2017 13th台灣軟體工程研討會

7/7-8, 逢甲大學

TCSE FCU 2017 13th

- 主辦單位 逢甲大學
- 大會日期 2017年7月7日(五)・2017年7月8日(六)
- 大會地點 台中逢甲大學
- 承辦單位 逢甲大學資訊工程系
- 指導單位 社團法人台灣軟體工程學會
  教育部資訊及科技教育司
  行政院科技部工程科技推廣中心

大會序言

「台灣軟體工程研討會」(Taiwan Conference on Software Engineering, TCSE) 是台灣提供給產官學專家與學者一個探討與發表軟體工程原理、實務與最新技術研究的園地，多年來經諸位先進積極參與，TCSE 已成為國內首屈一指的軟體相關會議。逢甲大學資工系曾經於 2007 年承辦第三屆會議，很榮幸十年後再次承辦第十三屆的研討會。

2017 年 TCSE 之大會主題為「*Software Engineering and Data Science*」，探討軟體工程與資料科學的相關研究，包含應用軟體工程技術於資料科學及應用資料科學於軟體工程。本年度共有 83 篇投稿，最後收錄 75 篇論文，涵蓋 Software Engineering, Data Engineering, Software Testing, Web Engineering, Artificial Intelligence, Internet of Things, Software Application 等研究與實務議題。

今年共有兩場大會主題演講。第一天特別邀請到美國 University of Texas at Dallas 的 Prof. Lawrence Chung 主講 Big Data Analytics: A Requirements Engineering Perspective，從需求工程的角度來看資料分析。第二天邀請中研院研究員，同時也是台灣資料科學協會理事長的陳昇偉博士主講「從大數據到人工智慧」，亦是非常精采的演講。此外，今年的專題論壇（Panel Discussion），主題為 Software Engineering and Data Science，由北科大鄭有進教授擔任論壇主持人，與談人包含台大資工李允中教授、Prof. Lawrence Chung 與帆宣系統科技韋建名副總等來自學術界與產業界的頂尖專家，透過此場合提供一個產官學界的交流平台，共同研討軟體工程與資料科學時代之整合與應用。此外大會亦舉辦三場業界的專題演講，從實務的角度提供不同的思考視野。

去年的 TCSE 研討會在國立海洋大學舉辦，首次開設了 English session，鼓勵以英文撰寫並報告論文，獲得廣大迴響，優秀作品並推薦到 JIT (Journal of Internet Technology) 國際期刊刊登。今年 English paper 篇數成長至 12 篇，我們與 JISE (Journal of Information and Engineering) 國際期刊合作，將優秀論文推薦到 JISE 的特刊。

為了鼓勵將軟體工程結合資料分析並鼓勵動手做，我們特別與 OpenEdu 線上學習平台及 ITSA 社群運算與巨量資料跨校資源中心合辦資料分析的黑客松競賽（Hackathon）。這也是 TCSE 首次舉辦黑客松競賽。OpenEdu 是台灣主要 MOOC (Massive Open and Online Course) 平台之一，提供優質且免費的線上課程。學習者的學習行為被完整的記錄在系統中，透過分析這些學習行為可以改善日後的線上學習方式。今年共有 6 校 13 隊 45 位同學參與初賽，通過者將於大會開始之時進行複賽，也感謝多家廠商的贊助，讓活動得以順利舉辦。

本次大會能順利舉辦，除了逢甲大學校方與台灣軟體工程學會的支持之外，同時也要感謝教育部資訊及科技教育司、科技部工程處工程科技推廣中心等單位的協助。對於議程主持人、議程委員、最佳論文評審所付出的精神與心力也要致上最大的感謝！此外也謝謝逢甲大學資訊工程系所有參與研討會籌備的教師以及工作人員，由於大家的群策群力才能逐步完成本次大會的規劃與目標，在此也一併致謝。最後，也敬祝 2017 年大會圓滿成功，希望與會嘉賓都能有一個愉快且有收穫的學術旅程！

大會主席： 許芳榮、楊東麟

議程主席： 薛念林、陳錫民

2017 逢甲大學

目錄

大會資訊

主辦單位 逢甲大學

大會日期 2017 年 7 月 7 日 (五)- 2017 年 7 月 8 日 (六)

大會地點 台中逢甲大學

承辦單位 逢甲大學資訊工程系

指導單位 社團法人台灣軟體工程學會
教育部資訊及科技教育司
行政院科技部工程科技推展中心

大會網站 http://tcse2017.iecs.fcu.edu.tw

投稿系統 https://easychair.org/conferences/?conf=tcse2017

社群網站 Facebook- 台灣軟體工程研討會

聯絡方式 信箱：mailto:tcse2017.fcu@gmail.com
電話：(04) 2451-7250 ext 3704

論文集網址 https://goo.gl/mttPFD (2017/8/31 前有效)

大會組織

大會榮譽主席

李秉乾　逢甲大學校長
竇其仁　逢甲大學資電學院院長

大會主席

許芳榮　逢甲大學資工系主任
楊東麟　逢甲大學資工系教授

議程主席

薛念林　逢甲大學資工系
陳錫民　逢甲大學資工系

指導委員

主席：　李允中

朱正忠	朱治平	余孝先	李漢銘	李健興	周忠信	林哲正	梁德容
盛敏成	莊育秀	郭耀煌	郭譽申	陳俊良	陳建村	陳英一	陳偉凱
黃世禎	黃為德	葉道明	劉立頌	劉瑞隆	劉龍龍	鄭永斌	鄭有進
鄭炳強							

特別議程主席

軟體展示:	李信杰	成功大學資工系
企業演講:	許懷中	逢甲大學資工系
黑客松:	黃奕欽	逢甲大學資工系
區域安排:	徐國勛	台中教育大學資工系
英文論文:	陳奕中, 英家慶	逢甲大學資工系

議程委員

丁培毅	王凡	王建民	王秉豐	王能中	孔令傑	孔崇旭	朱威達
李文廷	李俊宏	李強	李祈鈞	呂學展	沈文祥	江季翰	吳牧恩
吳秀陽	吳政瑋	吳漢銘	洪國鈞	林志敏	林文揚	林易泉	林峰正
林迺衛	林楚迪	范姜永益	馬尚彬	金台齡	孫培真	張文貴	張志宏
張詠淳	須上英	梅興	許乙清	許子衡	許見章	許煜亮	連耀南
郭忠義	郭家旭	童曉儒	陳立偉	陳炳文	陳恭	陳振炎	陳伶志
陳景祥	陳宗禧	黃冠寰	黃建宏	黃衍文	黃福銘	黃慶育	黃惠俞
黃俊穎	楊士萱	楊中皇	楊豐兆	詹大千	賈坤芳	廖峻鋒	廖珙洲
熊博安	劉豐豪	劉建宏	劉傳銘	劉正山	潘健一	蔡敦仁	鄭伯壎
賴聯福	錢炳全	鍾毓驥	謝仕杰	謝孟諺	謝政勳	藍啟維	羅榮華
戴志華							

大會議程

地點：逢甲大學 學思樓

	7 月 7 日			
8:30	報到			
9:20	大會開幕: 第九國際會議廳			
9:30	大會演講 I：*Big Data Analytics: A Requirements Engineering Perspective* [m] Prof. Lawrence Chung			
10:20	Coffee & Tea Break			
10:40	1A: SE I [a]	1B: Test I [b]	1C: Web [c]	1D: AI [d]
12:00	Lunch & 軟工學會會員大會			
13:20 14:50	2A: Best English [a]	2B: English [b]	2C: IoT [c]	2D: Industry talk I [m]
15:00	3A: Best regular [a]	3B: Test II [b]	3C: Demo [c]	
16:30	Coffee & Tea Break			
16:50 17:50	專題論壇：*Software Engineering and Data Science* [m] 鄭有進教授, 李允中教授, Prof. Lawrence Chung, 韋建名博士			
18:10	晚宴：星饗道餐廳			
	7 月 8 日			
9:00	報到			
9:30	大會演講 II：*從大數據到人工智慧* [m] 陳昇瑋 博士			
10:20	Coffee & Tea Break			
10:40	4A: Data [d]	4B: SE II [b]	4C: Application [c]	4D: Industry talk II [m]
12:00 12:30	TCSE Hacks 頒獎 [m]			

Room: [m] 第九國際會議廳, [a] 學 101, [b] 學 102, [c] 學 103, [d] 學 104。

分場詳細議程

1A: Software Engineering I

Chairman: 林哲正/高雄師範大學，時間: 10:40-12:00, 7/7，地點: 學思樓學 101

ID	作者	論文名稱
6	Shuo-Hong Kao and Dow-Ming Yeh	程式語言語法視覺化工具雛型
38	呂信緯, 陳英一	以 Nodejs 非同步設計框架建構語意分析加值系統之研究
58	郭忠義, 許聖泉	程式碼動態結構抄襲鑑定
70	薛念林, 莫剛, 陳錫民	OpenEdu 磨課師系統學習資料分析報告
69	Chang-Yen Lee, Hui-Juan Chen and Che-Chern Lin	從設計與實作觀點探討模糊專家系統在補救教材的應用 – 以高職數位邏輯課程為例

1B: Software Testing I

Chairman: 林楚迪/嘉義大學，時間: 10:40-12:00, 7/7，地點: 學思樓學 102

ID	作者	論文名稱
17	Wan-Chuan Lee and Yung-Pin Cheng	運用壓力測試腳本的同步來增進壓力測試效能
21	薛念林, 黃紫芳	測試驅動之設計樣式測試模型設計與實作
47	張振鴻, 林迺衛	基於限制邏輯圖的單元測試案例產生器
59	郭忠義, 彭柔瑄	Android 應用程式之資訊安全檢測
14	劉建宏, 陳偉凱, 黃映瑞, 林容榆	Android 手機手錶互動應用程式之相容性測試研究

1C: Web Engineering

Chairman: 范姜永益/輔仁大學，時間: 10:40-12:00, 7/7，地點: 學思樓學 103

ID	作者	論文名稱
3	Kao Pin, Huo Kuan-Hua, Chang Yi-Tzu, Cheng Yo-Tzu and Hu Chung-Hua	建構商轉雲端管理平台以應用於異質虛擬化環境之雲資源自動移轉與配置
4	Chih-Hung Chang, Chih-Ming Hsieh, Wen-Ching Chen, Y.J. Liao and C.M. Wang	具情境感知之個人化資訊服務框架
15	陳偉凱, 劉建宏, 黃映瑞, 陳科銘	以漸增式使用者指引增加爬蟲器之網頁覆蓋率
27	石聖銓, 范姜永益	應用遺傳規劃法於 Web Services 的服務品質預測
39	Li-Wei Huang and Ing-Yi Chen	運用 Express 中介軟體框架設計非同步責任鏈網路服務之研究

1D: Artificial Intelligence

Chairman: 郭忠義/台北科技大學，時間: 10:40-12:00, 7/7，地點: 學思樓學 104

ID	作者	論文名稱
25	蔡佳翰, 蘇翊翔, 謝孟諺	以中文關鍵字為基礎之隱性回饋的評分計算
41	Cheng-Yi Chang and Ing-Yi Chen	基於 Google Cloud Platform 設計高效能日誌分析平台之研究
56	Cheng Wei Wu, Yi Ren, Muhammad Alfiansyah, Hsin Wei Kao, Chen Wei Hsin and Yu-Chee Tseng	基於智慧購物車之排隊辨識
78	詹于瑩, 李婉如, 吳紹薇, 李心瑜, 呂孟蘋, 陳奕中, 陳錫民	GPS 定位影像補償系統
57	郭忠義, 呂紹清	M2M 語意規則推論應用架構設計研究

2A: Best English paper

Chairman: 周忠信/東海大學，委員：劉立頌/中正大學，馬尚彬/海洋大學，時間: 13:20-14:50, 7/7，地點: 學思樓學 101

ID	作者	論文名稱
30	Ru-Wei Fu and Farn Wang	Automatic Device Farm Management for the Testing of Android Mobile Apps
45	Chi Wen Chen and Farn Wang	Automated Cloud Sandbox Deployment for Implementing DevOps
53	Min-Huang Ho, Win-Tsung Lo, Ruey-Kai Sheu, Yen-Lin Lee and Deron Liang	Method of Distributed Node Management for High-Availability Clusters based on Kernel Virtual Machine
63	Hwai-Jung Hsu and Yves Lin	How Agile Works in a Software Corporation: An Empirical Study of Assessing Agile Methods from Viewpoints of Business Data Analytics
75	Chien-Hung Liu and Woie-Kae Chen	Coupling Analysis and Visualization of KDT Scripts
77	Tze Suen Lim, Yi Chung Chen, Sheng Min Chiu, Wei Lun Wang and Wei Hung Lin	Time Series Skyline Query and Its Neural Filter

2B: English paper

Chairman: 林迺衛/中正大學，時間: 13:20-14:50, 7/7，地點: 學思樓學 102

ID	作者	論文名稱
2	Chun-Hsiung Tseng, Lin Hui, Yung-Hui Chen and Jia-Long Li	A GPS Navigation System Leveraging Voice Based User Interface for Blind People
28	Jiun-Hao Lin and Farn Wang	User-Friendly Trace Viewer for Android Apps Testing
31	Jui Chieh Tai and Farn Wang	Cross-Browser Compatibility Testing of Web Applications
34	Guang-Qi Wang and Farn	Wang Automated Testing for Quality Android Applications
40	Jyun-Hua Jiang, Lin-Huang Chang and Tsung-Han Lee	Wireless sensor network under asynchronous mechanism to dynamically adjust the sleep schedule
52	Ming-Chi Liu and Yueh-Min Huang	Performing a phenomenographic text mining to understand the students' experiences of software programming

2C: IoT

Chairman: 廖峻峰/政治大學，時間: 13:20-14:50, 7/7，地點: 學思樓學 103

ID	作者	論文名稱
8	邱大洲, 陳文輝	以智慧型手機感測器結合機器學習演算法之雲端居家行為辨識系統
43	徐士展, 戴偉竹, 盧韋宏, 蔡漢霖, 趙宥勝, 劉立頌	具易用性之智慧家庭控制系統 (A Controlling System in Smart Home with Usability)
83	李允中, 任哲晨, 吳佳芷	物聯網中介軟體：感測器服務及網路服務與複雜事件處理之整合
68	郭家旭, 李文廷, 馬毓棣, 林敬祥	iRollCall - NFC 輕量級行動點名服務系統
71	Chia-Hsu Kuo, Wen-An Tsai and Tzung-Shi Chen	無線感測網路的行動充電策略之研究
32	陳映如, 廖峻鋒	資源導向智慧家庭服務維運機制的設計與實現
33	盧威辰, 廖峻鋒	適用於數位互動藝術的聚合式 BLE-MQTT 閘道設計

2D: Industry Talk I

Chairman: 英家慶/逢甲大學，時間: 13:20-14:50, 7/7，地點：學思樓第九國際會議廳

ID	作者	論文名稱
	林裕丞總經理 / 新加坡鈦坦科技	空手、緊握、到放手－敏捷路上學到的 5 件事
	林俊孝 Chief Technology Officer/ Picowork	協同式雲端作業系統 - 創造雲端社會的新生態

3A: Best regular paper

Chairman: 梁德容/中央大學，委員：劉建宏/台北科技大學，孔崇旭/台中教育大學，時間: 15:00-16:30, 7/7，地點: 學思樓學 101

ID	作者	論文名稱
9	廖峻鋒, 鄭敬儒, 陳恭, 賴晨禾, 邱天	基於行為驅動開發製程的區塊鏈智能合約整合測試服務平台
20	徐偉哲, 鄭永斌	Virtual Objects for Program Visualization in xDIVA
49	李信杰, 黃琪恩, 游傑麟	以 Text-Attribute-Context 為基礎識別演化網頁中變動元素之方法與自動化網頁回歸測試之實務應用
60	郭忠義, 潘家偉	應用堆疊式降噪自動編碼器建構學生退學預測模型
65	陳鵬中, 馬尚彬, 呂致緯	LODE: 鏈結開放資料之建立、查詢與服務生成平台
74	Wing Lun Siu, Yi-Chung Chen, Kuo-Cheng Ting, Don-Lin Yang and Hsi-Min Chen	在社群網路中利用 m-代表性天際線查詢搜尋 m-相似的用戶

3B: Software Testing II

Chairman: 王凡/台灣大學，時間: 15:00-16:30, 7/7，地點: 學思樓學 102

ID	作者	論文名稱
29	Yueh-Ru Lin, Ting-An Yeh and Cheng-Zen Yang	Android 手機 Unity 遊戲監測工具的設計與實作
48	吳尚諭, 林迺衛	基於限制邏輯圖的測試覆蓋標準管理及邊界測試案例產生
54	李培琴, 林迺衛	類別層級單元測試的限制式測試案例產生器
55	張朝翔, 林迺衛	測試物件初始化程序的自動產生
64	唐書麒, 林楚迪	奠基於主題模型之測試個案排序改進方法
61	郭忠義, 蘇翊棠, 賴岱佑	基於雲端分散式環境多用戶火災逃生系統
36	Mao-Jhe Fong, Farn Wang and Yu Ting Chang	Black-box Test Case Generation for Memory Leaks of Android Apps

3C: Demo paper

Chairman: 李信杰/成功大學，委員：鄭永斌/中央大學，陳英一/台北科技大學，時間: 15:00-16:30, 7/7，地點: 學思樓學 103

ID	作者	論文名稱
11	Wei-Yu Lai and Han-Ming Wu	drEDA：一個基於維度縮減技術的互動式探索性資料分析網頁應用程式
44	薛念林, 梁少榕	應用於資訊教育之可擴充性象棋對奕平台
50	Chu-Yu Wang, Yung-Li Hu, Yao-Tung Tsou, Yennun Huang and Sy-Yen Kuo	A Testing Tool for Mobile Edge Computing Applied on Smart Traffic
62	Chia-Hsu Kuo, Li-Wei Chen, Jing-Shiang Lin and Wen-An Tsai	iScreens – 智慧型多螢幕 eSOP 管理工具
66	何靖霆, 馬尚彬, 戴碩宏	基於流程引擎之對話機器人框架
67	李霽烝, 陳薇涵, 陳錫民	基於共享機制的計程車評價分享平台

4A: Data Engineering

Chairman: 何承遠/亞洲大學，時間: 10:40-12:00, 7/8，地點: 學思樓學 104

ID	作者	論文名稱
42	Chang You-Wei and Chihhsiong Shih	以最小偏差基因及粒子群演算法分析缺血性中風轉出血性中風成因探討
46	葉錦文, 葉明憲, 葉家舟, 邱宏彬, 吳梅君, 林迺衛	基於癌症登記資料庫的中西醫合併治療的肺癌存活分析
79	侯修平, 許懷中, 吳榮彬, 楊東麟	根據 Open edX 的課程設計的競賽式學習平台架構
80	邱毓宸, 許懷中, 吳榮彬, 楊東麟	使用線上學習行為紀錄預測學生的學習成效
76	Ming-Chi Liu, Chen-Hsiang Yu, Jungpin Wu and An-Chi Liu	System log analysis for understanding user engagement: The case of MOOCs
35	Cheng-Yuan Ho	電子票證大數據應用於台中市公車旅客型態之研究

4B: Software Engineering II

Chairman: 李文廷/高雄師範大學，時間: 10:40-12:00, 7/8，地點: 學思樓學 102

ID	作者	論文名稱
72	黃俊詠, 陳錫民, 陳奕中	以虛擬容器為基底的 IoT 服務管理機制
10	Feng-Shou Yu and Wei-Ling Chen	一個新的系統耦合度度量 (A New System Coupling Metric)
12	Chang-Yen Tsai, Kuo-Hsun Hsu and Chun-Han Lin	Identifying Aspects from Cross-Version Revision History in Software Development
18	薛念林, 廖健宏	以 3D 視覺化技術為基礎之開源軟體品質分析
23	劉建宏, 陳偉凱, 陳炳宏, 楊凱霖	支援程式作業壞味道偵測與批改之工具
73	李文廷, 許博淳	應用設計結構矩陣分析軟體設計氣味

4C: Application

Chairman: 徐國勛/台中教育大學，時間: 10:40-12:00, 7/8，地點: 學思樓學 103

ID	作者	論文名稱
13	林桂任, 張君丞, 陳世軒, 陳克勤 許乙清	整合健康存摺與開放資料於物聯網架構之用藥安全系統
19	Jia-Ching Jian and Yung-Ping Cheng	Very High Precision Optical Character Recognition for Clean-Fixed-Sized True Type Character
22	楊博善, 黃俊堯, 高國峰, 廖宜恩	從電視節目到鏈結開放資料的轉換系統設計與開發
24	陳湘諭, 鄭為民	改良之專利資料庫檢索系統
26	Su Yi-Shiang, 蔡佳翰, 謝孟諺	結合擴增實境與 OpenCV 於服飾業之研究

4D: Industry talk II

Chairman: 許懷中/逢甲大學，時間: 10:40-12:00, 7/8，地點：學思樓第九國際會議廳

ID	作者	論文名稱
	王燄 / MathWorks 財務工程技術經理	資料科學在物聯網之應用

大會演講 I

Big Data Analytics: A Requirements Engineering Perspective

時間: 9:30-10:20, 7/7, 地點：學思樓第九國際會議廳

Prof. Lawrence Chung, Department of Computer Science University of Texas at Dallas, U.S.A.

Big data promises to lead to better decisions, which can bring greater operational efficiency, productivity, reduced cost and risk, and the like to a variety of domains. But is the use of big data always going to be beneficial, and if so how? In answering this question, I will first give a light survey on the use of big data in practice, along with what big data might mean. Afterwards, I will describe a goal-oriented approach to beneficially using big data. This approach is intended to rationally "connect the dots", from stakeholders' problems and needs, business key performance indices, important insights through analytics for both AS-IS and TO-BE, machine learning techniques, SQL/NoSQL big queries, etc. I will talk about how this approach can aid business decision making in general and more specifically in business process reengineering, possibly with a tool support. At the end, I will outline challenges in more beneficially using big data.

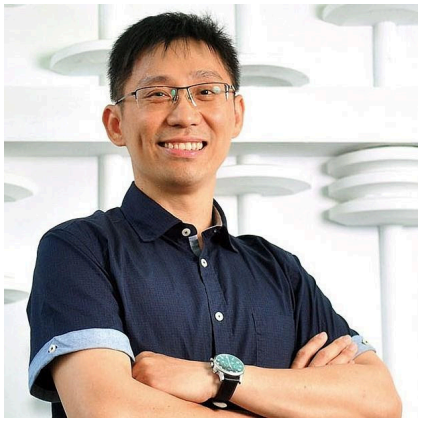
Lawrence Chung has been working in Requirements Engineering, System/Software Architecture and Systems Engineering. He was the principal author of the research monograph "Non-Functional Requirements in Software Engineering", and has been involved in developing "RE-Tools" (a muiti-notational requirements modeling tool kit) , "HOPE" (a smartphone app project for people with difficulties), and "Silverlining" (a Google-award winning project on cloud computing and big data). He has been a keynote speaker, invited lecturer, co-editor-in-chief for Journal of Innovative Software, editorial board member for Requirements Engineering Journal and International Journal of Networked and Distributed Computing, editor for ETRI Journal, and program co-chair for various international events. He is currently on the faculty of Computer Science at University of Texas at Dallas. He received his Ph.D. in Computer Science in 1993 from University of Toronto.

