國立政治大學資訊科學系

Department of Computer Science
National ChengChi University

碩士論文

Master's Thesis

基於行為驅動開發製程的區塊鏈智能合約整合測試

A Study of BDD-style Smart Contract Integration Test on the Blockchain

研 究 生：鄭敬儒

指導教授：廖峻鋒 博士

中華民國一〇七年六月

June 2018

DOI:10.6814/THE.NCCU.CS.001.2018.B02

基於行為驅動開發製程的區塊鏈智能合約整合測試

A Study of BDD-style Smart Contract Integration Test on the Blockchain

研究生：鄭敬儒　　Student：Ching-Ju Cheng

指導教授：廖峻鋒　　Advisor：Chun-Feng Liao

國立政治大學

資訊科學系

碩士論文

A Thesis

submitted to Department of Computer Science

National Chengchi University

in partial fulfillment of Requirements

for the degree of

Master

in

Computer Science

中華民國一〇七年六月

June 2018

DOI:10.6814/THE.NCCU.CS.001.2018.B02

摘要

近年來區塊鏈技術受到相當重視，相關應用也開始大量被開發。智能合約是運行於區塊鏈上，用於執行業務、交易的重要元件。近年來多學者不約而同地發現，如何驗證智能合約正確與完整反映繁瑣的業務規則，是區塊鏈應用程式開發的重要議題。然而，目前針對此議題仍欠缺系統化整合驗證與測試機制來確保所開發智慧合約的正確性。針對此一挑戰，本研究主要目的在於探究如何將行為驅動開發製程應用於區塊鏈智能合約的整合測試，利用行為驅動開發結合測試驅動開發製成實際開發一個支援 BDD 開發方法的 Solidity 智能合約語言自動整合測試平台，並且在本論文中以紅利點數交換為案例作為示範。此外，本研究亦以購物網站之購物車需求作為案例，就所開發系統進行使用者質性測試，針對易用性進行檢驗，研究結果顯示，本研究提出的機制能有效降低智能合約開發測試複雜度與負擔，提升合約品質。

關鍵字：區塊鏈、智能合約、行為驅動開發

DOI:10.6814/THE.NCCU.CS.001.2018.B02

Abstract

The importance of blockchain technologies and applications increases rapidly in recent years. A smart contract is a software component that encapsulates business and transaction logic of an application running on top of a blockchain network. Automatic integration testing and verification of smart contracts have become a vital software engineering issue of contract development. Nevertheless, there still lacks a systematic automated integration testing and verification mechanism. This thesis proposes a BDD-style automatic integration testing platform for Solidity–based smart contracts by considering the cross-cutting concerns of integration testing. Besides, this research has implemented a prototype system and a loyalty point exchanging scenario. The outcomes of this research are helpful for minimizing the cost and complexity of smart contract development and thus increase the quality of the blockchain applications.

Keywords: Blockchain, Smart contract, BDD

DOI:10.6814/THE.NCCU.CS.001.2018.B02

目錄

DOI:10.6814/THE.NCCU.CS.001.2018.B02

DOI:10.6814/THE.NCCU.CS.001.2018.B02

圖目錄

DOI:10.6814/THE.NCCU.CS.001.2018.B02

DOI:10.6814/THE.NCCU.CS.001.2018.B02

表目錄

DOI:10.6814/THE.NCCU.CS.001.2018.B02

第 1 章

緒論

1.1 研究背景

近年來由於比特幣的興起，底層的區塊鏈技術開始被重視及推廣應用。區塊鏈本質上是一個去中心化的資料庫，不屬於任何一個機構集中式的控管，所有參與紀錄的人員皆共同記錄資料庫資訊。簡言之，區塊鏈實現了一種不依賴第三方、通過自身分散式節點進行資料的存取、驗證、傳遞等功能。

1990 年 Nick Szaboo 提出了智能合約的概念[1]，其願景是希望所有條款可以透過電子化而被落實，一種常見的範例是自動販賣機，消費者投入規定的金額然後機器吐出金額對應的物品。但當年因為技術發展的限制，除此之外的實例並沒有太大的迴響。數年後於 2008 年網路上出現了一位匿名為中本聰的學者，發表了一篇描述比特幣技術的的文件[2]，之後，該文件所描述的概念於 2009 年被實現並且成功上線，即是比特幣區塊鏈平台。此舉開啟了金融科技的新革命，記帳不用依賴銀行或其他第三方中間單位作為信任依據，不信任的任何個體可以透過區塊連平台做交易，且不用畏怕銀行會有倒閉的可能、政府政策所造成的通貨膨脹、交易過程會有遺失或被欺騙的可能，而此正為比特幣的初衷，希望數位加密貨幣不被任何政府或中央銀行控管而又擁有可被依賴、信任的特性。

透過比特幣實現了去中心化不被控管的加密貨幣，以及智能合約電子化條款的概念，於 2013 年至 2014 年的期間，另一個開源的區塊鏈平台以太坊落實了智能合約概念，透過將合約條款電子化，撰寫成智能合約後將其部署在區塊鏈上且被執行，區塊鏈慢慢地

DOI:10.6814/THE.NCCU.CS.001.2018.B02

被視為具有自主性、可追溯、點對點、去中心化、具拜占庭容錯能力的通用分散式資料庫與合約執行平台，其應用也從單純的虛擬貨幣推展到一般金融應用如支付匯款、股權清算、群眾募資、資產登錄、甚至於結合物聯網[3]，在民生、工業、健康照護與文創等產業也產生許多新應用[4]。其中最有名的案例為跨境匯款，透過區塊鏈可信及智能合約的判定，原本需要三天才能完成的跨境匯款可以縮短至五秒即完成。

1.2 研究動機

目前區塊鏈技術仍在發展階段，許多軟體工程議題研發都仍未完善，Porru 等人於 2017 年指出[19]，基於區塊鏈的軟體架構議題將出現，因原本既有系統大多皆是中心式架構，而現今區塊鏈則是分散式架構，並不會有一種區塊鏈應用架構會適合所有既有的中心式系統架構升級。與此同時，區塊鏈的出現將會有新的專職出現，將需要一個新的職位來協助業務人員及資訊人員，此種職位的人不僅僅需要熟悉系統架構，且也需要具有金融、法律以及資訊科學知識。

區塊鏈技術是結合分散式系統與密碼學跨域技術，本身就具有一定開發門檻，若要開發區塊鏈應用系統，必需要熟知分散式系統架構、熟悉一般應用系統的開發，並且對密碼學需有基本認知。整合區塊鏈應用至既有系統的複雜度比整合一般應用系統高出許多，一般來說必須要有專職人員可以協調及提供建議，因此，要快速建構具備一定品質的區塊鏈應用服務比起建構一般企業應用系統明顯的代價明顯高出許多。

即使完成應用服務的建構，隨著應用規模擴大、整合層面擴張、業務邏輯愈來愈複雜，困難度相對也會越來越高，若未來區塊鏈技術若要真正成為創新應用的基底，仍有許多必須克服的挑戰，舉例來說，目前開發整合應用場域服務結合區塊鏈、智能合約的技術門檻高，且開發時程長。即使對已習得相關技術的研發人員來說，仍需耗費大量時間重覆寫作大量和實際業務邏輯不相關的「橫切面(Cross-cutting concerns)」程式碼。即

DOI:10.6814/THE.NCCU.CS.001.2018.B02

使是教學用的基本 Web 應用程式結合區塊鏈程式範例，在存取特定智能合約時，光參數設定就超過 100 行。

智能合約(Smart Contract)是運行在區塊鏈上用於執行業務、交易邏輯的元件(由程式碼構成)，可用於支援匿名的點對點交易，而交易的有效性驗證與記錄由區塊鏈結點予以執行，不需要中介服務。而開發人員在開發智能合約所面臨到的其中一大問題為其所寫作之智能合約是否真能正確與完整反應繁瑣的業務規則。更精確地說，目前尚欠缺系統化驗證與測試機制來確保所開發的智能合約正確性。

大部分的智能合約形式上看起來很像程式語言中的類別，由代表類別實體的狀態(state)的成員變數與其行為方法(method)所組成。因此，一個可能的驗證方法是採行TDD(Test-Driven Development)針對合約中的每一個 method 進行測試。TDD 的精神主要是開發軟體者先撰寫單元測試(Unit Test)程式，再以單元測試為依據，設法開發程式讓程式通過單元測試，只要通過單元測試即完成該單元軟體開發。目前智能合約開發社群主要的解決方案正是針對特定合約語言，配合特定的支援函式庫，採用特定的單元測試工具，以手動的方式，自行接合整個工具鏈[5]。然而，即使目前熟練的合約開發人員可以透過手動方式實現單元測試工作，但在邏輯複雜的情況下，這些單元測試工具並不足以確保整體業務邏輯的正確性[6]，因為即使每一個方法都正確被實作與測試通過，並不代表每個單元組成後，整體能依照特定業務邏輯正確運行。因此，整體來說，到目前為止，智能合約欠缺成熟的開發工具與方法，也欠缺易用整合度高的測試工具。

1.3 研究目標

針對欠缺系統化的智能合約驗證與測試機制的挑戰，近年來軟體工程領域有許多以「使用者角度描述系統應有行為的整合測試自動化」的想法，這些想法不約而同出自不同社群的學者或工程師在相近時間被提出，他們皆主張積極與使用者溝通，共同定義(user

DOI:10.6814/THE.NCCU.CS.001.2018.B02

story)敘述的需求文件。如行為驅動開發(BDD, Behavior-Driven Develop- ment) [7]，接受測試驅動開發 ATDD(Acceptance-Test Driven Development) [8]、可執行的規格案例(SBE, Speci cation By Example) [9] 等均是被提出來解決類似問題的方法。在實現步驟上雖有些許差異，但主要目的均是以「情境(scenario)」為核心來測試。

智能合約的開發與測試仍欠缺系統化的整合開發測試工具或平台。具體來說，目前並沒有適合智能合約的 BDD 工具被提出，也沒有使用 BDD 方法開發區塊鏈與智能合約結合應用場域的應用案例與經驗報告出現。因此，針對此一挑戰，本研究首先將探究基於 BDD(Behavior-Driven Development) / ATDD (Acceptance-Test Driven Development)，以「情境(scenario)」為核心來測試就業務邏輯的角度來驗證合約正確運作時應有的可行性。藉由實現一個可支援 Solidity 智能合約語言的自動整合測試平台進行研究，以紅利點數交換的智能合約開發為案例，就所開發的系統進行可用性驗證。此平台以 Web 方式提供自動整合測試服務，將整合數個現有工具，包含 Cucumber.js、Web3.js 與 Mocha，提供並解決目前開發與測試 Ethereum 上以 Solidity 寫作的智能合約的橫切面考量，也就是將每次使用這些工具開發、測試智能合約時都必須進行的重覆工作，整合封裝為 Web API，並提供簡易 Web 使用介面，能有效降低智能合約開發測試複雜度與負擔，提升合約品質。

1.4 相關研究

BDD 如何有效應用在各式開發場域，近年來是軟體工程研究的熱門議題。Li 等人[14]認為使用 cucumber 作為 BDD tool，因需要手動敘述測試場景，通常會造成測試場景的覆蓋度不夠，故他們建議並設計了一種 Model- Based Testing tool 可以自動生成有效的 cucumber 測試場景，藉由讀取 UML 圖(例如： state machine diagram)，識別 elements 後生成有效的測試場景，改良了使用 BDD 手動敘述測試場景所遺漏的部分。

DOI:10.6814/THE.NCCU.CS.001.2018.B02

Rahman 與 Gao[15]提出了當 BDD 應用於微服務(micro-service)時，針對重用性、可追縱性與可維護性形成了主要問題，他們提出了一個自動驗收測試架構來解決此問題。Sivanandan 與 Yogeesha 則探討了在敏捷開發過程中如何有效的使用 Model Based Testing 進行 BDD[16]，並對 BDD 的使用方式進行了深度的探討。

雖然藉由讀取 UML 圖形式別 elements 生成測試場景，雖然可以改善手動敘述而會遺漏的困難，但一般而言非資訊相關背景者，例如：業務人員、銀行背景人員.....等。在撰寫 UML 圖時便會遭遇困難，因為不是每個使用者都懂得如何撰寫，故在本篇論文使用撰寫需求的規格是使用 Gherkin 作為撰寫。

而使用 Model Based Testing 進行 BDD 可以解決系統難以維護的問題，以及可以更容易驗證所撰寫的系統是否符合業務需求，但使用這樣的架構在智能合約開發上面，雖然可以獲得此架構所帶來的好處，但這樣的架構使開發流程變得更為複雜，以及若直接只用此架構，在撰寫整合測試以及單元測試時仍需要了解區塊鏈背景才能順利使用此開發方法流程，故在本篇論文針對了此議題提供了相對應的解決方式。

DOI:10.6814/THE.NCCU.CS.001.2018.B02

第 2 章

技術背景

2.1 技術背景

本論文以解決區塊鏈應用系統開發所面臨的挑戰為目標，基於區塊鏈應用系統，本研究使用行為驅動開發（Behavior-driven development, BDD）技術來解決智能合約缺乏成熟的開發工具與方法、缺乏系統性的驗證與測試機制來確保所開發合約的正確性以及目前仍缺乏精通熟悉系統架構、金融、法律以及資訊科學知識的專職人員所面臨的問題並在此章節說明其技術背景。

2.1.1 區塊鏈 （Blockchain）

自網路問世以來，區塊鏈被形容為最了不起的發明，甚至有「昨日網際網路，今日區塊鏈」這樣的句子來形容區塊鏈將為生活所帶來的引響。2008 年 9 月雷曼兄弟公司申請破產不僅僅創下了美國史上最大金額的破產案，另外也造成了金融大海嘯的開始。雷曼兄弟公司從創建至申請破產時擁有 158 年的歷史，且在 2008 年還曾獲美國《財富雜誌》選為財富 500 前的公司之一，表示該公司在全美的營業額總計排名於前五百名之一，但就算是如此規模龐大且歷史悠久的公司，卻還是在次級房貸風暴的波及以及下申請破產了，而正因為此公司的規模宏大，在各個金融機構的業務系統盤根錯節，有密不可分的關係，故才引發了如此嚴重的金融風暴。

