รายงานโครงการวิจัยประจำปีงบประมาณ 2546

เรื่อง

การทดสอบซอฟต์แวร์อัตโนมัติด้วยแฟ้มเชื่อมประสาน XML

(ระยะที่ 1)

Automated Software Testing using XML File Interface

(Phase 1)

โดย

พรฤดี เนติโสภากุล

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2546

รายงานโครงการวิจัยประจำปีงบประมาณ 2546

เรื่อง

การทดสอบซอฟต์แวร์อัตโนมัติด้วยแฟ้มเชื่อมประสาน XML

(ระยะที่ 1)

Automated Software Testing using XML File Interface

(Phase 1)

โดย

พรฤดี เนติโสภากุล

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2546

การทดสอบซอฟต์แวร์อัตโนมัติด้วยแฟ้มเชื่อมประสาน XML (ระยะที่1)

Automated Software Testing using XML File Interface (Phase 1)

พรฤดี เนติโสภากุล

บทคัดย่อ

องค์ประกอบสำคัญองค์ประกอบหนึ่งของเครื่องมือทดสอบซอฟต์แวร์ คือคลังเก็บกรณีทดสอบ ที่ผู้เชี่ยวชาญในการทดสอบสร้างขึ้น การทดสอบซอฟต์แวร์แบบถดถอย สามารถทำได้โดยอัตโนมัติ โดยการนำกรณีทดสอบมาใช้ซ้ำ งานวิจัยนี้ได้ศึกษาความเป็นไปได้และรูปแบบที่เหมาะสมในการนำ แฟ้มเชื่อมประสาน มาช่วยให้การทดสอบซอฟต์แวร์เป็นไปโดยสะดวก รวดเร็ว ลดภาระของผู้ทดสอบ แฟ้มรูปแบบ XML เป็นแฟ้มที่ได้รับการยอมรับเป็นมาตรฐานในระบบเครือข่ายคอมพิวเตอร์โลก งานวิจัยนี้ได้นำเสนอ สถาปัตยกรรม ของเครื่องมือทดสอบซอฟต์แวร์โดยอัตโนมัติ และรูปแบบที่เหมาะสมของแฟ้ม XML ที่ใช้

Abstract

One of the important components of any software testing system is a test case repository which created by a number of software testing experts. The regression testing is able to conduct automatically by reusing the existing test cases. This research investigates the feasibility and the suitability of XML format to be used as interface files for a software testing system. The purpose is to aid a software tester to work easily, quickly and less tedious. A XML format is accepted as a standard for World Wide Web. Therefore, this research demonstrated architecture of an automated software testing tool and presented an XML format to be used as interface files.

สารบัญ

สารบัญภาพ

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ปัญหาและความสำคัญของงานวิจัย

เนื่องด้วยผู้เชี่ยวชาญในการทดสอบซอฟต์แวร์ต้องมีการแลกเปลี่ยนกรณีทดสอบอยู่เสมอ ซึ่งเครื่องมือที่ใช้ในการจัดการแลกเปลี่ยนกรณีทดสอบที่ใช้อาจแตกต่างกันไป ดังนั้นองค์ประกอบสำคัญองค์ประกอบหนึ่งของเครื่องมือทดสอบซอฟต์แวร์ คือคลังเก็บกรณีทดสอบ โดยสามารถนำกรณีทดสอบในคลังมาใช้ซ้ำเพื่อทำการทดสอบซอฟต์แวร์แบบถดถอยซึ่งสามารถทำได้โดยอัตโนมัติ แต่ด้วยที่ผู้เชี่ยวชาญเลือกใช้เครื่องมือในการจัดการต่างกันจึงจำเป็นต้องมีการติดต่อสื่อสารในรูปแบบที่เหมาะสม ซึ่งการนำแฟ้มเชื่อมประสานมาใช้ก็สามารถช่วยให้การทดสอบซอฟต์แวร์เป็นไปโดยสะดวกรวดเร็วในการแลกเปลี่ยนกรณีทดสอบ การเป็นที่ยอมรับให้เป็นแฟ้มมาตรฐานในระบบเครือข่ายคอมพิวเตอร์โลกสำหรับแฟ้มรูปแบบ XML ทำให้การนำมาประยุกต์ใช้เป็นแฟ้มเชื่อมประสานสำหรับการสื่อสารแลกเปลี่ยนกรณีทดสอบจึงมีความเป็นไปได้ งานวิจัยนี้ได้ศึกษาความเป็นไปได้และรูปแบบที่เหมาะสมในการนำแฟ้มเชื่อมประสานมาใช้ในการทดสอบซอฟต์แวร์ซึ่งจะนำเสนอสถาปัตยกรรมของเครื่องมือทดสอบซอฟต์แวร์โดยอัตโนมัติ และรูปแบบที่เหมาะสมของแฟ้ม XML ที่ใช้

1.2 วัตถุประสงค์ของงานวิจัย

1. ศึกษาการนำแฟ้มรูปแบบ XML มาให้เป็นแฟ้มเชื่อมประสานระหว่างระบบหรือระหว่างเครื่องมือการทดสอบซอฟต์แวร์
2. ศึกษาการนำแฟ้มเชื่อมประสาน XML มาช่วยในการทดสอบซอฟต์แวร์
3. นำเสนอสถาปัตยกรรมของเครื่องมือทดสอบซอฟต์แวร์โดยอัตโนมัติ และรูปแบบที่เหมาะสมของแฟ้ม XMLที่ใช้

