บทที่ 8

การทดสอบซอฟต์แวร์ การนำไปใช้และการบำรุงรักษา

(Software Testing, Deployment and Maintenance)

ในบทนี้จะกล่าวถึงการทดสอบซอฟต์แวร์แต่ละระดับและประเภทที่แตกต่างกัน อธิบายเทคนิคการเขียนการทดสอบเชิงหน่วยในแนวทางของ Test-Driven Development รวมถึงการทดสอบเว็บแอพพลิเคชัน จากนั้นจะกล่าวถึงกลไกการนำซอฟต์แวร์ไปใช้และขั้นตอนการบำรุงรักษา

8.1 การทดสอบซอฟต์แวร์

การทดสอบซอฟต์แวร์เป็นกระบวนการดำเนินการตรวจสอบเพื่อให้ผู้ที่เกี่ยวข้องกับระบบ เช่น เจ้าของหรือลูกค้าได้รับทราบถึงระดับคุณภาพของซอฟต์แวร์นั้นๆ รวมทั้งให้บุคคลทางฝ่ายธุรกิจที่เกี่ยวข้องกับระบบได้เข้าใจความเสี่ยงในการสร้างซอฟต์แวร์ การทดสอบซอฟต์แวร์มีหลายเทคนิคที่สามารถทำได้ เช่น การสั่งให้ซอฟต์แวร์ทำงานเพื่อที่จะค้นหาข้อบกพร่อง หรือการพิสูจน์ด้วยวิธีการทางฟอร์มอลเมธอด (Formal method) (Leroy & Blazy, 2008) เป็นต้น การทดสอบซอฟต์แวร์ขึ้นกับวิธีและกระบวนการ-พัฒนา โดยอาจจะเป็นระยะหลังจากการสร้างซอฟต์แวร์แล้ว หรืออาจเป็นส่วนหนึ่งของช่วงการพัฒนาก็ได้ ในขณะที่วิธีการพัฒนาบางแนวคิดอาจทำการทดสอบก่อนที่จะเขียนโปรแกรม เช่น Test-Driven Development (Beck, 2002)

8.1.1 ระดับของการทดสอบ

การทดสอบซอฟต์แวร์สามารถแบ่งออกเป็นระดับได้ดังนี้ (Pressman, 2010)

1) การทดสอบระดับหน่วย (Unit Testing) เป็นการทดสอบระดับที่ใกล้กับผู้พัฒนามากที่สุด เพื่อตรวจทานความสามารถบางส่วนของต้นรหัสที่สร้างขึ้น เช่น การทดสอบฟังก์ชันหรือการทดสอบเมธอด เป็นต้น โดยเมธอดหรือฟังก์ชันอาจมีการทดสอบมากกว่า 1 แบบ เพื่อให้ครอบคลุมกรณีที่ต้องระมัดระวังเป็นพิเศษ คุณสมบัติที่สำคัญของการทดสอบระดับหน่วย คือ ความเป็นอิสระต่อกัน ซึ่งจะทำให้มั่นใจได้ว่าพฤติกรรมของระบบได้รับการทดสอบเป็นส่วนๆ แล้ว

2) การทดสอบระดับบูรณาการ (Integration Testing) เป็นระดับของการทดสอบที่ใช้เพื่อตรวจทานการต่อประสานระหว่างชิ้นส่วนซอฟต์แวร์ว่าเป็นไปตามการออกแบบหรือไม่ การทดสอบเชิงบูรณาการจะเป็นระยะที่นำเอาโมดูลหรือชิ้นส่วนทางซอฟต์แวร์แต่ละตัวมารวมและทำการทดสอบร่วมกัน การทดสอบที่ตั้งขึ้นจะเป็นการทดสอบที่ครอบคลุมการทำงานของทุกชิ้นส่วน ซึ่งต้องมีการเรียกใช้คำสั่งข้ามชิ้นส่วนกัน การทดสอบประเภทนี้มักกระทำหลังจากที่ได้ทำการทดสอบระดับหน่วยเรียบร้อยแล้ว แนวทางการทดสอบระดับบูรณาการมี 3 แนวทาง คือ

- Big Bang เป็นการประกอบระบบแล้วทดสอบด้วยข้อมูลการใช้งานจริง
- การทดสอบจากบนลงล่าง เป็นการทดสอบระบบรวมก่อนระบบย่อย
- การทดสอบจากล่างขึ้นบน เป็นการทดสอบระบบย่อยก่อนแล้วค่อยๆ ทดสอบระบบที่ใหญ่ขึ้น

3) การทดสอบระดับระบบ (System Testing) เป็นการทดสอบระบบที่ประกอบกันอย่างสมบูรณ์แล้วว่าสามารถทำงานได้ตามข้อกำหนดเชิงฟังก์ชันหรือไม่ การทดสอบในลักษณะนี้จะเป็นการทดสอบพฤติกรรมของระบบว่าเป็นไปตามความคาดหวังของผู้ใช้หรือลูกค้าหรือไม่ โดยการทดสอบระดับระบบจะทำหลังการทดสอบระดับบูรณาการ

4) การทดสอบระดับบูรณาการระบบ (System Integration Testing) เป็นกระบวนการทดสอบระบบซอฟต์แวร์ในบริบทของการเชื่อมต่อกับระบบภายนอกอื่นๆ ที่ระบุไว้ในเอกสารข้อกำหนด เนื่องจากการทดสอบระดับนี้จะมีการใช้ระบบจริง ซึ่งเชื่อมต่ออยู่กับระบบอื่นมาใช้เป็นตัวทดสอบ ดังนั้นข้อมูลสำหรับใช้ในการทดสอบระดับนี้ควรมีขนาดเล็กที่สุดที่จะครอบคลุมกรณีทดสอบเพื่อลดเวลาในแต่ละระบบของการทดสอบ

5) การทดสอบแบบถดถอย (Regression Testing) เป็นการทดสอบที่ทำซ้ำหลังมีการเปลี่ยนแปลงแก้ไขต้นรหัสเกิดขึ้น โดยปกติในกระบวนการทดสอบจะมีการทดสอบระดับหน่วยสำหรับชุดของต้นรหัสที่เขียนเพิ่มขึ้น และเพื่อให้แน่ใจว่าส่วนที่เพิ่มขึ้นมานั้นไม่ทำให้เกิดผลกระทบต่อต้นรหัสทั้งหมดที่มีอยู่ก่อนแล้ว นักพัฒนาจะทำการทดสอบเชิงถดถอยโดยนำเอากรณีทดสอบเชิงหน่วยที่สร้างไว้ทั้งหมดมาทดสอบซ้ำ

