การพัฒนาซอฟต์แวร์

Software Development

Piyorot piyachan
http://www.chapterpiece.com

การพัฒนาซอฟต์แวร์ - Software Development

เกือบหนึ่งปีมาแล้วที่เพื่อนคนนึงอยากรู้ว่าการพัฒนาซอฟต์แวร์ที่เป็นขั้นเป็นตอนตั้งแต่ต้นจนจบเป็นอย่างไร ผมก็เขียนบทความที่ต่อเนื่องกันทั้งหมดหกบทความเพื่ออธิบายเรื่องราวทั้งหมดนี้ครับ

และเนื่องจากบทความอีกมากมายที่ผมเขียนผ่าน www.chapterpiece.com ที่อาจจะทำให้เพื่อนใหม่ๆมองไม่เห็นบทความทั้งหกนี้ ผมเลยถือโอกาสวันหยุดยาวสงกรานต์ปีนี้ (2554) เขียนหนังสือเล่มนี้ขึ้นมาครับ ... "How to Build Software Ebook"

เนื้อหาข้างในจะอธิบายกระบวนการพัฒนาซอฟต์แวร์อย่างมีระบบทั้งในมุมของ Software Development และ Project Management ครับ เริ่มต้นตั้งแต่การเก็บรวบรวมความต้องการของลูกค้า การวางแผนโครงการ จนถึงการทดสอบระบบครับ

เหตุการณ์สมมติถูกยกขึ้นมาเพื่อรวมกับทฤษฎีที่พอประมาณน่าจะช่วยให้ง่ายต่อความเข้าใจและการะประยุกต์ใช้กับการทำงานจริงของเพื่อนๆครับ

และเหมือนเดิม ... หวังว่าทุกคนจะได้ประโยชน์จาก Ebook เล่มนี้ครับ

ขอบคุณมากที่อ่านครับ

ปีโยรส ปียจันทร์

(kannique)

PUBLISHING INFORMATION

First published in Thailand in June 2010 and can be downloaded via chapterpiece.com

Designed with OpenOffice Writer – an open source office suite program by Sun Microsystems. Template created by Lela Iskandar Suhaimi from JustAddContents.com

Table of Contents

1 การเก็บรวบรวมความต้องการของลูกค้า

เพื่อนคนนึงอยากรู้ว่าการพัฒนาระบบขึ้นมาซักระบบต้องทำอะไรบ้างตั้งแต่ต้นจนจบ วันนี้ผมก็เลยรวบรวมข้อมูลมาให้ แต่เนื่องจากข้อมูลมันเยอะมากครับ ผมขอแบ่งเป็นบทความย่อยนะ เรามาเริ่มกันเลยครับ

ข้อมูลเบื้องต้นเกี่ยวกับ Project

หลังจากบริษัท XYZ Technology จำกัด ซึ่งเป็น Software house เล็กๆที่ถนัดงานที่เกี่ยวกับ Web application ทุกชนิดชนะการประมูลโครงการพัฒนาระบบเช่าหนังสือออนไลน์ให้กับร้านหนังสือ ยักษ์ใหญ่แห่งหนึ่งแล้ว คุณเจ้าของบริษัทก็บอกข่าวดีนี้กับสมาชิกทุกคนก่อนประกาศเริ่ม Project ทันที โดย Project นี้มี Project Sponsor (ก็ตัวเจ้าของเอง) 1 คน Project Manager 1 คน Business Analyst 1 คน Developer 3 คน และ QA 2 คน รวมเป็น 8 คน ... ขนาดกำลังดีเลยครับ

Requirement คร่าวๆของ Project นี้ก็คือทำ Web application สำหรับการเช่าหนังสือออนไลน์ที่มี

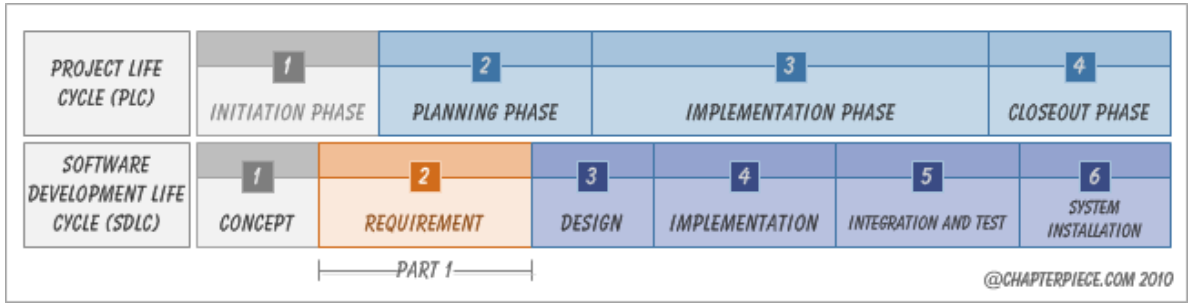
- ระบบจัดการการเช่าหนังสือ (Rental module)
- ระบบเพิ่ม-ลดหนังสือ (Book management module)
- ระบบจัดการข้อมูลลูกค้า (User management module)
- ระบบการจ่ายเงินออนไลน์ (Payment module)
- และระบบการจัดการเวปไซต์ (Administration module)

ทั้งหมดนี้ต้องทำให้เสร็จภายในเวลา 5 เดือน

ภาพรวมของงาน

ผมเคยเขียนบทความเกี่ยวกับภาพรวมของการทำ Software development project ไปแล้วที่นี่ วันนี้ขอทบทวนคร่าวๆอีกครั้งครับ การทำ Software development project ประกอบด้วยสองส่วนสำคัญนั้นคือ 1. Software development และ 2. Project management ซึ่งงานทั้งสองด้านจะต้องทำควบคู่กันไปตั้งแต่เริ่มต้นจนจบเลย

สำหรับภาคแรกนี้ผมจะขอเน้นไปที่ช่วงแรกของ Project ซึ่งจะประกอบไปด้วยการทำ Requirement และ Design เบื้องต้นในส่วนที่เป็น Software development ครับ



ถ้าดูจากรูปจะเหมือนว่าผมข้ามส่วนที่เป็น Initiation กับ Concept ไป จริงๆแล้วเป็นแบบนี้ครับ สองขั้นตอนที่ว่านี้เกี่ยวข้องโดยตรงกับการพิจารณาว่า Project นี้ควรจะมีหรือไม่ แล้วถ้ามีต้องทำอะไรบ้าง ในกรณีของเราบริษัทหนังสือมองเห็นโอกาสทางธุรกิจว่า เอ้อ ถ้าเราต่อยอดธุรกิจโดยให้บริการเช่าหนังสือออนไลน์มันน่าจะเพิ่มกำไรให้ บริษัทได้มากอยู่นะ จากวิสัยทัศน์ของลูกค้าก็กลายมาเป็น Project ครับ จากนั้นลูกค้าก็จะมองแนวโน้มทางธุรกิจต่อไปว่า เอ่ ได้ข่าวแว่วๆมาว่าบริษัทหนังสืออีกรายก็กำลังมองๆธุรกิจแบบนี้อยู่เหมือนกันนะ ไม่ได้การล่ะ แบบนี้ต้องเราต้องชิงลงมือก่อน เราต้องเปิดเวปไซต์นี้ให้ได้ภายใน 6 เดือนเพื่อชิงความได้เปรียบ จบตรงนี้เราได้กำหนดเวลาคร่าวๆของ Project มาแล้วครับ ลูกค้าก็จะคิดนั่นนี่ต่อไปอีกหน่อยเพื่อให้ได้มาซึ่ง Requirement เบื้องต้นก่อนเปิดประมูลเพื่อหาบริษัทมาทำระบบให้เค้า

เพื่อนๆจะเห็นว่า Concept ถูกทำจนเสร็จเรียบร้อยแล้วโดยลูกค้าเอง ผลลัพธ์ที่ได้ออกมาก็คือตัว Project กับ Requirement เบื้องต้นที่ถูกส่งมาให้เรา ถ้าเพื่อนๆอยากรู้เพิ่มเติมที่มาที่ไปของ Project หรือ Requirement ลองอ่านบทความนี้ดูครับ

สาหรับส่วนที่เป็น Initiation นั้นเป็นงานในส่วนที่บริษัทของเราเองต้องรับผิดชอบในเรื่องการพิจารณาความเป็นไปได้ในการเข้าร่วมประมูล นั่นรวมถึงการศึกษาทางการเงินด้วยว่าถ้ารับ Project นี้มาแล้วจะได้กำไรคุ้มค่าหรือไม่ ราคาประมูลควรเป็นเท่าไร

ในอีกมุมหนึ่งเราก็ต้องเตรียมทีมงานที่จะมาทำ Project นี้ไว้คร่าวๆด้วยครับ ซึ่งหน้าที่ตรงนี้ก็เป็นของ Project Sponsor คนที่มีอำนาจตัดสินใจว่าจะเข้าประมูลหรือไม่อย่างไร แล้วก็เป็นคนเลือก Project Manager ซึ่งจะมารับช่วงต่อในการเตรียมข้อมูลที่ต้องใช้ในการตัดสินใจของต่างๆของ Project Sponsor นั่นรวมถึงข้อดี ข้อเสีย โอกาส ความเสี่ยง ความพร้อมของบริษัทเราเอง ทุกเรื่องแหละครับ อีกคนที่สาคัญก็คือ Business Analyst ซึ่งมีความรู้ในงานที่เกี่ยวกับ Web application ดีก็ต้องช่วยให้คำปรึกษาและดูความเป็นไปได้ต่างๆในแง่ Technical กับ Project Manager ครับ

สุดท้ายเราชนะประมูล Project นี้เรียบร้อย เริ่มงานกันเลยดีกว่าครับมีเวลาแค่ 5 เดือน (เพื่อนๆอาจจะงงว่า ลูกค้าอยากให้เปิดเวปไซต์ภายใน 6 เดือน แต่ทำไมผมบอกว่าเรามีเวลาแค่ 5 เดือน สาเหตุก็เพราะกระบวนการ Initiation กับ Concept กินไป 1 เดือนแล้วครับ เลยเหลือแค่ 5 เดือนให้เราพัฒนาระบบจริง)

Software Development Life Cycle - Requirement

ก่อนเราจะสร้างอะไรซักอย่างเราต้องเข้าใจก่อนว่าอะไรซักอย่างมันคืออะไร (ไม่งงนะครับ) กระบวนการทำให้เราเข้าใจและเขียนเอกสารเกี่ยวกับอะไรซักอย่างก็คือ Requirement Analysis ครับ เพื่อให้ง่ายต่อการทำงานทั้งกับลูกค้าและในทีมเอง การทำ Requirement Analyst จะถูกแบ่งออกเป็นสองส่วนได้แก่ Customer requirements (C-requirements) และ Developer requirements (D-requirements) สองส่วนนี้แตกต่างกันทั้งในมุมมอง เทคนิคที่ใช้และผลลัพธ์ที่ได้ครับ เรามาดูทีละส่วนกันเลย

Customer Requirement (C-Requirement)

ในขั้นตอนนี้จะเป็นการเขียนเอกสารที่บอกถึง Requirement ทั้งหมดของระบบเช่าหนังสือเราโดยใช้ภาษาที่ลูกค้าเข้าใจได้ง่าย โดยทั่วไปแล้วตรงนี้จะเป็นหน้าที่ของ Business Analyst ที่จะทำการติดต่อพูดคุยกับลูกค้าโดยตรงเพื่อเก็บข้อมูลที่สำคัญก่อนส่งต่อให้ Developer ในขั้นตอนต่อไป

สาเหตุที่เราต้องมี C-requirements ก็เพราะว่าบางครั้งลูกค้าวาดภาพระบบที่อยากได้ออกมาแบบไม่ชัดเจนมากนัก แถมลูกค้าหลายคนก็มองระบบที่อยากได้ต่างกันอีก (อันนี้ธรรมชาติมากๆ) เช่น ถ้าพูดถึงระบบเช่าหนังสือ ในฐานะลูกค้าเพื่อนๆคิดถึงอะไรกันบ้างครับ?

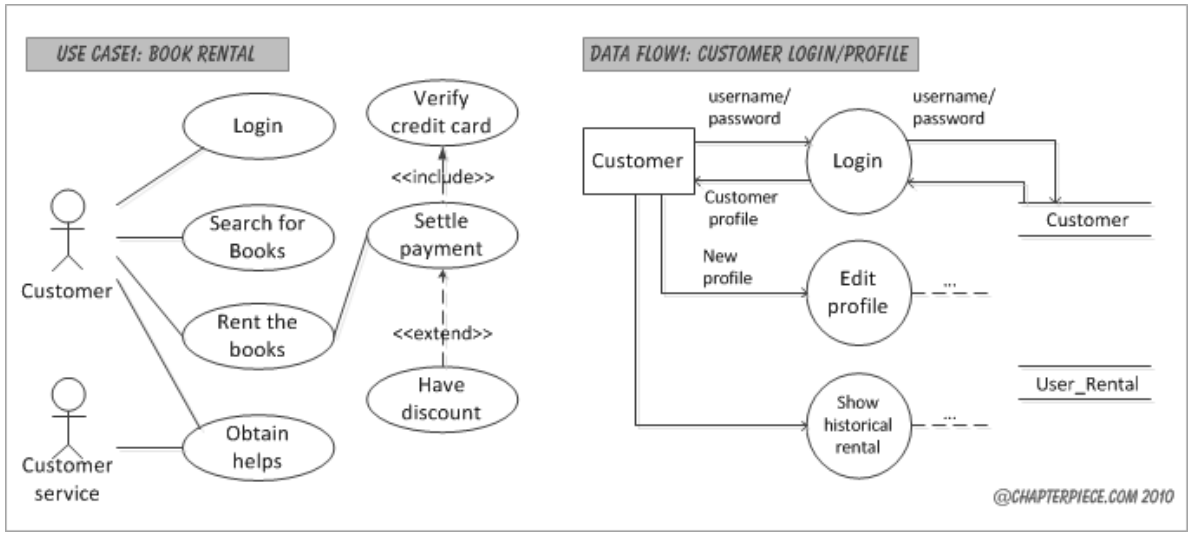
อยากได้แบบง่ายๆทั่วไป add/delete user, add/delete book อะไรแบบนี้แหละ

- อยากได้ระบบที่การจัดคิวลูกค้าถ้าหนังสือเล่มนั้นไม่ว่างด้วยนะ
- อ้อ แล้วถ้าหนังสือเล่มนั้นถูกคืนมาแล้ว ระบบต้องจัดการแจ้งลูกค้าคนแรกในคิวเลยนะ
- ผมอยากได้ระบบที่เชื่อมต่อกับไปรษณีย์เพื่อจัดส่งหนังสือให้ถึงมือลูกค้าได้เลย
- เอ่ แต่ดิฉันว่า ... ระบบเช่าหนังสือมันน่าจะมีแต่หน้าร้านนะ ทำไมต้องมีแบบออนไลน์หละ งั้นเรื่องจัดส่งก็ไม่จำเป็นซักหน่อย
- ผมว่ามันต้องครบวงจรกว่านี้นะ ต้องมีระบบติดต่อสำนักพิมพ์เพื่อดึงข้อมูลหนังสือใหม่ๆมาแสดงที่เวปไซต์เราด้วย
- เอ้อ เห็นด้วยๆ แต่จะให้ดีดิฉันว่าให้มีระบบ email marketing ด้วยเลยนะ ลูกค้าเราจะได้รับข่าวสารโปรโมชั่นหรือหนังสือใหม่ๆอัตโนมัติ
- ...

ถ้าเราไม่มีข้อมูลที่ชัดเจนว่าจริงๆแล้ว ลูกค้าอยากได้อะไร (Concept of Operations) โอกาสที่ Project จะล้มเหลวเพราะการเปลี่ยนแปลงที่จะเกิดขึ้นระหว่างเราทำงานก็มีสูงมาก ความสาคัญก็อยู่ตรงนี้แหละครับ สิ่งที่ท้าทายก็คือส่วนมากแล้วลูกค้าจะขาดความรู้ความเข้าใจทางด้านเทคนิคลึกๆ ดังนั้นเราจำเป็นต้องหาเทคนิคที่เหมาะสมมาใช้ เช่น Use cases, Data flow diagram หรือ Drafting user interface เป็นต้นครับ

Use Cases

โดยทั่วไปแล้ว Requirement จะถูกระบุออกมาเป็นความสัมพันธ์ระหว่างระบบ (system) และผู้ใช้ (user) ดังนั้นเราจะใช้ use case เพื่ออธิบายถึง System functionality ต่างๆที่จะให้บริการกับลูกค้า จากตัวอย่างที่ผมยกมาก็เช่นเราจะมีระบบจัดการการเช่าหนังสือ (Rental Module) ระบบเพิ่ม-ลดหนังสือ (Book Management Module) ระบบจัดการข้อมูลลูกค้า (User Management Module) ระบบการจ่ายเงินออนไลน์ (Payment Module) และระบบการจัดการเวปไซต์ (Administration Module) ทั้งหมดนี้จะถูกระบุให้ชัดเจนขึ้นด้วย use case ผมเขียน use case diagram กับ data flow diagram แบบคร่าวๆไว้เป็นตัวอย่างครับ



Data Flow Diagram

บางครั้งลูกค้าพยายามอธิบาย Requirement โดยใช้การไหลของข้อมูลผ่านกระบวนการต่างๆ ในกรณีนี้เราสามารถใช้ Data flow diagram เป็นเครื่องมือช่วยได้อย่างดีเลยครับ เช่นกระบวนการ Login ซึ่งต้องการข้อมูลคือ Username และ Password ถ้าถูกต้องระบบก็จะดึงข้อมูลของ Customer คนนั้นจากฐานข้อมูลตารางชื่อ Customer ก่อนส่งให้หน้าเวปเพื่อแสดงผลเป็นต้น

มีอีกหลายเทคนิคที่ใช้เพื่อสร้าง C-requirements เช่น state transition diagram หรือ activity diagram ลองหาข้อมูลเพิ่มเติมดูได้ที่นี่ครับ

User Interface

Requirement ที่เป็นตัวอักษรในเอกสารหรือรูปภาพที่เป็น diagram นั้นอาจจะตอบโจทย์ของลูกค้าและตัวเราได้ไม่ครบถ้วน โดยเฉพาะอย่างยิ่งข้อมูลและเอกสารพวกนั้นบอกลูกค้าไม่ได้ว่าสุดท้ายแล้วระบบ ที่ออกมาจะมีหน้าตาเป็นยังไง เรามาใช้ Drafting User Interface แก้ปัญหานี้กันดีกว่าครับ

จริงๆแล้ว User interface นั้นเป็นกระบวนการหนึ่งใน Design phase ของ Software development life cycle แต่ไม่เป็นไรครับเรามองว่างานนี้เป็นส่วนหนึ่งของ Requirement phase ได้เหมือนกันถ้าเราทำแค่ Draft user interface ครับผม ที่เราต้องใช้เทคนิคนี้ก็เพราะโดยทั่วไปแล้วลูกค้ามักจะมองเห็นภาพระบบเป็น Graphic user interface (GUI) เป็นหลัก ถ้าเราทำหน้าตาของระบบคร่าวๆไปให้ดู ลูกค้าจะมีโอกาสได้ให้ความคิดเห็นว่าอะไรขาด อะไรเกิน ในขณะเดียวกันก็เป็นการสร้างความเข้าใจในเรื่องหลักการทางธุรกิจ (Business function) ให้กับลูกค้าและตัวเราด้วยครับ

กระบวนการนี้ Business Analyst รับหน้าที่หนักหน่อยนะครับ แต่รับรองได้ว่าคุ้มค่าแน่นอน หลังจากเราเขียน C-requirements เสร็จแล้ว เราต้องมีการรีวิวเจ้าเอกสารเหล่านี้กับลูกค้าจนกว่าจะได้ข้อสรุปสุดท้าย ก่อนที่จะส่งข้อมูลนี้ให้กับ Developer เพื่อทำ D-requirements ในขั้นตอนต่อไปครับ

Developer Requirement (D-Requirement)

ตอนนี้เรารู้แล้วครับว่าลูกค้าอยากได้อะไร ขั้นตอนต่อไปก็คือจะทำยังไงให้ลูกค้าได้ตามนั้น กระบวนการนี้เป็นหน้าที่ของ Developer และ Business Analyst ที่จะทำงานร่วมกันเพื่อแปล C-requirements ออกมาเป็น D-requirements ซึ่งเป็นเอกสารที่ระบุคุณสมบัติ (Properties) และความสามารถ (Functionalities) ทั้งหมดที่ระบบต้องมีอย่างละเอียด D-requirement ถูกสร้างมาเพื่อ Developer โดยเฉพาะเลยครับดังนั้นมีคำศัพท์เทคนิคอะไรใส่เข้าไปได้หมด D-requirements ประกอบด้วย requirement หลักๆสามแบบคือ Functional, Nonfunctional, และ Inverse requirements ครับ

Functional Requirements

Functional requirement จะพูดถึงความสามารถหรือบริการต่างๆที่ระบบต้องตอบสนองผู้ใช้ได้ ถ้าพิจารณาจากระบบเช่าหนังสือออนไลน์ของเรา เพื่อนๆคิดถึง functional requirements อะไรบ้างครับ?

- ระบบต้องสร้าง/ลบ user ได้
- ระบบต้องแก้ไข user profile ได้
- ระบบต้องค้นหาหนังสือได้โดยระบุชื่อหนังสือ รหัสหนังสือ ชื่อผู้แต่ง และวันที่ตีพิมพ์
- ระบบต้องจัดการคำขอเช่าหนังสือจาก user ได้
- ระบบต้องจัดการการชำระเงินด้วยบัตรเครดิต Visa, Mastercard และ Paypal ได้

- ระบบต้องมีส่วนการช่วยเหลือ user
- ระบบต้องมีการจัดการคิวของหนังสือและ user
- ...

Nonfunctional Requirements: Performance Requirements

Performance requirements ส่วนใหญ่แล้วเกี่ยวข้องกับเวลา ความจุ และการใช้ทรัพยากรของเครื่องซึ่งระบบต้องตอบสนองให้ได้ เช่นเรื่องของ speed, capacity, และ memory and storage usage ครับ โดยทั่วไปแล้ว Performance requirements นี้จะสำคัญมากสำหรับระบบที่เป็น real-time เช่นระบบป้องกันการชน ระบบควบคุมเครื่องบินอัตโนมัติเป็นต้น แต่เห็นแบบนี้ระบบเช่าหนังสือออนไลน์ของเราก็จำเป็นต้องมีเหมือนกันนะครับ

- ระบบต้องตอบสนองการค้าหาข้อมูลหนังสือให้ได้ภายใน 3 วินาทีหลังจากกดปุ่ม “ค้นหา”
- ระบบต้องรองรับ user ได้อย่างน้อย 10,000 คน
- ระบบต้องรองรับการเข้าใช้งานพร้อมกันของ user 500 คนได้
- ...