1.3 ขอบเขตของงานวิจัย

งานวิจัยนี้เป็นการศึกษาความเป็นไปได้และรูปแบบที่เหมาะสมสำหรับแฟ้มเชื่อมประสานที่สามารถช่วยให้การทดสอบซอฟต์แวร์เป็นไปโดยสะดวกรวดเร็ว และลดภาระผู้เชี่ยวชาญหรือผู้ทดสอบ ในที่นี้จะทำการศึกษาความเป็นไปได้ในการเชื่อมประสานเพื่อแลกเปลี่ยนกรณีทดสอบ

ระหว่างฐานข้อมูลและการนำกรณีทดสอบที่ได้จากแฟ้มเชื่อมประสานไปใช้ในการทดสอบ ซึ่งจะนำเสนอสถาปัตยกรรมของเครื่องมือทดสอบซอฟต์แวร์โดยอัตโนมัติและรูปแบบที่เหมาะสมของแฟ้มเชื่อมประสาน XML

1.4 ประโยชน์ที่คาดว่าจะได้รับ

- นำไปสู่การออกแบบระบบหรือเครื่องมือสำหรับการแลกเปลี่ยนกรณีทดสอบผ่านแฟ้มเชื่อมประสาน XML
- สถาปัตยกรรม ของเครื่องมือทดสอบซอฟต์แวร์อัตโนมัติ และรูปแบบของแฟ้มเชื่อมประสาน XML สามารถนำไปศึกษาและพัฒนาเป็นระบบหรือเครื่องมือที่ช่วยให้การทดสอบซอฟต์แวร์เป็นไปโดยสะดวกมากขึ้น

บทที่ 2
ทฤษฎีที่เกี่ยวข้อง

2.1 แฟ้มรูปแบบ XML (Extensible Markup Language) (Castro 2001)

ภาษา XML เป็นภาษา Markup ที่มีโครงสร้างของภาษาเป็นข้อความ (Text-based) ซึ่งมีการจัดเตรียมความสามารถด้านโครงสร้างและสารสนเทศที่ใช้นิยามข้อมูล หรือ Metadata และ XML ก็เป็นส่วนหนึ่งของภาษา SGML)Standard Generalized Markup Language) ที่นิยมใช้และได้รับพัฒนาให้มีประสิทธิภาพในการทำงานบนเว็บเพียงแต่มีขนาดเล็กกว่า

Markup จะประกอบด้วยแท็ก) Tag) หรือ โทเคน (Token) เป็นพื้นฐานสำคัญที่ใช้เพิ่มเข้าไปในข้อความ โดยที่แท็กจะห่อหุ้มข้อความที่เรียกว่าโค้ด) Code) ซึ่งแท็กแต่ละแบบจะส่งผลให้มีการปรับเปลี่ยนรูปแบบมุมมองหรือความหมายของข้อความที่ห่อหุ้มอยู่ต่าง ๆ กันออกไป

การใช้งาน XML ร่วมกับข้อมูลภายนอกมีหลายวิธี เช่นการใช้ XML ในการเปลี่ยนแปลงรูปแบบของข้อมูลให้กับระบบดั้งเดิม (Legacy System) หรือการใช้ XML เป็นข้อมูลบนเว็บเพจโดย HTML จะเป็นเพียงการเก็บรูปแบบและการแสดงผล เป็นต้น นอกจากนี้ภาษา XML ยังเป็นภาษาที่ใช้นิยามข้อมูลให้กับแอพพลิเคชั่นอื่น ๆ ซึ่งมนุษย์สามารถอ่านเอกสารและเข้าถึงเนื้อหาของเอกสารหรือโครงสร้างได้ง่าย โดยข้อมูลในเอกสารจะเป็นอิสระจากรูปแบบการแสดงผล นั่นคือในเอกสารจะไม่มีการบอกถึงรูปแบบการแสดงผลข้อมูลใด ๆ

เอกสาร XML จะประกอบไปด้วยอิลิเมนต์ต่าง ๆ โดยมี อิลิเมนต์ PROLOG และ อิลิเมนต์ DOCUMENT เป็นส่วนประกอบสำคัญ โดยโครงสร้างกายภาพของ XML ประกอบไปด้วยหน่วยจัดเก็บข้อมูลที่เรียกว่า *เอนติตี้* (Entity) ซึ่งมันสามารถส่วนใดส่วนหนึ่งของเอกสารหรืออาจเป็นส่วนประกอบภายนอกที่นำเข้ามาใช้ในเอกสารก็ได้ โดยเอนติตี้แต่ละตัวจะสามารถระบุได้ด้วยชื่อที่เป็นเอกลักษณ์ เอนติตี้จะถูกประกาศอยู่ในส่วนอิลิเมนต์ PROLOG และนำมาอ้างอิงใช้ในอิลิเมนต์ DOCUMENT ซึ่งส่วนการประกาศเอนติตี้เป็นเพียงส่วนที่ทำให้โปรแกรมประมวลผลทราบว่ามีเอนติตี้อะไรในเอกสารบ้าง ซึ่งส่วนประกาศนี้อาจเขียนอยู่ในเอกสาร DTD เพื่อให้สามารถใช้งานได้ทุกทุกที่ถ้ามีการประกาศใช้ DTD