大會演講 II

從大數據到人工智慧

時間: 9:30-10:20, 7/8, 地點：學思樓第九國際會議廳

陳昇瑋博士，台灣資料科學協會理事長



大數據在台灣蔚為風潮，無論是政府官員或販夫走卒，人人皆聽聞大數據的威力。因此，產業界及各級政府皆努力建置所謂的大數據平台，以蒐羅及保存資料為己任，並導入資料的視覺分析工具，讓決策者們能夠快速地查看管理或施政成效，以客觀數據來輔助主觀評價，以分析輔助經驗，以事實取代臆測。

這些都是好的進展。收集資料並整理成視覺化的分析圖表，對於評估及掌控現況有非常大的幫助，讓我們不再只能依直覺及經驗做決策。但，其實，這只是把資料平台準備好而已，要充份發揮資料的價值，還沒有沾到邊。要發揮資料價值，不能光談大數據，機器學習與人工智慧是絕對不該忽略的。事實上，這三者環環相扣：大數據是材料，機器學習是處理方法，人工智慧是成品所呈現的特質。這個時代，蒐集了大量資料，只呈現給人看，而不是拿來餵給電腦學習，讓你的應用呈現人工智慧，就跟採集了大量松露結果拿來沾醬油一整碗吃掉一樣可惜。如同精靈寶可夢需要有訓練師才能發揮能力，擁有大數據後，我們也需要很多很多的機器學習專家（有人稱呼為 AI 訓練師），才能讓我們手中的大數據真正發揮價值。

在此演講中，我將為聽眾闡明資料科學、大數據、人工智慧、機器（深度）學習、資料探勘等相近但又不同的詞彙，再以各領域的實際案例來分享資料的可能應用及實用價值。同時，我將與聽眾分享其協助多家企業培訓資料科學家及導入資料科學團隊的各種經驗，從企業如何跳脫既有框架，讓資料科學團隊盡情揮灑無限創意著手，進而能活用資料、挖掘出潛藏在資料中不為人知的秘密，最終打造高信任度及高效率的工作環境，展現資料分析的價值。

陳昇瑋博士 現為台灣資料科學協會理事長、中央研究院資訊科學研究所研究員暨資料洞察實驗室主持人，研究領域為大數據分析、深度學習、計算社會科學及多媒體系統等，在使用者／社群意見及感受的萃取及量化方面持續有代表性的研究創見。陳博士堅信資料及資料分析的價值，長期推廣資料科學及其在各領域的應用，發起台灣資料科學協會及台灣資料科學年會，期能將對於資料科學的熱情傳達給大眾，一起來探索資料科學的潛力，並將資料科學引入每個人的專業領域之中。他期待讓資料分析在台灣不再是口號，而是大家真實拿來解決問題及創造價值的工具。更多資訊可參考其個人網頁。

專題論壇


鄭有進 教授


李允中 教授


Prof. Lawrence Chung


韋建名 博士/副總

Topic: Software Engineering and Data Science

時間: 16:50, 7/7, 地點：學思樓第九國際會議廳

主席：鄭有進教授 / 國立臺北科技大學

鄭有進教授任職於台北科技大學資工系，教授物件導向程式設計、樣式導向設計、軟體架構、軟體需求與規格、人工智慧等課程。最近，他為大學生學習物件導向程式設計寫了中文部落格 How To Solve It – OOP (http://htsioop.blogspot.tw/) 與英文部落格 How To Solve It – CPP (http://htsicpp.blogspot.tw/)。他於 2009/12 2012/11 三年間主持學界科專計畫，致力於推廣敏捷方法 Scrum，協助多家台灣的資通訊廠商改善其軟體專案的開發效率與品質；該科專計畫並產出自由軟體 ezScrum (http://sourceforge.net/projects/ezscrum/) 供全球社群使用。鄭教授現曾任台灣軟體工程學會秘書長、理事長，他也是 IEEE Computer Society 的會員，並於 2010 年獲得 Scrum Alliance 的 Certified ScrumMaster 資格。鄭教授畢業於台北工專 (1985)，隨後獲得美國 Johns Hopkins University 碩士學位 (1990) 與 University of Oklahoma 博士學位 (1993)。

與談人：李允中教授 / 國立臺灣大學資訊工程系

學歷: Computer Science 博士，Texas A&M University
現職: 教授，國立台灣大學資訊工程學系
經歷: 技術顧問，經濟部技術處
External Examiner UNITEN, Malaysia
理事長，台灣軟體工程學會副主席國際模糊學會 (IFSA)
召集人，經濟部技術處 SBIR 資通領域
召集人，教育部顧問室軟體工程聯盟
教授，國立中央大學資訊工程學系
主任，國立中央大學電子計算機中心
系主任，國立中央大學資訊工程學系
獲獎: IBM 全球聯合大學研究 (SUR) 獎助
傑出電機工程教授，中國電機工程師學會
特聘教授，國立中央大學

與談人：Prof. Lawrence Chung / Department of Computer Science, University of Texas at Dallas

Lawrence Chung has been working in Requirements Engineering, System/Software Architecture and Systems Engineering. He was the principal author of the research monograph "Non-Functional Requirements in Software Engineering", and has been involved in developing "RE-Tools" (a muiti-notational requirements modeling tool kit) , "HOPE" (a smartphone app project for people with difficulties), and "Silverlining" (a Google-award winning project on cloud computing and big data). He has been a keynote speaker, invited lecturer, co-editor-in-chief for Journal of Innovative Software, editorial board member for Requirements Engineering Journal and International Journal of Networked and Distributed Computing, editor for ETRI Journal, and program co-chair for various international events. He is currently on the faculty of Computer Science at University of Texas at Dallas. He received his Ph.D. in Computer Science in 1993 from University of Toronto.

與談人：韋建名博士 / 帆宣系統科技股份有限公司

韋博士在電腦，通信，網路及軟體產業界已有超過 40 年以上的經驗。過去曾服務於 IBM、貝爾實驗室 (Bell Laboratories)、Racal-Milgo、Fibronics/Spartacus、Axonet、瑞博强芯 (天津) 科技有限公司及目前的帆宣系統科技股份有限公司。他所負責的工作範圍涵蓋了策略規劃、系統架構設計、產品研發銷售、技術管理與行政管理等事務。他曾有五次創業及公司轉型的成功經驗，並成功地研發及推出超過 20 種以上的商業產品。

韋博士從美國康州大學（University of Connecticut）工學院，電機及計算機科學系 (Electrical Engineering and Computer Science Department）獲得他的學士（1971）、碩士（1973）及哲學博士（1978）學位。他也是美國康州大學工學院 1995 年的"傑出工程師校友獎" 的得獎人。

企業演講

空手、緊握、到放手 – 敏捷路上學到的 5 件事

林裕丞總經理 / 新加坡鈦坦科技

時間：13:20-14:05 (2D), 7/7, 地點：學思樓第九國際會議廳

為什麼吃飽沒事做要跑敏捷？從一開始新加坡 5 人，到現在跨國 150 人，這過程中有喜悅、有掙扎、也有不少的學習和成長。這是一個鈦坦科技十年來的演化史，經由長時間跨度與宏觀組織角度，跟大家分享我們在組織文化和產品面上的改變和學習。在這次演講中您將會有以下收穫：

1. 什麼時候要跑敏捷？
2. 如何評估敏捷的成效？
3. 在組織管理上有那些工具和方法？

協同式雲端作業系統 - 創造雲端社會的新生態

林俊孝 Chief Technology Officer/ Picowork

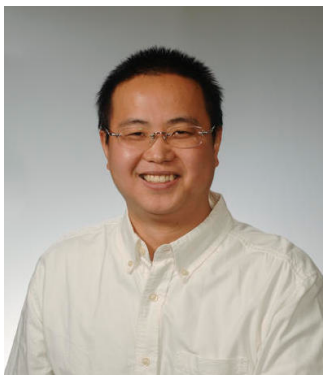
時間：14:05-14:50 (2D), 7/7, 地點：學思樓第九國際會議廳

現今的生活大量的依賴第三方應用服務來進行，然而不同的應用服務和服務平台間存在著商業經營上、技術架構上、及功能發展上及壁壘（Boundry），帶來產業全方位協同的挑戰。「協同式雲端作業系統」以作業系統的角度出發，透過技術思維上的創新，將整個網際網路看成是全球最大的一部計算機，重新定義了網際網路的定位，使各行各業可以擁有自己的雲端作業系統環境，又可以透過自己的雲端環境，與任何人(同事、顧客、夥伴、朋友)自由地進行協同作業，也能以最低的成本，讓過去、現在、及未來的網路資訊系統和網路資源擴充到雲端環境中，最後亦能藉此產生自我的大數據，為創造雲端社會的新生態，帶來了新的可能。

資料科學在物聯網之應用

王燚 / MathWorks 財務工程技術經理

時間：10:40-11:25 (4D), 7/8, 地點：學思樓第九國際會議廳



使用資料科學技巧將龐大複雜的資料轉化成可使用的資訊已經變成炙手可熱的趨勢，因為它能夠協助提升工程設計及決策流程的效率。不過，目前資料科學最大的挑戰，在於如何發展出有效的解析 (analytics) 結果，並將其整合到生產系統如無論是企業使用的 Hadoop、Spark、Tableau 或各種資料庫，甚至是各式各樣嵌入式感測器所構成的物聯網 (IoT) 環境中等等。本演講將介 MATLAB 針對這些挑戰所提出的解決方法和技術，對於那些從未聽說過 MATLAB 的您來說，將會體會到為何 MATLAB 為何會成為全球工程師和科學家進行研發工作的關鍵性分析和建模的高效平台；而對已經是 MATLAB 的使用者來說，本演講也將讓您了解 MATLAB 的最新功能，如何協助您進一步簡化並加速完成您的資料科學任務。

JISE Special Issue

Special Issue on Interdisciplinary Study on Software Engineering and Data Science

投稿 TCSE English Session, 並獲選為 Best paper 者，將推薦至此 Special Issue。

Data science is a cross-domain research field which integrates information technologies and statistical methods to extract knowledge and insights from various kinds of data. Along with the maturity of big data techniques and infrastructures, a huge amount of data can now be collected, stored, and processed for further exploration of hidden information and values. Recommendation systems, internet of things, and applications adopting machine learning and deep learning etc. are all developed on the basis of the rapid growth of data science.

However, the development of a data system, an information technology system working together with data deluge, is now still more an art than an engineering process. Conventional software engineering methods do not suit the development of a data system. For example, the capability of a data set, e.g., what a data set can be adopted for, cannot be revealed without data discovery. In other words, the requirements of a data system are gradually discovered during system development. The bugs within a data system which corrupt the data may dramatically influence the performance of a data system without generating any faulty behavior. Since data and data science techniques would play an essential role for future information systems, studies of software engineering methods and tools on the development of data systems can be essential.

On the other hand, the statistical data generated from conventional software engineering methods can also be used on improving the methods. Developing and running software generates a large amount of data about the software engineering process and the user usage, which can become useful information with the help of data science. Mining design patterns and performance impact factors are two interesting examples. Adopting data analytics techniques on software engineering becomes an important issue for software engineering study in software development life cycle.

Scopes The special issue welcomes the original contribution that addresses (1) the software engineering methods for data systems/products and data techniques, and (2) the data analytics techniques for software engineering methods.

- Utilities and environments for data engineering, analytics, and applications
- Software engineering for big data infrastructure
- Software engineering for data systems/products
- Software engineering for IoT
- Software engineering for data processing and engineering
- Inductive software engineering
- Data mining on software engineering repositories
- Data analytics for software engineering
- Validation and Verification of data systems/products

- Systematic methods for data de-identification

Author's Guidelines The submission should follow the formatting guidelines of the Journal of Information Science and Engineering (http://jise.iis.sinica.edu.tw/pages/authors/index.html). Any questions regarding this special issue should be sent to the guest editors. All submitted papers will be reviewed by at least three reviewers and selected based on their originality, significance, relevance, and clarity of presentation. Prospective authors should submit full manuscripts with PDF format electronically to http://journal.iis.sinica.edu.tw/url/jise-seds by 8/30/2017.

Publication Schedule

- Manuscript submission deadline: 8/30/2017
- Notification of Acceptance/Rejection/Revision: 11/30/2017
- Final Manuscript Due: 12/31/2017
- Tentative Publication Date: 2018
- For more information, please contact the guest editors.

Guest editors:

- Yu-Chin Cheng, National Taipei University of Technology, Taiwan, yccheng@csie.ntut.edu.tw
- Ji Zhang, The University of Southern Queensland, Australia, Ji.Zhang@usq.edu.au
- Chang-Shing Lee, National University of Tainan Taiwan, leecs@mail.nutn.edu.tw
- Don-Lin Yang, Feng Chia University, Taiwan, dlyang@fcu.edu.tw

TCSE Hacks 黑客松競賽

TCSE 2017 大會的主題為「Software Engineering and Data Science」，將探討軟體工程與資料科學的相關研究，包含應用軟體工程技術於資料科學及應用資料科學於軟體工程。為了推廣軟體工程與資料科學的整合應用，特舉辦此競賽。

今年我們將以 MOOCs (Massive Open Online Courses) 線上學習系統的資料為分析對象。OpenEdu 是台灣主要 MOOCs 平台之一，提供優質且免費的線上學習課程。學習者的學習行為被完整的記錄在系統中，透過分析這些學習行為可以改善日後的線上學習方式。本黑客松的活動希望大家能夠應用程式及資料分析的技巧，挖掘及了解學生值得探討的學習行為模式。

用於競賽的資料包含去識別化後，共約 80 門 OpenEdu 線上課程之學生登出入、觀看影片、回答線上問題、留言板討論等紀錄。而我們的最終目的是，希望參賽各組可以透過這些資料建立期初表現與期末表現的預估模型。因此在黑客松複賽前，我們將提供修習記錄等資料供參賽隊伍進行資料清理與理解的相關工作，而比賽當天主辦方將提供其中約 56 門課學生的期末表現，作為訓練資料集的標籤，然後使用餘下約 24 門課學生的期末表現結果，作為測試資料標籤。

各組所建構之模型在測試資料集上的表現 (如準確率、召回率等) 為競賽評分的主要依據，此外，為了鼓勵參賽隊伍使用軟體工程方法，主辦單位也將於黑客松進行過程中觀察各組的工作狀況，作為軟體工程實務獎項的評分依據。

重要日期

- MATLAB 工作坊：5/31
- 初賽報名截止：6/1
- 初賽：6/2-6/15
- 複賽：7/7-7/8, 逢甲大學

活動協辦：ITSA 社群運算與巨量資料跨校資源中心，OpenEdu, 逢甲大學大數據中心
活動贊助：鈦坦科技，鈦思科技，Picowork, 資拓宏宇, 誠雲科技, 台灣軟體工程學會

Section 1

Software Engineering, Software Testing, Web Engineering and AI

時間: 10:40-12:00, 7/7

1A Software Engineering I

Chairman: 林哲正/高雄師範大學，時間: 10:40-12:00, 7/7，地點: 學思樓學 101

ID	作者	論文名稱
6	Shuo-Hong Kao and Dow-Ming Yeh	程式語言語法視覺化工具雛型
38	呂信緯, 陳英一	以 Nodejs 非同步設計框架建構語意分析加值系統之研究
58	郭忠義, 許聖泉	程式碼動態結構抄襲鑑定
70	薛念林, 莫剛, 陳錫民	OpenEdu 磨課師系統學習資料分析報告
69	Chang-Yen Lee, Hui-Juan Chen and Che-Chern Lin	從設計與實作觀點探討模糊專家系統在補救教材的應用－以高職數位邏輯課程為例

Paper Title

Author One name, Author Two Name
Department of Your Department
TCSE University
City, Country
Email: {email1, email2}@domain.name

***Abstract*—Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear.**

I. Introduction

In the current trend, design pattern has been widely used for improving software quality. In the academic research, design patterns is one of important research area and applied in different areas [5], [11], [12], many design pattern detection approaches are proposed [6], [13], [14], some work discuss pattern quality metrics [1], [7], [8], design pattern formalization and specification [**?**], [**?**], [**?**], and its benefits [9], [10], [15], [16].

However, using design patterns is not easy, ... Therefore, several approaches are proposed to check the violence of pattern application in a system [2], [17]. We think only static checking to pattern structure is not enough, dynamic testing to test the patterns' semantics is necessary.

In this paper, we propose a test model for design patterns. ....

For example, when we apply *Singleton* pattern, we should conduct the following tests: creating two objects and check if they have the same references. Another example is *Strategy*, when we applied strategy a with this paper, and replace it by another algorithm b, they should have the same behavior, that is, the same output. The following code shows the testing context:

```
1  class  TestStrategyDesignPattern  {
2     void  testSameResult() {
3           Context c = new Context();
4           Strategy a = new StrategyA();
5           c.setStrategy(a);
6           int resultA  = c.doIt();
7           Strategy b = new StrategyB();
8           c.setStrategy(b);
9           int restultB  = c.doIt();
10          assertEquals(a, b); // should have same
                result
11    }
12 }
```

A. Paper outline

The remainder of this paper is structured as follows: In section 2, we describe the related work to this research.

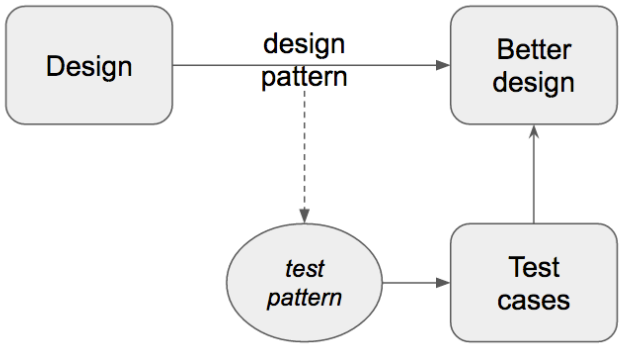


Figure 1. The concept of *test pattern*

Section 3 introduces the details of our approach by two design pattern examples. Section 4 summarizes our approach and future work.

II. Related work

以下針對此研究的相關研究做探討分析：

A. Pattern testing

Chu and Hsueh proposed test refactoring approach in a pattern driven developent [3]. Test-first strategy and code refactoring are the important practices of Extreme Programming for rapid development and quality support. ... However, when the developers perform a pattern-based refactoring to improve the quality, the effort of revising the test cases is much more than that in simple code refactoring (see Fig. 2).

To address this problem, ...

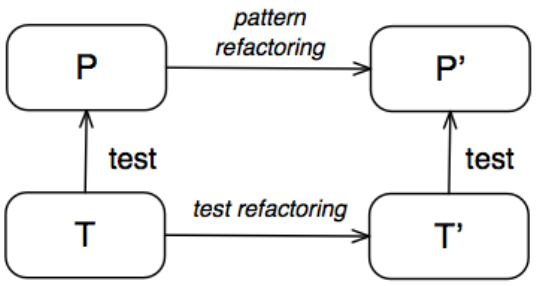


Figure 2. Pattern based test case refactoring

To understand the application context for a pattern in a system, Lin proposed a *Design Pattern Unit Test (DPUT)* approach, which utilizes java Annotation skill to record the pattern utilization and verifies with the expection in the DPUT. This research also design a software framework to help developers design the DPUP in a specification basis. The code is implemented as an Eclipsed plug-in which can automatically transform DPUT into class diagram for better understanding. For the system maintainers they can find out the errors in the earlier phases.

B. Pattern evaluation

設計樣式的效用一直是重要的研究議題。也因此許多學者專家開始投入這方面的研究。Huston [8] 設計了一個實驗以驗證設計樣式是否與軟體衡量相容，亦即，探討應用設計樣式所提昇的軟體品質是否真的可由軟體衡量顯示出來？Huston 的實驗將設計樣式的結構還原成非設計樣式的結構，並以相關的軟體衡量 (software metric) 分別衡量這兩個不同的架構，來比較有無採用設計架構在該軟體衡量上的差異。此實驗顯示設計樣式的品質效能可以透過軟體衡量顯現出來。

III. Our approach

To design the test model for pattern testing, we have to consider the following issues:

- Correctness and Precision. Correctness considers if all faults can be explored by the proposed test cases, and Precision considers if the test can explore the faults in an efficient way. Our test model should satisfy correctness and precision. ...
- Coverage. In general we have many coverage strategies for testing, for example statement coverage or branch coverage. When we test we should enlarge the coverage as we can. Therefore we should care about the normal cases, boundary cases and exception cases. When we utilize this concept to testing a pattern-applied system, what are the boundary cases and exception cases? ...
- Generalization. Gamma et al. classified patterns into creational patterns, structural patterns and behavioral patterns [4]. Patterns in the same group may have same testing issues and methods. We can modularize our testing programs by class generalization. ...

a) Potential Pattern Violation (PPV): To test the pattern-applied application more efficiently, ...

b) Test Pattern for Design Pattern (TP4DP): One of the benefits of design pattern is it is well structured and contains executable code for programmers. Our TP4DP follows the same concept. A test pattern has the following sections: ...

A. Testing "Singleton" pattern

a) Pattern name: Singleton: The intent of the singleton is to ensure a class has only one instance, and provide a global point of access to it.

b) Potential pattern violation: There are two PPV in the Singleton pattern:

- PPV1: When we create two objects from the same class, they refer to different objects.
- PPV2: The original object constructor is not set to be private.

c) Testing guideline: Create two objects of the same Singleton class, check if there are equal by using the operator "==".

d) Demo code: Here is the demo code for the class *Radio*

```
1 class TestRadio {
2     void testRadioSingletonPPE1() {
3         Radio r1 = Radio.instance();
4         Radio r2 = Radio.instance();
5         assertTrue(r1==r2): "Singleton is violated
                ";
6     }
7 }
```

To test the PPV2, we use static testing to check if the constructor is declared as **private**. The example below is the one that doesn't pass the PPV2 checking.

```
1 class Radio {
2     static Radio instance;
3     static Radio getInstance() {
4         if (instance == null)
5             instance = new Radio();
6         }
7         return instance;
8     }
9 }
```

IV. Conclusion

In this paper we propose an idea of pattern test for guiding the programs with patterns applied. We believe a program is error-prone when they are designed in delicate and complex structure. Design patterns are good for flexible design but not easy to understand and apply. For pattern beginners we need a guideline, framework or tool to help them. We demonstrate two examples of *Singleton* and *Observer* to illustrate our ideas. In the future we will explore more design patterns and develop a framework for the pattern test.

References

[1] Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Sofia Charalampidou, and Paris Avgeriou. The effect of gof design patterns on stability: a case study. *IEEE Transactions on Software Engineering*, 41(8):781–802, 2015.

