陳仲儼、徐子涵（2019），『建立一個在軟體反向工程中針對 UML 結構圖形之自動化本體品質檢測系統：以耦合度為例』，*中華民國資訊管理學報*，第二十六卷，第四期，頁 379-412。

建立一個在軟體反向工程中針對 UML 結構圖形之自動化本體品質檢測系統：以耦合度為例

陳仲儼*
國立中央大學資訊管理學系

徐子涵
和碩聯合科技股份有限公司

摘要

軟體反向工程在現今講求快速實作與需求變化的軟體發展情境下越發受到重視。以物件導向資訊系統發展來說，現有軟體反向工程研究皆針對特定的物件程式語言以產生與系統內容相符的 UML（統一塑模語言）的系統模型圖、並發展自動化工具。然而，這些研究大多是研發如何將程式碼準確地轉換成 UML 圖，鮮少是針對圖形內容進一步檢核品質。此外由於軟體驗證如 UML 圖之品質檢核是知識密集的工作，其需要來自組織經驗之不斷累積以使品質檢核更臻完善且與時俱進。為此，本研究運用本體論、並針對系統設計品質中重要的耦合度量來發展一套知識導向的軟體反向工程 UML 線上品質檢測系統，簡稱為 OntREDQ。在架構與概念上，OntREDQ 由領域知識、規則引擎與品質推論所構成。其中領域知識是以 UML 結構模型中的類別圖與耦合度量為主之品質本體模型；而規則引擎與品質推論包含推論規則之設計以及針對圖形內容依據品質本體來分析與推論品質缺失並提供改善建議。本研究所發展的工具僅為一研究雛形，但希望藉由雛型與創新概念拋磚引玉，以供學術研究者或是發展 CASE 的實務企業所參考。

關鍵詞：軟體反向工程、本體模型、統一塑模語言、類別圖、耦合

* 本文通訊作者。電子郵件信箱：cychen@mgt.ncu.edu.tw
2019/04/08 投稿；2019/06/23 修訂；2019/07/23 接受

Chen, C.Y. and Hsu, T.H. (2019), 'The development of online and ontological quality evaluation on UML structural design in software reverse engineering: With a focus on coupling', *Journal of Information Management*, Vol. 26, No. 4, pp. 379-412.

The Development of Online and Ontological Quality Evaluation on UML Structural Design in Software Reverse Engineering: With a Focus on Coupling

Chung-Yang Chen*
Department of Information Management, National Central University

Tzu-Han Hsu
Pegatron Corporation

Abstract

Purpose ─ This research is to develop a knowledge-based ontological system, abbreviated as OntREDQ, for online quality analysis on UML structural models in SRE.

Design/methodology/approach ─ In terms of structure and concept, OntREDQ comprises a domain-specific ontology that focuses on coupling and an inference engine to diagnose quality defects of the input diagrams and to provide improvement recommendations.

Findings ─ The system is found to be able to effectively identify design coupling defects, as well as to allow users to provide feedbacks for continual improvement of the system.

Research limitations/implications ─ As most of the SRE studies focus on how precisely the conversion can reflect the code, this paper further looks into the quality of the converted results.

Practical implications ─ The proposed work helps project teams ensure design

* Corresponding author. Email: cychen@mgt.ncu.edu.tw
2019/04/08 received; 2019/06/23 revised; 2019/07/23 accepted

quality while jumping into implementation without system design under the SRE environment.

Originality/value－This research contributes to developing a coupling measure that integrates the static and dynamic aspects, as well as to establishing an ontological application that is for the software design and reserve engineering domain.

Keywords: software reverse engineering, ontology, class diagrams, coupling

壹、前言

由於現今軟體環境之高度不確定性與市場的快速變遷，使得軟體專案常被要求在更短的時間內完成開發。為此，部分軟體發展策略提倡在了解需求後隨即進行系統開發，以讓用戶盡快看見實際成果並及早提供意見以利修正（Brhel et al. 2015; Dingsøyr et al. 2012; Di Lucca et al. 2004）。因此開發團隊常採如敏捷方式等彈性作法－先著重於系統實作而非繁複的分析設計等文件化工作，以避免因後續不時的系統修正而徒勞於文件的更改（Selic 2009; Brhel et al. 2015; Dingsøyr et al. 2012）。然而許多研究亦指出，若缺乏充分的分析設計內容，將導致後續維護與再造階段因對系統的理解不足而造成不良影響（Fernández-Sáez et al. 2015; Canfora et al. 2011）。

有鑒於系統設計之重要性、且在滿足快速開發的要求之情況下，軟體專案可先完成系統實作後即運用反（逆）向工程（Software Reverse Engineering; SRE）方式來回推並產生系統設計相關內容（Cipresso & Stamp 2010; Canfora et al. 2011）。以現今重要的分析設計語言 UML 來說，軟體專案運用 SRE 方式產生對應的 UML 圖形，可有效降低因為頻繁的需求或程式變更而需要額外更新 UML 等文件化工作的時間與成本，並可解決在設計階段所規劃之內容與後來的系統不一致的問題（Fernández-Sáez et al. 2015）。因此，近來許多 SRE 研究陸續探討如何將程式碼自動轉成特定 UML 圖形，且商業或開源之系統發展平台也都陸續發展出此項功能－如 Eclipse、MS Visual Studio、ModelGoon 或是 Object-Aid（ObjectAid 2018）等等。

這些研究或工具皆是針對特定的物件程式語言（如 Java）以產生與原系統最相符的 UML 圖。然而，若原本的程式碼在撰寫時就品質不佳，那麼據以所自動生成的 UML 圖形內容也就不正確；而現有的 SRE 研究或工具大多專注於如何將程式碼轉換成 UML 圖，鮮少是針對圖形進一步提供品質驗證的機制。雖然有部分研究是探討反向工程與工具的品質（Osman & Chaudron 2018; Kollmann et al. 2002），然而其所探討的品質係指如何精確地呈現程式碼到 UML 圖形的脈絡，仍非所生成的圖形內容之品質（Chen et al. 2018）。因此對於運用反向工程之軟體開發專案而言，仍需要一套方法與系統來協助其在運用 SRE 工具產生 UML 圖之際也能夠即時評估圖形品質，以利程式與設計圖同步修正與優化。

由於軟體設計驗證如 UML 模型之品質檢核是知識密集的工作（Bashir et al. 2016; Genero et al. 2011），其需要來自組織經驗之不斷累積以使品質檢核更臻完善並與時俱進。在知識工程領域中，本體論（Ontology）是一項重要之研究主題。本體論應用至資訊系統的研究愈發受到重視，這是由於其所提供的知識系統架構、透過可分享性和共同理解性以利關鍵人員的溝通與知識之傳遞（Morente-

Molinera et al. 2016; Lu et al. 2013）。因此本研究運用 Ontology、並針對軟體逆向工程所產生的結構圖形－類別圖為例，進一步發展出一個知識導向的軟體反向工程 UML 線上品質檢核系統雛形、稱為 OntREDQ。就類別圖之品質檢核而言，本研究則以描述系統內容結構品質的耦合度（Coupling）為主。

OntREDQ 具有以下特色。首先在架構與概念上，其是由品質本體模型、規則引擎、與品質檢核推論所組成。其中，UML 結構圖品質本體模型由相關學理文獻與使用者組織的經驗實例所構成。品質檢核推論模組則是對品質本體透過規則引擎來推論圖形品質缺失、以及圖形與對應程式之改善建議。此外在耦合度的定義與度量方面，本研究嘗試結合靜態耦合（Static Coupling）與動態耦合（Dynamic Coupling），使得可從系統的程式內容與使用結果來驗證其結構設計品質。特別一提的是，在此所謂的動態耦合與 UML 圖形中的動態模型雖有著同樣的「動態（Dynamic）」一詞，但為不同之涵義。另外，OntREDQ 是以網頁為界面，以利專案團隊使用者在分散式的開發模式下能夠協同與彈性使用。文章接下來之結構如下：第貳節探討相關文獻；第參節說明系統之設計與研發；第肆節為展示系統實作成果；第伍節則是系統討論與分析；第陸節總結本文與建議未來研究方向。

貳、文獻探討

一、軟體反向工程與 UML

「反向工程（Reverse Engineering）」一詞起源於硬體分析，根據 Rekoff（1985）的定義，其是由原始開發者以外者，為了複製硬體設備所採取的一套工程機制。直言之，反向工程「破解」已完成產品的原設計內容，主要是用於產品之改善，但也被用於分析商業競品或是軍事敵人設備。後來，反向工程的概念應用於軟體開發，其用於實作完成的軟體系統以擷取出抽象資訊，藉以瞭解軟體的原設計內容，並幫助組織與人員提升對系統之理解力（Canfora et al. 2011; Gahalaut & Khandnor 2010）。

就系統的理解性而言，以圖表或模型來視覺化設計的方式是較能促進對系統的了解與掌握；其中，用以描述系統物件化內容的統一塑模語言（UML）較廣為使用（Lange et al. 2006）。在軟體反向工程（SRE）領域中，已有許多研究與相關工具可將低階的系統程式轉換成高階的 UML 圖型。例如稍早的 Keschenau（2004）開發了一套 SRE 工具 class2uml，可將物件導向語言 Java 的 Bytecode 轉換成 UML 圖型，以解決過去無法呈現多重關聯以及組合關係的問題。Di Lucca 等（2004）則針對網頁應用程式發展一套反向工程流程 WARE，將網頁應用程式的程式碼轉換成類別圖、循序圖以及使用案例圖。Franke 等（2011）運用 SRE 概念於行動裝置應用程式，其利用行動裝置應用程式的 log 資料來隨時掌握並可回推

該程式的生命週期各階段之內容。而 Sarkar 等（2013）亦針對 Java 語言發展了一套自動轉換 UML 類別圖的方法，其透過了整合 top-down 與 bottom-up 方法來理解程式內容。此外許多商業軟體或開源平台也陸續開發出 UML 反向工程功能。例如，開發環境 Eclipse 所提供的 UML 產生器可將 Java 原始碼轉換成 UML 圖型（Eclipse 2019），而 MS Visual Studio 則是能將 C#原始碼轉成 UML 圖形（Microsoft 2018）。此外開放源碼者如 ModelGoon（Lias Webdesign 2018）、或 Object-Aid（ObjectAid 2018）等 UML 反向工程外掛工具，皆可整合於各式開發環境中進行程式碼與 UML 圖型的轉換。

從上述探討可知，現今已存有許多依據不同程式語言類型的 UML 反向工程領域之研究與工具。然而，若 SRE 工具的轉譯能力不良，所產出的 UML 內容也就隨之有問題。而為了能產出良好的 UML 圖型，有研究即開始探討現有 SRE 工具之間的品質差異。例如稍早 Kollmann 等（2002）針對 Java 程式碼與 Java bytecode 比較不同的 SRE 工具所產出之 UML 圖型的優劣。Gahalaut 與 Khandnor（2010）同樣為探討 Java 程式碼的反向工程，並將反向工程分為程式開發者與分析者兩觀點，藉以定義 SRE 工具之效率與圖形結果的完整性。Osman 與 Chaudron（2018）則是對現有主要 SRE 工具進行品質評估，探討工具所產生出的 UML 圖型與程式碼之間的正確性。雖然這些研究皆是探討軟逆向工程品質，但大多是針對如何能更準確將程式碼投射在反向產生的 UML 圖形內容。只是，若程式碼本身若品質不佳，亦會直接影響所據以產生的設計圖型之品質；這對於在缺乏設計即先行實作系統的軟體快速開發環境與訴求下，更是一項挑戰。

