

变更影响分析研究

介绍

变更管理是系统工程中的流程，是一种对系统变更请求、决定可达性等活动进行计划、实施和评估的过程。

变更影响分析是**变更管理**的其中一个环节，目的是为了评估变更的程度，具体来说，是识别变更的潜在后果，或估算完成变更所需的内容。

变更影响分析技术主要分为三个类型：

- 追溯型
- 依赖型
- 经验型

在追溯型变更影响分析中，会整理需求、规格、设计元素及测试之间的对应关系，根据这些关系确定更改的大致（初始）范围；在依赖型变更影响分析中会整理软件代码的变量、逻辑及模组等等之间的关系，根据这些关系来分析变更产生的影响；而经验型通常由专家设计知识来确认

研究趋势

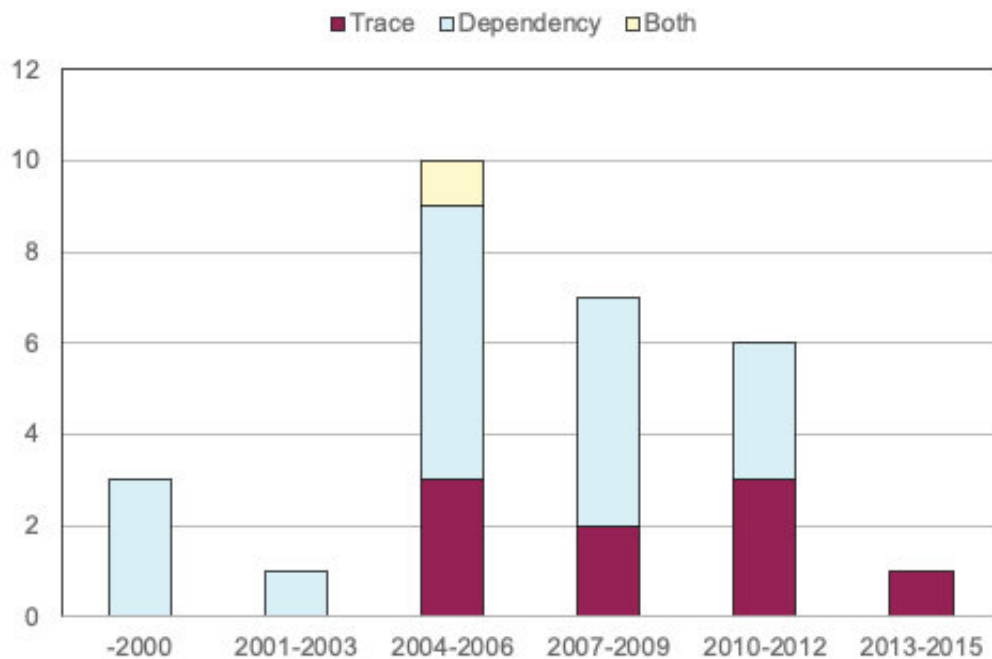


图1：影响分析工具发展演变图

据不完全统计，在2004年以前的变更影响分析主要着重于研究分析程序中的依赖关系；在2004年以后出现了追溯型的变更影响分析，主要利用代码仓库、代码历史挖掘等方式辅助变更影响分析，并且也存在工具同时使用了这两类型的分析技术。

表1：著名工具维护历史表

Tool Name	Year Created	Last Updated	Language(s)
CodeSurfer	1999	-	C
eROSE	2004	2005	Java
FaultTracer	2012	2016	Java
Imp	2012	-	C/C++
Frama-C	2007	-	C
ImpactMiner	2014	2015	Java
ImpactViz	2010	2011	Java
Impala	2008	-	Java
Indus	2005	2009	Java
JArchitect	2006	-	Java
JRipples	2005	-	Java
REST	2001	2008	C

上表是对一些较为出名的商业/开源变更影响分析工具的简单调查，主要包括项目创建年份，项目最后一次更新年份和变更分析所面向的程序语言。从上表可以看出，其中最早的工具 CodeSurfer 在1999年就被提出，并且使用至今；最新的工具为 ImpactMiner，该工具利用到了追溯型的分析方法，会在后文具体进行介绍。