Nonfunctional Requirements: Reliability and Availability

Reliability requirements ระบุถึงความเชื่อถือได้ของระบบให้เป็นตัวเลขที่วัดค่าได้ (quantified term) ที่มาของ requirement แบบนี้ก็คือความเชื่อ (ที่ถูกต้อง) ที่ว่าไม่มีระบบไหนจะสมบูรณ์แบบ 100% ดังนั้น Reliability จะระบุถึงขอบเขตของความบกพร่องที่ระบบมีอยู่ครับ เช่น ระบบเรดาห์ของสนามบินต้องทำงานผิดพลาดระดับสองได้ไม่เกิน 1 ครั้งต่อเดือนเป็นต้น แล้วระบบเช่าหนังสือออนไลน์ของเราหละ แบบนี้ดีมั้ยครับ?

- ส่วนการชำระเงินต้องทำงานผิดพลาดไม่เกิน 3 ครั้งต่อเดือน
- ส่วนการจัดส่งหนังสือต้องทำงานผิดพลาดไม่เกิน 2 ครั้งต่อเดือน
- ...

Availability requirements มีความหมายใกล้เคียงกับ reliability ครับแต่มองในมุมการให้บริการกับลูกค้า ตัวอย่างง่ายๆก็ Web hosting ทั่วไปที่บอกว่าการันตี 99% up-time นั่นแหละครับ ระบบของเราก็น่าจะคล้ายๆกันครับ

Nonfunctional Requirements: Constraints

Constraints คือข้อกำหนด ข้อจำกัดหรือเงื่อนไขที่ใช้ในการออกแบบและพัฒนาระบบ Constraint มีอยู่หลายรูปแบบครับ เช่นความเที่ยงตรง (accuracy) เครื่องมือและภาษาที่ใช้ในการพัฒนาระบบ มาตรฐานการออกแบบ (Design standard) มาตรฐานในการพัฒนาระบบ มาตรฐานในการเขียนเอกสาร รวมถึงมาตรฐานของ hardware และ software ที่จะใช้ด้วยครับ ยกตัวอย่าง

- ระบบต้องเขียนด้วย jsp servlet ajax technology และมีฐานข้อมูลเป็น oracle
- ระบบต้องใช้ UML ในการออกแบบ
- การพัฒนาระบบต้องเป็นไปตามมาตรฐาน CMMi ระดับ 3
- ระบบต้องถูกติดตั้งบนเครื่อง HP ProLiant DL100
- เอกสารทุกฉบับต้องเป็นไปตามมาตรฐานของลูกค้า
- ...

Inverse Requirements

Inverse requirements จะระบุสิ่งที่ระบบไม่ต้องทำแปลกดีมั้ยครับ? แต่เพื่อให้การเขียน requirement ของเราสมบูรณ์เราจำเป็นต้องรู้ด้วยว่าอะไรที่ลูกค้าไม่ต้องการให้ระบบทำ เราจะได้ไม่เสียเวลาคิดตอนหลังว่า เอ้อ แล้วลูกค้าอยากจะได้ feature นี้มั้ยเนี่ยะ ถ้าจะยกตัวอย่าง inverse requirement จากระบบเรา ผมคิดว่าน่าจะเป็น

- ระบบไม่ต้องดึงข้อมูลหนังสือภาษาอังกฤษ
- ระบบไม่ต้องสร้างรายงานเป็น pdf format
- ระบบไม่ต้องวิเคราะห์ข้อมูลลูกค้าเพื่อทำแผนส่งเสริมการตลาด (promotion)
- ...

เอาหละครับ หลังจากเขียน D-requirements เรียบร้อยแล้ว เราจำเป็นต้องส่งเอกสารนี้ให้ลูกค้ารีวิวเพื่อทำความเข้าใจอีกครั้งก่อนจะ

เริ่มการออกแบบระบบอย่างละเอียดต่อไป ตรงนี้จะเป็นหน้าที่ของ Business Analyst ครับ เมื่อลูกค้าตกลงเรียบร้อย ... ยังครับ ยังไม่เสร็จ

เพื่อง่ายต่อการทำงานและการสื่อสารของ สมาชิกในทีม เราจำเป็นจะต้องพูดคุยกับลูกค้าอีกครั้งเพื่อจัดลำดับความสำคัญของ Requirement ที่มีในมือด้วย กระบวนการนี้เป็นหน้าที่รับผิดชอบโดยตรงของ Project Manager โดยมี Business Analyst เป็นผู้ช่วยครับ จากนั้นข้อมูลทั้งหมดที่เราได้จาก C-requirements, D-requirements และ Prioritized requirements จะต้องถูกบันทึกลงในเอกสารที่เรียกว่า Requirement Traceability Matrix ด้วย งานนี้ Project Manager รับไปครับ

ในมุมมองของผมการเขียน Requirement เป็นทั้งศาสตร์และศิลป์เพราะว่าจะเขียนเน้นไปทาง Technical จำก็ไม่ได้ (ลูกค้าไม่เข้าใจ) จะเขียนแบบเป็นภาษาลูกค้าเกินไปก็ไม่ดี (ไม่ชัดเจนสำหรับ Developer) ตรงนี้จะว่าง่ายก็ง่ายจะว่ายากก็ยากครับ ตรงนี้ต้องอาศัยประสบการณ์มากพอสมควรที่จะสร้างความสมดุลตรงนี้ให้เกิดขึ้น ครับ ผมเขียนบทความเกี่ยวกับเรื่องนี้ไว้ที่นี่ครับ หวังว่าจะพอช่วยให้งานของเพื่อนๆง่ายลงฮะ

เกล็ดเล็กเกล็ดน้อยอีกนิดครับ ถ้าจะให้ดีเลยนะ ใน Requirement phase เราควรจะให้ QA มีบทบาทในการคิดและออกความเห็นเกี่ยวกับ Requirement (อาจรวมไปถึง Design เบื้องต้น) ที่ได้มาจากลูกค้าด้วยครับ ที่ทำแบบนี้เพราะว่า QA จะมีมุมมองที่แตกต่างออกไปจาก Business Analyst และ Developer โดยเฉพาะอย่างยิ่งพวก Performance และ Constraints ของระบบเราครับ ตอนนี้ผมก็ใช้วิธีนี้อยู่ครับ ได้ผลดีมากๆครับ

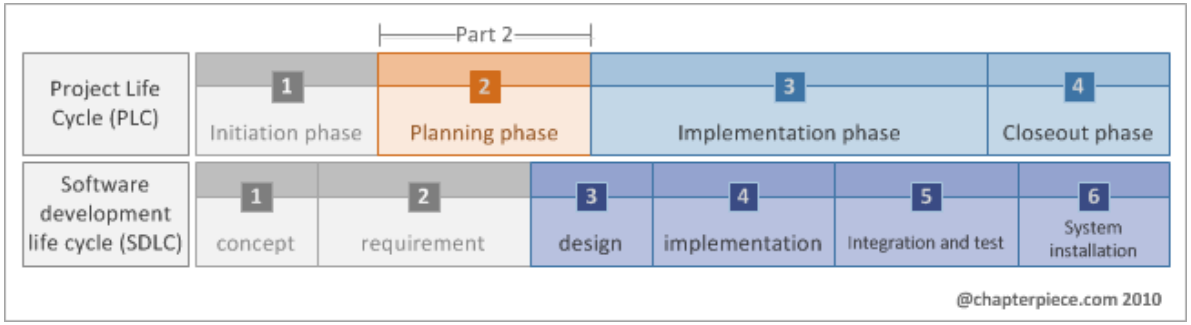
สรุปภาคหนึ่ง

เสร็จแล้วครับ Requirement phase เหนื่อยเลย คิดซะว่าเริ่มต้นดีมีชัยไปกว่าครึ่งแล้วกันครับ ถ้าเราใช้เวลาในช่วง Requirement อย่างเต็มที่เราจะได้ข้อมูลที่ถูกต้องและครบถ้วนที่จะใช้ในการพัฒนาระบบ ความถูกต้องและความครบถ้วนนี่แหละฮะที่จะช่วยลด Requirement change ที่จะเกิดขึ้นได้ Less change – More productivity ครับ

ผมขอจบภาคหนึ่งแค่นี้ก่อนนะครับ เขียนมายาวแล้วฮะกลัวเพื่อนๆอ่านแล้วจะเบื่อไปซะก่อน แต่ไม่ต้องห่วงครับ ภาคสองที่เกี่ยวกับ Project Management – Project Planning จะตามมาติดๆครับ

2 การวางแผนโครงการ

มาเริ่มฟื้นของเก่านิดนึงครับ ตอนนี้เราเตรียม Requirement เรียบร้อยไปแล้ว สิ่งที่ได้มาก็จะมี Customer requirements (C-requirements) มี Developer requirements (D-requirements) แล้วก็ Requirement Traceability Matrix (RTM) ที่เก็บรวบรวม requirement ทั้งหมดที่เรามีใน Project ครับ ถึงตรงนี้ก็คือว่าจบงานขั้นแรกในส่วน Software development life cycle ครับ



ในทางคู่ขนานเราต้องเตรียมวางแผนสาหรับ Project เราไปด้วย นี่เป็นงานในสวน Project Management และเป็นเรื่องที่เราจะคุยกันวันนี้ครับ บอกก่อนครับว่างานทั้งหมดนี้ Project Manager รับไปเต็มๆ ฮาๆ

Project Management – Project Planning

ผมเคยพูดถึงการทำ Project Plan เอาไว้ในหลายๆบทความก่อนหน้านี้ เพื่อนๆลองอ่านดูก่อนเพื่อเป็นแนวทางก็ได้นะครับ

บทเริ่มต้นการวางแผนโครงการ

ขั้นตอนในการวางแผนโครงการ

ตัวอย่างแผนโครงการ

ข้อมูลในบทความเหล่านั้นคงทำให้เพื่อนๆเห็นภาพการทำ Project plan ชัดเจนขึ้นบ้าง งั้นตอนนี้เรามาช่วยกันวางแผน Project เราดีกว่าครับ เริ่มแรกต้องมีอะไรก่อนก็ Statement of Work (SOW) ใช่ปะ ลุยครับ

Statement of Work (SOW), Project Specification, and Milestone Schedules

Statement of Work (SOW)

ส่วนประกอบที่สำคัญสองอย่างของ SOW คือ Project Objective กับ List of Deliverables ครับ จุดสำคัญคือผมจะพยายามเขียน Project Objective ด้วยหลักการ SMART เพื่อที่จะได้ Objective ที่สมบูรณ์และนำไปใช้งานได้จริงครับ ลองดู SOW ที่ผมคิดก่อนนะว่าเพื่อนๆเห็นด้วยหรือไม่

The BookRental has an objective to develop and install an online book rental system to support a new business opportunity of the ABC Publishing Company within 5 months under the budget of 2.5 million Baht. The project scope includes the following tasks:

- System requirements engineering and design
- Equipment procurement and installation
- System development and integration
- Technology transfer and user training
- Maintenance and on-site service for 12 months.

The project deliverables will include, but not limited to,

- System prototype

- Equipment installation
- Completed system
- System documents
- Training packages
- Project reports.

Project Specification

หลังจากได้ SOW มาแล้ว เราต้องเขียน Project specification เป็นงานต่อไปครับ จำเรื่อง Nonfunctional Requirements: Constraints จากบทความที่แล้วได้มั้ยครับ? ถ้าจำไม่ได้ผมจะทวนให้คร่าวๆ มันก็คือข้อจำกัดหรือเงื่อนไขที่ใช้ในการออกแบบและพัฒนาระบบ เราเอาข้อมูลที่ระบุใน requirement ตรงนั้นมาใช้เป็น Project specification ได้เลยครับ ไม่ว่าจะเป็นเรื่องของมาตรฐานการทำงาน เรื่องซอฟต์แวร์และฮาร์ดแวร์ที่จะใช้ในการพัฒนาและการติดตั้งระบบ หรือถ้าไม่ระบุตรงๆ เราจะอ้างถึงเอกสารฉบับอื่นที่อยู่ใน Project ก็ได้เหมือนกัน เช่น

The system must be developed on java-based technology, such as jsp, servlet, and ajax.

- The system database must be Oracle 11g.
- All hardware procurement must be according to RFP-001-2010 (reference document).
- All processes used in the project development must be complied with CMMi level 3.
- All operational deployment and support must be followed ITIL standard.

Milestone Schedules

ถัดมาก็จะเป็น Milestone schedules ซึ่งจะบอกเราคร่าวๆถึงวันเริ่มต้น-สิ้นสุดของ Project รวมไปถึงวันที่จะส่งมอบ Major deliverable แต่ละตัวให้ลูกค้าด้วย ตรงนี้สำคัญอย่างไร? สำคัญตรงที่

- เรื่องเงินครับ ตามปกติแล้วลูกค้าจะทยอยจ่ายเงินให้เราตามปริมาณงานที่ทำเสร็จ เช่น ถ้า design เสร็จเราลูกค้าจะจ่าย 25% ถ้าเราทำสวนงานหลักๆในระบบเสร็จจะจ่ายอีก 20% อะไรแบบนี้ครับ ต้องนี้ Project Manager ต้องวางแผนให้ดีเลยครับ

เพราะว่าต้องเป็นคนบริหาร Project cash flow

- ใช้เป็นข้อมูลในการเขียน Work Breakdown Structure (WBS) ในขั้นตอนถัดไป

ตัวอย่างของ Milestone schedule สำหรับ Project เราก็ตามข้างล่างนี่เลยครับ

Project Start Date = 1 May 2010 | Project End Date = 31 October 2010

Major milestone	*Key Date*
System prototype	25 May 2010
Equipment installation	25 September 2010
Completed system	12 October 2010
System documents	15 October 2010
Training packages	20 October 2010
Project reports	Bi-weekly

Work Breakdown Structure (WBS)

มาถึงทีเด็ดของการทำ Project planning แล้วครับ ผมเข้าใจว่าเพื่อนๆส่วนใหญ่น่าจะคุ้นเคยกับเจ้า WBS พอสมควรนะ เพราะถ้ามีประสบการณ์เกี่ยวกับ Software development project มาเนี่ยะคงเลี่ยงเจ้านี่ไม่พ้น เอาเป็นว่าบทความนี้ผมขออนญาตให้คำแนะนำหลักการเขียน WBS ที่ถูกต้องฝากไว้แทนการอธิบายว่า WBS คืออะไรอย่างละเอียดแล้วกันนะครับ

โดยหลักการแล้วเรามีแนวทางการเขียน WBS อยู่หลายแบบแบ่งตามมุมมองของการเขียน เช่น

- เขียนโดยพิจารณาถึง Deliverables (software, hardware, documents,…)
- เขียนโดยพิจารณาถึง Life cycle phases (requirement, design, implement, …)
- เขียนโดยพิจารณาถึง Organizational responsibility (engineer, production, operation, …)
- หรือเขียนโดยพิจารณาถึง Geographical location (USA., Thailand, Japan,…)

สาหรับบทความชุดนี้ผมขอเลือกเป็นการเขียนโดยพิจารณาถึง Life cycle phases เพื่อให้ง่ายที่จะเข้าใจและความเหมาะสมกับตัวอย่างของเราครับ

ผมขอให้ข้อมูลเพิ่มเติมเกี่ยวกับองค์ประกอบสาคัญสี่อย่างที่จะทำให้ WBS ของเราถูกต้องและครบถ้วนก่อนครับ

Completeness

WBS ที่ดีต้องมีความครบถ้วน นั่นคือเราต้องเอา Major deliverable ทั้งหมดที่มีมาขยายความต่อว่าการที่เราจะทำให้ได้ตามนั้น เราต้องมี task อะไรบ้าง เช่น System prototype เราต้องทำอะไรบ้าง? Design user interface เอย Design report format ต่างๆเอย

นอกจากนั้นแล้วเราจำเป็นต้องคิดไปถึงงานอื่นๆที่ไม่ได้ถูกระบุใน Major deliverable แต่มีอยู่ใน Project life cycle (PLC) และ Software development life cycle (SDLC) ด้วย เช่น งานที่เกี่ยวกับการทำ Requirement งานที่เกี่ยวกับการทำ Project planning เป็นต้นครับ ถ้าเราแบ่ง WBS ให้เป็นไปตาม Phase ของ Project เราจะมองเห็นภาพได้ง่ายขึ้นครับ ลองดูตัวอย่าง WBS ของผมข้างล่างได้ครับ

Level

WBS ต้องมีความเป็นลำดับขั้นหรือ level ครับ นั่นคือเราต้องแบ่งงานที่ใหญ่ให้เป็นกลุ่มงานที่เล็กลง เช่น ถ้าเราพูดถึง task ที่ชื่อ 1.1 Produce requirement documents งานย่อยคืออะไรบ้างครับ? ผมคิดว่าน่าจะเป็น

1.1.1 Produce C-requirements

1.1.2 Produce D-requirement

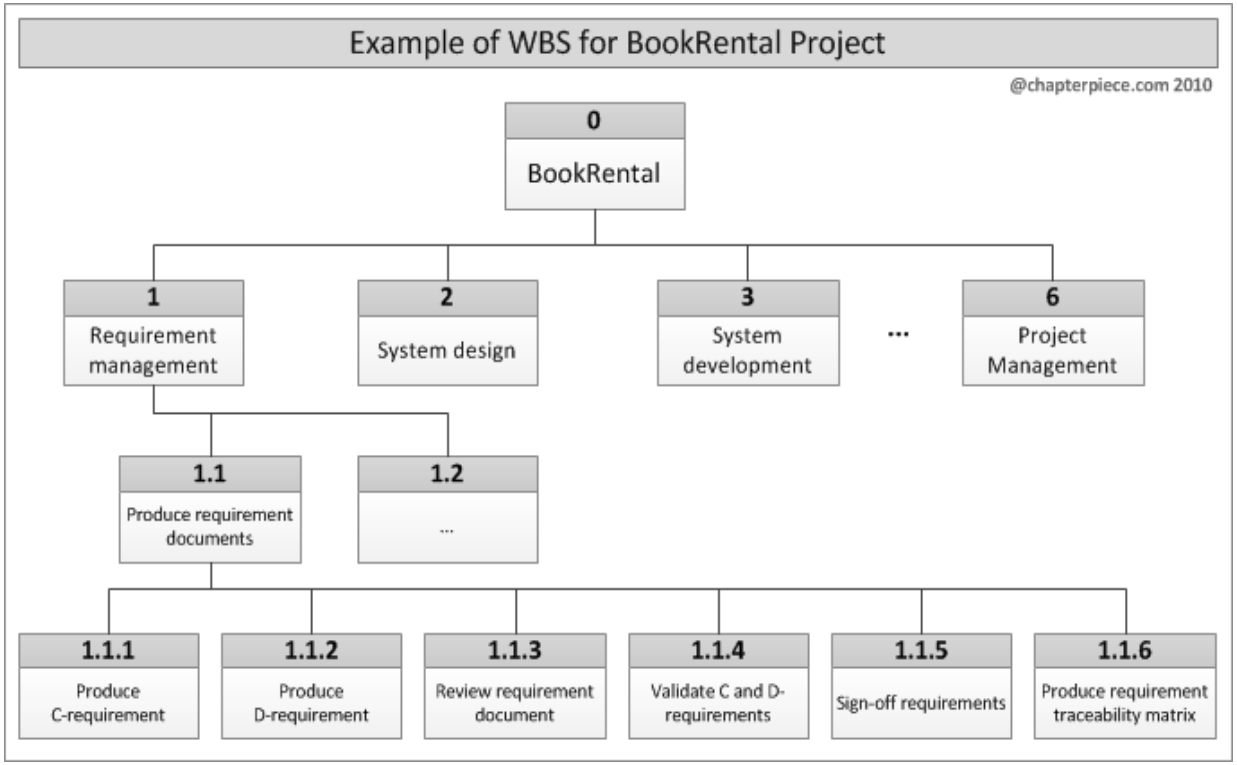
1.1.3 Review requirements document

1.1.4 Validate C and D-requirements

1.1.5 Sign-off requirements

1.1.6 Produce requirements traceability matrix

เพื่อนๆจะเห็นว่า 1 งานใหญ่แบ่งได้หลายๆงานย่อยที่เป็นลำดับขั้นกันลงมาครับ



Aggregation

อันนี้เป็นแนวคิดที่ต่อขยายมาจาก Level ครับ นั่นคือเวลาเราลงมือทำงานจริงๆ เราไม่ต้องทำงานที่เป็น level บน แต่เราทำ level ต่ำกว่าเพื่อให้ได้งานที่ระบุไว้ใน level ที่สูงกว่า งงปะครับ? ง่ายๆเลยครับ จากตัวอย่างเรื่อง Level ถ้าผมอยากได้ requirement documents ผมต้องทำยังไงครับ? ผมก็แค่ไปทำงานที่ 1.1 -1.7 ให้เสร็จผมก็ได้งาน 1 Produce requirement documents แล้วครับ กฎนี้แสดงให้เห็นว่าการทำงานกลุ่มย่อยๆผลลัพธ์ก็คืองานใหญ่ที่อยู่ในลำดับ ที่สูงกว่านั่นเอง

Not Sequence

WBS ด้วยตัวเองไม่สามารถบอกถึงลำดับของงานได้ครับดังนั้นเวลาเราเขียน WBS เราก็ไม่ต้องไปคิดถึงลำดับว่างานไหนต้องทำก่อนทำหลังให้เสียเวลาครับ เราพยายามคิดถึง task ที่ต้องมีมาให้ได้ครบถ้วนก่อน ส่วนเรื่องการจัดลำดับเก็บไว้ทำในส่วนที่เรียกว่า Network Diagram ครับ

เมื่อเราได้โครสร้างคร่าวๆของ WBS แล้วเราจะเอาไปเป็นส่วนประกอบหนึ่งในหนังสือสัญญาที่เราจะทำกับลูกค้า (contractual document) ด้วยก็ได้นะครับ เราจะเรียก WBS แบบนี้ว่า Contractual Work Breakdown Structure (CWBS) ครับ ถึงตรงนี้ Project Manager ก็ต้องเอาข้อมูลทั้งหมดนี้ไปคุยกับ Project Sponsor เพื่อตรวจสอบความถูกต้องอย่างละเอียดก่อนจะร่างหนังสือสัญญาฉบับเต็มที่ประกอบด้วย SOW, project specification, milestone schedule, และ CWBS ครับ ถ้าลูกค้าเซ็นเมื่อไรก็เป็นอันว่าได้เริ่มงานกันอย่างเป็นทางการแล้วหละครับ