8.1.2 การทดสอบแอพพลิเคชันทั่วไป

การทดสอบแอพพลิเคชันทั่วไปสามารถแบ่งออกได้เป็น 2 ประเภท ดังนี้ (Pressman, 2010)

1) การทดสอบแบบ White-box เป็นวิธีการที่ใช้ทดสอบโครงสร้างภายในของซอฟต์แวร์ การสร้างกรณีทดสอบประเภทนี้ผู้เขียนกรณีทดสอบจำเป็นต้องมีความรู้ความเข้าใจเกี่ยวกับระบบภายในของซอฟต์แวร์นั้นๆ รวมทั้งต้องมีทักษะการพัฒนาโปรแกรมเป็นอย่างดี ในการทดสอบลักษณะนี้ผู้พัฒนามักจะเป็นผู้เตรียมข้อมูลทดสอบเอง อย่างไรก็ตามแม้ว่าการทดสอบประเภท White-box จะสามารถนำไปใช้ได้ในทุกระดับของการทดสอบ แต่ก็มักจะทำในการทดสอบเฉพาะระดับหน่วย ตัวอย่างเครื่องมือประเภท White-box เช่น JUnit (Beck, 2002)

2) การทดสอบแบบ Black-box เป็นวิธีการทดสอบซอฟต์แวร์เฉพาะฟังก์ชันการทำงานของซอฟต์แวร์โดยไม่สนใจรายละเอียดภายในหรือโครงสร้างภายในของโปรแกรม การทดสอบจะถูกเตรียมขึ้นจากข้อมูลข้อกำหนดความต้องการเชิงซอฟต์แวร์รวมถึงการออกแบบ โดยปกติการทดสอบประเภท Black-box มักจะเป็นการทดสอบความต้องการเชิงฟังก์ชัน แต่ก็จะสามารถทดสอบความต้องการส่วนที่ไม่ใช่ฟังก์ชันได้เช่นกัน ตัวอย่างเครื่องมือประเภท Black-box เช่น Selenium หรือ Sikuli เป็นต้น

8.1.3 การทดสอบแอพพลิเคชันเชิงวัตถุ

หน่วยย่อยที่สุดที่สามารถทดสอบได้ในการสร้างโปรแกรมเชิงวัตถุคือคลาส จึงทำให้คุณสมบัติบางอย่างโดยเฉพาะ Encapsulation เข้ามาเกี่ยวข้องกับการทดสอบโดยตรง นั่นแปลว่าจะทำให้ไม่สามารถทดสอบเมธอดเดี่ยวได้ในภาวะที่แยกออกจากระบบอย่างสิ้นเชิง เนื่องจากการทดสอบเมธอดหนึ่งๆ ของคลาสจะทำให้ “สถานะ” ของวัตถุเปลี่ยนแปลงไป แนวคิดในการออกแบบกรณีทดสอบสำหรับระบบเชิงวัตถุจึงควรคำนึงถึงหลักดังต่อไปนี้ (Pressman, 2010; Beck, 2002)

1) แต่ละกรณีทดสอบควรมีความเฉพาะและเกี่ยวข้องกันอย่างชัดเจนต่อคลาสที่นำมาทดสอบ ในเฟรมเวิร์คบางประเภทจะมีข้อตกลงในการทดสอบ เช่น คลาส CarTest จะบรรจุกรณีทดสอบของคลาส Car
2) จุดประสงค์ของการทดสอบควรระบุไว้อย่างชัดเจน
3) แต่ละกรณีทดสอบประกอบด้วยขั้นตอนการทดสอบ โดยแต่ละขั้นตอนควรมี
   - วัตถุในสถานะที่ต่างกันของคลาสที่จะนำมาทดสอบ ในเฟรมเวิร์คสำหรับการทดสอบอาจเรียกวัตถุเหล่านี้ว่า Test Fixture หรือ Mocking Object
   - ชุดของเมธอดหรือโอเปอร์เรชั่นที่จะถูกเรียกใช้เป็นลำดับเพื่อทดสอบ
   - ชุด Exception ที่อาจจะเกิดขึ้นได้จากคลาสที่กำลังทดสอบ
   - ข้อมูลเพิ่มเติมที่จะช่วยทำความเข้าใจต่อการทดสอบ

8.1.4 การทดสอบเว็บแอพพลิเคชัน

การทดสอบเว็บแอพพลิเคชันเป็นกลุ่มของกิจกรรมที่ใช้เพื่อทดสอบเนื้อหา การเชื่อมต่อ การออกแบบ ส่วนติดต่อผู้ใช้ ฟังก์ชันการทำงาน การนำทาง สภาพการปรับแต่งที่ต่างกัน (เช่น เว็บเบราเซอร์ต่างกัน) รวมทั้งความมั่นคง สมรรถนะ และทดสอบการใช้งานด้วยผู้ใช้จริง (Pressman, 2010)

1) การทดสอบด้านเนื้อหา เป็นการทดสอบเพื่อหาข้อผิดพลาดในเชิงการพิมพ์ผิด หรือการสะกดคำไม่ถูกต้อง รวมถึงความหมายที่คลาดเคลื่อนของข้อความ และการจัดเรียงเนื้อหาที่แสดงต่อผู้ใช้ ระบบเว็บแอพพลิเคชันทั่วไปในปัจจุบันเก็บข้อมูลที่เป็นเนื้อหาไว้ในฐานข้อมูล การทดสอบเนื้อหาจึงอาจจะจำเป็นต้องทดสอบโดยผ่านการดึงข้อมูลจากฐานข้อมูล ซึ่งข้อผิดพลาดพลาดที่เกิดขึ้นเมื่อมีระบบฐานข้อมูลเข้ามาเกี่ยวข้อง คือ การเข้ารหัส (Encode) ข้อมูลตามรหัสภาษา เช่น TIS-620 หรือ UTF-8 เป็นต้น

2) การทดสอบส่วนติดต่อกับผู้ใช้ เป็นการทดสอบเพื่อค้นหาข้อผิดพลาดในส่วนของการแสดงผลและกลไกการนำทางในเว็บแอพพลิเคชัน