เอนติตี้มี 2 ชนิดได้แก่ ชนิดวิภาค (Parsed Entity) และชนิดอวิภาค (Unparsed Entity) โดยแต่ละเอนติตี้สามารถเป็นชนิดใดชนิดหนึ่งเท่านั้น เอนติตี้ชนิดวิภาคจะใช้บรรจุข้อมูลเท็กซ์ที่จะกลายมาเป็นเนื้อหาเอกสารของ XML หลังจากที่ผ่านการประมวลผลแล้ว นั่นคือ เอนติตี้ชนิดนี้สามารถอ่าน

และถอดความได้ด้วยโปรแกรมประมวลผล XML แล้วข้อมูลภายในเอนติตี้จะกลายเป็นข้อความของเนื้อเอกสารในตำแหน่งเดียวกับที่เอนติตี้อ้างอิงอยู่หลังผ่านการประมวลผล ส่วนเอนติตี้ชนิดอวิภาคนั้นจะใช้บรรจุข้อมูลอะไรก็ได้ไม่จะเป็นต้องเป็นเท็กซ์ แต่ปกตินิยมใช้เก็บข้อมูลไฟล์ไบนารี ซึ่งโปรแกรมประมวลผล XML จะไม่ประมวลผลเอนติตี้นี้โดยตรงแม้ว่าจะเก็บข้อมูลในรูปแบบเท็กซ์ก็ตาม แต่เอนติตี้ชนิดนี้จะต้องมีส่วน *โนเทชัน* (Notation) เพื่อใช้บอกรูปแบบหรือชนิดข้อมูลของเอนติตี้

กฎและไวยากรณ์ของ XML จะคล้ายกับ HTML เนื่องจากมีภาษาแม่เดียวกัน สิ่งที่สำคัญของไวยากรณ์ของ XML ได้แก่ แท็กเปิดและแท็กปิด และแอตทริบิวต์ (Attribute) สำหรับ XML แล้วทุก ๆ อิลิเมนต์จะต้องมีทั้งแท็กเปิดและแท็กปิดเสมอ แต่อิลิเมนต์บางตัวอาจไม่มีข้อมูลภายในก็ได้ เช่น <GENUS> </GENUS> หรือ <GENUS/> เป็นต้น ส่วนแอตทริบิวต์คือค่าข้อมูลที่อยู่ในอิลิเมนต์ที่ไม่ต้องสร้างเป็นส่วนหนึ่งของข้อมูลภายในอิลิเมนต์ เช่น ZONE ในอิลิเมนต์เป็นต้น

```
<PLANT    ZONE = 3>
    <COMMON> Columbine</COMMON>
</PLANT>
```

2.1.1 การกำหนดประเภทเอกสาร XML (Walmsley 2002)

เอกสาร XML สามารถกำหนดประเภทเอกสารตามกฎไวยากรณ์ที่ออกแบบได้ ซึ่งกฎหรือข้อตกลงดังกล่าวก็คือโครงสร้างเอกสาร XML ซึ่งปัจจุบันสามารถออกแบบได้ 2 รูปแบบได้แก่ DTD และ XML-Data Schema โดยทั้งสองรูปแบบจะถูกนำไปใช้สร้างเอกสาร XML ที่ถูกต้องตามที่ออกแบบไว้ตามแต่ละประเภทเอกสาร ได้แก่ เอกสาร Valid

1) ประเภทเอกสาร DTD (Document Type Definition)

DTD เป็นกฎข้อบังคับโครงสร้างเอกสาร หรือข้อตกลงให้ผู้สร้างนำไปใช้สร้างเอกสารตัวใหม่ที่มีคุณลักษณะ และเป็นเอกสารประเภทเดียวกันกับเอกสารต้นแบบ และDTD จะเป็นตัวที่นำไปตรวจสอบความถูกต้องของเอกสาร XML ที่ถูกสร้างบนพื้นฐานโครงสร้างที่กำหนดใน DTD นั้น ๆ เพื่อให้มั่นใจว่าเอกสารที่ได้สอดคล้องกับข้อตกลงที่กำหนดไว้

การประกาศประเภทเอกสาร DTD ใน XML จะประกาศอยู่ต่อท้ายส่วนการประกาศ XML ในอิลิเมนต์ PROLOG แต่อยู่ก่อนอิลิเมนต์ DOCUMENT โดยอาจสร้าง DTD เป็นไฟล์ภายนอกที่ใช้อ้างอิงในการตรวจสอบความถูกต้อง เช่น

<?xml version= "1.0"?>

<!DOCTYPE wildflowers SYSTEM "Wldflr.dtd">

2) XML-Data Schema

เนื่องจากผู้พัฒนาโปรแกรมหลายแห่งและบริษัทไมโครซอฟท์ต่างคิดว่าความสามารถของ DTD มีไม่เพียงพอสำหรับโปรแกรมที่ทำงานด้วย XML ที่มีอยู่ปัจจุบัน ด้วยความร่วมมือของหลาย ๆ บริษัทหรือหน่วยงานจึงมีการพัฒนารูปแบบใหม่ที่มีความสามารถ และ ยืดหยุ่นกว่าเพื่อหลีกเลี่ยงข้อจำกัดบางข้อของ DTD นั่นคือ การใช้เครื่องมือพิเศษเพื่อบำรุงรักษาหรือช่วยให้ผู้ใช้เข้าใจไวยากรณ์ของ DTD