กลับมาที่ WBS ต่อ หลังจากเตรียม task ครบแล้วจะถือว่า WBS เสร็จรึยัง? ก็ไม่เชิงนะครับ มี task แต่ไม่มีคนทำก็คงไม่ได้จริงมั้ย? หน้าที่ต่อไปของ Project Manager ก็คือใส่ชื่อผู้รับผิดชอบลงไปใน task ที่มีทั้งหมดครับ ตรงนี้มีประเด็นสาคัญอยู่ที่ประโยคนี้ครับ “Multiple owners = Zero owners” อธิบายว่าถ้าเรามอบหมายให้คนมากกว่าหนึ่งคนเป็นผู้รับผิดชอบงานนี้ มันก็ไม่ต่างอะไรกับเราไม่ได้มอบหมายงานให้ใครเลย เพราะว่าทุกๆคนก็จะคิดว่าไม่ต้องทำหรอก เดี๋ยวคนนั้นก็ทำให้เองแหละ แบบนี้ไม่ดีแน่นอนครับ ถ้าจะให้ถูกต้อง Project Manager ต้องมอบหมายให้คนแค่คนเดียวดูแลงานนั้นครับ (1 task/1 owner)

อีกนิดนึงกับเรื่อง Ownership ครับ อย่าสับสนว่าคนที่เป็นเจ้าของงาน (Owner) ต้องเป็นคนที่ทำ (Doer) นะครับ คนที่เป็น Owner คือคนที่รับผิดชอบ วางแผน จัดการงานนั้นซึ่งจะมอบหมายงานต่อให้คนอื่นทำก็ได้ตามความเหมาะสม แต่เจ้าตัวเองต้องรู้ความคืบหน้า ปัญหาที่เจอ และทุกๆอย่างที่เกี่ยวกับงานที่ตัวเองเป็นเจ้าของครับ ตัวอย่างเช่น Developer 1 ได้รับมอบหมายให้ทำ UI prototype ตอนนี้ Developer 1 เป็น Owner ของงาน แต่เนื่องจากตัวเองติดงานอื่นๆเยอะมากบวกกับที่เป็น Senior Developer 1 จึงมอบหมายงานต่อให้ Developer 2 ทำแทน (อย่าว่าใช้อำนาจบาตรใหญ่เลยนะ) ตอนนี้ Developer 2 ก็เป็น Doer ครับเป็นต้น

ถึงตรงนี้ WBS ที่เราได้ก็สมบูรณ์มากพอจะเริ่มขั้นตอนต่อไปแล้วครับ

Preliminary Schedule = Logical Relationship + Task Duration Estimates

ในเมื่อ WBS ไม่ได้บอกเราถึงลำดับการทำงาน เราก็เลยต้องมีขั้นตอนที่จัดลำดับการทำงานกันครับ งานส่วนใหญ่มีความเกี่ยวข้องกัน

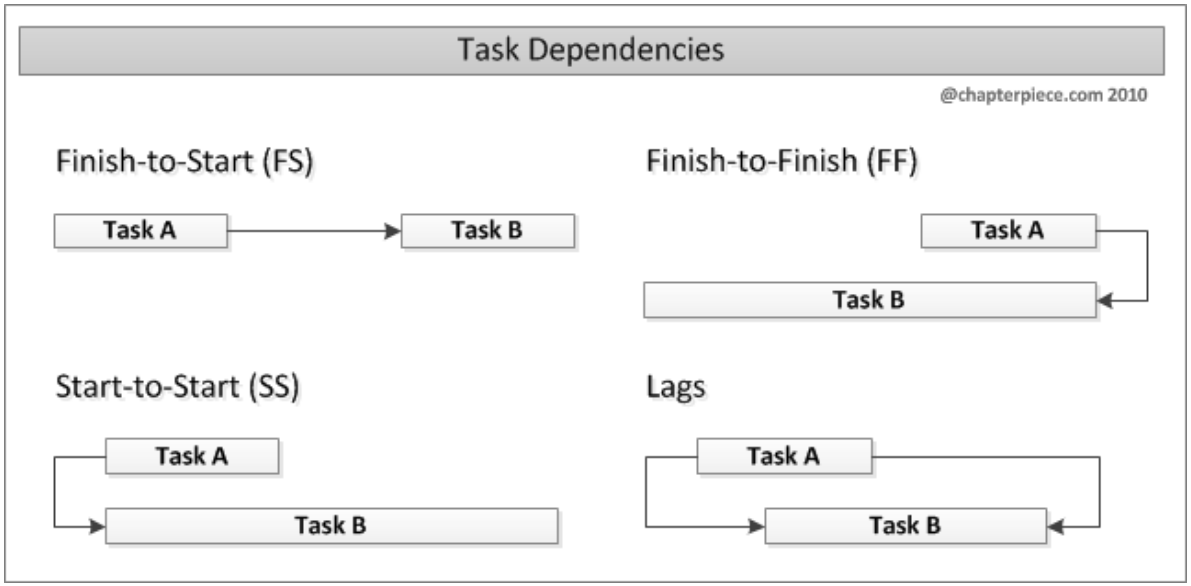
กับงานอื่นๆทั้งนั้น ไม่ว่าจะเป็นแบบที่ต้องรอให้งานแรกเสร็จก่อนถึงจะเริ่มงานนี้ได้หรือแบบที่ต้องรอให้งานแรกเริ่มก่อนถึงจะเริ่มงานนี้แบบคู่ขนานกันไปได้ Project Manager ต้องจัดการแปลง WBS ให้มาเป็นสิ่งที่เราเรียกว่า Network diagram หรือ Dependency diagram ขั้นตอนนี้เราเรียกว่าการทำ Logical relationship ครับ

ก่อนอื่นเรามารู้จักความสัมพันธ์ระหว่างงานมีอยู่กันก่อนครับ

- Finish-to-start (FS): ความสัมพันธ์คืองานหน้าต้องเสร็จสมบูรณ์ 100% ก่อนงานหลังจะเริ่มต้นได้ เช่น Integration test จะเริ่มได้ก็ต่อเมื่อ installation ทั้งระบบเรียบร้อยแล้ว
- Start-to-start (SS): ความสัมพันธ์คืองานหลังจะเริ่มได้ก็ต่อเมื่องานหน้าต้องเริ่มด้วย เช่น การเตรียมเขียนเอกสารของระบบจะเริ่มต้นได้ทันทีที่ Project เริ่ม
- Finish-to-finish (FF): ความสัมพันธ์คืองานหลังจะเสร็จได้ก็ต่อเมื่องานหน้าต้องเสร็จสมบูรณ์ด้วย เช่น System test จะเสร็จได้ก็ต่อเมื่อ fix bug ทั้งหมดเรียบร้อยแล้ว
- Lags: ความสัมพันธ์คือการเหลื่อมกันของงานสองงาน เช่น System test ต้องเริ่มหลัง coding ไปแล้ว 1 เดือน

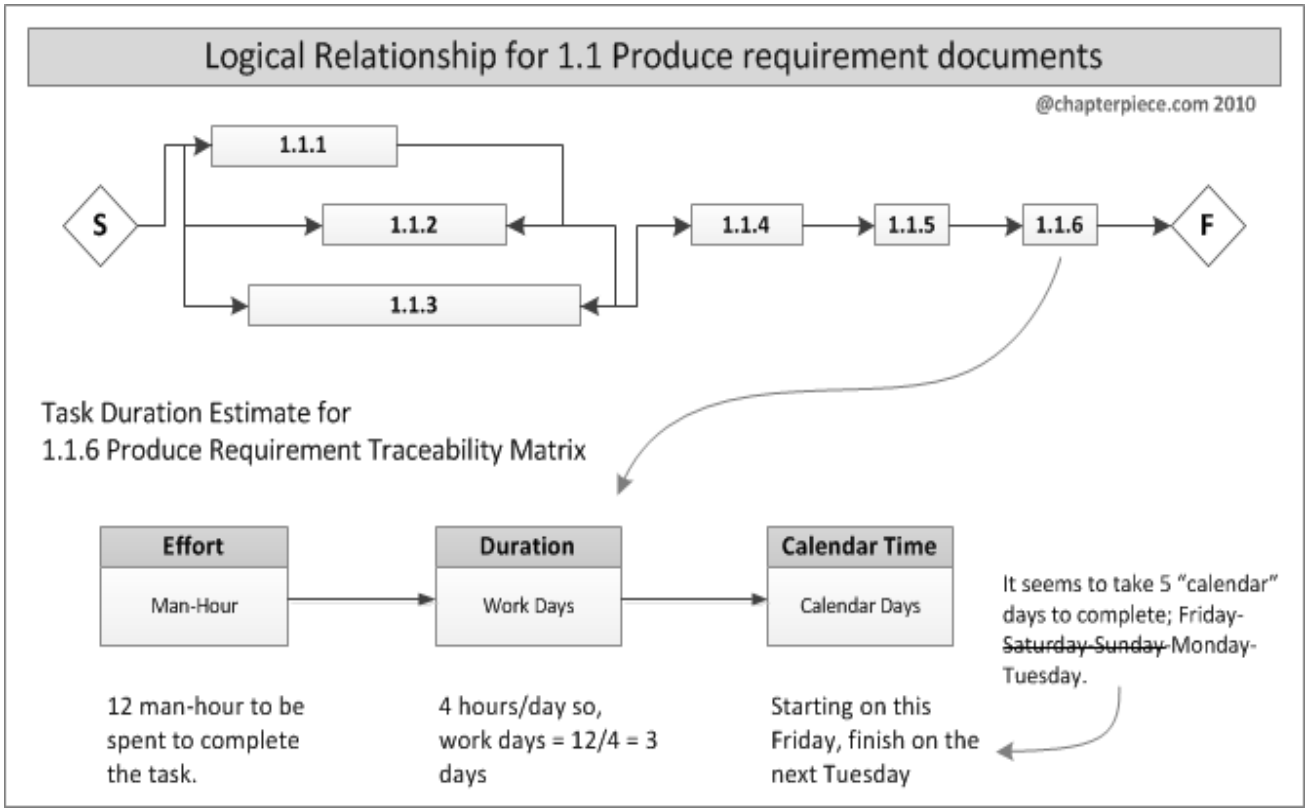
ทีนี้เราก็มาจัดลำดับของงานทั้งหมดที่มีใน Project ครับว่าอะไรควรมาก่อนมาหลัง มีความสัมพันธ์ในรูปแบบไหน จาก WBS แบนๆก็จะมีมิติขึ้นมาแล้วครับ

เอาล่ะ เรามี WBS แล้วเรามี Logical relationship แล้ว แต่เรายังไม่รู้เลยว่างานไหนจะเริ่มเมื่อไรจะเสร็จเมื่อไร แล้ว Project จะไปเสร็จวันไหน? ครับ ผมกำลังจะชวนให้เพื่อนมาทำขั้นตอนต่อไปของ Preliminary schedule นั่นก็คือ Task duration estimates พูดภาษาชาวบ้านหน่อยก็มาประเมิน (Estimate) เวลาที่จะใช้ในแต่ละงานนั่นแหละครับ



ขั้นตอนนี้ต้องอาศัยข้อมูลจากสมาชิกในทีมทุกคนรวมกับประสบการณ์ของ Project Manager เพื่อที่จะหาตัวเลขที่ใกล้เคียงที่สุดในการทำงานแต่ละงานครับ โดยทั่วไปแล้วการประเมินเวลาจะเริ่มที่การมองว่างานนี้ถ้าทำ 1 คนใช้เวลากี่ชั่วโมง เราเรียกตัวเลขนี้ว่า Effort มีหน่วยเป็นคนต่อชั่วโมงหรือ Man-hour ครับ จากนั้นก็แปลงตัวเลขนี้มาเป็นจำนวนวันที่ทำงานจริงที่เรียกว่า Work days ครับ จากนั้นสุดท้ายก็แปลงมาเป็นวันที่ต้องทำงานตามปฏิทินหรือ Calendar days หลายต่อจัง งงป่าวครับเนี่ยะ ผมยกตัวอย่างจากรูปแล้วกันฮะ

จากรูปจะเห็นว่า task ที่ชื่อ 1.1.6 Produce requirement traceability matrix ผมเป็นเจ้าของ (Owner) และคนทำ (Doer) ผมคิดว่าผมใช้เวลาทำ 12 ชั่วโมง ตอนนี้ผมได้ Effort มาแล้วเป็น 12 Man-hour ผมคิดว่าวันนึงผมทำงานกับ Project นี้ 4 ชั่วโมง นั่นคือผมต้องใช้เวลา 3 วันกว่าจะทำงานนี้เสร็จนี่คือ Work days ของผม สุดท้ายงานนี้ดันต้องมาเริ่มวันศุกร์นี้ ถามว่างานจะเสร็จวันจันทร์รึเปล่าครับ? ไม่ใช่ล่ะ เสาร์-อาทิตย์วันหยุดนะ มันต้องนับแบบนี้ครับ ศุกร์ – จันทร์ – อังคาร สรุปว่าผมจะทำงานเสร็จวันอังคารหน้าครับ นี่คือ Calendar days ของผมครับ



การประเมินเวลาที่ใช้ในแต่ละงานนั้นที่ผมบอกว่าต้องใช้ประสบการณ์ของ Project Manager ด้วยส่วนหนึ่งก็เพราะว่าเราต้องเผื่อเหลือเผื่อขาดไว้ให้กับความสามารถและประสบการณ์ของคนทำงานนั้นๆ เวลาที่จะเสียไปกับการรอนู่นรอนี่ เช่น ยังไม่ได้คำตอบเรื่องนั่นเรื่องนี้จากลูกค้า เป็นต้น บวกด้วยเวลาที่จะเสียไปกับเรื่อง process ต่างๆโดยเฉพาะกับ CMMi (แหะๆ) และวันลาป่วยวันลาพักร้อนของสมาชิกในทีมด้วย ถ้า Project Manager ที่มีประสบการณ์และรู้จักสมาชิกในทีมดีก็จะประเมินเวลาออกมาได้ตรงกับความเป็นจริงมากขึ้นครับ

อีกเรื่องคือไม่ต้องไปเสียเวลากับเรื่องนี้ให้มากนะครับ จากประสบการณ์ครับให้เอาจริงเอาจังและพยายามยังไงก็ตาม ไม่มีทางที่จะได้เวลาที่ตรงออกมาหรอก เสียเวลาเปล่าๆ ทำให้อยู่ในระดับที่ทั้งทีมมองว่าโอเคแล้วกับตัวเลขนี้ก็พอครับ เมื่อเราทำงานไปเรื่อยๆแล้ว

เห็นอะไรมากขึ้น เราค่อยกลับมาปรับตัวเลขปรับแผนได้ตลอดอยู่แล้วครับ

หลังจากนี้ Project Manager ก็เอา WBS ตัวสมบูรณ์ และ Network diagram ไปคุยกับ Project Sponsor อีกทีเพื่อขออนุมัติการเริ่มงาน Project Sponsor ก็จะพิจารณาดูภาพรวมของงานว่าอยู่ในเวลาที่กำหนด งบประมาณที่กำหนด มีแผนการรองรับข้อผิดพลาดที่อาจจะเกิดขึ้นแล้วรึยัง เป็นต้น ถ้าได้รับอนุมัติผมก็ยินดีด้วยครับ ถึงตรงนี้เราก็ได้ Project Plan ที่ค่อนข้างสมบูรณ์มากแล้วครับ พร้อมที่จะเริ่มทำงานตามนี้ได้เลย อ้อๆ Project Manager ต้องไม่ลืมขอความคิดเห็นจาก Project Sponsor เรื่องความสาคัญของ Scope, Schedule และ Resources มาทำ Flexibility matrix แล้วก็เตรียมการสาหรับ Stakeholder management ด้วยนะครับ สาคัญมาก

Other Tasks for Project Planning

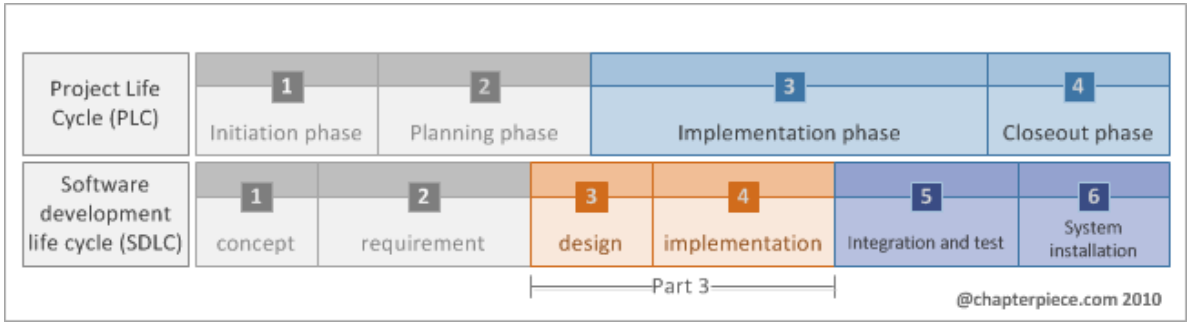
ได้แต่ Project plan ยังไม่ถือว่าทำ Planning phase จบนะครับ (อะไรมันจะมากมายขนาดนี้) เราจำเป็นต้องพิจารณาถึง Risk management เรื่อง Change management แล้วก็เรื่องการวางกฎเกณฑ์และนโยบาย (Rules and policies) ต่างๆที่จะใช้ใน Project ของเราด้วยครับ ผมขออนุญาตไม่พูดถึงในบทความนี้แล้วกันนะฮะ เดี๋ยวมันจะยาวจนอ่านกันไม่ไหวซะก่อน ถ้าเพื่อนๆคนไหนสนใจเป็นพิเศษ ติดต่อผมมาได้ครับ ผมจะหารายละเอียดมาให้

สรุปภาคสอง

จบภาคสองแล้วฮะ จากขั้นตอนนี้เราได้ Project plan ที่ค่อนข้างสมบูรณ์มาหนึ่งฉบับก็พอจะสรุปได้ว่าขั้นตอนแรกๆของการทำ Software development project ทั้ง Requirement management และ Project planning ได้ผ่านไปแล้วนะครับ บทความหน้าเราจะมาเริ่มงานที่เป็น Detailed design กับ Implementation (coding) กันอย่างเป็นทางการครับ

3 การออกแบบและพัฒนาซอฟต์แวร์

บทความนี้กลับมาต่อเรื่องที่เป็น Software development life cycle กันนะครับ หลังจากที่เราได้ Requirement ของลูกค้าที่ละเอียดมาแล้ว เราก็จะมาเริ่มกระบวนการออกแบบระบบอย่างเต็มตัวครับ งานนี้แบ่งออกเป็นสองส่วนก็คือ Architectural design กับ Detailed design



กระบวนการนี้จะเป็นหน้าที่รับผิดชอบร่วมกันของ Developer กับ Business Analyst (หรือ System Analyst) ครับ เราลองมาดูรายละเอียดกันเลย

Software Development Life Cycle – Design

Architectural Design

หลักใหญ่ใจความของกระบวนการนี้ก็คือการออกแบบโครงสร้างของระบบก็เหมือนการสร้างบ้านครับ ถ้าไม่สร้างฐานก่อนส่วนที่เหลือก็ไม่ต้องพูดถึงกันหละ ระบบซอฟต์แวร์ก็ไม่ต่างกัน การออกแบบโครงสร้างของเรานั้นจะแบ่งได้หลายมุมมองครับ แต่จากประสบการณ์ของผมเราควรพิจารณาสามมุมมองใหญ่ๆดังนี้

System Context Diagram (SCD)

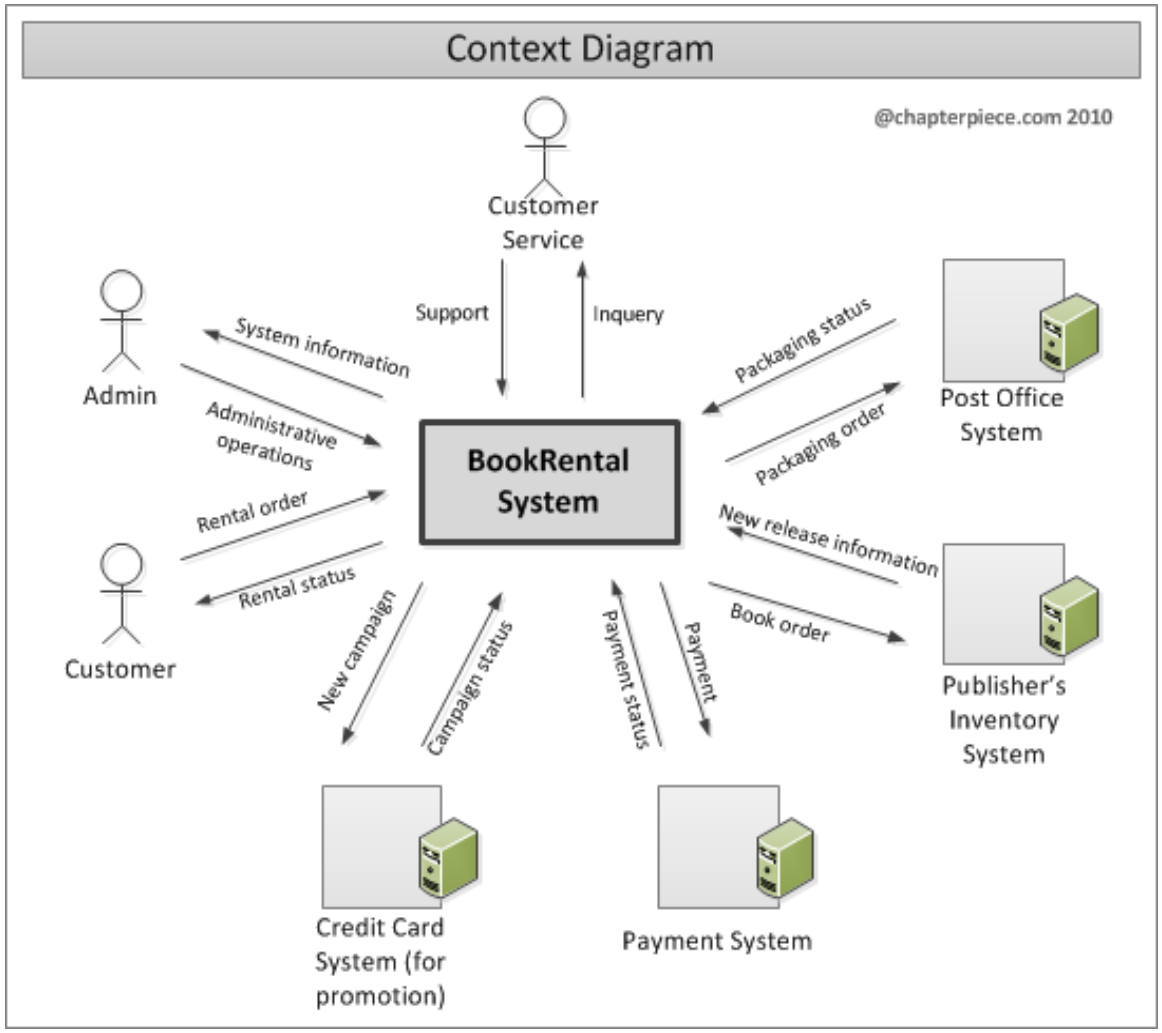
System context diagram เป็นโครงสร้างระบบที่แสดงถึงความสัมพันธ์ระหว่างระบบกับส่วนประกอบอื่นๆภายนอก (Actor) เช่น ผู้ใช้และระบบอื่นๆ ประโยชน์ของมันก็คือช่วยให้เรามองเห็นภาพรวมว่าระบบที่เรากำลังจะสร้างนั้นเป็นส่วนหนึ่งของอะไรบ้าง (อะไรที่ใหญ่กว่าตัวระบบเอง) ทำให้เรามองเห็นว่าระบบของเราต้องทำงานเพื่อตอบสนองอะไรให้กับใครบ้างครับ

