**iOS代码自测及覆盖率检测方案设计**

**一、背景**

随着云闪付APP、网络支付组件SDK业务的不断发展，市场对产品研发质量有更高的要求。为提高版本质量，拟通过在研发阶段引入自测工具，提升代码自测覆盖率，优化研发流程。

**二、现状、问题及解决思路**

现有研发流程中，研发包含方案设计、自测案例编写、编码、案例自测四个步骤，研发人员完成自测后方可提交代码至生产库进入测试流程。流程图如下所示：‘

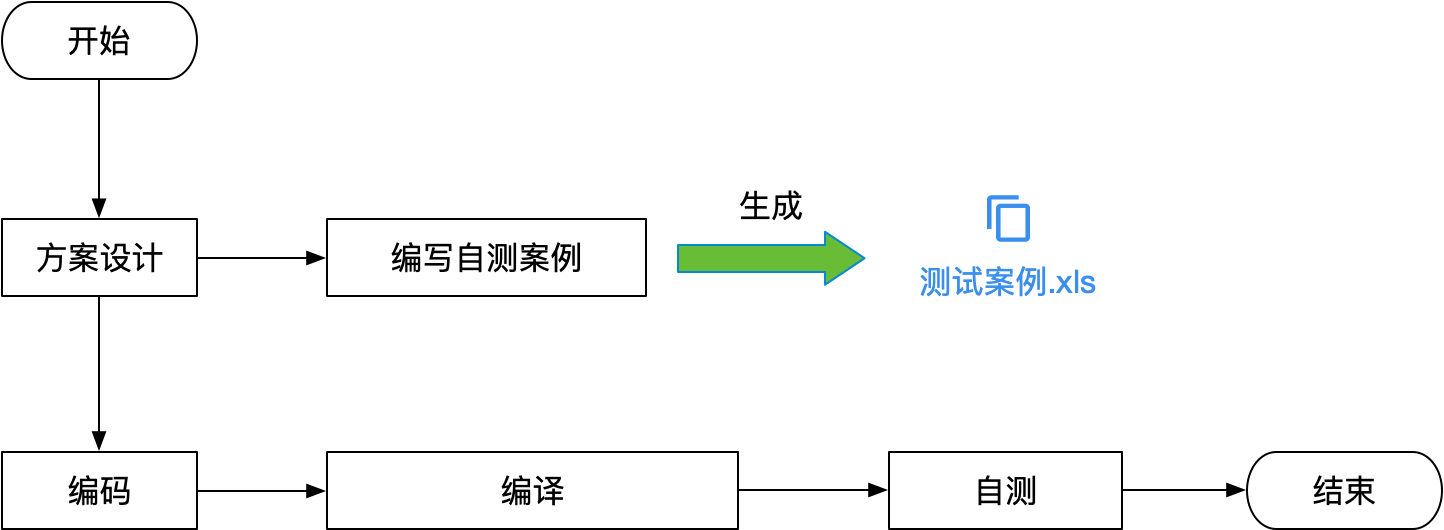


图1 现有研发流程（标蓝为版本提交物）

现有流程中，研发人员通过编写测试案例集自测，且自测阶段无工具辅助，因此测试质量依赖研发人员经验及主观能动性，存在下列问题：

1. 技术上，自测案例集无法保证对修改代码的覆盖，若研发人员经验不足，容易出现大量修改代码未测试的情况；
2. 管理上，管理人员缺乏手段检测研发人员自测情况。现流程中，研发人员只需提交自测案例集，是否真正完成自测并没有相应提交物佐证。

因此，针对研发流程中存在的问题，本方案拟通过在自测阶段引入代码覆盖率检测工具，辅助研发人员完善自测案例集，实现自测阶段修改代码全覆盖；通过工具生成的覆盖率报告，作为提测的附加物，确保开发人员在自测阶段真正完成自测。

**三、代码覆盖率工具测试选型**

当前，在iOS中主要有XCTest和llvm-cov两种工具提供代码覆盖率检测，工具相关的介绍如下：

**1. XCTest**

**1.1 简介**

XCTest是Xcode5中新引入的一个自动化测试框架，与 Xcode 的 IDE 直接集成，XCTest又可以分为两部分：Unit Test 和 UI Test，分别对应单元测试和UI测试。