- ส่วนติดต่อกับผู้ใช้จะถูกทดสอบเพื่อให้แน่ใจว่าการแสดงผลที่ช่วยในการนำเสนอเนื้อหาสามารถใช้งานได้ถูกต้อง โดยรวมถึงชนิดของฟอนต์ การใช้สี การจัดแบ่งเฟรม รูปภาพหรือตาราง โดยเทคโนโลยีหลักที่เกี่ยวข้อง คือ Cascaded Style Sheet, HTML
- การทดสอบส่วนติดต่อหนึ่งๆ ควรกระทำในลักษณะเชิงหน่วยด้วยการแยกทดสอบการแสดงผลเฉพาะส่วน ในกรณีนี้สามารถใช้เครื่องมือ เช่น Selenium ประกอบกับเฟรมเวิร์คการทดสอบระดับหน่วย เช่น JUnit มาทำการทดสอบได้

3) การทดสอบการออกแบบ เป็นการทดสอบการออกแบบเพื่อหาข้อผิดพลาดเกี่ยวกับการใช้งาน เช่น การโต้ตอบมีลักษณะการใช้งานอย่างไร เข้าใจง่ายหรือไม่ การวางเลย์เอ้าต์ของแอพพลิเคชันเป็นอย่างไร มีกลไกการนำทางอย่างไร เนื้อหาเข้าถึงได้ง่ายหรือไม่ ข้อความอ่านง่าย ทำความเข้าใจง่ายหรือไม่ รวมทั้งกราฟฟิกสามารถแทนสิ่งที่ต้องการสื่อได้หรือไม่ นอกจากนี้ยังรวมถึงลักษณะการแสดงผลอื่นๆ เช่น สี ตัวอักษร และการใช้งานภายใต้ขนาดจอภาพหรือความละเอียดจอที่เว็บแอพพลิเคชันต้องสนับสนุน การทดสอบการออกแบบอาจครอบคลุมถึงการทดสอบเวลาในการตอบสนองว่าผู้ใช้สามารถเข้าถึงฟังก์ชันหรือเนื้อหาที่ต้องการภายในเวลาที่เหมาะสมหรือไม่

3) การทดสอบฟังก์ชันของคอมโพเนนท์ เป็นการทดสอบเพื่อหาข้อผิดพลาดในฟังก์ชันการทำงานของเว็บแอพพลิเคชัน โดยแต่ละฟังก์ชันในเว็บแอพพลิเคชันสามารถพิจารณาเป็น Black-box และใช้เทคนิคการทดสอบสำหรับ Black-box เข้ามาช่วยได้ โดยทั่วไปจะเป็นการจำลองการป้อนข้อมูลของผู้ใช้และตรวจสอบผลลัพธ์ที่แต่ละฟังก์ชันตอบกลับมา เครื่องมือที่เกี่ยวข้องสำหรับขั้นตอนนี้ เช่น Selenium (Richardson, 2012)

4) การทดสอบอื่นๆ

- เพื่อให้เว็บแอพพลิเคชันมีความถูกต้องตามข้อกำหนด จึงจำเป็นต้องทำการทดสอบระบบนำทางให้ทั่วทั้งแอพพลิเคชัน
- เว็บแอพพลิเคชันจำเป็นต้องทดสอบความเข้ากันได้กับสภาพแวดล้อมการทำงานที่ต่างชนิดกัน บางครั้งเรียกการทดสอบแบบนี้ว่า การทดสอบความเข้ากันได้ข้ามเบราเซอร์
- การทดสอบความมั่นคงก็เป็นสิ่งจำเป็นต่อเว็บแอพพลิเคชัน เนื่องจากเว็บแอพพลิเคชันมักสามารถเข้าถึงได้โดยบุคคลภายนอก
- การทดสอบเชิงสมรรถนะเป็นอีกแง่มุมหนึ่งที่สำคัญ เพื่อให้ประมาณการได้ว่าผู้ใช้จำนวนเท่าใดจึงจะทำให้เว็บถึงขีดจำกัดของการบริการ

8.1.5 การเขียนการทดสอบระดับหน่วย

การเขียนการทดสอบระดับหน่วย โดยทั่วไปจะทำเพื่อทดสอบเงื่อนไขหรือข้อกำหนดที่กระทำบนคลาสหลัก ซึ่งก็คือโดเมนคลาสหรือโมเดล (M) ตามหลักของ MVC ซึ่งตัวอย่างการทดสอบระดับหน่วยจะใช้เทคนิคเชิง Test-Driven Development ในการเขียน โดยจะเขียนกรณีทดสอบก่อนเขียนโปรแกรม (Beck, 2002)

ตัวอย่างที่ 8.1 ตัวอย่างการเขียนการทดสอบระดับหน่วย
กำหนดให้มี User Story ดังต่อไปนี้

“ในบทบาทของพนักงาน เราต้องการเก็บข้อมูลเอกสารโดยเอกสารมีรหัสเป็นเลข 8 ตัวและห้ามซ้ำกัน เพื่อให้สามารถค้นหาได้ในภายหลัง”

จากตัวอย่าง User Story ข้างต้น จะสามารถเริ่มเขียนกรณีทดสอบระดับหน่วยได้ดังนี้

```
คลาส ทดสอบ_เอกสาร {
        กรณีทดสอบ_รหัสเอกสาร( ) {
                ตัวแปร เอกสาร1 = สร้างเอกสาร(รหัส: “12345678”)
                ยืนยัน เอกสาร1.ตรวจสอบความถูกต้อง( ) == จริง
                ยืนยัน เอกสาร1.รหัส == “12345678”
        }
}
```

กรณีทดสอบแรกนี้เป็นการสร้างเพื่อทดสอบการทำงานที่เป็นปกติของระบบ คือ เมื่อมีการรับรหัสเข้าไปแล้วควรจะมีการบันทึกลงฐานข้อมูลได้อย่างถูกต้อง เมื่อนำกรณีทดสอบนี้ไปรันครั้งแรกจะปรากฏผลออกมาว่า “ไม่ผ่าน” เนื่องจาก

1) ยังไม่มีการสร้างคลาส **เอกสาร**
2) คลาส **เอกสาร** ยังไม่มีการประกาศรหัส

```
คลาส เอกสาร {
        รหัส เป็นข้อความ
}
```