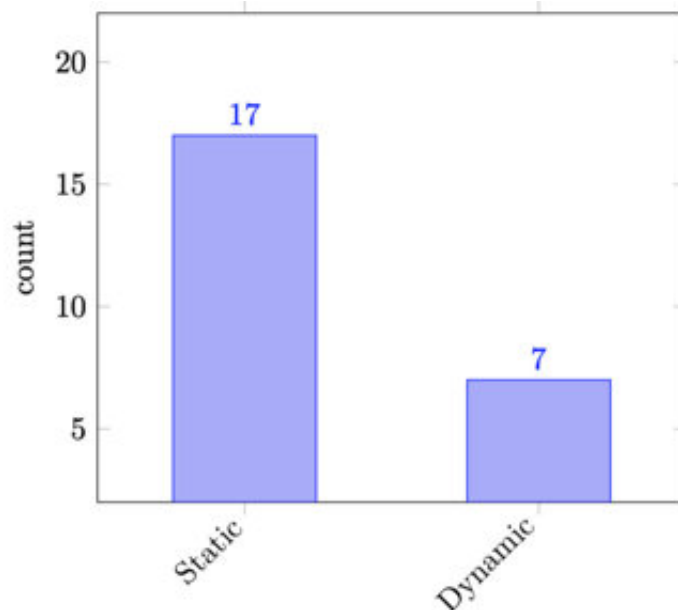


图2：按分析方法分

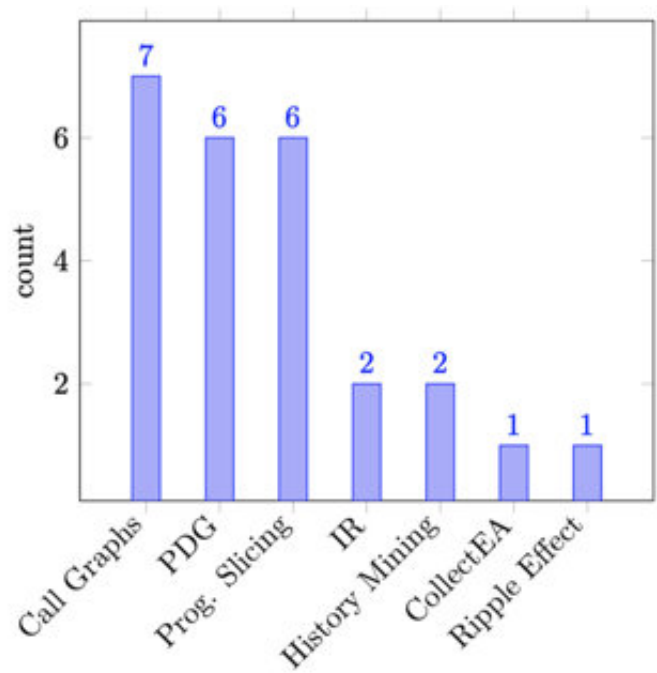


图3：按分析技术分

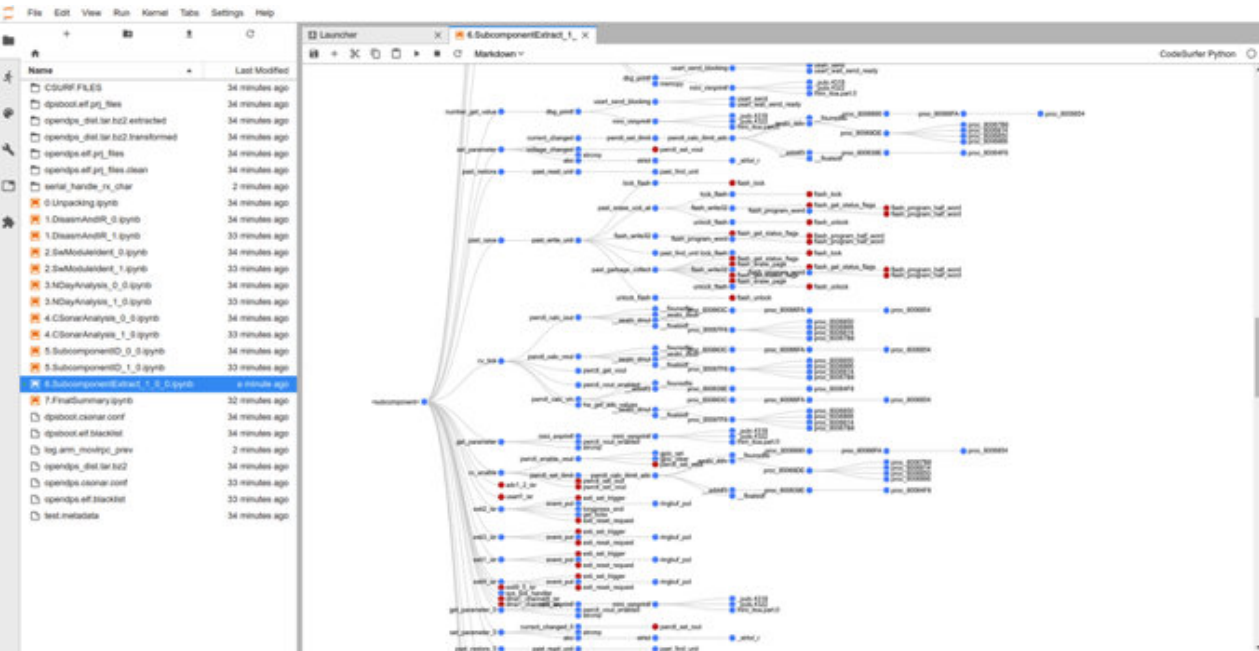
上表的工具可以依据两种划分规则进行划分，分别是分析方法和分析技术。从图2中可以看出变更影响分析工具以静态分析工具居多，而动态分析相对较少；从图3中可以看出，大多数的影响分析技术都考虑通过调用图和程序依赖图、程序切片的方式，这三种方式相互之间环环相扣，往往一个工具就同时具备这三种特征。

典型工具

1 CodeSurfer

名称	CodeSurfer
性质	商业工具
支持语言	Python, C/C ++和Java
特征	静态分析
维护周期	1999 – 2020
描述	通过可视化的程序切片帮助开发人员更好地理解代码，使用到的技术包括指针分析，调用图，数据流分析以及影响分析