แล้ว SCD สร้างขึ้นมาได้อย่างไร? จำ Use case ที่เราเขียนขึ้นมาใน Requirement phase ได้มั้ยครับ? นั่นแหละเป็นข้อมูลอย่างดีที่ช่วยเราเขียน SCD ครับ จาก Use case เราจะเห็นถึงความสัมพันธ์ของระบบกับส่วนประกอบภายนอก ในขณะเดียวกัน Data flow ต่างๆก็แสดงให้เห็นถึงการรับส่งข้อมูลต่างๆที่เกิดขึ้นระหว่างระบบกับส่วนประกอบภายนอกอีกด้วย

ขั้นตอนการเขียน SCD ก็ไม่ยากครับ

1. เขียนระบบเราอยู่ตรงกลางของภาพ
2. เขียนส่วนประกอบภายนอกทั้งหมดอยู่ในกล่องซึ่งวางอยู่รอบๆระบบของเรา (Use case)
3. เขียนเส้นเชื่อมต่อระหว่างระบบกับส่วนประกอบภายนอกเพื่อแสดงถึงข้อมูลที่ไหลผ่าน (Data flow)

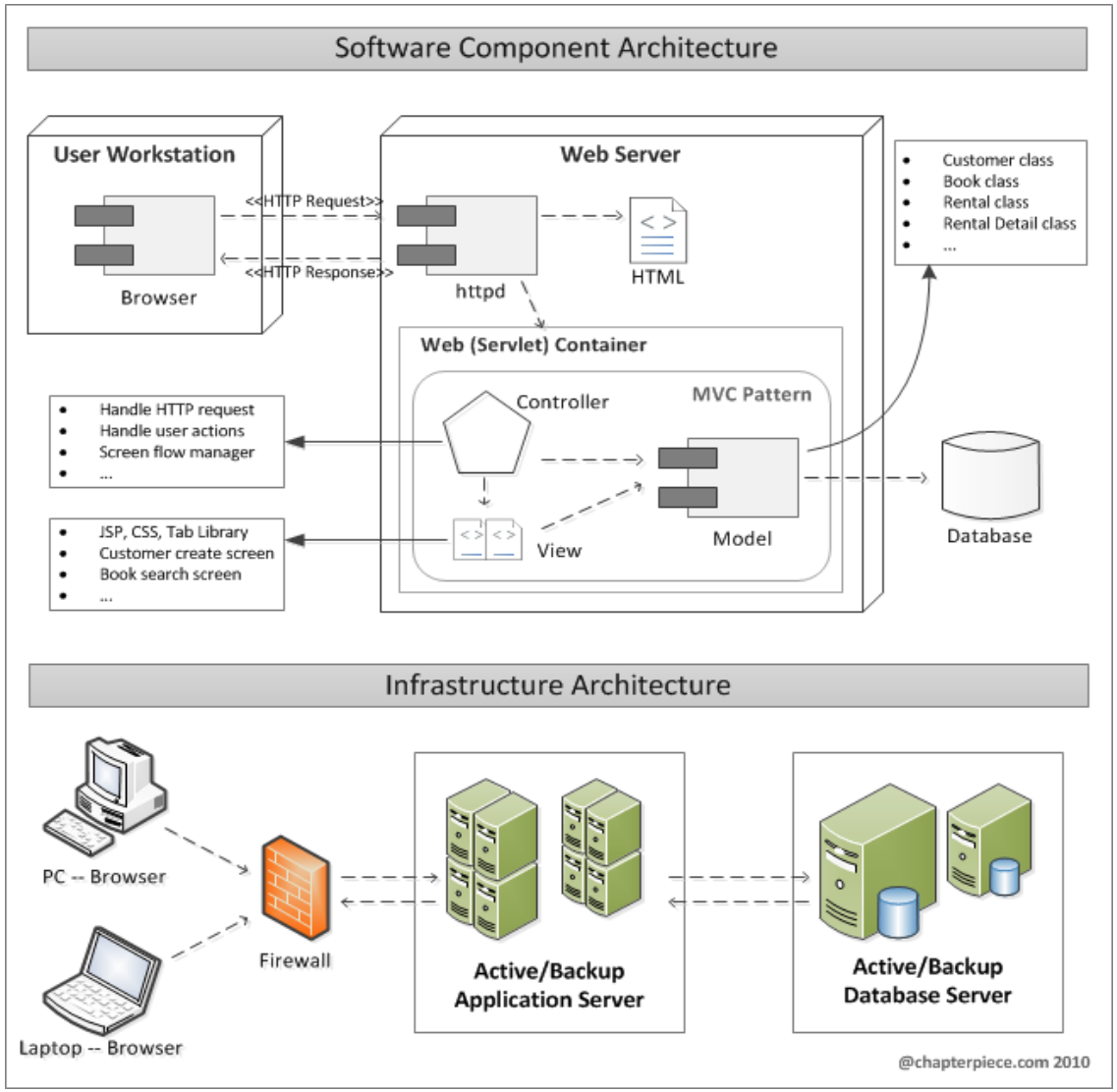
ผมเขียนตัวอย่าง SCD คร่าวๆของระบบเราไว้ข้างล่างครับ คิดว่าพอจะดูเป็นตัวอย่างได้



ตามหลักการแล้ว SCD คือ Data flow diagram ระดับสูงสุด (level 0) ครับ เราสามารถเขียน level ที่ต่ำกว่า (ละเอียดกว่า) ได้ซึ่งจะทำให้เรารู้ว่า Data flow ทั้งหมดของระบบเราเป็นอย่างไร และช่วยในการออกแบบโครงสร้างฐานข้อมูล (Database schema) ไปในตัวด้วยครับ

Software Component Architecture

การเขียน Software component architecture จะช่วยให้เรามองเห็นถึงโครงสร้างของระบบในมุมมองที่เป็นส่วนประกอบต่างๆของซอฟต์แวร์และความสัมพันธ์ระหว่างส่วนประกอบนั้นๆครับ



ระบบเช่าหนังสือออนไลน์ของเราเป็น Web application ที่พัฒนาด้วย Java technology ดังนั้นผมจะออกแบบโครงสร้างโดยใช้หลักการ MVC pattern นะครับ ลองดูตัวอย่างข้างล่างครับ

เมื่อดูจากรูป เราจะเห็นโครงสร้างโดยรวมและงานที่เราต้องทำในส่วนของซอฟต์แวร์จะมีอะไรบ้าง ดูได้ชัดๆจากส่วนที่อยู่ใน Web container เราต้องเขียน Object classes (Model) เพื่อทำหน้าที่เป็นตัวแทนของข้อมูลประเภทต่างๆที่มีใน Database รวมถึงงานที่เป็น Business rule ต่างๆ

ในส่วนของการแสดงผลผ่านทาง JSP, CSS และ Tag library (View) นั้นเราก็จะมองเห็นภาพโดยรวมว่าเราต้องมี JSP กี่ไฟล์เพื่อตอบสนอง Requirement ให้ได้ครบ เราต้องวางโครงสร้างของ CSS เพื่อการ Reuse อย่างไร เราจำเป็นต้องทำ Tag library เพื่อช่วยในเรื่อง Security อย่างไร เป็นต้นครับ

Infrastructure Architecture

เสร็จในส่วนซอฟต์แวร์เราก็มาออกแบบโครงสร้างของฮาร์ดแวร์กันต่อเลยครับ โครงสร้างในมุมมองนี้จะบอกเราว่าเพื่อจะใช้งานระบบเราได้อย่างสมบูรณ์เราต้องการ Infrastructure หรือ Environment อย่างไรครับ

การออกแบบ Infrastructure นั้น เราต้องพิจารณาถึง Nonfunctional requirements ด้วยเพื่อที่จะตอบสนองสิ่งต่างๆที่ระบุไว้ได้อย่างครบถ้วน เช่น ตัวอย่างของเรามีการระบุเรื่องของ Performance Reliability และ Availability เอาไว้อย่างชัดเจน เราก็ต้องเตรียมการรองรับไว้ให้ดี เช่น เราต้องมี Active และ Backup server สำหรับ Web และ Database เราต้องมี Firewall มาช่วยในเรื่องของ Security เป็นต้นครับ

ประโยชน์อีกอย่างของการเขียน Infrastructure architecture ก็คือเราจะได้เตรียม Infrastructure ที่ถูกต้องในการติดตั้งและทดสอบระบบของเราในช่วง Integration and system test ซึ่งผมจะลงรายละเอียดในบทความหลังๆครับ ดูตัวอย่างได้จากรูปข้างบนนะครับ

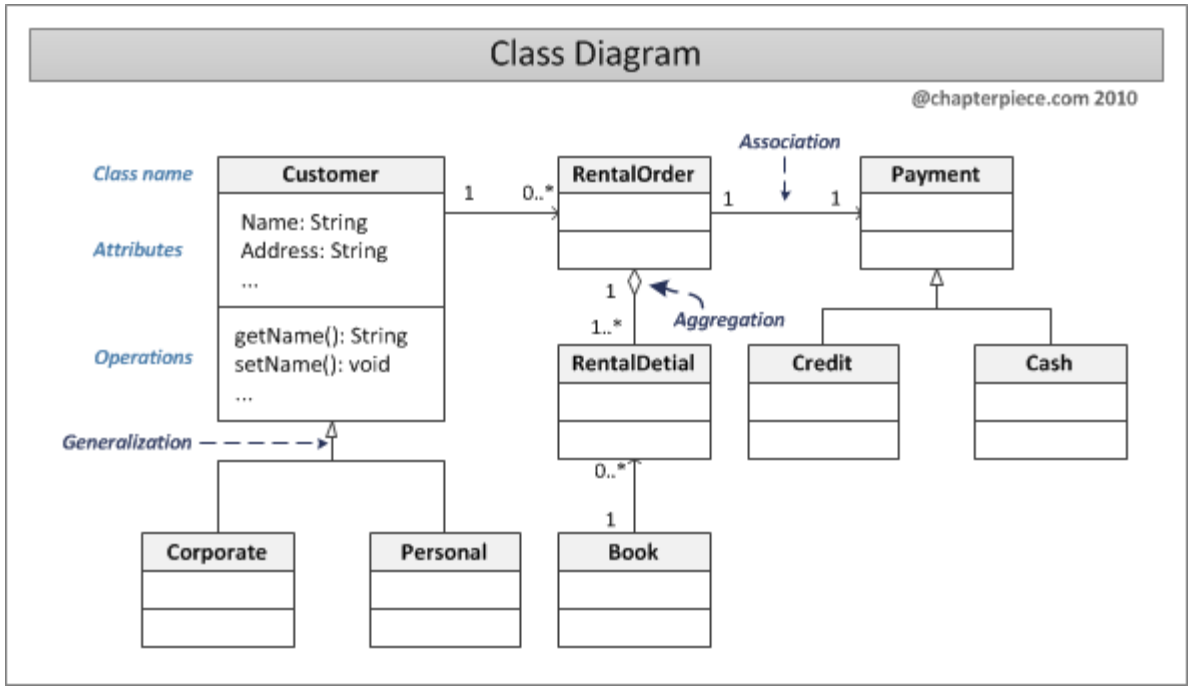
Detailed Design

หลังจากที่ได้โครงสร้าง พื้นฐาน (System architecture) แล้ว ถัดมาก็คือการออกแบบระบบในระดับที่ลึกลงไปอีกครับ ผลลัพธ์ที่ได้จากกระบวนการนี้จะเป็นแนวทางอย่างดีสำหรับ Developer ในการเขียนโค๊ด

Class Diagram

สำหรับการเขียนโปรแกรมแบบ Object Oriented Programming (OOP) เราหลีกเลี่ยงการเขียน Class diagram ไม่ได้เลยครับ (ถึงแม้จะเขียนยากก็ตาม) เพราะว่าเจ้า Diagram นี้จะบอกเราถึง Class ทั้งหมดที่ต้องมี สิ่งที่ Class นั้นทำได้ (Attributes and operation) และความสัมพันธ์ระหว่าง Class (Relationship) ซึ่งเป็นสิ่งที่จำเป็นมากในการเขียนโปรแกรมครับ

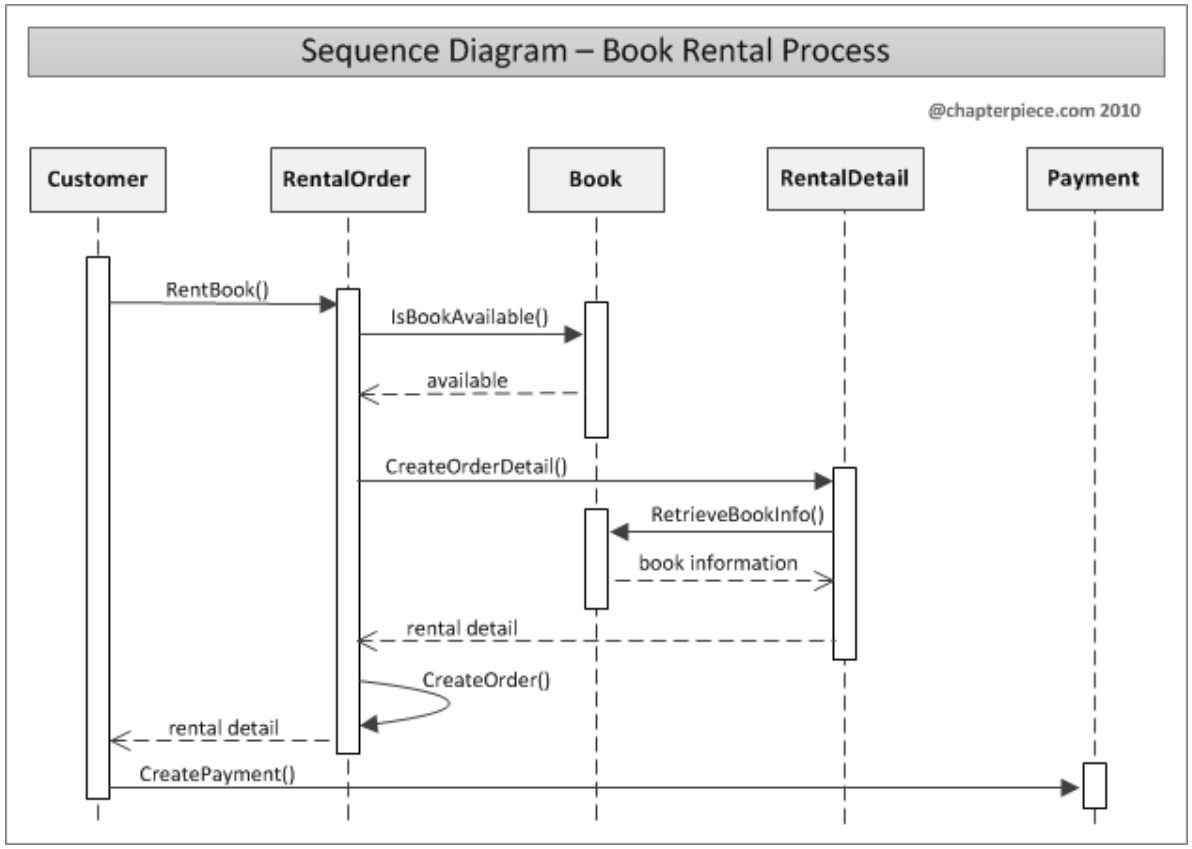
ข้อมูลเบื้องต้นที่เราใช้ในการเขียน Class diagram ก็มาจากหลายที่เช่น D-requirements ทั้งหลายแหล รวมถึง Software architectural design ด้วย ตัวอย่างก็ดูได้จากรูปข้างล่างครับ



ถ้า Developer มาเห็น Class diagram นี้ก็จะรู้ทันทีว่า อ้อ เค้าต้องสร้าง Class แม่ที่ชื่อ Customer ขึ้นมา แล้วตามด้วย Class ลูกที่ชื่อ Corporate Customer กับ Personal Customer นะ ถัดมาก็ต้องเตรียม Class ที่ชื่อ RentalOrder ที่ใช้เป็นตัวแทนของการเช่าหนังสือต่อด้วยการเชื่อมความสัมพันธ์ระหว่าง Class ต่างๆที่สร้างขึ้นมา เป็นต้น

Sequence Diagram

มีแค่ Class diagram เราก็ไม่รู้อยู่ดีว่าเมื่อถึงเวลาทำงานจริงๆ มันมีลำดับการเรียกสวนต่างๆในระบบของเราอย่างไรใช่ปะครับ เราแก้ปัญหานี้ได้ด้วย Sequence diagram ครับ หลักการที่สมบูรณ์เลยก็คือ Class diagram จะบอกความสัมพันธ์ระหว่าง Class ในขณะที่ Sequence diagram จะแสดงให้เห็นถึงข้อความ (message) ที่ส่งต่อกันระหว่าง Class ต่างๆแบบเป็นลำดับ



จากรูปข้างบน ผมยกตัวอย่าง Sequence diagram ของกระบวนการเชาหนังสือโดยลูกค้าครับ เพื่อนๆจะเห็นว่ารูปจะแสดงให้เห็นอย่างชัดเจนเลยว่ากระบวนการเริ่มจากใคร เรียกฟังก์ชั่นอะไรในการทำงานแต่ละขั้นตอน รวมไปถึงข้อมูลหรือข้อความที่ส่งผ่านกันระหว่าง Class ด้วย มีประโยชน์มาทีเดียวสาหรับ Sequence diagram ครับ Developer เห็นแล้วจะเขียนโค๊ดง่ายขึ้นเยอะมากๆ

Flowchart

ครับ รู้ Class ที่ต้องมีรวมถึง Attributes กับ Operations ของ Class ก็แล้ว รู้ลำดับการเรียกและส่งรับข้อมูลระหว่าง Class ในการทำงานอะไรซักอย่างก็แล้ว Developer ก็ยังไม่มั่นใจในการเขียนโค๊ดแบบเต็ม 100 เท่าไร จนกว่าจะรู้ว่าแต่ละส่วน (Function/Method) จะมี Algorithm อย่างไร เพื่อนๆคงจะเห็นนะครับว่าทั้ง Class diagram และ Sequence diagram บอกเราไม่ได้เลย ทำยังไงดีหละทีนี้ ... Flowchart ช่วยเราได้ครับ

Flowchart เป็นหนึ่งใน Diagram ที่เก่าแก่ที่สุดที่ใช้อธิบายถึง Algorithm เลยทีเดียวครับ ผมค่อนข้างมั่นใจว่าเพื่อนๆน่าจะคุ้นเคยกับเจ้านี่อยู่พอสมควร การเขียนก็ไม่ได้ยากอะไรมากครับ แต่รายละเอียดเยอะ ผมขอไม่พูดถึงในบทความนี้แล้วกันนะฮะ ถ้าอยากรู้อะไรเพิ่มเติม ตามไปที่นี่เลยครับ

สำหรับผมแล้วไม่มีอะไรที่จะละเอียดกว่า Flowchart แล้วหละครับ อันนี้คือสุดๆแล้วจริงๆสำหรับการออกแบบระบบ ข้อแนะนำที่มีก็คือ ละเอียดมากก็ใช้เวลาทำมาก เราเลือกใช้ Flowchart กับเฉพาะส่วนงานที่มี Algorithm ซับซ้อนจริงๆจะดีที่สุดครับ เหลือเวลาไว้เขียนโค๊ดบ้าง

Design Review

เฮ้อ กว่าจะออกแบบเสร็จ เหนื่อยครับขั้นตอนนี้ ข่าวร้ายคือยังไม่เสร็จ ฮ่าๆ ผมอยากแนะนำว่าเมื่อออกแบบหรือเขียน Diagram อะไรต่างๆเสร็จแล้ว เราควรจะให้คนในทีมช่วยกันตรวจสอบความถูกต้องอีกครั้งก่อนเอาไปใช้จริงด้วย นะครับ ป้องกันความผิดพลาดที่อาจจะเกิดขึ้น เช่น Design ไม่ตรงตาม Requirement หรือลืม Design ให้กับบาง Requirement (อันนี้แย่กว่า) อ้อ อีกอันก็คือ Design ผิด

เสียเวลาอีกนิดตรงนี้ดีกว่าเขียนโค๊ดไปครึ่งทางแล้วมาเจอที่หลังว่า Design ใช้ไม่ได้นะครับ อันนั้นเรื่องยาวเลยหละ

Software Development Life Cycle – Implementation

หลังจากที่เราออกแบบระบบอย่างละเอียดเรียบร้อยแล้ว เราก็ได้ฤกษ์เขียนโค๊ดกันซักทีครับ สำหรับกระบวนการนี้ผมขอไม่ลงในรายละเอียดนะครับเพราะว่ารายละเอียดเยอะมาก และเพื่อนๆแต่ละคนก็มีสไตล์การเขียนโค๊ดไม่เหมือนกัน อีกทั้งผมเข้าใจว่ากระบวนการนี้

