可擴充支援任意程式語言之版本紀錄追蹤輔助工具
Code Change History Tracing Tool for Arbitrary Programming Language

林庭書、鄭永斌
國立中央大學資訊工程學系
Ting-Shu Lin, Yung-Pin Cheng
Email: 103525003@cc.ncu.edu.tw, ypcheng@csie.ncu.edu.tw

摘　要

現今的版本控制系統及相關工具，因為所儲存的資訊缺乏對程式原始碼的理解，而導致軟體開發人員在對版本紀錄進行查詢時，難以得到他們實際所關心的資訊，例如原始碼內某一程式元素的變更歷程等。

本論文針對現有系統的不足，提出一個輔助追蹤版本歷史紀錄之工具 iNob。本工具能以高效的方式對程式原始碼進行簡易的分析，以讓軟體開發者以方法(method)為單位，查找其於版本控制系統(Version Control System)中的變更紀錄。使用者可以透過簡單而直觀的操作，得到出他們所關注的變更紀錄。透過這樣的輔助，能幫助開發者更加有效地除錯以及瞭解程式碼變更歷程，從而提高開發效率並降低程式維護成本。本工具同時也有著高度的擴充性，能夠藉由簡單的方式更換部分元件，以支援不同的程式語言。

關鍵字：VCS、Code History Analysis、Git

一、緒論

當代軟體專案的開發，多會使用版本控制系統來進行版本之管理並幫助團隊合作。而如何善用版控系統中的資訊，來幫助程式之開發與維護也就成為一個重要的問題。

當今被廣泛使用的知名版本控制系統如 Subversion[1]和 Git[2]等，因為泛用性與實作成本之限制，皆僅記錄純文字資訊，並以檔案為單位儲存。而如此簡便的特性也必然的帶來一定的缺陷：使用者在查看版本紀錄時，僅能看見純文字及檔案層面的資訊，而無法從程式結構(類別、方法等)的角度去搜尋檢索他們所關心的變更紀錄。

軟體開發者去查詢版本紀錄主要有以下兩種時機：1. 在看到一段難以理解的程式碼時，藉由了解程式碼變更的過程，來得知為何程式會有現在的樣貌。2. 在對某個單元除錯時，回溯與該單元相關的變動，以找出與產生錯誤的更動紀錄。

不幸的是，由於現有的版本控制系統純粹是基於文字及檔案做紀錄，我們只能找出「某個檔案的變動紀錄」或是「某段文字的變動紀錄」。若以前者來檢索，會找出大量同樣在該檔案中，卻與目標單元無關的變更記錄。而後者雖然稍有助於減少無關資訊，但因為變更中可能有複雜的修改搬動，所搜尋出來的也可能多屬無關內容。

因為版本控制系統發展至今仍未克服上述問題，在許多過往的相關研究之中，也對於如何從版本歷史紀錄的純文字資料中，分析獲得更加有用的資訊做了許多努力。然而相關的研究始終面對著兩個主要的問題，一是所分析出來的資訊是否能對使用者有真正的助益，二是在分析龐大的原始碼紀錄時，漫長的耗時始終是一核心的困難問題。

本研究由軟體開發者的角度出發，根據開發經驗，提出並實作了一個能夠幫助軟體開發者在開發與維護過程中，迅速而有效地檢索版本紀錄之輔助工具 iNob(acronym for “iNob is Not Only Blame”)。我們將 iNob 設計為 IDE 的插件(plugin)，讓使用者可以在習慣的介面中，以直觀方便的途徑使用之。使用者可以透過 iNob 搜尋出與程式中某方法(method)相關之變更紀錄，同時也能在查看變更紀錄時，看到有哪些相關聯的方法受到更動。這樣的搜尋方式不但可以有效的減少無關資訊的干擾，讓使用者能專注在特定方法相關之變更上，還能藉由觀察相關變動察覺到程式之耦合關係。

在設計上，iNob 將對檢索版本控制系統的功能與對程式原始碼之分析器(parser)分離，使分析器成為一個可抽換的元件，因此可以藉由抽換分析器而處理不同的程式語言。另一方面， iNob 對於分析器的需求僅是尋找出特定的程式區塊，而不要求分析出程式的正確結構。相較於傳統編譯器所使用之 Parser，iNob 的原始碼分析器不需要辨識出程式內的所有 Token，也不需要由語法規則完整的建立 AST。因此 iNob 的分析器在效能方面可以比傳統的編譯器或靜態分析工具有更好的表現。

