Benchmark 论文整理

1. Introduction

论文背景

在软件开发中，不同tool对检测not-quite-right的定义不同,技术债不一定相同。不同tools对类似文件问题的报告是否相同，那些tool有更可能找出可能导致大问题的技术债并不确定。

extrinsic maintainability measures

版本修改数，bug修复数，版本修改的代码行数，修复bug的代码行数

常见软件measure(度量标准)

Line of code, complexity, cycles,cycle包含file和package两个方面

两个概念

co-change(historical)

code-based(structural)

2. Research Questions

1. 通过对类似问题文件的报告，不同tool的检测效果相符合程度为多少?
2. 对一些常见问题，不同tool检测是否一致?
3. tool检测到的问题是否真包含技术债，

3. Empirical Study

A. Tool Selection

选择软件的标准：易获得，可以在文件层面分析，结果机器可读（文件是任务分配的最小单位，与maintenance cost有关)

论文所选的tool如下

Tool:Measure	Abbr.	Aggr.
File Measures		
ARCH:Dependent Partners	ARCH_fanIn	
ARCH:Depends on Partners	ARCH_fanOut	
ARCH:Total Dependencies	ARCH_deps	x
ARCH:CoChange Partners	ARCH_coCh(h)	
DES:Size	DES_size	
DES:Complexity	DES_comp.	
DES:Design Smells	DES_smells	x
DV8:LOC	DV8_size	
DV8:TotalIssues	DV8_issues	x
DV8:Crossing	DV8_crossing(h)	
DV8:ModularityViolation	DV8_MV(h)	
DV8:UnhealthyInheritance	DV8_UH	
DV8:UnstableInterface	DV8_UI(h)	
SQ:Issues	SQ_issues	x
SQ:CodeSmells	SQ_smells	
SQ:Bugs	SQ_bugs	
SQ:Vulnerabilities	SQ_vuls	
SQ:SecuritySpots	SQ_sec	
S101:Size	S101_size	
S101:Fat	S101_fat	
S101:XS	S101_xs	
SCC:CLOC	SCC_size	
SCC:COMPLEXITY	SCC_comp.	
Cycle Measures		
S101:ClassTangles	S101_fileCycle	
S101:PkgTangles	S101_pkgCycle	
DV8:Clique	DV8_fileCycle	
DV8:PackageCycle	DV8_pkgCycle	
DES:Cyclically-dependent Modularization	DES_fileCycle	
DES:Cyclic Dependency	DES_pkgCycle	

(h)表示此measure包含了一个计划提交历史的信息， 第三列表示此measure是否是其他measure的汇总
Security Hotspot（安全热点）：需要审查的安全敏感代码段

B. Subject Selection

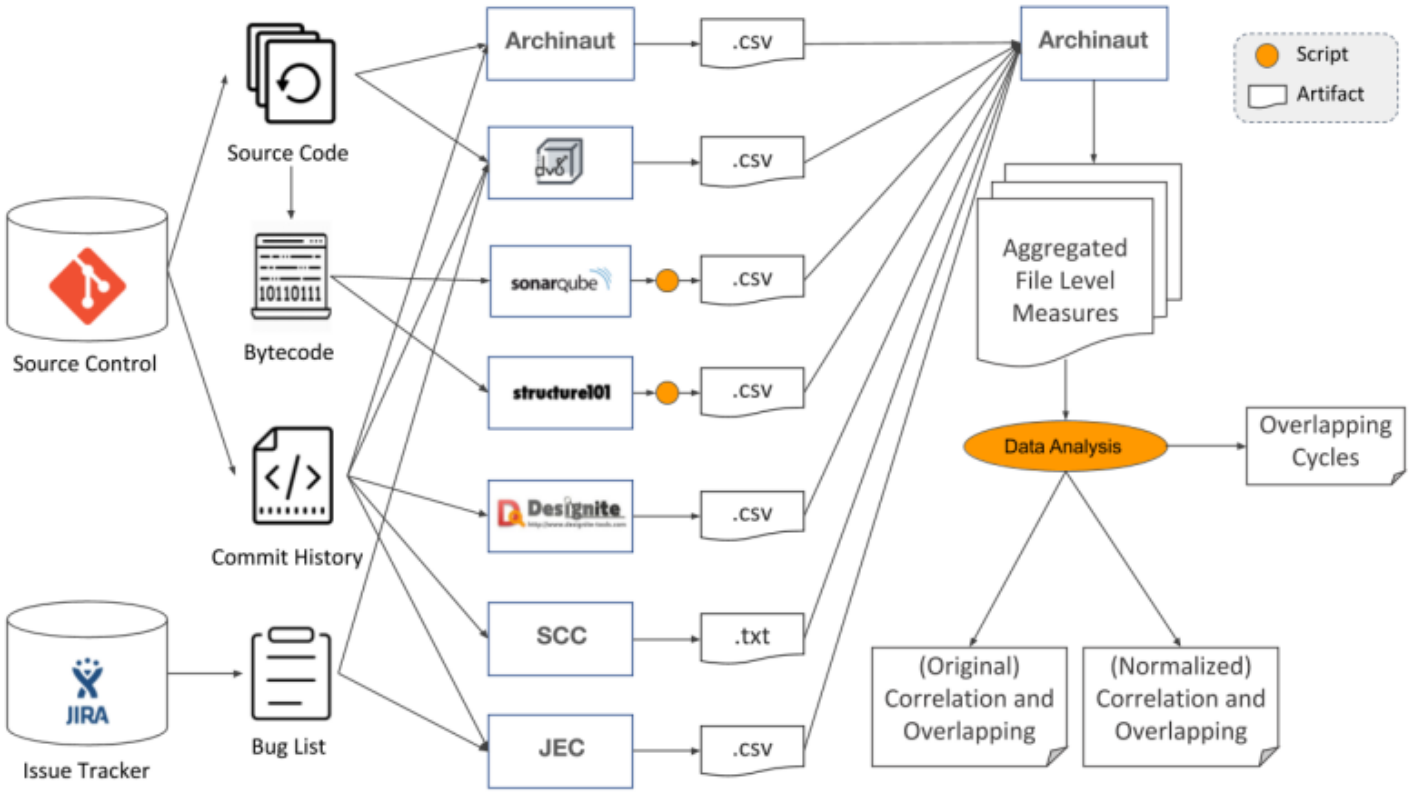
选择project的标准：至少1000次提交，源代码主要为Java。从20-MAD dataset中寻找