而中本聰正是於雷曼兄弟破產的後幾個月發表了比特幣的白皮書，而比特幣這種去中心化、全球可使用支付的電子加密貨幣正巧解決了交易過程中信任的問題。交易

DOI:10.6814/THE.NCCU.CS.001.2018.B02

過程中信任的問題一般而言不是這麼淺顯易懂，舉一個實際的範例來說，小王和小美是同窗十年的好友，有一天小王想和小美相約，兩人各押金 100 元，並約定好在三天後若下雨則小美獲得所有押金，反之小王獲得，那麼可能發生以下三種狀況：

1. 小王和小美信賴彼此，故沒有簽下任何條款的狀況下便開始了賭局。一般來說，如果是朋友這是一個很簡單的交易方式，三天後若下雨小王應給予小美 100 元，那麼這筆交易終止。然而，即便是朋友，也有可能會發生翻臉不認帳或者小王既輸了賭局卻忘記自己到底是壓下雨與否，私自拿取小美錢包現金等事發生，那麼小美則有可能不只是損失了 100 元的獲利，而且更失去了小美與小王原本信賴的關係。
2. 小王和小美定義好完整合約，並且雙方也簽署了賠償條款。但倘若三天過後果真下雨，小王仍舊翻臉不認帳，則小美就必須依法告訴，而小美又不懂得如何打法律官司，則必須花錢請律師協助，藉由法律途徑取得借款則既花時間且又必須花額外金費，成效不彰。
3. 最後小美與小王決定找一個中立的第三者，小林擔當中介，小美跟小王個別將 100 元押金交予小林，但三天過後若小林捲款潛逃，則不管下雨與否，小王與小美則個別損失了 100 元。

由以上範例可以看出，雷曼兄弟公司如同上述第三種可能，大部分的金融機構系統與雷曼兄弟公司盤根錯節，一旦雷曼公司兄弟申請破產，則資金流動則無法正常運行，造成金融大海嘯的起因。

比特幣底層的區塊鏈技術正是解決信任問題的重大發明，他使用了密碼學的基礎加密技術作為身份驗證機制，在區塊鏈系統中每個帳戶會有公鑰（Public Key）以及私鑰（Private Key）的設計，任何一筆交易皆需要密碼解鎖私鑰並搭配公鑰方能解鎖，避免資金被他人竊取。舉例一個簡單的匯款交易來說，當小美要將 100 元賄款給小王，一般而言會有以下幾件事情發生。

DOI:10.6814/THE.NCCU.CS.001.2018.B02

小美撰寫了一個交易訊息，內容為將 100 元傳送給小王，並使用小美的私鑰解鎖自己的帳戶，取出 100 元放至訊息內並使用小王的公鑰加密（公鑰在區塊鏈中實際意義為使用者的帳戶號碼），區塊鏈會透過公鑰傳送給小王此封訊息。當小王收到訊息後，因區塊鏈公鑰為公開，則小王取得訊息後，區塊鏈會告知小王此封訊息由誰傳送，小王得知傳送者為小美後可小美的公鑰解開小美的公私鑰鎖，而小美先前使用小王公鑰加密的公私鑰小王則可使用自身的私鑰解鎖並取得訊息內所包含的金額。如此在傳送訊息時可透過加密技術，區塊鏈中資金的動用必須透過本人帳戶的公私鑰解鎖才能夠動用，達到第一項訊息傳遞的信任問題，沒有人可以隨意的動用自己帳戶內的資金流動。

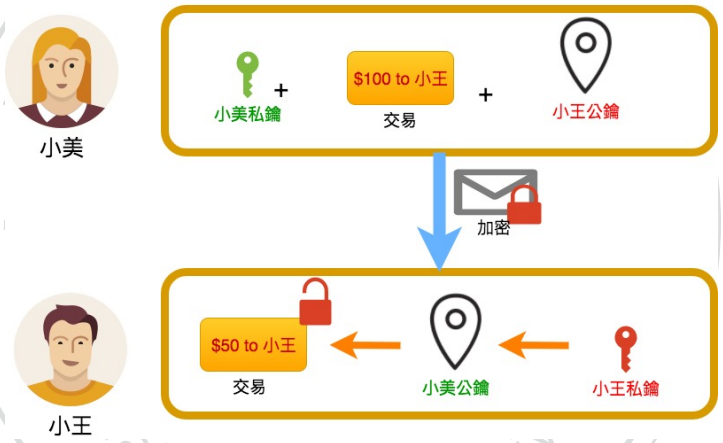

圖 1: 區塊鏈交易過程

而比特幣的區塊鏈平台上存在一種名為智能合約的物件，實際上為程式碼，用來輔佐區塊鏈交易訊行的自動化，例如上述小王及小美賭局的例子，小王與小美可撰寫一個智能合約，個別將 100 元押金放入至其撰寫好的智能合約中，三日後依據氣象快報，三日後不管下雨於否，智能合約的程式碼將透過外部資訊被觸發，並且自動將押金匯入正確的帳戶中。但因比特幣原系統的限制，每一筆交易的大小最大僅僅只有 1MB，比特幣平台中的智能合約無法做複雜的判斷，僅僅能夠做簡單的交易。數年後，於 2013 年 Vitalik Buterin 提出了《以太坊白皮書》在文內說明了以太坊讓開發者創建更具擴充性、易於開

DOI:10.6814/THE.NCCU.CS.001.2018.B02

發及應用的目標，並且在 2014 年透過 ICO 眾籌於並於 2015 年的 7 月正式上線，成為目前區塊鏈應用程式最常使用的區塊鏈平台。

在以太坊區塊鏈平台上的智能合約抽象架構如下圖所示，區塊鏈中的每一個參與者（Participabts）皆為一個節點，而每個節點們分別相互連結，形成一個部分連結網路拓樸（Partially Connected Mesh Topology）。每個節點將會將所有記錄的交易記錄在各自的帳本中，帳本實際上由一個接著一個的區塊（Block）鏈結而成，而區塊中又包含數個交易，每過一段時間，發出的交易會被打包成一個區塊，當有新的區塊被打包完成後，則會被廣播至所有相連的節點，並透過共識演算法進行驗證，驗證該區塊中的所有交易是否合法，以及是否安全，當通過後則會將該區塊串連至已被成功驗證為合法的區塊中，故每個節點將會各自擁有一份相同的帳本。

如圖 2 所示，在以太坊中，區塊中的交易（Transaction）可分為兩種，一種為一般金錢轉移的交易，而另一種則為智能合約，而以太坊區塊鏈平台提供每一筆交易內可擺放一些資料，智能合約撰寫完成並被成功編譯後會將其位元碼（Byte code）放入至交易中的資料欄位，當區塊成功的串連至前一個區塊後將會回傳該智能合約在區塊鏈網路中的帳號位址，故在以太坊中智能合約又被稱為內部帳戶（Internal Account），因智能合約存在於區塊鏈內部且不須透過公私鑰解鎖，但智能合約卻也有帳戶位置，可以存放資金至帳戶中，其存取資金的方式則仰賴於開發人員在撰寫程式碼時的設計。

DOI:10.6814/THE.NCCU.CS.001.2018.B02

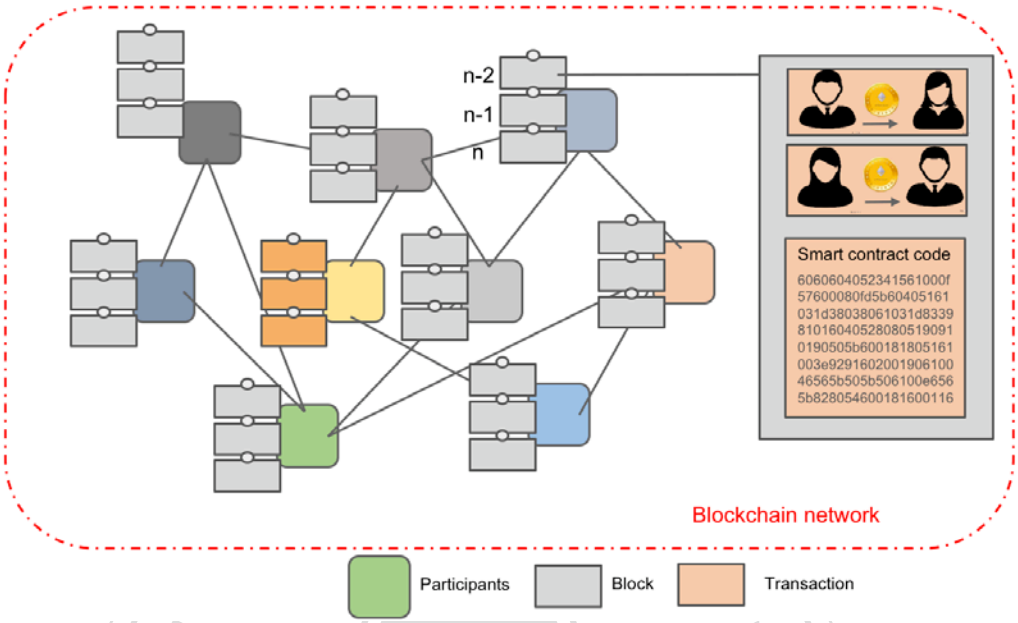

圖 2: 區塊鏈網路架構

區塊鏈內智能合約的存在，相當於把上述小美與小王的賭局透過智能合約，數位化成程式碼，並將其放入至區塊鏈平台中，當時間一到，由外部系統觸發智能合約判定結果的程式碼，便可以自動將資金移動至正確的帳戶中，解決了信任中的第二個問題，將合約數位化，當需要合約判定時該程式可以自動執行正確的資金流動，一旦合約數位化可以透過程式碼自動執行時，則可以大大減低需要外部人力介入的問題。

其區塊鏈系統之所以稱為區塊鏈實際上是因為區塊鏈系統其實是由一個個區塊串接而成，如下圖所示，每一個區塊內將會記載前一個區塊的號碼且其內部的交易將會形成一個樹狀結構，如此區塊鏈因各個節點個別記載相同的區塊鏈結的資料，又因為其鏈結關係，故其擁有公開透明且可追朔的特性。

DOI:10.6814/THE.NCCU.CS.001.2018.B02

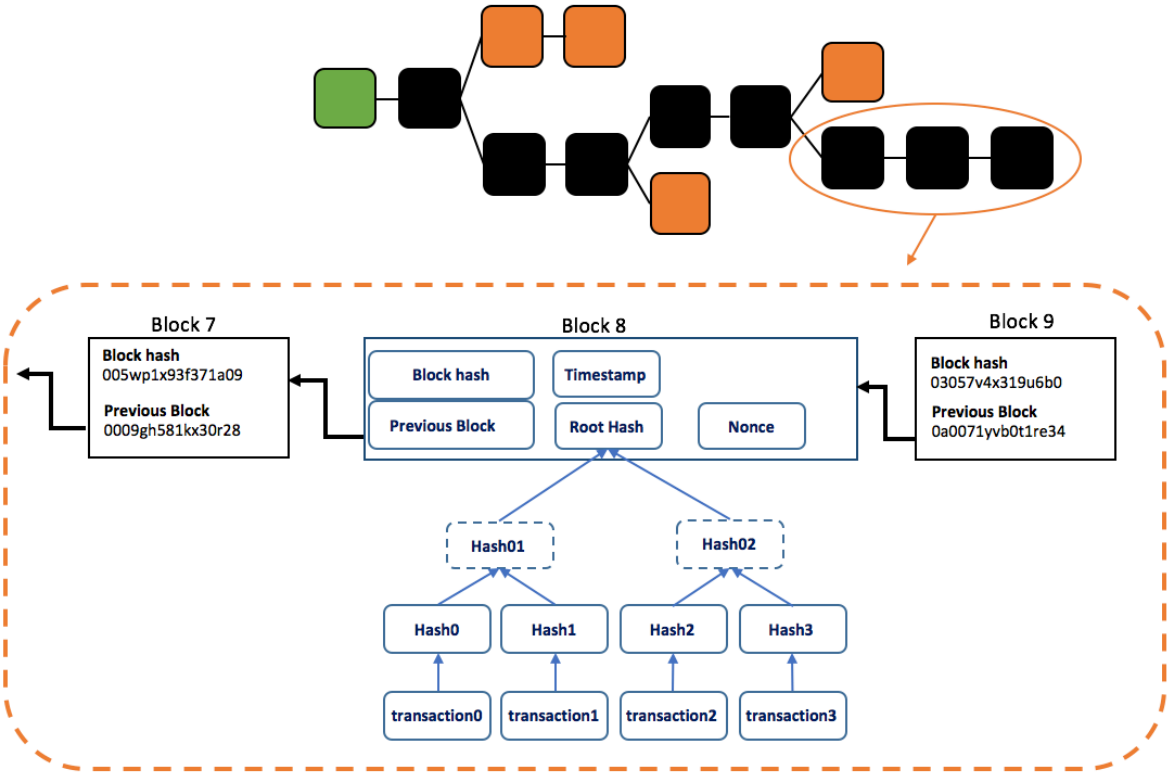

圖 3: 區塊架構

這樣的特性又解決了上述小美與小王的替三個問題，原本第三方可能會有捲款潛逃亦或者向雷曼兄弟公司那樣破產的可能，透過區塊鏈上智能合約以及可追朔、公開、透明且分散式的特性，儘管區塊鏈上有任何一個節點中途離開，都不至於影響整體區塊鏈的運作，因各個節點皆記載相同的帳本資料，交易可以不用仰賴第三方且又可以安全地進行交易。

智能合約大大的提升了複雜交易的效率，但欲開發智能合約，必須深度了解區塊鏈知識包含如何參與一個區塊鏈系統以及其運作原理為何，且當了解了區塊鏈的科學知識後，又必須擁有基本的金融知識，當開發一應用系統時，必須正確地將實體合約數位化，將該合約成功部署至區塊鏈平台後，必須學習要如何觸發合約的運行，須經過嚴謹的測試，確認其智能合約正確且完整的將實體合約數位化後，才能確定該區塊鏈應用程式的合約能依預期正常運行，故本論文提出了使用行為驅動開發(Behavior-driven development)

DOI:10.6814/THE.NCCU.CS.001.2018.B02

的方法提供一開發且測試智能合約的平台，解決目前缺乏區塊鏈專業開發人員撰寫智能合約、缺乏易用的智能合約開發平台問題。

2.1.2 測試驅動開發（Test-Driven Development, TDD）

在早期，開發圖靈機（Turing machine）程式，大部分的開發流程為：放入輸入磁帶(Tape)，並且準備一個預期的結果磁帶（標準答案），然後不斷的修改程式碼，直到其結果與預期完全相符則完成。而此方法正為 TDD 的前身測試先行（Test-first development,TFD），Kent Beck 在撰寫第一個 xUnit framework 時開始研究此方法，並且於 2003 年發表了 TDD 的概念[20]，被視為發展這項技術的元老或”rediscovered”的人。

