Code Diff实施规范V1.0

C-2 创建:王裕坤, 最后修改: 王琪 04-15 16:58

目录

目录

1 背景和目的

1. 在日常的测试过程中Code Diff并未正式纳入提测流程，更多的是RD之间互相进行Code Review，QA参与程度较低
2. 自赤峰项目试点Code Diff，发现很多Bug都可在Code Diff过程中发现，大大提前了Bug发现时间
3. Code Diff可以提升业务逻辑熟悉程度，将业务逻辑实现与需求进行印证。同时也作为传统测试手段和方法的一种补充，更倾向于白盒测试，减少黑盒盲测造成疏漏的可能性，帮助提升测试覆盖率，给早期的bug发现提供了一种高效便捷的手段

2 预期收益

CodeDiff带有强烈的目的性，其所带来的收益包括但不限于以下几个方面：

- **评估影响面**
- **明确/补充checklist的内容**
- **确认需求实现 / 发现需求漏洞**
- **提前发现bug / 提升交付效率**
- **加深对技术实现的理解**
- **确认发布步骤**

为了更好的说明Code Diff的收益，这里简单统计了下三个大小不同的项目的数据情况：

项目名称	测试排期	QA介入日期	提测前用时	提测后用时	测试完成日期	CodeDiff投入	发现Bug占比	备注	测试总结报告
赤峰接入（一期）	12/17-12/27 (9个工作日)	12/15	3D	7D	12/25	1.5PD (占比 15%)	22%	提前2D完成测试	赤峰项目测试总结报告
清远接入一期	01/10-01/16 (5个工作日)	01/08	1.5D	3.5D	01/14	0.5PD (占比 30%)	0%	无	清远线下绑卡0114-测试结束报告
退款功能优化	01/14-01/15 (1个工作日)	01/14	-	0.5D	01/14	1.5H (占比 25%)	67%	无	退款逻辑修改0114-测试结束报告

从上述统计中可发现Code Diff发现的Bug数占比超过20%（由于数据样本较小，更精确的数据还需进一步观察统计），**Bug发现以及交付提升预期：**

部分bug的发现时间提前1-2天（提测前介入），提测后测试周期预计缩短20%~30%（中小型项目，RD人日<7），交付效率预期提升20%左右

3 规范细则

为了更好的保证CodeDiff的事实，分别对QA和RD在此阶段所做工作进行一些规范。

3.1 对RD要求

1. 提供详细的技术实现方案，关键技术实现细节尽可能的细化
2. 明确本次提测的测试范围，以及需要重点进行覆盖的逻辑或者场景
3. 在进行交互逻辑说明时，尽量使用交互图、流程图等形式，同时添加必要的文字说明，示例（2018-11-25 赤峰开户需求技术文档）
4. 对于数据库表的修改（新增table、变更table、变动库）需要额外的详细说明，示例（2018-11-25_done 交易适配赤峰改造技术文档）
5. 在具体的业务流程处理上，采用Step By Step的方式进行细化，方便QA同学尽快的理清实现逻辑，示例（赤峰消费模块技术文档）

6. 对于外部组件的使用，包括但不限于MQ、Redis（Squirrl）、Leaf、OCTO等也需要进行标注

3.2 对QA的要求(基础)

1. 根据RD提供的交互流程图，理清代码的基本架构以及需求实现途径以及方法
2. 进行浅层次的代码Review，判断RD在逻辑实现上是否有明显的漏洞
3. 进一步重点检查实现与需求不符的情况，比如产品文档中要求某项功能，但是RD没有开发，或者开发的意图与产品文档明显存有出入
4. 针对具体的实现，进行模块化细分（如果存有多个模块），针对各模块进行详细的codeDiff
5. 针对通过Code Diff发现的bug，在ones上打上相关标签供后续统计分析，示例（https://ones.sankuai.com/ones/product/4944/defect/list?cn=1&filterId=2255）

目录

3.3 对QA的要求(进阶)