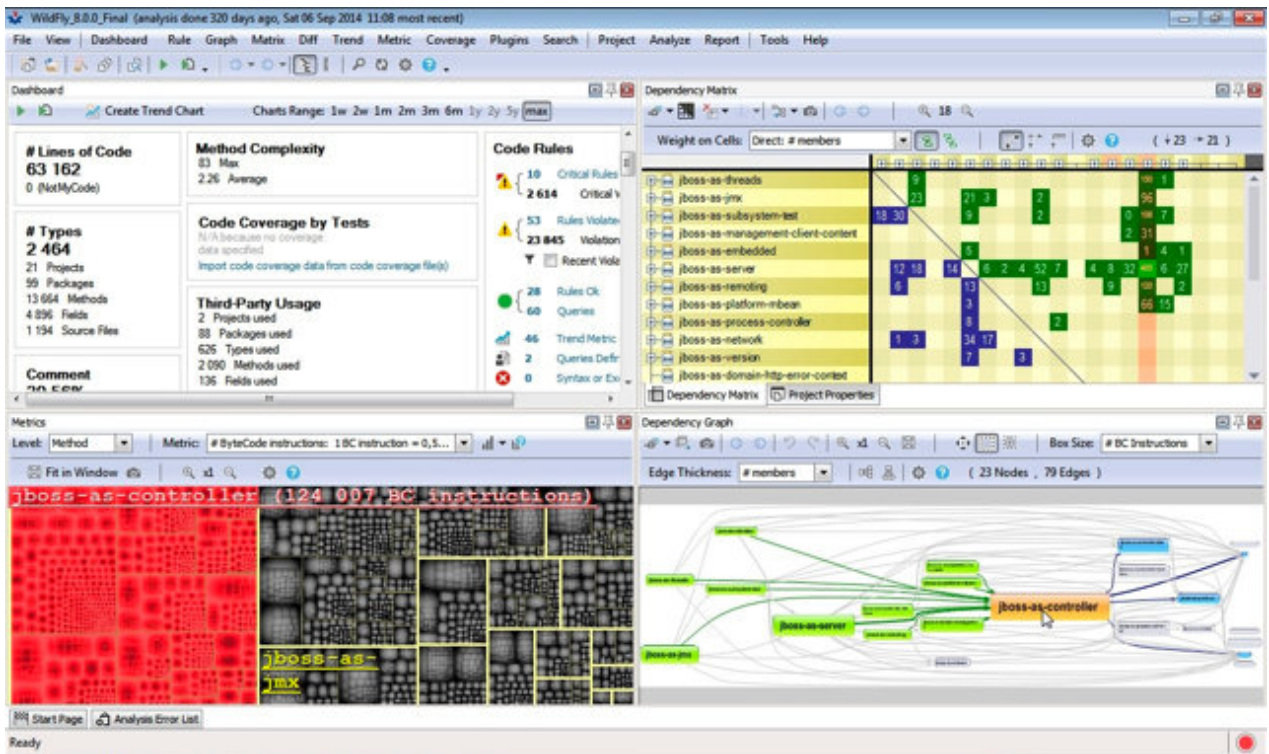
CodeSurfer 是一种静态分析工具，该工具的主要特点是能够构造可视化的程序依赖图。并在此基础上应用程序切片技术，包括前向切片和后向切片，最后通过高亮与变更相关的部分帮助开发者理解代码。工具的实际运行界面如下图所示：



2 JArchitect

名称	JArchitect
性质	商业工具
支持语言	Java
特征	静态分析
维护周期	2009 – 2020
描述	开发者选择他们工程中的一个实体，Jarchitect 会展示与该实体存在潜在联系的其他实体（projects、packages、methods、fields）

JArchitect 也是一种静态分析工具，该工具主要通过构造程序依赖图和依赖关系结构矩阵来进行变更影响分析，其中依赖关系图协助开发者判断变更信息 and 影响，关系结构矩阵提供工程中各个实体之间耦合的数据。界面如下图所示：

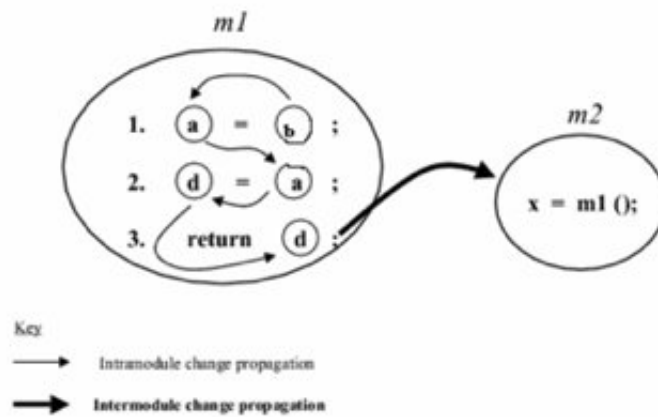


3 REST

名称	REST
性质	未开源工具
支持语言	Java
特征	静态分析
维护周期	2001 – 2008
描述	REST 通过计算 ripple effect 进行变更分析。

REST 的一大亮点是提出了波动影响的计算方法，使得程序员能够量化地分析每次变更对其他变量、模块等要素产生的影响。

其中波动影响主要是通过 4 个矩阵来计算，分别是 Matrix V, X, Z, C，他们分别表示当前模块的初始变更位置、当前模块内的变更传播信息、模块间的变更传播信息和模块控制流图的圈复杂度。模块内的变更传递和模块间的变更传递如下图所示：