整體來說，TDD 為 TFD 的改良版，其核心概念為在撰寫應用程式時先撰寫測試並專注於將各個功能（functionality）盡量切小撰寫測試，完成測試程式的設計與撰寫後僅撰寫可以通過測試的應用程式程式碼，當所有的測試皆通過後，更重要的是重構應用程式程式碼，已確保應用程式的品質（Quality），檢查各個功能內的程式碼是不是開發者實現功能的最好設計，並且沒有重複撰寫相同的功能，若不是則重構（Refactor）並且再次測試直到通過。

一般而言，會透過 Red-Green-Refactor cycle 來表示 TDD 流程，Red 表示一開始需先撰寫的程式碼必定會失敗，因為目標程式碼尚未撰寫，Green 表示撰寫足夠的目標程式碼使其通過，最後 Refactor 則是重構。

DOI:10.6814/THE.NCCU.CS.001.2018.B02

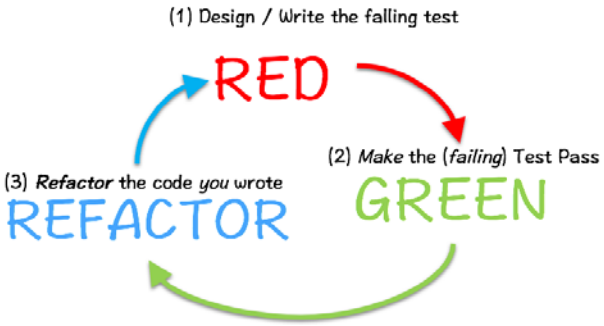

圖 4: Red-Green-Refavtor cycle

使用 TDD 作為開發方法的優點除了可以透過設計測試程式，清晰的了解功能需求，並且以通過測試為目標，使開發者更專注於達成功能的撰寫，最後可以透過不斷的重構獲得更佳品質的程式碼。

TDD 這種開發方法雖然可以透過設計測試更清晰地了解功能需求，但是在區塊鏈應用程式中，參與合作的人員一般會包含金融背景的專業人員，而這些人恰好大部分是沒有撰寫程式的經驗，若在一開始釐清商業流程的時候就出現誤解的問題，業務單位又看不懂測試程式，使用 TDD 這種方法並無法確保所開發之應用程式是否完全符合客戶需求。

2.1.3 行為驅動開發（Behavior-driven development, BDD）

軟體開發工程要完成的目標大致可分為兩類，其第一為”Build the software right”，意為確保所開發的程式執行面是正確的、品質優良的、建構正常運作的程式；其二為”Build the right software”，表示所產出的程式碼應符合業務需求、規格書、商業需求等規格面[7]。

BDD 可以說是 TDD 的擴展，因此種方法是基於 TDD 開發方法程式衍生而來。TDD 主張在寫程式前先設計及撰寫各個單元測試，接續進行撰寫足夠程式碼以及重構的動作，透過不斷地重構，改善所產出的程式碼品質，以及不斷地通過測試確保所產出的程式碼

DOI:10.6814/THE.NCCU.CS.001.2018.B02

必定可執行，因此 TDD 可解決上述軟體工程可執行面的確保，而 BDD 是架構在 TDD 迴圈之上，故 BDD 亦可確保程式執行面正確無誤。

而在需求面的部分，以傳統的大型專案來說，客戶會與系統分析師討論他們希望達到的目標，接著系統分析師將會撰寫系統分析文件，可能是使用 word 或任何的其他文書編輯程式撰寫，而系統開發者則依照系統分析師所撰寫的系統分析文件撰寫軟體，同時測試工程師也依據系統分析文件撰寫成測試案例，另外還需將軟體的維護方法撰寫成技術文件，當一切都正確無誤時才算開發完成。然而，就算在開發系統時開發者與測試工程師皆使用了 TDD 技術，這樣的開發方法仍會面臨幾項問題：

1. 眾多參與人員容易發生資訊不同步，可能會有資訊遺失的問題。
2. 在溝通時因專長的不同，非技術人員因不了解技術限制，有可能會答應客戶目前技術無法做到的需求。
3. 文件的傳遞有可能會遺失，一旦技術文件遺失，則系統的維護則需要耗費龐大的成本。
4. 在多次維護後，一旦忘記更新技術文件，將會造成文件與系統不一致，系統維護仍需耗費大量成本。

而 BDD 開發方法的核心概念除了建構在 TDD 之上，確保執行面正確外，透過專案參與者盡可能的共同討論及使用含自然語言撰寫可執行的需求規格書，則可確認所開發的程式符合定義的需求。

具體來說，如圖 5 所示，在寫程式碼之前軟體開發者與非技術人員將共同定義問題，接著共同使用自然語言寫作可執行的規格書，一種常見的做法為使用 Gherkin 語言撰寫規格書，而此規格書將會經由正規表示式轉換成可執行的情境測試（Step Definition）。當然一開始執行情境測試的結果必定會失敗，接著就是進入相對應的 TDD 迴圈，透過 TDD 迴圈確保單元測試全部通過後，再透過 Step Definition 測試程式接下待測系統，並藉由讓待測系統通過 BDD 測試，系統將逐漸符合可執行規格書。

DOI:10.6814/THE.NCCU.CS.001.2018.B02

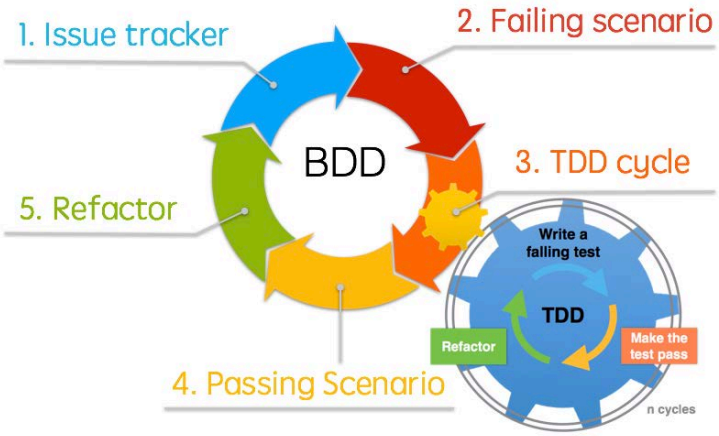

圖 5: BDD 結合 TDD 開發流程

僅使用 TDD 方法開發區塊鏈應用程式可能會發生在釐清商業流程時發生誤解的情形，而 BDD 這種將自然語言轉換成可執行的規格書，主要有兩大優點，其一，因 BDD 方法為鼓勵專案人員密切地溝通，並且從頭到尾共同使用同一份規格書，大幅降低了溝通上的落差；其二為若需求發生變更，因變更的需求將會產生新的情境測試，BDD 開發方法必須確保其情境測試及單元測試全部通過才算完成，若變更需求後開發人員尚未撰寫其功能，則在情境測試執行時必定發生錯誤，無法通過所有的情境測試，則可達到程式碼與規格書必為同一版本。換句話說，像 Gherkin 這種可執行規格又被稱為活文件(Living document)，指的是文件與程式碼隨時同步的規格書。在一個大型專案的實驗研究顯示[10]藉由結合 BDD 與 TDD 的優勢，可有效降低專案技術債(Technical Debt)。

一個可執行的規格書一般而言可拆分成兩個層次，規格層（specification layer）以及自動化執行層（Automation layer），而規格層事實上又由三項元件所組成：驗收標準（Acceptance criteria）、情境（Scenario）、通用語言（The ubiquitous language），自動行層則由模擬應用環境(Simulates a working application)、執行驗收測試(Runs acceptance tests）、不停的給予回饋（Uses feedback loops）所組成。

DOI:10.6814/THE.NCCU.CS.001.2018.B02

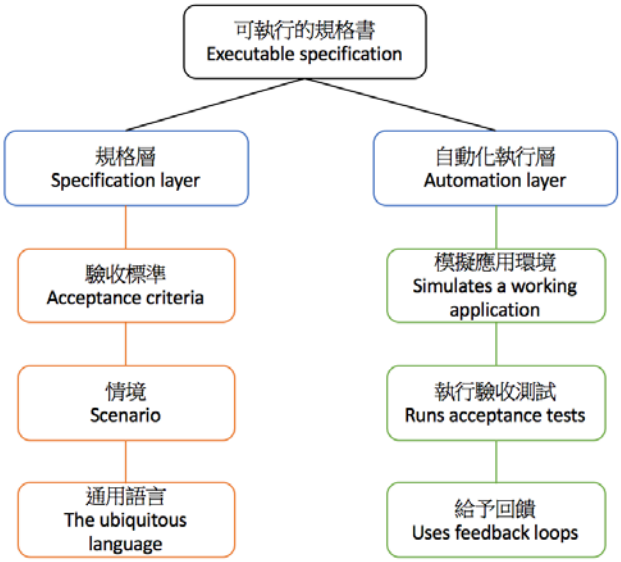

圖 6: 可執行的規格書之組成

以 Gherkin 撰寫規格為例，透過使用 Given-When-Then 的語法來規定驗收標準的流程，並使用這種標準撰寫情境，且在撰寫情境時亦可使用日常語言來敘述條件狀況，故這種既有一定規定但非技術人員亦可協同撰寫的格式，可大幅減輕在溝通上的誤解。而撰寫完規格後，執行層會產出一個應用時的環境，並且產出驗收測試的框架，當執行測試時會給予目前執行狀況的回饋。

整體來說，BDD 方法是由開發人員與業務領域專家針對業務邏輯進行深入溝通開始。對於一個將要開發的系統，開發人員與業務領域專家必須先訂出一至多個該系統所應具有的功能特徵(Features)。功能特徵通常會以敏捷開發社群常使用的特徵注入法(Feature Injection) 來寫作[11]。此種方法通常包含三步驟:

Hunt the value;

Inject the feature;

spot the example。

實務上的做法很多，其中，[12]所提倡的範本經常被採用，在這個範本中，每個 Feature 會以三段方式呈現:

DOI:10.6814/THE.NCCU.CS.001.2018.B02

In order to <meet some goal>

As a <type of stakeholder>

I want <a feature>

透過這種描述方法，可以很容易得知此需求文件的目標(In order to)、描述者角度(as a)與具體來說為協助使用者達成該目標或者此軟體應有的功能。Gherkin 是由 Wynne 與 Hellesoy [12]提出，用來做為 feature injection 的領域專屬語言。它支援了 60 種以上不同國家的語言，可以在 Cucumber 工具上自動執行多種程式語言的驗證[12]。

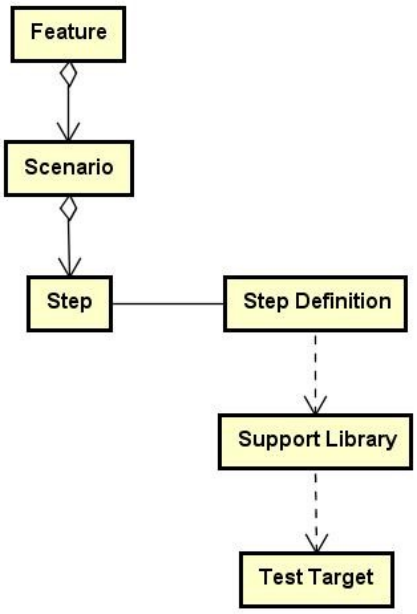

圖 7: BDD 重要元素概念模型

圖 7 為在使用 BDD 方法所使用到的重要元素所產生個概念模型，通常一個 Feature 中會包含多個 Scenarios，而一個 Scenario 會包含多個 Steps。一般來說 Scenario 會以結構化且符合易讀原則的文字來描述「就業務邏輯的角度來看，合約正確運作時應有的行為」。

DOI:10.6814/THE.NCCU.CS.001.2018.B02

Step 通常會以 Given、When、Then、And...來表示在什麼前提與環境(Context)下根據業務邏輯預期應該要有什麼結果。

Feature、Scenario 與 Step 構成需求文件的主要部份，而 Step 的常用格式如下:

Given <a context> And <another context>

When <an event is triggered>

And <another event>...

Then <outcome>

And <another outcome>...