二、背景與相關研究

版本控制系統是當代軟體開發過程中必然使用的軟體工程輔助工具。小至個人專案，大至作業系統及商業產品之開發，背後都使用版本控制系統來確保程式版本的可回復性，同時也是讓多人團隊能夠同時協力開發的必要工具。

在我們的開發經歷以及研究室的專案維護中，也頻繁的使用版本控制系統作為輔助工具。然而在使用的過程中，我們發現到現今的版本控制系統仍有諸多不足之處，無法便利地讓開發者取得所需要版本變更資訊。其主因為現今之版本控制系統所儲存的資料皆基於「檔案」與「純文字」系統，而非開發者所最關注的「程式結構」。亦即，版本控制系統可以告訴開發者在某個版本中有哪些「檔案」

被新增、刪除和變更，或是某個檔案內新增或刪除了幾行，但是無法確切告訴開發者有哪些類別(Class)或方法(Method)等程式元素受到影響。而實際使用情境上，開發者在追溯版本控制系統歷史資訊時，所想要知道的往往是後者。

以 Git 版本控制系統為例，當開發者想要找出某段程式相關的變動紀錄時，最常用的兩種工具應為「gitk」及「git blame」指令。開發者可以使用 gitk 查詢關於特定檔案的變更紀錄，其使用畫面如圖 1。我們可以看到上方有諸多版本的線圖、作者及修改時間等資訊，下方則有選取版本的變更紀錄，顯示該版本增減了哪些程式碼。這幾乎包含了版本控制系統關於該檔案所有能給予的資料，並以較為友善的圖表呈現，然而這樣的資訊對於開發者的助益卻仍然有限。其原因在於它給予了開發者過多的資訊。請注意我們在此查詢到的是關於整個檔案的變更紀錄，而非單一程式元素(例如一個方法)的更動。一個檔案中可能包含了數十個方法，而開發者關心的若只是其中一個，恐怕這份變更紀錄中的多數版本都不是其所需要的。而另外一個常用的指令「git blame」也有相似的問題，雖然可以將搜尋範圍限定到該檔案的特定幾行之間，但僅僅靠行數判斷並無法正確的追蹤程式的函式區塊，在歷經重構搬移等動作之後，基於程式行數的變動追蹤所找到的往往是錯誤的範圍。

造成上述困擾的主因，是稍前提及的版本控制系統之侷限。因為其本身並不具有理解程式語言的能力，僅僅將各種對程式變更的結果，紀錄為檔案及文字的變更，因而在需要查看程式內容的變更時，就表現得不盡理想了。

既然版本控制系統已成為現代軟體開發中的必要工具，而其所保存與提供的資訊又有明顯的不足，對於如何更加有效的了解與利用其所提供的資料，來提供實際有用的資訊，也就成為許多前人致力研究的方向了。

有一部分的研究是在對版本紀錄進行整體分析後，提供一份較為宏觀資訊報表或將得到的特徵作為預測未來的材料。Rongxin Wu 等人開發了 ReLink[3]系統，用資料探勘的技術，找出程式錯誤(bug)與版本變動間的關聯。也有人用較簡易的方法，如分析 commit log 中的關鍵字，來找出 bug 及版本之間的關聯[4-6]。而更進一步的研究則分析了版本控制系統中產生錯誤的部分特徵，用以預測可能造成錯誤的變更[7-9]。T. Zimmermann 等人則利用修改紀錄中的關聯性，來找出難以用一般程式分析工具找出的耦合關係[10]。

而也有另一個方向的研究，利用分析版本紀錄，來提供一些能讓開發者在開發過程中頻繁使用的輔助工具。Yun Young Lee 等人所開發的 Tempura 系統[11]，藉由分析版本控制系統中 API 層級的資訊，將已變更或刪除的 API 資訊在 IDE 的 autocomplete 功能中呈現，讓使用者可以找到其慣用的舊 API，並從其變更版本中了解程式的變更以快速地了解和適應新的程式。YoungSeok Yoon 等人實作之工具 AZURITE[12]透過查看版本紀錄及監控使用在 IDE 的操作，視覺化程式的變更紀錄，以幫助了解程式碼的變更歷程以進行還原(undo)或重做(redo)。微軟在 Visual Studio 2015 企業版中，提供了連結版本控制系統查看特定方法之變更紀錄的功能，不過只侷限於數種語言，並缺乏更加豐富的關聯資訊。