เพื่อนๆมีความคุ้นเคยมากอยู่แล้วครับ

Software Development Life Cycle — Unit Testing

หลังจากเขียนโค๊ดเสร็จแล้ว เราก็ต้องทดสอบว่าโค๊ดที่เขียนออกมาทำงานได้ถูกต้องหรือไม่ครับ ทำไมเราต้องทำการทดสอบหลังจากเขียนโค๊ดเสร็จทันทีด้วยหละ? เหตุผลก็เพราะว่าความร้ายแรงของข้อผิดพลาด (Defect) และความยากในการค้นหาและแก้ไขข้อผิดพลาดนั้นจะเพิ่มขึ้นตลอดเวลาที่ระบบหรือโค๊ดของเราใหญ่ขึ้น เช่น มีโอกาสที่จะเจอข้อผิดพลาดจากโค๊ด 100 บรรทัดมากกว่าโค๊ด 10 บรรทัด และเนื่องจากความซับซ้อนที่เพิ่มขึ้นตามขนาดของโค๊ด การตรวจสอบและแก้ข้อผิดพลาดในโค๊ด 100 บรรทัดก็ยากกว่าโค๊ด 10 บรรทัดตามไปด้วยครับ ดังนั้นเราควรที่จะทดสอบความถูกต้องของโค๊ดเราให้เร็วที่สุดและบ่อยที่สุด นี่คือที่มาของ Unit test ซึ่งเป็นกระบวนการทดสอบคู่กับการพัฒนาระบบนั่นเองครับ

Unit test เป็นการทดสอบส่วนย่อยของระบบแบบยากจากกัน คำว่า “ส่วนย่อย” นั้นตีความหมายได้หลายแบบครับ เช่น functions, procedures, หรือ methods ก็ได้แล้วแต่ความเหมาะสมของงาน

วัตถุประสงค์ของการทำ Unit test หลักๆเลยก็คือ ตรวจสอบว่าโค๊ดเราทำงานได้ถูกต้องหรือไม่ (มีความแตกต่างระหว่างผลลัพธ์กับสิ่งที่เราอยากให้ระบบหรือโค๊ดเป็นรึเปล่า) ซึ่งการทำ Unit test นั้นจะต้องแยกทำเป็นส่วนๆ (Unit) จริงๆครับ เช่น ถ้าเราอยากรู้ว่าส่วนการเช่าหนังสือถูกต้องรึเปล่า เราก็ต้องทดสอบเฉพาะ Method ที่ชื่อ RentBook เท่านั้น

โดยหน้าที่แล้ว Developer จะเป็นคนรับผิดชอบทำ Unit test ครับ ซึ่งถ้าอยากจะให้ผลออกมาดี เราก็ควรกำหนดให้ Unit test ต้องถูกทำโดย Developer ที่ไม่ได้เขียนโค๊ดส่วนนั้น เพื่อเป็นการเปลี่ยนมุมมองในการทดสอบ นั่นรวมถึงความโปร่งใสด้วย

เทคนิคที่ใช้กันมากในการทำ Unit testing เราเรียกว่าการทำ White box testing ครับ

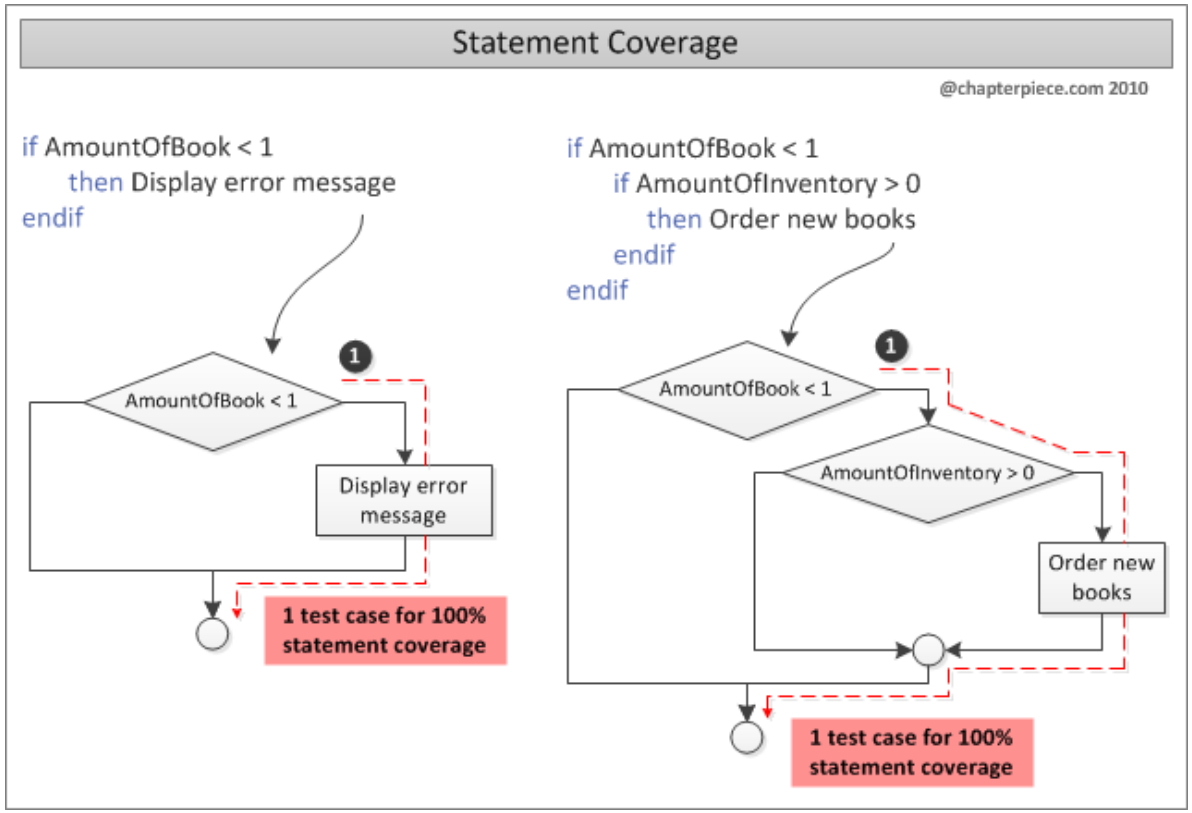
White Box Testing

ความหมายของ White box testing ก็คือการทดสอบทุกๆเส้นทางที่เป็นไปได้ของโค๊ดเราครับ คนที่จะทำ White box testing ได้นั้นต้องเป็นคนที่รู้โครงสร้างภายในของระบบหรือโค๊ดดีพอสมควรจึงจะสามารถมองเห็นเส้นทางที่เป็นไปได้ทั้งหมดของการทดสอบโค๊ดในส่วนนั้นๆ ไม่บอกก็รู้ว่างานนี้ต้องเป็นของ Developer ใช่ปะครับ

จริงๆแล้วมีเทคนิคอยู่มากมายที่ใช้ในการทำ White box testing แต่ในบทความนี้ผมของแนะนำแค่สองเทคนิคที่ผมมองว่าจะได้ใช้บ่อยที่สุดนะครับ

Statement Coverage

เทคนิคนี้คือการเขียนชุดทดสอบ (Test case) ที่ทุกๆบรรทัดของโค๊ดได้รับการทดสอบ (Execute) เราสามารถประเมินความครอบคลุมของการทดสอบ (Test coverage) ได้ออกมาเป็นเปอร์เซ็นต์ครับ เช่น ถ้าทุกบรรทัดของโค๊ดได้รับการทดสอบทั้งหมด เราจะเรียกว่า 100% coverage แล้วก็ลดหลั่นกันลงไปตามอัตราส่วน ปัจจุบันนี้มีเครื่องมือ (plugin) มากมายที่จะช่วยเราในเรื่องการทำ Test coverage ลองดูได้ที่นี่ครับ



โดยปกติแล้ว เราไม่จำเป็นต้องทดสอบให้ได้ 100% coverage หรอกครับมันเป็นงานที่หนักเกินไป ทางที่ดีเราควรกำหมดเป้าหมายไว้ก่อนทดสอบว่าสาหรับ Project นี้เราต้องการ Test coverage กี่เปอร์เซ็นต์แล้วพยายามทำให้ได้ตามเป้าหมายนั้นก็พอ ตรงนี้เป็นเรื่องของ Quality management ซึ่งเป็นข้อตกลงร่วมกันระหว่างสมาชิกทุกคนในทีมโดยมี Project Manager เป็นคนอนุมัติ ร่ายยาวเลย เราลองมาดูตัวอย่างของจริงกันดีกว่าครับว่า Statement coverage เป็นยังไง

จากรูปจะเห็นได้ว่า โค๊ดทางซ้ายมือเราต้องมี 1 Test case เพื่อที่จะทดสอบโค๊ดทั้งสามบรรทัดนี้ได้ครับ โดย Test case ของเราจะต้องทำให้เงื่อนไข if AmountOfBook < 1 เป็นจริงเพื่อ error message จะได้ถูกแสดงออกมา แล้วก็ปิดท้ายด้วย endif

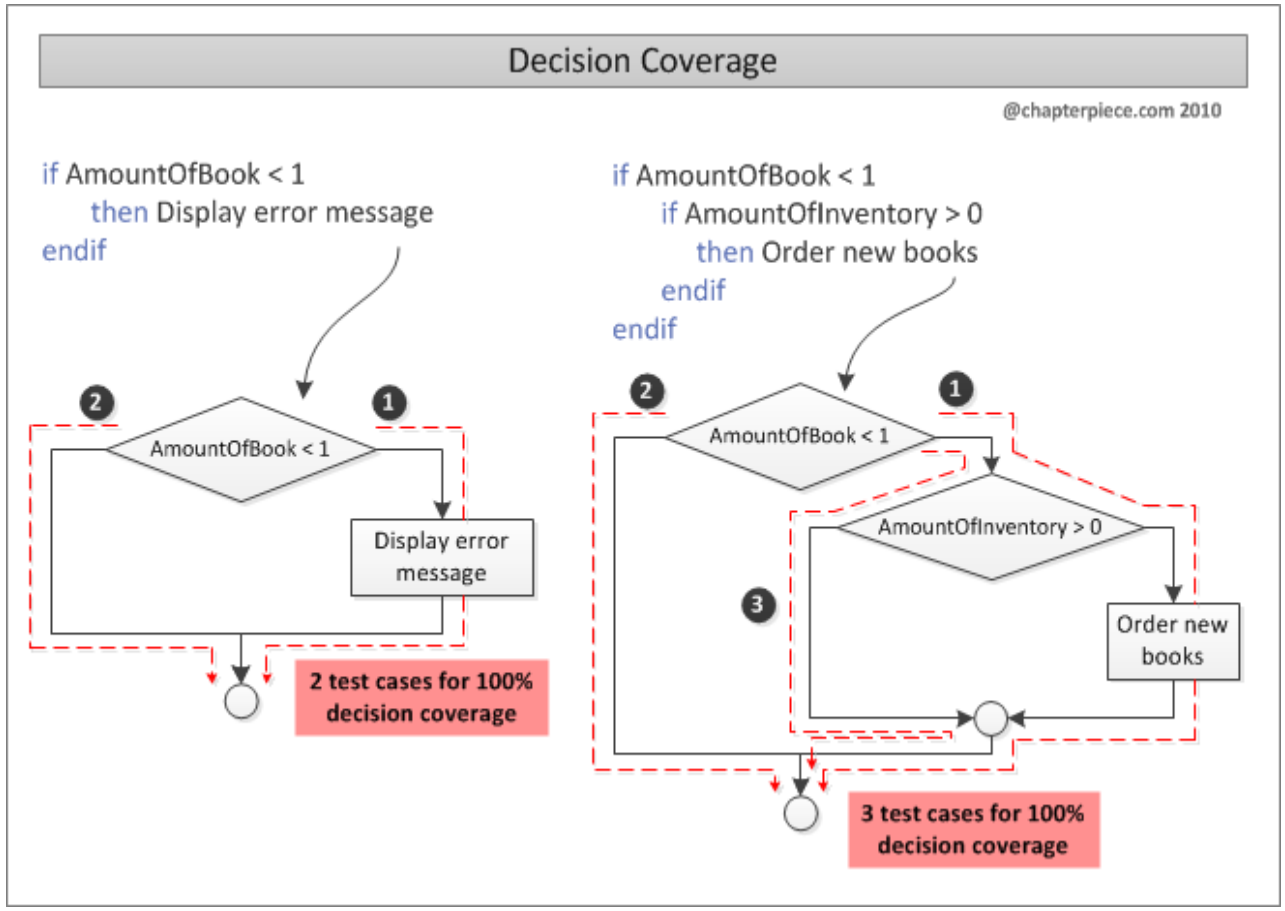
สวนทางขวา เราก็ต้องการแค่ 1 Test case เหมือนกันครับ โดย Test case ที่ว่าต้องทำให้ทั้งสองเงื่อนไขที่มีอยู่เป็นจริงทั้งคู่เพื่อให้โค๊ดทั้ง ห้าบรรทัดได้รับการทดสอบ ดูตามลูกศรครับ

Decision Coverage

เทคนิคนี้คือการเขียน Test case ที่ทุกๆการตัดสินใจของโค๊ดได้รับการทดสอบ ก็คล้ายๆกับ Statement coverage ในเรื่องของการตั้งเป้าหมายความครอบคลุมของการทดสอบครับ ไม่จำเป็นต้อง 100% หรอก เราลองมาดูตัวอย่างเทียบกันนะครับ

จากรูปด้านซ้าย เราจำเป็นต้องมี 2 Test case เพื่อทำให้ได้ 100% decision coverage โดย Test case แรก ของเราจะต้องทำให้เงื่อนไข if AmountOfBook < 1 เป็นจริงเพื่อ error message จะได้ถูกแสดงออกมา แล้วมาจบ endif (เหมือน Statement coverage เลย) สวน Test case ที่สองคือทำให้เงื่อนไขนั้นเป็นเท็จ แล้วโค๊ดก็จะมาจบที่ endif ทันที ตรงนี้เราได้ทำการทดสอบทั้งจริงและเท็จของเงื่อนไขการตัดสินใจของโค๊ดแล้วครับ

มาด้านขวาบ้าง อันนี้ซับซ้อนขึ้นมาหน่อยเราต้องมี 3 Test case ครับ ... เอ่อ ถ้าเขียนแล้วอาจจะงง ผมว่าดูตามลูกศรเลยดีกว่าครับ



มองเห็นอะไรจากตรงนี้บ้างรึเปล่าครับ? … Decision coverage มีประสิทธิภาพในการทดสอบมากกว่า Statement coverage เพราะ 100% decision coverage การันตีได้เลยว่านั่นคือ 100% statement coverage แต่ในทางกลับกันไม่ใช่นะ

ไม่ว่าจะใช้เทคนิคไหนก็ตาม เป้าหมายของเราคือการเขียน Test case จำนวนน้อยที่สุดที่จะสามารถครอบคลุมการทำงานของโค๊ดให้ได้มากที่สุดครับ

สรุปภาคสาม

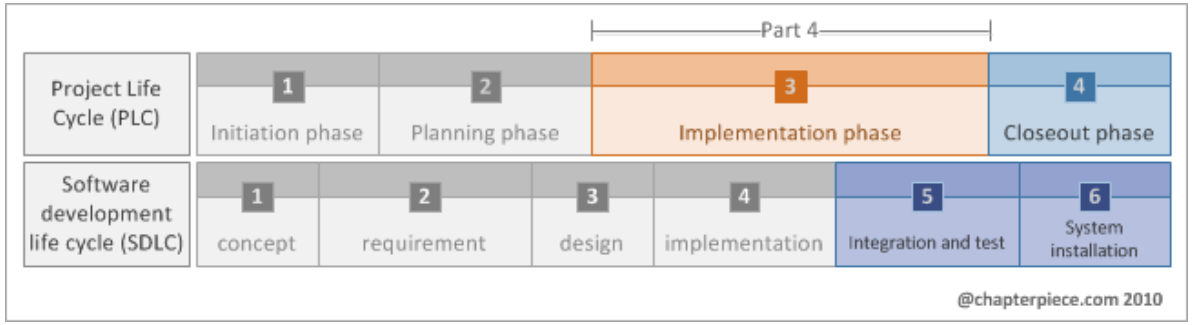
บทความนี้จะเน้นไปที่งาน ของ Developer เต็มๆ เริ่มจากการออกแบบระบบ (Design) พัฒนาระบบ (Implementation) จนถึงทดสอบระบบ (Unit testing) เลยครับ ถึงแม้หลักการและเทคนิคที่ใช้ในกระบวนการต่างๆจะมีมากมาย แต่ผมก็เลือกเฉพาะส่วนที่คิดว่ามีประโยชน์และเห็นภาพชัดเจนที่สุดโดยอาศัยประสบการณ์ของตัวเองครับ เพื่อนๆที่มีความรู้หรือประสบการณ์ต่างจากผมอาจจะเลือกเทคนิคอื่นๆก็ได้ ไม่ผิดครับ

อย่างไรก็ตามจุดสำคัญมากๆอีกจุดก็คือความพอดีในการทำงานทั้งหมดนี้ แน่นอนครับ ออกแบบละเอียด เขียนโค๊ดสุดยอด ทดสอบได้ครอบคลุมทั้งหมดย่อมดีอยู่แล้ว แต่ยากครับที่จะทำได้แบบนั้นเพราะว่าอย่าลืมครับ เวลาเรามีแค่ 5 เดือนเองสำหรับ Project นี้ ผมแนะนำว่าให้ทำอะไรแบบพอดีๆครับ พวก Diagram ต่างๆที่ใช้ในการออกแบบ ไม่ต้องทำทั้งหมดสำหรับทุก Feature หรือ Requirement ก็ได้ เลือกเฉพาะแต่ส่วนที่สำคัญๆ หรือการทดสอบต่างๆก็ตาม 100% test coverage มันงานหนักมาครับ เราตั้งเป้าซัก 70-80% ก็หรูแล้วฮะ

ตั้งแต่เขียนมาสามตอน ขอบอกเลยว่าตอนนี้เนื้อหาหนักสุด ใช้เวลาเขียนเยอะสุดเลยครับ ฮาๆ ยังไงจะพยายามให้บทความหน้าเกี่ยวกับ Project implementation ไม่หนักเท่านี้นะครับ

4 การบริหารจัดการและควบคุมโครงการ

จากรูปจะเห็นว่างานที่เป็น Implementation ของ Project management จะเริ่มเกือบพร้อมๆกับ Design phase ของ Software development life cycle งั้นผมขอสรุปเลยว่าตอนนี้เราอยู่ในช่วง Implementation ของ Project แล้วหละครับ



ขั้นตอนที่สำคัญของ Implementation หรือที่เรียกอีกอย่างว่า Execution นั้นประกอบด้วยสองส่วนหลักครับได้แก่ Execution กับ Tracking and controlling ครับ ในบทความนี้เราจะมาทำความรู้จักกับเจ้าสองกระบวนการที่ว่านี้กัน

Project Management – Execution

Project execution คือกระบวนการที่ใช้ทรัพยากรทั้งหมดที่มีทำงานตามแผน ทรัพยากรที่ว่าก็รวมถึงสมาชิกในทีมและเครื่องไม้เครื่องมือต่างๆที่เตรียมไว้ครับ เนื้อหาจริงๆก็มีอยู่แค่นี้แหละฮะ วางแผนมาแล้วก็ดำเนินการตามแผนไป ที่สำคัญกว่าอยู่ที่กระบวนการถัดไป

มากกว่าครับ

Project Management – Tracking And Controlling

กระบวนการนี้เป็นการวัดผลการทำงานว่าเป็นไปตามแผนที่วางไว้หรือไม่ เกิดความคลาดเคลื่อนหรือการเปลี่ยนแปลง (Variance) จากแผนที่วางไว้หรือไม่ มากน้อยแค่ไหน เช่น ทำงานไม่เสร็จตรงเวลาที่กำหนดหรือใช้งบประมาณเกินว่าที่ตั้งไว้ เป็นต้น ยิ่งไปกว่านั้นกระบวนการนี้เป็นการวิเคราะห์ถึงสาเหตุที่ทำให้เกิดความคลาด เคลื่อนนั้นและหาทางแก้ไขและป้องกันไม่ให้เกิดขึ้นอีกในอนาคตด้วยครับ

Tracking

เพื่อนๆคงสงสัยว่าแล้วเราจะรู้ได้อย่างไรว่าเกิด Variance ขึ้นหรือไม่? เรารู้ได้ด้วย Project tracking ซึ่งเป็นขั้นตอนในการเก็บรวบรวมข้อมูลที่สำคัญใน Project เพื่อนำไปวิเคราะห์ต่อในขั้นตอนต่อไปครับ ประโยชน์ของ Project tracking ก็มีอยู่หลายข้อ เช่น

- ทำให้เรารู้ถึงสถานะล่าสุดของ Project
- สร้างความเข้าใจที่ตรงกันระหว่างสมาชิกในทีม
- สนับสนุนให้ระบุและแก้ไขปัญหาที่เกิดขึ้นระหว่าง Project ซึ่งจะเป็นการเพิ่มประสิทธิภาพในการทำงานได้อีกทางหนึ่ง

แล้ว Tracking ต้องทำบ่อยแค่ไหนหละ? อันนี้ก็แล้วแต่ลักษณะของ Project นะครับ ไม่มีกฎเกณฑ์ตายตัวอะไร ขออย่างเดียวให้ทำเป็นประจำสม่ำเสมอ ถ้าเรากำหนดแล้วว่าจะ Track ทุกสัปดาห์ก็ให้ได้ตามนั้นจริงๆ

Types of Variance

โดยทั่วไปแล้วใน Project management เราจะให้ความสนใจกับความคลาดเคลื่อนอยู่สองประเภทได้แก่ Schedule variance กับ Cost variance ครับ

Schedule variance คือความคลาดเคลื่อนที่เกิดขึ้นจากการทำงานไม่เสร็จตรงตามเวลาที่วางแผนไว้ อาจจะเสร็จก่อนหรือหลังก็ได้นะครับ เช่น Task 2.1.3 วางแผนไว้ว่าต้องทำเสร็จเมื่อวันอังคาร แต่มาเสร็จจริงเอาวันศุกร์ ตรงนี้เกิด Schedule variance ขึ้นแล้ว 3 วัน

ครับ ตรงนี้มีผลกระทบกับงานที่เหลือโดยรวมเพราะว่าเราใช้เวลาหรือ Effort จาก Task อื่นๆมาทำงานให้ Task 2.1.3 ซะแล้ว

Cost variance คือความคลาดเคลื่อนที่เกิดขึ้นจากการใช้งบประมาณกับการทำงานใดๆมากหรือน้อยกว่าที่วางแผนไว้ เชน Task 2.1.3 วางแผนไว้ว่าต้องใช้เงิน 10,000 บาท แต่ตอนเสร็จงานเราใช้เงินไปแค่ 8,000 บาทก็ทำงานเสร็จแล้ว แบบนี้ก็ถือเป็น Variance นะครับ แต่เป็นไปในทางที่ดีเพราะเกิด Under budget = 8,000 – 10,000 = -2,000 บาทนั่นเอง

ผมสรุปสูตรคำนวณ Schedule variance กับ Cost variance มาให้ด้วยครับ

Schedule variance = วันที่ทำงานเสร็จ – วันที่วางแผนไว้ว่าจะทำงานเสร็จ

Cost variance = งบประมาณที่ใช้ไป – งบประมาณที่วางแผนไว้

ถ้าค่าออกมาเป็นลบถือว่าเป็น Ahead of schedule กับ Under budget ซึ่งเป็นผลดีกับเรา

ถ้าค่าออกมาเป็นบวกถือว่าเป็น Behind schedule กับ Over budget ซึ่งไม่ดีเท่าไร

จริงๆแล้วเรื่องนี้มีทฤษฎีอีกหลายตัวที่เกี่ยวข้อง แต่ผมขออนุญาตไม่พูดในบทความนี้ก่อนนะครับเพราะว่ามันค่อนข้างจะซับซ้อนและเป็นการคำนวณทางคณิตศาสตร์เยอะเหมือนกัน เอาเป็นว่าถ้าเพื่อนๆคนไหนสนใจเรื่อง Schedule variance, Cost variance และ Earn Value Analysis อย่างละเอียดก็บอกได้ครับ ผมจะเขียนบทความมาให้อีกที

How To

เพื่อนๆอาจจะสงสัยว่าแล้วเราจะเอาอะไรมา เป็นหลักในการ Track หละ? ครับ ไม่ต้องห่วงเรามีข้อมูลพวกนั้นอยู่เพียบ เริ่มด้วยนี่เลย WBS ฮะ เราติดตามผลงานของสมาชิกในทีมดูว่าผ่านมา 15 วันแล้ว งานอะไรควรจะเริ่ม ทำอยู่ หรือควรจะเสร็จไปแล้วบ้าง ตรงนี้มีประเด็นนิดนึงครับ กับการดูว่างานไหนควรเริ่มเนี่ยะไม่ยากฮะ ถ้าเริ่มแล้วก็บอกว่าเริ่ม ถ้าไม่เริ่มก็บอกไม่เริ่ม ไม่ต้องคิดมาก ... แต่กับงานที่ทำอยู่เนี่ยะครับยากหน่อย ลองดูตัวอย่างนี้ครับ

Project Manager อยากรู้ความคืบหน้าของงานอยู่ชิ้นหนึ่งก็เลยโทรไปถามข้อมูลจาก เอ ซึ่งเป็น Senior Developer และ Owner งานชิ้นนี้

PM: เอ พี่อยากรู้ว่างานที่เขียนโค๊ดสวนการคำนวณราคาเช่าหนังสือเสร็จรึยัง?

