
  - 验证状态
  - 排序的输出和随机的输出

可测试性—MVC例子



Form



MVC

软件的可测试性—讨论

Your peer developer is assigned to implement a program to calculate and output customers' sales tax every 24hours. The customers' data are stored in database. After quickly coding, your peer developer finishes and declares he completes his job, and hands over the following codes to you for testing.

//note the following codes are simplified to demonstrate the goal of this exercise.

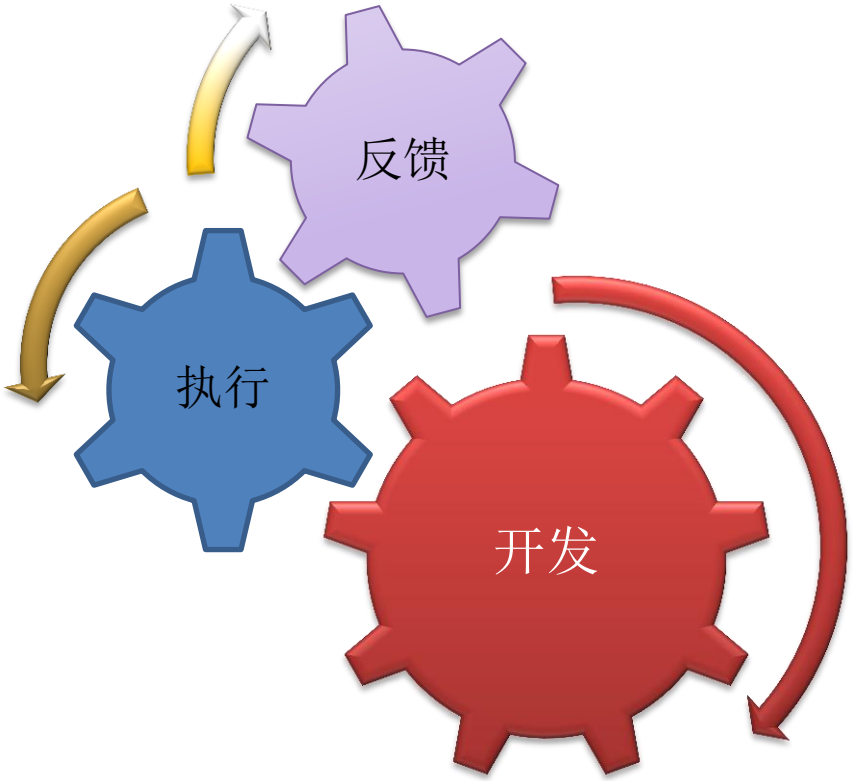
课堂讨论和练习

```
Static void Main()
{
    int lastcalctime=datetime.now;
    Int calcperiod = 24; //24hours
    while (1){
      while (datetime.now - lastcalctime < calcperiod)
      {
        sleep(); //wait for 1hour and check again
      }
     SqlConnection conn = new SqlConnection(SqlDatabaseConn);
      conn.Open();
      SqlCommand command = new SqlCommand(Settings.GetCustomersQuery, conn);
      SqlDataReader reader = command.ExecuteReader();
      while (reader.Read())
      {
         Int tax = calcTax(reader.GetString(0), reader.GetString(1));
         Console.WriteLine(reader.GetString(0)+”—“+tax);
       }
      reader.Close();   conn.Close();
      lastcalctime = datetime.now;
   }
}
```

持续集成

- 敏捷开发的具体实践
- 敏捷思想的充分体现
- 构建CI的基础设施

持续集成实践



持续集成

- 开发——集成
  - 产品代码
  - 测试代码
    - 基础设施
    - 测试工具脚本
    - 自动化测试用例
  - 开发平台工具
  - 版本控制

