# 基于语义影响范围分析的补丁兼容性分析

（申请清华大学工程硕士学位论文）

培 养 单 位：软 件 学 院
学　　科：软 件 工 程
研　究　生：潘　晓　梦
指 导 教 师：贺　飞　副　教　授

二〇一五年四月

Thesis Submitted to

**Tsinghua University**

in partial fulfillment of the requirement

for the professional degree of

**Master of Software Engineering**

by

**Pan Xiaomeng**

**( Software Engineering )**

Thesis Supervisor : Professor He Fei

**April, 2015**

# 关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：

清华大学拥有在著作权法规定范围内学位论文的使用权，其中包括：（1）已获学位的研究生必须按学校规定提交学位论文，学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文；（2）为教学和科研目的，学校可以将公开的学位论文作为资料在图书馆、资料室等场所供校内师生阅读，或在校园网上供校内师生浏览部分内容。

本人保证遵守上述规定。

**（保密的论文在解密后应遵守此规定）**

作者签名：_____　　导师签名：_____

日　　期：_____　　日　　期：_____

# 摘　要

我还不知道写什么，就先这样吧。

**关键词：**

# Abstract

The software system will continue evolving in its whole lift time,and patch is a common way to accomplish the task to update or fix bug and so on.As the software evolving is rapid, how to identify whether a patch for a specific version of software is applicable for an newer version of the same software.We here present a so called patch compatibility analysis to try to fix this problem.It consists of three kind of analysis—program differencing which is used to output the structural difference of the two versions,change impact analysis which is used to analysis the impact of every change including dependency and impacted expressions/statements and so on.And at last,the conflict analysis used to determine whether the two patches' impact are confictual.

**Key words:** TEX; LATEX; CJK; template; thesis

# 目 录

# 第 1 章 绪论

## 1.1 研究背景和意义

软件维护（Software maintenance）是软件开发周期中耗时最长、开销最大的过程。随着外部环境和用户需求的不断变化，软件系统需要随之进行适应和调整，同时也需要修复在实际运行中暴露出来的问题。

补丁（Patch）就是这样一类可以用于完成修补程序漏洞、增强软件功能、改善程序性能等任务的程序。补丁在工业界中得到了广泛的实际应用，是软件维护过程的重要组成部分。

补丁一般是通过 Diff 工具而产生的文本数据，用以表示以行为单位的程序间差异性。一般而言，Diff 工具属于程序差别分析工具中的一种，单纯进行文本的差异性比较，并给出行级别的差异性信息。然而这样产生出来的差异性信息（即 patch），只包含了文本差异，而失去了语法、语义等差异信息。

然而，补丁程序仍然具有一定的局限性，它一般只针对某个专门软件版本而开发，对于软件演进过程中获得的新版本而言，我们无法确定补丁程序是否也能适用于新版本的程序。然而补丁程序的应用一般都具有特定的目的，例如功能升级、漏洞修补等，新版本的程序中很可能仍然需要补丁程序的应用来完善自身。

## 1.2 本文所要解决的问题与主要工作

软件系统总是处于不断演进的过程中的。补丁 (patch) 是一种常见的软件系统版本演进的方法，常用于修补漏洞和缺陷、改进性能、升级等。在实际应用中，补丁往往针对某一个特定版本的代码而设计，然而现代软件系统往往更新换代较快，因而面临着将补丁应用于其他版本的代码时可能失效的问题。为了解决这个问题，我们为此提出了进行补丁兼容性分析的想法。

为此，我们考虑这样一个问题，针对软件系统 s，假设已有有版本 v1 和 v2，其中补丁 p1 是针对版本 v1 而设计，打上后会演进到版本 v3，问 p1 是否能够应用于版本 v2，并保证不会发生冲突。从问题中可以发现，p1 = diff(v1,v3)，设 p2 = diff(v1,v2)，那么该问题就可以规约成看两个补丁 p1 和 p2 间是否会发生冲突？

对该问题而言，其答案可以分为多个层面来回答：

1. 语法：从语法角度出发，则该问题主要关注的是在应用过程中是否会造成语法结构上的错误，如果能够将补丁 p1 成功应用于程序版本 v2，则认为该补

丁是可以兼容于新版本 v2 的。

2. 语义：从语义角度出发，则单纯的语法兼容并不能够完全解答这个问题。某行修改可能会影响多处源代码，从而导致程序的行为发生变化。因而从语义层面而言，我们需要保证补丁 p1 对程序版本 v2 造成的语义影响不会波及到从版本 v1 演进到 v2 时所造成的语义变化。