**2.1使用方式**

单元测试：通过编写单元案例主动调用被测试的代码实现自动化测试，所有案例运行完之后可看到测试期间代码覆盖率，单元测试主要用于非UI代码测试。

UI测试：通过编写脚本模拟用户点击启动测试化工具实现自动化测试，运行完之后可看到测试期间代码覆盖率。

测试覆盖率展示如下：



图2 XCTest中查看各个代码文件的覆盖率

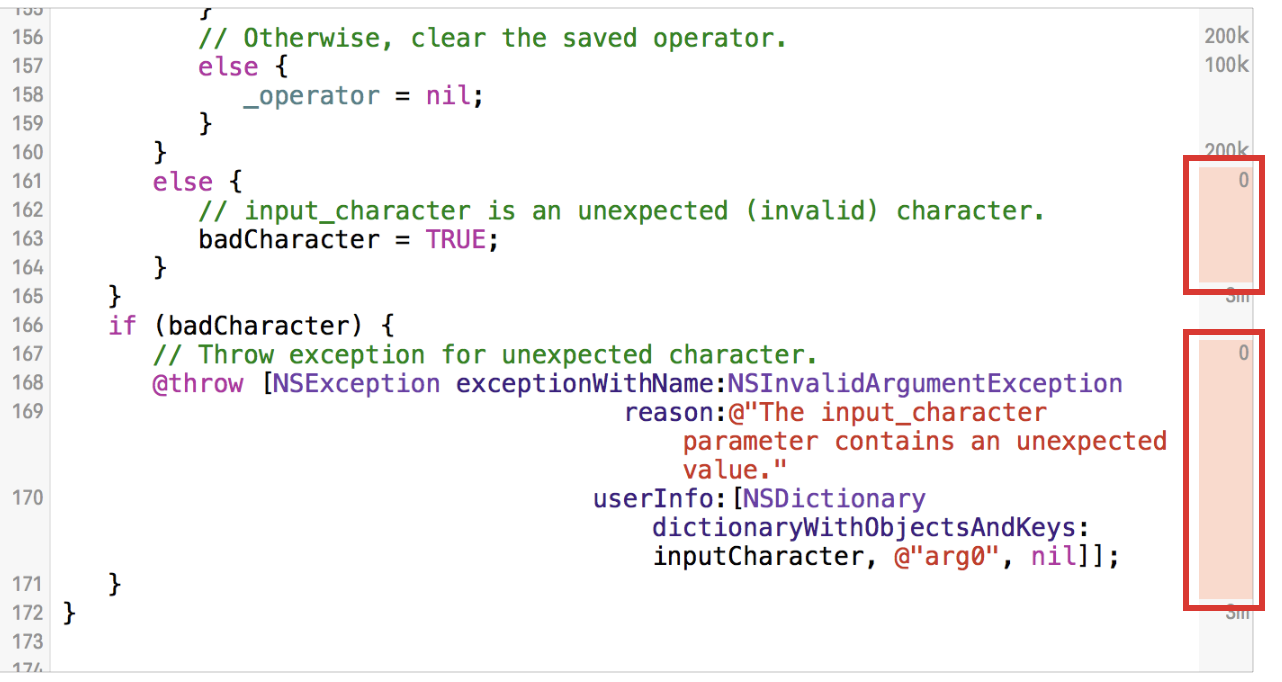


图3 XCTest中查看具体代码文件执行情况

**1.3 XCTest优缺点分析**

**优点**

1）通过编写案例实现全自动化测试，自动生成代码覆盖率报告，测试过程无需人工参与，使用简单；

2）在XCode中可以直接清晰看到代码覆盖率，以及具体哪些代码被调用到。

**缺点**

测试只能依赖自动化测试生成，无法通过操作APP的形式做代码覆盖率；因此需要人工编写写所有的测试案例集代码，维护成本高。

**1.4 XCTest小结**

XCTest为XCode官方提供的测试方案，测试过程必须依赖编写测试案例代码，无法人工参与。云闪付APP、网络支付SDK中包含大量动态化的页面，即页面的内容、布局依赖后台返回的数据，因此测试代码的维护难度大，基本不具备可行性。