持续集成

- 执行——自动
  - 每日构建
  - 测试执行
    - 产品部署
    - 运行测试
  - 定时构建或执行
  - 按需构建或执行

持续集成

- 反馈——及时
  - 测试报表
  - 日志收集
    - 产品日志
    - 测试日志
  - 反馈和追踪

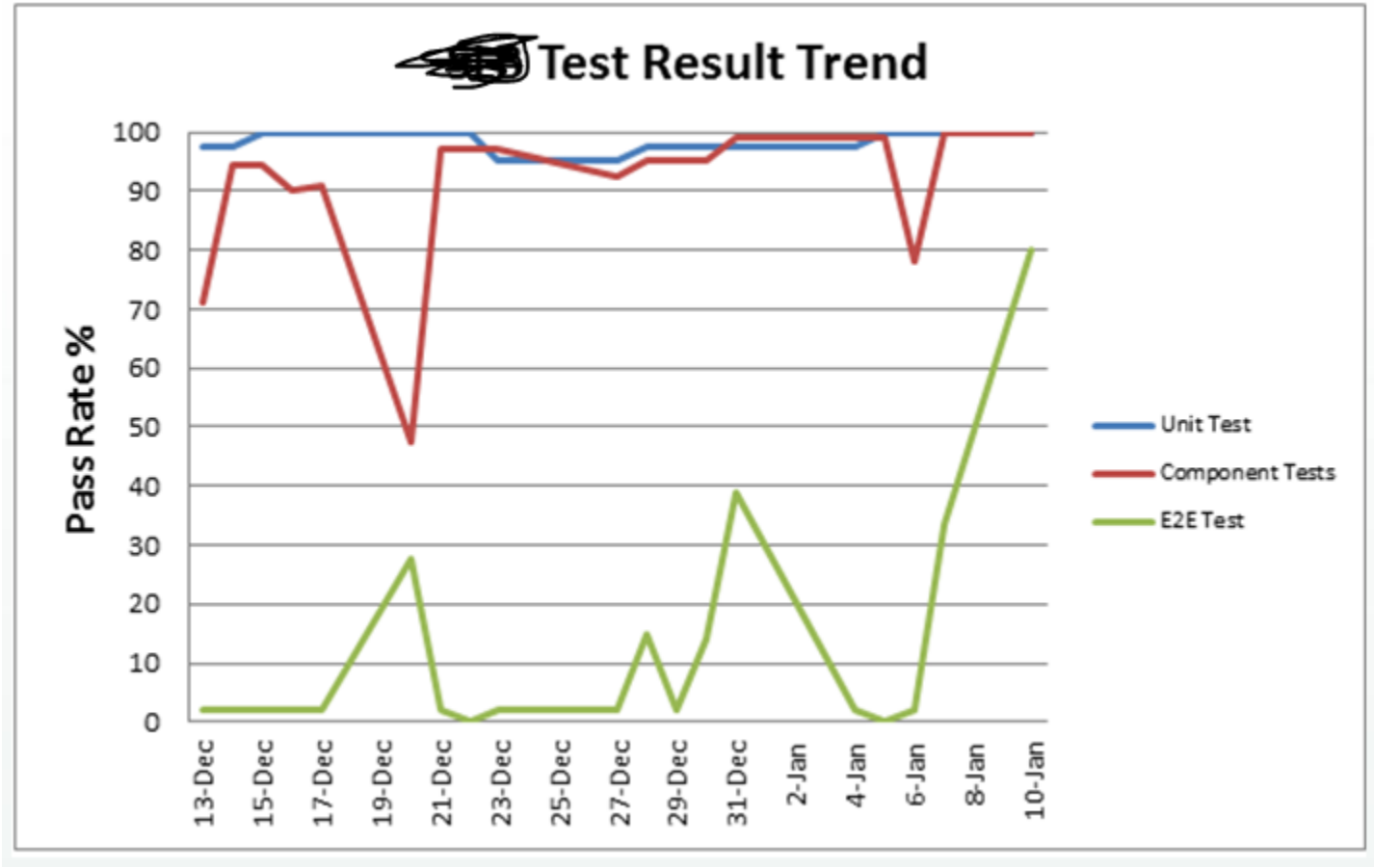
持续集成

- 测试环境管理
- 测试运行配置
  - 新的部署
  - 部署更新
- 测试运行类型
  - BVT
  - SR P0
  - SR P1

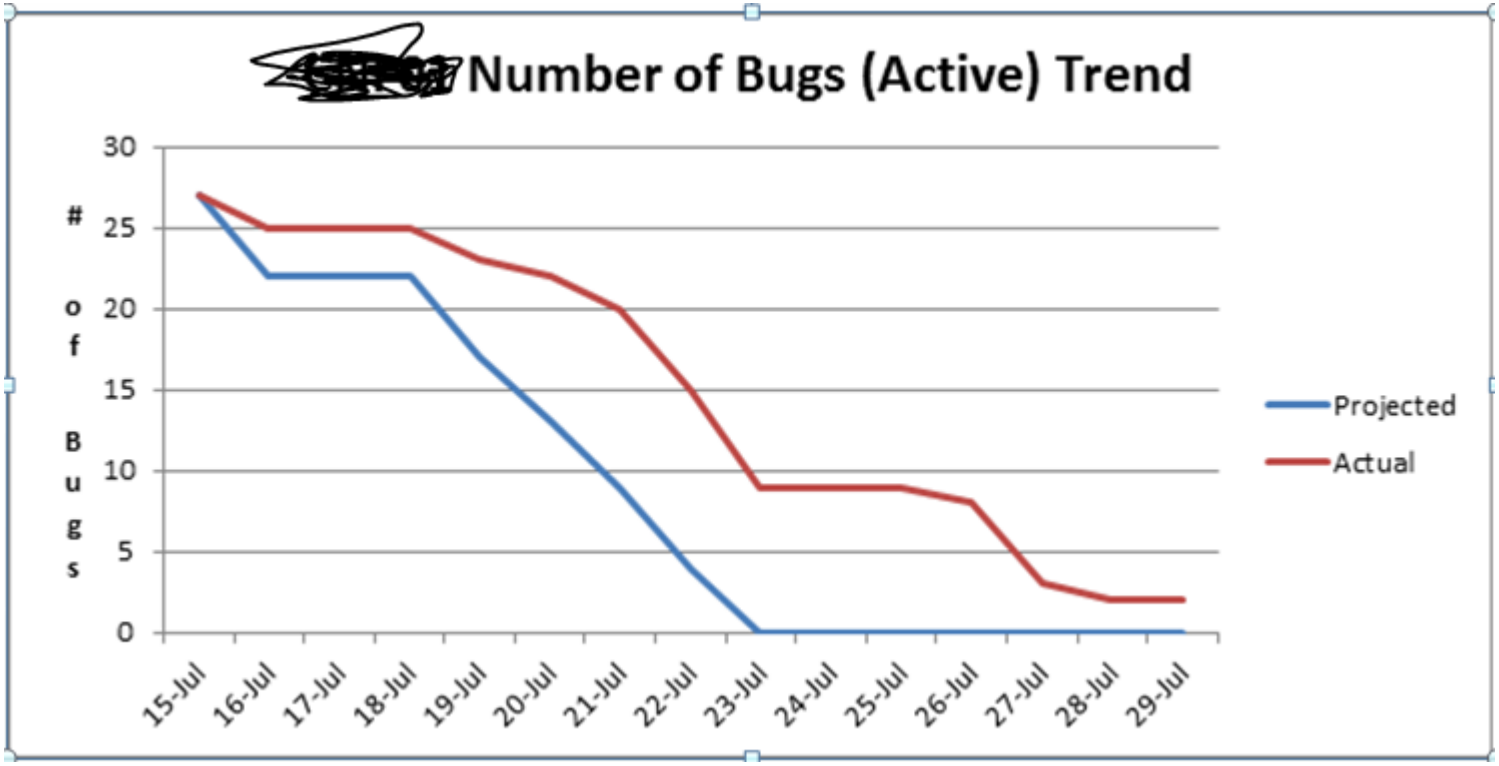
质量评估

- 测试进度
- 每日构建质量－状态、执行和趋势
- 缺陷管理－状态和趋势
- 代码覆盖率
- 测试用例自动化率
- 测试用例分析

质量评估


Test Result Trend
Pass Rate %
100
90
80
70
60
50
40
30