XML-Data มีรากฐานภาษามาจาก XML และ DTD ดังนั้นเอกสาร XML-Data สามารถตรวจสอบความถูกต้องได้ถ้าเอกสารอ้างอิงถึงส่วน DTD นั้น เอกสาร XML-Data ประกอบด้วยอิลิเมนต์บางอย่างเหมือน XML นั่นคือ อิลิเมนต์ PROLOG และ อิลิเมนต์ DOCUMENT ซึ่งเป็นส่วนประกอบสำคัญ แต่ต่างกันที่เอกสาร Schema ไม่มีส่วนที่เรียกว่า DTD หรือโครงสร้างเอกสารที่นิยามไว้ในอิลิเมนต์ Schema

ส่วนประกอบของ XML-Data Schema แบ่งเป็นระดับได้ 2 ระดับ ได้แก่ การประกาศระดับบน (Top-level Declaration) ซึ่งอิลิเมนต์และแอตทริบิวต์ทุกตัวที่ประกาศไว้ภายใต้อิลิเมนต์ Schema ในส่วนนี้จะสามารถนำไปใช้ในส่วนประกาศอื่นที่อยู่ในเอกสารได้ และการประกาศระดับท้องถิ่น (Local-level Declaration) ซึ่งเป็นส่วนที่ไม่ได้ประกาศไว้ระดับบน โดยสามารถนำมาใช้อ้างอิงได้เฉพาะในส่วนการประกาศของตนเท่านั้น

2.1.2 XSL (Extensible Stylesheet Language) (Cagle 2001)

XML มีภาษาที่ใช้ร่วมเพื่อการจัดรูปแบบการแสดงผลในรูปแบบสไตล์ชีต นั่นคือ XSL (Extensible Stylesheet Language) ซึ่งเป็นแอพพลิเคชั่นหนึ่งของ XML ดังนั้นโครงสร้างภาษาและไวยากรณ์จึงเหมือนกับ XML การสร้าง XSL สามารถทำได้จากเครื่องมือที่ใช้สร้างสไตล์ชีตทั่วไป โดยส่วนประกอบสำคัญของ XSL มี 2 ส่วนคือ ภาษาปริวรรต XSL (XSL Transformation Language) และภาษาสำหรับการจัดรูปแบบอ็อบเจ็กต์ (Formatting Object Specification) ทั้ง 2 ส่วนนำมาใช้ในการจัดเตรียมรูปแบบการแสดงผลให้กับเอกสารและนำไปใช้เป็นเนมสเปซของ XML

ภาษาปริวรรต XSL (เนมสเปซ *xsl*) ใช้บรรยายกระบวนการให้โปรแกรมประมวลผลแปรรูปเอกสาร XML จากโครงสร้างหนึ่งไปยังอีกโครงสร้างหนึ่ง นั่นคือการแปรรูปทรีเอกสารหนึ่งไปเป็นอีกทรีเอกสารหนึ่ง ส่วนใหญ่ที่นิยมจะเป็นการแปรรูปจากโครงสร้างภาษา (Semantic Structure) ไปเป็นโครงสร้างการแสดงผล (Display Structure) เช่นการแปรรูปจากเอกสาร XML ไปเป็นเอกสาร HTML เป็นต้น

ภาษาสำหรับการจัดรูปแบบ เนมสเปซ *fo* ใช้สำหรับสร้างภาษาสำหรับการจัดรูปแบบตัวใหม่ โดยมีการพัฒนาเป็น Vocabulary ของ XML ดังนั้นเอ็นจิน สำหรับการแสดงผลจึงสามารถประมวลผลข้อมูลการจัดการรูปแบบที่มีอยู่ภายในเนมสเปซ *fo* ได้โดยตรง (ข้อมูลไม่เหมือนกับข้อมูลในเนมสเปซ *xsl*) นั่นก็คือ โปรแกรมประมวลผลสามารถแปรรูปข้อมูลการจัดรูปแบบไปเป็นโครงสร้างการจัดรูปแบบลักษณะอื่น ๆ เช่น HTML โดยต่างจากวิธีการของเนมสเปซ *xsl* ตรงที่วิธีการนี้จะเกี่ยวข้องเฉพาะภาษาสำหรับการจัดรูปแบบซึ่งเป็นการอนุญาตให้ Vocabulary แต่ละตัวที่สร้างขึ้นมาใช้ได้กับแอพพลิเคชั่นเฉพาะงานได้ เช่น มัลติมีเดีย เป็นต้น

2.2 การทดสอบแบบถดถอย (Regression Testing)

การทดสอบแบบถดถอยเป็นการทดสอบซ้ำ ๆ กันในกรณีทดสอบเดิม ๆ กันหลายครั้ง ซึ่งการทดสอบดังกล่าวสามารถเกิดขึ้นได้เมื่อมีการปรับปรุงซอฟต์แวร์ให้ดีขึ้นและการปรับปรุงนั้นมีความสัมพันธ์กับกรณีทดสอบที่เคยทำไปแล้ว ดังนั้นจึงต้องทำการทดสอบอีกครั้งเพื่อให้มั่นใจว่าซอฟต์แวร์ที่ผ่านการปรับปรุงดังกล่าวได้มีการแก้ข้อผิดพลาดแล้ว (Poston. 1994 b: 124-129)

