Automated Localization for Unproducible Builds

* Verifying software checksum : 检验和发布一致，没有被篡改过。
* Build Environment can also be compromised:
* Method: 比较编译出来的二进制代码。但是会有一些误杀的现象。
* 预编译出来的头文件不一定一样，因为可能使用枚举类型或者随机。
* 目前大约有10%的package不能保证每次都有同样的二进制代码。
* 给定相同source code、environment、在两个构建环境下，做diff，看结果是不是一样。如果不可重现，如何定位，如何自动化定位。
* 效果：以文件作为粒度，准确率能达到90%。修复了6个新bug。
* 细节：框架：
* 输入：diff日志：可读性差；信息量少。
* 导致不可重现的原因可能有很多，统计发现大约有10种。
* RepLoc 框架；
* Diff日志=》抽取做初步查询=》查询增强，利用了构建日志作为辅助。
* 归纳了一批启发式规则，到文件扫描，得到一批疑似的文件。
* 到文件进行信息检索，一个重新排序。加权排序得到文件列表。
* Query Augmentation:
* 启发式规则：人工归纳的规则。
* Empirical study : alpha的影响？

代码自动修复：需求和收益（华为）

* tryCoder: 自动化review，
* ide里面

contract-based program repair without the contracts

* 基于契约的程序设计
  + 把程序认为一组紧密交互的软件元素。通过一些定义的契约来协同。
  + 契约通常写成断言的形式。Class invariants; routine; pre-|post-conditions
  + Eiffel Java .Net等语言都有对程序契约的支持。
  + 契约的形式可以是非常简单容易理解的。Require 条件；ensure
  + 通过程序方式给出契约。对于程序员来说容易理解。表达能力也非常强。能用程序检查的性质都可以用契约来检查。
  + 程序+契约，在动态执行时检查契约是否被满足。契约可以看成程序的预期形式。
* 提出了AutoFix.
  + Automatic test generation + APR + IDE Integration(for Eiffel).
  + Use contracts for fault localization and fix validation.
  + Only for Eiffel
* Motivation
  + Generalize to
* JAID:针对JAVA语言的自动修复工具
  + 程序会抛出异常。会在Lower下界
  + 输入：Java program & test cases
  + Fault Localization -> fix generation -> fix validation -> fix ranking
  + Key Features
  + With a set of Boolean expressions to represent the program.
* Abstraction of Execution Traces:
  + State snapshots: snapshot-expression location value 程序执行到这位置时表达式的值
    - Expressions to monitor 动态监控表达式求值：要求表达式没有副作用。
    - Boolean-typed snapshot expressions：一些操作：运算符；得到很多表达式：分别刻画了一些状态方面
    - 运行所有程序测试：对表达式做求值
    - 对于每一个输入test，会有一组“state snapshot”
  + Frequencies of each snapshot
  + Similarity between a snapshot expression and the code nearby its location.
* Fix generation
  + Construct fix actions to change the snapshot states

Why is APR so difficult?

-Devide and Conquer

根据抛出什么异常进行分类-

安卓应用的程序崩溃修复

安卓的修复系统的期望：

设计android 生命周期有一定规律

最常见的崩溃的原因：关于生命周期的原因，关于resource的原因，可以自动修复。但是其他原因可能

Life-cycle aware repair operator :retain stateful object

遗传算法寻找补丁—

需要遵从两个方面的property

Hard constraints

Soft constraints

Which patch is better

Challenges in symbolically executing framework-oriented programs

Existing solutions

Java symbolic execution

instrumented -based

between the 2 solutions:

android program : java part(condition )android library

symbolic domain -> concrete domain

program synthesis 模拟library behabior

重建丢失的关于路径的约束。

希望不需要通过建模，而是符号化的执行安卓软件。

希望在真实环境中进行execution。

最主要达到的目的：通过程序自动生成，重建丢失的路径。

程序生成的工作。

Program synthesis

A set of ordered component

Path

Android Library

JPF-Android 手动建模

面向DevSecOps的代码安全保障体系

* 董国伟 360
* 软件安全新挑战
  + BSI
  + 微软安全开发生命周期 SDL
  + 应对开发和运维团队协同的DevOps
    - 通过自动化“软件交付”和“架构变更的流程”。
  + 国内代码安全方面问题
* DevSecOps新要求
* 360代码安全保障体系
  + 供应链分析阶段—360开源项目检测计划：有一个网站。
  + 软件测试阶段—模糊测试
  + 软件测试阶段-渗透测试
* 未来展望

阿里代码确缺陷检测探索与实践

* 测试代码服务
* Alibaba Code Center
* Defect Commit Labelling
* Commit Defect Features
  + 比较传统的特征
  + 特征提取：做了一些改进 > 分支上
* Method level
* Code2vec:
* More models: google transformer
* Model metrics:
* 触发精度缺陷
* 定位精度缺陷
* 应用级的精度缺陷自动修复工具
  + 现有方法的集成
    - 检测 定位 生成 约简
    - 有输入空间（出问题的区间、大部分没有问题，只有很少一部分出问题）
    - 对产生问题的代码块进行排序。
    - 针对这样的代码块做修复和变换。
    - 利用符号执行思想，用数学等价变换思想做替换。
    - 补丁验证：回归测试。
    - 补丁修复和约简：
    - 通常补丁有更高精度，可扩大作用范围。
    - 实验基准
* 数值库级别：缺陷发现方法失效。
* 高精度执行 != 更精确程序输出。
* 病态问题：
* 误差分析：条件数函数
* 后向误差
* 程序对应的数学函数相关性大
* 一点误差-> 导致整个误差被放大
* 新的修复方法AutoRNP:发现精度误差、检测精度误差、定位精度误差、修复精度缺陷
* DEMC：找到出发
* 迭代精化：
* 生成数学函数的近似：