20
10
0
13-Dec
15-Dec
17-Dec
19-Dec
21-Dec
23-Dec
25-Dec
27-Dec
29-Dec
31-Dec
2-Jan
4-Jan
6-Jan
8-Jan
10-Jan
Unit Test
Component Tests
E2E Test

质量评估


Number of Bugs (Active) Trend
# of Bugs
30
25
20
15
10
5
0
15-Jul
16-Jul
17-Jul
18-Jul
19-Jul
20-Jul
21-Jul
22-Jul
23-Jul
24-Jul
25-Jul
26-Jul
27-Jul
28-Jul
29-Jul
Projected
Actual

敏捷测试常见误区

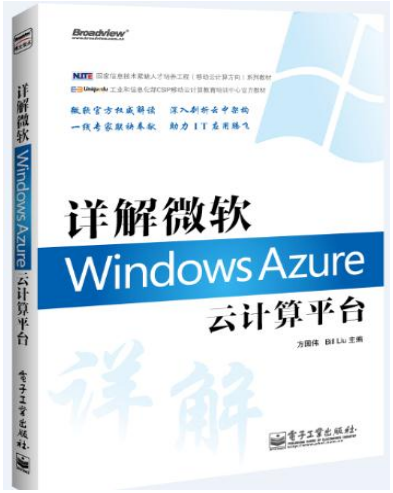
- 整个项目组不统一
- 没有完全理解敏捷方法
- 思想没有转变
  - 开发人员
  - 测试人员
- 期望过高