ในการทดสอบแบบถดถอยนี้สำหรับผู้เชี่ยวชาญหรือผู้ทดสอบซอฟต์แวร์แล้วจัดเป็นการทดสอบที่เสียเวลาในการเตรียมข้อมูลเข้าที่ใช้ในการทดสอบสำหรับทุกครั้งที่ทำการทดสอบ ซึ่งเป็นสิ่งที่ไม่จำเป็นและไม่สะดวกเป็นอย่างยิ่ง ดังนั้นถ้ามีระบบหรือเครื่องมือที่สามารถเตรียมข้อมูลการทดสอบได้เองตามกรณีทดสอบโดยที่ไม่จำเป็นต้องพึ่งผู้เชี่ยวชาญหรือผู้ทดสอบ หรืออาจทำการเตรียมข้อมูลสำหรับการทดสอบครั้งแรกเพียงครั้งเดียวได้ ก็จะสามารถลดภาระของผู้ทดสอบได้

2.3. การทดสอบแบบอัตโนมัติ (Automated Test Execution)

ในยุคแรกของการพัฒนาซอฟต์แวร์การทำการทดสอบคือ การทดสอบแบบ manual หรือการที่ผู้ทดสอบเป็นคนทำการป้อนข้อมูล 2-3 ค่าให้กับคอมพิวเตอร์แล้วดูการตอบสนองของซอฟต์แวร์ที่จะส่งกลับมา ซึ่งมีการทำงาน 3 ขั้นตอน (Poston. 94a.) คือ

1. Setup เป็นการตั้งสถานะให้อยู่ในสถานะพร้อมที่จะทำการทดสอบ โดยมีการให้ค่าข้อมูลเข้า (input) ที่จำเป็นในการทดสอบนั้น ๆ แก่ซอฟต์แวร์
2. Execute เป็นการเรียกใช้การทำงานที่ต้องการ เพื่อให้ได้ซึ่งผลลัพธ์ ซึ่งจะนำไปพิจารณาต่อไปว่าตรงตามที่ต้องการหรือไม่ภายหลัง
3. Cleanup เป็นการยืนยันผลที่ได้ และเปลี่ยนสถานะของซอฟต์แวร์ จากสถานะที่ทำการทดสอบเป็นสถานะปกติ นั่นคือ หากมีการเปลี่ยนแปลงระบบใดๆ จากขั้นตอน Setup ก็ต้องทำการเปลี่ยนกลับให้อยู่ในสถานะปกติก่อนการทดสอบ

แต่ในปัจจุบันก่อนที่จะทำการทดสอบจะมีการสร้าง test case ซึ่งเป็นชุดข้อมูลเข้าที่ใช้ในการทดสอบ (input) ขึ้นมาก่อน สรุปคือกิจกรรมหลักในการทำการทดสอบด้วยวิธีนี้มี 3 ประการด้วยกัน คือ การสร้าง test case การประมวลผล test case และ การประเมินการทดสอบหลังการประมวลผล

การทดสอบแบบ manual ก็มีความยุ่งยากพอสมควรสำหรับผู้ทำการทดสอบที่ต้องทดสอบซอฟต์แวร์หลาย ๆ ครั้งหรือเมื่อมีการปรับปรุงซอฟต์แวร์ให้ดีขึ้นก็จะต้องทำการทดสอบอีกครั้งเพื่อให้มั่นใจว่าซอฟแวร์ดังกล่าวได้แก้ปัญหาที่พบในครั้งก่อนแล้ว ซึ่งเรียกว่าการทำ Regression Testing การทำการทดสอบที่ซ้ำกันในกรณีเดิม ๆ ก็เป็นเรื่องที่น่าเบื่อหน่าย และเสียเวลาหากทำด้วยวิธี manual

แนวทางหนึ่งในการแก้ปัญหาก็ได้มีการพัฒนาการทดสอบแบบอัตโนมัติ ซึ่งมีการใช้เครื่องมือช่วยในการทดสอบที่เรียกว่า Test Execution Tool ซึ่งจะให้รายละเอียดในภายหลัง หลักการทำงานโดยทั่วไปของการทดสอบแบบอัตโนมัติจะคล้ายกับเครื่องเล่นวีดีโอเทป นั่นคือมีการบันทึก และการนำสิ่งที่บันทึกไว้มาเล่นซ้ำ ได้

การบันทึกของการทดสอบแบบอัตโนมัติ คือการบันทึกค่าต่าง ๆ ในการทำการทดสอบแบบ manual ทั้ง test case หรือค่าสถานะที่มีการตั้งใหม่เพื่อใช้ในการทดสอบซึ่งผู้ทำการทดสอบต้องทำในขั้นตอน Setup และหลังจากเริ่มการบันทึกค่าดังกล่าวแล้วผู้ทดสอบก็จะทำขั้นตอน Execute และ Cleanup ไปจนจบ แล้วจึงเลิกการบันทึก ซึ่งข้อมูลต่าง ๆ ที่ถูกบันทึกไว้หรือ Test Script ก็จะถูกเก็บไว้เพื่อนำกลับมาใช้ใหม่ หรือที่เรียกว่าการเล่นซ้ำ ในการทดสอบครั้งต่อ ๆไป กล่าวคือ Test Script จะถูกนำมาใช้ในการทำ Regression Testing นั่นเอง