二、本體論與本體語言

本體論（Ontology）源自於哲學領域，是用來描述真實世界中存在的實體或事物。自資訊系統時代的興起，本體論即應用於各種領域，如人工智慧（Fensel 2001）、語意網（Maedche & Staab 2001）、軟體工程（de Almeida Biolchini et al. 2007）、以及圖書館學（Guarino 1998）等其他領域。有關本體論的定義眾說紛紜，而被廣為人知的是 Gruber（1993）所提出的「本體論是概念模型的明確的規範」。其中，概念模型（Conceptualization）係指概念化或抽象化現實世界中複雜又多樣資訊之模型，而明確的規範（Explicit Specification）所指的是本體中的概念以及概念的限制必須有明確的定義（Chi 2009）。

為了建立某一特定領域的本體，其基本要素包含概念（Classes, or Concepts）、屬性（Attributes, Slots, or Roles）與實例（Instances）等等。而 Noy 與 McGuinness（2001）進一步定義概念為描述特定領域的抽象概念；屬性為描述各個概念的特徵及屬性或其之間關係；實例為概念下真實存在的案例。簡言之，本

體論所建立出來的概念模型可以視為一種讓機器理解的人類社會繁雜訊息的一種資訊模型。Martinez-Cruz 等（2012）指出，本體模型較適合被應用在複雜的知識管理。許多知識管理相關研究亦提到，知識管理系統是由許多的知識庫（Knowledge Base）所組成的，而本體論則是扮演重要之角色，這是由於本體論可幫助建立特定領域的概念模型以準確定義各種訊息的意涵，並在人與機器之間建立並共享該領域知識（Maedche & Staab 2001）。

為了協助各領域建立本體知識模型，許多研究便發展出各式本體語言以利實作，其中較廣為使用的即是 OWL（Web Ontology Language）。OWL 是 W3C 為了發展語意網（Semantic Web）而設計的語言，其目的是希望在未來能夠讓電腦了解人類所使用的網路上龐雜訊息之意義。OWL 並提供三種子語言－OWL Full、OWL DL（Description Logic）與 OWL Lite（McGuinness & Van Harmelen 2004）。OWL Full 雖然具有最大的表達能力以及語法自由度，但因其推論能力較低而較少有推論軟體可支援；而 OWL Lite 具有快速地分類能力，但其表達能力最弱，僅能允許 0 和 1 的基數值；表達能力介於上述兩者之間的 OWL DL，語法比 OWL Full 有較多的規範，但也因此而便於電腦進行推論的運算。

由於語意網的興起，使得推論規則與規則庫的建立越發受重視，其中規則的標準化與共用性是語意網研究者所為關注（O'Connor et al. 2005; Chi 2009）。為此，該領域發展出了許多規則撰寫的語言，像是 RuleML、SWRL、SQWRL、Metalog 與 ISO Prolog 等。其中 SWRL 與 SQWRL 是由 RuleML 演變而來的，用以補充 OWL DL 與 OWL Lite 的不足，也是目前 W3C 的主要規範之一（McGuinness & Van Harmelen 2004）。因此，透過 SWRL 或 SQWRL 所撰寫出來的規則可輕易的與 OWL 知識系統做結合，亦能直接使用 OWL 定義好的詞彙與關係而毋需另行定義。SWRL 與 SQWRL 的語法中可拆解為兩大部分 head 與 body，前者指的是推論的結果；後者指的是推論的前提，兩部分結合即為一個完整的推論規則。其規則中，「?」作為開頭代表變數，可表示某一個實例。

三、軟體工程本體

軟體工程與開發是一個複雜且涉及大量訊息的過程，這樣之特質加上每次開發的異質性，使得管理與傳承該領域知識為一項困難之任務。因此，利用本體論來概念並實例化軟體工程領域知識之研究逐漸受到重視，以便能共享領域相關知識（de Almeida Biolchini et al. 2007）。Happel 與 Seedorf（2006）提出軟體工程本體具有以下優點：(1)資訊形體化、(2)提供可共享的知識檢索與服務、(3)允許比較與配對知識或概念。如今，越來越多的軟體工程本體被建置，而 Zhao 等（2009）將這些的軟體工程本體分成數類－例如軟體流程本體、軟體文件本體、

軟體維護本體、與軟體品質本體等。

為了提升軟體品質，許多研究皆著手研究並建立相關的本體模型與應用。例如，Al Balushi 等（2007）發展出的 ElicitO 即是運用本體論方法協助捕捉品質需求，其品質本體是以 ISO/IEC 9126 品質模型為基礎所建立。Kayed 等（2009）蒐集 80 篇有關軟體品質屬性相關之研究報告，萃取內文中的術語來建立本體，以提供一個通用之語義框架。而 Chen 與 Tsai（2014）針對較需要經驗與知識才能挖掘的軟體需求建立一個需求品質本體 OWQFunc，用以挖掘潛藏的系統需求。Motogna 等（2015）為改善軟體品質，以 ISO/IEC 25010 的品質屬性與物件導向度量為基礎來建立本體，了解度量結果對應的品質影響。Bajnaid 等（2016）整理了數個主要的國際標準而建立軟體品質保證本體。Russo 等（2017）則根據軟體品質、軟體架構以及軟體流程三個不同面向，針對講求穩健運作的金控系統建立了 SQuAP-Ont 本體。就軟體品質而言，這些研究大多係針對軟體產品或相關議題而建立的本體研究，是偏向結果方面的品質確保。對於軟體發展過程中甚為重要的系統設計而言－特別現今常用的 UML 圖形，其內容與品質常因不同人員而異，因而有需要建置本體來彙集與管理組織或人員經驗與知識，以協助確保一致性與提升設計品質。

四、系統設計品質：耦合度（Coupling）

UML 圖形的品質向來是一項重要的議題（Chong & Lee 2015; Genero et al. 2011; Lange et al. 2006）。Chidamber 與 Kemerer（1994）指出，UML 結構模型的品質可從耦合度（Coupling）來評估。這是因為耦合度對於軟體系統的可維護性與可理解性具有較大的影響力。Gui 與 Scott（2007）定義耦合為兩物件之間存有互動時即具有耦合。Husein 與 Oxley（2009）再補充，具有耦合的兩物件之間的相互作用分別在兩種情況下發生，一種是方法間的呼叫，另一種則是變數的存取。許多研究指出，系統內容在設計上應減少耦合程度（Fowler 2001）。若類別間存在過度耦合，其將會妨礙模組化設計與再利用。此外耦合度越高亦代表模組或類別進行更動時將連帶影響其他模組或類別，因而降低了系統的可維護性（Chidamber & Kemerer 1994; Pfleeger & Atlee 2001）。

關於耦合度的測量，過去的研究大都是針對軟體的靜態特性。就耦合而言，軟體系統的靜態特性包括兩類：(1)設計資訊－諸如類別、方法、屬性或是類別間關係等較高階抽象的資訊；(2)原始碼資訊，可從中觀察出更詳細的訊息傳遞內容。早期較廣為運用的有 Chidamber 與 Kemerer（1994）所提出了耦合度量 CBO（Coupling Between Object Classes）以及訊息傳遞度量 RFC（Response For a Class）。CBO 是計算某系統中發生耦合的類別個數，其限於變數存取與方法呼叫

等耦合情況；而 RFC 則是計算一類別中訊息回應的次數，其是將類別的方法數與所直接呼叫的其他方法數加總而得。後來，Briand 等（2000）考量了間接的訊息傳遞，提出了 REC 的改良度量 RFC∞。此外 Li 與 Henry（1993）也延伸 RFC 而提出 MPC（Message Passing Coupling）與 DAC（Data Abstraction Coupling）度量－分別為計算類別中方法呼叫的訊息量，而後者是計算類別中採用其他類別作為屬性型態的屬性個數。然而 e Abreu 等（2000）認為，若僅計算耦合數或是訊息傳遞數量是無法呈現不同系統的耦合差異，並進而提出了 CF（Coupling Factor）。另外 Gui 與 Scott（2007）則提出了新的耦合度量 WTCoup，其不僅考慮了間接耦合，同時也可以區分出每項耦合的強弱，以解決類別功能複雜度不一的問題。

由上可知，靜態耦合（Static Coupling）度量是針對系統內容於設計階段時其方法呼叫和屬性參照做為基礎計算而得，是該系統在理論上的關聯設計。而動態耦合（Dynamic Coupling）則是測量系統在執行期間所發生的關聯，以反映實際所需要的耦合與程度。值得一提的是，在此謂之動態耦合與 UML 中針對功能演繹和方法訊息傳遞之「動態（Dynamic）」模型有著相同的字詞、但為不同的專注與涵義。Geetika 與 Singh（2014）指出，單以靜態耦合來看物件系統是較不足，這是因為物件特性像是多形、動態繫結及繼承皆是在系統執行時才可測得。動態耦合的計算依對象之不同而有類別與物件等兩觀點。Yacoub 等（1999）提出物件觀點的動態耦合度量 Export Object Coupling 與 Import Object Coupling，其分別針對系統於執行期間某物件對另一物件所傳出與傳入訊息之比例。Mitchell 與 Power（2004）則提出了類別觀點的動態耦合度量 Dynamic CBO，其計算在執行期間所發生耦合的類別數。Arisholm 等（2004）也提出一系列的動態耦合度量，其不僅考量不同之計算對象，還針對粒度（Granularity）以及訊息傳遞方向的不同進而定義出 12 種細部度量。爾後的研究也陸續提出各種動態耦合度量。例如，Hassoun 等（2004）提出的 DCM 是以物件間耦合的加總來呈現系統中物件觀點的動態耦合複雜度。而 Gupta（2011）所提出的方法則考量了系統執行期間物件所發生的繼承、多形等隱性耦合。

此外在軟體品質的研究中，為了提高程式的設計品質，作法之一即是找出程式碼中的「壞氣味（Bad Smell）」（Fowler et al. 1999），並對其進行重構（Refactoring）－即在不改變程式碼的外部行下修改系統的內部架構。在其著作中，Fowler 等（1999）列出 22 種壞氣味，如重複之原始碼（Duplicated Code）、依戀情節（Feature Envy）、惰性類別（Lazy Class）、無謂遺贈（Refused Bequest）等等。其中，與檢測未使用程式碼相關的則有 Lazy Class 與 Refused Bequest。Lazy Class 指的是當某類別被創造後其內容卻無充分使用，進而徒增該類別的理解與維護成本。而 Refused Bequest 是指子類別用不到來自父類別繼承的方法或屬性，此時該繼承的連結是浪費的。在耦合的觀點下，Liu 等（2012）和其他研究如

Arisholm 等（2004）所指出，這些設計瑕疵中，特別是未被使用的程式內容－包含 Lazy Class 所衍生出的 Useless Class、Useless Method 以及 Useless Field，對耦合最小化的原則而言有重大之負面影響。因此，本研究嘗試結合靜態與動態耦合並建立度量以檢測系統在兼具設計內容（靜態）與實際執行（動態）觀點下的未使用之耦合。