1. 熟悉服务之间，模块之间的依赖以及调用关系，了解交互过程中的报文格式，对于关键性字段要仔细留意
2. 熟悉新增或者变动表结构，在自动化工程中在底层封装DB操作库，供后续调用
3. 针对具体服务、业务逻辑实现进行可测性评估，评估的内容包括但不限于一下部分：
   - ☑ 是否需要进行Mock实现（无论接口类型，Http 或者 Thrift；内部还是外部）
   - ☑ 是否需要添加额外的调用桩（比如模拟Crane的定时任务）
   - ☑ 是否需要进行额外环境准备（包括搭建泳道以及配套的服务、新增地址的Ocean配置、Octo 泳道配置隔离等）

4. 实施办法(暂定)

关于实施办法主要从两个方面来进行考虑，分别是CodeDiff实践要点以及实施过程中的成效评估。

4.1 Code Diff操作指南

该部分指南，主要是针对不同LEVEL的同学，提供可操作性指导，如果需要外部协调资源，需自行来协调安排。

操作时间点	步骤	备注
RD自测阶段【不推荐】： • 适合中、小型紧急的项目 **联调阶段【推荐】：** • 适合大型紧急项目 • 适合中、小型项目 **提测之后【推荐】：** • 正常排期下的大、中、小型项目 **备注：** 一、项目规模定义【供参考】： **小型项目**- RD人日<= 7PD **中型项目**- 7PD < RD人日 <= 14PD **大型项目**- RD人日 > 14PD	1、结合需求文档，理解RD提供的技术方案，对于不清楚的地方重点标注 2、梳理完成后，对于不清楚点，及时和RD以及PM沟通，确保无疑问，且处理流程清晰 *-【根据自身情况，3/4 二选一】* ***3、下载代码，根据技术文档，进行第一遍浅层次Review，如果难度较大，需要申请RD资源进行代码逻辑讲解，如果要求合理且无紧急任务，相关RD不得拒绝【基础】*** 4、**下载代码，进行第一遍CodeReview，重点发现实现与需求不符、架构设计缺陷、异常流程考虑不足等层面的问题，避免后续大幅度改动【进阶】** *-【根据自身情况，5/6 二选一】* ***5、根据具体的业务逻辑实现，对于涉及到的点，按照CodeDiff实践要点P0级别条目，进行排查确认，对额外的校验条目暂不要求【基础】*** 6、**根据具体的业务逻辑实现，对于涉及到的点，除了P0级别条目外根据自身业务需求进行额外覆盖，进行排查确认，避免线上出现条目中的问题【进阶】** 7、对于Code Diff过程中发现的问题，及时在Ones上进行bug记录，并使用合适的标签（Reivew bug）进行标注 8、如果时间充足，建议再重复一次以上步骤	**级别解释：** 基础：适合初级级别的同学，Code Di 进阶： 适合非初级的同学，有多次Co **工具选择：** 针对各自的使用习惯，选用合适的Co • 直接下载代码，然后在IntelliJ ID • 要求RD在合并指定分支时，提交量较小的项目） • 通过git diff 自生成*.diff文件，然上2中方式的同学） **Code Diff占用时间分析：** • 通常Code Diff 占用时间应低于 • 上线时间紧急的项目，可根据业 **哪些项目更适合Code Diff：** • 涉及业务核心逻辑变更，或变更 • 涉及新增主要业务逻辑 • 传统功能测试方式质量保证风险

4.2 Code Diff实践要点

下面总结了不同的层面上可能存在的问题，可以重点进行关注，当然除了下方表中的要点也要关注其他部分。

类别	说明	思考要点	备注
业务逻辑实现	与需求是否一致	**需求文档里为A，代码实现为B，实现与需求不一致【P0】**	该类型Bug应最早发现，责成RD尽快
	业务逻辑是否闭合	**覆盖所有业务流程分支，正常情况，异常情况都要考虑，业务边界要清楚，接口定**	Saas与公交公共平台交互时，就存在