จากกรณีทดสอบและคลาส **เอกสาร** จะทำให้การทดสอบปรากฏผลออกมาเป็น “ผ่าน” จากนั้นจึงจะเริ่มพิจารณาส่วนอื่นของ User Story ต่อไป ขั้นตอนถัดไปคือการเขียนโปรแกรมสำหรับส่วน “รหัสเป็นตัวอักษร 8 ตัว” เริ่มจากการเขียนกรณีทดสอบคลาส **เอกสาร** ให้รับรหัสเป็นตัวอักษร 8 ตัว โดยเพิ่มกรณีทดสอบเข้าไปยังคลาส **ทดสอบ_เอกสาร**

```
คลาส ทดสอบ_เอกสาร {
        กรณี ทดสอบ_รหัส( ) { ... }
        กรณี ทดสอบ_ความยาวรหัส( ) {
                ตัวแปร เอกสาร1 = สร้างเอกสาร(รหัส: “1234567”)
                ยืนยัน เอกสาร1.ตรวจสอบความถูกต้อง( ) == เท็จ
                ตัวแปร เอกสาร2 = สร้างเอกสาร(รหัส: “123456789”)
                ยืนยัน เอกสาร2.ตรวจสอบความถูกต้อง( ) == เท็จ
        }
}
```

เมื่อนำกรณีการทดสอบทั้งสองมารัน **กรณีทดสอบ_รหัสเอกสาร** จะ “ผ่าน” แต่ กรณี **ทดสอบ_ความยาวรหัส** ที่เขียนเพิ่มเข้าไปจะ “ไม่ผ่าน” เนื่องจากคลาสเอกสารในขณะนี้รับความยาวรหัสทุกแบบโดยไม่มีเงื่อนไข จึงต้องทำการแก้เงื่อนไขรหัสของคลาส **เอกสาร** ให้รับข้อมูลความยาว 8 ตัวอักษรเท่านั้น ดังนี้

```
คลาส เอกสาร {
        รหัส เป็นข้อความ; ความยาว 8 เท่านั้น
}
```

เมื่อแก้ไขคลาสเอกสารแล้ว กรณีทดสอบทั้งสองจะให้ผลลัพธ์เป็น “ผ่าน” ขั้นตอนถัดไปเขียนโปรแกรมสำหรับส่วน “รหัสห้ามซ้ำกัน” เริ่มด้วยการเขียนกรณีทดสอบสำหรับคลาส **เอกสาร** ให้ป้องกันเอกสารที่มีเลขซ้ำกัน

```
คลาส ทดสอบ_เอกสาร {
        กรณี ทดสอบ_รหัสเอกสาร ( ) {...}
        กรณี ทดสอบ_ความยาวรหัส ( ) {...}
        กรณี ทดสอบ_รหัสต้องไม่ซ้ำกัน ( ) {
                ตัวแปร เอกสาร1 = สร้าง เอกสาร(รหัส: “12345678”)
                เอกสาร1.บันทึก( )
                ตัวแปร เอกสาร2 = สร้าง เอกสาร(รหัส: “12345678”)
                ยืนยัน เอกสาร2.ตรวจความถูกต้อง( ) == เท็จ
        }
}
```

เมื่อนำการทดสอบทั้ง 3 กรณีไปรันจะผ่านเพียง 2 กรณี ส่วนกรณีทดสอบที่เพิ่มเข้าไปใหม่ คือ **กรณีทดสอบ_รหัสต้องไม่ซ้ำ** จะมีผลเป็น “ไม่ผ่าน” จากนั้นจึงต้องทำการแก้เงื่อนไขให้คลาส **เอกสาร** มีรหัสไม่ซ้ำกัน

```
คลาส เอกสาร {
        รหัส เป็น ข้อความ; ความยาว 8 เท่านั้น; ไม่ซ้ำ
}
```

เมื่อทำการรันกรณีทดสอบทั้ง 3 กรณีจะได้ผลลัพธ์เป็น “ผ่าน” ทั้งหมด ขั้นตอนถัดไปคือเขียนโปรแกรมสำหรับส่วน “รหัสต้องเป็นตัวเลข” โดยเริ่มจากการเพิ่ม **กรณีทดสอบ_รหัสต้องเป็นตัวเลข** เข้าไปยังคลาส **ทดสอบ_เอกสาร**

```
คลาส ทดสอบ_เอกสาร {
        กรณี ทดสอบ_รหัส( ) {...}
        กรณี ทดสอบ_ความยาวรหัส( ) {...}
        กรณี ทดสอบ_รหัสต้องไม่ซ้ำกัน( ) {...}
        กรณี ทดสอบ_รหัสต้องเป็นตัวเลข( ) {
                ตัวแปร เอกสาร1 = สร้าง เอกสาร(รหัส: “AB345678”)
                ยืนยัน เอกสาร1.ตรวจสอบความถูกต้อง( ) == เท็จ
        }
}
```

เมื่อสั่งทดสอบระดับหน่วยจะพบว่ากรณีทดสอบ “ผ่าน” มี 3 กรณี และ “ไม่ผ่าน” 1 กรณี เนื่องจากคลาส **เอกสาร** ยังไม่ได้รองรับการป้องกันการใส่รหัสที่เป็นตัวอักษรอื่น จึงจำเป็นต้องแก้ไข คลาส **เอกสาร** ดังนี้

```
คลาส เอกสาร {
        รหัส เป็น ข้อความ; ความยาว 8 เท่านั้น; ไม่ซ้ำ; ต้องเข้ากับ /\d+/
}
```

โดย /\d+/ เป็น regular expression และ \d คือ ตัวเลข และ \d+ คือ ตัวเลขตั้งแต่ 1 ตัวขึ้นไป เมื่อทำการแก้ไขคลาสเอกสารให้รับเฉพาะตัวเลขเข้าเป็นรหัสแล้วจะทำการรันกรณีทดสอบอีกครั้ง ซึ่งจะได้ผลลัพธ์เป็น “ผ่าน” ทั้ง 4 กรณี เมื่อทดสอบเงื่อนไขครบถ้วนแล้วจะได้โปรแกรมที่ทำงานได้และการทดสอบระดับหน่วยที่รับประกันว่าโปรแกรมทำงานได้ถูกต้องตาม User Story ที่กำหนด