论文所选的subject如下

Project	Version	LOC	#File	#Com.	Period
DeltaSpike	1.9.2	147125	1824	1266	2014/03/16-2019/12/13
Flume	1.9.0-rc3	80266	441	1082	2012/02/14-2018/12/17
HBase	1.4.12	439093	1786	1088	2017/01/04-2019/11/20
Knox	1.3.0	82549	863	1098	2016/01/30 -2019/06/29
NiFi	1.10.0	629298	4112	1051	2018/05/03-2019/10/28
Oozie	5.2.0	151221	812	1059	2014/05/20-2019/10/29
Qpid-B[1]	7.1.6	321853	2000	1148	2017-04-17 2019/12/02
Qpid-J[1]	6.3.4	104181	561	1139	2016/01/25-2019/05/14
Tajo	0.12.0-rc0	265253	1557	1095	2014/03/11-2019/09/11
TEZ	0.9.2	159251	839	1082	2015/05/18-2019/03/19

"#Com" 为 commit数目

C. Data Processing



不同的tool有不同的输入，有不同的输出，Archinaut(ARCH)既是measure producer的一部分，也是不同tool结果整合的工具
SCC是用来测量代码大小和复杂度的一个工具（可以认为是准确的）
JEC是作者单独制作的tool

在Data Analysis部分，file 部分与cycle部分分离开，研究不同file measure的相关度，既对原数据分析，也对通过SCC数据normalized过后的数据分析，并看每个measure重叠的最多的20个最差的文件。同时对cycle，研究其overlapping

4. Results

RQ1
File measure的分析

Table III: Size Measure Pair-Wise Comparison

Avg. #Top-20-Overlaps (above diagonal)
Avg. Correlation (below diagonal)

Names	id	(1)	(2)	(3)	(4)
DV8_size	(1)	-	17.2	16.8	19.9
DES_size	(2)	0.98	-	15.7	17.1
S101_size	(3)	0.97	0.95	-	16.7
SCC_size	(4)	1.00	0.98	0.97	-

Table IV: Complexity Measure Comparison

Avg. #Top-20-Overlaps (above diagonal)
Avg. Correlation (below diagonal)

Names	id	(1)	(2)	(3)	(4)
DES_comp.	(1)	-	13.5	12.7	12.8
SCC_comp.	(2)	0.91	-	10.6	10.7
S101_fat	(3)	0.66	0.61	-	18.7
S101_xs	(4)	0.61	0.56	0.92	-

Table V: Normalized Complexity Measure Comparison

Avg. #Top-20-Overlaps (above diagonal)
Avg. Correlation (below diagonal)