	业务逻辑是否闭合	**覆盖所有业务流程分支，正常情况、异常情况都要考虑，业务边界要清楚，接口定义明确【P0】**	SaaS与公共公共平台交互时，就停止题，边界划分不清
	新增业务逻辑是否与老逻辑兼容	**新老逻辑在实现上，不应存在冲突的情况【P0】**	新逻辑依据A字段判断，老逻辑依据B
	架构实现是否合理	**某项功能的实现，从架构层面看该功能的设计思路是否最优？有没有更好的架构方案？【P0】**	在前期介入的时候，就应该指出；后续变更的成本非常大
业务代码	添加或修改了方法	• **该方法的功能是什么【P0】** • **具体实现的思路是否清晰【P0】** • **该方法什么时机被调用(涉及影响范围)【P0】**	在考虑该方法的功能以及影响范当前的业务实现场景，着重异常场景的
	添加或修改了调用接口	• 该接口的接口类型，Http/Thrift, 如果为HTTP是否涉及HttpMessageConvertor • 该接口的加签/验签机制是什么 • **该接口的同步机制是阻塞还是非阻塞【P0】** • **阻塞类接口是否明确了超时时间设定【P0】** • 该接口是内部接口还是外部接口 • 该接口是否为强依赖，是否有降级方案 • 该接口在无明确返回时，是否有重试机制 • 对于该接口的不同异常返回，是否有针对性的进行处理 • **对于异常的返回值，超出约定的错误类型范围时，是否有兜底方案【P0】** • 如果为高频调用接口，性能如何，是否需要进行压测 • **接口服务异常、超时、无响应等是否添加了相关监控进行报警【P0】**	在涉及到公司外部的接口调用时，要求条件
	异常处理	• 是否有自定义异常类型 • Catch异常时，是否优先捕获自定义业务异常 • 对于多个自定义异常，捕获时是否进行特异性处理 • **确认所有异常是内部处理还是无法处理时向上层抛出【P0】** • **对于所有异常内部处理，校验是否存有兜底的异常（Exception）捕获【P0】** • 如果内部向外部抛出异常（Throw），上层是否进行了处理 • **对于可能需要清理的事务，是否添加了finally部分（比如释放锁）【P0】**	对于异常的处理倾向于如果内部可处理出，则优先内部处理
	并发与同步	• **梳理可能出现的并发场景【P0】** • 并发场景下，使用的数据结构或者变量线程安全 • 使用锁机制进行同步时，是否有发生死锁的风险 • **使用的锁机制是否正确，是否存在可能的性能问题【P0】** • 如果采用分布式锁，分布式锁的底层实现(Redis、Zookeeper)有无风险 • 某些场景下是否可以用乐观锁而非悲观锁 • 锁机制的使用是否正确	在高并发的场景下，同步场景的考虑要
	语法	• **赋值时是否对Null值进行判空并额外的处理【P0】** • 对于Json数据处理时，解析或者转换失败的Case是否有特定的异常捕获并正确的返回错误信息 • **对可能存在数组下标越界访问的场景是否进行了特别处理【P0】** • **|| 和&& 连用时，是否考虑Java的短路特性【P0】** • 对于循环部分，是否有正确的退出机制，是否有死循环的风险 • 使用JAVA 8新特性时，关于异常的捕获处理是否经过验证	之前在List.transform函数中存有发生的情况
DB操作	编码层级	• **新的数据方案与老的数据方案是否兼容【P0】** • **对于某业务逻辑进行事务添加是否合适【P0】** • 相关事务添加是否正确 • 项目回滚时，DB需要进行那些操作	CodeDiff时重点结合Mapper文件进行