8.1.6 วิธีการทดสอบเว็บแอพพลิเคชัน

การทดสอบเว็บแอพพลิเคชันในปัจจุบันมีความจำเป็นอย่างมากเพื่อให้สามารถประกันคุณภาพของเว็บแอพพลิเคชันที่พัฒนาขึ้นได้ Selenium เป็นเครื่องมือทดสอบเว็บแอพพลิเคชันที่มีโปรแกรม Selenium IDE สำหรับช่วยเตรียมกรณีทดสอบ โดยทำงานบนเว็บเบราเซอร์ Firefox ซึ่งสามารถบันทึกการทำงานของผู้ใช้รวมทั้งสร้างกรณีทดสอบขึ้นเองด้วยคำสั่งที่ Selenium เตรียมไว้ให้ (Richardson, 2012)

ตัวอย่างคำสั่งใน Selenium

คำสั่งใน Selenium ประกอบไปด้วย 3 ส่วน คือ

1) ชื่อคำสั่ง
2) เป้าหมายของคำสั่ง
3) ข้อความ

ชื่อคำสั่ง	เป้าหมายของคำสั่ง	ข้อความ
Open	/	
WaitForPageToLoad		
AssertTitle	Download	
ClickAndWait	xpath = id(‘header’)/button	

เช่น คำสั่ง open มีเป้าหมายคือ / หมายถึงเปิดหน้าแรกของเว็บ คำสั่ง waitForPageToLoad เป็นคำสั่งที่ไม่มีพารามิเตอร์จึงไม่ต้องใช้ค่าเป้าหมาย

คำสั่งใน Selenium แบ่งออกได้เป็น 3 กลุ่ม คือ

1) Action ใช้สำหรับเปลี่ยนแปลงสถานะของโปรแกรมที่กำลังทดสอบ เช่น open, clickAndWait, type
2) Accessor ใช้สำหรับเข้าถึงสถานะบางส่วนของโปรแกรมแล้วเก็บค่าเข้าตัวแปร
3) Assertion ใช้สำหรับทวนสอบสถานะของโปรแกรมว่าเป็นไปตามที่คาดหวังหรือไม่ ใน Assertion ยังแบ่งออกได้เป็น 3 โหมด คือ
   (a) โหมด assert ถ้าค่าที่ต้องการทดสอบไม่เป็นไปตามที่คาดหวัง การทดสอบที่กระทำอยู่จะถูกยกเลิก
   (b) โหมด verify ถ้าค่าที่ต้องการทดสอบไม่เป็นไปตามที่คาดหวังการทดสอบจะกระทำต่อไปเรื่อยๆ
   (c) โหมด waitFor เป็นโหมดที่การทดสอบจะคอยจนกว่าบางเงื่อนไขจะเป็นจริง ใช้มากในการทดสอบโปรแกรมประเภท AJAX

คำสั่งเพื่อการทดสอบที่ใช้บ่อย

open	สำหรับเปิดหน้าเว็บโดยใช้ URL
click	สำหรับทำการคลิก
clickAndWait	เหมือนคำสั่ง click แต่จะเพิ่มการรอให้หน้าเว็บโหลด
verifyTitle/assertTitle	ตรวจสอบชื่อ (Title) ของหน้าเว็บ
verifyTextPresent	ตรวจสอบว่ามีข้อความอยู่บนหน้าเว็บหรือไม่
verifyElementPresent	ตรวจสอบว่ามีองค์ประกอบของ UI ที่ประกาศโดย HTML บนหน้าเว็บหรือไม่
verifyText	ตรวจสอบข้อความและแท็กที่เกี่ยวข้อง
verifyTable	ตรวจสอบข้อความในตาราง
verifyForPageToLoad	หยุดการทดสอบชั่วคราว จนกระทั่งเว็บหน้าใหม่โหลดเรียบร้อยแล้วจึงทำการทดสอบต่อ
waitForElementPresent	หยุดการทดสอบชั่วคราวจนกว่าองค์ประกอบ UI จะปรากฏบนหน้าเว็บ

การตรวจสอบองค์ประกอบบนหน้าเว็บ

การตรวจสอบองค์ประกอบบนหน้าเว็บเป็นสิ่งหลักที่ต้องทำด้วย Selenium องค์ประกอบบนหน้าเว็บสามารถอ้างถึงได้ด้วยวิธีการที่แตกต่างกัน

1) การอ้างถึงด้วย Identifier (ค่าใน Attribute “id” ของแต่ละแท็ก) สามารถระบุดังนี้ เช่น identifier = loginForm
2) การอ้างอิงด้วย name (ค่าใน Attribute “name” ของแต่ละแท็ก) สามารถระบุดังนี้ เช่น name = username
3) การอ้างอิงด้วย xpath เป็นเส้นทางระบุตำแหน่งในลักษณะต้นไม้ เริ่มต้นจากราก (Root) ของไฟล์ HTML สามารถระบุได้ดังนี้ เช่น xpath = //form[@id = ‘loginForm’] การอ้างอิงด้วยวิธีการนี้มีประโยชน์ชัดเจนเมื่อองค์ประกอบที่ต้องการตรวจสอบไม่มีการระบุ id หรือ name ตามข้อ 1) หรือ 2)

ตัวอย่างการทวนสอบหน้าเว็บ

หน้า Facebook.com มีการล็อกอินด้วยอีเมล์และรหัสผ่าน เพื่อเข้าสู่ระบบการทดสอบ การทดสอบนี้เป็นการทดสอบในกรณีที่มีการใส่ Username และรหัสผ่านไม่ถูกต้องแล้วหน้าเว็บเปลี่ยนไปยังหน้าแจ้งข้อความผิดพลาด

Base URL http://www.facebook.com/

บรรทัดที่	ชื่อคำสั่ง	เป้าหมายของคำสั่ง	ข้อความ
1	Open	/index.php	
2	Type	Email	user@nowhere.com
3	Type	Pass	test
4	clickAndWait	xpath = //input[@value = ‘Login’]	
5	AssertTextPresent	Incorrect Email	

บรรทัดที่ 1 ทำการเปิดเว็บ http://www.facebook.com/index.php

บรรทัดที่ 2 พิมพ์คำว่า user@nowhere.com ลงในช่องอีเมล์ของบรรทัดนี้ Identifier เป็นตัวระบุตำแหน่งให้กับการพิมพ์