[2] Alex Blewitt, Alan Bundy, and Ian Stark. Automatic verification of java design patterns. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 324–327. IEEE, 2001.

[3] Peng-Hua Chu, Nien-Lin Hsueh, Hong-Hsiang Chen, and Chien-Hung Liu. A test case refactoring approach for pattern-based software development. *Software Quality Journal*, 20(1):43–75, 2012.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[5] Alan R Graves and Chris Czarnecki. Design patterns for behavior-based robotics. *IEEE Transactions on Systems, Man, and cybernetics-part A: systems and humans*, 30(1):36–41, 2000.

[6] Dirk Heuzeroth, Thomas Holl, Gustav Hogstrom, and Welf Lowe. Automatic design pattern detection. In *Program Comprehension, 2003. 11th IEEE International Workshop on*, pages 94–103. IEEE, 2003.

[7] Nien-Lin Hsueh, Peng-Hua Chu, and William Chu. A quantitative approach for evaluating the quality of design patterns. *Journal of Systems and Software*, 81(8):1430–1439, 2008.

[8] Brian Huston. The effects of design pattern application on metric scores. *Journal of Systems and Software*, 58(3):261–269, 2001.

[9] Clemente Izurieta and James M Bieman. A multiple case study of design pattern decay, grime, and rot in evolving software systems. *Software Quality Journal*, 21(2):289–323, 2013.

[10] TH Ng, SC Cheung, WK Chan, and Yuen-Tak Yu. Do maintainers utilize deployed design patterns effectively? In *Proceedings of the 29th international conference on Software Engineering*, pages 168–177. IEEE Computer Society, 2007.

[11] Michael Oduor, Tuomas Alahäivälä, and Harri Oinas-Kukkonen. Persuasive software design patterns for social influence. *Personal and ubiquitous computing*, 18(7):1689–1704, 2014.

[12] Cagri Sahin, Furkan Cayci, Irene Lizeth Manotas Gutiérrez, James Clause, Fouad Kiamilev, Lori Pollock, and Kristina Winbladh. Initial explorations on design pattern energy usage. In *Green and Sustainable Software (GREENS), 2012 First International Workshop on*, pages 55–61. IEEE, 2012.

[13] Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, and Spyros T Halkidis. Design pattern detection using similarity scoring. *Software Engineering, IEEE Transactions on*, 32(11):896–909, 2006.

[14] Marco Zanoni, Francesca Arcelli Fontana, and Fabio Stella. On applying machine learning techniques for design pattern detection. *Journal of Systems and Software*, 103:102–117, 2015.

[15] Cheng Zhang and David Budgen. What do we know about the effectiveness of software design patterns? *Software Engineering, IEEE Transactions on*, 38(5):1213–1231, 2012.

[16] Cheng Zhang and David Budgen. A survey of experienced user perceptions about software design patterns. *Information and Software Technology*, 55(5):822–835, 2013.

[17] Chunying Zhao, Jun Kong, and Kang Zhang. Design pattern evolution and verification using graph transformation. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 290a–290a. IEEE, 2007.

1B Software Testing I

Chairman: 林楚迪/嘉義大學，時間: 10:40-12:00, 7/7，地點: 學思樓學 102

ID	作者	論文名稱
17	Wan-Chuan Lee and Yung-Pin Cheng	運用壓力測試腳本的同步來增進壓力測試效能
21	薛念林, 黃紫芳	測試驅動之設計樣式測試模型設計與實作
47	張振鴻, 林迺衛	基於限制邏輯圖的單元測試案例產生器
59	郭忠義, 彭柔瑄	Android 應用程式之資訊安全檢測
14	劉建宏, 陳偉凱, 黃映瑞, 林容榆	Android 手機手錶互動應用程式之相容性測試研究

Paper Title

Author One name, Author Two Name
Department of Your Department
TCSE University
City, Country
Email: {email1, email2}@domain.name

***Abstract*—Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear.**

I. Introduction

In the current trend, design pattern has been widely used for improving software quality. In the academic research, design patterns is one of important research area and applied in different areas [5], [11], [12], many design pattern detection approaches are proposed [6], [13], [14], some work discuss pattern quality metrics [1], [7], [8], design pattern formalization and specification [**?**], [**?**], [**?**], and its benefits [9], [10], [15], [16].

However, using design patterns is not easy, ... Therefore, several approaches are proposed to check the violence of pattern application in a system [2], [17]. We think only static checking to pattern structure is not enough, dynamic testing to test the patterns' semantics is necessary.

In this paper, we propose a test model for design patterns. ....

For example, when we apply *Singleton* pattern, we should conduct the following tests: creating two objects and check if they have the same references. Another example is *Strategy*, when we applied strategy a with this paper, and replace it by another algorithm b, they should have the same behavior, that is, the same output. The following code shows the testing context:

```
1  class TestStrategyDesignPattern {
2      void testSameResult() {
3          Context c = new Context();
4          Strategy a = new StrategyA();
5          c.setStrategy(a);
6          int resultA = c.doIt();
7          Strategy b = new StrategyB();
8          c.setStrategy(b);
9          int restultB = c.doIt();
10         assertEquals(a, b); // should have same
               result
11     }
12 }
```

A. Paper outline

The remainder of this paper is structured as follows: In section 2, we describe the related work to this research.

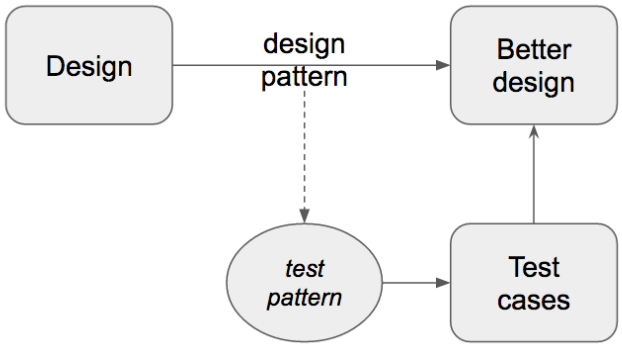


Figure 1. The concept of *test pattern*

Section 3 introduces the details of our approach by two design pattern examples. Section 4 summarizes our approach and future work.

II. Related work

以下針對此研究的相關研究做探討分析：

A. Pattern testing

Chu and Hsueh proposed test refactoring approach in a pattern driven developent [3]. Test-first strategy and code refactoring are the important practices of Extreme Programming for rapid development and quality support. ... However, when the developers perform a pattern-based refactoring to improve the quality, the effort of revising the test cases is much more than that in simple code refactoring (see Fig. 2).

To address this problem, ...

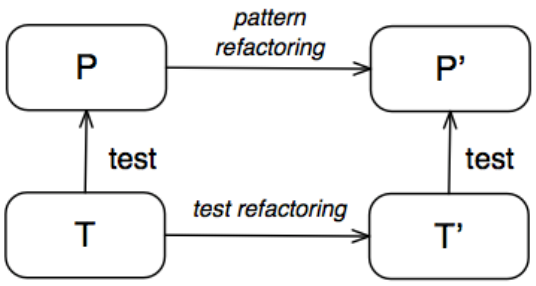


Figure 2. Pattern based test case refactoring

To understand the application context for a pattern in a system, Lin proposed a *Design Pattern Unit Test (DPUT)* approach, which utilizes java Annotation skill to record the pattern utilization and verifies with the expection in the DPUT. This research also design a software framework to help developers design the DPUP in a specification basis. The code is implemented as an Eclipsed plug-in which can automatically transform DPUT into class diagram for better understanding. For the system maintainers they can find out the errors in the earlier phases.

B. Pattern evaluation

設計樣式的效用一直是重要的研究議題。也因此許多學者專家開始投入這方面的研究。Huston [8] 設計了一個實驗以驗證設計樣式是否與軟體衡量相容，亦即，探討應用設計樣式所提昇的軟體品質是否真的可由軟體衡量顯示出來？Huston 的實驗將設計樣式的結構還原成非設計樣式的結構，並以相關的軟體衡量 (software metric) 分別衡量這兩個不同的架構，來比較有無採用設計架構在該軟體衡量上的差異。此實驗顯示設計樣式的品質效能可以透過軟體衡量顯現出來。

III. Our approach

To design the test model for pattern testing, we have to consider the following issues:

- Correctness and Precision. Correctness considers if all faults can be explored by the proposed test cases, and Precision considers if the test can explore the faults in an efficient way. Our test model should satisfy correctness and precision. ...
- Coverage. In general we have many coverage strategies for testing, for example statement coverage or branch coverage. When we test we should enlarge the coverage as we can. Therefore we should care about the normal cases, boundary cases and exception cases. When we utilize this concept to testing a pattern-applied system, what are the boundary cases and exception cases? ...
- Generalization. Gamma et al. classified patterns into creational patterns, structural patterns and behavioral patterns [4]. Patterns in the same group may have same testing issues and methods. We can modularize our testing programs by class generalization. ...

a) Potential Pattern Violation (PPV): To test the pattern-applied application more efficiently, ...

b) Test Pattern for Design Pattern (TP4DP): One of the benefits of design pattern is it is well structured and contains executable code for programmers. Our TP4DP follows the same concept. A test pattern has the following sections: ...

A. Testing "Singleton" pattern

a) Pattern name: Singleton: The intent of the singleton is to ensure a class has only one instance, and provide a global point of access to it.

b) Potential pattern violation: There are two PPV in the Singleton pattern:

- PPV1: When we create two objects from the same class, they refer to different objects.
- PPV2: The original object constructor is not set to be private.

c) Testing guideline: Create two objects of the same Singleton class, check if there are equal by using the operator "==".

d) Demo code: Here is the demo code for the class *Radio*

```
1 class TestRadio {
2     void testRadioSingletonPPE1() {
3         Radio r1 = Radio.instance();
4         Radio r2 = Radio.instance();
5         assertTrue(r1==r2): "Singleton is violated
                ";
6     }
7 }
```

To test the PPV2, we use static testing to check if the constructor is declared as **private**. The example below is the one that doesn't pass the PPV2 checking.

```
1 class Radio {
2     static Radio instance;
3     static Radio getInstance() {
4         if (instance == null)
5             instance = new Radio();
6         }
7         return instance;
8     }
9 }
```

IV. Conclusion

In this paper we propose an idea of pattern test for guiding the programs with patterns applied. We believe a program is error-prone when they are designed in delicate and complex structure. Design patterns are good for flexible design but not easy to understand and apply. For pattern beginners we need a guideline, framework or tool to help them. We demonstrate two examples of *Singleton* and *Observer* to illustrate our ideas. In the future we will explore more design patterns and develop a framework for the pattern test.

References

[1] Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Sofia Charalampidou, and Paris Avgeriou. The effect of gof design patterns on stability: a case study. *IEEE Transactions on Software Engineering*, 41(8):781–802, 2015.

[2] Alex Blewitt, Alan Bundy, and Ian Stark. Automatic verification of java design patterns. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 324–327. IEEE, 2001.

[3] Peng-Hua Chu, Nien-Lin Hsueh, Hong-Hsiang Chen, and Chien-Hung Liu. A test case refactoring approach for pattern-based software development. *Software Quality Journal*, 20(1):43–75, 2012.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[5] Alan R Graves and Chris Czarnecki. Design patterns for behavior-based robotics. *IEEE Transactions on Systems, Man, and cybernetics-part A: systems and humans*, 30(1):36–41, 2000.

[6] Dirk Heuzeroth, Thomas Holl, Gustav Hogstrom, and Welf Lowe. Automatic design pattern detection. In *Program Comprehension, 2003. 11th IEEE International Workshop on*, pages 94–103. IEEE, 2003.

[7] Nien-Lin Hsueh, Peng-Hua Chu, and William Chu. A quantitative approach for evaluating the quality of design patterns. *Journal of Systems and Software*, 81(8):1430–1439, 2008.

[8] Brian Huston. The effects of design pattern application on metric scores. *Journal of Systems and Software*, 58(3):261–269, 2001.

[9] Clemente Izurieta and James M Bieman. A multiple case study of design pattern decay, grime, and rot in evolving software systems. *Software Quality Journal*, 21(2):289–323, 2013.

[10] TH Ng, SC Cheung, WK Chan, and Yuen-Tak Yu. Do maintainers utilize deployed design patterns effectively? In *Proceedings of the 29th international conference on Software Engineering*, pages 168–177. IEEE Computer Society, 2007.

[11] Michael Oduor, Tuomas Alahäivälä, and Harri Oinas-Kukkonen. Persuasive software design patterns for social influence. *Personal and ubiquitous computing*, 18(7):1689–1704, 2014.

[12] Cagri Sahin, Furkan Cayci, Irene Lizeth Manotas Gutiérrez, James Clause, Fouad Kiamilev, Lori Pollock, and Kristina Winbladh. Initial explorations on design pattern energy usage. In *Green and Sustainable Software (GREENS), 2012 First International Workshop on*, pages 55–61. IEEE, 2012.

[13] Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, and Spyros T Halkidis. Design pattern detection using similarity scoring. *Software Engineering, IEEE Transactions on*, 32(11):896–909, 2006.

[14] Marco Zanoni, Francesca Arcelli Fontana, and Fabio Stella. On applying machine learning techniques for design pattern detection. *Journal of Systems and Software*, 103:102–117, 2015.

[15] Cheng Zhang and David Budgen. What do we know about the effectiveness of software design patterns? *Software Engineering, IEEE Transactions on*, 38(5):1213–1231, 2012.

[16] Cheng Zhang and David Budgen. A survey of experienced user perceptions about software design patterns. *Information and Software Technology*, 55(5):822–835, 2013.

[17] Chunying Zhao, Jun Kong, and Kang Zhang. Design pattern evolution and verification using graph transformation. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 290a–290a. IEEE, 2007.

1C Web Engineering

Chairman: 范姜永益/輔仁大學，時間: 10:40-12:00, 7/7，地點: 學思樓學 103

ID	作者	論文名稱
3	Kao Pin, Huo Kuan-Hua, Chang Yi-Tzu, Cheng Yo-Tzu and Hu Chung-Hua	建構商轉雲端管理平台以應用於異質虛擬化環境之雲資源自動移轉與配置
4	Chih-Hung Chang, Chih-Ming Hsieh, Wen-Ching Chen, Y.J. Liao and C.M. Wang	具情境感知之個人化資訊服務框架
15	陳偉凱, 劉建宏, 黃映瑞, 陳科銘	以漸增式使用者指引增加爬蟲器之網頁覆蓋率
27	石聖銓, 范姜永益	應用遺傳規劃法於 Web Services 的服務品質預測
39	Li-Wei Huang and Ing-Yi Chen	運用 Express 中介軟體框架設計非同步責任鏈網路服務之研究

Paper Title

Author One name, Author Two Name
Department of Your Department
TCSE University
City, Country
Email: {email1, email2}@domain.name

***Abstract*—Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear.**

I. Introduction

In the current trend, design pattern has been widely used for improving software quality. In the academic research, design patterns is one of important research area and applied in different areas [5], [11], [12], many design pattern detection approaches are proposed [6], [13], [14], some work discuss pattern quality metrics [1], [7], [8], design pattern formalization and specification [**?**], [**?**], [**?**], and its benefits [9], [10], [15], [16].

However, using design patterns is not easy, ... Therefore, several approaches are proposed to check the violence of pattern application in a system [2], [17]. We think only static checking to pattern structure is not enough, dynamic testing to test the patterns' semantics is necessary.

In this paper, we propose a test model for design patterns. ....

For example, when we apply *Singleton* pattern, we should conduct the following tests: creating two objects and check if they have the same references. Another example is *Strategy*, when we applied strategy a with this paper, and replace it by another algorithm b, they should have the same behavior, that is, the same output. The following code shows the testing context:

```
1  class  TestStrategyDesignPattern  {
2     void  testSameResult() {
3           Context c = new Context();
4           Strategy a = new StrategyA();
5           c.setStrategy(a);
6           int resultA  = c.doIt();
7           Strategy b = new StrategyB();
8           c.setStrategy(b);
9           int restultB  = c.doIt();
10          assertEquals(a, b); // should have same
                result
11    }
12 }
```

A. Paper outline

The remainder of this paper is structured as follows: In section 2, we describe the related work to this research.

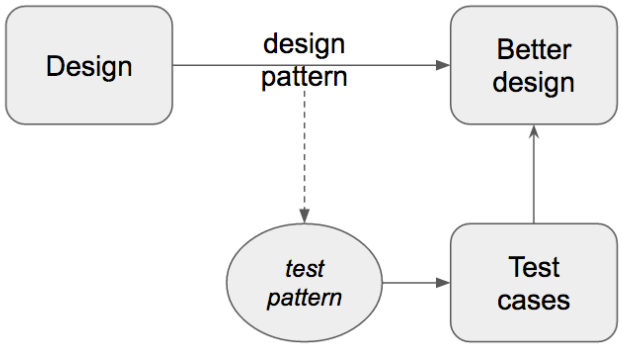


Figure 1. The concept of *test pattern*

Section 3 introduces the details of our approach by two design pattern examples. Section 4 summarizes our approach and future work.

II. Related work

以下針對此研究的相關研究做探討分析：

A. Pattern testing

Chu and Hsueh proposed test refactoring approach in a pattern driven developent [3]. Test-first strategy and code refactoring are the important practices of Extreme Programming for rapid development and quality support. ... However, when the developers perform a pattern-based refactoring to improve the quality, the effort of revising the test cases is much more than that in simple code refactoring (see Fig. 2).

To address this problem, ...

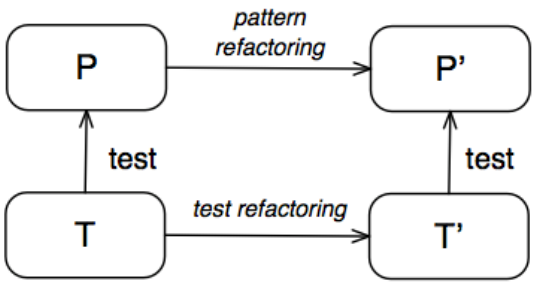


Figure 2. Pattern based test case refactoring

To understand the application context for a pattern in a system, Lin proposed a *Design Pattern Unit Test (DPUT)* approach, which utilizes java Annotation skill to record the pattern utilization and verifies with the expection in the DPUT. This research also design a software framework to help developers design the DPUP in a specification basis. The code is implemented as an Eclipsed plug-in which can automatically transform DPUT into class diagram for better understanding. For the system maintainers they can find out the errors in the earlier phases.

B. Pattern evaluation

設計樣式的效用一直是重要的研究議題。也因此許多學者專家開始投入這方面的研究。Huston [8] 設計了一個實驗以驗證設計樣式是否與軟體衡量相容，亦即，探討應用設計樣式所提昇的軟體品質是否真的可由軟體衡量顯示出來？Huston 的實驗將設計樣式的結構還原成非設計樣式的結構，並以相關的軟體衡量 (software metric) 分別衡量這兩個不同的架構，來比較有無採用設計架構在該軟體衡量上的差異。此實驗顯示設計樣式的品質效能可以透過軟體衡量顯現出來。

III. Our approach

To design the test model for pattern testing, we have to consider the following issues:

- Correctness and Precision. Correctness considers if all faults can be explored by the proposed test cases, and Precision considers if the test can explore the faults in an efficient way. Our test model should satisfy correctness and precision. ...
- Coverage. In general we have many coverage strategies for testing, for example statement coverage or branch coverage. When we test we should enlarge the coverage as we can. Therefore we should care about the normal cases, boundary cases and exception cases. When we utilize this concept to testing a pattern-applied system, what are the boundary cases and exception cases? ...
- Generalization. Gamma et al. classified patterns into creational patterns, structural patterns and behavioral patterns [4]. Patterns in the same group may have same testing issues and methods. We can modularize our testing programs by class generalization. ...

a) Potential Pattern Violation (PPV): To test the pattern-applied application more efficiently, ...

b) Test Pattern for Design Pattern (TP4DP): One of the benefits of design pattern is it is well structured and contains executable code for programmers. Our TP4DP follows the same concept. A test pattern has the following sections: ...

A. Testing "Singleton" pattern

a) Pattern name: Singleton: The intent of the singleton is to ensure a class has only one instance, and provide a global point of access to it.

b) Potential pattern violation: There are two PPV in the Singleton pattern:

- PPV1: When we create two objects from the same class, they refer to different objects.
- PPV2: The original object constructor is not set to be private.

c) Testing guideline: Create two objects of the same Singleton class, check if there are equal by using the operator "==".

d) Demo code: Here is the demo code for the class *Radio*

```
1 class TestRadio {
2     void testRadioSingletonPPE1() {
3         Radio r1 = Radio.instance();
4         Radio r2 = Radio.instance();
5         assertTrue(r1==r2): "Singleton is violated
                ";
6     }
7 }
```

To test the PPV2, we use static testing to check if the constructor is declared as **private**. The example below is the one that doesn't pass the PPV2 checking.

```
1 class Radio {
2     static Radio instance;
3     static Radio getInstance() {
4         if (instance == null)
5             instance = new Radio();
6         }
7         return instance;
8     }
9 }
```

IV. Conclusion

In this paper we propose an idea of pattern test for guiding the programs with patterns applied. We believe a program is error-prone when they are designed in delicate and complex structure. Design patterns are good for flexible design but not easy to understand and apply. For pattern beginners we need a guideline, framework or tool to help them. We demonstrate two examples of *Singleton* and *Observer* to illustrate our ideas. In the future we will explore more design patterns and develop a framework for the pattern test.

References

[1] Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Sofia Charalampidou, and Paris Avgeriou. The effect of gof design patterns on stability: a case study. *IEEE Transactions on Software Engineering*, 41(8):781–802, 2015.

[2] Alex Blewitt, Alan Bundy, and Ian Stark. Automatic verification of java design patterns. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 324–327. IEEE, 2001.

[3] Peng-Hua Chu, Nien-Lin Hsueh, Hong-Hsiang Chen, and Chien-Hung Liu. A test case refactoring approach for pattern-based software development. *Software Quality Journal*, 20(1):43–75, 2012.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[5] Alan R Graves and Chris Czarnecki. Design patterns for behavior-based robotics. *IEEE Transactions on Systems, Man, and cybernetics-part A: systems and humans*, 30(1):36–41, 2000.

[6] Dirk Heuzeroth, Thomas Holl, Gustav Hogstrom, and Welf Lowe. Automatic design pattern detection. In *Program Comprehension, 2003. 11th IEEE International Workshop on*, pages 94–103. IEEE, 2003.

[7] Nien-Lin Hsueh, Peng-Hua Chu, and William Chu. A quantitative approach for evaluating the quality of design patterns. *Journal of Systems and Software*, 81(8):1430–1439, 2008.

[8] Brian Huston. The effects of design pattern application on metric scores. *Journal of Systems and Software*, 58(3):261–269, 2001.

[9] Clemente Izurieta and James M Bieman. A multiple case study of design pattern decay, grime, and rot in evolving software systems. *Software Quality Journal*, 21(2):289–323, 2013.