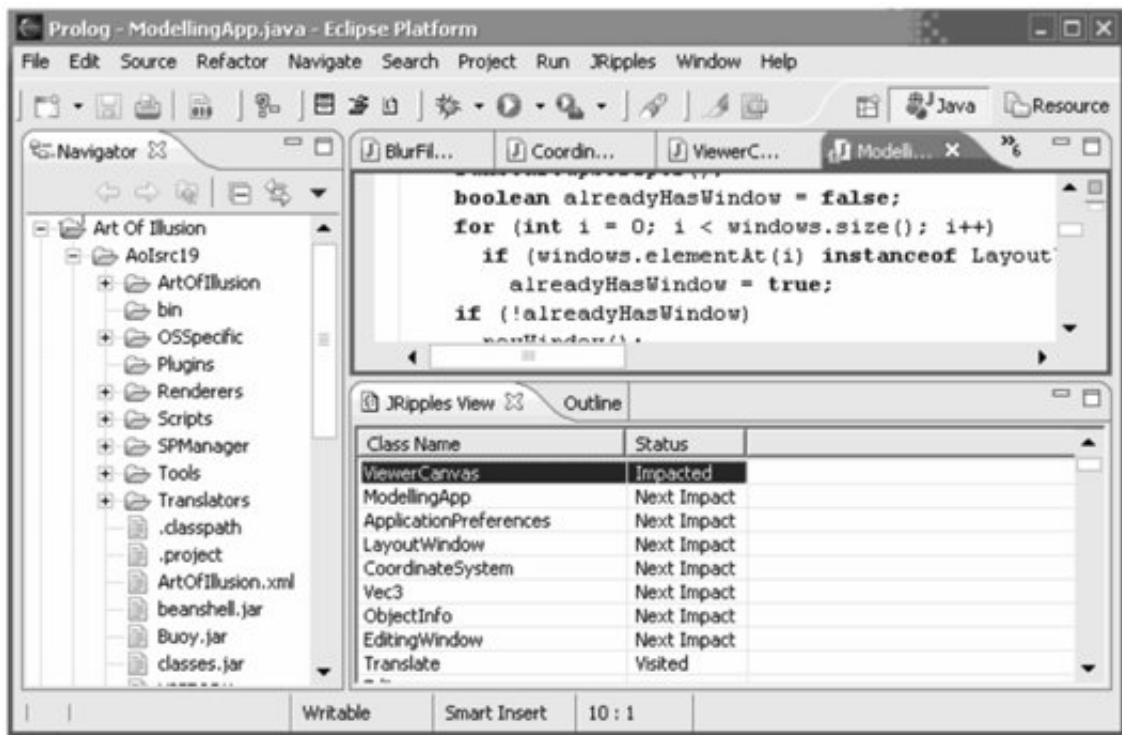


其中变量 b 的变更传递到变量 a，然后传递到d，这一条路径则为模块内的变更传播；当 m1 的值传递到 m2 的变量 x 时，这种则为模块间的变更传递。

4 JRipples

名称	JRipples
性质	开源工具， Eclipse插件
支持语言	Java
特征	静态分析, 软件信息检索
维护周期	2005 – 2020
描述	JRipples 由解析器，具有组织规则的数据库和用户界面组成，属于半自动工具

JRipples 分析Java项目中的依赖关系，并返回可能受影响的类的结果集，供开发人员查看。然后，开发人员逐一检查结果，并将其标记为“Impacted”或“Visited”。其中软件信息检索（DepIR），IR的作用是补足依赖分析在某些方面的缺陷，如处理数据库依赖（database dependency），如果切片过大，IR能够根据用户的需求定位到相关的位置。其界面如下图所示：