從上述格式可觀察到，此種格式具有如同程式碼的結構化樣態，但又不如程式碼嚴格(因為它的部份內文允許自由文字格式)。情境定義完成後，由情境引導開發者開發軟體，並將情境腳本撰寫成自動測試程式，這種可自動被執行、測試的情境腳本又稱可執行規格(runnable specification) [7, 8]。

DOI:10.6814/THE.NCCU.CS.001.2018.B02

第 3 章

系統設計

本章首先說明基於典型 BDD 加以擴充，適用於智能合約的開發流程加以介紹，接下來說明支援此開發流程的整合測試平台系統架構與設計。為提供開發人員進行自動化的整合測試開發環境，本論文將整合 Cucumber 整合測試工具、Mocha(單元測試工具)、Solc(智能合約編譯器)、Web3.js(合約佈署與管理工具)與 TestRPC(測試用合約容器)，建構成一個 Web based 自動整合測試平台 (圖 8)。

圖 8: BDD 整合測試平台系統架構

DOI:10.6814/THE.NCCU.CS.001.2018.B02

整個平台的建構可分為三層，最上層是以 Web 為基礎可供開發人員於線上使用的開發與整合測試環境，包含 Gherkin IDE, Step Defs IDE, Unit TestsIDE, Contract Management GUI, Process Controller 與 Admin Console 等六個子系統。

中間層則是架構在 Express.js 上，以 Restful Web API 的形式與介面層互動，提供開發服務、整合測試服務、單元測試服務與合約管理服務，這些服務介接了整合測試驅動框架 Cucumber、單元測試驅動框架 Mocha、合約編譯器 Solc、區塊鏈存取介面 Web3.js 與合約測試 Mock 容器 TestRPC 等服務。

需注意的是，Mocha 無法直接對智能合約進行測試。主要原因為二，首先，Mocha 只能針對 JavaScript 進行測試，而智能合約以 Solidity 開發，二者語言和執行平台不同。其次，智能合約必須佈署到區塊鏈上才能運行並被呼叫。針對上述第一個問題，本研究提出的解決方案是透過 Mocha 呼叫 Web3.js 間接呼叫智能合約。針對第二個問題，本研究使用 Ethereum 專案提供的 TestRPC 工具，TestRPC 是一個智能合約的虛擬容器，在測試時期，開發人員可在本機佈署合約到虛擬容器中進行測試，而不需要佈署到直正的區塊鏈。本研究將上述功能實作並封裝為 Contract Management Service，自動為開發人員處理這些測試工具、函式庫的連結，降低智能合約的開發測試負擔。

3.1 系統功能分析

智能合約的開發有其一定的複雜性，目前大部份的合約測試方式都是在寫作完成後，手動佈署到區塊鏈，並透過 browser-solidity 或自行寫作測試腳本，設定特定的 Gas，以遠端呼叫的方式進行呼叫，取得回應，再比對是否為預期值的方式驗證。即使熟練的合約開發人員可善用相關單元測試工具如 Mocha 的方式來組建自己的工具鏈，但在邏輯複雜的情況下，這些單元測試工具並不足以確保整體業務邏輯的正確性，即使每一個方法

DOI:10.6814/THE.NCCU.CS.001.2018.B02

都正確被實作與測試通過，並不代表每個單元組合後，整體能依照特定業務邏輯正確運行。

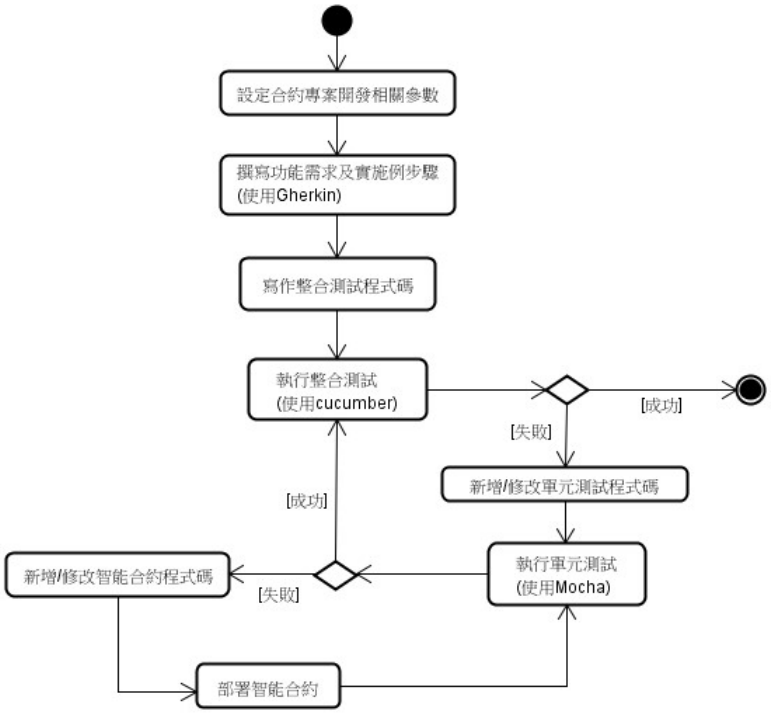

圖 9: 用於智能合約的 BDD 開發流程

為了提供合約開發者更快、更有用的回饋來降低測試及修復錯誤成本，達到有效率地測試智能合約是否符合商業目標(Business goal)並提升程式品質(Quality)的目標，因此，本研究基於 BDD 原有流程加以擴充，規劃適用於智能合約的 BDD 自動整合測試流程(圖 9)，詳細步驟說明如下:

1. 設定合約專案開發相關參數:

包含開發人員帳戶、keys 與 eNode 節點資訊等等。

2. 撰寫功能需求與 Scenario 步驟:

以 Gherkin 語言描述合約的功能規格。若僅使用自然語言撰寫功能規格，若撰寫功能規格書時使用了不明確的語意，仍然會有很大的可能會造成誤解。例如:客戶表示需要新增轉帳功能至系統中，但因為使用制定格式敘述需求，使得開發團隊並

DOI:10.6814/THE.NCCU.CS.001.2018.B02

不了解轉帳應該被觸發的時間、角色、規則為何。Gherkin 格式使用有結構化的自然語言敘述功能規格，每個情境(Senario)由多個步驟組成，每個步驟都以 Gherkin 關鍵字開頭，並提供具體範例做為測試資料。Gherkin 撰寫完成後整合測試將會以此為依據，分別將每一行切割為一個步驟(Step)，透過正規表示法(Regular expression)對應到整合測試程式碼並執行。

3. 寫作整合測試程式碼與執行整合測試:

寫作由 Gherkin 所定的 Step 對應的 Step Definitions。當整合測試軟體執行時，將依序執行測試 Step Definitions，並視情況將提供的測試案例透過參數代入測試程式碼中，自動測試程式執行完畢時，依測試結果給予回饋。

4. 新增/修改單元測試程式碼與進行單元測試:

為了讓整合測試通過，必須新增新的合約與方法，在開發時，使用 TDD 方式，先寫單元測試程式，再開發合約。

5. 開發與佈署合約:

撰寫單元測試程式後，為了讓單元測試通過，因此開發智能合約設法使智能合約通過測試。智能合約撰寫完成後，需先進行編譯，並產生 ABI/BIN 檔案後，將智能合約部屬至區塊鏈環境中。

3.2 狀態引導

本論文利用 BDD 方法作為智能合約的開發流程，因 BDD 事實上會有多種不同的狀態，舉例來說，當驗收測試發生錯誤並且修改完程式碼後，應該再次執行單元測試，確定通過後再進行驗收測試，也就是說開發者必須要熟悉 BDD 方法才能理解下一步應該進行何種步驟，更精確地來說，若非熟知 BDD 方法的人使用 BDD 方法開發專案，其需額外耗費一些時間。

DOI:10.6814/THE.NCCU.CS.001.2018.B02

Jakob Nielsen 於 1995 年提出了” 10 Usability Heuristics for User Interface Design”[21]，說明了十大原則去說明要如何改善軟體的易用程度，內容說到軟體應該在適當的時間內，給予使用者適當的回饋，讓使用者知道現在發生了什麼事情，且應該使用用戶熟悉的詞彙、圖像或概念作為回饋。因此，為了改善本平台在易用性（Usability）的程度，本研究設計了三種狀態引導至介面，讓使用者可有更好的使用體驗(圖 10)。

圖 10: 狀態引導

表示尚未進行該步驟 ：已撰寫，但發生錯誤需要修改 ：此步驟已完成

在本論文也提供 圖示指示使用者下一步應點選哪一個按鈕，畫面會自動跳至該步驟的 IDE 供使用者繼續開發專案。

3.3 樣板標記 (Skeketon Annotation)

BDD 開發方法一般而言是由正規表示法對應到整合測試碼，透過產生框架並且提供開發者依據測試框架撰寫測試程式並實作，而本論文利用 BDD 結合 TDD 的方法實作智能合約開發工具，首先會遇到幾個問題，其一是要用什麼語法撰寫測試程式碼，其二是要如何協助智能合約的開發，使一般未熟悉區塊練開發的開發人員更為容易入手。

智能合約是由 Ethereum 提供之特殊語法 Solidity 撰寫智能合約，因此，第一個考量為使用 Solidity 撰寫智能合約測試程式碼，但用此方法目前則會遭遇到很大的困難。舉例來說，在撰寫智能合約時必須在 Solidity 程式碼中撰寫版本等資訊，並且需熟知 Solidity 語法風格，且目前 Solidity 語法仍不支持字串陣列的回傳等，且當錯誤發生時，區塊鏈系統不會給予有用的錯誤回饋，目前 Ethereum 提供的錯誤回饋通常只會給予失敗及成功兩種為簣。故測試程式碼的語言本研究選擇使用 javascript 作為測試程式之語法，提

DOI:10.6814/THE.NCCU.CS.001.2018.B02

供更彈性的測試環境來解決 Solidity 語法尚不成熟所帶來的挑戰。另外，本研究使用 BDD 結合 TDD 作為開發方法協助開發人員可以更容易的開發目標程式，使用此種方法會產出程式碼框架，開發人員可以利用填空的方式撰寫測試程式或智能合約內容。在這種做法中本研究提供一種樣板標記，使得開發人員在撰寫測試程式碼時，使用本論文提出之工具預先產出下一步驟程式碼之框架，開發人員僅需要用填空的方式將內容填入即可(圖 11)。

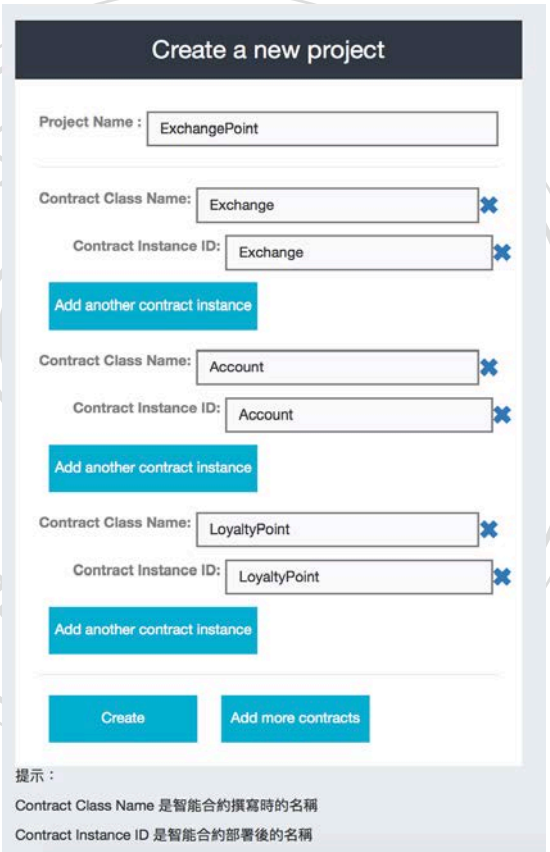

圖 11: 創建專案

在創建專案時本論文提供更友善的介面，供創建者理解目標程式碼的物件為何，並且預先產出 javascript 在呼叫智能合約時所需要的基礎元件，例如智能合約創建後的位址、應用程式二維介面（application binary interface,abi）以及位元碼（Bytecode）如圖 12。

DOI:10.6814/THE.NCCU.CS.001.2018.B02

```
let Account_abi = []
let Exchange_abi = []
let LoyaltyPoint_abi = []
let Account_bytecode
let Exchange_bytecode
let LoyaltyPoint_bytecode

let AccountA_contract = new web3.eth.Contract(Account_abi)
let AccountA_address
let AccountB_contract = new web3.eth.Contract(Account_abi)
let AccountB_address

let Exchange_contract = new web3.eth.Contract(Exchange_abi)
let Exchange_address

let LoyaltyPointA_contract = new web3.eth.Contract(LoyaltyPoint_abi)
let LoyaltyPointA_address
let LoyaltyPointB_contract = new web3.eth.Contract(LoyaltyPoint_abi)
let LoyaltyPointB_address
```

圖 12: JavaScript 呼叫智能合約基礎元件

而撰寫整合測試時本研究提供樣板標記語法(圖 13)，使用類似 meta-annotation 的方式提供開發者將方法的框架寫入設計之註解標記中，例如“///@method“表示樣板標記前置詞，利用目前最常使用的 JSON 格式作為物件判別，其中 contract 表示智能合約名稱，而 name 表示其方法名稱，以及 argument 表示該方法所用到的參數個數。

```
///@method {"contract":<contract_name>, "name":<contract_method_name>,"argument":<method_argument_number>}
```

圖 13: 整合測試框樣板標記

以圖 14 為例，在此範例中開發者在撰寫整合測試時，設計了“addPoint“方法，並且此方法的參數個數為“1“，開發者可在此透過註解標記協助撰寫框架，表示該方法存在於”LoaltyPointAA”智能合約中，並且此方法參數個數為”1”。

```
Given('original alp account of A is {int}', function (int, callback) {
    AccountA_contract.methods.getLocalLoyaltyPoint().call().then((address)=>{
        LoyaltyPointAA_address = address
        LoyaltyPointAA_contract.options.address = LoyaltyPointAA_address

        ///@method {"contract":"LoyaltyPointAA","name":"addPoints","argument":1}
        //增加第一家公司的點數
        LoyaltyPointAA_contract.methods.addPoints(int).send({
            from: web3.eth.defaultAccount,
            gas: 44444444
        }, function(error, transactionHash){
            callback(assert.equal(error,null,error))
        });

    })
});
```

圖 14: 整合測試框樣板標記範例

DOI:10.6814/THE.NCCU.CS.001.2018.B02

有了整合測試的註解標記，本平台會在第一次執行單元測試時偵測此註解，即平台會產出該方法的單元測試框架（圖 16），而開發者可以在設計單元測試時設計再該方法是否會改變區塊鏈內原本資料的狀態（圖 15），若會則在“isConstant”的部分則設定為”true”。

```
///@method {"contract":<contract_name>, "name":<contract_method_name>,"argument":<method_argument_number>,"isConstant":<method_constant_type>}
```

圖 15:單元測試樣板標記

```
describe('Successfully Use addPoints(arg1)', function(){
    it('should work', function(done){
        ///@method {"contract":"LoyaltyPointAA","name":"addPoints","argument":1}
        LoyaltyPointAA_contract.methods.addPoints(arg1)
        done()
    })
})
```

圖 16: 單元測試樣板標記範例

最後，單元測試撰寫完畢後，平台會依照樣板標記提供的內容產出智能合約程式碼框架，開發者可以依照此框架來撰寫智能合約，並繼續執行 BDD 結合 TDD 開發方法。

3.4 生成帶入

Pattern-Oriented Software Architecture Volume 4（POSA 4）[22]提到了分散式系統的主要設計模式，其中說明到當 client side 要遠端存取服務時則必須透過特別的資料格式（data format）以及網路協議（networking protocol）才能達成，一種做法是將元件的資料格式化以及協議處理都交給發佈的 client side 負責，但這樣的做法會造成過度依賴該 client side，若其他 client side 遠端呼叫時該 client side 沒有解碼好該資料及處理協議，或者甚至中途離開了，則該元件便無法使用。

另一種理想的方式為將元件獨立化（location-independent），使得遠端呼叫與本地呼叫時在呼叫上無差異存在，其作法為發佈者提供一個 client proxy 且定義 client proxy

DOI:10.6814/THE.NCCU.CS.001.2018.B02

的使用介面（Introspective Interface），故當發佈者把 client proxy 發佈至分散式系統後，遠端或本地端在做存取時可以透過使用介面對被發佈的元件做存取。