參、研究方法

一、系統架構

於本章，本研究提出一個針對軟體開發的逆向工程設計品質之線上檢測系統（Online Ontological Design Quality Assessment for SRE），簡稱為 OntREDQ。本系統可讓軟體開發團隊在先完成軟體程式碼後，在本地端（Local）透過反向工程工具產出 UML 設計圖形之際，於伺服器端（Server）進行圖形品質之即時運算並以網頁方式將結果線上同步呈現給團隊成員與專案相關關鍵人員，以利協同發展。此外在系統中，品質檢測之內容與規則是以本體論方法建置而成，其目的是為了讓組織的品質檢測知識重複利用、共享與修改。

如圖 1 所示，OntREDQ 在運作架構上包含三個部分：(1)擷取資料、(2)建置本體實例、及(3)品質運算與推論。首先，OntREDQ 收集受測系統經逆向工程工具轉換後的 UML 圖形資訊，並將收集到的資訊以實例方式存入本體模型內，再藉由規則推論出品質資訊，並從推論結果進行品質運算，並將計算結果呈現於網頁畫面中。接下來，我們將於以下各小節進一步說明這三部分之具體內容。

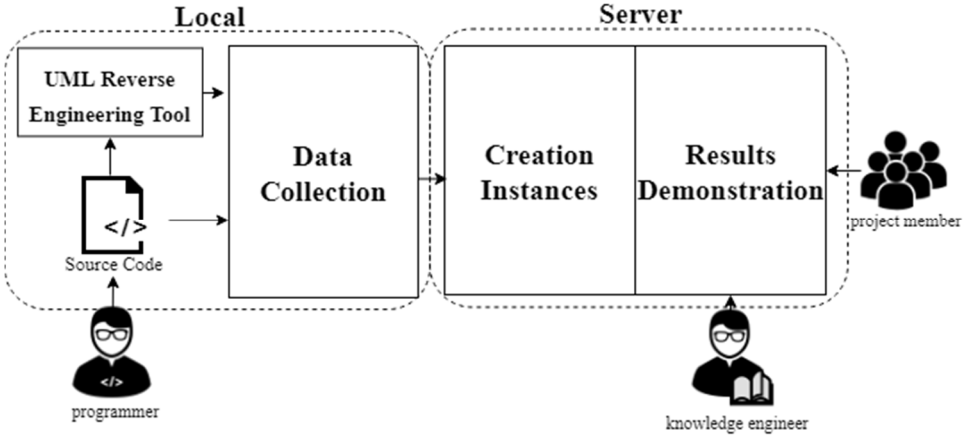

圖 1：OntREDQ 系統架構圖

二、資料擷取

目前現有的 UML 反向工程工具中，ModelGoon（Lias Webdesign 2018）是一個可將 Java 程式碼轉換成 UML 圖形之反向工程工具。由於該工具是一項開放原始碼的程式，適合學術研究之用，故本研究運用 ModelGoon 以搭配研發 OntREDQ 雛形系統。就設計品質而言，本研究專注於類別圖，這是由於類別圖能夠完整描述資訊系統的結構模型，綜合了系統為了實踐功能而所需要的所有類別、屬性與方法，是對於整個系統的結構而言有較為完整之描述。此外對於結構模型圖的品質來說，本研究以耦合度為檢測主軸。品質檢測除了使用靜態分析之外還搭配動態分析－即透過系統實際運作過程中來擷取動態連結，以讓檢測結果能從實務觀點來詮釋實際所需之耦合。於擷取資料時，本研究即是從 ModelGoon 中擷取系統在設計上的靜態資料與執行中的動態資料。

為了計算耦合度與建立以耦合度為主之品質檢測本體實例，OntREDQ 所需輸入的靜態資料是以待檢測系統的結構性內容為主，其包含系統內之套件、套件中的類別、以及類別中所有屬性與方法等各級成員。由於靜態分析為系統內成員間的結構性耦合是在呼叫（Invocation）、參照（Reference）與一般化（Generalization）三種情境下所產生，故擷取的資料中包含類別成員間的方法之呼叫、成員的型態參照、及類別繼承與實作。而在動態分析中，本研究運用 Hassoun 等（2004）所建議之方法－AOP（Aspect-Oriented Programming），並結合 AspectJ 來擷取系統動態耦合資訊（Hassoun et al. 2004）；而 Aspect 則包含了方法呼叫（Method Call）、建構子呼叫（Construction Call）、建構子執行（Construction Execution）、設定屬性（Set Attribute）及取得屬性（Get Attribute）等。例如以圖 2 的簡易示例來說，系統在運作時若有執行到 Demo 之第 7 行（圖 2-1），此時透過 AspectJ 可擷取到屬於 Method Call 的動態耦合紀錄。

```
1 package OntDQIS;
2
3 public class Demo{
4   public static void main(String [] args){
5     int a = 0;
6     Hello h new Hello();
      ....
7     if(a == 1){ h.display(); }
      ....
8   }
9 }
```

圖 2-1：Demo Class

```
1 package OntDQIS;
2
3 public class Hello{
4   public void display(){
5     System.out.println(
6       "Welcome to Ontological World");
7   }
     ....
     ....
8 }
```

圖 2-2：Hello Class

三、本體模型與規則建置

在伺服器端接收到本地端收集到的專案資料時，OntREDQ 透過已定義的本體概念模型來建置該受測系統的實例內容，並透過規則以推論其品質資訊。如第二章所提到，目前的本體研究中尚未有建立 UML 反向工程下的品質檢測本體，因此本研究依照建置本體的七個步驟（Noy & McGuinness 2001; Chi 2009; 陳仲儼 & 蔡鴻儒 2014）來發展之。本體的建置分成兩部分：建立概念（Build Concept）以及建立規則（Build Rule），而在概念與規則建立完成後即能據以建置實例。為了建立本研究的品質檢測本體，我們應用史丹佛大學所開發的一套「Protégé Ontology Editor and Knowledge Acquisition System」本體建置工具（以下簡稱 Protégé），並透過 Protégé 提供的外掛模組 Jess 來發展推論所需之規則（Chi 2009）。此外本研究採用 Apache Jena（簡稱 Jena）來自動將本地端接收到的資料進行實例轉換以及規則推論。

本研究的品質檢測本體在概念上包含四大類：靜態成員、動態成員、靜態耦合與動態耦合，我們使用上述的 Protégé 工具進一步以圖形來描述其內容。其中，如圖 3-1 所示，靜態成員中包含五概念（以橢圓形表示）：專案、套件、類別、類別成員與參數；類別包含兩子概念－屬性與方法。概念進一步包含屬性（即圖中的黑點），其中三個推論屬性（以粗體字示之）：is_Descendant、is_Accessed、與 is_Executed 分別為類別中的成員是否繼承父類別、屬性是否被存取以及方法是否被執行。動態成員本體（圖 3-2）則包含訊息與物件兩概念。動態成員的概念是參考 Arisholm 等（2004）提出的動態分析的資料模型：訊息的源頭與目標會是某一方法、每一個訊息的產生會有物件接收回傳結果、以及本研究的以下擴充：訊息的目標可能會是一個屬性且回傳的結果不一定會有物件接收其結果。

靜態耦合本體（如圖 3-3）包含呼叫、型態參照與一般化等三概念。呼叫有兩子概念－呼叫屬性（Invoke_Attribute）與呼叫方法（Invoke_Method）；型態參照有三子概念：屬性參考（Attribute_Refer）、方法參考（Method_Refer）與參數參考（Parameter_Refer）；一般化則有兩子概念：繼承（Inheritance_S）與實現（Realization）。本研究另外定義第四概念 ClassLevelCoupling，用來表示類別間的呼叫、型態參照與一般化的總體分數。此外圖中 is_Used 為推論屬性，表示某耦合關係是否於系統執行時期發生。最後是動態耦合本體，其包含兩個概念：動態繫結（Dynamic Binding）與繼承（Inheritance_D），並以圖 3-4 來展示動態耦合概念中的屬性關係。值得一提的是，動態耦合的繼承是指在程式執行期間因有實踐了繼承關係而產生成員之間的連接；此不同於靜態耦合的繼承－即設計時期類別間的繼承關係。也就是說，兩類別之間的繼承雖然在設計階段有建立（即構成理論上的耦合），但在系統實際執行時是有可能從未運用到該繼承特色－如子類別

從未執行父類別所提供之方法與變數等等。此時在重構（Refactoring）上會建議移除該繼承連結或是將兩者（類別）加以整併以減少不必要的耦合與系統複雜度。

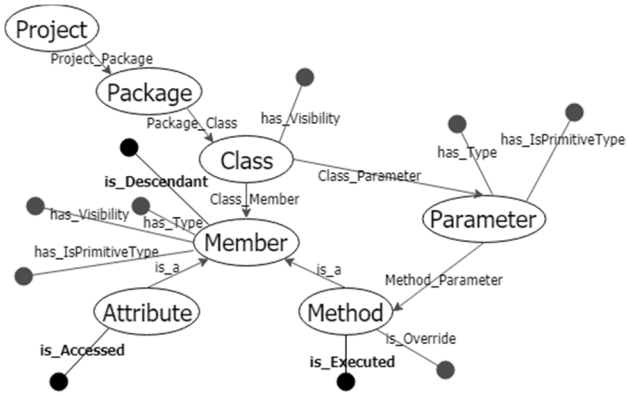

圖 3-1：靜態成員本體

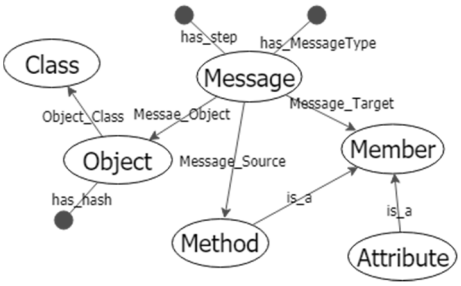

圖 3-2：動態成員本體

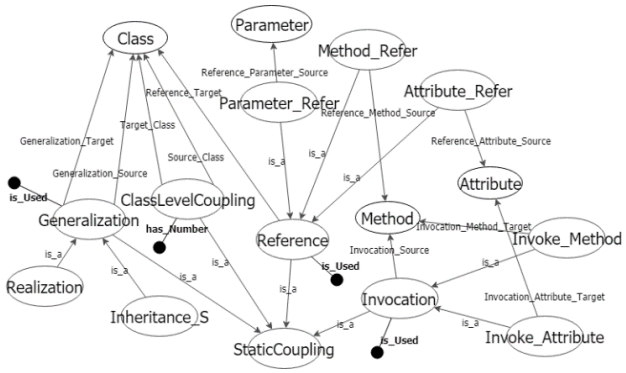

圖 3-3：靜態耦合本體

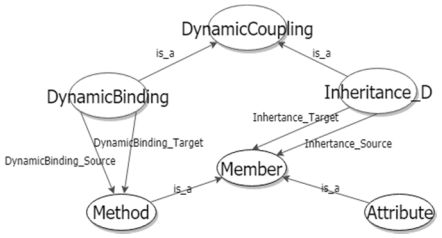

圖 3-4：動態耦合本體

在本體中，吾人可以藉由事實與宣告屬性等關係來找到相關實例，而進一步透過規則則可找到隱藏於這些已定義好的關係中的隱性知識（Madeche 2001; Wand & Weber 2002; Lee et al. 2006）。針對規則的撰寫，本研究利用 Apache Jena 所提供的 SQWRL（Semantic Query-Enhanced Web Rule Language），而內容包含 is_Descendant、is_Accessed、is_Executed 與 is_Used 等二階屬性，作為本體模型生成實例時推論屬性值等隱性知識。此外系統於執行期間所獲得的耦合資訊（動態繫結與繼承）也需由規則推論之。以下即是以 SQWRL 為格式的規則內容：