**2. llvm-cov**

**2.1 简介**

llvm-cov为编译器llvm提供的代码覆盖检测工具。该工具通过解析APP运行过程中生成的代码覆盖文件，结合源码以及编译过程中生成代码标识文件，生成APP运行过程中代码覆盖率报告。

**2.2 使用方式**

1）添加编译参数“Generate Test Coverage Files”和“Instrument Program Flow”，该参数告诉编译器编译过程中需要对代码进行标识插装，并且生成对应代码覆盖标识文件（.gcno）

2）在代码合适的位置调用“\_\_gcov\_dump()”函数，该语句会将代码覆盖文件（.gcda）缓存APP沙盒内，示例代码如下：

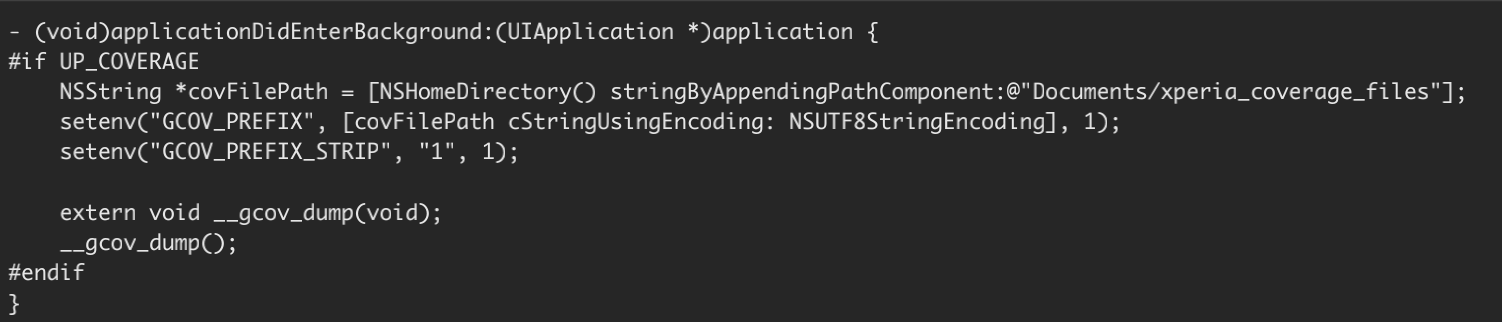


图4 App切换后台时调用\_\_gcov\_dump()

3）导出代码覆盖文件（.gcda），结合标识文件（.gcno）、源码（.m\.mm\.c\.cpp\.h）调用llvm-cov指令生成代码执行覆盖率文件（.gcov），覆盖率文件如下图（图中左侧一列标识该行代码执行情况）：