[10] TH Ng, SC Cheung, WK Chan, and Yuen-Tak Yu. Do maintainers utilize deployed design patterns effectively? In *Proceedings of the 29th international conference on Software Engineering*, pages 168–177. IEEE Computer Society, 2007.

[11] Michael Oduor, Tuomas Alahäivälä, and Harri Oinas-Kukkonen. Persuasive software design patterns for social influence. *Personal and ubiquitous computing*, 18(7):1689–1704, 2014.

[12] Cagri Sahin, Furkan Cayci, Irene Lizeth Manotas Gutiérrez, James Clause, Fouad Kiamilev, Lori Pollock, and Kristina Winbladh. Initial explorations on design pattern energy usage. In *Green and Sustainable Software (GREENS), 2012 First International Workshop on*, pages 55–61. IEEE, 2012.

[13] Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, and Spyros T Halkidis. Design pattern detection using similarity scoring. *Software Engineering, IEEE Transactions on*, 32(11):896–909, 2006.

[14] Marco Zanoni, Francesca Arcelli Fontana, and Fabio Stella. On applying machine learning techniques for design pattern detection. *Journal of Systems and Software*, 103:102–117, 2015.

[15] Cheng Zhang and David Budgen. What do we know about the effectiveness of software design patterns? *Software Engineering, IEEE Transactions on*, 38(5):1213–1231, 2012.

[16] Cheng Zhang and David Budgen. A survey of experienced user perceptions about software design patterns. *Information and Software Technology*, 55(5):822–835, 2013.

[17] Chunying Zhao, Jun Kong, and Kang Zhang. Design pattern evolution and verification using graph transformation. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 290a–290a. IEEE, 2007.

1D Artificial Intelligence

Chairman: 郭忠義/台北科技大學，時間: 10:40-12:00, 7/7，地點: 學思樓學 104

ID	作者	論文名稱
25	蔡佳翰, 蘇翊翔, 謝孟諺	以中文關鍵字為基礎之隱性回饋的評分計算
41	Cheng-Yi Chang and Ing-Yi Chen	基於 Google Cloud Platform 設計高效能日誌分析平台之研究
56	Cheng Wei Wu, Yi Ren, Muhammad Alfiansyah, Hsin Wei Kao, Chen Wei Hsin and Yu-Chee Tseng	基於智慧購物車之排隊辨識
78	詹于瑩, 李婉如, 吳紹薇, 李心瑜, 呂孟蘋, 陳奕中, 陳錫民	GPS 定位影像補償系統
57	郭忠義, 呂紹清	M2M 語意規則推論應用架構設計研究

Paper Title

Author One name, Author Two Name
Department of Your Department
TCSE University
City, Country
Email: {email1, email2}@domain.name

***Abstract*—Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear.**

I. Introduction

In the current trend, design pattern has been widely used for improving software quality. In the academic research, design patterns is one of important research area and applied in different areas [5], [11], [12], many design pattern detection approaches are proposed [6], [13], [14], some work discuss pattern quality metrics [1], [7], [8], design pattern formalization and specification [**?**], [**?**], [**?**], and its benefits [9], [10], [15], [16].

However, using design patterns is not easy, ... Therefore, several approaches are proposed to check the violence of pattern application in a system [2], [17]. We think only static checking to pattern structure is not enough, dynamic testing to test the patterns' semantics is necessary.

In this paper, we propose a test model for design patterns. ....

For example, when we apply *Singleton* pattern, we should conduct the following tests: creating two objects and check if they have the same references. Another example is *Strategy*, when we applied strategy a with this paper, and replace it by another algorithm b, they should have the same behavior, that is, the same output. The following code shows the testing context:

```
1  class TestStrategyDesignPattern {
2      void testSameResult() {
3          Context c = new Context();
4          Strategy a = new StrategyA();
5          c.setStrategy(a);
6          int resultA = c.doIt();
7          Strategy b = new StrategyB();
8          c.setStrategy(b);
9          int restultB = c.doIt();
10         assertEquals(a, b); // should have same
               result
11     }
12 }
```

A. Paper outline

The remainder of this paper is structured as follows: In section 2, we describe the related work to this research.

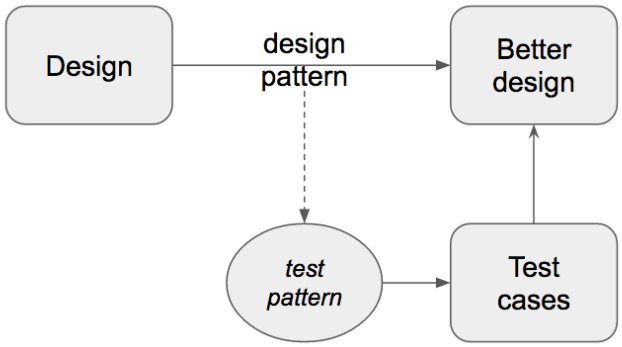


Figure 1. The concept of *test pattern*

Section 3 introduces the details of our approach by two design pattern examples. Section 4 summarizes our approach and future work.

II. Related work

以下針對此研究的相關研究做探討分析：

A. Pattern testing

Chu and Hsueh proposed test refactoring approach in a pattern driven developent [3]. Test-first strategy and code refactoring are the important practices of Extreme Programming for rapid development and quality support. ... However, when the developers perform a pattern-based refactoring to improve the quality, the effort of revising the test cases is much more than that in simple code refactoring (see Fig. 2).

To address this problem, ...

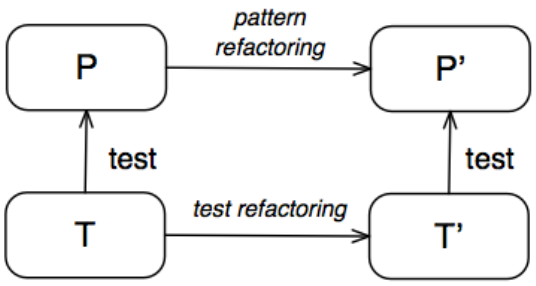


Figure 2. Pattern based test case refactoring

To understand the application context for a pattern in a system, Lin proposed a *Design Pattern Unit Test (DPUT)* approach, which utilizes java Annotation skill to record the pattern utilization and verifies with the expection in the DPUT. This research also design a software framework to help developers design the DPUP in a specification basis. The code is implemented as an Eclipsed plug-in which can automatically transform DPUT into class diagram for better understanding. For the system maintainers they can find out the errors in the earlier phases.

B. Pattern evaluation

設計樣式的效用一直是重要的研究議題。也因此許多學者專家開始投入這方面的研究。Huston [8] 設計了一個實驗以驗證設計樣式是否與軟體衡量相容，亦即，探討應用設計樣式所提昇的軟體品質是否真的可由軟體衡量顯示出來？Huston 的實驗將設計樣式的結構還原成非設計樣式的結構，並以相關的軟體衡量 (software metric) 分別衡量這兩個不同的架構，來比較有無採用設計架構在該軟體衡量上的差異。此實驗顯示設計樣式的品質效能可以透過軟體衡量顯現出來。

III. Our approach

To design the test model for pattern testing, we have to consider the following issues:

- Correctness and Precision. Correctness considers if all faults can be explored by the proposed test cases, and Precision considers if the test can explore the faults in an efficient way. Our test model should satisfy correctness and precision. ...
- Coverage. In general we have many coverage strategies for testing, for example statement coverage or branch coverage. When we test we should enlarge the coverage as we can. Therefore we should care about the normal cases, boundary cases and exception cases. When we utilize this concept to testing a pattern-applied system, what are the boundary cases and exception cases? ...
- Generalization. Gamma et al. classified patterns into creational patterns, structural patterns and behavioral patterns [4]. Patterns in the same group may have same testing issues and methods. We can modularize our testing programs by class generalization. ...

a) Potential Pattern Violation (PPV): To test the pattern-applied application more efficiently, ...

b) Test Pattern for Design Pattern (TP4DP): One of the benefits of design pattern is it is well structured and contains executable code for programmers. Our TP4DP follows the same concept. A test pattern has the following sections: ...

A. Testing "Singleton" pattern

a) Pattern name: Singleton: The intent of the singleton is to ensure a class has only one instance, and provide a global point of access to it.

b) Potential pattern violation: There are two PPV in the Singleton pattern:

- PPV1: When we create two objects from the same class, they refer to different objects.
- PPV2: The original object constructor is not set to be private.

c) Testing guideline: Create two objects of the same Singleton class, check if there are equal by using the operator "==".

d) Demo code: Here is the demo code for the class *Radio*

```
1 class TestRadio {
2     void testRadioSingletonPPE1() {
3         Radio r1 = Radio.instance();
4         Radio r2 = Radio.instance();
5         assertTrue(r1==r2): "Singleton is violated
                ";
6     }
7 }
```

To test the PPV2, we use static testing to check if the constructor is declared as **private**. The example below is the one that doesn't pass the PPV2 checking.

```
1 class Radio {
2     static Radio instance;
3     static Radio getInstance() {
4         if (instance == null)
5             instance = new Radio();
6         }
7         return instance;
8     }
9 }
```

IV. Conclusion

In this paper we propose an idea of pattern test for guiding the programs with patterns applied. We believe a program is error-prone when they are designed in delicate and complex structure. Design patterns are good for flexible design but not easy to understand and apply. For pattern beginners we need a guideline, framework or tool to help them. We demonstrate two examples of *Singleton* and *Observer* to illustrate our ideas. In the future we will explore more design patterns and develop a framework for the pattern test.

References

[1] Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Sofia Charalampidou, and Paris Avgeriou. The effect of gof design patterns on stability: a case study. *IEEE Transactions on Software Engineering*, 41(8):781–802, 2015.

[2] Alex Blewitt, Alan Bundy, and Ian Stark. Automatic verification of java design patterns. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 324–327. IEEE, 2001.

[3] Peng-Hua Chu, Nien-Lin Hsueh, Hong-Hsiang Chen, and Chien-Hung Liu. A test case refactoring approach for pattern-based software development. *Software Quality Journal*, 20(1):43–75, 2012.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[5] Alan R Graves and Chris Czarnecki. Design patterns for behavior-based robotics. *IEEE Transactions on Systems, Man, and cybernetics-part A: systems and humans*, 30(1):36–41, 2000.

[6] Dirk Heuzeroth, Thomas Holl, Gustav Hogstrom, and Welf Lowe. Automatic design pattern detection. In *Program Comprehension, 2003. 11th IEEE International Workshop on*, pages 94–103. IEEE, 2003.

[7] Nien-Lin Hsueh, Peng-Hua Chu, and William Chu. A quantitative approach for evaluating the quality of design patterns. *Journal of Systems and Software*, 81(8):1430–1439, 2008.

[8] Brian Huston. The effects of design pattern application on metric scores. *Journal of Systems and Software*, 58(3):261–269, 2001.

[9] Clemente Izurieta and James M Bieman. A multiple case study of design pattern decay, grime, and rot in evolving software systems. *Software Quality Journal*, 21(2):289–323, 2013.

[10] TH Ng, SC Cheung, WK Chan, and Yuen-Tak Yu. Do maintainers utilize deployed design patterns effectively? In *Proceedings of the 29th international conference on Software Engineering*, pages 168–177. IEEE Computer Society, 2007.

[11] Michael Oduor, Tuomas Alahäivälä, and Harri Oinas-Kukkonen. Persuasive software design patterns for social influence. *Personal and ubiquitous computing*, 18(7):1689–1704, 2014.

[12] Cagri Sahin, Furkan Cayci, Irene Lizeth Manotas Gutiérrez, James Clause, Fouad Kiamilev, Lori Pollock, and Kristina Winbladh. Initial explorations on design pattern energy usage. In *Green and Sustainable Software (GREENS), 2012 First International Workshop on*, pages 55–61. IEEE, 2012.

[13] Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, and Spyros T Halkidis. Design pattern detection using similarity scoring. *Software Engineering, IEEE Transactions on*, 32(11):896–909, 2006.

[14] Marco Zanoni, Francesca Arcelli Fontana, and Fabio Stella. On applying machine learning techniques for design pattern detection. *Journal of Systems and Software*, 103:102–117, 2015.

[15] Cheng Zhang and David Budgen. What do we know about the effectiveness of software design patterns? *Software Engineering, IEEE Transactions on*, 38(5):1213–1231, 2012.

[16] Cheng Zhang and David Budgen. A survey of experienced user perceptions about software design patterns. *Information and Software Technology*, 55(5):822–835, 2013.

[17] Chunying Zhao, Jun Kong, and Kang Zhang. Design pattern evolution and verification using graph transformation. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 290a–290a. IEEE, 2007.

Section 2

English paper, IoT and Industry Talk I

時間: 13:20-14:50, 7/7

2A Best English paper

Chairman: 周忠信/東海大學，委員：劉立頌/中正大學，馬尚彬/海洋大學，時間: 13:20-14:50, 7/7，地點: 學思樓學 101

ID	作者	論文名稱
30	Ru-Wei Fu and Farn Wang	Automatic Device Farm Management for the Testing of Android Mobile Apps
45	Chi Wen Chen and Farn Wang	Automated Cloud Sandbox Deployment for Implementing DevOps
53	Min-Huang Ho, Win-Tsung Lo, Ruey-Kai Sheu, Yen-Lin Lee and Deron Liang	Method of Distributed Node Management for High-Availability Clusters based on Kernel Virtual Machine
63	Hwai-Jung Hsu and Yves Lin	How Agile Works in a Software Corporation: An Empirical Study of Assessing Agile Methods from Viewpoints of Business Data Analytics
75	Chien-Hung Liu and Woie-Kae Chen	Coupling Analysis and Visualization of KDT Scripts
77	Tze Suen Lim, Yi Chung Chen, Sheng Min Chiu, Wei Lun Wang and Wei Hung Lin	Time Series Skyline Query and Its Neural Filter

Paper Title

Author One name, Author Two Name
Department of Your Department
TCSE University
City, Country
Email: {email1, email2}@domain.name

***Abstract*—Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear.**

I. Introduction

In the current trend, design pattern has been widely used for improving software quality. In the academic research, design patterns is one of important research area and applied in different areas [5], [11], [12], many design pattern detection approaches are proposed [6], [13], [14], some work discuss pattern quality metrics [1], [7], [8], design pattern formalization and specification [**?**], [**?**], [**?**], and its benefits [9], [10], [15], [16].

However, using design patterns is not easy, ... Therefore, several approaches are proposed to check the violence of pattern application in a system [2], [17]. We think only static checking to pattern structure is not enough, dynamic testing to test the patterns' semantics is necessary.

In this paper, we propose a test model for design patterns. ....

For example, when we apply *Singleton* pattern, we should conduct the following tests: creating two objects and check if they have the same references. Another example is *Strategy*, when we applied strategy a with this paper, and replace it by another algorithm b, they should have the same behavior, that is, the same output. The following code shows the testing context:

```
1  class TestStrategyDesignPattern {
2      void testSameResult() {
3          Context c = new Context();
4          Strategy a = new StrategyA();
5          c.setStrategy(a);
6          int resultA = c.doIt();
7          Strategy b = new StrategyB();
8          c.setStrategy(b);
9          int restultB = c.doIt();
10         assertEquals(a, b); // should have same
               result
11     }
12 }
```

A. Paper outline

The remainder of this paper is structured as follows: In section 2, we describe the related work to this research.

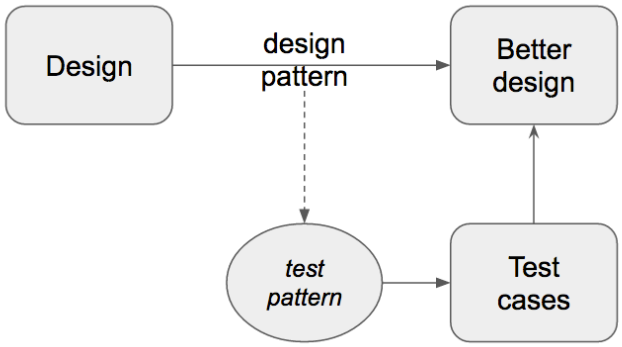


Figure 1. The concept of *test pattern*

Section 3 introduces the details of our approach by two design pattern examples. Section 4 summarizes our approach and future work.

II. Related work

以下針對此研究的相關研究做探討分析：

A. Pattern testing

Chu and Hsueh proposed test refactoring approach in a pattern driven developent [3]. Test-first strategy and code refactoring are the important practices of Extreme Programming for rapid development and quality support. ... However, when the developers perform a pattern-based refactoring to improve the quality, the effort of revising the test cases is much more than that in simple code refactoring (see Fig. 2).

To address this problem, ...

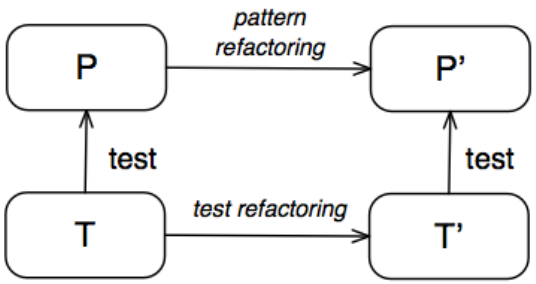


Figure 2. Pattern based test case refactoring

To understand the application context for a pattern in a system, Lin proposed a *Design Pattern Unit Test (DPUT)* approach, which utilizes java Annotation skill to record the pattern utilization and verifies with the expection in the DPUT. This research also design a software framework to help developers design the DPUP in a specification basis. The code is implemented as an Eclipsed plug-in which can automatically transform DPUT into class diagram for better understanding. For the system maintainers they can find out the errors in the earlier phases.

B. Pattern evaluation

設計樣式的效用一直是重要的研究議題。也因此許多學者專家開始投入這方面的研究。Huston [8] 設計了一個實驗以驗證設計樣式是否與軟體衡量相容，亦即，探討應用設計樣式所提昇的軟體品質是否真的可由軟體衡量顯示出來？Huston 的實驗將設計樣式的結構還原成非設計樣式的結構，並以相關的軟體衡量 (software metric) 分別衡量這兩個不同的架構，來比較有無採用設計架構在該軟體衡量上的差異。此實驗顯示設計樣式的品質效能可以透過軟體衡量顯現出來。

III. Our approach

To design the test model for pattern testing, we have to consider the following issues:

- Correctness and Precision. Correctness considers if all faults can be explored by the proposed test cases, and Precision considers if the test can explore the faults in an efficient way. Our test model should satisfy correctness and precision. ...
- Coverage. In general we have many coverage strategies for testing, for example statement coverage or branch coverage. When we test we should enlarge the coverage as we can. Therefore we should care about the normal cases, boundary cases and exception cases. When we utilize this concept to testing a pattern-applied system, what are the boundary cases and exception cases? ...
- Generalization. Gamma et al. classified patterns into creational patterns, structural patterns and behavioral patterns [4]. Patterns in the same group may have same testing issues and methods. We can modularize our testing programs by class generalization. ...

a) Potential Pattern Violation (PPV): To test the pattern-applied application more efficiently, ...

b) Test Pattern for Design Pattern (TP4DP): One of the benefits of design pattern is it is well structured and contains executable code for programmers. Our TP4DP follows the same concept. A test pattern has the following sections: ...

A. Testing "Singleton" pattern

a) Pattern name: Singleton: The intent of the singleton is to ensure a class has only one instance, and provide a global point of access to it.

b) Potential pattern violation: There are two PPV in the Singleton pattern:

- PPV1: When we create two objects from the same class, they refer to different objects.
- PPV2: The original object constructor is not set to be private.

c) Testing guideline: Create two objects of the same Singleton class, check if there are equal by using the operator "==".

d) Demo code: Here is the demo code for the class *Radio*

```
1 class TestRadio {
2     void testRadioSingletonPPE1() {
3         Radio r1 = Radio.instance();
4         Radio r2 = Radio.instance();
5         assertTrue(r1==r2): "Singleton is violated
                ";
6     }
7 }
```

To test the PPV2, we use static testing to check if the constructor is declared as **private**. The example below is the one that doesn't pass the PPV2 checking.

```
1 class Radio {
2     static Radio instance;
3     static Radio getInstance() {
4         if (instance == null)
5             instance = new Radio();
6         }
7         return instance;
8     }
9 }
```

IV. Conclusion

In this paper we propose an idea of pattern test for guiding the programs with patterns applied. We believe a program is error-prone when they are designed in delicate and complex structure. Design patterns are good for flexible design but not easy to understand and apply. For pattern beginners we need a guideline, framework or tool to help them. We demonstrate two examples of *Singleton* and *Observer* to illustrate our ideas. In the future we will explore more design patterns and develop a framework for the pattern test.

References

[1] Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Sofia Charalampidou, and Paris Avgeriou. The effect of gof design patterns on stability: a case study. *IEEE Transactions on Software Engineering*, 41(8):781–802, 2015.

[2] Alex Blewitt, Alan Bundy, and Ian Stark. Automatic verification of java design patterns. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 324–327. IEEE, 2001.

[3] Peng-Hua Chu, Nien-Lin Hsueh, Hong-Hsiang Chen, and Chien-Hung Liu. A test case refactoring approach for pattern-based software development. *Software Quality Journal*, 20(1):43–75, 2012.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[5] Alan R Graves and Chris Czarnecki. Design patterns for behavior-based robotics. *IEEE Transactions on Systems, Man, and cybernetics-part A: systems and humans*, 30(1):36–41, 2000.

[6] Dirk Heuzeroth, Thomas Holl, Gustav Hogstrom, and Welf Lowe. Automatic design pattern detection. In *Program Comprehension, 2003. 11th IEEE International Workshop on*, pages 94–103. IEEE, 2003.

[7] Nien-Lin Hsueh, Peng-Hua Chu, and William Chu. A quantitative approach for evaluating the quality of design patterns. *Journal of Systems and Software*, 81(8):1430–1439, 2008.

[8] Brian Huston. The effects of design pattern application on metric scores. *Journal of Systems and Software*, 58(3):261–269, 2001.

[9] Clemente Izurieta and James M Bieman. A multiple case study of design pattern decay, grime, and rot in evolving software systems. *Software Quality Journal*, 21(2):289–323, 2013.

[10] TH Ng, SC Cheung, WK Chan, and Yuen-Tak Yu. Do maintainers utilize deployed design patterns effectively? In *Proceedings of the 29th international conference on Software Engineering*, pages 168–177. IEEE Computer Society, 2007.

[11] Michael Oduor, Tuomas Alahäivälä, and Harri Oinas-Kukkonen. Persuasive software design patterns for social influence. *Personal and ubiquitous computing*, 18(7):1689–1704, 2014.

[12] Cagri Sahin, Furkan Cayci, Irene Lizeth Manotas Gutiérrez, James Clause, Fouad Kiamilev, Lori Pollock, and Kristina Winbladh. Initial explorations on design pattern energy usage. In *Green and Sustainable Software (GREENS), 2012 First International Workshop on*, pages 55–61. IEEE, 2012.

[13] Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, and Spyros T Halkidis. Design pattern detection using similarity scoring. *Software Engineering, IEEE Transactions on*, 32(11):896–909, 2006.