语法层面的回答很容易就能给出，现有的版本控制系统等都能在一定程度上给出相应的答案。而语义层面的答案，就目前所知尚未有这方面的工作。

因而本文将着重从语义层面尝试去解决这个问题。该问题可以较形式化地描述如下：

**问题 1.1：** 具体定义

目前而言，本文的主要工作包括以下几个部分：

1. 补丁版本迁移

   补丁 s1 本来是适用于程序版本 p1 的，如果想要适用于程序 p2，可能需要进行一定的版本迁移工作。

   该部分工作可以从语法层面给出兼容性的答案，并进行补丁兼容性的语法结构修正。

2. 语义影响范围分析

   本文主要采用程序间差异分析、变更影响分析等手段来对界定补丁 s1 对程序 p2 所造成的语义影响范围。

   所谓的语义影响范围（semantic impact area）可以具体定义如下，其中 structure 意为程序语法结构，可以采用不同级别的程序语法结构（如 statement、basic block 等）进行分析，来获得不同粒度的影响范围：

   (a) change(s) = structurei | structurei 属于 p，并且 structurei 发生了变更，该集合可以采用程序间差异分析获得。

   (b) impact(s) = structurei| structurei 属于 p，并且 structurei 受到 change(s) 的影响，该集合可以采用变更影响分析获得。

   最后得到的 impact(s) 即为我们所需的变更的语义影响范围。

3. 补丁兼容性分析

   在有了语义影响集合 impact(s) 之后，可以进行具体的兼容性分析工作。

## 1.3 本文组织结构

本文主要包括七个章节，第一章是绪论，介绍本文的研究背景和主要工作；第二章主要介绍与本文所述内容相关的国内外的工作；第三章主要介绍了补丁兼

容性分析的具体方法；第四章主要介绍了如何将各阶段的不同分析方法进行整合，形成具体的流程；第五章详细介绍了具体的补丁兼容性分析算法；第六章介绍了实验过程和结果；第七章主要对本文的工作进行了总结，并提出了进一步的工作方向。

# 第 2 章　相关工作

## 2.1　软件演进

## 2.2　程序间差异分析

对于软件演进分析而言，如何确定一个程序的不同版本之间的变更是一个关键性的问题。程序间差异分析能够通过分析同一程序的不同版本间的差异，来确定版本间的变更集合。

按分析的深度而言，程序间差异分析可以分为三类：

- 文本差异：单纯的对比文本间的不同，这是最简单也最广泛应用的分析方法，如 Unix Diff 工具。
- 语法差异：对比并获得源代码间语法结构上的不同。
- 语义差异：对比并获得源代码间语义层面上的不同。

现有的帮助工程师进行软件维护和演进活动的工具往往都受限于低质量的变更信息。例如现在源代码的变更信息，往往都存储于版本管理系统中，如 CVS 等。他们会追踪对某个特定文件的文本行的增加/删除等操作，并没有考虑代码中的结构化变更。

考虑到源代码可以抽象语法树（Abstract Syntax Tree）的形式进行表达，可以考虑采用树间差异分析的方法来抽取出这些变更信息。Change Distilling 就是这样一类进行树间差异分析的算法。该算法能够从两棵 AST 之间寻找匹配节点，并找到一个能够令一棵树转化为另一棵树的最小变更集合，该变更集合即为所求的程序间差异。由于是从 AST 的实体和语句中抽取信息，该算法可以获取结构化的变更信息。

Change Distilling 中采用二元字符串相似性来匹配源代码语句，并使用子树相似性来匹配源代码结构（如语句、循环等）。在寻找变更集合时，它采用基本的树变更操作来描述源代码的变更，包括更新、删除、增加等。在实际的使用过程中，该算法可能会受限于如何找到合适数量的移动操作。

## 2.3　程序变更影响分析

软件维护是软件开发周期中最复杂、高成本和劳动密集的活动。软件产品天然的需要跟随系统需求的变更而进行适应和变化，以满足用户的需求。软件变更 (software change) 是软件维护 (software maintainace) 中的基础组件。变更可能从新

的需求、缺陷修复、变更请求等而来，当变更应用于软件时，他们不可避免的会带来一些副作用，也可能会与原软件的其他部分产生不协调。

而 CIA 正是这样一类用于确定变更对于软件其他部分的影响的技术集合，它在软件开发、维护和回归测试等过程中都起到着重要的作用。一般而言，CIA 可以用于程序理解、变更影响预测、影响追踪、变更传播、测试用例的选取等。