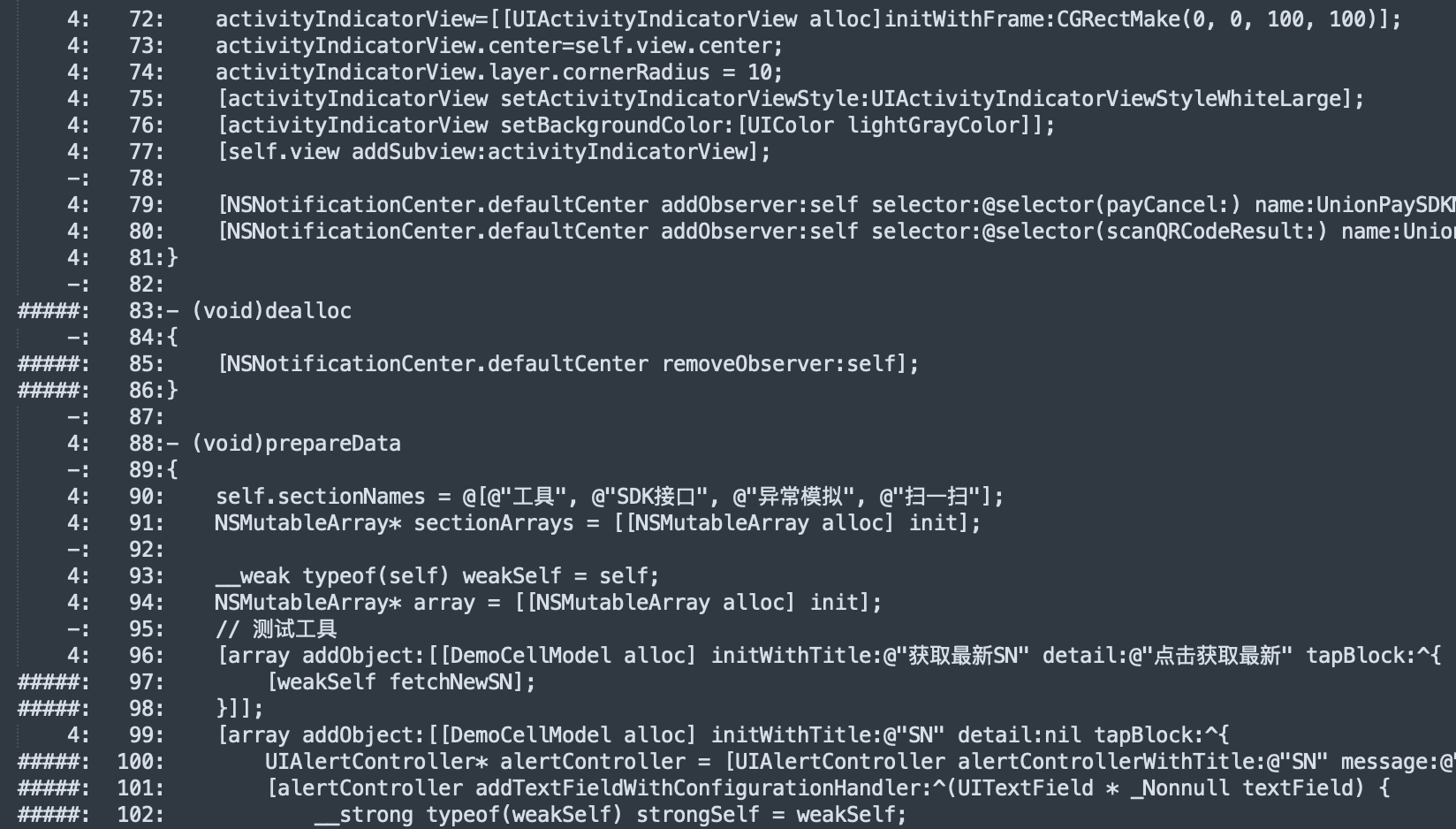


图5 App切换后台时调用\_\_gcov\_dump()

**2.3 llvm-cov优缺点分析**

优点

1. 代码覆盖计算不依赖测试工具，支持手工测试生成覆盖率报告；
2. \_\_gcov\_dump()函数可多次重复调用，自动合并生成的代码覆盖文件（.gcda），对于分支代码处理更加友好。

缺点

非UI操作，每次生成生成覆盖率报告需要手动导出覆盖文件，操作繁琐。

**2.3 llvm-cov小结**

相较于XCTest，llvm-cov为更纯粹的代码覆盖率检测工具，支持自测阶段手工操作测试覆盖率，更加适合云闪付App、网络支付组件SDK研发场景。

**四、代码覆盖率测试方案**

根据上文分析，本方案采用llvm-cov作为研发人员代码覆盖率检测工具。自测过程中，研发人员参照自测案例测试代码，生成对应的代码覆盖率报告，若代码覆盖率未达到100%，则研发人员需更新测试案例集，继续自测，直到代码覆盖率达到要求。提测时，研发人员需同时提交相应的代码覆盖率文件，以便后续检查。流程图如下：