在我們的研究中，最重要的挑戰有二：以怎樣的方式呈現變更紀錄對使用者而言是最有用而易懂的，以及如何能夠給使用者足夠迅速的回饋。關於第一點我們借鑑了 SourceTree[13]和 Git GUI 等知名的 Git 圖形介面，以貼近使用者習慣。在效能方面，前述 Yun Young Lee 等人之研究也曾提及對各版本總和大小 4.6Gbyte 的 Java 原始碼進行完整分析即須七百餘秒，這顯然是令使用者難以忍受的漫長時間。我們將在系統實作中，說明更加靈活的解決方案。

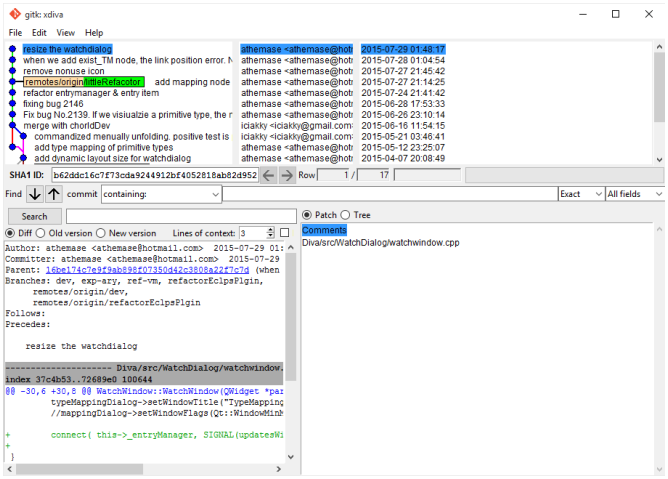

圖 1 gitk 之介面

三、系統實作

本系統由實用角度出發，力求易用與快速，並提供良好的擴充性，以面對將來支援其他程式語言之需求。我們將 iNob 實作為知名 IDE Eclipse[14]的 plugin，讓使用者能在習慣的介面中透過簡易的操作使用之。iNob 透過 JGit[15]檢索 Git 版本控制系統中的紀錄，交由自訂的語言分析器解讀原始碼結構，在分析完成後將視覺化的變更紀錄呈現給使用者。

3-1 User Scenario

我們可以從一個簡單的使用情境瞭解 iNob 的使用方式：系統維護者 John 收到告知，在 Foo 類別的 method bar()行為不符合預期，John 知道這個功能以前是正確運作的，認為是在最近的修改中產生了錯誤。於是 John 在 Eclipse IDE 中打開 Foo.java 這個檔案，他找到 method bar()之後將之選取，在右鍵點擊的選單之中呼叫 iNob。iNob 迅速地將所有曾變動 method bar() 的紀錄顯示出來，包含日期、作者與 commit log 等資訊， John 可以

滑鼠點擊各個紀錄，並在下方看見詳細的內容變動，如圖 2。如此就能較高效的尋找可能埋藏錯誤的更動。此外，當 John 想知道其中某次的版本變更中，還有哪些其它檔案受到更動時，可以右鍵點擊關注的版本，來查看相關的變動內容，如圖 3。

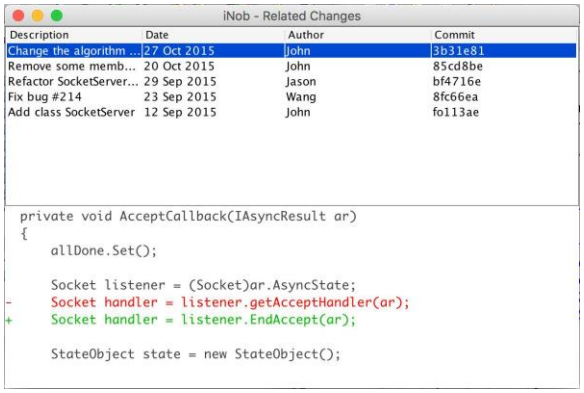

圖 2 iNob 查詢結果主畫面

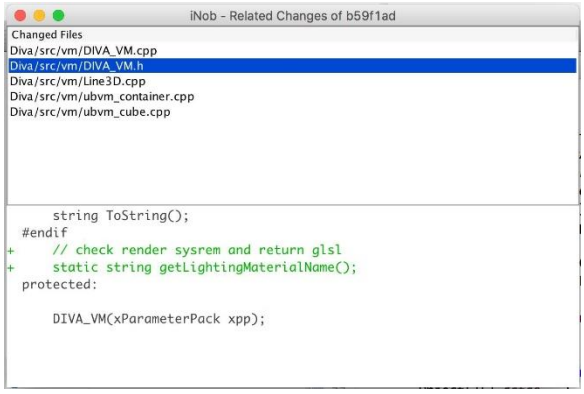

圖 3 iNob 同版本關聯變動檢視畫面

3-2 系統架構

根據前述情境，iNob 的三項重要能力為：

1. 解讀特定語言之程式碼，以判斷指定方法在程式碼檔案中之位置。
2. 操作版本控制系統，檢索檔案之變更紀錄，結合前項功能即可追蹤特定程式元素的變動。
3. 將前二項取得的資訊透過良好的介面呈現給使用者查看。