[14] Marco Zanoni, Francesca Arcelli Fontana, and Fabio Stella. On applying machine learning techniques for design pattern detection. *Journal of Systems and Software*, 103:102–117, 2015.

[15] Cheng Zhang and David Budgen. What do we know about the effectiveness of software design patterns? *Software Engineering, IEEE Transactions on*, 38(5):1213–1231, 2012.

[16] Cheng Zhang and David Budgen. A survey of experienced user perceptions about software design patterns. *Information and Software Technology*, 55(5):822–835, 2013.

[17] Chunying Zhao, Jun Kong, and Kang Zhang. Design pattern evolution and verification using graph transformation. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 290a–290a. IEEE, 2007.

2B English paper

Chairman: 林迺衛/中正大學，時間: 13:20-14:50, 7/7，地點: 學思樓學 102

ID	作者	論文名稱
2	Chun-Hsiung Tseng, Lin Hui, Yung-Hui Chen and Jia-Long Li	A GPS Navigation System Leveraging Voice Based User Interface for Blind People
28	Jiun-Hao Lin and Farn Wang	User-Friendly Trace Viewer for Android Apps Testing
31	Jui Chieh Tai and Farn Wang	Cross-Browser Compatibility Testing of Web Applications
34	Guang-Qi Wang and Farn	Wang Automated Testing for Quality Android Applications
40	Jyun-Hua Jiang, Lin-Huang Chang and Tsung-Han Lee	Wireless sensor network under asynchronous mechanism to dynamically adjust the sleep schedule
52	Ming-Chi Liu and Yueh-Min Huang	Performing a phenomenographic text mining to understand the students' experiences of software programming

Paper Title

Author One name, Author Two Name
Department of Your Department
TCSE University
City, Country
Email: {email1, email2}@domain.name

***Abstract*—Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear.**

I. Introduction

In the current trend, design pattern has been widely used for improving software quality. In the academic research, design patterns is one of important research area and applied in different areas [5], [11], [12], many design pattern detection approaches are proposed [6], [13], [14], some work discuss pattern quality metrics [1], [7], [8], design pattern formalization and specification [**?**], [**?**], [**?**], and its benefits [9], [10], [15], [16].

However, using design patterns is not easy, ... Therefore, several approaches are proposed to check the violence of pattern application in a system [2], [17]. We think only static checking to pattern structure is not enough, dynamic testing to test the patterns' semantics is necessary.

In this paper, we propose a test model for design patterns. ....

For example, when we apply *Singleton* pattern, we should conduct the following tests: creating two objects and check if they have the same references. Another example is *Strategy*, when we applied strategy a with this paper, and replace it by another algorithm b, they should have the same behavior, that is, the same output. The following code shows the testing context:

```
1  class TestStrategyDesignPattern {
2      void testSameResult() {
3          Context c = new Context();
4          Strategy a = new StrategyA();
5          c.setStrategy(a);
6          int resultA = c.doIt();
7          Strategy b = new StrategyB();
8          c.setStrategy(b);
9          int restultB = c.doIt();
10         assertEquals(a, b); // should have same
               result
11     }
12 }
```

A. Paper outline

The remainder of this paper is structured as follows: In section 2, we describe the related work to this research.

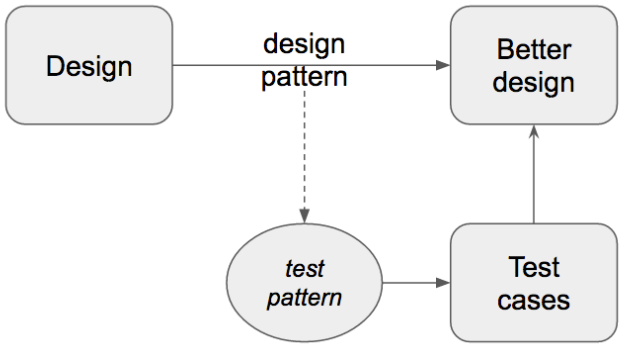


Figure 1. The concept of *test pattern*

Section 3 introduces the details of our approach by two design pattern examples. Section 4 summarizes our approach and future work.

II. Related work

以下針對此研究的相關研究做探討分析：

A. Pattern testing

Chu and Hsueh proposed test refactoring approach in a pattern driven developent [3]. Test-first strategy and code refactoring are the important practices of Extreme Programming for rapid development and quality support. ... However, when the developers perform a pattern-based refactoring to improve the quality, the effort of revising the test cases is much more than that in simple code refactoring (see Fig. 2).

To address this problem, ...

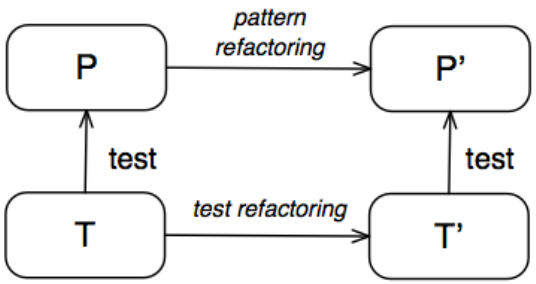


Figure 2. Pattern based test case refactoring

To understand the application context for a pattern in a system, Lin proposed a *Design Pattern Unit Test (DPUT)* approach, which utilizes java Annotation skill to record the pattern utilization and verifies with the expection in the DPUT. This research also design a software framework to help developers design the DPUP in a specification basis. The code is implemented as an Eclipsed plug-in which can automatically transform DPUT into class diagram for better understanding. For the system maintainers they can find out the errors in the earlier phases.

B. Pattern evaluation

設計樣式的效用一直是重要的研究議題。也因此許多學者專家開始投入這方面的研究。Huston [8] 設計了一個實驗以驗證設計樣式是否與軟體衡量相容，亦即，探討應用設計樣式所提昇的軟體品質是否真的可由軟體衡量顯示出來？Huston 的實驗將設計樣式的結構還原成非設計樣式的結構，並以相關的軟體衡量 (software metric) 分別衡量這兩個不同的架構，來比較有無採用設計架構在該軟體衡量上的差異。此實驗顯示設計樣式的品質效能可以透過軟體衡量顯現出來。

III. Our approach

To design the test model for pattern testing, we have to consider the following issues:

- Correctness and Precision. Correctness considers if all faults can be explored by the proposed test cases, and Precision considers if the test can explore the faults in an efficient way. Our test model should satisfy correctness and precision. ...
- Coverage. In general we have many coverage strategies for testing, for example statement coverage or branch coverage. When we test we should enlarge the coverage as we can. Therefore we should care about the normal cases, boundary cases and exception cases. When we utilize this concept to testing a pattern-applied system, what are the boundary cases and exception cases? ...
- Generalization. Gamma et al. classified patterns into creational patterns, structural patterns and behavioral patterns [4]. Patterns in the same group may have same testing issues and methods. We can modularize our testing programs by class generalization. ...

a) Potential Pattern Violation (PPV): To test the pattern-applied application more efficiently, ...

b) Test Pattern for Design Pattern (TP4DP): One of the benefits of design pattern is it is well structured and contains executable code for programmers. Our TP4DP follows the same concept. A test pattern has the following sections: ...

A. Testing "Singleton" pattern

a) Pattern name: Singleton: The intent of the singleton is to ensure a class has only one instance, and provide a global point of access to it.

b) Potential pattern violation: There are two PPV in the Singleton pattern:

- PPV1: When we create two objects from the same class, they refer to different objects.
- PPV2: The original object constructor is not set to be private.

c) Testing guideline: Create two objects of the same Singleton class, check if there are equal by using the operator "==".

d) Demo code: Here is the demo code for the class *Radio*

```
1 class TestRadio {
2     void testRadioSingletonPPE1() {
3         Radio r1 = Radio.instance();
4         Radio r2 = Radio.instance();
5         assertTrue(r1==r2): "Singleton is violated
                ";
6     }
7 }
```

To test the PPV2, we use static testing to check if the constructor is declared as **private**. The example below is the one that doesn't pass the PPV2 checking.

```
1 class Radio {
2     static Radio instance;
3     static Radio getInstance() {
4         if (instance == null)
5             instance = new Radio();
6         }
7         return instance;
8     }
9 }
```

IV. Conclusion

In this paper we propose an idea of pattern test for guiding the programs with patterns applied. We believe a program is error-prone when they are designed in delicate and complex structure. Design patterns are good for flexible design but not easy to understand and apply. For pattern beginners we need a guideline, framework or tool to help them. We demonstrate two examples of *Singleton* and *Observer* to illustrate our ideas. In the future we will explore more design patterns and develop a framework for the pattern test.

References

[1] Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Sofia Charalampidou, and Paris Avgeriou. The effect of gof design patterns on stability: a case study. *IEEE Transactions on Software Engineering*, 41(8):781–802, 2015.

[2] Alex Blewitt, Alan Bundy, and Ian Stark. Automatic verification of java design patterns. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 324–327. IEEE, 2001.

[3] Peng-Hua Chu, Nien-Lin Hsueh, Hong-Hsiang Chen, and Chien-Hung Liu. A test case refactoring approach for pattern-based software development. *Software Quality Journal*, 20(1):43–75, 2012.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[5] Alan R Graves and Chris Czarnecki. Design patterns for behavior-based robotics. *IEEE Transactions on Systems, Man, and cybernetics-part A: systems and humans*, 30(1):36–41, 2000.

[6] Dirk Heuzeroth, Thomas Holl, Gustav Hogstrom, and Welf Lowe. Automatic design pattern detection. In *Program Comprehension, 2003. 11th IEEE International Workshop on*, pages 94–103. IEEE, 2003.

[7] Nien-Lin Hsueh, Peng-Hua Chu, and William Chu. A quantitative approach for evaluating the quality of design patterns. *Journal of Systems and Software*, 81(8):1430–1439, 2008.

[8] Brian Huston. The effects of design pattern application on metric scores. *Journal of Systems and Software*, 58(3):261–269, 2001.

[9] Clemente Izurieta and James M Bieman. A multiple case study of design pattern decay, grime, and rot in evolving software systems. *Software Quality Journal*, 21(2):289–323, 2013.

[10] TH Ng, SC Cheung, WK Chan, and Yuen-Tak Yu. Do maintainers utilize deployed design patterns effectively? In *Proceedings of the 29th international conference on Software Engineering*, pages 168–177. IEEE Computer Society, 2007.

[11] Michael Oduor, Tuomas Alahäivälä, and Harri Oinas-Kukkonen. Persuasive software design patterns for social influence. *Personal and ubiquitous computing*, 18(7):1689–1704, 2014.

[12] Cagri Sahin, Furkan Cayci, Irene Lizeth Manotas Gutiérrez, James Clause, Fouad Kiamilev, Lori Pollock, and Kristina Winbladh. Initial explorations on design pattern energy usage. In *Green and Sustainable Software (GREENS), 2012 First International Workshop on*, pages 55–61. IEEE, 2012.

[13] Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, and Spyros T Halkidis. Design pattern detection using similarity scoring. *Software Engineering, IEEE Transactions on*, 32(11):896–909, 2006.

[14] Marco Zanoni, Francesca Arcelli Fontana, and Fabio Stella. On applying machine learning techniques for design pattern detection. *Journal of Systems and Software*, 103:102–117, 2015.

[15] Cheng Zhang and David Budgen. What do we know about the effectiveness of software design patterns? *Software Engineering, IEEE Transactions on*, 38(5):1213–1231, 2012.

[16] Cheng Zhang and David Budgen. A survey of experienced user perceptions about software design patterns. *Information and Software Technology*, 55(5):822–835, 2013.

[17] Chunying Zhao, Jun Kong, and Kang Zhang. Design pattern evolution and verification using graph transformation. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 290a–290a. IEEE, 2007.

2C IoT

Chairman: 廖峻峰/政治大學，時間: 13:20-14:50, 7/7，地點: 學思樓學 103

ID	作者	論文名稱
8	邱大洲, 陳文輝	以智慧型手機感測器結合機器學習演算法之雲端居家行為辨識系統
43	徐士展, 戴偉竹, 盧韋宏, 蔡漢霖, 趙宥勝, 劉立頌	具易用性之智慧家庭控制系統 (A Controlling System in Smart Home with Usability)
83	李允中, 任哲晨, 吳佳芷	物聯網中介軟體：感測器服務及網路服務與複雜事件處理之整合
68	郭家旭, 李文廷, 馬毓棣, 林敬祥	iRollCall - NFC 輕量級行動點名服務系統
71	Chia-Hsu Kuo, Wen-An Tsai and Tzung-Shi Chen	無線感測網路的行動充電策略之研究
32	陳映如, 廖峻鋒	資源導向智慧家庭服務維運機制的設計與實現
33	盧威辰, 廖峻鋒	適用於數位互動藝術的聚合式 BLE-MQTT 閘道設計

Paper Title

Author One name, Author Two Name
Department of Your Department
TCSE University
City, Country
Email: {email1, email2}@domain.name

***Abstract*—Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear.**

I. Introduction

In the current trend, design pattern has been widely used for improving software quality. In the academic research, design patterns is one of important research area and applied in different areas [5], [11], [12], many design pattern detection approaches are proposed [6], [13], [14], some work discuss pattern quality metrics [1], [7], [8], design pattern formalization and specification [**?**], [**?**], [**?**], and its benefits [9], [10], [15], [16].

However, using design patterns is not easy, ... Therefore, several approaches are proposed to check the violence of pattern application in a system [2], [17]. We think only static checking to pattern structure is not enough, dynamic testing to test the patterns' semantics is necessary.

In this paper, we propose a test model for design patterns. ....

For example, when we apply *Singleton* pattern, we should conduct the following tests: creating two objects and check if they have the same references. Another example is *Strategy*, when we applied strategy a with this paper, and replace it by another algorithm b, they should have the same behavior, that is, the same output. The following code shows the testing context:

```
1  class TestStrategyDesignPattern {
2      void testSameResult() {
3          Context c = new Context();
4          Strategy a = new StrategyA();
5          c.setStrategy(a);
6          int resultA = c.doIt();
7          Strategy b = new StrategyB();
8          c.setStrategy(b);
9          int restultB = c.doIt();
10         assertEquals(a, b); // should have same
               result
11     }
12 }
```

A. Paper outline

The remainder of this paper is structured as follows: In section 2, we describe the related work to this research.

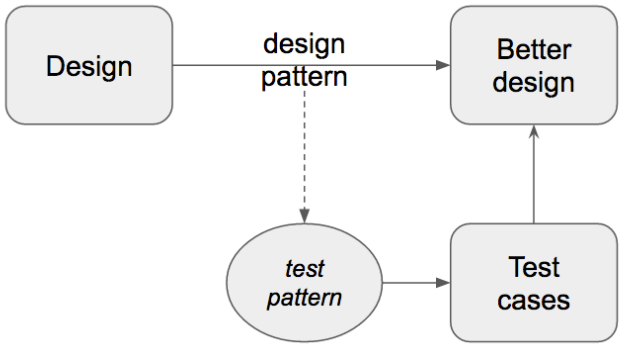


Figure 1. The concept of *test pattern*

Section 3 introduces the details of our approach by two design pattern examples. Section 4 summarizes our approach and future work.

II. Related work

以下針對此研究的相關研究做探討分析：

A. Pattern testing

Chu and Hsueh proposed test refactoring approach in a pattern driven developent [3]. Test-first strategy and code refactoring are the important practices of Extreme Programming for rapid development and quality support. ... However, when the developers perform a pattern-based refactoring to improve the quality, the effort of revising the test cases is much more than that in simple code refactoring (see Fig. 2).

To address this problem, ...

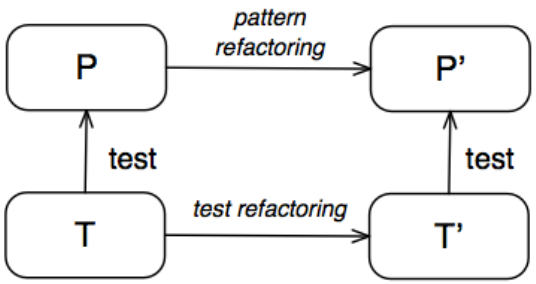


Figure 2. Pattern based test case refactoring

To understand the application context for a pattern in a system, Lin proposed a *Design Pattern Unit Test (DPUT)* approach, which utilizes java Annotation skill to record the pattern utilization and verifies with the expection in the DPUT. This research also design a software framework to help developers design the DPUP in a specification basis. The code is implemented as an Eclipsed plug-in which can automatically transform DPUT into class diagram for better understanding. For the system maintainers they can find out the errors in the earlier phases.

B. Pattern evaluation

設計樣式的效用一直是重要的研究議題。也因此許多學者專家開始投入這方面的研究。Huston [8] 設計了一個實驗以驗證設計樣式是否與軟體衡量相容，亦即，探討應用設計樣式所提昇的軟體品質是否真的可由軟體衡量顯示出來？Huston 的實驗將設計樣式的結構還原成非設計樣式的結構，並以相關的軟體衡量 (software metric) 分別衡量這兩個不同的架構，來比較有無採用設計架構在該軟體衡量上的差異。此實驗顯示設計樣式的品質效能可以透過軟體衡量顯現出來。

III. Our approach

To design the test model for pattern testing, we have to consider the following issues:

- Correctness and Precision. Correctness considers if all faults can be explored by the proposed test cases, and Precision considers if the test can explore the faults in an efficient way. Our test model should satisfy correctness and precision. ...
- Coverage. In general we have many coverage strategies for testing, for example statement coverage or branch coverage. When we test we should enlarge the coverage as we can. Therefore we should care about the normal cases, boundary cases and exception cases. When we utilize this concept to testing a pattern-applied system, what are the boundary cases and exception cases? ...
- Generalization. Gamma et al. classified patterns into creational patterns, structural patterns and behavioral patterns [4]. Patterns in the same group may have same testing issues and methods. We can modularize our testing programs by class generalization. ...

a) Potential Pattern Violation (PPV): To test the pattern-applied application more efficiently, ...

b) Test Pattern for Design Pattern (TP4DP): One of the benefits of design pattern is it is well structured and contains executable code for programmers. Our TP4DP follows the same concept. A test pattern has the following sections: ...

A. Testing "Singleton" pattern

a) Pattern name: Singleton: The intent of the singleton is to ensure a class has only one instance, and provide a global point of access to it.

b) Potential pattern violation: There are two PPV in the Singleton pattern:

- PPV1: When we create two objects from the same class, they refer to different objects.
- PPV2: The original object constructor is not set to be private.

c) Testing guideline: Create two objects of the same Singleton class, check if there are equal by using the operator "==".

d) Demo code: Here is the demo code for the class *Radio*

```
1 class TestRadio {
2     void testRadioSingletonPPE1() {
3         Radio r1 = Radio.instance();
4         Radio r2 = Radio.instance();
5         assertTrue(r1==r2): "Singleton is violated
                ";
6     }
7 }
```

To test the PPV2, we use static testing to check if the constructor is declared as **private**. The example below is the one that doesn't pass the PPV2 checking.

```
1 class Radio {
2     static Radio instance;
3     static Radio getInstance() {
4         if (instance == null)
5             instance = new Radio();
6         }
7         return instance;
8     }
9 }
```

IV. Conclusion

In this paper we propose an idea of pattern test for guiding the programs with patterns applied. We believe a program is error-prone when they are designed in delicate and complex structure. Design patterns are good for flexible design but not easy to understand and apply. For pattern beginners we need a guideline, framework or tool to help them. We demonstrate two examples of *Singleton* and *Observer* to illustrate our ideas. In the future we will explore more design patterns and develop a framework for the pattern test.

References

[1] Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Sofia Charalampidou, and Paris Avgeriou. The effect of gof design patterns on stability: a case study. *IEEE Transactions on Software Engineering*, 41(8):781–802, 2015.

[2] Alex Blewitt, Alan Bundy, and Ian Stark. Automatic verification of java design patterns. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 324–327. IEEE, 2001.

[3] Peng-Hua Chu, Nien-Lin Hsueh, Hong-Hsiang Chen, and Chien-Hung Liu. A test case refactoring approach for pattern-based software development. *Software Quality Journal*, 20(1):43–75, 2012.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[5] Alan R Graves and Chris Czarnecki. Design patterns for behavior-based robotics. *IEEE Transactions on Systems, Man, and cybernetics-part A: systems and humans*, 30(1):36–41, 2000.

[6] Dirk Heuzeroth, Thomas Holl, Gustav Hogstrom, and Welf Lowe. Automatic design pattern detection. In *Program Comprehension, 2003. 11th IEEE International Workshop on*, pages 94–103. IEEE, 2003.

[7] Nien-Lin Hsueh, Peng-Hua Chu, and William Chu. A quantitative approach for evaluating the quality of design patterns. *Journal of Systems and Software*, 81(8):1430–1439, 2008.

[8] Brian Huston. The effects of design pattern application on metric scores. *Journal of Systems and Software*, 58(3):261–269, 2001.

[9] Clemente Izurieta and James M Bieman. A multiple case study of design pattern decay, grime, and rot in evolving software systems. *Software Quality Journal*, 21(2):289–323, 2013.

[10] TH Ng, SC Cheung, WK Chan, and Yuen-Tak Yu. Do maintainers utilize deployed design patterns effectively? In *Proceedings of the 29th international conference on Software Engineering*, pages 168–177. IEEE Computer Society, 2007.

[11] Michael Oduor, Tuomas Alahäivälä, and Harri Oinas-Kukkonen. Persuasive software design patterns for social influence. *Personal and ubiquitous computing*, 18(7):1689–1704, 2014.

[12] Cagri Sahin, Furkan Cayci, Irene Lizeth Manotas Gutiérrez, James Clause, Fouad Kiamilev, Lori Pollock, and Kristina Winbladh. Initial explorations on design pattern energy usage. In *Green and Sustainable Software (GREENS), 2012 First International Workshop on*, pages 55–61. IEEE, 2012.

[13] Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, and Spyros T Halkidis. Design pattern detection using similarity scoring. *Software Engineering, IEEE Transactions on*, 32(11):896–909, 2006.

[14] Marco Zanoni, Francesca Arcelli Fontana, and Fabio Stella. On applying machine learning techniques for design pattern detection. *Journal of Systems and Software*, 103:102–117, 2015.

[15] Cheng Zhang and David Budgen. What do we know about the effectiveness of software design patterns? *Software Engineering, IEEE Transactions on*, 38(5):1213–1231, 2012.

[16] Cheng Zhang and David Budgen. A survey of experienced user perceptions about software design patterns. *Information and Software Technology*, 55(5):822–835, 2013.

[17] Chunying Zhao, Jun Kong, and Kang Zhang. Design pattern evolution and verification using graph transformation. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 290a–290a. IEEE, 2007.