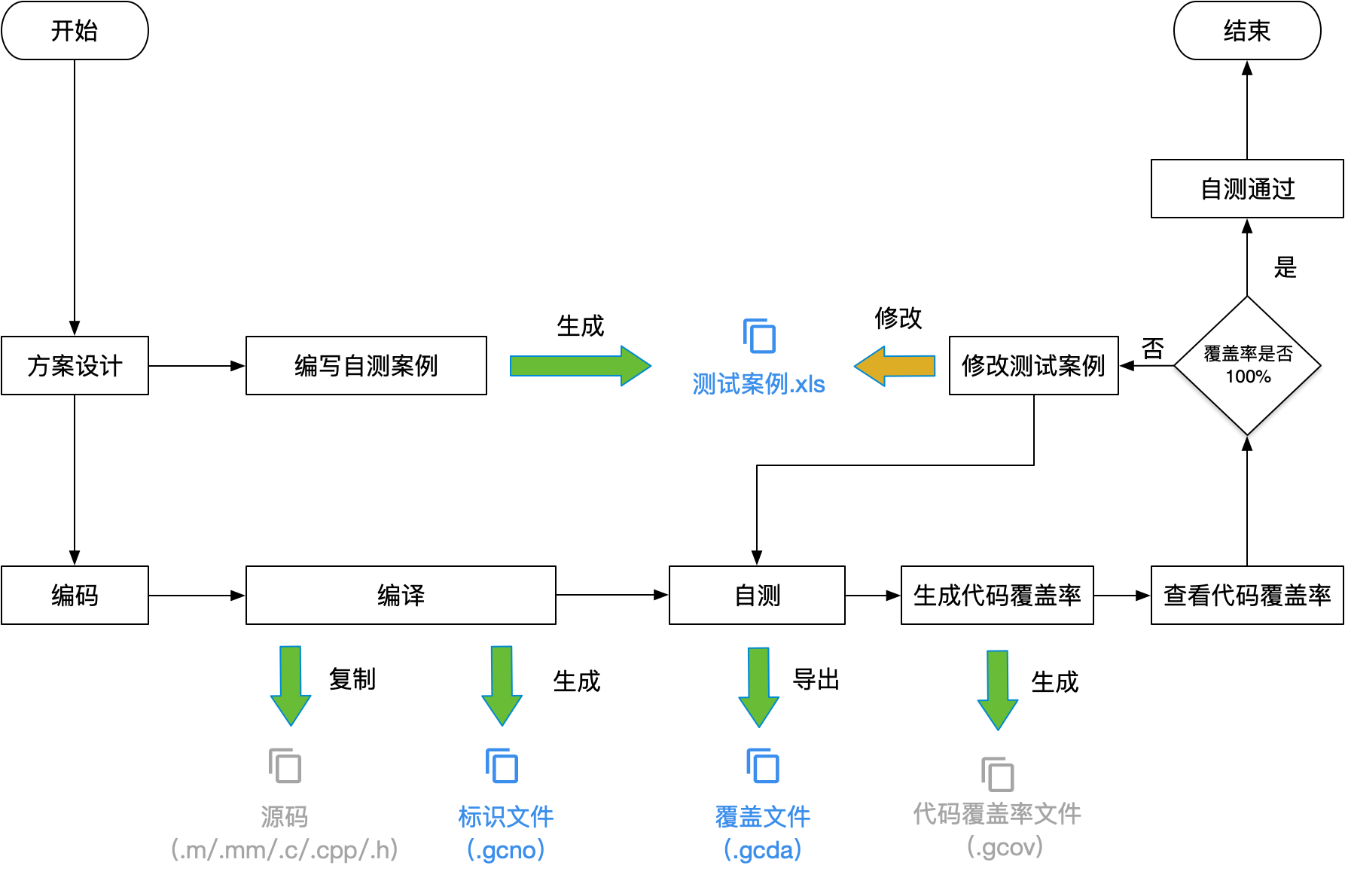


图6 修改后研发流程（标蓝为版本提交物）

**具体修改点**

1） 编译参数：Debug环境下配置，添加编译参数“Generate Test Coverage Files”和“Instrument Program Flow”，实现代码打桩；

2）代码：Debug环境下加入测试代码，实现手机“摇一摇”调用“\_\_gcov\_dump()”方法生成覆盖文件（.gcda）；

3）脚本：新增shell脚本，提供给开发人员自测时生成覆盖率文件；

4）版本提交物：新增gcno、gcda文件作为版本提交物，以便后续检查。