而在區塊鏈系統中，智能合約撰寫完成後會將智能合約程式碼編譯成二維碼（Bytecode），後將此二維碼擺放至區塊鏈中，當廣播至各個節點的時候，各節點可直接執行二維碼，不需要重新編譯後才能執行而浪費節點們的資源，但當智能合約已將二維碼擺放至區塊鏈時，若要使用智能合約上的方法而僅有二維碼卻無法達成，其原因為當智能合約已被編譯成二維碼，若要呼叫方法時必須透過應用程式二維介面（application binary interface,ABI）將其解碼後，將參數帶入重新編碼再執行。也就是說，欲呼叫一智能合約方法，除了智能合約位址外，還必須要擁有該智能合約的 Bytecide 以及 ABI 才可進行呼叫。

一個簡單的智能合約範例如圖 17 所示，此合約包含了一個建構子與一個方法，當合約建立時指定所有人為誰，且提供原持有人一個轉移持有人權限的方法。

```
pragma solidity ^0.4.11;

contract Ownable{
    address public owner;

    function Ownable() public {
        owner = msg.sender;
    }

    modifier onlyOwner(){
        if(msg.sender!= owner){
            revert();
        }
        _;
    }

    function transferOwnership(address newOwner) onlyOwner public{
        if (newOwner != address(0)){
            owner = newOwner;
        }
    }


}
```

圖 17: 智能合約範例

DOI:10.6814/THE.NCCU.CS.001.2018.B02

當撰寫完智能合約後開法者必須嘗試編譯此智能合約，而目前以太坊提供的智能合約編譯器為 solc，在編譯前開發者必須詳讀安裝以及使用方法，並且確保編譯的作業系統沒有限制，否則將會編譯失敗。

舉例來說，在 Windows 作業系統上，檔案命名規則不允許檔案名稱中包含 ":"，但若以上述智能合約為例，solc 在編譯時若 Solidity 檔名為"Contracts.sol"，則會產出 "Contracts:Ownable.bin"，而此種檔案名稱將會被拒絕，必須參考其他 solc 的設定文件使輸出檔名合乎作業系統檔案命名規則。

當撰寫完成智能合約並且成功編譯後會產出 Bytecode 及 ABI，Bytecode 為中間代碼，可透過直譯器轉譯後提供智能合約可以在虛擬機上運行，而 ABI 為智能合約的介面整體來說如圖 18 所示。

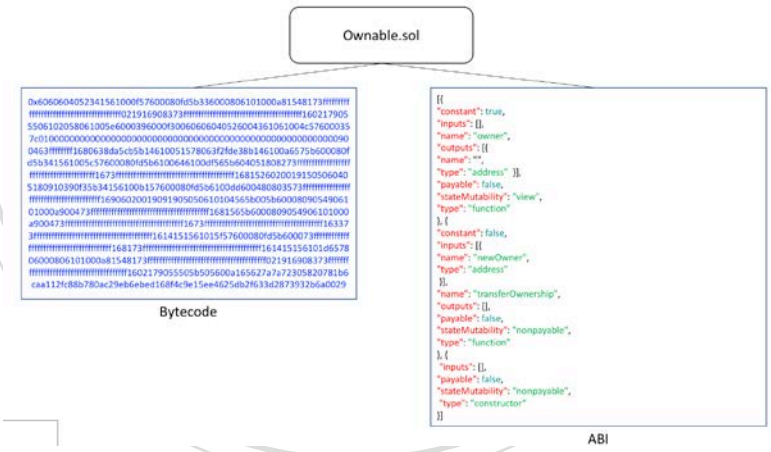

圖 18: 智能合約編譯完成後所產出之 Bytecode 及 ABI

因此，基於本論文是使用 javascript 語言撰寫測試程式碼的前提下，在基礎建設的部分除了協助開發者將基礎元件設立好外，另外還將編譯器整合至平台中當編譯完成後會將該智能合約之 Bytecode 及 ABI 帶入至測試程式碼中，減輕開發者的負擔。

DOI:10.6814/THE.NCCU.CS.001.2018.B02

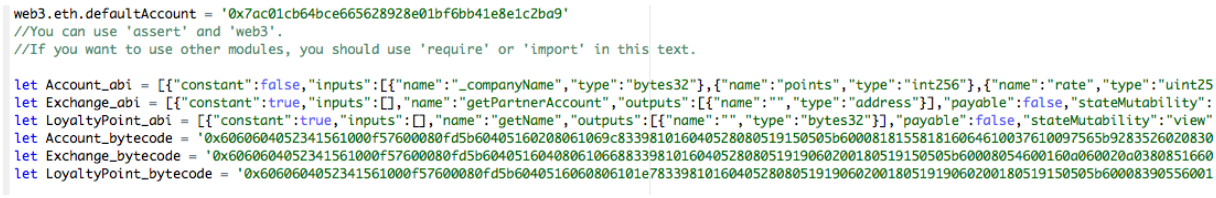

圖 19: 測試程式將 ABI 及 Bytecode 帶入

如圖 19 所示，當開發者依照 BDD 結合 TDD 開發方法，執行編譯後，平台會以自動帶入之形式將值帶入至物件，開發者可直接使用此物件。

3.5 Web Application Programming Interface (Web API)

一般來說，軟體是由不同的模組銜接而成，且顯而易見的，目前大多數服務皆是建立在 Web 上，為了提供其他平台可以不使用本平台提供之 IDE 亦可使用本平台提供的服務，本研究在中介層的部分設計了 Web API ，將服務包裝成 Web service 提供其他平台串接 BDD 方法開發智能合約，並且使用了 REST 風格將其封裝。

使用 RESTful 封裝服務可以讓要串接的平台減輕負擔，也就是串接的平台本身為 Client 端不需要花費太多額外的效能去執行服務，並且可以利用 chache 來達到更快的回應速度，且在各個作業平台皆可串接，減輕串接者的負擔。

為明確切割 UI 層及 Services 層邏輯，本論文將 UI 層執行的動作皆設計成可透過相對應的 Web API 達成。本論文將本平台提供的服務切割成三大類，分別為登入服務、專案管理、及自動化測試（圖 20）。

DOI:10.6814/THE.NCCU.CS.001.2018.B02

	說明	HTTP Method	URL
登入服務	登入(sign in)	GET	/api/sign_in?userID=xxxxx&password=xxxxx
	註冊(sign up)	POST	/api/sign_up
專案管理	查看現有projects	GET	/api/projects?apikey=APIKEY
	取得project內容	GET	/api/project/:project_name?apikey=APIKEY
	更新專案	PUT	/api/project/:project_name?apikey=APIKEY
	新增專案	POST	/api/project/:project_name?apikey=APIKEY
	刪除專案	DELETE	/api/project/:project_name?apikey=APIKEY
自動化測試	執行cucumber	GET	/api/run/cucumber/:project_name?apikey=APIKEY
	執行mocha	GET	/api/run/mocha/:project_name?apikey=APIKEY
	solidity compile	GET	/api/run/compile/:project_name?apikey=APIKEY

圖 20: Web API 服務

以下將分別使用四大情境說明 Web API 之使用方法，分別為註冊帳戶、專案管理包含新增後查看專案內容並進行專案修改，以及執行智能合約自動化整合測試（圖 21）。

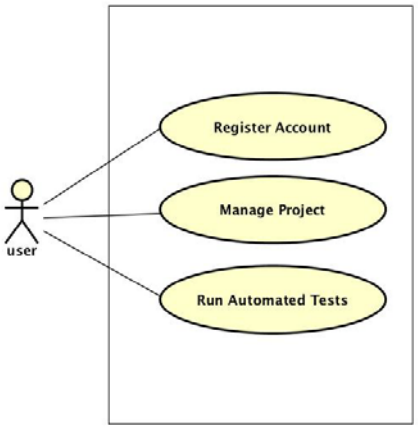

圖 21: Web API 使用案例

3.5.1 註冊帳戶

使用者可以透過登入服務內的註冊帳戶功能將使用者資料（包含使用者名稱及密碼等）發送給伺服器，當伺服器收到使用者申請註冊的訊息後會將資訊儲存至資料庫，並且透過 hash 產出一組 API key 回傳給使用者。

DOI:10.6814/THE.NCCU.CS.001.2018.B02

3.5.3 專案管理

使用者成功創建帳戶後可取得帳戶之 API Key，而創建專案時可透過此 API Key 創見多個專案，一般來說一個帳戶內會包含個專案，而創建專案時將會回傳成功與否的訊息，若成功創建專案則之後則可透過 API Key 搭配專案名稱管理專案狀態。

若需要取得帳號下的專案資訊時可以透過查詢專案的功能發訊請求，伺服器將以使用折送出之 API Key 進行查詢，回傳帳號內所包含之專案陣列，或者使用者亦可針對特定專案進行查詢，使用 API Key 搭配專案名稱查詢後取得最後一次上傳之 feature file (Gherkin)、Step Definitions、單元測試程式、智能合約程式碼。當要修改專案內容時使用者可以直接呼叫提供的 Web API 方法對專案進行修改或刪除。

3.5.4 自動化整合測試

而在進行以 BDD 方法開發智能合約的過程中，提供了可進行整合測試執行、單元測試執行、智能合約編譯功能進行呼叫的 Web API，在此過程中值得注意的是，若不使用本平台所提供之使用者介面進行開發而是使用 Web API 的形式進行開發，則開發者在進行呼叫時必須熟知 BDD 結合 TDD 流程，如此才能確保所開發出來之程式碼與預期完全相符合，否則仍有可能產生開完全但卻與預期不同之情況發生。

DOI:10.6814/THE.NCCU.CS.001.2018.B02

第 4 章

系統實作

本論文使用 Express 建構基於行為驅動開發製程的區塊鏈智能合約整合測試平台，Express 是一種由 Node.js 衍生出的開發框架，透過 Express 開發框架此研究將整個 Web 應用服務切割成 MVC 模式（Model–View–Controller）中的 View 以及 Controller 也就是本論文的 User Interface 及 Web API。利用 Express 內的 Routes 提供了 Web API 的服務，則透過此服務 User Interface 可呼叫對應的方法而建構出一智能合約開發平台，如圖 22。

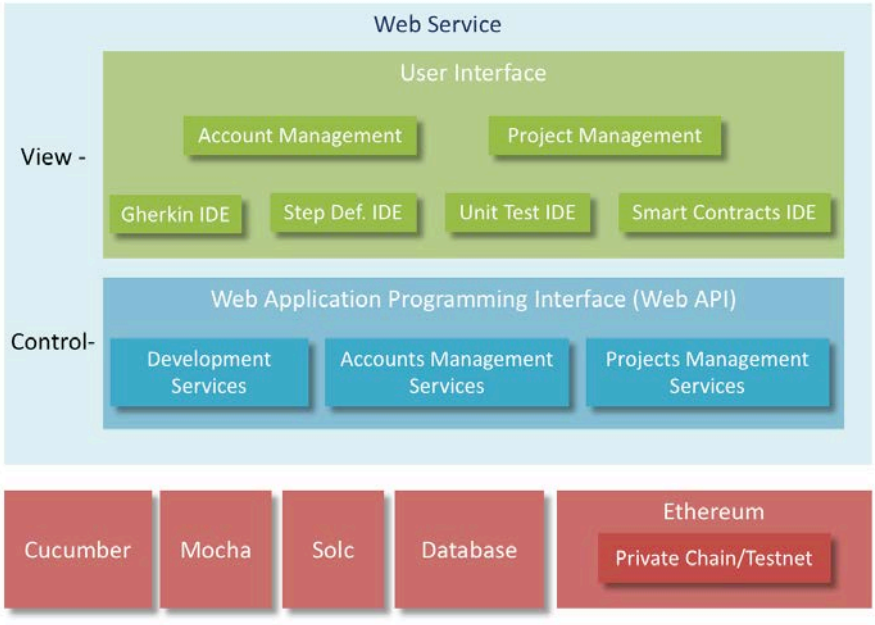

圖 22: 系統實作架構

DOI:10.6814/THE.NCCU.CS.001.2018.B02

4.1 帳號管理

本論文在專案管理部分依據金鑰概念設計，當使用者申請帳號時輸入了帳號及密碼申請帳號，則 Accounts Management Services 將會透過此組密碼及伺服器特定的密碼經過 SHA256 加密後產生一組唯一的 API Key 供使用者使用，故當使用者欲透過 Web API 直接使用 Account Manage Services 時可直接輸入該 API Key 即可使用，亦可避免在使用 Web API 直接開發時密碼顯示為明碼提高使用者帳戶安全性。如圖 23 所示，使用者在 Account Management 介面輸入了使用者帳號及密碼，則 Account Management Services 經過加密後即會產出一組 API Key，再將此 API Key 連同帳號密碼儲存至資料庫。當使用者註冊完帳號登入後，此平台則會再透過帳號與密碼從資料庫內傳回 API Key 進行其他呼叫。

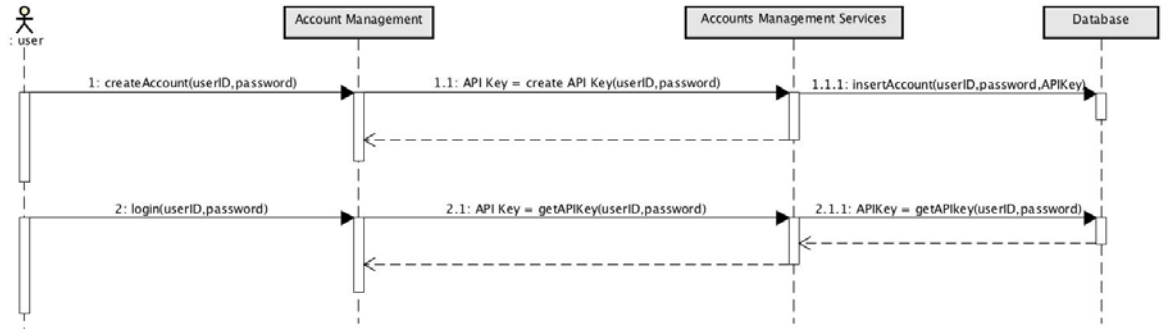

圖 23: 帳號管理

4.2 專案管理

當使用者透過 Account Management 介面登入後，Account Management 將換把 API Key 帶給 Project Management 或其他介面，因此，當要進行其他專案管理動作時，則不用再次查詢 API Key 即可使用 Web API 提供的其他服務。