RL-1: Generalization(?g), Generalization_Target(?g, ?t), Generalization_Source(?g, ?s), Class_Member(?t, ?m), Attribute(?m) -> sqwrl:select(?t, ?s, ?m) ➔推論 Attribute 之 is_Descendant 屬性
RL-2: Generalization(?g), Generalization_Target(?g, ?t), Generalization_Source(?g, ?s), Class_Member(?t, ?m), Method(?m) -> sqwrl:select(?t, ?s, ?m) ➔推論 Method 之 is_Descendant 屬性與 is_Override 屬性
RL-3: Message(?m), has_MessageType(?m, ?t), Mesaage_Target(?m, ?a), Attribute(?a) -> sqwrl:select(?t,?a) ➔推論 Attribute 之 is_Accessed 屬性
RL-4: Message(?m), has_MessageType(?m, ?t), Mesaage_Target(?m, ?a), Method(?a) -> sqwrl:select(?t,?a) ➔推論 Method 之 is_Executed 屬性
RL-5: Message(?m), has_MessageType(?m, ?tp), Mesaage_Target(?m, ?msgtg), Message_Source(?m, ?msgsr), Invocation_Method_Target(?i, ?msgtg), Invocation_Source(?i, ?msgsr) -> sqwrl:select(?tp,?i) ➔列出所有擁有相同來源與目標的動態訊息與靜態呼叫(方法)
RL-6: Message(?m), has_MessageType(?m, ?tp), Mesaage_Target(?m, ?msgtg), Message_Source(?m, ?msgsr), Invocation_Attribute_Target(?i, ?msgtg), Invocation_Source(?i, ?msgsr) -> sqwrl:select(?tp,?i) ➔推論 Dynamic Binding 實例
RL-7: Message(?m), has_MessageType(?m, ?tp), has_step(?m, ?s) , Mesaage_Target(?m, ?msgtg)-> sqwrl:select(?tp, ?msgtg, ?s) ➔推論 Inheritance_D 實例
RL-8: Class(?c1), Class(?c2), Class_Member(?c1, ?msgtg1), Class_Member(?c2, ?msgtg2), Generalization(?g), Generalization_Target(?g, ?c1), Generalization_Source(?g, ?c2) -> sqwrl:select(?msgtg1, ?msgtg2) ➔驗證兩方法分別對應的類別間是否存在繼承關係

四、品質運算

本研究中，以耦合度為主的品質運算包含了三部分：(1)靜態耦合、(2)未使用耦合、以及(3)呈現未使用耦合部分的程式成員清單。許多稍早研究指出，系統內容宜避免不必要的耦合以降低系統的複雜性並利於系統的模組化和封裝（Chidamber & Kemerer 1994; Pfleeger & Atlee 2001; Fowler 2001）。在此訴求下、加上反向工程環境中系統已完成實作的條件上，本研究針對受測系統的實際執行所發生的耦合（動態耦合）來檢討在類別圖中的靜態耦合的必要性，並提出一個新型度量：未使用耦合（Unused Coupling; UC）。UC 與所實作之系統（見稍後之章節）可用以檢測系統內過度的耦合關係，並找出相關的程式冗員（如類別、方法、屬性等）以供專案團隊判斷是否移除或修改之決策參考。

在靜態耦合度量部分，本研究運用 Gui 與 Scott（2007）所提出之WTCoup，

這是由於其不僅考慮了間接耦合，同時也可以區分出每條耦合的強弱，甚至能解決類別功能複雜度不一的問題。假設受測系統中有 n 個類別，類別集合的表達方式為$C \equiv \{C_1, C_2, \ldots, C_n\}$。而$M_i$代表類別$C_i$的所有方法之集合，而$X_{i,j}$為$C_i$呼叫$C_j$的所有成員集合。值得一提的是，X是有向性的，因此$X_{i,j} \neq X_{j,i}$。$X_i$代表了$C_i$呼叫的外部成員集合，其表示如下：

$$X_i = \bigcup_{1 \le j \le n} X_{i,j} \tag{1}$$

接著我們計算C_i對C_j的耦合程度WCD(i,j)，其計算方式是C_i呼叫C_j的總成員數去除以C_i呼叫的總外部成員數與C_i的方法數總和，如公式(2)所示。WCD(i,j)的值域介於 0 到 1 之間。

$$WCD(i,j) = \frac{|X_{i,j}|}{|X_i| + |M_i|} \tag{2}$$

而為了考慮間接耦合，我們需要進一步識別出C_i到C_j的所有路徑π，並將每條路徑相乘以得WCT(i,j,π)。由於C_i到C_j的路徑並非僅有一條，為了計算出有意義的耦合度量，該度量僅考慮了權重值為最高之路徑。最後，我們將該系統中所有類別之間的WCT(i,j,π)進行加總，除以系統中任兩類別所可能發生的耦合數量，可得受測系統整體的耦合度WTCoup，如下所示：

$$WCT(i,j,\pi) = \prod_{edge(p,q) \in \pi} WCD(p,q) = \prod_{edge(p,q) \in \pi} \frac{|X_{p,q}|}{|X_p| + |M_p|} \tag{3}$$

$$\text{WTCoup} = \frac{\sum_{i,j=1}^{n} MaxWCT(i,j)}{n^2 - n} \tag{4}$$

在獲得靜態耦合資訊後，UC 進一步用於觀察系統的靜態耦合於實際執行時未使用的比例。在計算上，若 UC 越接近 1 代表幾乎所有的靜態耦合關係不在動態執行時發生，反之若趨於 0 則表示幾乎所有的耦合關係在動態執行時發生，是為最佳耦合狀況。本研究中，UC 是基於動態耦合所發生的三種情境：呼叫、型態參照與一般化。首先，在呼叫情境下的未使用耦合－如式(5)所示，是指有多少在靜態設定的方法呼叫於系統執行時未實際發生。再則是型態參照情境下的未使用耦合，如式(6)。其指的是有多少比例的屬性、方法（或方法內的參數）其宣告時型態參照另一類別然在系統執行期未被使用。最後是一般化情境下的未使用耦合，如式(7)。其代表有多少比例的屬性和方法是繼承來自父類別，然在系統執行期間未被子類別所使用。在三種耦合情境並考慮其重要程度不一時（例如物件導向系統使用者強調未使用之繼承時，可將其相關之權重調高），系統的加權 UC

（標示為 WUC）可進一步如式(8)所述。

$$\text{Unused Invocation(UI)} = \frac{number\ of\ invocation\ in\ runtime}{number\ of\ \ static\ invocation} \tag{5}$$

$$\text{Unused Reference(UR)} = \frac{number\ of\ member\ referring\ other\ class\ not\ used\ in\ runtime}{number\ of\ member\ refer\ other\ class} \tag{6}$$

$$\text{Unused Generalization(UG)} = \frac{number\ of\ member\ inheriting\ from\ super\ class\ isn't\ used\ in\ runtime}{number\ of\ member\ inherit\ from\ super\ class} \tag{7}$$

$$\text{Weighted Unused Coupling (WUC)} = W_{UI}UI + W_{UR}UR + W_{UG}UG \tag{8}$$

最後，為了獲得上述度量所需的數值，本研究是透過推論規則自動運算出的推論屬性來計算。其中，is_Descendant 與 is_Override 指的是靜態時各個 Method/Attribute 是否繼承自父類別，最後可用於計算 UG（式(7)）的分母（number of member inherit from super class）；is_Accessed 與 is_Executed 指的是程式於運作時，是否有使用該 Method/Attribute，其可用於 UR（式(6)）與 UG 的分子（number of member isn't used in runtime）；以及 is_Used 是指靜態的 Invocation 的關係中是否在程式運作時也執行過，可用於計算 UI（式(5)）的分子（number of invocation in runtime）。

肆、系統成果

一、系統預備與資料擷取

本章以主要操作案例來展現 OntREDQ 的實作成果。整體而言，系統運作包含四個步驟：(1)輸入受測系統程式碼、(2)產生 UML 類別圖、(3)AspectJ 環境準備、與(4)執行欲檢測專案。首先，專案團隊在行系統預備與資料擷取時先匯入受測系統程式碼。在本研究中，軟體工程師使用開發環境 Eclipse 撰寫程式或可藉由其所提供的功能來匯入現有程式，該功能於「File/Import/Existing Projects into Workspace」。為了幫助說明與簡化說明過程，程式碼部分本研究是取材自 Java 程式專書（Deitel & Deitel 2018）中的系統例來輔以展示。如圖 4 左半部所示，使用者將受測系統專案名為「atm」，並匯入該系統計有 12 支已實作之類別程式碼。接著，本研究在 Eclipse 中結合反向工程工具 ModelGoon 來讓專案團隊得以在匯入程式碼後自動產生 UML 類別圖，此時會產生.mgc 的檔案並於畫面中呈現，如圖 4 右半部所示。

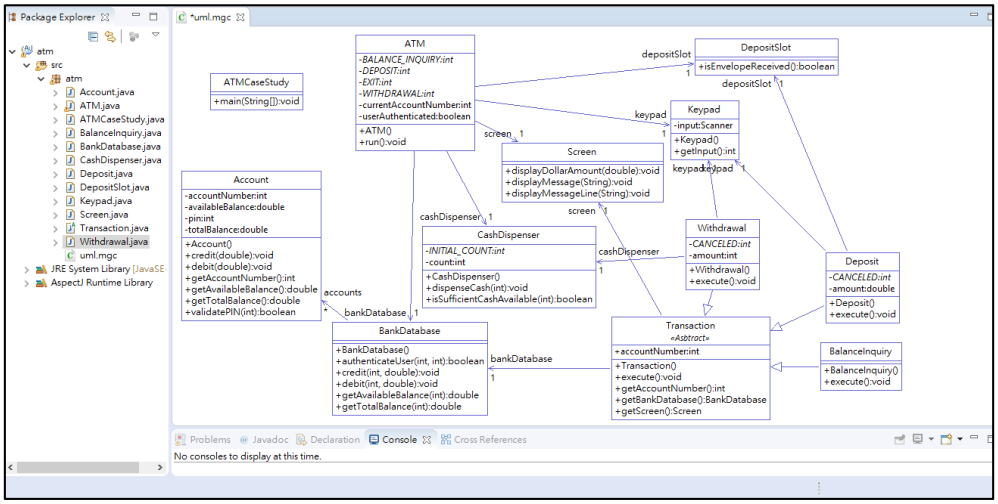

圖 4：系統預備(1)－產生類別圖

接著，系統透過 AspectJ 收集動態資料。此時而在操作檢測專案之前，需先設定好 AspectJ 的環境。首先，於專案的根目錄上按下右鍵，在清單中點選「Configure」下的「Convert to AspectJ Project」。接著，同上一個步驟至選擇畫面，於圖 5 中「OntREDQ」資料夾下選擇「Dynamic Analysis」，系統則產生型態為.aj 之檔案後即準備完畢。值得一提的是，在動態分析的研究中，目前最常見的方法是 AOP（Aspect-Oriented Programming）。在 Hassoun 等（2004）的研究中，為使用 AOP 方法獲取 Java 專案的動態資訊即是採用 AspectJ 工具來實現。由於 AspectJ 是目前最常見的 AOP 工具之一，且該工具可運行於 Eclipse 環境中，因此本研究所建立之系統亦是結合它來擷取動態資料。