我們將這三項功能個作為一個 iNob 的組成元件，分別命名為：Source Code Analyzer、History Tracer 與 User Interface。總和其架構如圖 4。

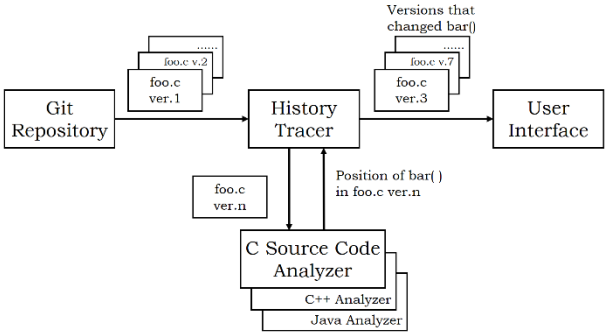

圖 4 iNob 架構圖

在本章後續會詳細說明各個元件，在此先簡述之。Source Code Analyzer 為一可抽換之組件。在要擴充對其他程式語言的支援時，可依制定之介面與 History Tracer 溝通。而 History Tracer 為負責追蹤歷史版本之主要程式邏輯部分，透過 JGit 存取 Git Repository 中的紀錄，並交由 Source Code Analyzer 分析。User Interface 為讓使用者操作及瀏覽的介面。我們將這項功能設計為 Eclipse 的 plugin，可用以取得使用者輸入並傳訊至 History Tracer 請求分析資料，並將結果呈現給使用者。以下將對各個元件的功能與設計進行詳細說明。

3-2-1 Source Code Analyzer

Source Code Analyzer 的設計為 iNob 靈活性的關鍵。一般會直覺地認為這部分的工作相當於編譯器之剖析器(Parser)的部分。不過 iNob 並沒有規定 Source Code Analyzer 應該要如何去分析程式，只要能夠正確的查詢方法區塊的位置，並將之回傳給 iNob 即可。在擴充這部分的功能時，可以依照需求自行設計。

我們所實作的兩種分析器正好展現了這方面的靈活度。我們做出的第一個 Java 分析器基於實作成本考量下，直接呼叫了 Eclipse 所提供的 JDT ASTParser 來分析原始碼，再取得所求的方法區塊。這個方法的優點是正確性無虞，而且透過現成的 API 呼叫，實作成本也很低。其缺點在於分析程式時，完整的建構了整個 AST(abstract syntax tree)，在效能上有多餘的花費，在稍後會講述效能方面的成本問題。

從我們的需求來看，如同一般 Parser 完整地建構程式的 AST 是沒有必要的。標準的編譯器 Parser 在這個過程中，需要具備所有的語法規則，識別所有的 Token，在分析的過程中匹配正確的對應規則逐層建構出 AST。然而 iNob 需要的只有找出特定方法的區塊位置，這個方法以外的程式碼，乃至方法內的程式意義都是無須理會的。在此需求之下，其實只需要一個用一個非常簡易的字串模式配對，找出指定方法所在位置，並將區塊上下限找到即可。我們在實作 C++分析器時，就採用了這個簡單而快速的方法。分析器所接收到的查詢是一個 method signature，亦即一個 method 的名稱、參數類型及回傳型別。依照這幾項資訊，即可建立一個正規表達式，在程式原始碼中進行配對。在忽略註解與字串中的大括號後，即可簡單地以方法後方成對的大括號找出程式區塊。

值得注意的是，iNob 並不要求也不保證 Source Code Analyzer 的分析結果在任何情況下都完全正確。以前述 C++分析器之例，最極端的情況可能專案中有一個 header file 以 macro 定義「BEGIN」為左大括弧，「END」為右大括弧，並將所有的程式區間用 BEGIN 和 END 框起。而因為分析器只對單檔分析，自然難以對付這個狀況。