Senior Dev: อ๋อ เสร็จไป 50% แล้วครับพี่

PM: หรอ ดีๆ เร็วดี ขอบคุณครับ

เช้าวันรุ่งขึ้น Project Manager เดินสวนกับ บี Junior Developer ซึ่งเป็น Doer ก็เลยถือโอกาสทักทายซักหน่อย

PM: บี เมื่อวานพี่คุยกับเอมา เค้าบอกว่างานส่วนคำนวณราคาเช่าเสร็จไป 50% แล้วหรอ? เร็วดีจัง

Junior Dev: เอ่อ จริงๆผมมองว่าเสร็จไปแค่ 30% นะครับพี่

PM: ...

เอาหละ... แบบนี้ Project Manager ควรจะเชื่อใคร? ปัญหามันอยู่ที่แต่ละคนมีมาตรฐานในการมองความคืบหน้าของงานไม่เหมือนกันครับ ดังนั้นเพื่อตัดปัญหาเราขอให้น้องเค้าเลือกแค่สองคำตอบนี้พอ ไม่เสร็จ (เอาไป 0%) กับเสร็จ (เอาไปเลย 100%) ง่ายๆตรงๆไม่งงครับ

ใช้ WBS หรอ? เพื่อนๆอาจจะเริ่มกังวลว่า Project นี้เรามีตั้งเกือบ 200 tasks เลย ไล่เก็บข้อมูลทั้งหมดก็ไม่ต้องทำอะไรกินกันแล้ว ... จริงครับ ถ้าเก็บหมดนี่ก็เกินไปฮะ ผมแนะนำว่าเราเลือก Track ลงไปไม่เกิน Level 3 ของ WBS ก็ได้ครับ ประหยัดเวลา ข้อมูลไม่ล้นมือและมองเห็นภาพรวมได้ดีกว่ามากครับ

นอกจาก WBS แล้วเราควร Track อะไรอีก? ผมแนะนำว่ามีอีกสองตัวที่น่าสนใจครับ ตัวแรก Risk ที่มีอยู่ใน Project ครับ (ขออภัยที่ไม่ได้ลงรายละเอียดเรื่องนี้ แต่ถ้าเพื่อนๆคนไหนสนใจ บอกได้นะ ผมจะเขียนบทความแนะนำ Project risk management ให้ครับ) Risk คือความเสี่ยง (ที่อาจจะเกิดขึ้นในอนาคต) ซึ่งพวกเราก็มักจะนึกถึงสิ่งไม่ดีที่จะเกิดขึ้นได้กับ Project เรา เช่น Servers crash (เซิร์ฟเวอร์เจ๊ง) หรือ Project manager resigns (PM ลาออก 555) เป็นต้น คิดคร่าวๆนะครับ ถ้า Risk พวกนี้เกิดขึ้นจริงมันจะส่งผลกระทบต่อ Project เราแน่นอน ถ้าเป็นแบบนั้นเราจำเป็นต้องติดตามนิดนึงว่าตอนนี้มีโอกาสแค่ไหนที่ความเสี่ยงนี้จะเป็นจริงขึ้นมา นอกจากนั้นแล้ว เราก็ต้องหาทางป้องกันและแก้ไขปัญหาที่อาจจะเกิดตามมาด้วยครับ

ตัวสุดท้ายก็คือ Change ที่เกิดขึ้นใน Project ครับ Change หรือความเปลี่ยนแปลงที่เกิดขึ้นใน Project มาได้ในหลายรูปแบบเช่น Requirement (scope) change, Deadline (schedule) change หรือ Effort/Cost (resource) change โดยมากที่ผมเจอบ่อยๆก็จะเป็น Requirement change ครับ เราจำเป็นต้อง Track เหตุการณ์ที่เกิดขึ้นตรงนี้ด้วยเราจะได้รู้ว่าตอนนี้มี Change เกิดขึ้นกี่ตัวแล้ว สถานะล่าสุดเป็นอย่างไร เรารับหรือไม่รับ Change ตัวนี้ และรวมไปถึงว่า ถ้าเกิด Change ขึ้นบ่อยๆ เราคงต้องกลับมานั่งทบทวนแล้ว

หละครับว่าสาเหตุมันเกิดจากอะไร เราเก็บ Requirement มาถูกต้องครบถ้วนมั้ย? หรือว่าเรา Design ไม่ดีหว่า? อะไรแบบนี้ครับ

การ Track ในส่วนของ Risk กับ Change อาจจะไม่ได้บอกเราโดยตรงว่าตอนนี้เกิด Variance ขึ้นกับ Project แล้วหรือยังหรือว่าเกิดขึ้นมากน้อยแค่ไหน แต่ประโยชน์ที่จะได้โดยตรงก็คือเราจะรู้แนวโน้มว่างานของเราจะมี Variance รึเปล่าในอนาคตอันใกล้นี้ เท่าให้เราตื่นตัวและเตรียมการป้องกันแก้ไขได้อย่างทันท่วงทีครับ

Identify and Analyze Variances

เอาหละครับ หลังจากที่เก็บข้อมูลทั้งหมดที่จำเป็นต่อการ Track งานมาแล้ว ขั้นตอนต่อไปก็คือการนำข้อมูลเหล่านั้นมาระบุถึงความคลาดเคลื่อน (Variance) ต่างๆที่เกิดขึ้นตอนนี้

กระบวนการตรงนี้ก็ไม่ได้มีอะไรมากครับ เราก็แค่หาว่าเกิด Schedule variance กับ Cost variance ขึ้นหรือเปล่า มากน้อยแค่ไหน? บางครั้งการมีแต่ตัวเลขดิบ เช่น Cost variance = -2,000 บาท (under budget) เราจะมองเห็นไม่ชัดเจนว่ามันคลาดเคลื่อนไปจากที่วางแผนไว้มากหรือน้อยแค่ไหน เท่ากับการคำนวณออกมาเป็นเปอร์เซ็นต์ครับ จากตัวอย่างข้างบน Task 2.1.3 มี Cost variance = (8,000 – 10,000)/10,000 = -0.2 = -20% แบบนี้เห็นชัดกว่านะครับ

โดยทั่วไปแล้วก่อนเริ่ม Project ในช่วง Project planning เราควรจะมีการกำหนดฐานหรือ Baseline ของ Variance ที่จะเกิดขึ้นไว้ด้วย (ศัพท์อีกคำหนึ่งก็คือ Trigger) เช่น ถ้า Schedule variance หรือ Cost variance เกิน +15% หรือ -15% จะถือว่า Project มีความผิดปกติ (ขอโทษที่ไม่ได้กล่าวถึงเรื่องนี้ในภาค 2 ครับ) ซึ่งจำเป็นต้องหาสาเหตุและแนวทางแก้ไขอย่างเร่งด่วน ซึ่งในกรณีของเรานี้ Cost variance -20% ถือว่าอยู่ในสถานการณ์ไม่ปกติแล้วครับ อย่าได้ชะล่าใจไปว่า เราเก่งนะเนี่ยะประหยัดงบได้ตั้ง 20% แหนะ ... ประหยัดได้ก็ดีครับ แต่มากเกินไปนี่น่ากลัวจะมีอะไรผิดพลาดนะ เช่น เรื่องของคุณภาพงาน ทางที่ดีเราไม่ควรประมาทครับ ตรวจสอบหาข้อเท็จจริงซักหน่อยดีกว่า

Controlling

Determine the Causes

เข้าสู่ขั้นตอนการทำ Project controlling กันต่อเลยครับ หลังจากที่รู้แล้วว่า Cost variance มันดีเกินขนาด เราก็ต้องทำการตรวจสอบหาสาเหตุว่าทำไมเราถึงประหยัดงบประมาณได้มากขนาดนั้น?

กระบวนการนี้ก็เป็นการระดมสมองจากสมาชิกที่เกี่ยวข้องในทีมเพื่อหาสาเหตุที่ว่าครับ ไม่แน่การที่ Cost variance ติดลบเยอะอาจจะมาจาก

- เราประเมินงบประมาณผิดพลาดมาตั้งแต่แรกรึเปล่า
- เกิด Requirement change ขึ้นทำให้ Task 2.1.3 มีอะไรให้ทำน้อยลง
- หรือน้องๆในทีมเก่งจริงเลยทำงานเสร็จเร็ว (ถ้าได้แบบนี้จริงก็สุดยอดฮะ)

คิดไปคิดมา เอ้อ มันเป็นเพราะสาเหตุที่สองนี่ ลูกค้าขอแก้ไข Requirement โดยการลด Feature ที่ต้องทำใน 2.1.3 แต่ขอเพิ่ม Requirement ใหม่เข้ามาอีกหนึ่งตัวเป็นการแลกเปลี่ยน!!! ตรงนี้งานเข้านะครับ

Plan and Execute Adaptive Action

ขั้นตอนนี้เป็นการวางแผนและลงมือทำอะไรซักอย่างเพื่อจัดการกับสาเหตุที่ทำให้เกิดความคลาดเคลื่อน (Variance) นั้นๆ ซึ่งต้องการความคิดเห็นจากสมาชิกในเกี่ยวข้องทุกคน รวมถึงข้อมูลและเครื่องไม้เครื่องมืออีกหลายๆอย่างที่เรามี เช่น Project plan ฉบับสมบูรณ์ Risk กับ Change ที่มีใน Project และรวมถึง Flexibility matrix ที่เตรียมไว้ด้วยครับ

จากตัวอย่างเราจะเห็นได้ว่า Variance ที่เกิดขึ้นถึงแม้จะเป็นผลดีต่อ Task 2.1.3 นั้นมันมีผลกระทบใหญ่หลวงต่อ Project โดยรวม เพราะ Requirement ใหม่ที่เพิ่มเข้ามานั่นเอง นี่คือเรื่องที่เราต้องมาคุยกันเพื่อหาทางป้องกันและแก้ไขปัญหาที่อาจจะเกิดตามมาต่อไป

ตอนนี้ดูเหมือนว่าเรากำลังเจอปัญหาหนักเรื่อง Scope เพิ่ม ด้วยความสัมพันธ์ของ Scope, Schedule และ Resources ที่เรารู้อยู่แล้ว เรามีทางเลือกอยู่ว่า

- ขอเลื่อนเวลาส่งมอบ (Postpone schedule)
- เพิ่มทรัพยากรเข้ามา (Add resources)
- หรือตัดงานที่ไม่สาคัญออกไป (Cut scope)

ตรงนี้เราจะเห็นประโยชน์ของ Flexibility matrix แล้วครับว่า Schedule is the least flexible. นั่นคือตรงเสร็จตรงเวลา ... ตัวเลือกแรกตัดไป ถัดมาจะเห็นว่า Scope is the most flexible. นั่นคือถ้าจำเป็นจริงๆ เลือกตัด Requirement บางตัวที่ไม่สำคัญออกไปเพื่อ

ให้งานเสร็จทันเวลาได้นี่ ... เราก็เลือกวิธีนี้เป็น Adaptive action ของเราเลยครับ

อืม ... ผมว่าตัวเลือกเดียวไม่พอนะ เตรียมไว้อีกทางนึงด้วยดีกว่า งั้นเราไปเกริ่นกับ Project sponsor เพื่อขอเพิ่ม Resources เข้ามาใน Project ไว้ก่อน เผื่อเหลือเผื่อขาดไว้ครับ

ตรงนี้เราได้แนวทางป้องกันและแก้ไขปัญหาที่จะเกิดขึ้นในอนาคตแล้ว ต่อไปเป็นเรื่องจำเป็นอย่างมากที่จะต้องเพิ่มงานพวกนี้เข้าไปใน Project plan ด้วยครับ ไม่งั้นไม่รอดชัวร์ ทำไมหนะหรอ? ง่ายๆเลยครับ ลืมทำไงฮะ เราต้องมอง Adaptive action ให้เป็นงานจริงๆ ที่ต้องมีวันเริ่ม วันเสร็จ และคนรับผิดชอบด้วย อันนี้สาคัญมากครับ

Result

ผลลัพธ์ที่ได้จากการทำ Project tracking and controlling ก็คือ Project progress document ครับ แหะๆ คุยกันมาตั้งยาว ไม่เขียนไม่จดบันทึกอะไรไว้เลยอีกสองสัปดาห์มา Track กันใหม่ก็ลืมไปหมดแล้วหละครับ นี่เป็นเรื่องจำเป็นที่ Project Manager ต้องทำ และต้องนำเสนอความคืบหน้าของงานให้กับ Project Sponsor อย่างสม่ำเสมอครับ

เอกสารอีกฉบับที่อาจจะต้องมีการเปลี่ยน แปลงแก้ไขก็คือ Project plan ซึ่งรวมถึง SOW, WBS และ Network diagram ด้วย เพราะการที่เรามี Task ใหม่เพิ่มเข้ามันส่งผลกระทบโดยตรงกับเจ้า Project plan นี่แหละครับ

สรุปภาคสี่

เนื้อหาของภาคนี้จะเน้นไป ที่งานของ Project Manager เป็นส่วนใหญ่เลยครับซึ่งขั้นตอนนี้อาจจะเป็นขั้นตอนที่กินเวลายาวนานที่สุดใน Project ก็ได้ถ้า Project นั้นใช้เวลาทำนานเช่น 6 เดือนขึ้นไป

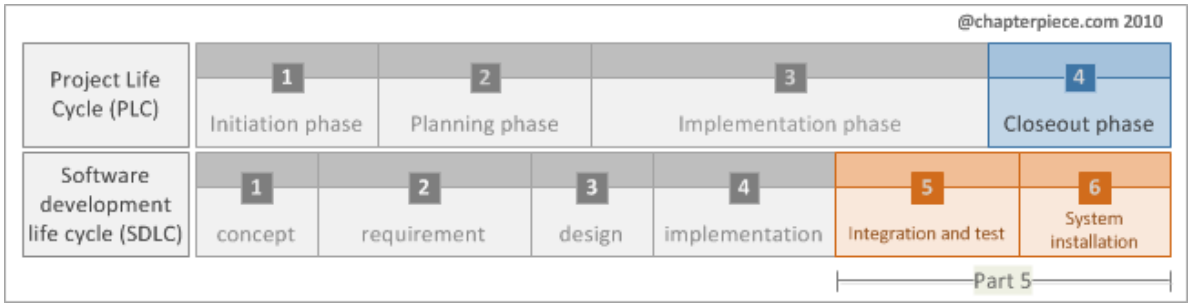
การที่มองว่างานของ Project Manager เสร็จตั้งแต่การเขียน SOW หรือ WBS นั้นเป็นความคิดที่ผิดอย่างยิ่งครับ การวางแผนก็เป็นส่วนสาคัญ การปฏิบัติตามแผน การติดตามความคืบหน้าของงานและการแก้ไขปัญหาที่เกิดขึ้นระหว่างการทำงานนั้นก็สาคัญไม่แพ้กันครับ ซึ่งกระบวนการตรงนี้อาจจะเหนื่อยสาหรับ Project Manager อยู่ซักหน่อยถ้าทำงานกับ Project ใหญ่ๆ แต่เราก็ไม่จำเป็นต้องตามเก็บข้อมูลทุกชิ้นทุกงานหรอกนะครับ แบบนั้นมันจะกลายเป็นจู้จี้จุกจิกและกดดันคนทำงานเกินไป แต่เราในฐานะ Project Manager ก็เลือกที่จะสอบถามถึงความคืบหน้าของงานในระดับที่ไม่ลึกเกินไป เช่น ไม่เกิน Level 3 ใน WBS เป็นต้นครับ

ผลลัพธ์ที่ได้จากความทุ่มเทตรงนี้คุ้มค่าแน่นอนครับ เพราะเราจะได้รู้ความเป็นไปโดยตลอดของ Project และที่สาคัญมากคือเราจะรู้แนวโน้มในอนาคตด้วยว่าจะเกิดอะไรที่ดีและไม่ดีขึ้นบ้างล่วงหน้าซึ่งทำให้เรามีเวลาเตรียมการป้องกันแก้ไขได้อย่างทันท่วงทีครับ

เอาหละครับ เราจบภาคสี่กันแล้ว ต่อไปผมจะพาเพื่อนๆทุกคนไปทำ Integration and system testing กันในภาคห้าครับ

5 การทดสอบซอฟต์แวร์

บทความนี้กลับมาที่ Software development life cycle กันอีกครั้งนะครับ หลังจากที่เราเขียนโค๊ดและทำ Unit testing เสร็จแล้ว ต่อมาก็เป็นเรื่องของ ... Testing อีกนั่นแหละ ฮาๆ แต่เป็นมุมมองใหม่ๆที่ไม่ซ้ำเดิม สนุกแน่นอนครับ



Integration Testing

วัตถุประสงค์สำคัญของ Integration testing ก็คือการประกอบร่างส่วนเล็ก (Module) ที่ถูกพัฒนาโดย Developers หลายๆคนเข้าด้วยกันเพื่อสร้างระบบที่ทำงานได้และทดสอบว่าระบบนั้นทำงานได้ตาม Requirement ที่กำหนดไว้ครับ

เพื่อนๆอาจจะสงสัยว่า "ขั้นตอนที่แล้วเราก็ทำ Unit testing อย่างละเอียดมาแล้วนะ ในเมื่อส่วนเล็กๆทำงานได้ถูกต้องแล้ว พอเอามารวมกันก็ต้องทำงานได้ซิ" การจะตั้งสมมติฐานแบบนั้นมันใช้ไม่ได้กับทุกกรณีนะครับ (จริงๆแล้วจะบอกว่าแทบใช้ไม่ได้เลยด้วยซ้ำ) ก็เพราะว่าปัญหามันเกิดตอนที่เราเอาส่วนเล็กๆมารวมกันนี่แหละ ความผิดพลาดตรงนี้เรียกว่า Interface error ซึ่งเกี่ยวข้องกับความไม่

ถูกต้องของตัวแปรอะไรต่างๆที่อยู่นอกสวนงานแต่ถูกเรียกใช้โดยสวนงานของเรา ผมขอยกตัวอย่าง Interface error ขึ้นมาเพื่อให้เพื่อนๆเห็นภาพที่ชัดเจนขึ้นนะครับ (ข้อมูลข้างล่างถูกรวบรวมโดย Perry และ Evangelist)

ประเภทของ Interface และ Interface Errors

Inadequate Functionality

แปลง่ายๆว่า Module ของเราทำงานได้ไม่ครบตามที่คาดหวังไว้สงผลให้คนที่มาเรียกใช้งาน Module เราทำงานที่ต้องการไม่ได้ เช่น Module เราไม่มีสวนการจัดการการจ่ายเงินผ่านบัตร Mastercard แบบนี้ตอนเทสก็จบเห่เลยครับ

Changes in Functionality

ปัญหานี้เกิดจากการเปลี่ยนแปลงรายละเอียดบางอย่างใน Module ของเรา แต่ไม่ได้มีการเปลี่ยนแปลง Module อื่นๆที่มีความสัมพันธ์กันให้เหมาะสม เช่น อยู่ดีๆเพื่อนเราเปลี่ยนจำนวน parameter ของ Module การเช่าหนังสือจาก 4 ตัวเป็น 5 ตัว แล้วไม่แจ้งเราที่เป็นคนเรียกใช้ (Caller) แบบนี้เมื่อเอาทั้งสองสวนมารวมกันก็เทสไม่ผ่านครับ

Misuse of Interface

ปัญหานี้ก็คือการที่ Caller เรียกใช้ Module อื่นๆอย่างผิดวิธี คำว่าผิดก็ตีความได้หลายแบบครับ เช่น สง parameter ไม่ครบ หรือสง parameter ผิดประเภทและผิดลำดับ เป็นต้น ตัวอย่างก็คล้ายๆข้อบนเลย