2D Industry Talk I

Chairman: 英家慶/逢甲大學，時間: 13:20-14:50, 7/7，地點：學思樓第九國際會議廳

ID	作者	論文名稱
	林裕丞總經理 / 新加坡鈦坦科技	空手、緊握、到放手 – 敏捷路上學到的 5 件事
	林俊孝 Chief Technology Officer/ Picowork	協同式雲端作業系統 - 創造雲端社會的新生態

題目：空手、緊握、到放手 – 敏捷路上學到的 5 件事

演講者：林裕丞總經理 / 新加坡鈦坦科技

為什麼吃飽沒事做要跑敏捷？從一開始新加坡 5 人，到現在跨國 150 人，這過程中有喜悅、有掙扎、也有不少的學習和成長。這是一個鈦坦科技十年來的演化史，經由長時間跨度與宏觀組織角度，跟大家分享我們在組織文化和產品面上的改變和學習。在這次演講中您將會有以下收穫：(1) 什麼時候要跑敏捷？(2) 如何評估敏捷的成效？(3) 在組織管理上有那些工具和方法？

題目：協同式雲端作業系統 - 創造雲端社會的新生態

演講者：林俊孝 Chief Technology Officer/ Picowork

現今的生活大量的依賴第三方應用服務來進行，然而不同的應用服務和服務平台間存在著商業經營上、技術架構上、及功能發展上及壁壘（Boundry），帶來產業全方位協同的挑戰。「協同式雲端作業系統」以作業系統的角度出發，透過技術思維上的創新，將整個網際網路看成是全球最大的一部計算機，重新定義了網際網路的定位，使各行各業可以擁有自己的雲端作業系統環境，又可以透過自己的雲端環境，與任何人 (同事、顧客、夥伴、朋友) 自由地進行協同作業，也能以最低的成本，讓過去、現在、及未來的網路資訊系統和網路資源擴充到雲端環境中，最後亦能藉此產生自我的大數據，為創造雲端社會的新生態，帶來了新的可能。

Section 3

Best regular paper, Software testing II, and Demo paper

時間: 15:00-16:30, 7/7

3A Best regular paper

Chairman: 梁德容/中央大學，委員：劉建宏/台北科技大學，孔崇旭/台中教育大學，時間: 15:00-16:30, 7/7，地點: 學思樓學 101

ID	作者	論文名稱
9	廖峻鋒, 鄭敬儒, 陳恭, 賴晨禾, 邱天	基於行為驅動開發製程的區塊鏈智能合約整合測試服務平台
20	徐偉哲, 鄭永斌	Virtual Objects for Program Visualization in xDIVA
49	李信杰, 黃琪恩, 游傑麟	以 Text-Attribute-Context 為基礎識別演化網頁中變動元素之方法與自動化網頁回歸測試之實務應用
60	郭忠義, 潘家偉	應用堆疊式降噪自動編碼器建構學生退學預測模型
65	陳鵬中, 馬尚彬, 呂致緯	LODE: 鏈結開放資料之建立、查詢與服務生成平台
74	Wing Lun Siu, Yi-Chung Chen, Kuo-Cheng Ting, Don-Lin Yang and Hsi-Min Chen	在社群網路中利用 m-代表性天際線查詢搜尋 m-相似的用戶

Paper Title

Author One name, Author Two Name
Department of Your Department
TCSE University
City, Country
Email: {email1, email2}@domain.name

***Abstract*—Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear.**

I. Introduction

In the current trend, design pattern has been widely used for improving software quality. In the academic research, design patterns is one of important research area and applied in different areas [5], [11], [12], many design pattern detection approaches are proposed [6], [13], [14], some work discuss pattern quality metrics [1], [7], [8], design pattern formalization and specification [**?**], [**?**], [**?**], and its benefits [9], [10], [15], [16].

However, using design patterns is not easy, ... Therefore, several approaches are proposed to check the violence of pattern application in a system [2], [17]. We think only static checking to pattern structure is not enough, dynamic testing to test the patterns' semantics is necessary.

In this paper, we propose a test model for design patterns. ....

For example, when we apply *Singleton* pattern, we should conduct the following tests: creating two objects and check if they have the same references. Another example is *Strategy*, when we applied strategy a with this paper, and replace it by another algorithm b, they should have the same behavior, that is, the same output. The following code shows the testing context:

```
1  class TestStrategyDesignPattern {
2      void testSameResult() {
3          Context c = new Context();
4          Strategy a = new StrategyA();
5          c.setStrategy(a);
6          int resultA = c.doIt();
7          Strategy b = new StrategyB();
8          c.setStrategy(b);
9          int restultB = c.doIt();
10         assertEquals(a, b); // should have same
               result
11     }
12 }
```

A. Paper outline

The remainder of this paper is structured as follows: In section 2, we describe the related work to this research.

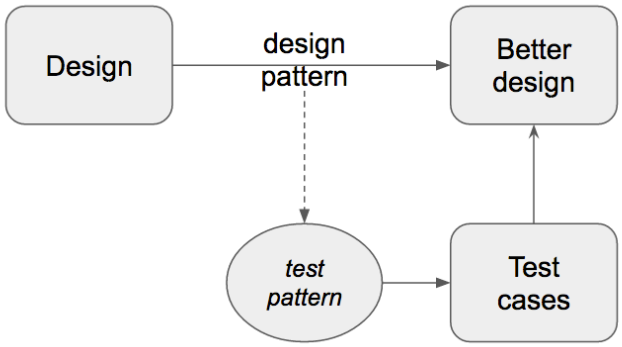


Figure 1. The concept of *test pattern*

Section 3 introduces the details of our approach by two design pattern examples. Section 4 summarizes our approach and future work.

II. Related work

以下針對此研究的相關研究做探討分析：

A. Pattern testing

Chu and Hsueh proposed test refactoring approach in a pattern driven developent [3]. Test-first strategy and code refactoring are the important practices of Extreme Programming for rapid development and quality support. ... However, when the developers perform a pattern-based refactoring to improve the quality, the effort of revising the test cases is much more than that in simple code refactoring (see Fig. 2).

To address this problem, ...

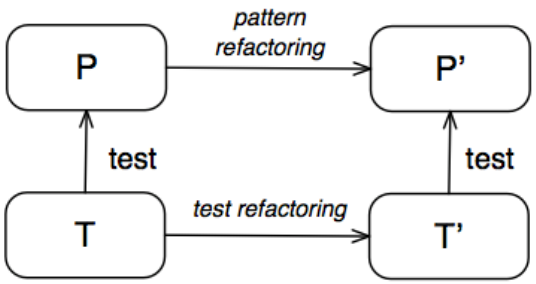


Figure 2. Pattern based test case refactoring

To understand the application context for a pattern in a system, Lin proposed a *Design Pattern Unit Test (DPUT)* approach, which utilizes java Annotation skill to record the pattern utilization and verifies with the expection in the DPUT. This research also design a software framework to help developers design the DPUP in a specification basis. The code is implemented as an Eclipsed plug-in which can automatically transform DPUT into class diagram for better understanding. For the system maintainers they can find out the errors in the earlier phases.

B. Pattern evaluation

設計樣式的效用一直是重要的研究議題。也因此許多學者專家開始投入這方面的研究。Huston [8] 設計了一個實驗以驗證設計樣式是否與軟體衡量相容，亦即，探討應用設計樣式所提昇的軟體品質是否真的可由軟體衡量顯示出來？Huston 的實驗將設計樣式的結構還原成非設計樣式的結構，並以相關的軟體衡量 (software metric) 分別衡量這兩個不同的架構，來比較有無採用設計架構在該軟體衡量上的差異。此實驗顯示設計樣式的品質效能可以透過軟體衡量顯現出來。

III. Our approach

To design the test model for pattern testing, we have to consider the following issues:

- Correctness and Precision. Correctness considers if all faults can be explored by the proposed test cases, and Precision considers if the test can explore the faults in an efficient way. Our test model should satisfy correctness and precision. ...
- Coverage. In general we have many coverage strategies for testing, for example statement coverage or branch coverage. When we test we should enlarge the coverage as we can. Therefore we should care about the normal cases, boundary cases and exception cases. When we utilize this concept to testing a pattern-applied system, what are the boundary cases and exception cases? ...
- Generalization. Gamma et al. classified patterns into creational patterns, structural patterns and behavioral patterns [4]. Patterns in the same group may have same testing issues and methods. We can modularize our testing programs by class generalization. ...

a) Potential Pattern Violation (PPV): To test the pattern-applied application more efficiently, ...

b) Test Pattern for Design Pattern (TP4DP): One of the benefits of design pattern is it is well structured and contains executable code for programmers. Our TP4DP follows the same concept. A test pattern has the following sections: ...

*A. Testing "*Singleton*" pattern*

a) Pattern name: Singleton: The intent of the singleton is to ensure a class has only one instance, and provide a global point of access to it.

b) Potential pattern violation: There are two PPV in the Singleton pattern:

- PPV1: When we create two objects from the same class, they refer to different objects.
- PPV2: The original object constructor is not set to be private.

c) Testing guideline: Create two objects of the same Singleton class, check if there are equal by using the operator "==".

d) Demo code: Here is the demo code for the class *Radio*

```
1 class TestRadio {
2     void testRadioSingletonPPE1() {
3         Radio r1 = Radio.instance();
4         Radio r2 = Radio.instance();
5         assertTrue(r1==r2): "Singleton is violated
                ";
6     }
7 }
```

To test the PPV2, we use static testing to check if the constructor is declared as **private**. The example below is the one that doesn't pass the PPV2 checking.

```
1 class Radio {
2     static Radio instance;
3     static Radio getInstance() {
4         if (instance == null)
5             instance = new Radio();
6         }
7         return instance;
8     }
9 }
```

IV. Conclusion

In this paper we propose an idea of pattern test for guiding the programs with patterns applied. We believe a program is error-prone when they are designed in delicate and complex structure. Design patterns are good for flexible design but not easy to understand and apply. For pattern beginners we need a guideline, framework or tool to help them. We demonstrate two examples of *Singleton* and *Observer* to illustrate our ideas. In the future we will explore more design patterns and develop a framework for the pattern test.

References

[1] Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Sofia Charalampidou, and Paris Avgeriou. The effect of gof design patterns on stability: a case study. *IEEE Transactions on Software Engineering*, 41(8):781–802, 2015.

[2] Alex Blewitt, Alan Bundy, and Ian Stark. Automatic verification of java design patterns. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 324–327. IEEE, 2001.

[3] Peng-Hua Chu, Nien-Lin Hsueh, Hong-Hsiang Chen, and Chien-Hung Liu. A test case refactoring approach for pattern-based software development. *Software Quality Journal*, 20(1):43–75, 2012.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[5] Alan R Graves and Chris Czarnecki. Design patterns for behavior-based robotics. *IEEE Transactions on Systems, Man, and cybernetics-part A: systems and humans*, 30(1):36–41, 2000.

[6] Dirk Heuzeroth, Thomas Holl, Gustav Hogstrom, and Welf Lowe. Automatic design pattern detection. In *Program Comprehension, 2003. 11th IEEE International Workshop on*, pages 94–103. IEEE, 2003.

[7] Nien-Lin Hsueh, Peng-Hua Chu, and William Chu. A quantitative approach for evaluating the quality of design patterns. *Journal of Systems and Software*, 81(8):1430–1439, 2008.

[8] Brian Huston. The effects of design pattern application on metric scores. *Journal of Systems and Software*, 58(3):261–269, 2001.

[9] Clemente Izurieta and James M Bieman. A multiple case study of design pattern decay, grime, and rot in evolving software systems. *Software Quality Journal*, 21(2):289–323, 2013.

[10] TH Ng, SC Cheung, WK Chan, and Yuen-Tak Yu. Do maintainers utilize deployed design patterns effectively? In *Proceedings of the 29th international conference on Software Engineering*, pages 168–177. IEEE Computer Society, 2007.

[11] Michael Oduor, Tuomas Alahäivälä, and Harri Oinas-Kukkonen. Persuasive software design patterns for social influence. *Personal and ubiquitous computing*, 18(7):1689–1704, 2014.

[12] Cagri Sahin, Furkan Cayci, Irene Lizeth Manotas Gutiérrez, James Clause, Fouad Kiamilev, Lori Pollock, and Kristina Winbladh. Initial explorations on design pattern energy usage. In *Green and Sustainable Software (GREENS), 2012 First International Workshop on*, pages 55–61. IEEE, 2012.

[13] Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, and Spyros T Halkidis. Design pattern detection using similarity scoring. *Software Engineering, IEEE Transactions on*, 32(11):896–909, 2006.

[14] Marco Zanoni, Francesca Arcelli Fontana, and Fabio Stella. On applying machine learning techniques for design pattern detection. *Journal of Systems and Software*, 103:102–117, 2015.

[15] Cheng Zhang and David Budgen. What do we know about the effectiveness of software design patterns? *Software Engineering, IEEE Transactions on*, 38(5):1213–1231, 2012.

[16] Cheng Zhang and David Budgen. A survey of experienced user perceptions about software design patterns. *Information and Software Technology*, 55(5):822–835, 2013.

[17] Chunying Zhao, Jun Kong, and Kang Zhang. Design pattern evolution and verification using graph transformation. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 290a–290a. IEEE, 2007.

3B Software Testing II

Chairman: 王凡/台灣大學，時間: 15:00-16:30, 7/7，地點: 學思樓學 102

ID	作者	論文名稱
29	Yueh-Ru Lin, Ting-An Yeh and Cheng-Zen Yang	Android 手機 Unity 遊戲監測工具的設計與實作
48	吳尚諭, 林迺衛	基於限制邏輯圖的測試覆蓋標準管理及邊界測試案例產生
54	李培琴, 林迺衛	類別層級單元測試的限制式測試案例產生器
55	張朝翔, 林迺衛	測試物件初始化程序的自動產生
64	唐書麒, 林楚迪	奠基於主題模型之測試個案排序改進方法
61	郭忠義, 蘇翊棠, 賴岱佑	基於雲端分散式環境多用戶火災逃生系統
36	Mao-Jhe Fong, Farn Wang and Yu Ting Chang	Black-box Test Case Generation for Memory Leaks of Android Apps

Paper Title

Author One name, Author Two Name
Department of Your Department
TCSE University
City, Country
Email: {email1, email2}@domain.name

***Abstract*—Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear.**

I. Introduction

In the current trend, design pattern has been widely used for improving software quality. In the academic research, design patterns is one of important research area and applied in different areas [5], [11], [12], many design pattern detection approaches are proposed [6], [13], [14], some work discuss pattern quality metrics [1], [7], [8], design pattern formalization and specification [**?**], [**?**], [**?**], and its benefits [9], [10], [15], [16].

However, using design patterns is not easy, ... Therefore, several approaches are proposed to check the violence of pattern application in a system [2], [17]. We think only static checking to pattern structure is not enough, dynamic testing to test the patterns' semantics is necessary.

In this paper, we propose a test model for design patterns. ....

For example, when we apply *Singleton* pattern, we should conduct the following tests: creating two objects and check if they have the same references. Another example is *Strategy*, when we applied strategy a with this paper, and replace it by another algorithm b, they should have the same behavior, that is, the same output. The following code shows the testing context:

```
1  class  TestStrategyDesignPattern  {
2     void  testSameResult () {
3           Context c = new Context();
4           Strategy a = new StrategyA();
5           c. setStrategy (a);
6           int resultA  = c. doIt ();
7           Strategy b = new StrategyB();
8           c. setStrategy (b);
9           int restultB  = c. doIt ();
10          assertEquals (a, b); // should have same
                result
11    }
12 }
```

A. Paper outline

The remainder of this paper is structured as follows: In section 2, we describe the related work to this research.

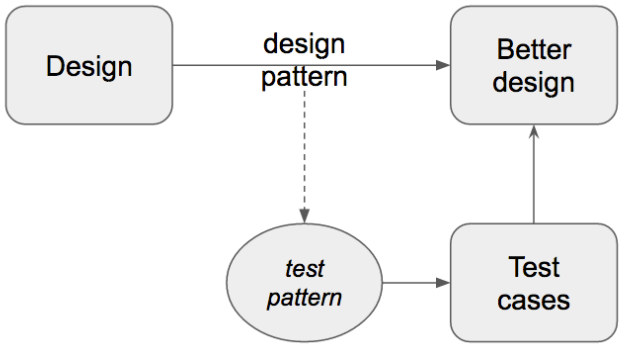


Figure 1. The concept of *test pattern*

Section 3 introduces the details of our approach by two design pattern examples. Section 4 summarizes our approach and future work.

II. Related work

以下針對此研究的相關研究做探討分析：

A. Pattern testing

Chu and Hsueh proposed test refactoring approach in a pattern driven developent [3]. Test-first strategy and code refactoring are the important practices of Extreme Programming for rapid development and quality support. ... However, when the developers perform a pattern-based refactoring to improve the quality, the effort of revising the test cases is much more than that in simple code refactoring (see Fig. 2).

To address this problem, ...

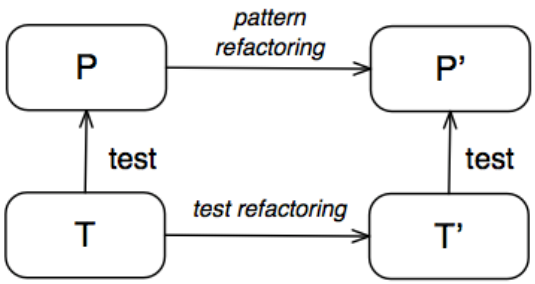


Figure 2. Pattern based test case refactoring

To understand the application context for a pattern in a system, Lin proposed a *Design Pattern Unit Test (DPUT)* approach, which utilizes java Annotation skill to record the pattern utilization and verifies with the expection in the DPUT. This research also design a software framework to help developers design the DPUP in a specification basis. The code is implemented as an Eclipsed plug-in which can automatically transform DPUT into class diagram for better understanding. For the system maintainers they can find out the errors in the earlier phases.

B. Pattern evaluation

設計樣式的效用一直是重要的研究議題。也因此許多學者專家開始投入這方面的研究。Huston [8] 設計了一個實驗以驗證設計樣式是否與軟體衡量相容，亦即，探討應用設計樣式所提昇的軟體品質是否真的可由軟體衡量顯示出來？Huston 的實驗將設計樣式的結構還原成非設計樣式的結構，並以相關的軟體衡量 (software metric) 分別衡量這兩個不同的架構，來比較有無採用設計架構在該軟體衡量上的差異。此實驗顯示設計樣式的品質效能可以透過軟體衡量顯現出來。

III. Our approach

To design the test model for pattern testing, we have to consider the following issues:

- Correctness and Precision. Correctness considers if all faults can be explored by the proposed test cases, and Precision considers if the test can explore the faults in an efficient way. Our test model should satisfy correctness and precision. ...
- Coverage. In general we have many coverage strategies for testing, for example statement coverage or branch coverage. When we test we should enlarge the coverage as we can. Therefore we should care about the normal cases, boundary cases and exception cases. When we utilize this concept to testing a pattern-applied system, what are the boundary cases and exception cases? ...
- Generalization. Gamma et al. classified patterns into creational patterns, structural patterns and behavioral patterns [4]. Patterns in the same group may have same testing issues and methods. We can modularize our testing programs by class generalization. ...

a) Potential Pattern Violation (PPV): To test the pattern-applied application more efficiently, ...

b) Test Pattern for Design Pattern (TP4DP): One of the benefits of design pattern is it is well structured and contains executable code for programmers. Our TP4DP follows the same concept. A test pattern has the following sections: ...

A. Testing "Singleton" pattern

a) Pattern name: Singleton: The intent of the singleton is to ensure a class has only one instance, and provide a global point of access to it.

b) Potential pattern violation: There are two PPV in the Singleton pattern:

- PPV1: When we create two objects from the same class, they refer to different objects.
- PPV2: The original object constructor is not set to be private.

c) Testing guideline: Create two objects of the same Singleton class, check if there are equal by using the operator "==".

d) Demo code: Here is the demo code for the class *Radio*

```
1 class TestRadio {
2     void testRadioSingletonPPE1() {
3         Radio r1 = Radio.instance();
4         Radio r2 = Radio.instance();
5         assertTrue(r1==r2): "Singleton is violated
                ";
6     }
7 }
```

To test the PPV2, we use static testing to check if the constructor is declared as **private**. The example below is the one that doesn't pass the PPV2 checking.

```
1 class Radio {
2     static Radio instance;
3     static Radio getInstance() {
4         if (instance == null)
5             instance = new Radio();
6         }
7         return instance;
8     }
9 }
```

IV. Conclusion

In this paper we propose an idea of pattern test for guiding the programs with patterns applied. We believe a program is error-prone when they are designed in delicate and complex structure. Design patterns are good for flexible design but not easy to understand and apply. For pattern beginners we need a guideline, framework or tool to help them. We demonstrate two examples of *Singleton* and *Observer* to illustrate our ideas. In the future we will explore more design patterns and develop a framework for the pattern test.

References

[1] Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Sofia Charalampidou, and Paris Avgeriou. The effect of gof design patterns on stability: a case study. *IEEE Transactions on Software Engineering*, 41(8):781–802, 2015.

[2] Alex Blewitt, Alan Bundy, and Ian Stark. Automatic verification of java design patterns. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 324–327. IEEE, 2001.

[3] Peng-Hua Chu, Nien-Lin Hsueh, Hong-Hsiang Chen, and Chien-Hung Liu. A test case refactoring approach for pattern-based software development. *Software Quality Journal*, 20(1):43–75, 2012.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[5] Alan R Graves and Chris Czarnecki. Design patterns for behavior-based robotics. *IEEE Transactions on Systems, Man, and cybernetics-part A: systems and humans*, 30(1):36–41, 2000.

[6] Dirk Heuzeroth, Thomas Holl, Gustav Hogstrom, and Welf Lowe. Automatic design pattern detection. In *Program Comprehension, 2003. 11th IEEE International Workshop on*, pages 94–103. IEEE, 2003.

[7] Nien-Lin Hsueh, Peng-Hua Chu, and William Chu. A quantitative approach for evaluating the quality of design patterns. *Journal of Systems and Software*, 81(8):1430–1439, 2008.

[8] Brian Huston. The effects of design pattern application on metric scores. *Journal of Systems and Software*, 58(3):261–269, 2001.

[9] Clemente Izurieta and James M Bieman. A multiple case study of design pattern decay, grime, and rot in evolving software systems. *Software Quality Journal*, 21(2):289–323, 2013.

[10] TH Ng, SC Cheung, WK Chan, and Yuen-Tak Yu. Do maintainers utilize deployed design patterns effectively? In *Proceedings of the 29th international conference on Software Engineering*, pages 168–177. IEEE Computer Society, 2007.

[11] Michael Oduor, Tuomas Alahäivälä, and Harri Oinas-Kukkonen. Persuasive software design patterns for social influence. *Personal and ubiquitous computing*, 18(7):1689–1704, 2014.

[12] Cagri Sahin, Furkan Cayci, Irene Lizeth Manotas Gutiérrez, James Clause, Fouad Kiamilev, Lori Pollock, and Kristina Winbladh. Initial explorations on design pattern energy usage. In *Green and Sustainable Software (GREENS), 2012 First International Workshop on*, pages 55–61. IEEE, 2012.

[13] Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, and Spyros T Halkidis. Design pattern detection using similarity scoring. *Software Engineering, IEEE Transactions on*, 32(11):896–909, 2006.