往往越完美越正確的分析器，就需要知道即處理越多資訊，將對的也就越慢。

iNob 在 Source Code Analyzer 方面的設計目標是給予使用者更多的選擇。當使用者需要其他程式語言的支援時，可以換上該語言的分析器。當使用者對於我們較注重效能的分析器有所不滿時，也可以選擇更換為較為注重正確性的分析器。在有特殊需求無法被現有工具滿足時，甚至也可以自行撰寫程式接上 iNob 的介面即可。

由於 iNob 目前是作為 Eclipse 的 plugin，當使用者要替換的時候，是使用 plug-in fragment 的功能替換。未來我們也計畫開放其他形式的介面，以讓非 Java 的程式也可以簡易接上 iNob。

3-2-2 History Tracer

History Tracer(以下簡稱 Tracer)的功用是用來操縱 Git 系統以獲得歷史紀錄中的檔案，將之送交 Source Code Analyzer 分析之後，整合分析完的資料交由 User Interface 呈現。其工作流程簡述如下：當收到對於一個方法的查詢(包含 method signature 與檔案名稱)時，History Tracer 會啟動 Git Adapter，並透過 Git Adapter 查詢版本控制系統中的資料，以找出所有與該檔案相關的變動。在抓取到所有與該檔案相關的變更紀錄之後，逐一分析該次的變動是否有針對目標 method 進行修改。在篩選出所有與目標 method 相關的變動之後，就可以把這些修改紀錄送交給 UI 讓使用者看到了。

在此處值得注意的一個要點是，這個流程為了提高檢索效率，而做了一個假設：「所有關於該指定 method 的修改都位於同一個檔案之中」。這個假設條件，將搜尋範圍由整個專案縮限到了單一檔案，是讓 iNob 能夠做出即時回應的重要條件。實際上，這個假設並非對任何程式語言都適用。有些程式語言的一個類別可以寫在多個檔案之中，例如 C#、Ruby 等，甚至較傳統的語言 C++也可以將方法的實作寫在任意檔案之中。這樣的例外情況在設計時雖也有列入考量，但實作時依然決定維持此假設。原因是本假設雖然可能出現例外之情況，但在大多數有受到規範的軟體開發中，將類別內的方法在不同檔案之間搬移的做法是很少見的。倘若要為了如此鮮見的例外情形，放棄對單檔的檢索而改為完整搜索整份專案，在大型專案之中的執耗時將會是百千倍以上，可謂因小失大。

3-2-3 User Interface

在 User Scenario 一節中，我們已經展示過 iNob 的主要畫面。這個設計參考 Unix 系統上常用的 diff 工具，及 SourceTree 等 Git GUI 工具，以人們所熟悉的傳統方式呈現原始碼的變動。使用者可以滑鼠點擊變更紀錄來查看詳細內容的變動，又或用鍵盤上下鍵在不同的變更紀錄間切換，以快速瀏覽變動歷史。而以右鍵點擊變更紀錄時，可以選擇查看在這個版本還有哪些檔案同時受到變更。

四、實驗評估

本章中，我們就 iNob 要投入實際應用的兩個必要條件進行實驗測試。其一，開發人員在有不同需求時，是否能透過 iNob 所提供的機制，自行擴增所需支援的程式語言。其二，使用者在使用本工具時，是否能夠獲得足夠即時的回饋。

4-1 對程式語言的擴充性

如同在本文 3-3 節所提及的，iNob 的開發團隊為了提供對程式語言 C++的支援，利用 iNob 的制定的介面，撰寫了一個簡易的 C++程式法分析器。在這個過程之中，並未更動 iNob 原有的程式碼，即完成了擴增支援其他語言的需求。也就是說，即使是對 iNob 程式內部一無所知的一般開發人員，也能夠以 iNob 所提供的方法，在不對 iNob 程式本身做任何修改的情況下，加入新的分析器以達新增支援程式語言的目標。

4-2 搜索耗時

在此實驗中，我們以本研究室的另一個大專案 xDIVA[16]作為測試的目標。xDIVA 是一個發展了十年，有七萬多行程式碼，版本控制系統中資料容量近 300MB 的大型專案。測試過程中，我們隨機挑選了 xDIVA 內的五個檔案，並於各個檔案內選取三個 method，使用 iNob 進行搜尋。實驗的結過，對上述所有 method 的搜尋，都能在一秒之內看到完整的搜尋結果。