การทำการทดสอบแบบอัตโนมัติในช่วงการบันทึกค่าผู้ทำการทดสอบต้องทำงานเหมือนกันการทดสอบแบบ manual ซึ่งมีใช้เวลามากในการทำแต่ในครั้งต่อไปผู้ทดสอบจะใช้เวลาลดลงเพราะไม่ต้องมีการตั้งค่า หรือ ใส่ข้อมูลสำหรับการทดสอบเอง และจากที่ Boris Beizer ได้ประมาณค่าเฉลี่ย

ในการนำ Test Script ไปใช้งานใหม่ไว้ที่ประมาณ 20-40 ครั้งนั้นก็ยิ่งทำให้เห็นได้ว่าวิธีนี้ช่วยให้ผู้ทดสอบทำงานได้เร็วขึ้นกว่าวิธี manual

สำหรับการสร้าง test case ในการทดสอบนั้นจำเป็นต้องมีข้อมูลที่ประกอบด้วยความหมายของ logic ที่จะทำให้รู้ว่าการทำงานต่าง ๆ ถูกจำกัดไว้อย่างไรบ้าง ,ความหมายของเหตุการณ์หรือสถานะ เหตุการณ์ที่ระบุได้ว่าการทำงาน หรือ การเปลี่ยนสถานะควรเกิดขึ้น และสถานะนั้นควรเริ่มในการทำงานใด และค่าที่เป็นไปได้ของข้อมูลที่ใช้ในกระบวนการทดสอบ

2.4. กรณีทดสอบ (Test Case) (Poston . 1994 b: 48-58)

กรณีในการทดสอบเป็นชุดข้อมูลเข้าที่ใช้ในการทดสอบ (Input) ซึ่งจะนำไปใช้เพื่อทำการประมวลผล และ ประเมินผลการทดสอบ สำหรับการสร้างกรณีการทดสอบ อาจออกแบบได้ด้วย 6 กลุ่มการออกแบบได้แก่

การทดสอบการทำงาน (Functional testing) แต่ละการทำงานหลัก ๆ นั้นจะมีการทดสอบอย่างน้อยหนึ่งครั้ง นั่นคือจะมีอย่างน้อยหนึ่ง กรณีทดสอบ ที่มีชื่อ ค่าข้อมูลเข้าหรืออินพุต และค่าที่คาดหวัง

การวิเคราะห์ค่าขอบเขต (Boundary value Analysis) การหาข้อผิดพลาดในเรื่องของขอบเขตของค่าต่าง ๆ หรือ ตรวจหาในช่วงของข้อมูลที่มากกว่าและน้อยกว่าค่าขอบเขต

การวิเคราะห์ชั้นสมมูล (Equivalence Class(เป็นการวิเคราะห์ตรวจสอบค่าที่เป็นไปได้ที่ถูกแบ่งแยกออกเป็นกลุ่มเล็ก ๆ ที่เรียกว่า Equivalence classes ซึ่งเราจะสร้าง กรณีทดสอบ ที่มีข้อมูลเข้าอย่างน้อยหนึ่งค่าสำหรับแต่ละ sub domain และ equivalence class ที่ระบุ

Cause-effect Graphing การทดสอบเชิงตรรกะหรือเงื่อนไข ซึ่งทุก ๆ เงื่อนไขจะถูกตรวจสอบค่าความจริงที่เป็นจริง หรือ ค่าความจริงที่เป็นเท็จ กล่าวได้ว่าเป็นการทดสอบ Boolean Logic ซึ่ง กรณีทดสอบ จะถูกสร้างสำหรับตรวจสอบทุก ๆ เงื่อนไข ในทุก ๆ การกระทำในค่าที่เป็นจริง และ ค่าที่เป็นเท็จ

การทดสอบเหตุการณ์ (Event-directed Testing) เป็นการสร้าง กรณีทดสอบ ที่ทำทุก ๆ เหตุการณ์อย่างน้อยหนึ่งครั้ง โดยเหตุการณ์หมายถึงสิ่งที่เกิดขึ้นจากภายนอกระบบแล้วมีผลทำให้เกิดการกระทำขึ้นภายในระบบโปรแกรม นั่นคือ ต้องหาเหตุการณ์หรือสิ่งภายนอกระบบที่สามารถก่อให้ระบบเกิดการทำงานหรือการกระทำขึ้น