Data Structure Alteration

ถ้าจะพูดถึงปัญหานี้ให้ Developer เข้าใจง่ายๆก็คือเรื่องของ Buffer size ไม่พอครับ เช่น ผมเป็นคนเขียนโค๊ดในสวนการค้นหาลูกค้าโดยสร้าง Array ที่เก็บข้อมูลลูกค้าไว้ขนาด 512 แต่พอเอาเข้าจริงจำนวนลูกค้ามีมากกว่านั้นครับ แบบนี้ก็เกิดปัญหา Array Index Out Of Bound ขึ้นมา

Inadequate Error Processing

ปัญหานี้เกิดขึ้นเมื่อ Module ที่ถูกเรียกใช้ส่ง Error code คืนกลับให้ Caller แต่ Caller เอา Error นั้นไปประมวลผลต่ออย่างไม่ถูกต้อง เช่น ถ้าในกรณีที่ค้นหาชื่อลูกค้าไม่เจอ (เพราะ search criteria ไม่ตรง) ผมก็ส่ง Error code ที่ว่า "Record not found" กลับไปให้ Module ของเพื่อนผม แต่เพื่อนผมลืมแสดง Error code นี้ที่หน้าจอซึ่งอาจจะสร้างความสับสนให้กับผู้ใช้งานว่า เออ ตกลงว่าหาไม่เจอ หรือว่าระบบเจ๊งไปแล้วกันแน่

Inadequate Interface Support

ปัญหานี้ก็คือ Module ที่ถูกเรียกทำงานเพื่อตอบสนองความต้องการของคนเรียกได้ไม่เพียงพอ เช่น Module ที่ใช้ออกใบเสร็จรับเงิน คิดว่าเค้าสามารถคิดเงินเป็นสกุลดอลล่าร์ได้ แต่เราคนเขียน Module การคำนวณค่าเช่าหนังสือไม่ได้เตรียมส่วนการแปลงสกุลเงิน (currency converter) ไว้ให้ พอมีการเรียกใช้งาน Module ของเรา เพื่อนเราก็ไม่ได้ผลตามที่เค้าต้องการ เป็นต้น

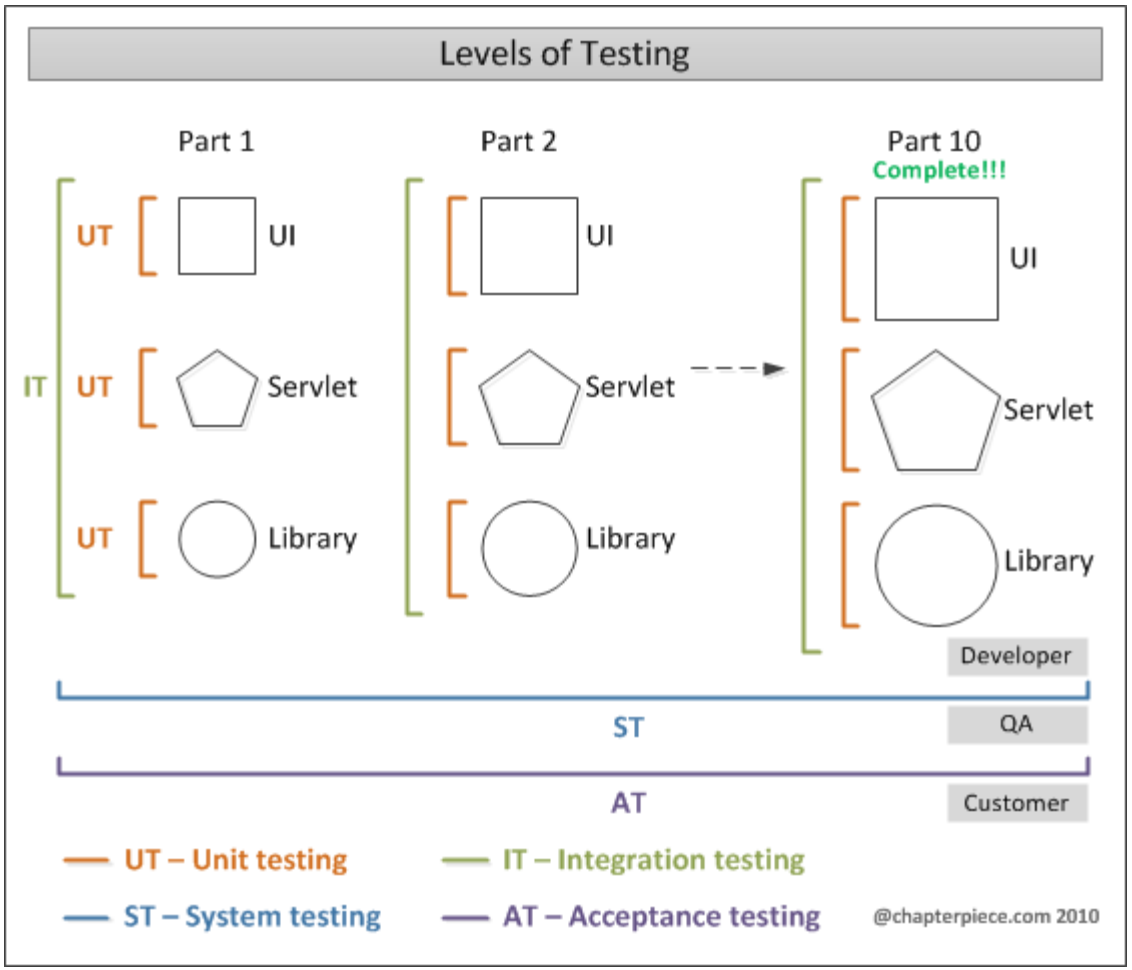
ครับ นี่เป็นเพียงตัวอย่างสวนหนึ่งที่ยืนยันว่า การที่เราทำ Unit test ผ่านหมดไม่ได้เป็นการรับประกันได้เลยว่าเมื่อนำ Module เล็กๆมารวมกันแล้วจะไม่มีปัญหาเกิดขึ้น ถึงตรงนี้เพื่อนๆคงจะเห็นความสำคัญของ Integration test มากขึ้นแล้ว จากประสบการณ์ของผมเอง คนที่รับผิดชอบทำ Integration test ก็ควรจะเป็น Developer เพราะมีความใกล้ชิด เข้าใจและเชี่ยวชาญในการออกแบบและการเขียนโค๊ดมากที่สุดครับ

Integration Testing Technique

Incremental

มีเทคนิคหลายรูปแบบให้เราเลือกใช้ในการ ทำ Integration testing แต่ที่ผมใช้อยู่ (และคิดว่าเหมาะสมที่สุดจากประสบการณ์) ก็คือ เทคนิค Incremental ครับ โดยสรุปก็คือ ค่อยๆทำ Integration testing ไปเรื่อยๆทีละนิดหลังจากที่เราทำ Unit testing ของ Module ต่างๆเสร็จแล้ว มองอีกมุมก็คล้ายๆทำเป็นรอบ (Cycle) ไปครับ เพื่อนๆลองดูตัวอย่างจากรูปครับ

ในแต่ละรอบ เราค่อยๆทำ Unit testing แล้วต่อด้วย Integration testing ไปเรื่อยๆ เมื่อเจอบั๊กก็ให้ Developer แก้ทันที เมื่อเรียบร้อยแล้วก็ค่อยเริ่มรอบใหม่ๆต่อไปเรื่อยๆจนครบสมบูรณ์ทั้งระบบ ซึ่งตรงนั้นก็จะพร้อมส่งงานของเราให้ QA ไปทำ System testing แล้วหละครับ



System Testing

เอาล่ะครับ หลังจากทำ Integration testing จนเสร็จสมบูรณ์แล้ว ต่อไปในฐานะ Developer เราก็ส่งมอบภาระอันใหญ่หลวงให้ทาง QA ช่วยทำต่อได้แล้วหละครับ QA ต้องทำอะไร? งานหลักของเค้าเลยก็คือ System testing ครับ

ความหมายที่เข้าใจง่ายๆก็คือ System testing เป็นกระบวนการทดสอบระบบของเรา (อีกแล้ว) ว่าทำงานได้ตาม Requirement ที่ระบุไว้หรือไม่ครับ

Black Box Testing

เพื่อนๆน่าจะเคยได้ยินคำ นี้กันมาบ้างแล้วนะ ครับ Black box testing ก็เป็นอีกหนึ่งเทคนิคสำหรับ Software testing ที่มีมุมมองต่างไปจาก White box testing ซึ่งจะเน้นการทดสอบที่ละเอียดและเป็นไปตามโค๊ดที่เราเขียนไว้ แต่สำหรับ Black box testing นั้น เราจะทำการทดสอบว่าระบบของเราทำงานได้ตาม Requirement หรือ Specification ที่กำหนดไว้รึเปล่า โดยที่เราไม่ต้องรู้ว่าการทำงานภายในของระบบเป็นอย่างไร ที่เราสนใจก็แค่ (1) Input รูปแบบต่างๆ (2) Output ที่ควรจะเป็น และ (3) System environment ของการทดสอบแต่ละครั้งครับ ข้อดีของเทคนิคนี้ก็คือ

- ไม่มีความลำเอียงเพราะคนเขียนโค๊ด (Developer) กับคนทดสอบ (QA) เป็นคนละคนกัน
- คนทำไม่ต้องมีความรู้เกี่ยวกับการเขียนโปรแกรมในระดับลึก
- การออกแบบ Test case สามารถทำได้เลยหลังจากทำ Requirement specification เสร็จ

ประเภทของ System Testing

เวลาทำ System testing นั้น QA ควรคำนึงถึงหลายเรื่องเพื่อเป็นการยืนยันได้อย่างมั่นใจว่างานของเราตอบสนอง ความต้องการของลูกค้าได้ครบทุกด้านจริงๆ ผมขอยกตัวอย่างบางเรื่องที่ QA ต้องสนใจให้ดูกันครับ

Basic test

ก็ตรงตามชื่อครับ การทดสอบเบื้องต้นซึ่งตามปกติแล้วก็จะประกอบไปด้วยการทดสอบการติดตั้งระบบ (System installation) การตั้งค่าระบบ (System configuration) และการเริ่มใช้งานระบบ (System operation) ครับ อันนี้เป็น Test case แรกๆที่ทุก Project ต้องมีเลยหละครับ

Functionality test

สวนนี้ก็เป็นการทดสอบว่าระบบของเราทำงานได้ครบถ้วนตามความต้องการของลูกค้าหรือไม่ ทุกอย่างที่มีระบุไว้ในสัญญาหรือ Requirement Traceability Matrix จะต้องถูกทดสอบทั้งหมดครับ เช่น สร้าง/ลบ user ได้ ค้นหาหนังสือได้ เช่าหนังสือได้ และอื่นๆ อีกมากครับ

Robustness test

อันนี้เป็นการทดสอบว่าระบบเรามีความ สามารถในการจัดการกับความผิดพลาดที่เกิดขึ้นได้ดีแค่ไหน ตัวอย่างนะครับ ระบบเรามีการจัดการกับ input ที่ผิดประเภทอย่างไร เช่น กรอกตัวอักษรลงไปในช่องที่รับข้อมูลเบอร์โทรศัพท์ ระบบป้องกันอย่างไร? หรือว่าเจ๊งไปเลย (อันนี้แย่นะ) หรือทดสอบว่าถ้ามีคนเปิดใช้งานระบบอยู่แล้ว ให้อีกคนไปดึงสาย LAN ออก ระบบเราจะจัดการยังไง? ระบบเรากู้ข้อมูลขึ้นมาได้มั้ย ได้ระดับไหน หรือว่าทำแค่นี้แล้วฐานข้อมูลเจ๊งไปเลย (อันนี้แย่กว่าอีก) เราจะมองว่าเป็น Negative test ก็ได้เหมือนกันครับ

Interoperability test

เป็นการทดสอบการทำงานร่วมกับระหว่างระบบเรากับผลิตภัณฑ์อื่นๆ (Third-party products) ทั้งหลายซึ่งรวมถึงทั้ง Hardware และ Software ครับ เช่น ใน Constraint ของระบบระบุว่าต้องใช้งานอยู่บนเครื่อง HP ProLiant DL100 โดยมี SUSE Linux Enterprise 11 เป็น Operating System มี Apache-tomcat 6.x เป็น Web server และ Oracle 11g เป็น Database เราก็ต้องทดสอบให้ครอบคลุมครับว่าระบบเราทำงานกับ Third-party เหล่านี้ได้อย่างสมบูรณ์

ข้อมูลที่ช่วยแนะนำแนวทางให้กับเราได้นอกจาก Requirement Traceability Matrix แล้วก็จะมี Infrastructure Architecture ด้วยครับ (จำได้กันรึเปล่า?) เพื่อนๆจะเห็นว่าถ้าเรามีการออกแบบโครงสร้างด้าน Infrastructure ที่ชัดเจน QA ก็จะรู้ว่าจะต้องเตรียมอะไรอย่างไรเพื่อทำการเทสครับ

งานนี้ค่อนข้างหนักทีเดียวครับ ยิ่งถ้าเราต้อง Support หลาย OS หลายเวอร์ชั่น Oracle ยิ่งจะทำให้เราต้องมีการวางกลยุทธ์การเทส (Test strategy) อย่างรอบคอบที่สุดครับ

Performance test

เป็นการทดสอบประสิทธิภาพในด้านต่างๆของระบบเราครับ เช่น ความเร็วที่ใช้ในการทำงานแต่ละงาน ความสิ้นเปลืองทรัพยากรของ

เครื่อง (CPU/Memory consumption rate) เป็นต้น ข้อมูลตรงนี้เราหาได้จาก Nonfunctional Requirements: Performance Requirements ครับ

Scalability test

เป็นหนึ่งในมุมมองด้าน Performance ของระบบครับ แต่จะเน้นไปในส่วนที่ว่าระบบเรารองรับการขยายตัวของข้อมูล จำนวนผู้ใช้ หรือจำนวน Hardware ได้ดีแค่ไหน เช่น ระบบเรารองรับผู้ใช้ซัก 10,000 คนได้มั้ย? ระบบเรารองรับการเข้าใช้งานพร้อมกันซัก 500 คนได้มั้ย? หรือว่าถ้าเราจะเพิ่มปริมาณของ Server จากที่มีอยู่แค่เครื่องเดียว ไปเป็นสามเครื่องจะทำได้มั้ย? เป็นต้นครับ

Stress test

นี่ก็อีกหนึ่งในมุมมองด้าน Performance ของระบบ แต่เป็นการเทสที่แปลกนิดนึงคือเราตั้งใจไว้เลยครับว่า เอาล่ะ วันนี้เราจะทำทำให้ระบบพังกันนะ แล้วเราก็ลุยเลยครับ เช่น อัดจำนวน user เข้าไปในฐานข้อมูลซัก 100,000 คน ... เจ๊งปะ? ยังหรอ งั้นเอาเข้าไปอีกเป็น 150,000 คน ... อ่าว เจ๊งล่ะ ระบบไม่ตอบสนองหรือว่าทำงานช้าลง ตรงนี้สำคัญครับ เราต้องบันทึกข้อมูลไว้ด้วยว่า มันเจ๊งที่จำนวนลูกค้าเท่าไรเพื่อหนึ่งเอาไปเทียบดูกับ Requirement ว่าระบบเราทำตามนั้นได้จริงๆนะ (กรณีนี้ผ่านเพราะ Requirement บอกว่ารองรับ user ได้อย่างน้อย 10,000 คน) และสองเอาไปเป็นข้อมูลในการเขียนเอกสารรับรองความสามารถของระบบครับว่าถ้าลูกค้าที่ใช้ระบบเรามี user เกิน 120,000 คน (ตัวเลขสมมติ) จะทำให้ระบบตอบสนองช้าลงและอาจจะทำงานผิดพลาดนะ

Load and stability test

ก็ Performance อีกแหละครับ คราวนี้เรามาลองดูกันว่าระบบเรามีความเสถียรแค่ไหนกับการใช้งานแบบเต็ม ประสิทธิภาพ ตัวอย่างเช่น ใช้งานระบบด้วยลักษณะการทำงานคล้ายๆของจริง เช่น มี user ซัก 5,000 คน มีหนังสือในระบบซัก 1,500 เล่ม แล้วก็มีคนเข้าใช้งานพร้อมๆกันซัก 50 คน อะไรแบบนี้ครับ เรามาดูกันว่าระบบเราจะทำงานได้ราบรื่น รวดเร็ว และถูกต้องได้นานขนาดไหน

ใจเราก็อยากให้มันทำงานได้ตลอดไปใช่ปะครับ? ผมก็หวังแบบนั้น แต่ความจริงมักจะโหดร้ายเสมอ บางครั้งเราลองเล่นไปซัก 2 ชั่วโมงอาจจะเกิด Out of Memory หรือ Database connection full ขึ้นมาก็ได้ แบบนี้เราก็ต้องมานั่งดูครับว่ามันเป็นที่เราเขียนโค๊ดไม่ดีรึเปล่า? หรือว่า Hardware ที่เราใช้มันมีประสิทธิภาพไม่สูงพอจะรองรับการทำงานในสถานการณ์ปกติได้

Reliability test

ก็เป็นการทดสอบที่มองว่าระบบเราสามารถเปิดให้บริการกับผู้ใช้ได้นานต่อเนื่องแค่ไหน การเทสก็ไม่ยาก เราก็แค่เปิดระบบทิ้งไว้เฉยๆ เลยครับ ว่างๆก็เข้ามากดๆเล่นๆดูซักหน่อย แล้วดูว่าผ่านไป 1 อาทิตย์ระบบเรามีอะไรเสียหายบ้างรึเปล่า ถ้าไม่มีปล่อยไป 2 อาทิตย์ 3 อาทิตย์ อะไรแบบนี้ครับ ลองดูว่าถ้าทิ้งไว้นานๆ ระบบเรามันจะเจ๊งไปเองรึเปล่า ... อันนี้เทสไม่อยากแต่ถ้าเจ๊งขึ้นมาหละ หาสาเหตุกับแก้ปัญหาไม่ง่ายเลยนะครับ

Regression test

อันนี้สาคัญมากครับ เป็นการทดสอบดูว่าระบบเราเนี่ยะมันทำงานได้ถูกต้องตามปกติอยู่ตลอดเวลาหรือ ไม่ สาเหตุที่หนึ่งที่เราต้องมีการทดสอบแบบนี้ก็เพราะเวลาเราทำ Integration testing แบบ Incremental นั่นคืองานค่อยๆเพิ่มขึ้นมา งานที่เพิ่มขึ้นมาไม่ได้ไปทำให้งานเก่าที่เคยใช้งานได้ดีเสียหายนะ และสองเรามองไปในอนาคตนิดนึงว่าถ้าระบบเราจะมีการเพิ่ม Feature ต่างๆขึ้นมา เจ้า Feature ใหม่ๆเนี่ยะต้องไม่ไปทำให้ของเก่าเค้าเสียหายไปด้วยนะ ... นี่แหละครับ Regression test ซึ่งผมคิดว่าจำเป็นมากๆที่ต้องทำก่อนส่งมอบระบบให้กับลูกค้า

Documentation test

อันนี้ไม่ใช่การทดสอบแต่เป็นการตรวจสอบว่าเอกสารต่างๆที่เกี่ยวกับระบบของเรา เช่น User manual หรือ Installation guide นั้นเขียนอย่างถูกต้องและครบถ้วน ไม่ใช่ว่าเราไปติดตั้งระบบตามขั้นตอนที่เขียนไว้ใน Installation guide แล้วทำไม่ได้ ไม่สาเร็จ อ่านไม่รู้เรื่องแบบนี้ไม่ได้นะครับ

เพื่อนๆจะเห็นว่างาน QA ไม่ใช่น้อยๆเลยถึงแม้จะดูเหมือนว่ามีบทบาทสาคัญอยู่เฉพาะในช่วงที่ทำ System testing ยิ่งถ้าต้องทำอะไรที่เกี่ยวข้องกับ Performance test แล้วด้วยยิ่งเหนื่อยหนัก ตัวอย่างง่ายๆเลยครับ Stress test ที่ว่าให้สร้าง user 100,000 คน แบบนี้มานั่งทำทีละคนจะไหวมั้ยครับ เป็นลมพอดี ประเด็นของผมมีอยู่ว่าถ้าเราเลือกได้และพร้อมเราควรพิจารณาใช้ Automated testing และเทคนิคอื่นๆเข้ามาช่วยครับ

Acceptance Testing

หลังจากที่เราทำ System testing เสร็จ เราก็มั่นใจแล้วหละว่าระบบของเราทำงานได้ยอดเยี่ยมพร้อมส่งมอบให้ลูกค้า ... และเก็บเงินงวดสุดท้าย แต่ลูกค้าเค้าจะคิดเหมือนเรารึเปล่าหละครับ การที่เราบอกว่าเราทดสอบระบบแล้วเรียบร้อย ลูกค้าอาจจะไม่เชื่อหรือไม่แน่ใจ 100% ว่าเค้ากำลังจะจ่ายเงินให้กับของดีมีคุณภาพและตรงตามความต้องการของเค้าจริงๆหรือไม่ เมื่อลูกค้าไม่แน่ใจ ลูกค้าก็ขอทดสอบระบบเองอีกครั้งก่อนรับมอบงาน คิดไปก็คล้ายๆเวลาเราจะซื้อบ้านซักหลังนะครับ ช่างปูน ช่างไม้ ช่างประปา ช่างไฟ ช่างสี สารพัดช่างยืนยัน นั่งยัน นอนยันว่างานเรียบร้อยดีแล้วครับพี่ ... เราเชื่อเค้าปะครับ? ไม่หรอก จริงมั้ยครับ บ้านหลังนึงเป็นล้านไม่เห็นด้วยตาไม่สัมผัสด้วยตัวเองไม่ได้หรอกครับ นี่แหละครับที่มาของ Acceptance testing

Acceptance testing หรือ User acceptance testing (UAT) เป็นการทดสอบครั้งสุดท้ายก่อนส่งมอบงานว่าระบบนั้นทำงานได้ตามมาตรฐาน (Acceptance criteria) ของลูกค้าหรือไม่ซึ่งจะเป็นการช่วยลูกค้าให้ตัดสินใจรับหรือไม่รับระบบนี้ โดยทั่วไปแล้วเจ้า Acceptance criteria นั้นจะถูกเขียนจากความคาดหวังหรือมาตรฐานของลูกค้าเองและการทดสอบก็จะทำโดยตัวลูกค้าเองด้วย (จริงๆก็เป็นไปได้ที่ลูกค้าจะไปว่าจ้างอีกบริษัทหนึ่งให้มาทำ Acceptance testing ในนามของลูกค้าครับ)