CIA 方法的分类：

- 基于追踪性的 CIA，追踪两个不同抽象级别上的元素间的依赖性，目的在于链接不同类型的软件工件（需求、设计等）
- 基于依赖的 CIA，致力于衡量变更的潜在影响，通常尝试去分析程序的语法关系，他们表示了程序实体间的一些语义依赖关系，或者说他们在同一个抽象级别上对软件工件实施了影响分析。现在最主要的这类 CIA 主要在源代码级别上进行研究。

软件系统上的变更可能导致不可意料的副作用，CIA 的目标就在于辨认这些副作用 (side effect 或者 ripple effect)，并防止之。CIA 从分析变更请求和源代码开始，以便获得变更集合 (change set)。最后实际获得的影响集合叫做 EIS（estimated impact set），而真实的集合叫做 AIS（actual impact set）。

整个软件变更影响的分析过程可以大致分为以下流程，更详细的流程可以参考图 2.3。

1. 确定变更集合，需要针对变更请求进行分析，这步一般叫做特征定位 (feature location)，用于确认源代码中实现了对应功能的起始位置。
2. 衡量由变更集合引入的影响。这一步是目前大部分 CIA 技术的着重点。若按分析技术的类型划分，则主要的两类包括静态分析和动态分析。
    - 静态分析：历史分析、文本分析、结构分析。静态分析通过分析程序的语法、语义或者历史依赖实现，通常会产生很多错报（false positive）。
        - 结构分析着重于分析程序的结构依赖性，构建依赖关系图
        - 文本分析根据注释和标识符提取出概念依赖性
        - 历史分析从多个软件演进版本中挖掘出信息

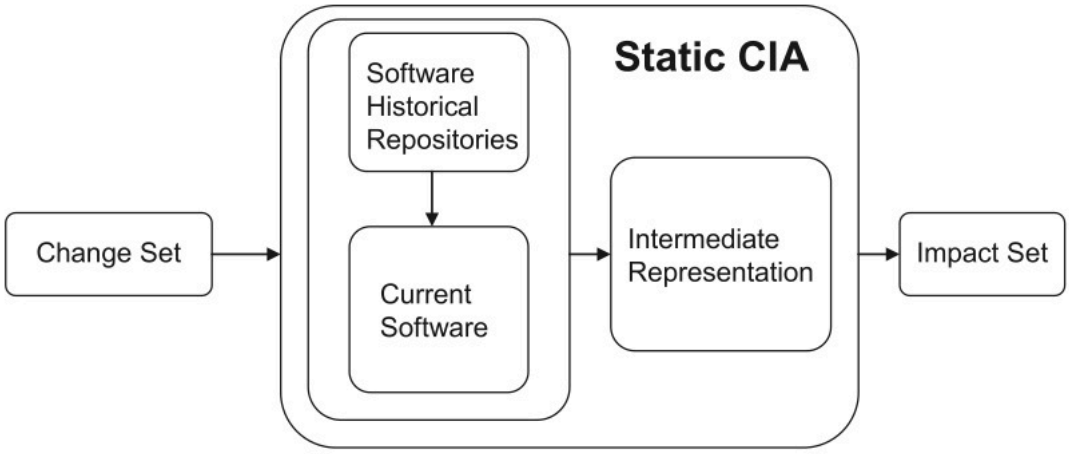


图 2.1　静态分析过程

- 动态分析：在线分析和离线分析。考虑特定的输入，并且依赖于程序运行时收集到的信息进行分析（如运行时的追踪信息、覆盖信息等）。其得到的影响集合往往比静态分析京都更高，但其开销也相应更大，而且通常包括误报（false negative）。
  - 在线 CIA 使用程序运行时收集的信息实施所有的分析