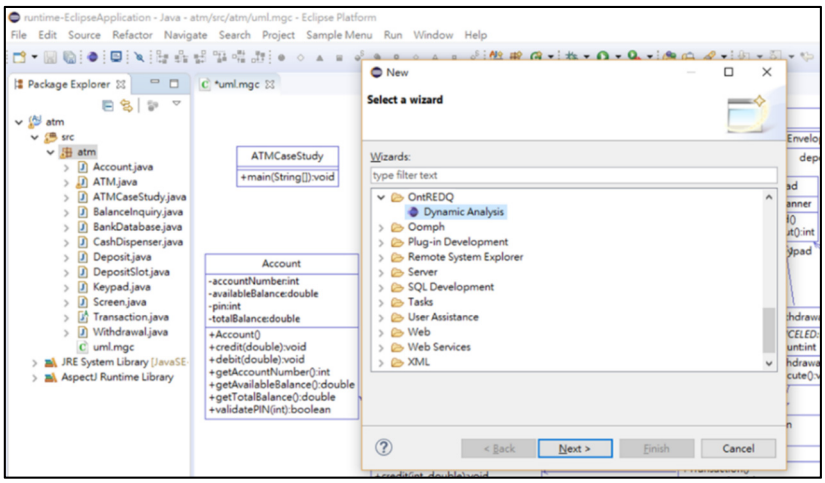

圖 5：系統預備(2)－呼叫耦合度計算模組

二、品質檢測

在完成上述資料匯入步驟後，即可開始執行耦合度檢測。在專案資料上傳至伺服器端後，專案團隊可透過網站查看受測系統的品質檢核結果。圖 6 之上半部是 OntREDQ 針對受測系統品質度量的計算結果。其中，第一項數值 0.0656 為靜態耦合指標 WTCoup。然而僅觀察靜態耦合無法得知系統可改善之處，因此可搭配本研究所提出的「未使用耦合指標 WUC」，即是第二項數值（0.2244）。此意謂有兩成左右的耦合關係是需要進一步檢視是否不必要，以降低系統的複雜度。而該圖之下半部則是進一步分析「未使用耦合」在三種情境（即呼叫、型態參考、與一般化）－即 WUI、WUR 與 WUG，其數值分別為 0.1361、0.037 與 0.5。

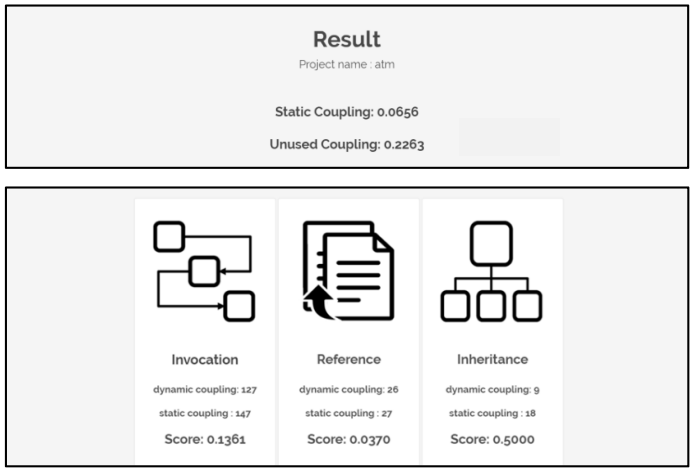

圖 6：系統耦合度檢測結果

以上數值進一步分析如下。首先，如第參節所述，本研究的靜態耦合（WTCoup）－即式(4)之計算是同時考慮各類別間的直接與間接耦合。本例之受測系統（ATM）中的間接耦合值為 8.6657，而類別數為 12，因此 WTCoup 為 0.0656。而未使用耦合則是透過規則所推論出的結果加以運算，並獲得靜態與動態耦合的個數。進而言之，對於是「呼叫」（即圖 6 中的 Innovation 部分）的耦合關係而言，透過系統的計算後得知靜態耦合共有 147 對關係、而動態耦合有 127 對關係產生，因此未使用耦合有 20 對，故 WUI 為 0.1361。就「繼承」（Inheritance）的耦合概念來說，系統於靜態結構中有 18 個來自父類別的 Method/Attribute，然而動態運作時實際使用數為 9，依式(7)得 WUG 值為 0.5。而「型態參照」（Reference）的耦合概念而言，靜態時有 27 個 Method/Attribute 型態參照了其他類別，而動態的實際使用數為 26，依式(6)得 WUR 為 0.037；綜合以上三種情境，並假設在相同權重設定下，未使用耦合 WUC（即式(8)）值為

0.2244。

除了分數的計算之外，本系統進一步視覺化呈現受測系統的耦合資訊與檢測結果。圖即是描述的是 atm 系統之整體類別層級（Class Level）的耦合關係圖。該圖左半邊呈現該系統的類別間之關係圖。若是類別間存在呼叫、型態參考與一般化關係則以一條直線呈現，並於連線上顯示關係的數量，且該連線具有方向性。而連線上的數字為類別耦合數，其包含了呼叫與型態參考兩類型。圖的右半部則是類別與連線的進一步說明，它分別顯示類別內的成員及有連線的成員資訊。系統耦合資訊的視覺化可助成員直覺觀察受測系統中的耦合關係，也彌補了目前 UML 反向工程工具所產生的類別圖資訊之不足。

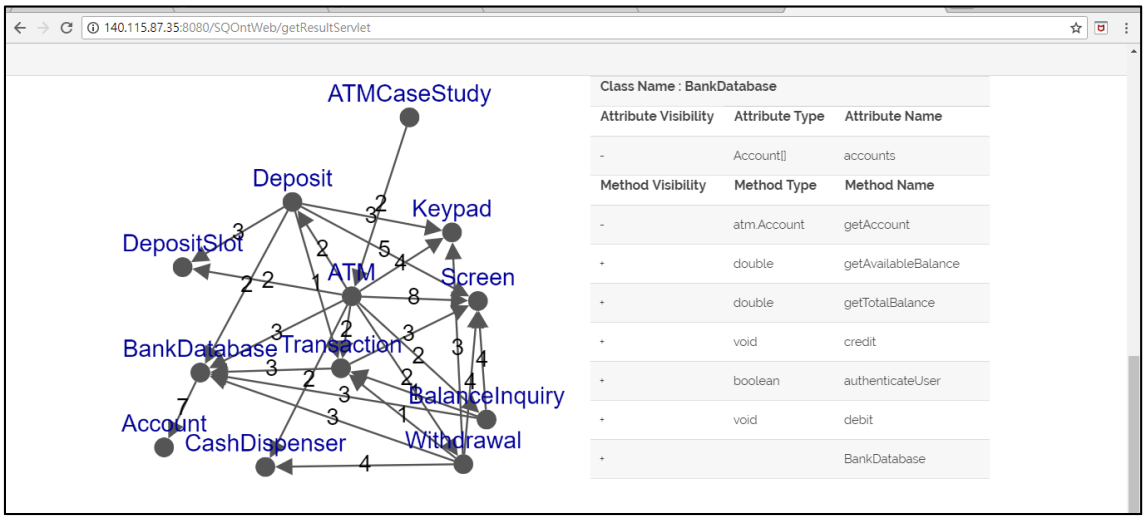

圖 7：系統展示－類別層級耦合關係圖

而有關圖 6 的檢測結果，OntREDQ 也視覺化呈現其三種類型（未使用呼叫、未使用型態參照與未使用一般化關係）中各數值之動態圖解。由於篇幅限制，本文僅以一般化關係來展示。圖 8 左半部為檢測後所發現的 Unused Generalization 之關係圖，圖中顏色較淺的點代表繼承自父類別之屬性或方法。而點與點的連線為繼承或實作關係，倘若有一般化關係的屬性或方法未在系統執行期間被使用，則該連線以黑色粗體線顯示。圖 8 右半部則是呈現未被使用的一般化成員列表以讓使用者進一步檢視並加以改善。值得一提的是，圖 8 是呈現完整的靜態耦合關聯之上、再輔以粗體線條凸顯出來（如圖 7 或圖 8 中的粗體現部分）不必要的耦合關聯以讓專案團隊檢視是否需要進一步重構。故圖 8 是結合動靜態耦合之展現，非僅是靜態部分而已。

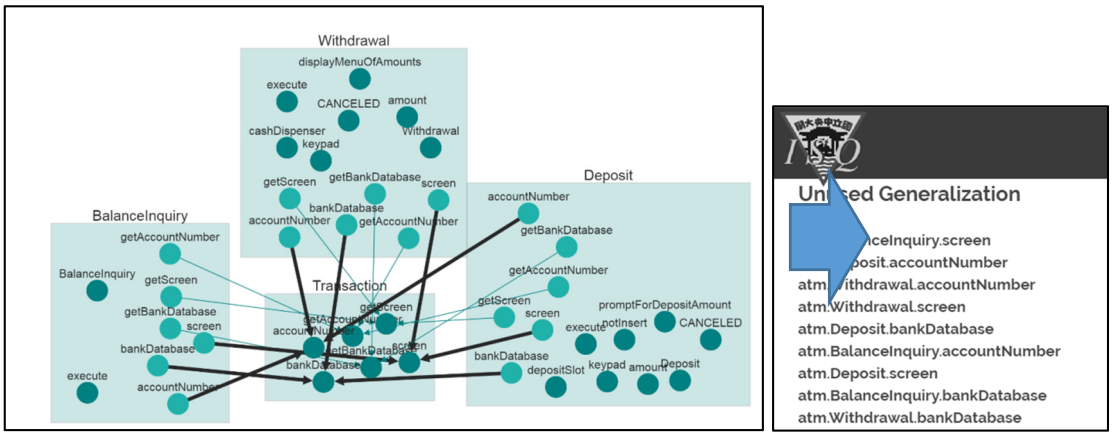

圖 8：系統展示－Unused Generalization 關係圖與列表頁面

伍、分析與討論

一、模型與系統驗證

就本體相關應用研究而言，Noy 與 McGuinness（2001）認為，知識本體的建置並沒有所謂的正確與否，其端看是否能符合其領域應用而定。雖如此，本研究認為仍可藉由其他要素的檢討來衡量所建置的知識本體具有結構優良、可擴充性與重複使用之效益。此外對於新模型與非量化模型的運作結果之探索，在以人為本（Human Centric）的使用環境來說，質性驗證是一可行的檢討方式（Seaman 1999; Darke et al. 1998）。由於本研究的實作內容與檢測標的具有高度人本特性，故以質性方式就模型與系統本身以及執行結果加以檢討。至於探討的內容，由於使用對象均為專業使用者，我們運用科技接受模式（Technology Acceptance Model; TAM）（Venkatesh & Davis 2000; Holden & Karsh 2010）、其主要以「認知有用（Perceived Usefulness）」與「認知易用（Perceived Ease of Use）」兩個主題來檢測使用者對於新資訊系統或服務的接受程度，並進而推論該系統對使用者之價值所在。本研究即依 TAM 模式與此兩問項為題綱進一步探討系統之效益。