5 ImpactMiner

名称	ImpactMiner
性质	开源工具， Eclipse插件
支持语言	Java
特征	静态分析， 动态分析， 历史信息挖掘
维护周期	2014 - 2015
描述	该工具提出了一种SVN仓库挖掘的方法，并结合了另外三种分析技术：信息检索，动态分析，历史分析。

ImpactMiner 提出了一种SVN仓库挖掘的方法（MSR），并结合了另外三种分析技术：信息检索（Information Retrieval），动态分析（Dynamic Analysis），历史分析（Historical Analysis）。开发者可以任意选择分析技术进行变更影响分析。

MSR

其中 MSR 的主要方式是

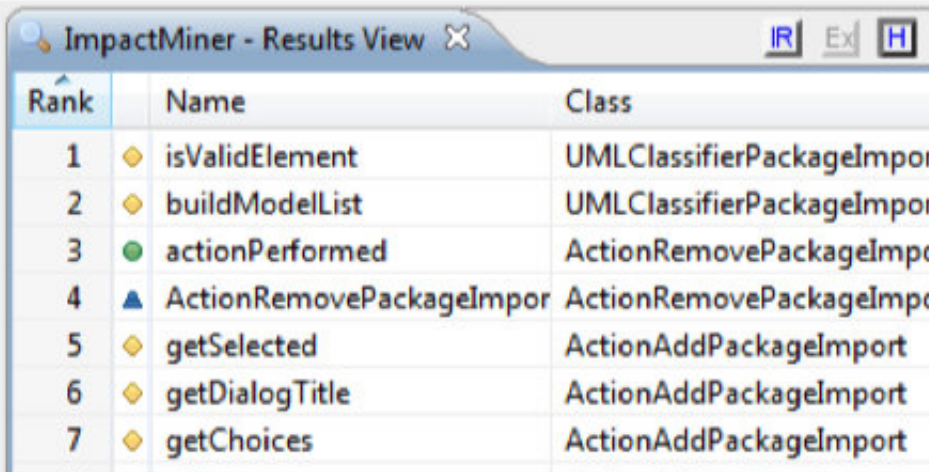
1. 通过SVN获取项目的变更集(输入SVN URL和限制范围)

$$\frac{N}{\text{fileName.java.rev}N}$$

$$\frac{N}{\text{fileName.java.rev}N - 1}$$

2. 如果某个版本发生了变更，则生成变更后和变革发生前版本的AST，比较两者的AST，提取出与每个变更版本相关联的方法集合 *itemsets*
3. 计算 $\text{supp}(X)$ ，表示 item X 在所有的commits中所占比例
4. 计算 confidence of rule $X \Rightarrow Y$, $\text{conf}(X \Rightarrow Y) = \text{supp}(X \cup Y) / \text{supp}(X)$
 1. 这些规则可以解释为“过去修改过方法 X 的开发人员，也修改过方法 Y, Z 等
 2. 开发者可以选择自己感兴趣的方法X，工具会根据 confidence scores 建议相应的方法，如 Y

其结果会根据confidence自上而下排序显示于列表中：



Rank	Name	Class
1	isValidElement	UMLClassifierPackageImpor
2	buildModelList	UMLClassifierPackageImpor
3	actionPerformed	ActionRemovePackageImpc
4	ActionRemovePackageImpor	ActionRemovePackageImpc
5	getSelected	ActionAddPackageImport
6	getDialogTitle	ActionAddPackageImport
7	getChoices	ActionAddPackageImport

Figure 3 Results of generated by the MSR technique when the method UMLClassifierPackageImportsListModel was used as a seed

IR

IR 的工作方式是将一组 artifacts（例如源代码文件）与 requests（例如变更请求）进行比较，并根据它们与查询的相关性对这些 artifacts 进行排名。

1. 建立语料库：获取每种方法的文本
2. NLP：删除运算符和编程语言的关键字，对复合词进行分割（如：impactAnalysis 变成 impact 和 analysis）；将单词还原成词根形式（如：impacted 变成 impact）
3. 用IR索引语料库(倒排索引)：构建矩阵，行对应语料库中的单词，列是相应的方法； m_{ij} 表示该单词i与方法j的相关度，然后利用SVD降维。
4. 执行查询：将 change request 作为工具的输入
5. 评估影响集：将change request 转化成向量表示并与步骤3每个method的向量