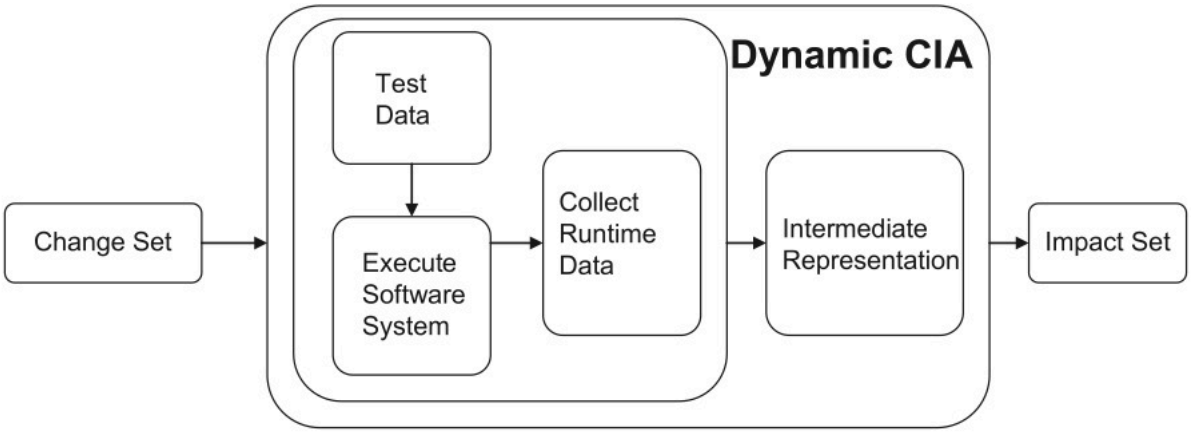


图 2.2　动态分析过程

近年来，有一些以 CIA 技术作为支撑而实现的工具，他们一般在实现 CIA 技术之后，利用得到的中间结果进行后续分析，帮助进行软件维护和演进。下面将对他们进行简要的介绍：

1. Chianti：应用于 Java 程序，Eclipse 插件

   应用场景：

   (a) 使用回归测试来看变更以前的功能是否能正常使用

   (b) 若失败了，利用 Chianti 进行变更影响分析，将 change 拆分成若干原子

change，并分析他们之间的依赖关系，结合原程序生成中间表示形式，看可能是哪些部分影响到了 test 的运作

网址：http://aria.cs.vt.edu/projects/chianti/ Ren X, Shah F, Tip F, et al. Chianti: a tool for change impact analysis of java programs[C]//ACM Sigplan Notices. ACM, 2004, 39(10): 432-448.

2. JRipples：应用于 Java 程序，Eclipse 插件
   应用场景：利用 dependency graph 等，自动标注可能受到被变更 class 影响的其他 class，并展示出对应的变更传播路径。可利用人工标注来修正结果网址：http://jripples.sourceforge.net/
3. ROSE: 应用于 Java CVS 工程，Eclipse 插件
   应用场景：挖掘软件代码仓库，当用户对代码变更时提示用户可能相关的变更（类似于“变更了这个函数的人通常还变更了另一个函数”）网址：https://www.st.cs.uni-saarland.de/softevo/erose/
4. jpf-regression: 应用于 Java 程序，Eclipse 插件/命令行工具
   应用场景：利用程序切片技术进行 change impact analysis，利用得到的 impact sets 来驱动 symbolic execution，获取被改变部分代码的行为（path condition）