五、結論

本研究針對現有版本控制系統之資訊不足，導致軟體開發者檢索歷史紀錄時之不便，提出並實作了一實用之輔助工具 iNob。藉著本工具，開發人員可以在熟悉的 IDE 環境中，經由簡易的點擊操作查詢到指定方法的變更紀錄，以幫助了解程式碼及尋找以前修改過程中產生之錯誤。由於 iNob 是為搜尋一方法變更紀錄而特別設計的工具，能利用搜尋目標皆位在同一檔案之內的假設而大幅減少所需耗時，相較於前人們對版本紀錄進行完整分析的研究，能更加即時的給予使用者回饋。而也由於採用了易於擴充的架構，只要透過更換分析器元件，就能夠支援其他各種程式語言。

經過實際的嘗試，iNob 確實可以簡單地透過更換程式碼分析器來達成對不同程式語言的支援。執行時間方面，在面對有七萬行原始碼與十年發展歷史之大專案，依然能夠即時的搜尋完成。可見 iNob 在支援對象及效能表現上都已有良好成果，應可於短期內投入實際應用。

未來展望

本研究是一剛取得初步成果的新專案，因此還有許多值得深入研究的發展空間。在支援廣度方面，本研究以 Eclipse 作為第一個實現平台，未來應考慮對更多 IDE 的支援，例如 IntelliJ IDEA 和 Visual Studio 等其他知名 IDE。在分析器的擴充性

方面，由於目前是使用 Eclipse plug-in fragment 作為替換方法，使得這個元件的擴充方式被侷限於 JVM 語言。若之後加上如 socket 等較為通用的溝通介面，應更有助於與其他的分析器之連結。

除了支援對象以外，目前 iNob 在查看一個版本紀錄的相關變動時尚有一明顯的遺憾。iNob 以追蹤程式內 method 的變更為目標，現在已可以偵測一個 method 的變更歷程，不過在查看相連變動時卻仍以檔名而非方法名的方式列出(見圖 3)。這是因為現在分析器的規格上只要求能夠的找出指定 method 的位置，而未要求能夠反過來查詢一段程式碼被包含在哪個 method 的區間。在加上這項要求之後，可以使得 iNob 的功能更加完整，然而也勢必增加實作新分析器之成本，將來是否應加入這項功能是值得考慮的。

最後，由於 iNob 處於剛開發完成的階段，尚未被開發團隊外的使用者所使用。iNob 所提供的功能對其他開發團隊是否足夠實用，還需要改進和增加那些功能等等，應於積極推廣投入實用後收集回饋，以持續精進之。

參考文獻

[1] *Subversion*. Available: https://subversion.apache.org/

[2] *Git*. Available: https://git-scm.com/

[3] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "ReLink: recovering links between bugs and changes," presented at the Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, Szeged, Hungary, 2011.

[4] A. Mockus and L. G. Votta, "Identifying reasons for software changes using historic databases," in *Software Maintenance, 2000. Proceedings. International Conference on*, 2000, pp. 120-130.

[5] A. Schröter, T. Zimmermann, R. Premraj, and A. Zeller, "If your bug database could talk," presented at the Proceedings of the 5th International Symposium on Empirical Software Engineering, 2006.

[6] T. Zimmermann and P. Weißgerber, "Preprocessing CVS Data for Fine-Grained Analysis," presented at the MSR'04, Edinburgh, Scotland, UK, 2004.

[7] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?," *SIGSOFT Softw. Eng. Notes,* vol. 30, pp. 1-5, 2005.

[8] A. E. Hassan and R. C. Holt, "The top ten list: dynamic fault prediction," in *21st IEEE International Conference on Software Maintenance (ICSM'05)*, 2005, pp. 263-272.

[9] S. Kim, T. Zimmermann, J. E. J. Whitehead, and A. Zeller, "Predicting Faults from Cached History," in *29th International Conference on Software Engineering (ICSE'07)*, 2007, pp. 489-498.

[10] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller, "Mining Version Histories to Guide Software Changes," presented at the Proceedings of the 26th International Conference on Software Engineering, 2004.

[11] Y. Y. Lee, D. Marinov, and R. E. Johnson, "Tempura: Temporal Dimension for IDEs," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, 2015, pp. 212-222.

[12] Y. Yoon, B. A. Myers, and S. Koo, "Visualization of Fine-Grained Code Change History," presented at the IEEE Symposium on Visual Languages and Human-Centric Computing, 2013.

[13] *SourceTree*. Available: https://www.sourcetreeapp.com

[14] *Eclipse*. Available: https://eclipse.org/

[15] *JGit*. Available: http://www.eclipse.org/jgit

[16] *xDIVA*. Available: http://oolab.csie.ncu.edu.tw/xDIVA/