[14] Marco Zanoni, Francesca Arcelli Fontana, and Fabio Stella. On applying machine learning techniques for design pattern detection. *Journal of Systems and Software*, 103:102–117, 2015.

[15] Cheng Zhang and David Budgen. What do we know about the effectiveness of software design patterns? *Software Engineering, IEEE Transactions on*, 38(5):1213–1231, 2012.

[16] Cheng Zhang and David Budgen. A survey of experienced user perceptions about software design patterns. *Information and Software Technology*, 55(5):822–835, 2013.

[17] Chunying Zhao, Jun Kong, and Kang Zhang. Design pattern evolution and verification using graph transformation. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 290a–290a. IEEE, 2007.

3C Demo paper

Chairman: 李信杰/成功大學，委員：鄭永斌/中央大學，陳英一/台北科技大學，時間: 15:00-16:30, 7/7，地點: 學思樓學 103

ID	作者	論文名稱
11	Wei-Yu Lai and Han-Ming Wu	drEDA：一個基於維度縮減技術的互動式探索性資料分析網頁應用程式
44	薛念林, 梁少榕	應用於資訊教育之可擴充性象棋對奕平台
50	Chu-Yu Wang, Yung-Li Hu, Yao-Tung Tsou, Yennun Huang and Sy-Yen Kuo	A Testing Tool for Mobile Edge Computing Applied on Smart Traffic
62	Chia-Hsu Kuo, Li-Wei Chen, Jing-Shiang Lin and Wen-An Tsai	iScreens – 智慧型多螢幕 eSOP 管理工具
66	何靖霆, 馬尚彬, 戴碩宏	基於流程引擎之對話機器人框架
67	李霽烝, 陳薇涵, 陳錫民	基於共享機制的計程車評價分享平台

Paper Title

Author One name, Author Two Name
Department of Your Department
TCSE University
City, Country
Email: {email1, email2}@domain.name

***Abstract*—Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear.**

I. Introduction

In the current trend, design pattern has been widely used for improving software quality. In the academic research, design patterns is one of important research area and applied in different areas [5], [11], [12], many design pattern detection approaches are proposed [6], [13], [14], some work discuss pattern quality metrics [1], [7], [8], design pattern formalization and specification [**?**], [**?**], [**?**], and its benefits [9], [10], [15], [16].

However, using design patterns is not easy, ... Therefore, several approaches are proposed to check the violence of pattern application in a system [2], [17]. We think only static checking to pattern structure is not enough, dynamic testing to test the patterns' semantics is necessary.

In this paper, we propose a test model for design patterns. ....

For example, when we apply *Singleton* pattern, we should conduct the following tests: creating two objects and check if they have the same references. Another example is *Strategy*, when we applied strategy a with this paper, and replace it by another algorithm b, they should have the same behavior, that is, the same output. The following code shows the testing context:

```
1  class  TestStrategyDesignPattern  {
2     void  testSameResult() {
3           Context c = new Context();
4           Strategy a = new StrategyA();
5           c.setStrategy(a);
6           int resultA  = c.doIt();
7           Strategy b = new StrategyB();
8           c.setStrategy(b);
9           int restultB  = c.doIt();
10          assertEquals(a, b); // should have same
                result
11    }
12 }
```

A. Paper outline

The remainder of this paper is structured as follows: In section 2, we describe the related work to this research.

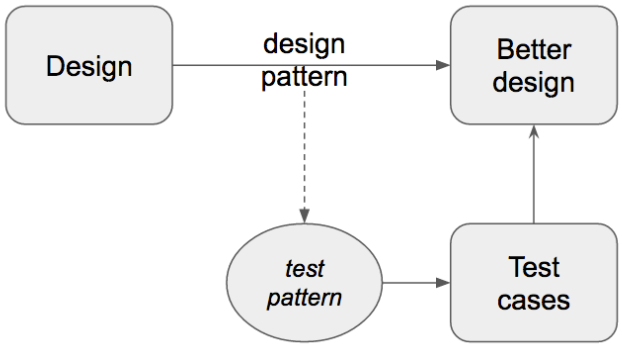


Figure 1. The concept of *test pattern*

Section 3 introduces the details of our approach by two design pattern examples. Section 4 summarizes our approach and future work.

II. Related work

以下針對此研究的相關研究做探討分析：

A. Pattern testing

Chu and Hsueh proposed test refactoring approach in a pattern driven developent [3]. Test-first strategy and code refactoring are the important practices of Extreme Programming for rapid development and quality support. ... However, when the developers perform a pattern-based refactoring to improve the quality, the effort of revising the test cases is much more than that in simple code refactoring (see Fig. 2).

To address this problem, ...

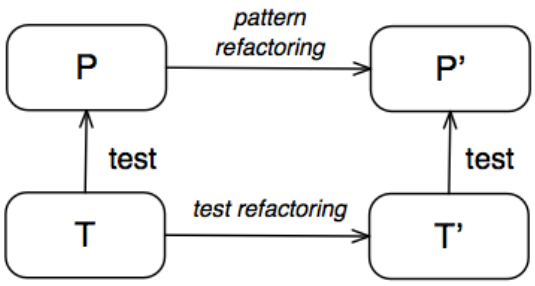


Figure 2. Pattern based test case refactoring

To understand the application context for a pattern in a system, Lin proposed a *Design Pattern Unit Test (DPUT)* approach, which utilizes java Annotation skill to record the pattern utilization and verifies with the expection in the DPUT. This research also design a software framework to help developers design the DPUP in a specification basis. The code is implemented as an Eclipsed plug-in which can automatically transform DPUT into class diagram for better understanding. For the system maintainers they can find out the errors in the earlier phases.

B. Pattern evaluation

設計樣式的效用一直是重要的研究議題。也因此許多學者專家開始投入這方面的研究。Huston [8] 設計了一個實驗以驗證設計樣式是否與軟體衡量相容，亦即，探討應用設計樣式所提昇的軟體品質是否真的可由軟體衡量顯示出來？Huston 的實驗將設計樣式的結構還原成非設計樣式的結構，並以相關的軟體衡量 (software metric) 分別衡量這兩個不同的架構，來比較有無採用設計架構在該軟體衡量上的差異。此實驗顯示設計樣式的品質效能可以透過軟體衡量顯現出來。

III. Our approach

To design the test model for pattern testing, we have to consider the following issues:

- Correctness and Precision. Correctness considers if all faults can be explored by the proposed test cases, and Precision considers if the test can explore the faults in an efficient way. Our test model should satisfy correctness and precision. ...
- Coverage. In general we have many coverage strategies for testing, for example statement coverage or branch coverage. When we test we should enlarge the coverage as we can. Therefore we should care about the normal cases, boundary cases and exception cases. When we utilize this concept to testing a pattern-applied system, what are the boundary cases and exception cases? ...
- Generalization. Gamma et al. classified patterns into creational patterns, structural patterns and behavioral patterns [4]. Patterns in the same group may have same testing issues and methods. We can modularize our testing programs by class generalization. ...

a) Potential Pattern Violation (PPV): To test the pattern-applied application more efficiently, ...

b) Test Pattern for Design Pattern (TP4DP): One of the benefits of design pattern is it is well structured and contains executable code for programmers. Our TP4DP follows the same concept. A test pattern has the following sections: ...

A. Testing "Singleton" pattern

a) Pattern name: Singleton: The intent of the singleton is to ensure a class has only one instance, and provide a global point of access to it.

b) Potential pattern violation: There are two PPV in the Singleton pattern:

- PPV1: When we create two objects from the same class, they refer to different objects.
- PPV2: The original object constructor is not set to be private.

c) Testing guideline: Create two objects of the same Singleton class, check if there are equal by using the operator "==".

d) Demo code: Here is the demo code for the class *Radio*

```
1 class TestRadio {
2     void testRadioSingletonPPE1() {
3         Radio r1 = Radio.instance();
4         Radio r2 = Radio.instance();
5         assertTrue(r1==r2): "Singleton is violated
                ";
6     }
7 }
```

To test the PPV2, we use static testing to check if the constructor is declared as **private**. The example below is the one that doesn't pass the PPV2 checking.

```
1 class Radio {
2     static Radio instance;
3     static Radio getInstance() {
4         if (instance == null)
5             instance = new Radio();
6         }
7         return instance;
8     }
9 }
```

IV. Conclusion

In this paper we propose an idea of pattern test for guiding the programs with patterns applied. We believe a program is error-prone when they are designed in delicate and complex structure. Design patterns are good for flexible design but not easy to understand and apply. For pattern beginners we need a guideline, framework or tool to help them. We demonstrate two examples of *Singleton* and *Observer* to illustrate our ideas. In the future we will explore more design patterns and develop a framework for the pattern test.

References

[1] Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Sofia Charalampidou, and Paris Avgeriou. The effect of gof design patterns on stability: a case study. *IEEE Transactions on Software Engineering*, 41(8):781–802, 2015.

[2] Alex Blewitt, Alan Bundy, and Ian Stark. Automatic verification of java design patterns. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 324–327. IEEE, 2001.

[3] Peng-Hua Chu, Nien-Lin Hsueh, Hong-Hsiang Chen, and Chien-Hung Liu. A test case refactoring approach for pattern-based software development. *Software Quality Journal*, 20(1):43–75, 2012.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[5] Alan R Graves and Chris Czarnecki. Design patterns for behavior-based robotics. *IEEE Transactions on Systems, Man, and cybernetics-part A: systems and humans*, 30(1):36–41, 2000.

[6] Dirk Heuzeroth, Thomas Holl, Gustav Hogstrom, and Welf Lowe. Automatic design pattern detection. In *Program Comprehension, 2003. 11th IEEE International Workshop on*, pages 94–103. IEEE, 2003.

[7] Nien-Lin Hsueh, Peng-Hua Chu, and William Chu. A quantitative approach for evaluating the quality of design patterns. *Journal of Systems and Software*, 81(8):1430–1439, 2008.

[8] Brian Huston. The effects of design pattern application on metric scores. *Journal of Systems and Software*, 58(3):261–269, 2001.

[9] Clemente Izurieta and James M Bieman. A multiple case study of design pattern decay, grime, and rot in evolving software systems. *Software Quality Journal*, 21(2):289–323, 2013.

[10] TH Ng, SC Cheung, WK Chan, and Yuen-Tak Yu. Do maintainers utilize deployed design patterns effectively? In *Proceedings of the 29th international conference on Software Engineering*, pages 168–177. IEEE Computer Society, 2007.

[11] Michael Oduor, Tuomas Alahäivälä, and Harri Oinas-Kukkonen. Persuasive software design patterns for social influence. *Personal and ubiquitous computing*, 18(7):1689–1704, 2014.

[12] Cagri Sahin, Furkan Cayci, Irene Lizeth Manotas Gutiérrez, James Clause, Fouad Kiamilev, Lori Pollock, and Kristina Winbladh. Initial explorations on design pattern energy usage. In *Green and Sustainable Software (GREENS), 2012 First International Workshop on*, pages 55–61. IEEE, 2012.

[13] Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, and Spyros T Halkidis. Design pattern detection using similarity scoring. *Software Engineering, IEEE Transactions on*, 32(11):896–909, 2006.

[14] Marco Zanoni, Francesca Arcelli Fontana, and Fabio Stella. On applying machine learning techniques for design pattern detection. *Journal of Systems and Software*, 103:102–117, 2015.

[15] Cheng Zhang and David Budgen. What do we know about the effectiveness of software design patterns? *Software Engineering, IEEE Transactions on*, 38(5):1213–1231, 2012.

[16] Cheng Zhang and David Budgen. A survey of experienced user perceptions about software design patterns. *Information and Software Technology*, 55(5):822–835, 2013.

[17] Chunying Zhao, Jun Kong, and Kang Zhang. Design pattern evolution and verification using graph transformation. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 290a–290a. IEEE, 2007.

Section 4

Software Engineering II, Software Application, and Industry Talk II

時間: 10:40-12:00, 7/8

4A Data Engineering

Chairman: 何承遠/亞洲大學，時間: 10:40-12:00, 7/8，地點: 學思樓學 104

ID	作者	論文名稱
42	Chang You-Wei and Chihhsiong Shih	以最小偏差基因及粒子群演算法分析缺血性中風轉出血性中風成因探討
46	葉錦文, 葉明憲, 葉家舟, 邱宏彬, 吳梅君, 林迺衛	基於癌症登記資料庫的中西醫合併治療的肺癌存活分析
79	侯修平, 許懷中, 吳榮彬, 楊東麟	根據 Open edX 的課程設計的競賽式學習平台架構
80	邱毓宸, 許懷中, 吳榮彬, 楊東麟	使用線上學習行為紀錄預測學生的學習成效
76	Ming-Chi Liu, Chen-Hsiang Yu, Jungpin Wu and An-Chi Liu	System log analysis for understanding user engagement: The case of MOOCs
35	Cheng-Yuan Ho	電子票證大數據應用於台中市公車旅客型態之研究

Paper Title

Author One name, Author Two Name
Department of Your Department
TCSE University
City, Country
Email: {email1, email2}@domain.name

***Abstract*—Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear.**

I. Introduction

In the current trend, design pattern has been widely used for improving software quality. In the academic research, design patterns is one of important research area and applied in different areas [5], [11], [12], many design pattern detection approaches are proposed [6], [13], [14], some work discuss pattern quality metrics [1], [7], [8], design pattern formalization and specification [**?**], [**?**], [**?**], and its benefits [9], [10], [15], [16].

However, using design patterns is not easy, ... Therefore, several approaches are proposed to check the violence of pattern application in a system [2], [17]. We think only static checking to pattern structure is not enough, dynamic testing to test the patterns' semantics is necessary.

In this paper, we propose a test model for design patterns. ....

For example, when we apply *Singleton* pattern, we should conduct the following tests: creating two objects and check if they have the same references. Another example is *Strategy*, when we applied strategy a with this paper, and replace it by another algorithm b, they should have the same behavior, that is, the same output. The following code shows the testing context:

```
1  class TestStrategyDesignPattern {
2      void testSameResult() {
3          Context c = new Context();
4          Strategy a = new StrategyA();
5          c.setStrategy(a);
6          int resultA = c.doIt();
7          Strategy b = new StrategyB();
8          c.setStrategy(b);
9          int restultB = c.doIt();
10         assertEquals(a, b); // should have same
               result
11     }
12 }
```

A. Paper outline

The remainder of this paper is structured as follows: In section 2, we describe the related work to this research.

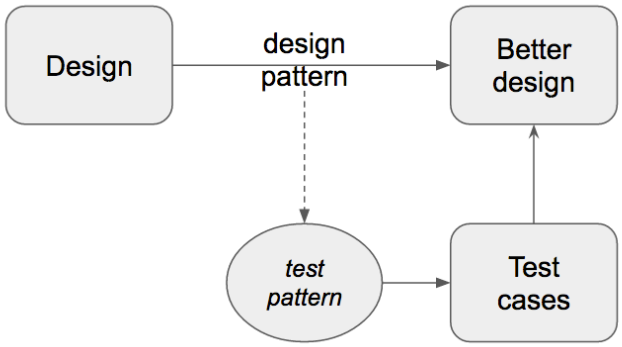


Figure 1. The concept of *test pattern*

Section 3 introduces the details of our approach by two design pattern examples. Section 4 summarizes our approach and future work.

II. Related work

以下針對此研究的相關研究做探討分析：

A. Pattern testing

Chu and Hsueh proposed test refactoring approach in a pattern driven developent [3]. Test-first strategy and code refactoring are the important practices of Extreme Programming for rapid development and quality support. ... However, when the developers perform a pattern-based refactoring to improve the quality, the effort of revising the test cases is much more than that in simple code refactoring (see Fig. 2).

To address this problem, ...

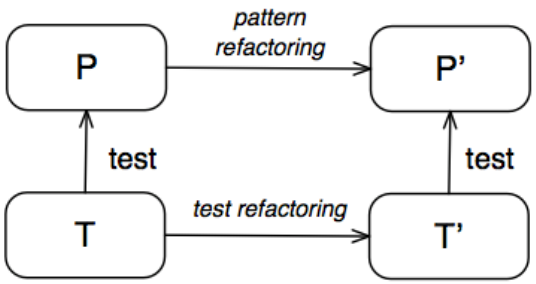


Figure 2. Pattern based test case refactoring

To understand the application context for a pattern in a system, Lin proposed a *Design Pattern Unit Test (DPUT)* approach, which utilizes java Annotation skill to record the pattern utilization and verifies with the expection in the DPUT. This research also design a software framework to help developers design the DPUP in a specification basis. The code is implemented as an Eclipsed plug-in which can automatically transform DPUT into class diagram for better understanding. For the system maintainers they can find out the errors in the earlier phases.

B. Pattern evaluation

設計樣式的效用一直是重要的研究議題。也因此許多學者專家開始投入這方面的研究。Huston [8] 設計了一個實驗以驗證設計樣式是否與軟體衡量相容，亦即，探討應用設計樣式所提昇的軟體品質是否真的可由軟體衡量顯示出來？Huston 的實驗將設計樣式的結構還原成非設計樣式的結構，並以相關的軟體衡量 (software metric) 分別衡量這兩個不同的架構，來比較有無採用設計架構在該軟體衡量上的差異。此實驗顯示設計樣式的品質效能可以透過軟體衡量顯現出來。

III. Our approach

To design the test model for pattern testing, we have to consider the following issues:

- Correctness and Precision. Correctness considers if all faults can be explored by the proposed test cases, and Precision considers if the test can explore the faults in an efficient way. Our test model should satisfy correctness and precision. ...
- Coverage. In general we have many coverage strategies for testing, for example statement coverage or branch coverage. When we test we should enlarge the coverage as we can. Therefore we should care about the normal cases, boundary cases and exception cases. When we utilize this concept to testing a pattern-applied system, what are the boundary cases and exception cases? ...
- Generalization. Gamma et al. classified patterns into creational patterns, structural patterns and behavioral patterns [4]. Patterns in the same group may have same testing issues and methods. We can modularize our testing programs by class generalization. ...

a) Potential Pattern Violation (PPV): To test the pattern-applied application more efficiently, ...

b) Test Pattern for Design Pattern (TP4DP): One of the benefits of design pattern is it is well structured and contains executable code for programmers. Our TP4DP follows the same concept. A test pattern has the following sections: ...

A. Testing "Singleton" pattern

a) Pattern name: Singleton: The intent of the singleton is to ensure a class has only one instance, and provide a global point of access to it.

b) Potential pattern violation: There are two PPV in the Singleton pattern:

- PPV1: When we create two objects from the same class, they refer to different objects.
- PPV2: The original object constructor is not set to be private.

c) Testing guideline: Create two objects of the same Singleton class, check if there are equal by using the operator "==".

d) Demo code: Here is the demo code for the class *Radio*

```
1 class TestRadio {
2     void testRadioSingletonPPE1() {
3         Radio r1 = Radio.instance();
4         Radio r2 = Radio.instance();
5         assertTrue(r1==r2): "Singleton is violated
                ";
6     }
7 }
```

To test the PPV2, we use static testing to check if the constructor is declared as **private**. The example below is the one that doesn't pass the PPV2 checking.

```
1 class Radio {
2     static Radio instance;
3     static Radio getInstance() {
4         if (instance == null)
5             instance = new Radio();
6         }
7         return instance;
8     }
9 }
```

IV. Conclusion

In this paper we propose an idea of pattern test for guiding the programs with patterns applied. We believe a program is error-prone when they are designed in delicate and complex structure. Design patterns are good for flexible design but not easy to understand and apply. For pattern beginners we need a guideline, framework or tool to help them. We demonstrate two examples of *Singleton* and *Observer* to illustrate our ideas. In the future we will explore more design patterns and develop a framework for the pattern test.

References

[1] Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Sofia Charalampidou, and Paris Avgeriou. The effect of gof design patterns on stability: a case study. *IEEE Transactions on Software Engineering*, 41(8):781–802, 2015.

[2] Alex Blewitt, Alan Bundy, and Ian Stark. Automatic verification of java design patterns. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 324–327. IEEE, 2001.

[3] Peng-Hua Chu, Nien-Lin Hsueh, Hong-Hsiang Chen, and Chien-Hung Liu. A test case refactoring approach for pattern-based software development. *Software Quality Journal*, 20(1):43–75, 2012.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[5] Alan R Graves and Chris Czarnecki. Design patterns for behavior-based robotics. *IEEE Transactions on Systems, Man, and cybernetics-part A: systems and humans*, 30(1):36–41, 2000.

[6] Dirk Heuzeroth, Thomas Holl, Gustav Hogstrom, and Welf Lowe. Automatic design pattern detection. In *Program Comprehension, 2003. 11th IEEE International Workshop on*, pages 94–103. IEEE, 2003.

[7] Nien-Lin Hsueh, Peng-Hua Chu, and William Chu. A quantitative approach for evaluating the quality of design patterns. *Journal of Systems and Software*, 81(8):1430–1439, 2008.

[8] Brian Huston. The effects of design pattern application on metric scores. *Journal of Systems and Software*, 58(3):261–269, 2001.

[9] Clemente Izurieta and James M Bieman. A multiple case study of design pattern decay, grime, and rot in evolving software systems. *Software Quality Journal*, 21(2):289–323, 2013.

[10] TH Ng, SC Cheung, WK Chan, and Yuen-Tak Yu. Do maintainers utilize deployed design patterns effectively? In *Proceedings of the 29th international conference on Software Engineering*, pages 168–177. IEEE Computer Society, 2007.

[11] Michael Oduor, Tuomas Alahäivälä, and Harri Oinas-Kukkonen. Persuasive software design patterns for social influence. *Personal and ubiquitous computing*, 18(7):1689–1704, 2014.

[12] Cagri Sahin, Furkan Cayci, Irene Lizeth Manotas Gutiérrez, James Clause, Fouad Kiamilev, Lori Pollock, and Kristina Winbladh. Initial explorations on design pattern energy usage. In *Green and Sustainable Software (GREENS), 2012 First International Workshop on*, pages 55–61. IEEE, 2012.

[13] Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, and Spyros T Halkidis. Design pattern detection using similarity scoring. *Software Engineering, IEEE Transactions on*, 32(11):896–909, 2006.

[14] Marco Zanoni, Francesca Arcelli Fontana, and Fabio Stella. On applying machine learning techniques for design pattern detection. *Journal of Systems and Software*, 103:102–117, 2015.

[15] Cheng Zhang and David Budgen. What do we know about the effectiveness of software design patterns? *Software Engineering, IEEE Transactions on*, 38(5):1213–1231, 2012.

[16] Cheng Zhang and David Budgen. A survey of experienced user perceptions about software design patterns. *Information and Software Technology*, 55(5):822–835, 2013.

[17] Chunying Zhao, Jun Kong, and Kang Zhang. Design pattern evolution and verification using graph transformation. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 290a–290a. IEEE, 2007.