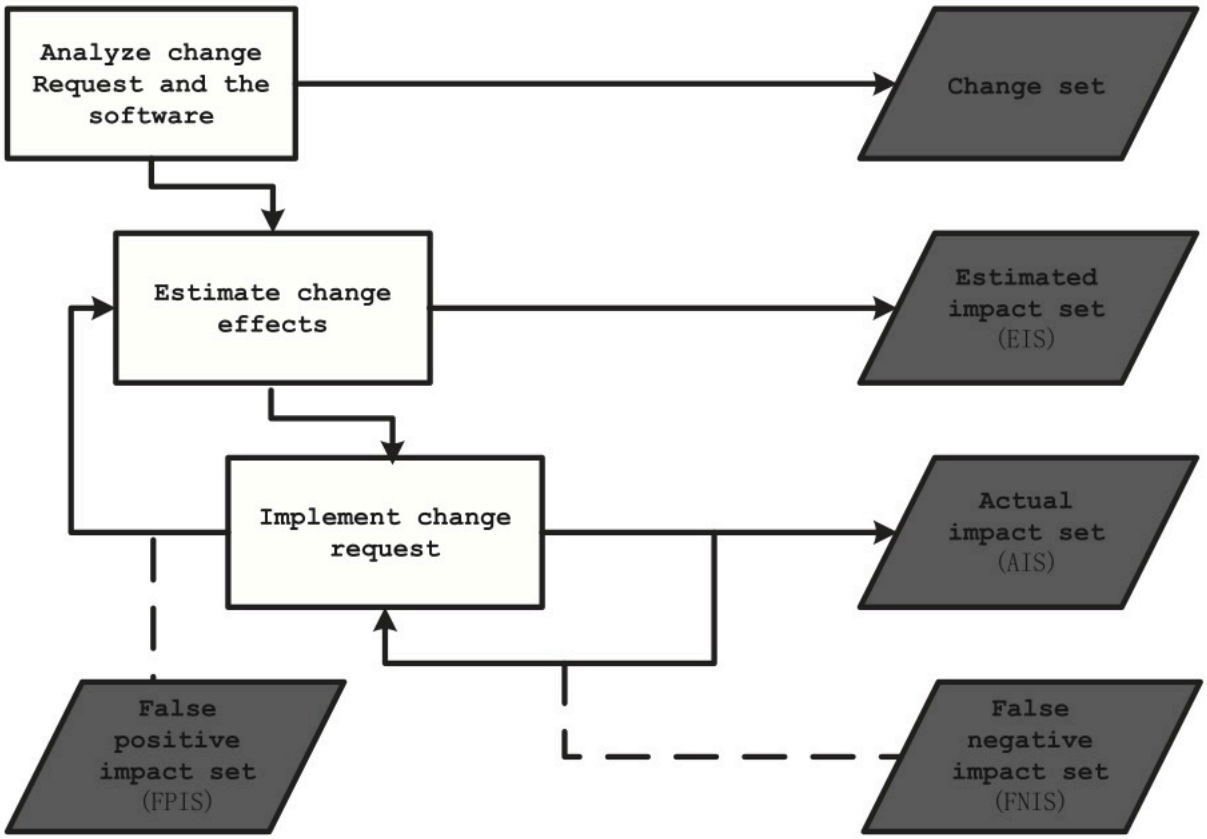


图 2.3　CIA 过程

## 2.4　相关工具

本节将主要介绍本文中采用到的相关工具。

### 2.4.1　git

git 是一个分布式的版本控制系统，最初由 Linus Torvalds 在 2005 年为 Linux 内核而开发，现在已经成为最流行的版本控制系统。

与 CVS 和 SVN 等集中式的 C/S 版本控制系统不同，git 是分布式的版本库，每个本地的 git 工作目录都包含了完整的历史数据和版本追踪能力，无需网络连接或服务器端。

本文中主要考虑以 git 作为版本管理系统的应用场景，类似于 GitHub，假定为项目开发了 new version 和 patch version 两个不同的分支，并使用 git 的分支合并策略实现补丁的版本迁移过程。

### 2.4.2　Beyond Compare

Beyond Compare 是一款内容比较工具，可以用于文件、目录、压缩包的比较，横跨 Windows、Mac OS X、Linux 三大操作系统，可用作版本控制系统的文本比较和合并工具，例如 git。

本文中主要采用其作为 git 的文本比较和合并工具，用于解决补丁版本迁移时的冲突（conflict）问题。

### 2.4.3　jpf-regression

变更影响分析常用于衡量软件变更的潜在影响。CIA 的结果通常可用做其他程序分析技术的输入，例如回归测试可利用 CIA 来确定程序的哪些部分需要进行再分析。而由于软件开发过程的演进特性，并且软件系统的大小和复杂性越来越高，刺激了人们对于有效的变更影响分析的需求。

一般而言，“单行变更”就足以引发广泛而未知的影响，因而变更影响分析在软件演进和维护过程中扮演着重要的作用。

目前，大部分的自动分析工具都将变更的影响以程序句法结构的形式表现出来，例如函数和程序语句。基于依赖的分析方法则通过分析程序组件间的内部关系来衡量变更的影响。这类技术能够以程序位置信息来描述变更的影响，然而却缺失了与受影响位置相关的程序运行路径的信息，而这类信息往往对于程序行为的验证、调试等起到很大的作用，而且能够将关注的范围缩小，只关注受到影响的程序行为集合。

（Directed Incremental Symbolic Execution ）DiSE 能够结合静态分析的效率和符号化执行的精度，分析方法内部的变更影响，并描述程序变更对其运行行为的影响。

DiSE 的静态分析部分采用程序切片技术来衡量变更对代码中其他位置的影响，其生成的影响集合可以用于引导符号化执行去分析受到影响的程序行为，并生成对应的路径条件（path condition），这些路径条件用于描述受到影响的程序行为。这些路径条件之后可以利用 SMT 等技术进行求解，用于后续的验证、调试等过程。