进行余弦相似度比较，越高则越相关。

Dyn

利用 Java 平台调试器体系结构（JPDA 3）和测试和性能工具平台（TPTP4）收集有关软件系统的运行时信息。

结合

IRDyn: 从IR结果中消除了没有出现在执行轨迹中的结果

nIRHist:

- IR 在位置 i 上的结果出现在位置 2i-1 的 IRHist 列表中
- Hist 在位置 i 上产生的结果出现在位置 2i 的 IRHist 列表中
- 如果某个方法同时出现在“IR”和“Hist”列表中，则它在 IRHist 中仅出现一次

DynHist:此组合仅返回出现在执行跟踪和关联规则结果中的方法。

IRDynHist: 首先计算 IRDyn, 然后与 IRHist 规则相同，区别是 IRDyn 替代 IRHist 中的 IR

这几种结合方式的实验结果如下：

Table III. Precision (P) and recall (R) percentages results of IR _{CR} , combination IR _{CR} Dyn _{CR} , combination IR _{CR} Hist _{seed} , and combination IR _{CR} Dyn _{CR} Hist _{seed} approaches to IA for all systems using various cut points.																							
Cut Points																							
		5		10		20		30		40				5		10		20		30		40	
Precision (P) and Recall (R)		P	R	P	R	P	R	P	R	P	R	jEdit	P	R	P	R	P	R	P	R	P	R	
IR _{CR}		7	4	6	6	5	12	4	14	4	18		10	7	9	13	6	20	5	26	5	30	
IR _{CR} Dyn _{CR}		11	7	8	10	6	19	6	26	5	28		17	14	14	25	10	35	8	23	7	50	
IR _{CR} Hist _{seed}		15	14	12	19	9	25	7	28	6	33		11	11	9	22	7	34	5	43	5	47	
IR _{CR} Dyn _{CR} Hist _{seed}		17	16	13	22	10	31	8	37	7	41		18	23	14	37	9	53	8	64	7	75	
IR _{CR}		9	4	11	11	8	22	7	25	5	28	muComander	7	9	6	13	5	19	4	20	4	24	
IR _{CR} Dyn _{CR}		14	9	11	14	8	24	6	29	5	31		11	11	9	17	7	22	5	27	5	34	
IR _{CR} Hist _{seed}		9	4	11	14	9	24	7	38	6	40		8	14	6	22	5	30	4	32	4	36	
IR _{CR} Dyn _{CR} Hist _{seed}		14	15	11	21	8	33	6	45	5	48		12	19	9	25	6	34	5	37	5	46	

从图中能够看出同时使用这三种方式的效果最好。

参考文献

[1] Galbo, Stephanie Perez Day. *A Survey of Impact Analysis Tools for Effective Code Evolution*. Diss. 2017.

[2] Li, Bixin, et al. "A survey of code-based change impact analysis techniques." *Software Testing, Verification and Reliability* 23.8 (2013): 613-646.

[3] Bilal, Haider, and Sue Black. "Using the ripple effect to measure software quality." *SOFTWARE QUALITY MANAGEMENT-INTERNATIONAL CONFERENCE-*. Vol. 13. 2005.

[4] Dit, Bogdan, et al. "Impactminer: A tool for change impact analysis." Companion Proceedings of the 36th International Conference on Software Engineering. 2014.

[5] Gethers, Malcom, et al. "Integrated impact analysis for managing software changes." 2012 34th International Conference on Software Engineering (ICSE). IEEE, 2012.

[6] Buckner, Jonathan, et al. "JRipples: A tool for program comprehension during incremental change." 13th International Workshop on Program Comprehension (IWPC'05). IEEE, 2005.

[7] Black, Sue. Computation of ripple effect measures for software. Diss. London South Bank University, UK, 2001.