สำหรับตัวลูกค้าเองนั้นคำถามสำคัญก่อนเขียน Acceptance criteria ก็คือระบบต้องทำอะไรได้หรือไม่ได้บ้างเพื่อที่จะผ่านการทดสอบครั้งนี้? ซึ่ง Acceptance criteria นั้นต้องประเมินผลได้และวัดปริมาณได้แบบที่จะเขียนว่า “ระบบต้องค้นหา user ได้เร็ว” ไม่เอานะครับ คำว่า “เร็ว” คือเร็วแค่ไหน? ในสถานการณ์ยังไง? ถ้าตอบไม่ได้ก็ทดสอบระบบไม่ได้ครับ

ประเภทของ Acceptance Criteria

สำหรับตัวเราเองในฐานะคนพัฒนาระบบ เราก็ควรจะรู้ว่าลูกค้าเค้าจะทดสอบอะไรบ้างในช่วง Acceptance Criteria เพื่อที่จะหาทางป้องกันด้วยวิธีต่างๆให้ระบบสามารถตอบสนองกับสิ่งเหล่านั้นได้อย่างสมบูรณ์ เรามาดูกันครับว่าในมุมมองลูกค้าเค้าจะมองอะไรกันเป็นพิเศษ

Functional correctness and completeness

จุดนี้ก็จะดูคล้ายๆกับ System testing ครับว่าระบบเราทำงานได้ถูกต้องและครบถ้วนตามที่ Requirement เขียนไว้หรือไม่

Accuracy

ก็จะดูว่าระบบทำการประมวลผลอะไรต่างๆได้อย่างถูกต้อง 100% หรือเกือบ 100% (ขึ้นอยู่กับ Acceptance criteria) หรือไม่ โดยทั่วไปจะเน้นไปที่ส่วนการทำงานที่มีการคำนวณหรือวิเคราะห์ข้อมูลที่เป็นตัวเลขต่างๆ เช่น คำนวณราคาเช่าหนังสือ เป็นต้น

Data integrity

ลูกค้าจะดูว่าข้อมูลที่ถูกเก็บอยู่ในฐานข้อมูลนั้นอยู่ในสถานะที่ถูกต้องสมบูรณ์อยู่ตลอดเวลาซึ่งเป็นการตรวจสอบการออบแบบหรือเขียนโปรแกรมในระดับที่ค่อนข้างลึกเลยครับ เช่น ถ้าผมลบ user คนหนึ่งทิ้งไป ข้อมูลของคนๆนี้จะต้องหายเกลี้ยงออกไปจากตารางทุกตารางที่มีอยู่ในฐานข้อมูลนะ ไม่ใช่ว่าประวัติการเช่าหนังสือของเค้าก็ยังอยู่ หนังสือที่เค้าจองคิวไว้ก็ยังอยู่ แบบนี้เกิด Data corruption แล้วครับ ปัญหานี้มีความสำคัญต่อลูกค้าอย่างมากครับ

Data conversion

ก็เป็นการทดสอบดูว่าการเปลี่ยนแปลงประเภทหรือรูปแบบข้อมูลไปมาทำได้ถูกต้อง เช่น ถ้าลูกค้าต้องการให้ระบบนำข้อมูล user ทั้งหมดออกมาอยู่ในรูปแบบ CSV เพื่อที่ลูกค้าจะเอาไปเป็น Input ให้กับระบบการวิเคราะห์และวางแผนการตลาด ระบบเราทำได้อย่างถูกต้องหรือไม่? เป็นต้นครับ

Backup and recovery

ระบบใหญ่ทั่วไปแล้วต้องมีความสามารถในการ Backup และ Recovery ที่เชื่อถือได้ครับ ตรงนี้ก็จะเป็นจุดสำคัญที่ลูกค้าจะทดสอบ ซึ่ง Acceptance criteria ตรงนี้นอกจากจะทำการ Backup และ Recovery ได้อย่างถูกต้องแล้ว ลูกค้าจะดูด้วยว่ากระบวนการทั้งหมดใช้เวลาไม่นานจนเกินไปและระดับความสมบูรณ์ของข้อมูลที่ Recover ได้ด้วยครับ

Usability

อันนี้เกี่ยวข้องกับความรู้สึกของลูกค้าอยู่ซักหน่อยกับคำถามที่ว่า “ระบบนี้ใช้งานง่ายแค่ไหนและใช้เวลาในการเรียนรู้ที่จะใช้งานได้นานหรือ ไม่?” Acceptance criteria เพิ่มเติมก็จะมี เช่น ระบบยืดหยุ่นต่อการใช้งานมากแค่ไหน การตั้งค่าต่างๆของระบบยากมั้ย หรือมีระบบช่วยเหลือ (help) ที่ดีหรือไม่? เป็นต้น

Performance

จุดนี้ก็สำคัญสำหรับลูกค้าอย่างมากครับ เรื่อง Performance ของระบบทั้งหลายแหล่ ลูกค้าจะทดสอบในหลายๆมุมมองคล้ายๆกับที่เราทำใน System testing เลย เช่น Performance Stress Reliability and Availability และอื่นๆครับ ผมขอไม่พูดซ้ำในจุดนี้นะครับ

Documentation

สุดท้ายก็เรื่องเอกสารครับ ลูกค้าจะลองอ่านดูว่าอ่านง่าย เข้าใจง่าย มีรูปภาพประกอบคำอธิบายที่ชัดเจน ใช้ภาษาถูกต้องตามหลักไวยกรณ์หรือไม่ เป็นต้น

System Installation

หลังจากจบการ Testing แล้ว กระบวนการสุดท้ายของ Software development life cycle ก็คือ System installation ซึ่งโดยทั่วไปแล้วก็จะเป็นหน้าที่ของเรา (อาจจะเป็นทีม Field engineer) ที่จะไปติดตั้งระบบทั้ง Hardware และ Software ให้กับลูกค้าครับ

ผมเองไม่มีประสบการณ์ตรงกับกระบวนการนี้เท่าไรครับ ผมขออนุญาตไม่ลงรายละเอียดนะครับ กลัวจะเขียนผิดทำให้เพื่อนๆเข้าใจอะไรผิดๆไปด้วย

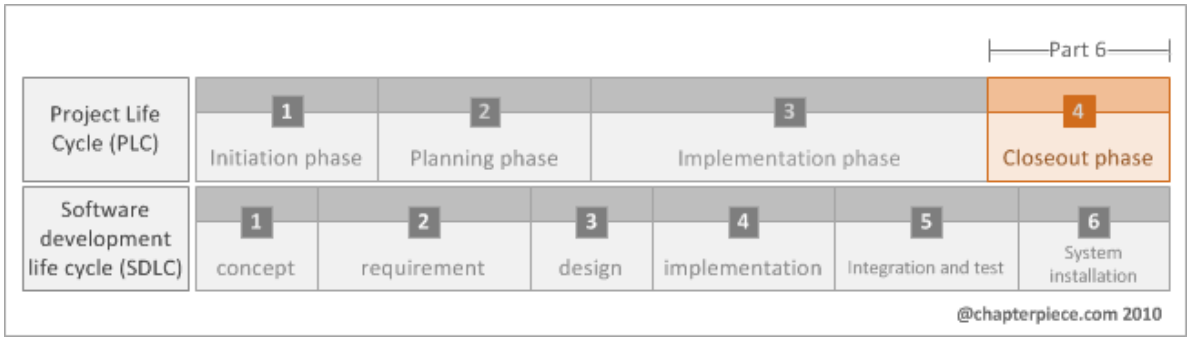
สรุปภาคห้า

ภาคนี้เน้นหนักไปที่การทดสอบระบบแบบต่างๆซึ่งเป็นหน้าที่รับผิดชอบของทั้ง Developer QA หรือแม้แต่ตัวลูกค้าเองครับ เพื่อนๆจะเห็นว่าผมนำเสนอข้อมูลที่เป็นทฤษฎีซะเป็นส่วนใหญ่เพราะว่าการทดสอบ ที่จะเกิดขึ้นนั้นขึ้นอยู่กับระบบของใครของมันจริงๆครับ เช่น ระบบเช่าหนังสือออนไลน์ของเราไม่จำเป็นต้องการ Reliability ที่สูงเท่าระบบโอนเงินธนาคารหรือระบบคาดการณ์แผ่นดินไหว เป็นต้น ผมก็หวังว่าทฤษฎีที่สาคัญของการทดสอบระบบเหล่านี้จะช่วยเป็นแนวทางให้เพื่อนๆได้นำไปต่อยอดให้เหมาะสมกับงานอีกทีครับ

การเดินทางนี้ช่างยาวไกลยิ่งนัก ทั้งคนอ่านและคนเขียน ... ผมเองก็เขียนมาจนเพิ่งรู้สึกตัวว่า Project เราเกือบเสร็จแล้วนะครับเนี่ยะ งานที่เป็น Software development นั้นเสร็จสมบูรณ์แล้ว แต่ผมจะขอปิดท้ายเรื่องที่เกี่ยวกับ Project management อีกนิดนึงในบทความหน้าซึ่งจะเกี่ยวกับกระบวนการที่เราควรทำเมื่อ Project (เกือบ)เสร็จแล้วครับ

6 การสรุปผลและปิดโครงการ

งานเลี้ยงย่อมมีวันเลิกรา ... คำนี้เป็นอมตะและแน่นอนใช้ได้กับ Project ของเราเหมือนกันครับ ความมุ่งหมายหลักของเราในฐานะ Project Manager ก็คือส่งมอบงานที่เราสัญญาไว้และทำให้ผู้ที่เกี่ยวข้องทุกฝ่ายพอใจกับผลงาน เหล่านั้น เมื่อสิ่งนี้เกิดขึ้น ... ก็ถือได้ว่าภาระหน้าที่ของเราหมดลงแล้วครับ



ขั้นตอนสุดท้ายของ Project Management ที่เรียกว่า Project Closeout นั้นเป็นงานที่มักจะถูกมองข้ามอยู่เสมอๆเลยครับ พอส่งงานให้ลูกค้าได้ เราก็ดีใจกันใหญ่ว่า Project ปิดแล้ว งั้นเก็บข้าวเก็บของ โยนเอกสารลงลิ้นชัก แล้วก็โยกย้ายไปทำงานกับ Project ใหม่ ... ช้าก่อนพี่น้อง ... ขอเวลาอีกนิด ช่วยทำเรื่องนี้ให้ผมก่อนได้มั้ย?

Lesson Learned

ความผิดพลาดเป็นบทเรียน ความสำเร็จก็เป็นบทเรียนเช่นเดียวกันครับ โดยส่วนตัวแล้วผมเชื่อว่าประสบการณ์เป็นแหล่งความรู้ที่ยิ่งใหญ่มากๆที่จะ ช่วยให้เราทำอะไรได้ดีขึ้นในอนาคต ช่วยเรามองเห็นและป้องกันปัญหาที่อาจจะเกิดขึ้นได้อย่างถูกต้องเหมาะสม ตลอดเวลาที่ทำ Project มา 6 เดือน เราผ่านอะไรกันมามากครับ ทั้งดีและไม่ดี เราทำอะไรสำเร็จหลายครั้ง แล้วเราก็เจอปัญหาอะไรมาหลายเรื่อง ... อย่าปล่อยให้ความรู้ที่ไม่มีสอนในหนังสือเล่มไหนหายไปกับสายลมและคำว่า "Project เสร็จแล้ว" นะครับ เรามาเก็บบันทึกสิ่งเหล่านี้กันดีกว่า

หลังจากปิด Project แล้วประมาณไม่เกินหนึ่งสัปดาห์ เราจะขอสละเวลาสมาชิกในทีมซัก 1-2 ชั่วโมงเพื่อประชุมสรุปสิ่งต่างๆที่เราเรียนรู้จาก Project ที่ทำเสร็จไปทั้งหมด เราเรียกการประชุมแบบนี้ว่า Project Debriefing ครับ ในการประชุมนี้ถ้าเป็นไปได้ก็อยากให้ทุกคนได้มีสวนร่วมครับ ทั้ง Developer, QA, Project Manager, System Analyst และอื่นๆ มีใครสงสัยมั้ยครับว่าทำไมต้องประชุม? ที่ต้องประชุมก็เพราะต้องการแบ่งปันความรู้ประสบการณ์ที่แต่ละคนเจอมาซึ่งไม่เหมือนกันหรอกให้กับเพื่อนๆคนอื่น ตำแหน่งอื่นได้เข้าใจ และเปิดโอกาสให้ทีมสร้างข้อตกลงร่วมกันครับว่าจากข้อมูลที่ได้มาทั้งหมดนี้เราจะพัฒนาและปรับปรุงกระบวนการทำงานของเราให้ดีขึ้นได้อย่างไรครับ

ขั้นตอนของการจัดประชุมแบบนี้ก็จะพิเศษนิดนึง

1. มองหาคนทำหน้าที่ผู้ประสานงานการประชุม โดยมากแล้วมักจะเป็น Project Manager หละ (ยินดีด้วย)
2. สมาชิกในทีม (รวมถึงผู้ประสานงานการประชุมด้วยนะครับ) จะส่งข้อมูล ประสบการณ์ และโอกาสในการพัฒนาอะไรต่างๆให้กับผู้ประสานงาน
3. ผู้ประสานงานจะทำหน้าที่เก็บรวมรวบความคิดเห็นทั้งหลายที่สมาชิกในทีมส่งมาให้ ทำเป็นข้อสรุป และเตรียมการนำเสนอในห้องประชุม
4. ในห้องประชุม ผู้ประสานงานก็จะเล่าเรื่องราวต่างๆให้กับสมาชิกในทีมฟังว่า ... ได้ข้อมูลอะไรมาบ้าง แล้วก็เปิดโอกาสให้คนในทีมเสนอความคิดเห็นเพิ่มเติมและวางแนวทางสำหรับอนาคตร่วมกันครับ โดยปกติแล้วข้อมูลที่ได้จากสมาชิกในทีมจะเป็นความลับนะครับ เป็นจรรยาบรรณของผู้ประสานงานที่จะไม่เปิดเผยว่าใครเป็นเจ้าของข้อมูลหรือ ความคิดนี้ ... แต่ถ้าสมาชิกในทีมสนิทกันมากๆ แค่เห็นสานวนการเขียนก็รู้แล้วว่าใคร 55 อันนี้ไม่ว่ากัน
5. หลังจากนั้นแล้วผู้ประสานงานก็ต้องเขียนข้อสรุปการประชุม ทั้งข้อมูลสำคัญและแนวทางแก้ไขลงในเอกสารเพื่อเก็บไว้อ้างอิงต่อไปครับ

ผมเพิ่มเติมให้อีกนิดนึงครับ เกี่ยวกับคำถามหรือข้อมูลที่ควรจะเก็บรวบรวมมาเพื่อทำ Project Debriefing ครับ

ปัจจัยอะไรที่ทำให้ Project ประสบความสำเร็จ?

- ทีมเวิร์ค
- ความรับผิดชอบของสมาชิกในทีม
- การตรวจสอบและแก้ไขปัญหาอย่างรวดเร็วทันท่วงที
- ...

ปัจจัยอะไรที่ขัดขวางทำให้ Project ไม่ประสบความสำเร็จ?

- การสื่อสารระหว่างสมาชิกในทีมและลูกค้า
- ปัญหาเกี่ยวกับ Computer Network และ Software ที่จำเป็น
- ความไม่สมดุลระหว่างเวลากับปริมาณงาน
- ...

คำแนะนำเพื่อปรับปรุงกระบวนการทำงานใน Project ถัดไป

- จัดประชุมกับลูกค้าทุกสัปดาห์
- นำหลักการ Operation concept มาช่วยในการเขียน Requirement
- จัดหา Computer Network และ Software ที่จำเป็นมาใช้งาน
- เพิ่ม Buffer เข้าไปใน Project plan เพื่อรองรับกับความเปลี่ยนแปลงหรือปัญหาที่ไม่คาดคิด
- สร้าง Central software library เพื่อจัดเก็บและเรียกใช้เอกสารของสมาชิกในทีม
- ...

ผู้บริหารจะให้ความช่วยเหลืออะไรได้บ้าง?

- หา Developer และ QA เพิ่ม
- ให้คำปรึกษาสาหรับปัญหาที่มีความรุนแรงและเกี่ยวข้องกับลูกค้าโดยตรงอย่างสม่ำเสมอ
- อนุมัติเงินล่วงเวลาเมื่อจำเป็น
- ...

คำแนะนำอื่นๆ

- ขอ OT ครับ
- จัดงานเลี้ยงหลังจบ Project ครับ

- ...

ผมผ่านการมีสวนร่วมใน Project Debriefing มาหลายครั้งแล้ว บอกได้เลยว่ามีประโยชน์มากจริงๆครับ การพัฒนากระบวนการในการทำงานก็มองเห็นเป็นรูปธรรม การวางแนวทางป้องกันปัญหาที่อาจจะเกิดขึ้นในอนาคตก็มีหลักมีการและสุดท้ายสมาชิกในทีมทุกคนมีโอกาสได้ระบายความในใจที่อัดอั้นมานาน !!! อันนี้สาคัญนะครับ คอนเฟิร์ม ฮาๆ

Releasing Project Team

เพื่อนๆที่ทำงานบริษัท Vendor คงพอจะเข้าใจธรรมชาติของงานอยู่แล้วว่าพวกเราก็เหมือนเป็นมือปืนรับจ้างครับ เมื่อทำงานนี้เสร็จก็ต้องเตรียมตัวให้พร้อมรอรับงานใหม่ๆที่จะเข้ามา ดังนั้นเมื่อ Project นี้จบแล้วพวกเราก็ถูกส่งตัวคืนหน่วยงานต้นสังกัดครับ

กระบวนการนี้ไม่จำเป็นต้องทำอย่างเป็นทางการหรอกนะครับ แต่สาหรับ Project Manager แล้วมีหน้าที่แจ้งให้กับ Line Manager (Functional Manager) และ Project Manager คนอื่นๆทราบว่า Project ของเราใกล้จะเสร็จแล้วนะ (บอกก่อนซัก 1 เดือนกำลังดีครับ) เพื่อที่ Manager คนอื่นๆเค้าจะได้มีเวลาวางแผนว่าจะมอบหมายงานอะไรให้กับน้องๆที่กำลังจะหมดหน้าที่กับ Project เก่าครับ

Celebration!!!

ทุ่มเทมาตลอด 6 เดือน ในฐานะคนทำงานเราก็อยากได้อะไรตอบแทนกันบ้างครับ จริงๆแล้วทางบริษัทควรใช้โอกาสนี้ในการขอบคุณสมาชิกในทีมที่ได้ช่วยทุ่มเททำงานจนเสร็จตรงตามเวลา สร้างรายได้ (มหาศาล) ให้กับบริษัท ที่จะทำได้ก็คือ Project Sponsor อนุมัติเงินซักก้อนนึงมาจัดงานเลี้ยงให้กับสมาชิกในทีมครับ เรื่องแบบนี้สาคัญมากนะครับ มันจะช่วยเพิ่มกำลังใจในการทำงานให้น้องๆในทีม ช่วยสร้างความสัมพันธ์ที่ดีขึ้นระหว่างคนในทีมและอีกหลายๆเรื่องเลยครับ

แต่ถ้าบริษัทไม่มีนโยบายนี้หละ เราจะทำยังไงดี? บริษัทไม่ให้ ... เราก็หากันเองครับ หลังปิด Project ทุกครั้งผมและเพื่อนๆในทีมจะมีกินเลี้ยงกันตลอด ส่วนใหญ่ก็จะไปกินบุฟเฟต์ตามโรงแรม (ดูดีนิดนึง) หรือถ้าไม่สะดวก (ไม่ค่อยมีกะตังค์) ก็สั่งสมตำข้างตึกมากินกันครับ ฮาๆ ค่าใช้จ่ายก็ออกกันเอง ตามลำดับความอาวุโส เช่น Project Manager จ่ายเยอะสุด ตามมาด้วย Team Leader แล้วก็ Senior ทั้งหลายครับ ... น้องๆ Junior ฟรีครับ

ทั้งหมดนี้ก็คือสิ่งที่ควรจะต้องทำก่อนที่จะพูดได้อย่างเต็มปากว่า “เย้ๆ Project เสร็จแล้ว” ครับ

ปล. ผมทำสรุปขั้นตอนทั้งหมดในการพัฒนาซอฟต์แวร์มาให้ด้วยครับ (เป็น Mindmap นะครับ)
How_To_Build_Software_Mindmap

สุดท้ายจริงๆ

ที่ผ่านมาทั้ง 6 ตอน ผมพยายามที่จะผสมผสานระหว่างทฤษฎีและตัวอย่างในชีวิตจริงของพวกเราที่มีส่วน ร่วมใน Software Development Project ครับ แน่นอน เนื้อหานั้นคงไม่ครบถ้วนแบบ 100% แต่ผมก็หวังว่าทั้งหมดจะเป็นแนวทางที่ช่วยให้เพื่อนๆเอาไปปรับใช้ได้จริง

ขอบคุณทุกคนที่ติดตามผลงานมาตั้งแต่ตอนแรกครับ ... ที่สำคัญผมต้องขอบคุณคุณ OAK (ขอออกชื่อหน่อยฮะ) ที่แนะนำให้ผมเขียนเรื่องนี้ มันเป็นการเปิดโอกาสให้ผมได้ประมวลความรู้ที่มีออกมาเป็นบทความที่น่าจะเป็น ประโยชน์กับเพื่อนๆอีกหลายคน และผมก็ภูมิใจมากๆกับทั้ง 6 บทความนี้ นี่เป็นหนึ่งในเหตุผลที่ผมทำบล็อกนี้ขึ้นมาครับ ... “ผู้เขียนและผู้อ่านเป็นได้ทั้งผู้ให้และผู้รับในเวลาเดียวกัน”

หลังจากอ่านหกบทความนี้แล้ว เพื่อนๆมองเห็นภาพรวมของการพัฒนาซอฟต์แวร์มากขึ้นมั้ยครับ? หวังว่าจะมีประโยชน์บ้างนะครับ

ถ้าเพื่อนๆคนไหนอยาก[แสดงความคิดเห็นหรือติชม]เกี่ยวกับ Ebook เล่มนี้ ผมก็ยินดีและจะเก็บเอาคำแนะนำนั้นไปปรับปรุงต่อไปครับ

หวังว่า Ebook เล่มนี้จะถูกใจเพื่อนๆนะ ขอบคุณครับ

ปีโยรส ปียจันทร์

(kannique)