บรรทัดที่ 3 พิมพ์คำว่า test ลงในช่องรหัสผ่าน ใช้ Identifier เพื่อระบุตำแหน่งเช่นกัน

บรรทัดที่ 4 คลิกปุ่ม Login แล้วรอโหลดหน้าใหม่ โดยใช้ตัวระบุตำแหน่งของปุ่มเป็น xpath = //input[@value = ‘Login’] หมายความว่า ปุ่มที่ต้องการเป็นแท็ก input ที่มี Attribute ‘value’ เป็นค่า “Login” เนื่องจากไม่สามารถใช้ id ในการระบุปุ่มได้

บรรทัดที่ 5 เป็นการยืนยันว่าการเข้าระบบไม่ได้นั้นดำเนินการถูกตามขั้นตอนหรือไม่ โดยการตรวจสอบคำว่า “Incorrect Email” บนหน้าเว็บที่เปิดขึ้นใหม่จากการคลิกปุ่ม Login

8.2 การนำไปใช้ (Software Deployment)

การนำไปใช้คือ กิจกรรมที่นำระบบซอฟต์แวร์ที่พัฒนาขึ้นไปติดตั้งเพื่อการใช้งาน เป็นกระบวนการที่เกี่ยวข้องกับหลายกิจกรรม เช่น การปล่อยซอฟต์แวร์ การปรับปรุงรุ่น เป็นต้น โดยกิจกรรมย่อยจะขึ้นกับซอฟต์แวร์นั้นๆ เนื่องจากในทางปฏิบัติแล้วการนำซอฟต์แวร์ไปใช้มีความแตกต่างกันค่อนข้างมาก และมักขึ้นกับเครื่องมือทางวิศวกรรมซอฟต์แวร์ที่นำมาจัดการกระบวนการด้วย (Pressman, 2010)

8.2.1 กิจกรรมในกระบวนการนำไปใช้ มีดังต่อไปนี้

1) การปล่อยซอฟต์แวร์ เป็นสิ่งที่กระทำหลังจากกระบวนการพัฒนาเสร็จสิ้นลง การปล่อยซอฟต์แวร์เป็นขั้นตอนการเตรียมความพร้อมของระบบและประกอบทุกอย่างเข้าด้วยกัน ก่อนนำตัวระบบไปยังจุดติดตั้งของผู้ใช้ กระบวนการนี้จึงมีความจำเป็นในการจัดสรรทรัพยากรบุคคลที่ต้องไปทำงาน ณ จุดติดตั้งโปรแกรม รวมทั้งดำเนินกิจกรรมอื่นเพื่อสนับสนุนการนำไปใช้ของลูกค้า

2) การติดตั้งและเปิดใช้งาน เป็นกิจกรรมเพื่อเริ่มต้นใช้งานชิ้นส่วนของซอฟต์แวร์ในระบบ โดยปกติการติดตั้งและเปิดใช้งานของระบบขนาดเล็กจะเป็นการใช้คำสั่งเพื่อติดตั้ง เช่น ระบบจัดการทรัพยากรวิสาหกิจอาจใช้เครื่องแม่ข่ายหลายเครื่องสำหรับฐานข้อมูล และแอพพลิเคชันแม่ข่าย กรณีที่เป็นระบบขนาดใหญ่มาก เช่น Twitter ที่มีซอฟต์แวร์กระจายตัวอยู่ตามเครื่องต่างๆ ในระดับพัน หรือหมื่นเครื่อง กลไกการติดตั้งจะแตกต่างออกไป ซึ่งอาจจะใช้ระบบกระจายประเภท Peer-to-Peer ช่วยในการติดตั้ง เป็นต้น

3) การปิดใช้งาน เป็นกิจกรรมที่หมายถึงการปิดใช้ชิ้นส่วนของซอฟต์แวร์ในระบบ กิจกรรมนี้มักจะทำเพื่อให้สามารถดำเนินกิจกรรมอื่นต่อไปได้ เช่น ต้องปิดการใช้งานก่อนจึงจะปรับปรุงชิ้นส่วนของซอฟต์แวร์ที่เกี่ยวข้องได้ เป็นต้น

4) การดัดแปลง เป็นกระบวนการในการปรับเปลี่ยนระบบซอฟต์แวร์ที่ติดตั้งอยู่ก่อนแล้วให้สามารถทำงานได้ในสภาวะการทำงานอื่นที่ต้องการปรับชิ้นส่วนซอฟต์แวร์ภายนอกบางชิ้น หรือการเปลี่ยนรุ่นของระบบปฏิบัติการที่ซอฟต์แวร์ทำงานอยู่ เป็นต้น การดัดแปลงจะต่างจากการปรับปรุงตรงที่การปรับปรุงจะเริ่มจากการติดตั้งซอฟต์แวร์ที่นำมาจากผู้พัฒนา

5) การปรับปรุง เป็นการเปลี่ยนหรือแทนที่ซอฟต์แวร์รุ่นก่อนหน้าด้วยรุ่นที่ใหม่กว่า โดยทั่วไปจะมีทั้งการแทนที่ทั้งระบบซอฟต์แวร์หรือแทนที่เฉพาะบางส่วน

6) การติดตามรุ่น ระบบติดตามรุ่นจะช่วยผู้ใช้งานระบบในการค้นหาและติดตั้งการปรับปรุงชิ้นส่วนซอฟต์แวร์ของระบบ โดยปกติระบบในลักษณะนี้จะสร้างสำหรับแอพพลิเคชันประเภทตั้งโต๊ะ ตัวอย่างเช่น เว็บเบราเซอร์ Google Chrome มีระบบติดตามรุ่นที่ตรวจสอบและดาวน์โหลดซอฟต์แวร์รุ่นใหม่ที่ทำงานอยู่เบื้องหลังในขณะที่มีผู้ใช้ใช้งานเบราเซอร์

7) การยกเลิกการติดตั้ง การยกเลิกการติดตั้งเป็นกิจกรรมที่ตรงข้ามกับการติดตั้ง โดยการลบชิ้นส่วนหนึ่งหรือทั้งหมดของระบบออกเมื่อไม่ต้องการใช้แล้ว สิ่งที่ต้องระมัดระวังคือการยกเลิกการติดตั้งอาจกระทบกับการปรับแต่งของระบบโดยรวม โดยชิ้นส่วนของซอฟต์แวร์ส่วนอื่นอาจใช้งานไม่ได้หากส่วนใดส่วนหนึ่งของระบบถูกลบออกจากการยกเลิกการติดตั้งซอฟต์แวร์อื่นได้