由於探討資訊系統相關問題之質性研究中，半結構式訪談（Semi-Structured Interview）普遍被運用（Myers & Newman 2007）。因此我們採用此方式來蒐集資料以獲得較深入的資訊；受訪對象設定為潛在系統使用者－即 5 位具有軟體專案開發經驗之專家與從業人員，如表 1 所列示。在深度訪談後，本研究將受訪者之答測內容進行歸納與分析。此外本研究採用了目的性樣本，該樣本即先前描述的「atm」專案，以求在與讀者所見之相同系統例與內容下來進行分析與討論。

表 1：訪談成員簡介

編號	職稱與年資	相關業務範圍
1	資深專案主管（15 年以上）	專案管理、軟體研發、系統設計
2	資深專案主管（5~10 年）	專案管理、部門管理、系統設計
3	資淺專案主管（3~5 年）	專案管理、系統設計
4	資淺軟體工程師（3~5 年）	軟體研發、系統設計
5	新進軟體工程師（3~5 年）	軟體研發、系統設計

二、認知有用性

認知有用性（Perceived Usefulness）是指使用者相信使用某一特定系統將能夠提升其工作績效（Davis 1989; Holden & Karsh 2010）。在此題綱下，本研究的深訪係針對四個半結構式問題來進行與追問；並因篇幅起見，僅節錄重點呈現如下。

是否會因為開發時程過短而略過系統文件設計、並直接撰寫程式碼？又於該情況下是否需透過 SRE 工具來產生 UML 文件？ 受訪者回答如下：

> 「一般來說，在進行開發之前我們會先與客戶進行需求分析，並透過需求來撰寫系統文件，最後才會交給工程師進行程式撰寫。但常因客戶要先看到系統樣貌或是客戶的需求不明確時，我們會先撰寫程式再回頭整理設計內容…..。在此情況下，如果接續的工程師非同一人時，設計內容常就無法被理解。」（編號 1）
>
> 「傳統專案開發時，我們部門通常會先撰寫好系統文件再寫系統。但最近公司正在嘗試敏捷發展法，有同仁在需求分析後即開始寫程式了……不過在驗收時常發現程式品質有問題導致也花許多時間重寫該程式……此時即便有 SRE 工具來自動產出 UML 文件，內容還是一樣有問題。」（編號 2）
>
> 「理論上，我們都應先完成系統設計才開發……就算如此原先做好的設計文件與最終產出的真實系統還是會有明顯落差……因此我們只好從新再寫一份符合最新程式碼內容的 UML 文件。以這樣的情況來看，若有 SRE 工具的輔助是可以節省許多時間。」（編號 5）

首先，我們以本題來先確認受訪者的開發經驗是否符合本研究背景。此外本問題用於瞭解在真實情況中，SRE 所產生的 UML 圖形是否有品質檢測之需要。綜合受訪者的訪談內容可得，在目前軟體專案開發的現況中，先實作出系統再補（回建）文件的情況確為實務趨勢，且品質會因沒有系統設計而致終究在維護或未來發展有問題。由此可知雖然為了某些原因而先行實作系統，但後來仍需花時間來補齊設計內容；而在寫完程式的情況下藉 SRE 工具建立 UML 圖形並輔以組

織知識的支援檢測圖形品質是很有需要的。

1. OntREDQ 針對 UML 圖形的耦合度提供了兩項指標（靜態耦合度、未使用耦合）是否有助於理解設計品質？受訪者回答如下：

「可藉由分數知道耦合程度以及其未被使用的比例……特別是動態分析的部分，可幫助我們知道記憶體空間或 CPU 在執行期間被使用了多少。」（編號 1）

「我認為可以。分數可以拿來做參考。」（編號 2）

「若是用以理解系統的互動程度以及未使用耦合的比例是有幫助……但若有門檻值建議會更佳，例如高於多少需要被注意，低於多少可以忽略，好讓人員不要浪費過多的成本在小問題上。」（編號 3）

「對於驗證耦合是有用的，但若只考慮耦合度是無法瞭解其他設計問題。」（編號 4）

「分數可有效看出耦合程度並附加呈現了未使用的耦合成員……針對 Unused Coupling 總分的話，我認為不同公司對於這三個分數的權重有所不同，也許可以給使用者自行定義三種未使用耦合之間的權重。」（編號 5）

由於本研究是以靜態耦合度量來計算專案內類別間的互動程度，並以未使用耦合度量作為反指標來描述有多少比例的耦合關係在執行期間未被使用。兩指標的數值皆介於 0 與 1 之間且數值越低代表品質越好。透過訪談內容可知，這兩指標確實可助使用者得知系統內容的耦合設計品質，但對於陳述系統品質的完整性是不夠，可再研發加入其他品質指標。為此本研究認為，系統耦合程度常是不同的系統設計訴求下折衷的結果。例如在講求高度物件化的情況下，耦合值越低越好；然對於再利用而言則會增加系統成員的耦合。因此在未來發展上，系統確實應再納入其他品質考量，例如內聚力、複雜度等指標。

另外，本研究提出的兩項耦合指標的數值可考慮設定其門檻值，好讓使用者知道在哪些間距內的數值是有其被修正的需要。最後是本研究所提供的未使用耦合是針對呼叫、型態參照與繼承若進一步可設定權重則更能符合使用者需求。

2. OntREDQ 提供的未使用耦合建議是否有助於改善設計品質？受訪者回答如下：

「……可以從這個系統清楚看出三種情況下發生的未使用耦合。尤其是繼承關係的部分，有時候我很難解釋給新進工程師瞭解為何不可把變數宣告成全域變數……但藉由該系統所列出的未使用耦合可以讓他們知道自己程式碼有這類的品質問題。」（編號 2）

「我覺得寫程式的人會有些不自知的盲點而寫出像是多餘的引入與宣告

等。這在沒能預先規劃好系統的情況下會更嚴重……透過這個系統的提點，除了可瞭解系統成員間的相依程度外、還能夠幫助改善程式品質，算是一舉兩得」（編號 1）

「有的，該系統可以明確指出有疑慮的方法或屬性。不過，除了列出未使用耦合外，可以再多增加一些簡單的修改之建議。像是 Unused Generalization 的推論中，若繼承自父類別的方法卻未在系統執行中被使用，則子類別應是不需要繼承該方法。因此推論結果或許可提供更明確的改善推薦－像是建議該方法修正成 Private，而不是僅列出 Unused。」（編號 3）

本研究根據耦合發生於方法呼叫、型態參照以及繼承的三種情境下，列出不必要的關聯以供使用者行設計品質改善時參考。從回答中得知這樣的研究訴求確實是有助益、且除了設計品質外所研發的系統也能進一步呈現程式品質。本研究認為這是因為在非形式或非做做文件交差了事的前提下，軟體設計與系統實作是相互輝印且互有脈絡，故系統所提供的資訊同時也可作為改善程式之參考。另外，部分受訪者表示系統所推論出來的建議有可待加強之處，像是如上述的讀取權限宣告修正建議，是可以在日後增加於本系統的建議規則中。

3. 在有用性方面 OntREDQ 是否有其他改善之建議？受訪者的回答節錄如下：

「我覺得系統在計算動態耦合有個前提，就是它必須運作一段期間後所收集到的執行紀錄才能真確展現系統所檢測到的耦合冗員之真實性……因此我會建議系統在未來改版時可進一步提供執行軌跡的統計……例如某方法被呼叫的次數來表示冗員或不必要的耦合的程度」（編號 1）

「系統所做的品質檢測可作在測試階段所使用……當我們寫完並跑過每個功能時，可以同時藉由此系統得知有哪些方法或模組應該被使用而未被使用。在越大規模的跑測中，系統可以再進一步呈現耦合程度，以作為日後在維護程式與迴歸測試（Regression Test）時參考。」（編號 2）

本題項之目的在於收集系統未來發展方向。就受訪者所關注而言，由於動態耦合的計算是根據實際操作下所產生的呼叫與引用，因此若僅是形式地執行系統（即僅是紀錄呼叫與引用與否），是有可能降低結果之可信度（即所謂真實性）。透過受訪者所提出的建議，在未來發展上，我們將進一步研究提供各耦合路徑的使用次數，以更精確呈現耦合關係的使用程度，以讓專案團隊能據以掌握各個模組的使用頻率而訂製不同的管理方針、並更有效地分配物件記憶空間與物件垃圾清運（Garbage Collection）之動態規劃。此外在透過實際執行而得之各模組使用頻率，將有助於未來

類似專案的測試階段在時間與成本壓力下得以行更有效益的測試－如採取不同嚴謹程度的測試策略與測試案例之準備。

三、認知易用性

根據（Davis 1989; Holden & Karsh 2010）的定義，認知易用性是指使用者相信某一系統是容易上手的。為了探討使用者對系統之認知易用性，本研究針對以下兩個半開放式問題加以討論。

1. OntREDQ 是否容易操作？對此受訪者皆表示可接受，也提出幾項有關未來發展的建議：

「反向工程的設計品質檢測整件事需要先開啟 local 端的 Java 環境，然後再將資料上傳給系統評分。這部分對於研究雛型來說我認為是足夠的，但若要商業化，建議在未來可將整個流程都在伺服器端完成，將更簡易」（編號 2）

「在操作本地端系統時，有部分的動作是略嫌複雜……且系統主要功能應把該功能按鍵或連結放置於容易看到的地方，讓使用者的操作更容易」（編號 4）

「由於本身從事研發與設計，較能了解系統用意。但對新手工程師來說，很難不透過介紹來了解系統的主要用途與操作……建議在系統－例如網頁入口或工具列表附上教學頁面，又若能夠提供系統品質檢測背後的方法與原理來讓使用者參考，將會更優異」（編號 5）

2. 畫面是否容易閱讀？對此受訪者皆認為可接受並回饋如下：

「……對於系統提供未使用耦合列表與關係圖的部分，若要圖與表能夠互相參照，建議是左右擺放，而不是上下……」（編號 1）

「呈現受測系統的關係圖可能會因系統規模而致不易閱讀，且缺乏圖例解釋……但其他部分在閱讀上是沒問題……系統有放大功能，但若可局部放大更佳」（編號 3）

「這些關係圖中，除了系統原本所提供的耦合相關資訊外，我發現還能看到受測系統被最多人使用的類別成員是哪一些，因此可以找出關鍵程式成員，並針對該成員分配更多的心力去維護，做到重點管理之目的」（編號 4

「雖然整個系統的資訊量是足夠，可完整地了解系統品質的問題所在……但若系統的關係圖可提供標示圖例說明更佳……很難不透過人工介紹來了解這項系統的主要用途。此外，該系統的網頁入口或工具列表中若能夠加上系統品質檢測的方法與原理來讓使用者參考，使用者才會更加信任該系統的檢測結果」（編號 5）