在 DiSE 方法中，程序变更影响分析是其进行后续分析过程的基础，它的变更影响分析过程主要特点在于：

1. 粒度：语句级别。即具体的影响分析过程中，以程序语句为基本单位进行影响集合的计算。
2. 影响范围：方法内部。即以方法的作用域作为单次影响集合计算的限制范围，最后得到变更对其所处的方法内部其他语句所造成的影响。也就是说不计算变更对于其他方法的影响，即不考虑对方法内部函数调用的影响。
3. 影响来源：主要从控制流和数据流两个层面考虑变更所造成的影响，实际上采用了语句间的控制依赖和数据依赖关系进行计算。

DiSE 中，受影响的集合主要分为两类：

- ACN: 受影响的条件语句节点（affected conditional nodes）。
- AWN: 受影响的赋值语句节点（affected write nodes）。

为了计算得到这两类影响集合，DiSE 中一共使用了四条规则来迭代计算，从最初的变更集合出发，不断应用规则向外扩展，最后计算得到闭包，即为所求的影响集合：

1. 如果 ACN 中有一个节点 ni，且存在一个条件节点 nj，其中 nj 控制依赖于 ni，那么将 nj 加入到 ACN 中。
2. 如果 nj 是一条赋值语句，且控制依赖于 ACN 中的节点 ni，那么 nj 加入到 AWN 中。

   前两条公式表明，如果条件语句和赋值语句控制依赖于变更后的 CFG 中的节点，那么他们就应当被加进来。
3. 如果 AWN 中的一条赋值语句节点 ni 对于变量的赋值在条件语句 nj 中被使用了，且 CFG 中存在一条从 ni 到 nj 的路径，那么将 nj 加入到 ACN 中。
4. 如果一个写语句节点 ni 对于对于变量的赋值在 AWN 或 ACN 中的某个节点 nj 被使用了，且 CFG 中存在一条从 ni 到 nj 的路径，那么将 ni 加入到 AWN 中。

jpf-regression 是 DiSE 的工具实现，它是基于 Java Path Finder 框架而实现的工具，可作为 Eclipse 的插件使用，支持 Java 语言。

### 2.4.4　ASTro

本文中所采用的 AST 比较工具是由内布拉斯加大学林肯分校的 Josh Reed，Suzette Person 和 Sebastian Elbaum 等人所开发的 ASTro，它也是 jpf-regression 中使用的前置工具，用于对比两个不同版本的源代码，并获取其抽象语法树上的差异，以 XML 文件的格式进行输出。该工具支持 Java 语言。

# 第 3 章　分析方法

## 3.1　问题定义

该问题可以拆分为若干个子问题，并分别阐述并定义之：

### 3.1.1　补丁版本迁移

### 3.1.2　变更的语义影响分析

### 3.1.3　兼容性分析

## 3.2　应用场景

## 3.3　解决方案

介绍补丁兼容性分析方法的整体架构：补丁版本迁移、语义影响范围分析（程序差异性分析、程序变更影响分析）、补丁兼容性分析。

## 3.4　流程

介绍补丁兼容性分析的整体流程:【可给出一张抽象的流程图】。

### 3.4.1　补丁版本迁移

### 3.4.2　语义影响范围分析

#### 3.4.2.1　程序差异性分析

#### 3.4.2.2　变更影响分析

### 3.4.3　实际流程

1. 采用 git 进行版本管理。
2. 将新版本和补丁版本两个分支进行合并，得到应用于新版本的补丁版本。
3. 采用 AST Differ 生成程序间差异性文件，XML 格式.
4. 采用 jpf-regression 进行变更影响分析，获取三个版本间两两的影响集合，dot 格式
5. 对两个影响集合求交集，若无交集，则兼容之，若有交集，进一步采用分析算法，看是否确实不兼容

【此时可给出一张具体的流程图】

# 第 4 章　组合方法

介绍如何将版本管理、程序间差异分析、变更影响分析、兼容性分析算法组合成整体的流程。

# 第 5 章　兼容性分析算法

介绍具体的兼容性算法，如果两个影响集合存在交集，是否确实是不兼容的。

# 第 6 章 实验

以 Eclipse jdt core 为例，进行实验，并分析得到的实验结果。

## 6.1 补丁版本迁移

## 6.2 语义影响范围分析

### 6.2.1 程序间差异性分析

### 6.2.2 变更影响分析

## 6.3 兼容性分析

## 6.4 结论

# 第 7 章 结论

## 7.1 工作总结

## 7.2 未来工作

# 参考文献

# 致　谢

感谢贺飞老师对我的悉心指导，他的言传身教令我受益匪浅，并在我完成毕设工作时给出了许多有用的意见。

最后要感谢 ThuThesis，帮助我完成了本文的写作。

# 声　明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签　名：_____ 日　期：_____