DOI:10.6814/THE.NCCU.CS.001.2018.B02

4.2.1 查詢專案

查詢帳號下已創立的專案名稱時，透過前面登入後自動帶入至 Projects Management 的 API Key，對 Projecys Management Services 進行呼叫，此服務將會對資料庫針對帳號進行查詢，並且回傳一個專案名稱清單，在取得帳號名稱後，再次將要查詢的專案名稱加入參數內請 Projecys Management Services 對資料庫進行查詢，資料庫將回傳專案相關檔案，使用者即可獲得當前專案檔案，其回傳檔案包括了 feature File、Step Definition File、Unit Test File 以及 Smart Contracts File (如圖 24)。

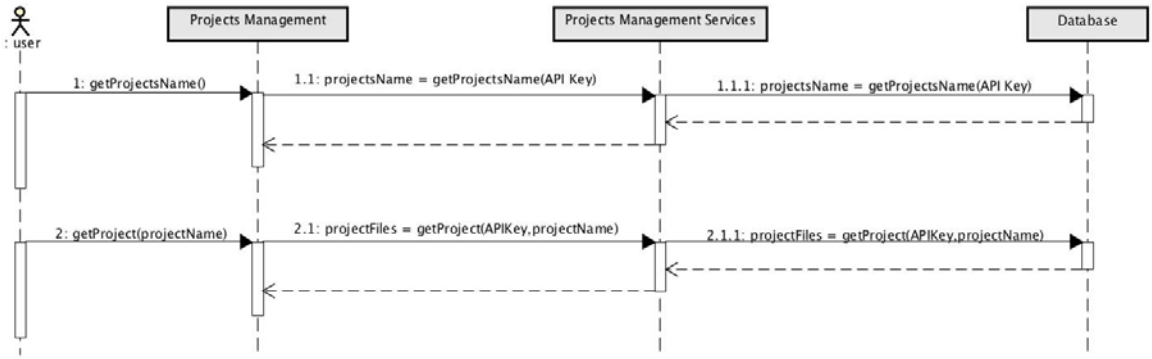

圖 24: 查詢專案清單及專案內容

4.2.2 新增專案

新增專案時 Projects Management Servicet 除了將專案加入至新增該帳號之專案外，還會透過使用者所填入的 contracts name 及 instance 生成出呼叫智能合約所需要的 javascript 預先加入至 Step Def. IDE 以及 Unit Test IDE 減少使用者在撰寫測試程式上所面臨的負擔(圖 25)。（如：address、instance 的宣告。）

DOI:10.6814/THE.NCCU.CS.001.2018.B02

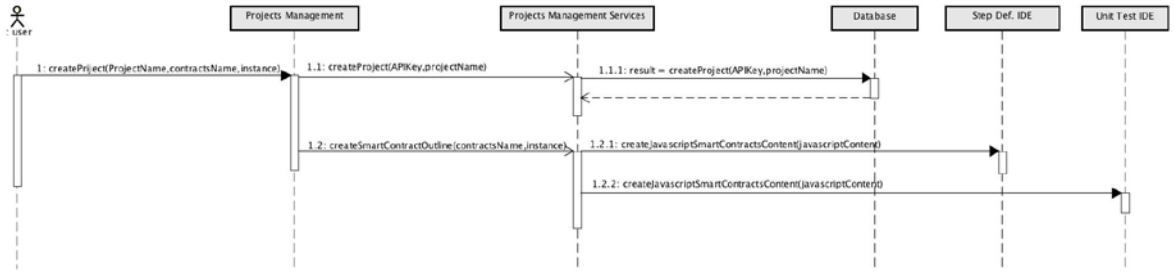

圖 25: 新增專案

4.2.2 修改與刪除專案

修改專案時 Project Management 將會把專案更新的內容一併交由 Projects Management Servicet 將資料庫內的資料覆蓋，並且傳回資料庫最後所處存的狀態，而刪除專案時則是直接給予 API Key 以及專案名稱，直接對資料庫進行刪除(圖 26)。

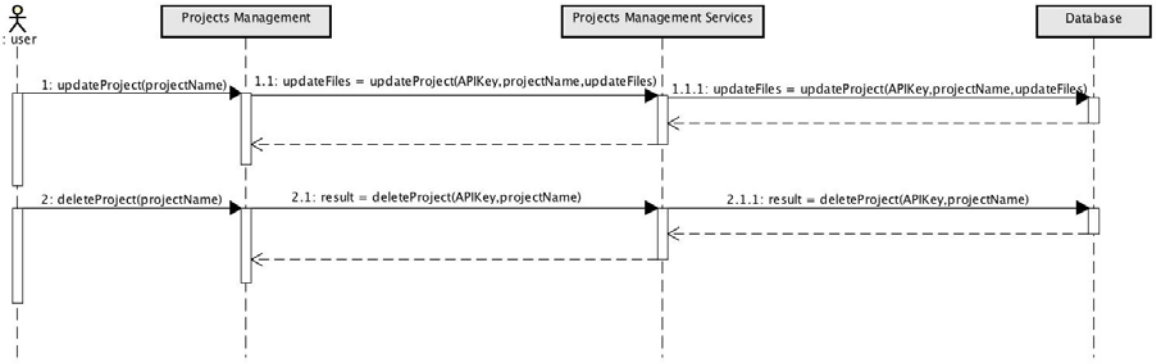

圖 26: 修改與刪除專案

4.2 規格寫作

DOI:10.6814/THE.NCCU.CS.001.2018.B02

當使用者撰寫完成 Gherkin 後，Development Services 會透過 Cucumber 產出的 Step Definition File 及呼叫智能合約所需要的 javascript 程式碼生成至 Step Def. IDE，使用者可依此為框架進行整合程式碼的撰寫，進而降低開發團隊了解框架規範所面臨的負擔(圖 27)。

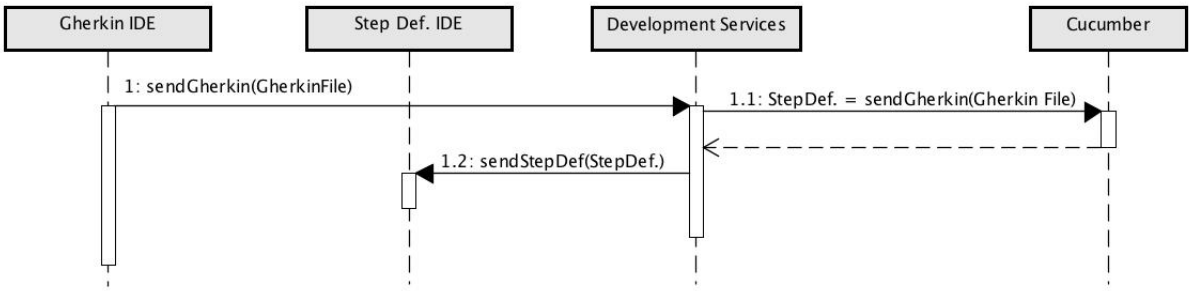

圖 27: 撰寫規格寫作

4.3 單元測試寫作

整合測試定義好每一個 Step 後，Development Services 將會根據 Step 內所撰寫的 Annotations 產出單元測試的框架，開發團隊依據框架並將每一個單元測試的測試目標物填入後，Development Services 則會再送給 Mocha 進行執行，並將結果傳回 Development Services(圖 28)。

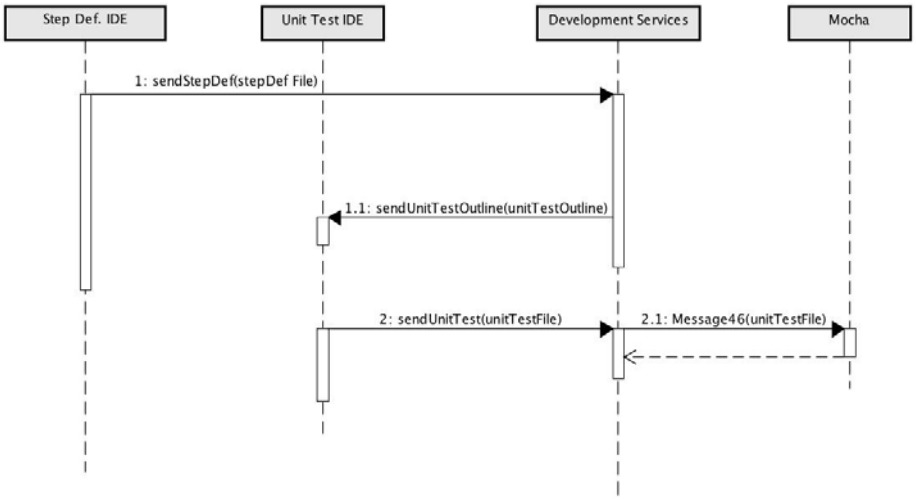

圖 28: 撰寫單元測試

DOI:10.6814/THE.NCCU.CS.001.2018.B02

4.4 合約寫作

當單元測試撰寫完成後 Development Services 會根據單元測試內的 Annotations 產出智能合約程式碼的框架，而開發團隊即可根據此框架撰寫程式碼，若程式碼撰寫完成，則 Development Service 會將智能合約送至 Solc 編譯，編譯完成後 Solc 會將結果（Byte code、ABI）傳回至 Development Service，而此時 Development Service 會將智能合約部署至區塊鏈網路上，部署成功後 Development Service 會將區塊鏈網路回傳的智能合約位置以及先前編譯的結果傳回至單元測試以及整合測試程式碼中，此時，即可執行測試（圖 29）。

圖 29: 撰寫智能合約並執行測試

DOI:10.6814/THE.NCCU.CS.001.2018.B02

第 5 章

系統評估

5.1 案例研討

為說明所提出與實作之 BDD 整合測試平台將如何運作，本章以「紅利點數交換」的其中一項功能為例，說明整個 BDD 整合測試平台將如何協助使用者開發與測試智能合約。

紅利點數交換為一個相度複雜度足夠的應用範例，在 Gherkin 的敘述中用到了大多數的 Gherkin 語法（Given、When、Then、And），且在此範例中設計開發了多個智能合約，並且一個智能合約對應到多個實例(instance)，使用此應用作為本篇論文的案例研討為複雜度相對足夠，且亦不會過於複雜以至於難以理解。

首先，開發人員先與客戶(Stakeholders)討論業務需求。從客戶的描述得知其中一項合約功能(Feature)為將點數於兩公司之間，以一定的規則與價值比率交換。此時，開發人員首先在平台上開啟了一個新專案，並在 Gherkin IDE(圖 8)中，以 Feature Injection 方式，基於 In order to /As /I want to 格式，寫作了如圖 30 的敘述。

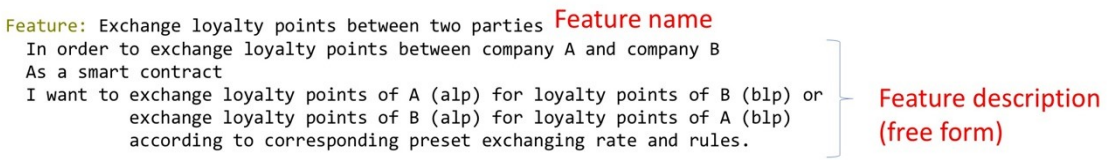

圖 30: Feature Injection

DOI:10.6814/THE.NCCU.CS.001.2018.B02

接下來，開發人員開始依據細部功能敘述，定義情境(Scenario)與完成此 Scenario 所需要進行的步驟(Step)，如圖 31。

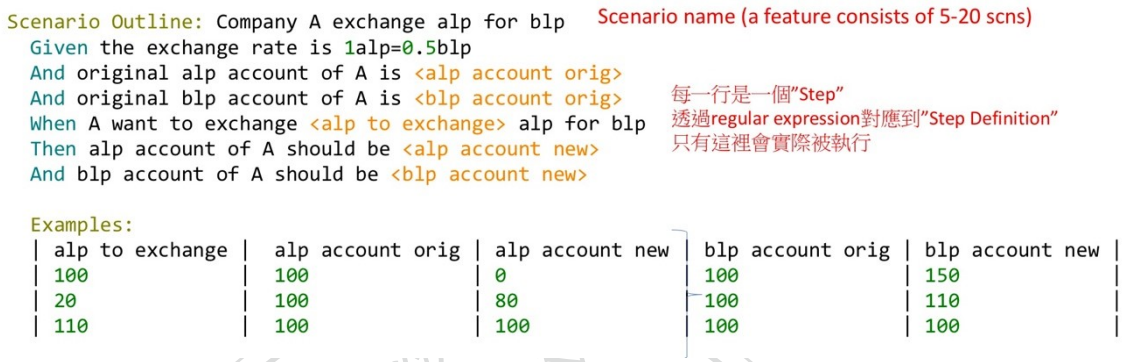

圖 31: Scenario

圖 31 描述了一個 A 公司要和 B 公司交換點數的 Scenario。可看出該 Scenario 包含了 6 個 Steps，在此無論 Given, And, When, 或 Then 都算是一個 Step。以 Company A exchange alp for blp 為例，意指將 A 公司點數(alp)交換成 B 公司點數(blp)。其第一個 Step 是 Given the exchange rate is 1 alp = 0.5 blp，意指此情境下的前提條件(Given)為 1 點 A 公司點數可換成 0.5 點 B 公司點數，第 2-3 個 Step 明確敘述了 A 公司與 B 公司之帳戶初始值。第 4-6 個 When Step 敘述了當 A 之交換條件發生(When)時，A 公司與 B 公司之帳戶餘額應有何種改變(Then)。在第 2-6 steps 都有以「<」與「>」括起來的變數，這些變數在執行時，會以下方表格中提供的實際數值(稱為 Example)逐一代入變數中進行測試。在 BDD 中，本研究稱有提 Example 的 Scenario 為 Scenario Outline。以圖 31 為例，由於有 3 組不同 examples 可代入 5 個包括「<」與「>」括起來變數的 Steps，因此共可產生 30 組不同的測試組合。以 BDD 的慣例來說，圖 30 和圖 31 的內容通常會寫在同一份文件中，以.feature 的副檔名儲存，而其內容這種半結構化的規格敘述方式即為 Gherkin 格式。