目录

		• 敏感信息存储时，是否进行了加密 • 使用敏感数据进行数据检索时，是否进行了正常的加解密 • **事务部分是否对外部环境有强依赖(比如依赖mq)【P0】** • MyBaits Mapper文件编写是否正确，是否正确返回业务需求的字段 • 如果出现主从延迟会造成哪些影响，有哪些解决办法	
	数据表(Table)	• **新增表或者变更表是否和技术方案中描述一致【P0】** • **表的注释字段是否正确，对于公用表的各字段，确保各方对每个字段都理解一致【P0】** • 表字段的类型、长度设置是否合适，对于varchar类型是否存有截断的可能 • **表的非空字段默认值是否满足需求以及被正确处理【P0】** • 表字段的非空约束设置是否合理 • 是否根据业务逻辑建立表的索引，建立的索引字段是否合适 • **是否设置了UNIQUE KEY，设置是否符合业务需求【P0】**	可能还需要额外补充
系统	缓存	• 熟悉使用的内部缓存工具，Squirrel 或者Tair • **缓存工具的使用方法是否正确【P0】** • **读操作时是否有超时时间(失效时间)设定【P0】** • 是否对外提供了Rest设置接口(比较重要，特别涉及频繁修改配置的场景) • 缓存的更新以及触发方式是否正确设置 • **如果和DB结合使用时，获取和设置数据的顺序是否正确【P0】**	缓存的使用主要分为2个方面，单独作使用；结合DB，降低对高频数据访问性能影响。
	性能	• 自身频繁被外部访问的接口，明确QPS值是否满足业务要求 • 其所依赖的服务或者组件是否可以满足当前性能要求(比如Leaf生成ID的时间) • **清楚当前链路性能瓶颈的位置，新增的改动或添加是否再次降低瓶颈的性能指标【P0】**	这部分可以进行适当关注
	第三方依赖	• 熟悉当前的第三方组件依赖关系(MQ/Squirrel/Leaf/Crane等) • **对依赖组件的配置是否正确，比如Crane定时任务是否符合业务预期【P0】** • 判断对组件的强依赖关系是否可变更为弱依赖关系 • 对于弱依赖关系是否有降级方案以及故障预案 • **对于所依赖的第三方服务(比如OCR)，在调用时是否添加了相关监控【P0】**	对于外部的强依赖关系，一直是风险点
	安全	• 对于敏感数据是否进行了加密处理，比如手机号、身份证号、姓名等 • 对于敏感数据的访问，建议添加白名单来获取原数据，方便测试	安全这部分整体测试涉及较少，可不重
监控与日志	日志	• 当前日志记录确认是否接入LogCenter • 关键的业务点，关键的核心逻辑处是否添加了必要的日志打点 • **对于Exception异常，是否打印出异常调用堆栈【P0】**	异常堆栈的打印比较重要，否则不利于排查
	监控	• **核心业务指标是否添加了监控【P0】** • **依赖的接口调用是否添加了监控，监控的频次设置是否合适【P0】** • 对于特定性接口的监控是否明确了相关的责任人(可快速响应、快速解决) • 监控异常时，报警方式选择是否合理(大象、短信、电话) • 是否添加了没有必要的监控，报警的级别是否设置恰当	监控报警这块RD主R，QA负责
配置	OCTO	• MCC配置相关是否被滥用，key值设定是否合理 • 对于频繁访问的逻辑，是否在本地添加了缓存，通过通知机制而非主动查询进行数据更新 • **上线前相关的配置是否已经按照Checklist进行过确认【P0】** • 敏感数据比如密钥相关是否被放置在Mcc中而非KMS • **是否存在该使用MCC但是使用硬编码的情况【P0】**	合理的应用MCC,防止滥用和错用

目录

4.3 实施成效评估

在具体实施时，采用打分机制来进行评估RD和QA在此阶段所做工作成效。

目录

角色（Role）	指标（Index）	检测项(Check)	得分(Score)	备注
RD	技术实现方案	☐ 是否提供了技术方案 ☐ 明确测试场景以及重点覆盖范围 ☐ 对于关键技术细节，是否添加了详细说明 ☐ 技术方案中是否添加了交互图以及流程图 ☐ 数据表相关变更是否做了说明 ☐ 外部使用组件有详细说明 ☐ 业务流程描述是否采用Step by Step的方式		总共100分。对于扣分项可以在备
QA	基础指标	☐ 是否遗漏逻辑架构上明显漏洞 ☐ 是否发现实现与需求不符 ☐ 是否发现低质量bug ☐ 是否发现代码逻辑漏洞，比如检索条件有误等 ☐ 是否提供代码优化建议 ☐ 是否准确预估部分代码可能存有的风险，比如组件依赖相关		本指标及检测项目前还有待完善，项目缺陷在项目中已存在。
QA	进阶指标	待补充	-	-

5 备注

- Code Diff相关对于QA自身有着较高的要求，如果根据RD提供的相关文档仍然没有办法理清脉络，可以向RD要求面对面梳理，RD在无高优事情的前提下不应该拒绝
- QA在完成规范细则中的基础要求时，应该使用进阶部分相关细则来要求自身