4B Software Engineering II

Chairman: 李文廷/高雄師範大學，時間: 10:40-12:00, 7/8，地點: 學思樓學 102

ID	作者	論文名稱
72	黃俊詠, 陳錫民, 陳奕中	以虛擬容器為基底的 IoT 服務管理機制
10	Feng-Shou Yu and Wei-Ling Chen	一個新的系統耦合度度量 (A New System Coupling Metric)
12	Chang-Yen Tsai, Kuo-Hsun Hsu and Chun-Han Lin	Identifying Aspects from Cross-Version Revision History in Software Development
18	薛念林, 廖健宏	以 3D 視覺化技術為基礎之開源軟體品質分析
23	劉建宏, 陳偉凱, 陳炳宏, 楊凱霖	支援程式作業壞味道偵測與批改之工具
73	李文廷, 許博淳	應用設計結構矩陣分析軟體設計氣味

Paper Title

Author One name, Author Two Name
Department of Your Department
TCSE University
City, Country
Email: {email1, email2}@domain.name

***Abstract*—Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear.**

I. Introduction

In the current trend, design pattern has been widely used for improving software quality. In the academic research, design patterns is one of important research area and applied in different areas [5], [11], [12], many design pattern detection approaches are proposed [6], [13], [14], some work discuss pattern quality metrics [1], [7], [8], design pattern formalization and specification [**?**], [**?**], [**?**], and its benefits [9], [10], [15], [16].

However, using design patterns is not easy, ... Therefore, several approaches are proposed to check the violence of pattern application in a system [2], [17]. We think only static checking to pattern structure is not enough, dynamic testing to test the patterns' semantics is necessary.

In this paper, we propose a test model for design patterns. ....

For example, when we apply *Singleton* pattern, we should conduct the following tests: creating two objects and check if they have the same references. Another example is *Strategy*, when we applied strategy a with this paper, and replace it by another algorithm b, they should have the same behavior, that is, the same output. The following code shows the testing context:

```
1  class TestStrategyDesignPattern {
2      void testSameResult() {
3          Context c = new Context();
4          Strategy a = new StrategyA();
5          c.setStrategy(a);
6          int resultA = c.doIt();
7          Strategy b = new StrategyB();
8          c.setStrategy(b);
9          int restultB = c.doIt();
10         assertEquals(a, b); // should have same
               result
11     }
12 }
```

A. Paper outline

The remainder of this paper is structured as follows: In section 2, we describe the related work to this research.

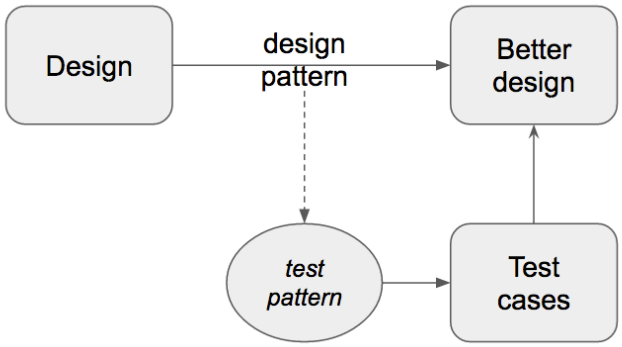


Figure 1. The concept of *test pattern*

Section 3 introduces the details of our approach by two design pattern examples. Section 4 summarizes our approach and future work.

II. Related work

以下針對此研究的相關研究做探討分析：

A. Pattern testing

Chu and Hsueh proposed test refactoring approach in a pattern driven developent [3]. Test-first strategy and code refactoring are the important practices of Extreme Programming for rapid development and quality support. ... However, when the developers perform a pattern-based refactoring to improve the quality, the effort of revising the test cases is much more than that in simple code refactoring (see Fig. 2).

To address this problem, ...

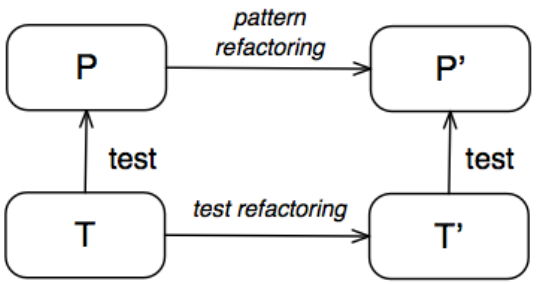


Figure 2. Pattern based test case refactoring

To understand the application context for a pattern in a system, Lin proposed a *Design Pattern Unit Test (DPUT)* approach, which utilizes java Annotation skill to record the pattern utilization and verifies with the expection in the DPUT. This research also design a software framework to help developers design the DPUP in a specification basis. The code is implemented as an Eclipsed plug-in which can automatically transform DPUT into class diagram for better understanding. For the system maintainers they can find out the errors in the earlier phases.

B. Pattern evaluation

設計樣式的效用一直是重要的研究議題。也因此許多學者專家開始投入這方面的研究。Huston [8] 設計了一個實驗以驗證設計樣式是否與軟體衡量相容，亦即，探討應用設計樣式所提昇的軟體品質是否真的可由軟體衡量顯示出來？Huston 的實驗將設計樣式的結構還原成非設計樣式的結構，並以相關的軟體衡量 (software metric) 分別衡量這兩個不同的架構，來比較有無採用設計架構在該軟體衡量上的差異。此實驗顯示設計樣式的品質效能可以透過軟體衡量顯現出來。

III. Our approach

To design the test model for pattern testing, we have to consider the following issues:

- Correctness and Precision. Correctness considers if all faults can be explored by the proposed test cases, and Precision considers if the test can explore the faults in an efficient way. Our test model should satisfy correctness and precision. ...
- Coverage. In general we have many coverage strategies for testing, for example statement coverage or branch coverage. When we test we should enlarge the coverage as we can. Therefore we should care about the normal cases, boundary cases and exception cases. When we utilize this concept to testing a pattern-applied system, what are the boundary cases and exception cases? ...
- Generalization. Gamma et al. classified patterns into creational patterns, structural patterns and behavioral patterns [4]. Patterns in the same group may have same testing issues and methods. We can modularize our testing programs by class generalization. ...

a) Potential Pattern Violation (PPV): To test the pattern-applied application more efficiently, ...

b) Test Pattern for Design Pattern (TP4DP): One of the benefits of design pattern is it is well structured and contains executable code for programmers. Our TP4DP follows the same concept. A test pattern has the following sections: ...

*A. Testing "*Singleton*" pattern*

a) Pattern name: Singleton: The intent of the singleton is to ensure a class has only one instance, and provide a global point of access to it.

b) Potential pattern violation: There are two PPV in the Singleton pattern:

- PPV1: When we create two objects from the same class, they refer to different objects.
- PPV2: The original object constructor is not set to be private.

c) Testing guideline: Create two objects of the same Singleton class, check if there are equal by using the operator "==".

d) Demo code: Here is the demo code for the class *Radio*

```
1 class TestRadio {
2     void testRadioSingletonPPE1() {
3         Radio r1 = Radio.instance();
4         Radio r2 = Radio.instance();
5         assertTrue(r1==r2): "Singleton is violated
                ";
6     }
7 }
```

To test the PPV2, we use static testing to check if the constructor is declared as **private**. The example below is the one that doesn't pass the PPV2 checking.

```
1 class Radio {
2     static Radio instance;
3     static Radio getInstance() {
4         if (instance == null)
5             instance = new Radio();
6         }
7         return instance;
8     }
9 }
```

IV. Conclusion

In this paper we propose an idea of pattern test for guiding the programs with patterns applied. We believe a program is error-prone when they are designed in delicate and complex structure. Design patterns are good for flexible design but not easy to understand and apply. For pattern beginners we need a guideline, framework or tool to help them. We demonstrate two examples of *Singleton* and *Observer* to illustrate our ideas. In the future we will explore more design patterns and develop a framework for the pattern test.

References

[1] Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Sofia Charalampidou, and Paris Avgeriou. The effect of gof design patterns on stability: a case study. *IEEE Transactions on Software Engineering*, 41(8):781–802, 2015.

[2] Alex Blewitt, Alan Bundy, and Ian Stark. Automatic verification of java design patterns. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 324–327. IEEE, 2001.

[3] Peng-Hua Chu, Nien-Lin Hsueh, Hong-Hsiang Chen, and Chien-Hung Liu. A test case refactoring approach for pattern-based software development. *Software Quality Journal*, 20(1):43–75, 2012.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[5] Alan R Graves and Chris Czarnecki. Design patterns for behavior-based robotics. *IEEE Transactions on Systems, Man, and cybernetics-part A: systems and humans*, 30(1):36–41, 2000.

[6] Dirk Heuzeroth, Thomas Holl, Gustav Hogstrom, and Welf Lowe. Automatic design pattern detection. In *Program Comprehension, 2003. 11th IEEE International Workshop on*, pages 94–103. IEEE, 2003.

[7] Nien-Lin Hsueh, Peng-Hua Chu, and William Chu. A quantitative approach for evaluating the quality of design patterns. *Journal of Systems and Software*, 81(8):1430–1439, 2008.

[8] Brian Huston. The effects of design pattern application on metric scores. *Journal of Systems and Software*, 58(3):261–269, 2001.

[9] Clemente Izurieta and James M Bieman. A multiple case study of design pattern decay, grime, and rot in evolving software systems. *Software Quality Journal*, 21(2):289–323, 2013.

[10] TH Ng, SC Cheung, WK Chan, and Yuen-Tak Yu. Do maintainers utilize deployed design patterns effectively? In *Proceedings of the 29th international conference on Software Engineering*, pages 168–177. IEEE Computer Society, 2007.

[11] Michael Oduor, Tuomas Alahäivälä, and Harri Oinas-Kukkonen. Persuasive software design patterns for social influence. *Personal and ubiquitous computing*, 18(7):1689–1704, 2014.

[12] Cagri Sahin, Furkan Cayci, Irene Lizeth Manotas Gutiérrez, James Clause, Fouad Kiamilev, Lori Pollock, and Kristina Winbladh. Initial explorations on design pattern energy usage. In *Green and Sustainable Software (GREENS), 2012 First International Workshop on*, pages 55–61. IEEE, 2012.

[13] Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, and Spyros T Halkidis. Design pattern detection using similarity scoring. *Software Engineering, IEEE Transactions on*, 32(11):896–909, 2006.

[14] Marco Zanoni, Francesca Arcelli Fontana, and Fabio Stella. On applying machine learning techniques for design pattern detection. *Journal of Systems and Software*, 103:102–117, 2015.

[15] Cheng Zhang and David Budgen. What do we know about the effectiveness of software design patterns? *Software Engineering, IEEE Transactions on*, 38(5):1213–1231, 2012.

[16] Cheng Zhang and David Budgen. A survey of experienced user perceptions about software design patterns. *Information and Software Technology*, 55(5):822–835, 2013.

[17] Chunying Zhao, Jun Kong, and Kang Zhang. Design pattern evolution and verification using graph transformation. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 290a–290a. IEEE, 2007.

4C Application

Chairman: 徐國勛/台中教育大學，時間: 10:40-12:00, 7/8，地點: 學思樓學 103

ID	作者	論文名稱
13	林桂任, 張君丞, 陳世軒, 陳克勤 許乙清	整合健康存摺與開放資料於物聯網架構之用藥安全系統
19	Jia-Ching Jian and Yung-Ping Cheng	Very High Precision Optical Character Recognition for Clean-Fixed-Sized True Type Character
22	楊博善, 黃俊堯, 高國峰, 廖宜恩	從電視節目到鏈結開放資料的轉換系統設計與開發
24	陳湘諭, 鄭為民	改良之專利資料庫檢索系統
26	Su Yi-Shiang, 蔡佳翰, 謝孟諺	結合擴增實境與 OpenCV 於服飾業之研究

Paper Title

Author One name, Author Two Name
Department of Your Department
TCSE University
City, Country
Email: {email1, email2}@domain.name

***Abstract*—Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear.**

I. Introduction

In the current trend, design pattern has been widely used for improving software quality. In the academic research, design patterns is one of important research area and applied in different areas [5], [11], [12], many design pattern detection approaches are proposed [6], [13], [14], some work discuss pattern quality metrics [1], [7], [8], design pattern formalization and specification [**?**], [**?**], [**?**], and its benefits [9], [10], [15], [16].

However, using design patterns is not easy, ... Therefore, several approaches are proposed to check the violence of pattern application in a system [2], [17]. We think only static checking to pattern structure is not enough, dynamic testing to test the patterns' semantics is necessary.

In this paper, we propose a test model for design patterns. ....

For example, when we apply *Singleton* pattern, we should conduct the following tests: creating two objects and check if they have the same references. Another example is *Strategy*, when we applied strategy a with this paper, and replace it by another algorithm b, they should have the same behavior, that is, the same output. The following code shows the testing context:

```
1  class  TestStrategyDesignPattern  {
2     void  testSameResult() {
3           Context c = new Context();
4           Strategy a = new StrategyA();
5           c.setStrategy(a);
6           int resultA  = c.doIt();
7           Strategy b = new StrategyB();
8           c.setStrategy(b);
9           int restultB  = c.doIt();
10          assertEquals(a, b); // should have same
                result
11    }
12 }
```

A. Paper outline

The remainder of this paper is structured as follows: In section 2, we describe the related work to this research.

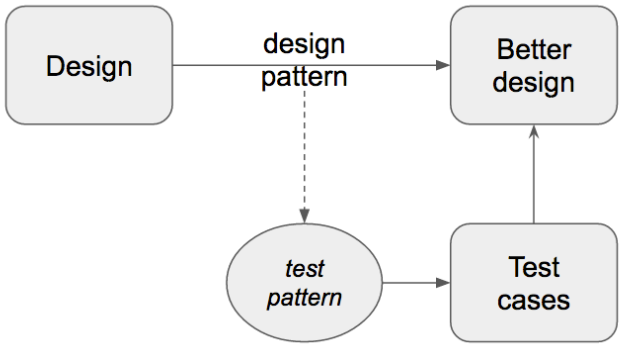


Figure 1. The concept of *test pattern*

Section 3 introduces the details of our approach by two design pattern examples. Section 4 summarizes our approach and future work.

II. Related work

以下針對此研究的相關研究做探討分析：

A. Pattern testing

Chu and Hsueh proposed test refactoring approach in a pattern driven developent [3]. Test-first strategy and code refactoring are the important practices of Extreme Programming for rapid development and quality support. ... However, when the developers perform a pattern-based refactoring to improve the quality, the effort of revising the test cases is much more than that in simple code refactoring (see Fig. 2).

To address this problem, ...

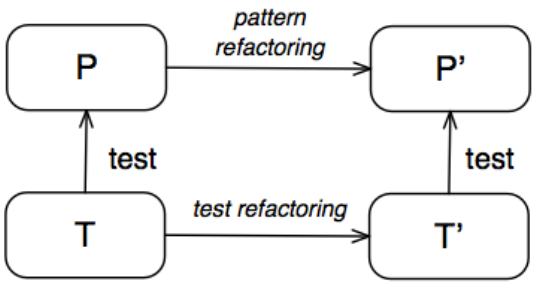


Figure 2. Pattern based test case refactoring

To understand the application context for a pattern in a system, Lin proposed a *Design Pattern Unit Test (DPUT)* approach, which utilizes java Annotation skill to record the pattern utilization and verifies with the expection in the DPUT. This research also design a software framework to help developers design the DPUP in a specification basis. The code is implemented as an Eclipsed plug-in which can automatically transform DPUT into class diagram for better understanding. For the system maintainers they can find out the errors in the earlier phases.

B. Pattern evaluation

設計樣式的效用一直是重要的研究議題。也因此許多學者專家開始投入這方面的研究。Huston [8] 設計了一個實驗以驗證設計樣式是否與軟體衡量相容，亦即，探討應用設計樣式所提昇的軟體品質是否真的可由軟體衡量顯示出來？Huston 的實驗將設計樣式的結構還原成非設計樣式的結構，並以相關的軟體衡量 (software metric) 分別衡量這兩個不同的架構，來比較有無採用設計架構在該軟體衡量上的差異。此實驗顯示設計樣式的品質效能可以透過軟體衡量顯現出來。

III. Our approach

To design the test model for pattern testing, we have to consider the following issues:

- Correctness and Precision. Correctness considers if all faults can be explored by the proposed test cases, and Precision considers if the test can explore the faults in an efficient way. Our test model should satisfy correctness and precision. ...
- Coverage. In general we have many coverage strategies for testing, for example statement coverage or branch coverage. When we test we should enlarge the coverage as we can. Therefore we should care about the normal cases, boundary cases and exception cases. When we utilize this concept to testing a pattern-applied system, what are the boundary cases and exception cases? ...
- Generalization. Gamma et al. classified patterns into creational patterns, structural patterns and behavioral patterns [4]. Patterns in the same group may have same testing issues and methods. We can modularize our testing programs by class generalization. ...

a) Potential Pattern Violation (PPV): To test the pattern-applied application more efficiently, ...

b) Test Pattern for Design Pattern (TP4DP): One of the benefits of design pattern is it is well structured and contains executable code for programmers. Our TP4DP follows the same concept. A test pattern has the following sections: ...

A. Testing "Singleton" pattern

a) Pattern name: Singleton: The intent of the singleton is to ensure a class has only one instance, and provide a global point of access to it.

b) Potential pattern violation: There are two PPV in the Singleton pattern:

- PPV1: When we create two objects from the same class, they refer to different objects.
- PPV2: The original object constructor is not set to be private.

c) Testing guideline: Create two objects of the same Singleton class, check if there are equal by using the operator "==".

d) Demo code: Here is the demo code for the class *Radio*

```
1 class TestRadio {
2     void testRadioSingletonPPE1() {
3         Radio r1 = Radio.instance();
4         Radio r2 = Radio.instance();
5         assertTrue(r1==r2): "Singleton is violated
                ";
6     }
7 }
```

To test the PPV2, we use static testing to check if the constructor is declared as **private**. The example below is the one that doesn't pass the PPV2 checking.

```
1 class Radio {
2     static Radio instance;
3     static Radio getInstance() {
4         if (instance == null)
5             instance = new Radio();
6         }
7         return instance;
8     }
9 }
```

IV. Conclusion

In this paper we propose an idea of pattern test for guiding the programs with patterns applied. We believe a program is error-prone when they are designed in delicate and complex structure. Design patterns are good for flexible design but not easy to understand and apply. For pattern beginners we need a guideline, framework or tool to help them. We demonstrate two examples of *Singleton* and *Observer* to illustrate our ideas. In the future we will explore more design patterns and develop a framework for the pattern test.

References

[1] Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Sofia Charalampidou, and Paris Avgeriou. The effect of gof design patterns on stability: a case study. *IEEE Transactions on Software Engineering*, 41(8):781–802, 2015.

[2] Alex Blewitt, Alan Bundy, and Ian Stark. Automatic verification of java design patterns. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 324–327. IEEE, 2001.

[3] Peng-Hua Chu, Nien-Lin Hsueh, Hong-Hsiang Chen, and Chien-Hung Liu. A test case refactoring approach for pattern-based software development. *Software Quality Journal*, 20(1):43–75, 2012.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[5] Alan R Graves and Chris Czarnecki. Design patterns for behavior-based robotics. *IEEE Transactions on Systems, Man, and cybernetics-part A: systems and humans*, 30(1):36–41, 2000.

[6] Dirk Heuzeroth, Thomas Holl, Gustav Hogstrom, and Welf Lowe. Automatic design pattern detection. In *Program Comprehension, 2003. 11th IEEE International Workshop on*, pages 94–103. IEEE, 2003.

[7] Nien-Lin Hsueh, Peng-Hua Chu, and William Chu. A quantitative approach for evaluating the quality of design patterns. *Journal of Systems and Software*, 81(8):1430–1439, 2008.

[8] Brian Huston. The effects of design pattern application on metric scores. *Journal of Systems and Software*, 58(3):261–269, 2001.

[9] Clemente Izurieta and James M Bieman. A multiple case study of design pattern decay, grime, and rot in evolving software systems. *Software Quality Journal*, 21(2):289–323, 2013.

[10] TH Ng, SC Cheung, WK Chan, and Yuen-Tak Yu. Do maintainers utilize deployed design patterns effectively? In *Proceedings of the 29th international conference on Software Engineering*, pages 168–177. IEEE Computer Society, 2007.

[11] Michael Oduor, Tuomas Alahäivälä, and Harri Oinas-Kukkonen. Persuasive software design patterns for social influence. *Personal and ubiquitous computing*, 18(7):1689–1704, 2014.

[12] Cagri Sahin, Furkan Cayci, Irene Lizeth Manotas Gutiérrez, James Clause, Fouad Kiamilev, Lori Pollock, and Kristina Winbladh. Initial explorations on design pattern energy usage. In *Green and Sustainable Software (GREENS), 2012 First International Workshop on*, pages 55–61. IEEE, 2012.

[13] Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, and Spyros T Halkidis. Design pattern detection using similarity scoring. *Software Engineering, IEEE Transactions on*, 32(11):896–909, 2006.

[14] Marco Zanoni, Francesca Arcelli Fontana, and Fabio Stella. On applying machine learning techniques for design pattern detection. *Journal of Systems and Software*, 103:102–117, 2015.

[15] Cheng Zhang and David Budgen. What do we know about the effectiveness of software design patterns? *Software Engineering, IEEE Transactions on*, 38(5):1213–1231, 2012.

[16] Cheng Zhang and David Budgen. A survey of experienced user perceptions about software design patterns. *Information and Software Technology*, 55(5):822–835, 2013.

[17] Chunying Zhao, Jun Kong, and Kang Zhang. Design pattern evolution and verification using graph transformation. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 290a–290a. IEEE, 2007.

4D Industry talk II

Chairman: 許懷中/逢甲大學，時間: 10:40-12:00, 7/8，地點：學思樓第九國際會議廳

ID	作者	論文名稱
	王燄 / MathWorks 財務工程技術經理	資料科學在物聯網之應用

地點：逢甲大學 學思樓

	7 月 7 日			
8:30	報到			
9:20	大會開幕: 第九國際會議廳			
9:30	大會演講 I：*Big Data Analytics: A Requirements Engineering Perspective* [m] Prof. Lawrence Chung			
10:20	Coffee & Tea Break			
10:40	1A: SE I [a]	1B: Test I [b]	1C: Web [c]	1D: AI [d]
12:00	Lunch & 軟工學會會員大會			
13:20 14:50	2A: Best English [a]	2B: English [b]	2C: IoT [c]	2D: Industry talk I [m]
15:00	3A: Best regular [a]	3B: Test II [b]	3C: Demo [c]	
16:30	Coffee & Tea Break			
16:50 17:50	專題論壇：*Software Engineering and Data Science* [m] 鄭有進教授, 李允中教授, Prof. Lawrence Chung, 韋建名博士			
18:10	晚宴：星饗道餐廳			
	7 月 8 日			
9:00	報到			
9:30	大會演講 II：*從大數據到人工智慧* [m] 陳昇瑋 博士			
10:20	Coffee & Tea Break			
10:40	4A: Data [d]	4B: SE II [b]	4C: Application [c]	4D: Industry talk II [m]
12:00 12:30	TCSE Hacks 頒獎 [m]			

Room: [m] 第九國際會議廳, [a] 學 101, [b] 學 102, [c] 學 103, [d] 學 104。