敏捷测试常见误区

- 过多手工测试
- 没有持续集成的基础设施
- 质量差的测试自动化代码

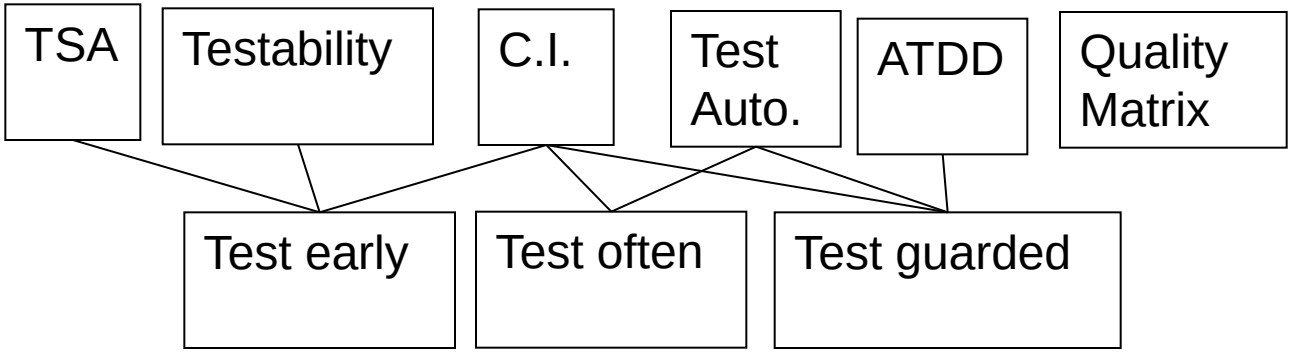
Q&A

- 学习资料：
  - 推特/微博: @billliu_seattle
  - 博客： http://blogs.msdn.com/b/billliu/
  - 主编：《详解微软Windows Azure云计算平台》
  - 咨询和内训：云计算、软件测试



Agile Testing

Tester skills:
- Domain knowledge
- Coding
- Communication
- Proactive



Quality is built in
- Team owns quality
- Drive quality upstream
- Reduce feedback latency
- Defect prevention over detection