8) การสิ้นสุดการสนับสนุน เป็นกลไกการกำหนดให้ซอฟต์แวร์นั้นล้าสมัยและประกาศไม่สนับสนุนต่อโดยผู้พัฒนา ซอฟต์แวร์หลายตัว เช่น Windows XP หรือ Java Development Kit 1.4 มีลักษณะเป็นซอฟต์แวร์ที่สิ้นสุดการสนับสนุนแล้ว เป็นต้น

8.2.2 บทบาทในกระบวนการนำไปใช้

เนื่องจากกระบวนการนำไปใช้เกี่ยวข้องโดยตรงกับความหลากหลายของผลิตภัณฑ์ เมื่อต้องจัดการความซับซ้อนที่เกิดขึ้น ทำให้ต้องมีการกำหนดบทบาทเฉพาะขึ้นสำหรับจัดการกระบวนการนำไปใช้

1) ผู้พัฒนาแอพพลิเคชัน ผู้พัฒนาแอพพลิเคชันเป็นบทบาทในระยะ Pre-production โดยผู้พัฒนามีหน้าที่สร้างระบบตามการวิเคราะห์และการออกแบบ

2) วิศวกรสร้างและปล่อยซอฟต์แวร์ วิศวกรสร้างและปล่อยซอฟต์แวร์เป็นบทบาทที่มีหน้าที่คอมไพล์และประกอบต้นรหัสให้อยู่ในรูปของผลิตภัณฑ์ที่เสร็จสมบูรณ์ โดยมีแง่มุมที่น่าสนใจในกระบวนการ เช่น ความสามารถในการทำซ้ำ เป็นการประกอบต้นรหัสชิ้นส่วนซอฟต์แวร์อื่นรวมถึงข้อมูลและระบบภายนอกเข้าด้วยกัน เพื่อให้สามารถรับประกันความมีเสถียรภาพในการทำงานของระบบหรือมีความสอดคล้อง ซึ่งเป็นการสร้างกรอบงานที่มีเสถียรภาพสำหรับการพัฒนาและประกอบชิ้นส่วนของซอฟต์แวร์เข้าด้วยกัน บทบาทนี้อยู่ในระยะ Pre-production

3) ผู้จัดการการปล่อยซอฟต์แวร์ ผู้จัดการการปล่อยซอฟต์แวร์เป็นผู้ทำหน้าที่ประสานงานระหว่างแต่ละหน่วยในการพัฒนาเพื่อให้การปล่อยซอฟต์แวร์และชุดของการปรับปรุงสามารถนำส่งลูกค้าได้อย่างเรียบร้อยและทันเวลา รวมทั้งมีส่วนช่วยในการตั้งกระบวนการเพื่อให้การปล่อยซอฟต์แวร์เป็นไปอย่างมีประสิทธิภาพ บทบาทนี้อยู่ในระยะ Pre-production ประเด็นสำคัญที่ต้องคำนึงถึงในบทบาทของผู้จัดการการปล่อยซอฟต์แวร์ อาจรวมถึงทรัพย์สินทางปัญญาที่เกี่ยวข้อง ความเสี่ยง การเปลี่ยนแปลงที่ลูกค้าต้องการเร่งด่วน การพัฒนาคุณลักษณะเพิ่มเติม เป็นต้น

4) ผู้ดูแลระบบ มีหน้าที่ดูแลรักษาระบบคอมพิวเตอร์ ซึ่งอาจรวมถึงระบบเครือข่าย บทบาทนี้มีหน้าที่อยู่ในระยะ Production โดยต้องรับผิดชอบดูแลให้ระบบของลูกค้าสามารถทำงานได้อย่างราบรื่น และผู้ดูแลระบบอาจมีความจำเป็นต้องแก้ปัญหาเฉพาะหน้า เมื่อเกิดเหตุการณ์ที่ไม่คาดคิดขึ้น เทคโนโลยีในปัจจุบัน เช่น Virtualization มีบทบาทในการลดความซับซ้อนของการดูแลระบบ

5) ผู้ดูแลระบบฐานข้อมูล มีหน้าที่รับผิดชอบในการบำรุงรักษาและดูแลระบบฐานข้อมูลของซอฟต์แวร์รวมถึงการทำงานร่วมกับผู้พัฒนาในการกำหนดสิทธิ์การเข้าถึงฐานข้อมูลแต่ละส่วน ผู้ดูแลฐานข้อมูลอาจมีหน้าที่เพิ่มเติม เช่น การสำรองและการกู้คืนข้อมูลของระบบ รวมทั้งการเตรียมทรัพยากรระดับฮาร์ดแวร์เพื่อทำซ้ำข้อมูลขณะทำงานจริงเพื่อรับจำนวนผู้ใช้ที่อาจจะเพิ่มขึ้น เป็นต้น บทบาทนี้เป็นบทบาทในระยะ Production

6) ผู้ประสานงานการปล่อยซอฟต์แวร์ เป็นบทบาทที่มีหน้าที่ประสานงานการนำซอฟต์แวร์ไปใช้จากระยะ Pre-production ไปสู่ระยะ Production โดยจะประสานงานระหว่างกลุ่มทำงานหลายกลุ่มเพื่อให้เป้าหมายในการนำซอฟต์แวร์ไปใช้บรรลุผล ผู้ประสานงานการปล่อยซอฟต์แวร์ต่างจากผู้จัดการการปล่อยซอฟต์แวร์ตรงที่ผู้จัดการการปล่อยซอฟต์แวร์มักเน้นไปยังการวางแผนรับมือต่อการเปลี่ยนแปลงของซอฟต์แวร์ บทบาทนี้อยู่ในระยะ Production

8.3 การบำรุงรักษาซอฟต์แวร์

การบำรุงรักษาซอฟต์แวร์เป็นกระบวนการทางวิศวกรรมซอฟต์แวร์ เพื่อแก้ไขข้อผิดพลาดของซอฟต์แวร์ที่ส่งมอบแล้วให้สามารถทำงานได้ถูกต้อง หรือเพื่อทำการปรับปรุงประสิทธิภาพของซอฟต์แวร์ และอาจรวมถึงคุณสมบัติด้านอื่นๆ เช่น การแก้ไขช่องโหว่ที่มีผลต่อความมั่นคงของซอฟต์แวร์ เป็นต้น (Pressman, 2010)