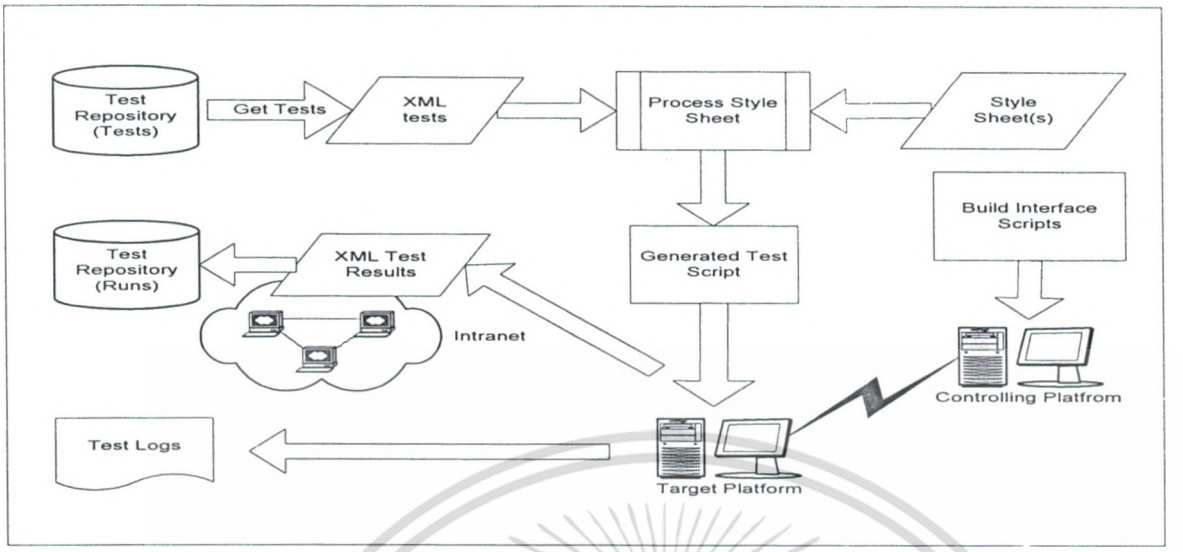
การทดสอบสถานะ (State-directed Testing) คือ กรณีทดสอบ ที่ทำให้มีการเปลี่ยนสถานะ และจะต้องมีการสร้าง กรณีทดสอบ อย่างน้อย 1 ชุดต่อ 1 สถานะในการทดสอบ ซึ่งสถานะคือกลุ่มของข้อมูลหรือคุณลักษณะ หรือความสัมพันธ์ระหว่างค่านั้น ๆ และการเปลี่ยนสถานะคือการเปลี่ยนค่าเหล่านั้น

บทที่ 3
สถาปัตยกรรมยูนิเวอร์แซลเทสต์ไดรเวอร์

3.1 หลักการทำงานของยูนิเวอร์แซลเทสต์ไดรเวอร์ (Cordrey. 2002)

โดยปกติแล้วเราจะมีซอฟต์แวร์ที่ต้องทำการทดสอบทั้งในเรื่องสถาปัตยกรรม)Architecture), ระบบปฏิบัติการ หรือ เวอร์ชันที่ต่าง ๆ กันเป็นจำนวนมาก ซึ่งในการพัฒนาซอฟต์แวร์แต่ละส่วนนั้นควรทำการทดสอบในหลาย ๆ รูปแบบด้วยกัน เช่น การทดสอบแต่หน่วยย่อย ๆ ของซอฟต์แวร์ (Unit Testing) การทดสอบโดยรวมแต่ละหน่วยย่อย ๆ เข้าด้วยกัน (Integration Testing) ,การทดสอบซ้ำ ๆ กันในเรื่องเดิมหลังจากที่มีการปรับปรุงซอฟต์แวร์เรื่องนั้น ๆ ให้ดีขึ้น (Regression Testing) การทดสอบประสิทธิภาพการทำงานของซอฟต์แวร์ (Performance Testing) และการทดสอบเกี่ยวกับขนาดของซอฟต์แวร์ (Scalability Testing) เป็นต้น นั่นคือเราต้องทดสอบให้ครบในทุก ๆ ระบบปฏิบัติการที่ซอฟต์แวร์สนับสนุน หรือทดสอบหลาย ๆ เวอร์ชัน ซึ่งถ้าหากทำการทดสอบโดยใช้คนเพียงอย่างเดียวอาจต้องมีการปรับเกี่ยวกับระยะเวลาที่จะทำการทดสอบ และสภาพแวดล้อมที่ใช้ให้เหมาะสมเนื่องจากช่วงเวลาก่อนที่เราจะส่งซอฟต์แวร์ให้แก่ลูกค้านั้นมีจำกัด หรืออาจมีน้อย ดังนั้นการทดสอบแบบอัตโนมัติจึงเป็นทางเลือกหนึ่งที่จะช่วยให้การทดสอบเสร็จเร็วขึ้นและคนที่มีหน้าที่รับผิดชอบก็ไม่ต้องทำการทดสอบเองทุกขั้นตอนซ้ำๆ กันซึ่งเป็นสิ่งที่น่าเบื่อหน่าย แต่กระนั้นการทดสอบที่ทำนั้นก็ยังต้องคำนึงว่าได้ทดสอบครอบคลุมในทุกเรื่อง หรือเพียงพอที่จะประกันได้ถึงคุณภาพของซอฟต์แวร์ที่จะส่งให้ลูกค้าด้วย หลักการทำงานของยูนิเวอร์แซลเทสต์ไดรเวอร์นี้เป็นการทดสอบบนพื้นฐานของการทำ Script หรือ Batch ไฟล์เพื่อใช้ในการทดสอบ ตัวอย่างเช่นการที่เราเรียกเครื่องมือที่ใช้ในทำการทดสอบด้วย Script ไฟล์และส่งข้อมูลของการทดสอบจากไฟล์นั้นไปในรูปแบบของคำสั่ง (command line) ซึ่งจะถูกนำไปเป็นข้อมูลเข้าสำหรับโปรแกรมที่ต้องการทดสอบ