整體而言，受訪者皆認為 OntREDQ 是相當具創新與應用價值，是因為目前在實務上並無一套類似方法或系統可幫助敏捷團隊在快速發展之際也能自動化地掌握品質，並可藉由這樣的系統對 UML 類別圖進行耦合度的品質檢測。此外該系統檢測的結果展示中，品質資訊是充足的、是有助於專案成員察覺潛在的系統品質問題。在訪談的過程中，專家們也給予許多系統優化的建議。例如就系統操作與閱讀來說，受訪者表示是容易操作的，但希望檢測結果之畫面除了文字敘述外還能搭配視覺化的關係圖將幫助使用者能更容易理解。

四、效度檢視

於本節中，我們針對研究訪談之效度加以檢視與說明。對於研究效度的確保，本研究從結構效度（Construct Validity）、外部效度（External Validity）、結論效度（Conclusion Validity）與內部效度（Internal Validity）（Kitchenham et al. 1995; Kitchenham & Pfleeger 2008; Shull et al. 2008）等來逐一探討。首先就結構效度來說，其可分為問項與對象兩方面。對於訪談對象的組成，本研究針對資訊年資來確保能藉由不同系統經驗角度來共同檢視系統的有用性與易用性，也讓讀者能更清楚掌握訪談對象與訪談內容的關聯性與實務面向。對於問項設計，本研究是依據科技接受模式並從其兩個重要前置因素來展開相關的題項，以確保問題結構之對稱與相關性。

有關外部效度而言，我們針對研究標的之外部擴散性與永續性等加以探討。就外部擴散性來說，本研究所發展的系統是有其應用範圍限制。其主要是針對專案團隊在採用敏捷發展或是在快速開發、早先體驗的發展策略與情境下，而非所有的開發方式均適用。對於系統使用效益的永續，我們建議軟體組織可運用 CMMI（Capability Maturity Model Integration）的永續化措施〔例如 II（Implementation Infrastructure）、GOV（Governance）與 MC（Monitoring & Control）等 Practice areas，詳（CMMI Institute 2018）〕來協助掌握與確保。

就內部效度來說，本研究著重於所獲得的驗證資訊之內部一致性。因此對於訪談方式，本研究採取半結構化之方式，是希望在先了解受訪者的回答意向同時也能進一步探索其思考邏輯與深度以確保在回答內容的內部一致性。最後就結論效度而言，可由受訪者代表性以及結論與相關推論的資料品質兩方面（Chen and Chong 2011）加以探討。首先就受訪者而言，本研究均事先加以挑選是有從事敏捷開發經驗、以確保其有足夠之代表性、並且針對受訪者事先實施系統教育訓練，以掌握其對受測標的是具有一定的熟悉度，進而確保結論的正確性。此外本研究亦將訪談一級資料與彙整後之資訊均先徵得受訪者之同意與其他受訪者之交叉檢視〔即 Member checking（Seaman 1999）〕，以確保結論與其來源的有效性。

陸、結論與未來方向

一、研究貢獻

在現代軟體環境中，軟體專案常需要在短時間內需盡速開發完成，使得開發人員省略了初期的系統設計文件而逕行系統開發。但缺乏文件化的設計資訊將導致人員理解力不足而造成系統在永續維護的影響。透過軟體反向工程工具所產生的 UML 圖有效降低專案在文件化的成本與時間，但卻也產生了潛在的設計品質問題以致系統同樣在永續管理的困難。本研究運用本體論方法、針對耦合來建立軟體設計品質檢測之知識架構，並實作一個系統以線上進行品質檢測，讓開發人員在快速發展下能兼顧設計品質。本研究之貢獻整理如下所列：

1. 現有用於評核 UML 圖形的耦合度量大多是針對系統的靜態內容，本研究則進一步搭配系統執行期間獲得的動態資訊，提出了一項新的品質度量，以能綜合檢視系統成員相依程度與存在之必要。
2. 軟體品質評核是一項知識密集的過程，需要透過組織不斷累積的經驗與知識讓品質檢測更完善並與時俱進。本研究透過本體論方法建立了 UML 品質檢測的知識系統，幫助建立品質檢測規則與累積實例，具有共享性以及共同理解性。
3. 在 UML 反向工程領域中，尚未有文獻是建立一套結合本體論方法的品質檢測系統、且無透過本體論方法來推論靜態與動態的軟體品質。本研究結合了這兩項研究議題而發展出 OntREDQ，以 Web-based 的方式呈現檢測結果。

就軟體開發而言，若循正向（即一循分析設計實作過程完成系統開發）來發展系統，即能有脈絡地將需求展開並佈署至系統功能與程式碼。而對於現今快速開發環境與用戶早期體驗策略來說，SRE 為一種補償性的做法。在特別情況下－即系統設計不可考時、或有需要快速開發、早期體驗下，SRE 是有助益的。本研究結論則進一步呼籲反向系統開發也應該要重視設計品質、而非僅以 SRE 工具把程式碼自動轉成設計圖形而已。此外就本文章的研究對象－類別圖來說，其是用以描述系統的靜態內容結構。而對於結構內容的耦合，本研究則是在 SRE 的基礎上進一步結合系統實際運作情況行交叉驗證。也就是說，在軟體正向發展過程中，系統在設計階段時由於尚未實作，是無法透過實際運作來檢討這些成員間耦合的必要性或可簡化與否。加上在現今所謂的早期開發與體驗的軟體發展趨勢與情境（註：非全部開發模式都如此）下，本研究專注於 SRE 的價值並結合靜態與動態的耦合資訊來協助這一開發情境同時也能兼顧有品質的系統設計。

二、研究限制與未來發展

本研究限制與未來發展敘述如下。首先，在系統優化與改善方面，由於 OntREDQ 為研究雛形，故許多畫面是以呈現研究成果為主而或感粗糙或是有待美化。因此在未來研究與推廣上，可進一步改善使用者介面、優化系統成果的使用度。第二、本研究對 UML 圖形的檢測指標為耦合度量並用以了解受測系統成員間的互動程度。未來研究可納入其他品質度量－例如複雜度與內聚力等，使得 SRE 設計品質檢測更加完整。第三、本研究所建立的系統可再進一步建立耦合度門檻值。對於門檻值的定義應根據系統型態、維護與未來擴充之需求性而調整數值。此外系統在累積使用後可收集各受測系統的結果，以進一步迴歸出耦合度的趨勢表現。

第四，針對動態分析，本研究所使用的方式為單元測試，是藉由執行全部功能後所得之動態資訊以推論系統中不必要的耦合。而未來可藉由一段時間的系統測試來收集動態資訊，得出的動態資訊透過未使用耦合的計算可瞭解系統生命週期中模組使用程度與狀況，據以建立專案維護計畫或給予不同的測試資源，使永續維護的成本得以有效配置。第五，對於耦合度的探討範圍，目前文獻大多是針對系統的程式成員；而本研究亦是針對程式成員（類別層級、或是類別的方法與變數層級）來探討系統內容之耦合。在未來研究我們建議可進一步涵蓋資料庫的呼叫，並將研究擴大至系統之間的資訊生態（Information Ecology）層級。最後，本研究係針對類別圖。在未來，我們將檢核對象延伸至其他設計圖形－如行為模型的活動圖或是循序圖等，以建構更完整的設計品質檢核。

參考文獻

陳仲儼、蔡鴻儒（2014），『應用本體論來線上展開資訊系統潛藏與知識密集之品質需求』，*資訊管理學報*，第二十一卷，第一期，頁 45-82。

Al Balushi, T.H., Sampaio, P.R.F., Dabhi, D. and Loucopoulos, P. (2007), 'ElicitO: A quality ontology-guided NFR elicitation tool', *The International Conference on Requirements Engineering: Foundation for Software Quality*, Trondheim, Norway, June 11-12, pp. 306-319.

Arisholm, E., Briand, L.C. and Foyen, A. (2004), 'Dynamic coupling measurement for object-oriented software', *IEEE Transactions on Software Engineering*, Vol. 30, No. 8, pp. 491-506.

Bajnaid, N.O., Benlamri, R., Pakstas, A. and Salekzamankhani, S. (2016), 'An ontological approach to model software quality assurance knowledge domain', *Software Engineering Lecture Notes*, Vol. 4, No. 3, pp. 193-198.

Bashir, R.S., Lee, S.P., Khan, S.U.R., Farid, S. and Chang, V. (2016), 'UML Models Consistency Management: Guidelines for Software Quality Manager', *International Journal of Information Management*, Vol. 36, No. 6, pp. 883-899.

Brhel, M., Meth, H., Maedche, A. and Werder, K. (2015), 'Exploring principles of user-centered agile software development: A literature review', *Information & Software Technology*, Vol. 61, No. 5, pp. 163-181.

Briand, L.C., Wüst, J., Daly, J. and Porter, D. (2000), 'Exploring the relationship bt/n design measures and software quality in object-oriented systems', *Journal of Systems and Software*, Vol. 51, No. 3, pp. 245-273.

Canfora, G., Di Penta, M., and Cerulo, L. (2011), 'Achievements and challenges in software reverse engineering', *Communications of the ACM*, Vol. 54, No. 4, pp. 142-151.

Chen, C.Y. and Chong, P. (2011), 'Software Engineering Education: A Study on Conducting Collaborative Senior Project Development', *Journal of Systems and Software*, Vol. 84, No. 3, pp. 479-491.

Chen, C.Y. and Tsai, H.R. (2014), 'Applying ontology to online functional deployment of latent and knowledge-intensive IS quality requirements', *Journal of Information Management*, Vol. 21, No. 1, pp. 45-82.

Chen, C.Y., Yang, C.Y. and Wu, K.W. (2018), 'Knowledge-based design quality in software reverse engineering: An introductory review', *International Conference on Information & Knowledge Engineering*, Las Vegas, USA, August 1-2, pp. 62-67.

Chi, Y.L. (2009), 'Ontology-based curriculum content sequencing system with semantic rules', *Expert Systems with Applications*, Vol. 36, No. 4, pp. 7838-7847.

Chidamber, S.R. and Kemerer, C.F. (1994), 'A metrics suite for object oriented design', *IEEE Transactions on Software Engineering*, Vol. 20, No. 6, pp. 476-493.

Chong, C.Y. and Lee, S.P. (2015), 'Analyzing maintainability and reliability of object-oriented software using weighted complex network', *Journal of Systems and Software*, Vol. 110, No. 12, pp. 28-53.

CMMI Institute (2018), 'Capability Maturity Model Integration for Development v.2.0', Retrieved from https://cmmiinstitute.com/cmmi

Cipresso, T. and Stamp, M. (2010), 'Software reverse engineering', *Handbook of Information and Communication Security*, Springer Berlin Heidelberg.

de Almeida Biolchini, J.C., Mian, P.G., Natali, A.C.C., Conte, T.U. and Travassos, G.H. (2007), 'Scientific research ontology to support systematic review in software engineering', *Advanced Engineering Informatics*, Vol. 21, No. 2, pp. 133-151.

Darke, P., Shanks, G. and Broadbent, M. (1998), 'Successfully completing case study research: Combining rigor, relevance and pragmatism', *Information Systems Journal*, Vol. 8, No. 4, pp. 273-289.

Davis, F.D. (1989), 'Perceived usefulness, perceived ease of use, and user acceptance of information technology', *MIS Quarterly*, Vol. 13, No. 3, pp. 319-340.