DOI:10.6814/THE.NCCU.CS.001.2018.B02

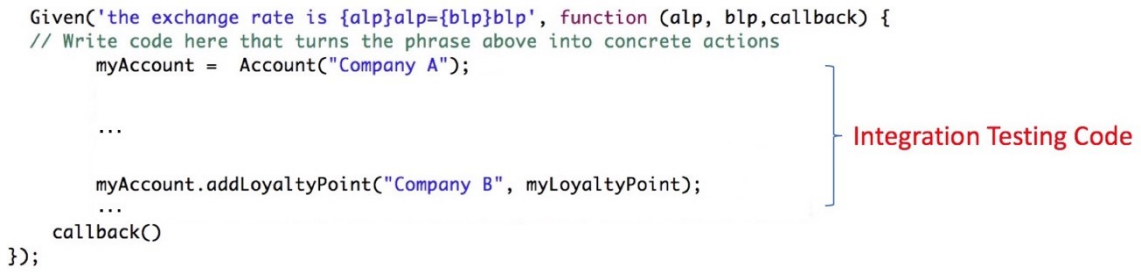

圖 32: 在 StepDefs IDE 中發整合測試程式

寫作完 Gherkin 後，必須為每一個 Step 定義一個 Step Definition，Step Definition 本質上為進行整合測試的程式碼(圖 32)，以 JavaScript 撰寫，且可在 Step Defs IDE 中開發(圖 32)。

目前 Step Defs IDE 中尚未寫作 任何程式碼，但仍然可以進行第一次整合測試，當然，初次整合測試因為尚未撰寫整合測試程式必會失 敗，此時整合測試工具中的 Integration Testing Service，它會呼叫底層的 Cucumber 進行測試) 會自動產生程式碼樣版到 Step Defs IDE，開發人員可以根據此樣板開始寫作整合測試程式。基於整合測試工具所產生的樣板，開發人員開始在 Step Defs IDE 中寫作整合測試程式碼。由於目前真正的智能合約程式碼完全都還沒有開發，所以直覺上，開發人員會開始以「完成此整合功能，並讓它們通過測試」的方式來思考寫出的程式碼，透過這種過程，開發人員會 發現需要開發一到多個智能合約模組，且每個智能合約要提供一些方法 (methods)，才能完成該整合測試。當開發人員開始寫作後，會發現若二家公司需要交換點數需要 A 公司和 B 公司都有一個 Account(帳戶)合約，來維護 A 公司、B 公司目前各存有多少 A 公司點數和 B 公司點數。雖然目前還沒實作 Account 合約細節，但以測試先行(Test-Driven)的原則，可以透過寫作測試程式來推論出實際的合約模組需要那些欄位與方法。完成後，開發人員可再度於 Step Defs IDE 中呼叫 Integration Testing Service 進行整 合測試，此時尚未實作 Account 合約，所以還是會失敗，因此接下來進到合約設計的層次。

DOI:10.6814/THE.NCCU.CS.001.2018.B02

開發合約一樣以 TDD 方式進行，平台提供的合約單元測試介面稱為 Unit Test IDE，提供開發人員線上寫作合約單元測試的場域。

```
describe('Account can add loyal point', function () {
        it('should add loyal points success', function (done) {
            myAccount.addLoyaltyPoint("Company A",myLoyaltyPoint,function (err , result)  {

                    ...
                    assert.strictEqual(result[0], 'Company A');
                    ...
                    done();
            })
        })
    })
```

圖 33: 在 UnitTests IDE 中撰寫單元測試程式

一個典型的合約單元測試範例如圖 33，由於開發人員發現 Account 中需要一個 addLoyaltyPoint 方法來為帳戶增加特定公司的紅利點數，因此在圖 33 中，先透過寫作單元測試程式碼來規範 addLoyaltyPoint 方法的行為，接下可進行一次單元測試(會導致失敗，因為尚未寫作合約)，下一步就是依此行為來實作合約(圖 34)。

```
contract Account{
    bytes32 private companyName;
    mapping (bytes32 => address) private loyaltyPoints;

    function Account(bytes32 _companyName) public {

     ...
    }

    function addLoyaltyPoint(bytes32 companyName, int points, uint rate) public returns( bool  ) {
     ...
       return true;

    }
```

圖 34: 以 Solidity 實作智能合約

實作完成智能合約後，透過 Contract Management Service 將智能合約部署至合約容器(TestRPC)之後執行單元測試，若通過將給予測試通過之回饋，否則將給予錯誤之參考資訊。若單元測試通過表示撰寫之智能合約所撰寫合約與其方法皆符合預期，此時可開發下一組方法，直到讓圖 32 中的整合測試可通過為止。在圖 32 通過測試後，接下來依

DOI:10.6814/THE.NCCU.CS.001.2018.B02

照類似流程繼續設法讓圖 31 中的其它 5 個 Step 的 Step Definitions 通過，即可完成此 feature 的開發。若所有 features 均開發完成，則智能合約便開發完成。

5.2 質性使用者測試

易用性測試（Usability Testing）是驗證產品是直覺和產品價值的方法之一[17]，有效的做可用性測試可以快速的更新設計師所設計的產品原型，且更快的讓使用者上手，並可以做到讓使用者更直覺的使用，引領設計師以使用者為中心發展設計。基於上述實作，根據[18] 本研究將找來 5 位使用者進行對於易用性（usability）相關的測試，對他們進行 BDD 及 TDD 相關的教育訓練，請他們先簡單填寫基本資料與閱讀實驗要求，經由受測者同意後，開始進行操作，受測時間約 60 分鐘，其中包含說明實驗與測試時間。實驗結束後再分別說明受測過程、受測結果分析和系統改善與修正等三小節。

5.2.1 受測過程說明

以「是否有智能合約程式撰寫經驗」為可變因子設計的可操作文件，內容將分為：「TDD 概要說明」、「BDD 說明」、「以紅利點數交換為範例說明」，漸進式根據操作文件進行專案開發，情境設計、整合測試撰寫、單元測試撰寫到最後的智能合約撰寫與使用 BDD 方法測試。引導受測者入門，同時搭配影片輔助說明開發方法，並設計一專案應用情境，讓受測者實際操作。

在做實際測試的時候會給予使用者使用情境，讓使用者更快進入狀況，並且測試結果可以更接近真實的使用情況，例如：想像現在要開發一個購物網站之購物車行為，目的為物品放入購物車，而其中的條件包含使用者所持有的折購利率（圖 35）。

DOI:10.6814/THE.NCCU.CS.001.2018.B02

```
# features/ShoppingCart.feature
Feature: ShoppingCart
  In order to do Shopping
  As a customer
  I want to manipulate the shopping cart
Scenario Outline: add Item with discount code
    Given a discount code is <code>
    And the item price is <price>
    When the shipping method applies to the item
    Then the discount price should should be <discount>
    And the price user should pay should be <finalPrice>
    Examples:
    | code     | discount      | price    | finalPrice
    | 1        | 90            | 10000    | 9910
    | 2        | 80            | 1000     | 920
    | 3        | 70            | 100      | 30
```

圖 35 購物車網站購物功能情境/規格

這 6 名受測者分為「無智能合約程式撰寫經驗」、「有智能合約程式撰寫經驗」各 3 名做對照，而有智能合約程式撰寫經驗者的條件為：有受過相關訓練相關課程、必須有手動設定開發智能合約所需環境以及有手動建立智能額約且成功部署至區塊鏈者。

因 BDD 的基礎概念建立在針對規格時須與使使用者進行密切的討論，進而撰寫出各種複雜邏輯的規格書以及測試案例，故本實驗針對情境敘述、整合測試、單元測試之時間不加入評估範圍，本實驗針對智能合約撰寫之輔助狀況進行質性評估。在實驗進行過程中測量開發者撰寫智能合約直到通過單元測試與整合測試之時間，並且將結果與平均值進行比較。實驗完成後根據擬定的問卷調查如表 1，由 1 至 5 分審查，5 為同意程度最高分，反之 1 為同意程度最低分。根據此調查結果，觀察使用者撰寫智能合約所給予的回饋，並且根據問卷調查的結果進行分析。

表 1 問卷調查表

調查問題	1	2	3	4	5
1. 因為操作順暢，若需要開發智能合約時，我願意使用此選軟體。					
2. 我發現此介面系統有過於複雜、不必要的設計。					
3. 我認為此介面系統易於使用。					

DOI:10.6814/THE.NCCU.CS.001.2018.B02

4. 在第一次使用時，我認為我需要技術人員的支援才能使用此介面系統。					
5. 使用完一次後，我認為我需要技術人員的支援才能使用此介面系統。					
6. 我發現此系統良好地整合了各個功能。					
7. 我認為此介面系統有太多的不一致。					
8. 我能想見大部分的人都能快速地學會操作此介面系統。					
9. 我認為此介面系統使用起來很麻煩。					
10. 我認為在定義規格(feature)時沒有撰寫程式碼經驗的人也可以參與。					
11. 我可以很輕易地找到此系統的各種功能。					
12. 在我開始使用此系統前我需要大量的學習。					
13. 我認為此系統可以協助我撰寫智能合約。					

5.2.2 受測結果說明

在實驗過程中經由使用者同意，進行計時記錄以利分析，6 名受測者教育程度皆在大學以上，男女比例為 2：4，而在是否有智能合約開發經驗之兩組受測者中，每組具有程式開發經驗 1 年者 1 位，3 年以上者 2 位。

在受測結果中，有智能合約程式撰寫經驗之受測者 1 僅使用了四分 17 秒開發智能合約，根據觀察可以發現，在受測過程中受測者若對於介面有任何疑問都可以主動提問，而此受測者為對論文所提供之平台發問率最高者，其問題包含標籤名稱之意思以及按鈕名稱所代表的意義，根據此觀察發現平台在介面上仍有很大的改善空間，可以減短開發者開發智能合約所需時間。

表 2A、B 完成實驗時間（單位: 分：秒）

	受測者 1	受測者 2	受測者 3	平均時間

DOI:10.6814/THE.NCCU.CS.001.2018.B02

A. 無智能合約程式撰寫經驗	10:33	12:50	12:06	11:50
B. 有智能合約程式撰寫經驗	4:17	13:47	8:29	8:51

表 2 紀錄受測者操作本平台完成整個實驗的操作時間，時間單位為四捨五入到秒。最後的總平均時間為 10 分 20 秒。完成實驗時間計算包含使用者理解規格與整合測試與單元測試時間且撰寫智能合約直到通過，但不包含說明操作時間 20 分鐘以及 10 分鐘的規格說明。

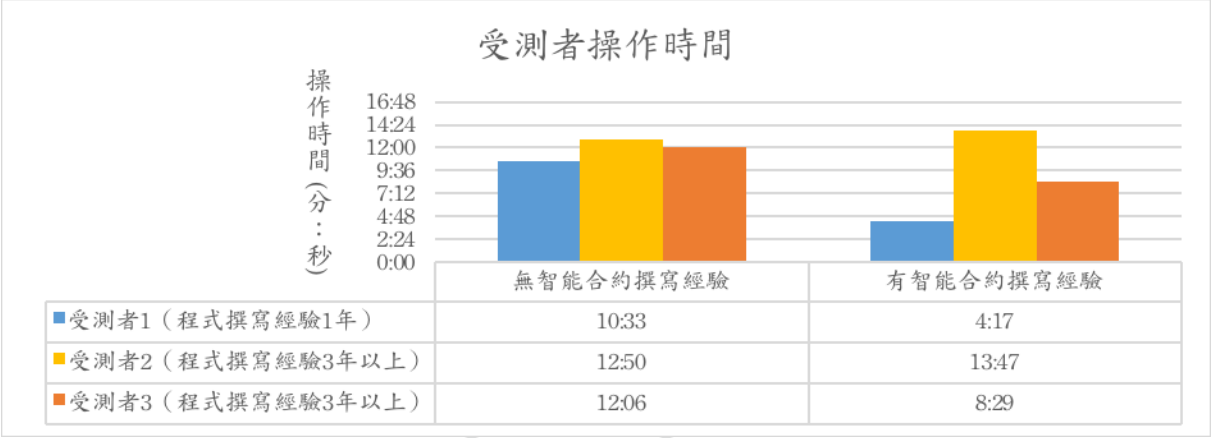

圖 36 受測者操作時間

比較各組受測者操作本研究平台之操作時間，可以發現有智能合約撰寫經驗者在操作時間上仍有明顯的差距，例如在有智能合約撰寫經驗之受測者 2 以及受測者 3 可以發現時間有明顯的差距，根據問卷調查結果發現在本平台操作介面上針對介面說明上仍有可改善的空間，例如 Mocha 在部分使用者中仍不易連結為單元測試。

DOI:10.6814/THE.NCCU.CS.001.2018.B02

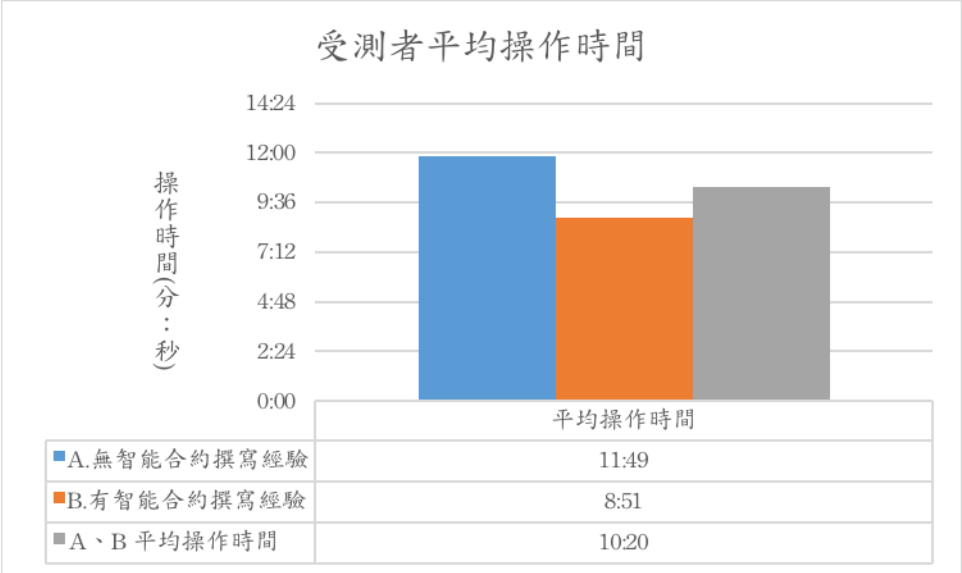

圖 37 受測者平均操作時間

比較各組受測者之操作本研究平台之平均操作時間，有智能合約撰寫經驗 B 組在操作時間優於無智能合約操作經驗 A 組，發現 A 組平均的受測時間比平均值（10 分 20 秒）所花費還要多 14%，B 組平均則是小於平均值 14%。在此比較中可發現此平台對於有智能合約開發經驗者仍是比較易於使用。