Names	id	(1)	(2)	(3)	(4)
DES_comp.	(1)	-	2.9	1.0	1.0
SCC_comp.	(2)	0.50	-	1.3	1.4
S101_fat	(3)	0.13	0.15	-	15.6
S101_xs	(4)	0.11	0.14	0.77	-

表格直观地显示三个不同measure的相关系数，可以看出的是size大致都一样，但complexity（尤其是normalized后的）不同tool的结果不一样，且complexity受size影响大

Table VI: #Projects where Measures Agree

#Top-20-Overlaps $\geq$ 10 (above diagonal); Correlation $\geq$ 0.5 (below diagonal)

Names	id	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)
DES_comp.	(1)	-	9	9	9	2				2			2			7
SCC_comp.	(2)	10	-	6	6	1				3			2		1	5
S101_fat	(3)	9	7	-		1				3			2			3
S101_xs	(4)	10	8		-	1				3			2			4
ARCH_fanOut	(5)	6	8	3	2	-				7		1	8		1	2
ARCH_deps	(6)	1					-			2						
ARCH_coCh	(7)	3	1					-		3	1	1		1	1	
DES_smells	(8)					1			-							1
DV8_issues	(9)	7	7	3	1	8	4	5		-						2
DV8_crossing	(10)	1				1		2			-					
DV8_MV	(11)	2	1			3		6				-				
DV8_UH	(12)	5	5	4	4	8	1	2					-			1
DV8_UI	(13)	1	1	1		2	1	3						-		
SQ_issues	(15)	10	10	5	6	6	1		1	2			3	1		-

Table VII: #Projects where Normalized Measures Agree

#Top-20-Overlaps $\geq$ 10 (above diagonal)
Correlation $\geq$ 0.5 (below diagonal)

Names	id	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
DES_comp.	(1)	-							
SCC_comp.	(2)	4	-						
ARCH_deps	(3)			-					
ARCH_coCh	(4)				-				
DV8_issues	(5)			1		-			
DV8_MV	(6)				2		-		
DV8_UI	(7)				1			-	1
SQ_issues	(8)								-

此两个表格中数字为project的数目,可以看出的是不同tool所确认的最有问题的文件非常不同

Cycle measure 的分析

Table VIII: Package and File Cycle Comparison

Project	Package Cycle Comparison					File Cycle Comparison				
	DES-DV8-S101		DES ∩ DV8	DES ∩ S101	DV8 ∩ S101	DES-DV8-S101		DES ∩ DV8	DES ∩ S101	DV8∩ S101
	#set	#pkg				#sets	#file			
DeltaSpike	2-29-9	8-34-44	7	8	34	1-9-12	15-33-45	10	15	32
Flume	1-12-5	3-18-20	3	3	18	1-8-11	25-47-47	21	21	43
HBase	10-122-10	52-71-86	36	37	68	1-36-40	371-521-656	309	348	495
Knox	4-32-10	7-40-50	5	4	36	1-20-15	57-82-50	43	21	32
NiFi	16-145-37	41-157-164	36	34	141	1-86-62	144-330-354	90	118	217
Oozie	3-76-4	28-44-45	28	27	41	1-13-20	168-389-253	159	111	237
Qpid-B	8-117-8	15-82-97	15	15	78	1-54-41	263-332-586	190	236	283
Qpid-J	3-23-4	12-23-29	11	12	23	1-7-8	263-332-586	147	170	205
Tajo	8-77-11	37-68-75	37	37	68	1-33-39	182-452-527	160	174	442
TEZ	5-67-10	26-56-65	25	26	54	1-17-17	79-136-155	56	69	113

分析DES，DV8，S101三个tool对各个project所能检测到的package cycle和file cycle，结果如上，可以看出不同tool对cycle的结果还是很不一样的

RQ2

分析了size，complexity，cycle,smell detection不同的原因

size作者无法解释

complexity源于循环复杂度和文件层面复杂度计算不同

cycle源于计算cycle的instance不同和cycle的member计算不同

RQ3

通过一个文件在提交中出现的次数(#Change),和bug修复有关的文件修改数(#Bug),提交中增加或删除的代码行数(#ChangeChurn),bug修复有关提交中增加或删除的代码行数(#BugChurn)来预估maintenance cost，最后得出结论，除了需要历史提交的一些measure，对识别易错和易变的文件来说，大部分tool并不比常见measure更加indicative

5. Conclusion

选出6个tool，对10个project测试，比较各个tool不同之处，分析其不同源于哪，那个tool找出最针对maintenance的问题。最后得出的结论是一些基本的measure不同tools的结果都不同，也无法找出最针对maintenance的问题,也说明利用historical而不单单是structural的重要性。