Deitel, P. and Deitel, H. (2011), '*Java How to Program*', Prentice Hall Press.

Di Lucca, G.A., Fasolino, A.R. and Tramontana, P. (2004), 'Reverse engineering Web applications: The WARE approach', *Journal of Software Maintenance and Evolution*, Vol. 16, No. 1-2, pp. 71-101.

Dingsøyr, T., Nerur, S., Balijepally, V. and Moe, N.B. (2012), 'A decade of agile methodologies: Towards explaining agile software development', *Journal of Systems and Software*, Vol. 85, No. 6, pp. 1213-1221.

e Abreu, F.B., Pereira, G. and Sousa, P. (2000), 'A coupling-guided cluster analysis approach to reengineer the modularity of object-oriented systems', *The 4th European Conf. on Software Maintenance and Reengineering*, Zurich, Switzerland, March 3, pp. 13-22.

Eclipse. (2019), *Eclipse: The Platform for Innovation and Collaboration*, retrieved from https://www.eclipse.org/

Fensel, D. (2001), *Ontologies*, pp. 11-18, Springer, Berlin, Germany.

Fernández-Sáez, A.M., Genero, M., Chaudron, M.R., Caivano, D., and Ramos, I. (2015), 'Are forward designed or reverse-engineered UML diagrams more helpful for code maintenance?: A family of experiments', *Information and Software Technology*, Vol. 57, No. 1, pp. 644-663.

Fowler, M. (2001), 'Reducing coupling', *IEEE Software*, Vol.18, No.4, pp.102-104.

Fowler, M., Beck, K., Brant, J., Opdyke, W. and Roberts, D. (1999), *Refactoring: Improving the Design of Existing Code*, Addison-Wesley Professional, Berkeley, CA, USA, 1999.

Franke, D., Elsemann, C., Kowalewski, S. and Weise, C. (2011), 'Reverse engineering of mobile application lifecycles', *The 18th Working Conf. on Reverse Engineering*, Lero, Ireland, Oct. 11-17, pp. 283-292.

Gahalaut, A.K. and Khandnor, P. (2010), 'Reverse engineering: An essence for software re-engineering and program analysis', *International Journal of Engineering Science and Technology*, Vol. 2, No. 6, pp. 2296-2303.

Geetika, R. and Singh, P. (2014), 'Dynamic coupling metrics for object oriented software systems', *ACM SIGSOFT Software Engineering Notes*, Vol. 39, No. 2, pp. 1-8.

Genero, M., Fernández-Saez, A.M., Nelson, H.J., Poels, G. and Piattini, M. (2011), 'Research review: A systematic literature review on the quality of UML models', *Journal of Database Management*, Vol. 22, No. 3, pp. 46-70.

Gruber, T.R. (1993), 'A translation approach to portable ontology specifications', *Knowledge Acquisition*, Vol. 5, No. 2, pp. 199-220.

Guarino, N. (1998), *Formal ontology in information systems: Proceedings of the first international conference (FOIS'98)*, June 6-8, Trento, Italy.

Gui, G. and Scott, P.D. (2007), 'Ranking reusability of software components using coupling metrics', *Journal of Systems and Software*, Vol. 80, No. 9, pp. 1450-1459.

Gupta, V. (2011), 'Validation of dynamic coupling metrics for object-oriented software', *ACM SIGSOFT Software Engineering Notes*, Vol. 36, No. 5, pp. 1-3.

Happel, H.J. and Seedorf, S. (2006), 'Applications of ontologies in software engineering', *The 5th Int'l Semantic Web Conference*, Nov. 5-9, Athens, USA.

Hassoun, Y., Johnson, R. and Counsell, S. (2004), 'A dynamic runtime coupling metric for meta-level architectures', *The 8th European Conference on Software Maintenance and Re-Engineering*, pp. 339-346, March 24-26, Tampere, Finland.

Holden, R.J. and Karsh, B.T. (2010), 'The technology acceptance model: Its past and its future in health care', *Journal of Biomedical Informatics*, Vol. 43, No. 1, pp. 159-172.

Husein, S. and Oxley, A. (2009), 'A coupling and cohesion metrics suite for object-oriented software', *IEEE Conf. on Computer Technology & Development*, pp. 421-425, Nov. 13-15, Kota Kinabalu, Malaysia.

Kayed, A., Hirzalla, N., Samhan, A.A. and Alfayoumi, M. (2009), 'Towards an ontology for software product quality attributes', *The 4th Int'l Conf. on Internet and Web Applications and Services*, pp. 200-204, May 24-28, Venice, Italy.

Keschenau, M. (2004), 'Reverse engineering of UML specifications from Java programs', *The 19th annual ACM SIGPLAN Conference*, pp. 326-327, Oct. 24-28, Vancouver, Canada.

Kitchenham, B.A. and Pfleeger, S.L. (2008), 'Personal opinion surveys', in *Guide to Advanced Empirical Software Engineering*, pp. 63-92. Springer, London.

Kitchenham, B., Pickard, L. and Pfleeger, S.L. (1995), 'Case studies for method and tool evaluation', *IEEE Software*, Vol. 12, No. 4, pp. 52-62.

Kollmann, R., Selonen, P., Stroulia, E., Systa, T. and Zundorf, A. (2002), 'A study on the current state of the art in tool-supported UML-based static reverse engineering', *The 9th Int'l Conf. on Reverse Engineering*, pp. 22-32, Oct. 29-31, Richmond, USA.

Lange, C.F., Chaudron, M.R. and Muskens, J. (2006), 'In practice: UML software architecture and design description', *IEEE Software*, Vol. 23, No. 2, pp. 40-46.

Lee, S.W., Muthurajan, D., Gandhi, R.A., Yavagal, D. and Ahn, G. (2006), 'Building decision support problem domain ontology from natural language requirements for software assurance', *International Journal of Software and Knowledge Engineering*, Vol. 16, No. 6, pp. 851-884.

Li, W. and Henry, S. (1993), 'Object-oriented metrics that predict maintainability', *Journal of Systems and Software*, Vol. 23, No. 2, pp. 111-122.

Lias Webdesign (2018), ModelGoon. Retrieved from http://www.modelgoon.org/

Liu, H., Ma, Z., Shao, W. and Niu, Z. (2012), 'Schedule of bad smell detection and resolution: A new way to save effort', *IEEE Transactions on Software Engineering*, Vol. 38, No. 1, pp. 220-235.

Lu, Y., Panetto, H., Ni, Y. and Gu, X. (2013), 'Ontology alignment for networked enterprise information system interoperability in supply chain environment', *International Journal of Computer Integrated Manufacturing*, Vol. 26, No. 1-2, pp. 140-151.

Madeche, A. (2001), 'Ontology learning for the semantic Web', *IEEE Transaction on Intelligent Transportation System*, Vol. 16, No. 2, pp. 72-79.

Maedche, A. and Staab, S. (2001), 'Ontology learning for the semantic web', *IEEE Intelligent Systems*, Vol. 16, No. 2, pp. 72-79.

Martinez-Cruz, C., Blanco, I.J. and Vila, M.A. (2012), 'Ontologies versus relational databases: Are they so different?', *Artificial Intelligence Review*, Vol. 38, No. 4, pp. 271-290.

McGuinness, D.L. and Van Harmelen, F. (2004), 'OWL web ontology language overview', *W3C Recommendation*, Vol. 10, No. 10, pp. 1-22.

Mitchell, Á. and Power, J.F. (2004), 'An empirical investigation into the dimensions of run-time coupling in Java programs', *The 3rd Int'l Symposium on Principles and Practice of Programming in Java*, pp. 9-14, June 16-18, Las Vegas, USA.

Morente-Molinera, J.A., Wikström, R., Herrera-Viedma, E. and Carlsson, C. (2016), 'A linguistic mobile decision support system based on fuzzy ontology to facilitate knowledge mobilization', *Decision Support Systems*, Vol. 81, pp. 66-75.

Motogna, S., Ciuciu, I., Serban, C. and Vescan, A. (2015), 'Improving software quality using an ontology-based approach', *International Conference on Meaningful Internet Systems*, pp. 456-465.

Myers, M.D. and Newman, M. (2007), 'The qualitative interview in IS research:

Examining the craft. Information and Organization', Vol. 17, No. 1, pp. 2-26.

Noy, N.F. and McGuinness, D.L. (2001), 'Ontology development 101: A guide to creating your first ontology', *Stanford Knowledge Systems Laboratory Technical Report*, Stanford, CA, USA.

O'connor, M., Knublauch, H., Tu, S., Grosof, B., Dean, M., Grosso, W., & Musen, M. (2005), 'Supporting rule system interoperability on the semantic web with SWRL', *The 2005 Int'l Semantic Web Conference*, pp. 974-986, Nov. 6-10, Galway, Ireland.

ObjectAid. (2018), *The ObjectAid UML Explorer for Eclipse.* Retrieved from http://objectaid.com

Osman, H. and Chaudron, M.R. (2018), 'Correctness and completeness of CASE tools in reverse engineering source code into UML Model', *GSTF Journal on Computing*, Vol. 2, No. 1, pp. 193-201.

Pfleeger, S.L. and Atlee, J.M. (2001), *Software Engineering: Theory and Practice*, Pearson Education Inc.

Rekoff, M.G. (1985), 'On reverse engineering', *IEEE Transactions on System, Man, and Cybernetics*, Vol. 2, No. 4, pp. 244-252.

Russo, D., Ciancarini, P., Falasconi, T. and Tomasi, M. (2017), 'Software quality concerns in the Italian bank sector: The emergence of a meta-quality dimension', *The 39th IEEE/ACM Int'l Conf. on Soft. Eng.*, pp. 63-72, May 20-28, Buenos Aires, Argentina.

Sarkar, M.K., Chatterjee, T. and Mukherjee, D. (2013), 'Reverse engineering: An analysis of static behaviors of object oriented programs by extracting UML class diagram', *International Journal of Advanced Computer Research*, Vol. 3, No. 3, pp. 135.

Seaman, C. (1999), 'Qualitative methods in empirical software engineering', *IEEE Transactions on Software Engineering*, Vol. 25, No. 4, pp. 557-572.

Selic, B. (2009), 'Agile documentation, anyone?', *IEEE Software*, Vol. 26, No. 6, pp. 11-12.

Shull, F.J., Carver, J.C., Vegas, S. and Juristo, N. (2008), 'The role of replications in empirical software engineering', *Empirical Software Engineering*, Vol. 13, No. 2, pp. 211-218.

Venkatesh, V. and Davis, F.D. (2000), 'A theoretical extension of the technology acceptance model: Four longitudinal field studies', *Management Science*, Vol. 46, No. 2, pp. 186-204.

Wand, Y. and Weber, R. (2002), 'Research commentary: Information systems and conceptual modeling', *Information Systems Research*, Vol. 13, No. 4, pp. 363-376.

Yacoub, S.M., Ammar, H.H. and Robinson, T. (1999), 'Dynamic metrics for object oriented designs', *The 6th Int'l Conf. on Software Metrics Symposium*, pp. 50-61, Nov. 4-6, Boca Raton, USA.

Zhao, Y., Dong, J. and Peng, T. (2009), 'Ontology classification for semantic-web-based software engineering', *IEEE Transactions on Services Computing*, Vol. 2, No. 4, pp. 303-317.