กระบวนการในการบำรุงรักษาซอฟต์แวร์ แบ่งออกเป็นขั้นตอนดังนี้

1) วางแผนกระบวนการเตรียมความพร้อมของซอฟต์แวร์และถ่ายโอนซอฟต์แวร์ วางแผนการบำรุงรักษา การเตรียมการสำหรับจัดการปัญหาที่จะพบระหว่างการพัฒนาและติดตามการจัดการ Configuration

2) วิเคราะห์ปัญหาและการแก้ไข นักพัฒนาที่รับผิดชอบการบำรุงรักษาจะทำการวิเคราะห์การร้องขอจากลูกค้าเพื่อเปลี่ยนแปลงตัวซอฟต์แวร์ จากนั้นทำการยืนยันว่ามีปัญหาเกิดขึ้นจริงตามการร้องขอโดยการทำซ้ำปัญหานั้นๆ และวิเคราะห์ว่าปัญหาดังกล่าวสมเหตุสมผลหรือไม่ เมื่อวิเคราะห์แล้วนักพัฒนาจะเสนอแนวทางแก้ไขและสร้างต้นรหัสเพื่อแก้ปัญหานั้นเพื่อขออนุมัติการแก้ไข

3) พิจารณาการดำเนินการแก้ไข

4) ยอมรับกระบวนการแก้ไขโดยทำการยืนยันว่าสิ่งที่เปลี่ยนแปลงในระบบสามารถแก้ไขปัญหาได้จริง

5) การบำรุงรักษาอาจรวมถึงการย้ายแอพพลิเคชันข้ามแพล็ตฟอร์ม ซึ่งเป็นกระบวนการพิเศษที่อยู่นอกเหนือการบำรุงรักษาปกติ

6) การสิ้นสุดการสนับสนุนของซอฟต์แวร์ก็เป็นกระบวนการบำรุงรักษาเช่นกัน โดยเป็นสิ่งที่เกิดขึ้นเพียงครั้งเดียวต่อซอฟต์แวร์ 1 ระบบ เพื่อถอนการติดตั้งและประกาศยกเลิกการสนับสนุนโดยผู้พัฒนา

ประเภทของการบำรุงรักษาซอฟต์แวร์

1) การบำรุงรักษาเชิงแก้ไข คือ การเปลี่ยนแปลงที่กระทำต่อระบบหลังส่งมอบแล้ว เพื่อแก้ไขปัญหาโดยตรงหลังจากพบว่าปัญหาคืออะไร

2) การบำรุงรักษาเชิงดัดแปลงเป็นการเปลี่ยนแปลงที่กระทำต่อระบบหลังส่งมอบแล้ว เพื่อให้ซอฟต์แวร์สามารถทำงานได้ในสภาพแวดล้อมที่เปลี่ยนไปจากเดิม

3) การบำรุงรักษาเชิงสมบูรณ์ คือ การเปลี่ยนแปลงที่กระทำต่อระบบหลังส่งมอบแล้วเพื่อทำให้ระบบสมบูรณ์ยิ่งขึ้น เช่น การปรับปรุงประสิทธิภาพหรือเพิ่มความสามารถที่ช่วยในการบำรุงรักษาเข้าสู่ตัวระบบ

4) การบำรุงรักษาเชิงป้องกัน คือ การเปลี่ยนแปลงที่กระทำต่อระบบหลังส่งมอบแล้ว เพื่อตรวจสอบหรือแก้ไขข้อผิดพลาดที่แฝงอยู่ก่อนที่จะกลายเป็นข้อผิดพลาดร้ายแรงที่อาจก่อให้เกิดความเสียหายต่อระบบได้

8.4 บทสรุป

ในบทนี้ได้กล่าวถึงการทดสอบซอฟต์แวร์ในระดับต่างๆ เช่น ระดับหน่วยซึ่งอยู่ใกล้นักพัฒนามากที่สุด ระดับบูรณาการซึ่งเป็นการทดสอบการทำงานร่วมกันของชิ้นส่วนซอฟต์แวร์ย่อย เป็นต้น จากนั้นได้กล่าวถึงประเภทของการทดสอบซอฟต์แวร์แบบ White-Box และ Black-Box และแนวทางการทดสอบแอพพลิเคชันประเภทต่างๆ ในส่วนถัดมาได้กล่าวถึงการเขียนกรณีทดสอบโดยใช้เทคนิคของ Test-Driven Development รวมทั้งการทดสอบเว็บแอพพลิเคชันด้วย Selenium ในบทนี้ยังได้กล่าวถึงกระบวนการการนำไปใช้และกิจกรรมรวมถึงบทบาทที่เกี่ยวข้อง ในส่วนสุดท้ายของบทนี้ได้กล่าวถึงการบำรุงรักษาซอฟต์แวร์ ขั้นตอนการแก้ไขจุดบกพร่องและการบำรุงรักษาประเภทต่างๆ

8.5 คำถามท้ายบทที่ 8

1) การทดสอบซอฟต์แวร์คืออะไร

2) การทดสอบเชิงหน่วยคืออะไร

3) การทดสอบแบบถดถอยคืออะไร

4) การทดสอบแบบ Black-box คืออะไร

5) จงเขียนการทดสอบเชิงหน่วยสำหรับเรื่องจากผู้ใช้ *“ในบทบาทของพนักงานทะเบียนราษฎร์เราต้องการเก็บข้อมูลประชาชน โดยมีรหัสบัตรประชาชนเป็นเลข 13 ตัวและห้ามซ้ำกันเพื่อให้สามารถอ้างอิงบุคคลได้”*

6) การปรับปรุงรุ่นคืออะไร

7) จงเปรียบเทียบการติดตามรุ่นใน Google Chrome และ Mozilla Firefox

8) จงอธิบายการปรับปรุงรุ่นใน Ubuntu

9) บทบาทที่เกี่ยวข้องกับกระบวนการนำไปใช้มีบทบาทอะไรบ้าง

10) การบำรุงรักษาซอฟต์แวร์คืออะไร มีกี่ประเภท