5.2.3 改善與修正方向

上述驗證本研究的可行性與易用程度，另外，在受測結束後根據受測者的問卷調查整理成易用性評分表，使用者可以給予 1-5 分，其中 1 為最不同意，5 為最同意，本實驗調查正向與非正向易用性正向調查問題並整理如下表。

表 3 易用性正向調查

	問題敘述	總分
1.	因為操作順暢，若需要開發智能合約時，我願意使用此選軟體	26
2.	我認為此介面系統易於使用	22
3.	我發現此系統良好地整合了各個功能	25

DOI:10.6814/THE.NCCU.CS.001.2018.B02

4.	我能想見大部分的人都能快速地學會操作此介面系統	23
5.	我認為在定義規格(feature)時沒有撰寫程式碼經驗的人也可以參與定義	28
6.	我可以很輕易地找到此系統的各種功能	24
7.	我認為此系統可以協助我撰寫智能合約	28

表 4 易用性非正向調查

	問題敘述	總分
1.	我發現此介面系統有過於複雜、不必要的設計	10
2.	使用完一次後，我認為我需要技術人員的支援才能使用此介面系統	13
3.	我認為此介面系統有太多的不一致	10
4.	我認為此介面系統使用起來很麻煩	15
5.	在我開始使用此系統前我需要大量的學習	19

本研究根據調查結果易用性正向調查小於平均基本分數(15 分)者以及易用性非正向調查大於基本分者進行探討。根據調查結果顯示，受測者認為在使用本平台時需要大量的學習，其原因為受測對象皆對於 BDD 開發方法仍尚未非常熟悉，故若使用者在第一次使用本平台時，因不熟悉 BDD 開發方法，故仍需花費一些時間習慣根據測試程式所定義的方法而撰寫程式。例如，一般開發者在開發程式時可自行定義方法名稱，但在 BDD 開發方法中，因必須先撰寫完成測試程式再開發方法，所以開發者必須依照定義的方法名稱開發，在測試程式執行時才能找到開發法並且執行。因本實驗若由討論需求且撰寫測試程式階段開始測試會花費過長時間，故本實驗之開發者由撰寫智能合約階段進行測試。因此，造成了試驗過程中測試程式不由受測者自行撰寫，受測者必須先閱讀完測試程式了解方法名稱定義的依據後才能進行智能合約開發。一般來說，開發者會由討論需

DOI:10.6814/THE.NCCU.CS.001.2018.B02

求時進入 BDD 流程並且撰寫測試程式，經過不斷地討論且撰寫測試的流程，開發者可以更為熟悉 BDD 開發方法，所以方法的名稱一般來說將為開發者自行定義或者熟悉，並不會有此問題產生。

DOI:10.6814/THE.NCCU.CS.001.2018.B02

第六章

結論

本論文針對區塊鏈服務（BssS,Blockchain as a Service）產出使用基於行為測試驅動開發的智能合約測試技術，針對目前智能合約欠缺成熟的開發工具之議題開發一智能合約整合測試平台。此平提供 BDD 風格的智能合約自動整合測試，可在整個智能合約開發週期提供開發支援，協助將業務邏輯轉為可執行的規格書，到自動化部屬智能合約且驗證業務邏輯正確性與功能正確性，並且導入中介技術協助區塊鏈應用更快速的發展。此論文提出可支援 Solidity 智能合約語言的自動整合測試平台，並且以紅利點數交換為案例說明本平台，就所開發的系統進行可行性驗證，以 Web 方式提供自動整合測試服務，整合目前智能合約開發與測試所需工具，提供並降低目前開發與測試 Ethereum 上以 Solidity 寫作智能合約與測試時，必須面臨的高複雜度工作所產生的負擔。

根據易用性實驗結果，本研究未來可以透過中介層改善，將介面封裝的更完善，替換整合測試程式所使用的 Web3.js，改以更簡易的使用介面，將 address、abi、byteCode 等一般無智能合約開發經驗者不易理解的參數封裝，簡化撰寫測試程式的難易度。另外，將針對 Web 介面的使用性進行優化，例如將介面文字與 BDD 開發方法的名稱一致，隱藏 Mocha 等標籤頁面名稱改為一般開發者可以理解的單元測試等。

DOI:10.6814/THE.NCCU.CS.001.2018.B02

參考文獻

[1] N. Szabo, "Formalizing and securing relationships on public networks," First Monday, vol. 2, no. 9, 1997.

[2] Nakamoto, Satoshi Bitcoin: A peer-to-peer electronic cash system. 2009.

[3] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," IEEE Access, vol. 4, pp. 2292–2303, 2016.

[4] M. Swan, Blockchain: Blueprint for a new econ- omy. “O’Reilly Media, Inc.”, 2015.

[5] D. Mamnani, “Testing of smart contracts in the blockchain world,” Blog post, 2017. [Online]. Available: https://www. capgemini.com/blog/capping-it-off/2017/01/ testing-of-smart-contracts-in-the-blockchain-world

[6] L. Crispin and J. Gregory, Agile testing: A prac- tical guide for testers and agile teams. Pearson Education, 2009.

[7] J. F. Smart, BDD in Action. Manning, 2014.

[8] M. Gärtner, ATDD by example: a practical guide to acceptance test-driven development. Wesley, 2012.

[9] M. Hüttermann, “Speci cation by example,” De- vOps for Developers, pp. 157–170, 2012.

DOI:10.6814/THE.NCCU.CS.001.2018.B02

[10] W. Trumler and F. Paulisch, “How speci cation by example and test-driven development help to avoid technial debt,” in Managing Technical Debt (MTD), 2016 IEEE 8th International Workshop on. IEEE, 2016, pp. 1–8

[11] C. Matts and G. Adzic, “Feature injec- tion: three steps to success,” 2011. [On- line]. Available: https://www.infoq.com/articles/ feature-injection-success

[12] M. Wynne and A. Hellesoy, The cucumber book: behaviour-driven development for testers and de- velopers. Pragmatic Bookshelf, 2012.

[13] R. Lawrence and P. Rayner, Behavior-Driven De- velopment with Cucumber. Pearson, 2016.

[14] N. Li, A. Escalona, and T. Kamal, “Sky re: Model-based testing with cucumber,” in Software Testing, Veri cation and Validation (ICST), 2016 IEEE International Conference on. IEEE, 2016, pp. 393–400.

[15] M. Rahman and J. Gao, “A reusable auto- mated acceptance testing architecture for microservices in behavior-driven development,” in Service-Oriented System Engineering (SOSE), 2015 IEEE Symposium on. IEEE, 2015, pp. 321– 325.

[16] S. Sivanandan et al., “Agile development cy- cle: Approach to design an e ective model based testing with behaviour driven automation frame- work,” in Advanced Computing and Communica- tions (ADCOM), 2014 20th Annual International Conference on. IEEE, 2014, pp. 22–25.

[17] J. S. Dumas and J. Redish, A practical guide to usability testing. Ablex Pub. Corp., 1993.

DOI:10.6814/THE.NCCU.CS.001.2018.B02

[18] J. Nielsen,” Why You Only Need to Test with 5 Users”, User Testing, in Nielsen Norman Group, 2000

[19] S. Porru, A. Pinna, M. Marchesi, R. Tonelli, Blockchain-oriented software engineering: challenges and new directions. In Proceedings of the 39th International Conference on Software Engineering Companion, 2017, pp. 169-171

[20] K. Benk, Test-driven development: by example. Addison-Wesley Professional, 2003

[21] J. Nielsen, "10 usability heuristics for user interface design," Fremont: Nielsen Norman Group. [Consult. 20 maio 2014]. Disponível na Internet, 1995.

DOI:10.6814/THE.NCCU.CS.001.2018.B02

附錄

附錄一 相關發表著作

1. 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," 台灣軟體工程研討會 (TCSE), 台中, 台灣, 2017.
2. Chun-Feng Liao, Ching-Ju Cheng, Kung Chen, Chen-Ho Lai, Tien Chiu, and Chi Wu- Lee, "Toward a Service Platform for Developing Smart Contracts on Blockchain in BDD and TDD styles," in Proc. 10th IEEE International Conference on Service- Oriented Computing & Applications (IEEE SOCA), 2017.

DOI:10.6814/THE.NCCU.CS.001.2018.B02

附錄二 Web API 說明

1. 應用程式介面

- 基本入門

  智能合約整合測試平台 API 是以 HTTP 為基礎。

- 發出要求

  您可以透過 HTTP 介面存取智能合約整合測試平台 API

- 常用參數詞彙說明

  feature：

  表示功能敘述，使用 Gherkin 格式敘述所需要開發的功能目的及步驟結果。

  step-definitions：

  表示整合測試，此為 javascript 所撰寫，定義 feature 所敘述的每一個步驟(Step)所表示的實際運行內容。

  Mocha：

  表示單元測試，使用 mocha 框架所撰寫。

  Solidity

  區塊鏈中智能合約之程式碼。

2. 登入服務

- 登入(Sign in)

  HTTP Method

  GET

  網址(url)

  /api/sign_in?userID=xxxxx&password=xxxxx

  參數說明

字段	類型	說明
userID	string	會員登入帳號
password	string	會員登入密碼

  200 成功 回傳值(json)

DOI:10.6814/THE.NCCU.CS.001.2018.B02

字段	類型	說明
inf	string	登入成功
apikey	string	使用者之 API KEY

4XX 失敗 回傳值(json)

名稱	描述
UserIDError	找不到此帳號
PasswordError	密碼錯誤

- 註冊(sign up)

  HTTP Method

  post

  網址(url)

  /api/sign_up

  參數說明

字段	類型	說明
userID	string	會員登入帳號
password	string	會員登入密碼
email	string	會員註冊電子郵件

200 成功 回傳值(json)

字段	類型	說明
inf	string	註冊成功

4XX 失敗 回傳值

名稱	描述
userIDExistError	帳號已被註冊過錯誤
formatError	帳號或密碼格式錯誤

3. 專案管理

- 查看現有 projects

  HTTP Method

DOI:10.6814/THE.NCCU.CS.001.2018.B02

GET

網址(url)

/api/projects?apikey=APIKEY

參數說明

字段	類型	說明
apikey	string	使用者之 API KEY

回傳值(json 陣列)

字段	類型	說明
name	string	project 名稱
create_date	string	創建日期，yyyy-mm-dd hh:mm:ss
last_update	string	上次儲存日期，yyyy-mm-dd hh:mm:ss

4XX 失敗 回傳值

名稱	描述
ApiKeyInvalidError	API KEY 損毀或不合法

- 取得 project 內容

HTTP Method

GET

網址(url)

/api/project/:project_name?apikey=APIKEY

參數說明

字段	類型	說明
apikey	string	使用者之 API KEY
:project_name	stirng	專案名稱

200 成功 回傳值(json)

字段	類型	說明
feature	string	
step-definitions	string	
mocha	string	
solidity	string	

DOI:10.6814/THE.NCCU.CS.001.2018.B02

4XX 失敗 回傳值

名稱	描述
ApiKeyInvalidError	API KEY 損毀或不合法
ProjectNameError	查無此 project

- 更新專案

HTTP Method

PUT

網址(url)

/api/project/:project_name?apikey=APIKEY

參數說明

字段	類型	說明
apikey	string	使用者之 API KEY
:project_name	stirng	專案名稱

成功回傳值(json) 200 OK

字段	類型	value
feature	string	如不要更新請略過
step-definitions	string	如不要更新請略過
mocha	string	如不要更新請略過
solidity	string	如不要更新請略過

4XX 失敗 回傳值(json)

名稱	描述
ApiKeyInvalidError	API KEY 損毀或不合法
ProjectNameError	查無此 project

- 新增專案

HTTP Method

DOI:10.6814/THE.NCCU.CS.001.2018.B02

post

網址(url)

/api/project/:project_name?apikey=APIKEY

參數說明

字段	類型	說明
apikey	string	使用者之 API KEY
:project_name	stirng	專案名稱
contract_name	stirng 陣列	智能合約陣列
instance	stirng 陣列	宣告陣列

成功回傳值(json) 200 OK

字段	類型	說明
inf	string	新增專案成功

4XX 失敗 回傳值(json)

名稱	描述
ApiKeyInvalidError	API KEY 損毀或不合法
ProjectNameExistError	Project name 已使用過

- 刪除專案

HTTP Method

delete

網址(url)

/api/project/:project_name?apikey=APIKEY

參數說明

字段	類型	說明
apikey	string	使用者之 API KEY
:project_name	string	專案名稱

成功回傳值(json) 200 OK

字段	類型	說明
inf	string	註冊成功

4XX 失敗 回傳值(json)

DOI:10.6814/THE.NCCU.CS.001.2018.B02

名稱	描述
ApiKeyInvalidError	API KEY 損毀或不合法
ProjectNameError	查無此 project

DOI:10.6814/THE.NCCU.CS.001.2018.B02

4. 自動化測試

- 執行 cucumber

HTTP Method

GET

網址(url)

/api/run/cucumber/:project_name?apikey=APIKEY

參數說明

字段	類型	說明
apikey	string	使用者之 API KEY
:project_name	string	專案名稱

成功回傳值(json) 200 OK

字段	類型	說明
result	string	執行結果

4XX 失敗 回傳值(json)

名稱	描述
ApiKeyInvalidError	API KEY 損毀或不合法
ProjectNameError	查無此 project

- 執行 mocha

HTTP Method

GET

網址(url)

/api/run/mocha/:project_name?apikey=APIKEY

參數說明

字段	類型	說明
apikey	string	使用者之 API KEY
:project_name	string	專案名稱

成功回傳值(json) 200 OK

DOI:10.6814/THE.NCCU.CS.001.2018.B02

字段	類型	說明
result	string	執行結果

4XX 失敗 回傳值(json)

名稱	描述
ApiKeyInvalidError	API KEY 損毀或不合法
ProjectNameError	查無此 project

- solidity compile

HTTP Method

GET

網址(url)

/api/run/compile/:project_name?apikey=APIKEY

參數說明

字段	類型	說明
apikey	string	使用者之 API KEY
:project_name	string	專案名稱

成功回傳值(json) 200 OK

字段	類型	說明
result	string	執行結果

4XX 失敗 回傳值(json)

名稱	描述
ApiKeyInvalidError	API KEY 損毀或不合法
ProjectNameError	查無此 project

DOI:10.6814/THE.NCCU.CS.001.2018.B02