การทำงานของยูนิเวอร์แซลเทสต์ไดรเวอร์ประกอบไปด้วยส่วนต่าง ๆ ที่ใช้ในการทำการทดสอบแบบอัตโนมัติบนพื้นฐานของ Script ไฟล์บนหลาย ๆ ระบบปฏิบัติการในการทำงานบนระบบเครือข่าย ดังรูปที่ 3.1 ซึ่งแสดงหลักการทำงานของยูนิเวอร์แซลเทสต์ไดรเวอร์



รูปที่ 3.1 แผนผังการทำงานของยูนิเวอร์แซลเทสไดรเวอร์

3.2. องค์ประกอบของการทำงาน

องค์ประกอบของการทำงานของยูนิเวอร์แซลเทสไดรเวอร์แบ่งได้ 6 องค์ประกอบหลักดังนี้

- คลังข้อมูลในการทดสอบ (Test Repository)
- เอกสารในการทดสอบ (Test Document)
- การเชื่อมต่อ เอกสารของการทดสอบแบบเอกซ์เอ็มแอล กับคลังข้อมูลในการทดสอบ (XML Interface to Repository)
- Style Sheets ในการแปลงข้อมูลของการทดสอบ (Test Transformation Style Sheets)
- ส่วนประมวลผลรูปแบบ (Style Processor)
- ตัวช่วยในการสร้าง Script (Script Building Utilities)

3.2.1 คลังข้อมูลในการทดสอบ (Test Repository)

เป็นส่วนที่อำนวยความสะดวกในการจัดดูแลและจัดเก็บการทดสอบต่าง ๆ ซึ่งโดยหลักการแล้วจะไม่เก็บในรูปแบบคำบรรยายรายละเอียดเป็นขั้นเป็นตอนแต่จะเก็บเพียงค่าตัวแปรที่จำเป็นในการแทนค่าใน Script ไฟล์ที่ใช้ในการทดสอบเท่านั้น ซึ่งคลังข้อมูลในการทดสอบมีข้อดีคือเป็นการเก็บรวบรวมข้อมูลในการทดสอบไว้ที่เดียวทำให้สามารถนำข้อมูลของการทดสอบมาใช้ใหม่อีกครั้งได้ (Test reuse) อีกทั้งยังสามารถเข้าข้อมูลถึงได้สะดวก ซึ่งเปรียบเสมือนว่าข้อมูลของการทดสอบนั้นเป็นทรัพยากรที่มีค่าที่จะยังคงอยู่แม้ว่าบุคลากรที่ทำงานด้านนี้จะมีการสลับเปลี่ยนไปก็

ตาม นอกจากนี้การเลือกใช้คลังข้อมูลในการทดสอบนั้นอาจเป็นสร้างขึ้นใหม่โดยใช้แพกเกจจากที่อื่นหรือใช้คลังข้อมูลที่สร้างไว้แล้วก็ได้

โดยสรุปแล้วคลังข้อมูลในการทดสอบจะถูกใช้เป็นที่เก็บข้อมูลของการทดสอบ ซึ่งอาจประกอบไปด้วยตัวทดสอบ (Tests) หรือของกลุ่มทดสอบ (Test Groups) และรายชื่อเครื่องที่ใช้ในระบบ (Machine Lists) เป็นต้น

- ***ตัวทดสอบ (Tests)*** ตัวทดสอบประกอบไปด้วยความสัมพันธ์ระหว่าง Keyword กับค่าของมัน ซึ่งมีความสัมพันธ์กับทุก ๆ ค่าข้อมูลที่เป็นตัวแปรใน Script ของตัวทดสอบ กล่าวคือเมื่อทำการระบุค่าตัวแปรต่าง ๆ ใน Script ได้แล้วก็จะกำหนด Keyword สำหรับแต่ละตัวแปรนั้น โดยเป็นไปในรูปแบบของ "พารามิเตอร์" ของตัวทดสอบซึ่งจะสามารถกำหนดได้จากการทำการวิเคราะห์ Script ที่จะใช้งานและจากการกำหนดค่าต่าง ๆ ที่จะมีการเปลี่ยนค่าจากตัวทดสอบหนึ่ง ๆ ไปยังตัวทดสอบตัวต่อไป ตัวอย่างเช่น

  test_program = test_move

  (หมายถึงโปรแกรมที่จะทดสอบชื่อ test_move)

  นอกจากนี้เพื่อความสะดวกและประโยชน์ในการใช้งานเราอาจเพิ่ม Attributc ของตัวทดสอบคือ "Subtype" ที่ใช้จัดกลุ่มการทดสอบโดยใช้ลักษณะการทำงานหรือตามพฤติกรรมในการทดสอบเช่นแบ่งเป็น Unit Tests, Integration Tests และSmoke Tests ซึ่งในแต่ละ Subtype ก็จะมี Script ที่ต่างกันออกไป
- ***กลุ่มตัวทดสอบ (Test Groups)*** ในคลังข้อมูลยังประกอบไปด้วยกลุ่มต่าง ๆ ของการทดสอบซึ่งทั้งหมดในกลุ่มจะถูกเรียกทำงานพร้อมกัน ซึ่งอาจจัดกลุ่มโดยใช้ลักษณะการทำงานหรือตามพฤติกรรมในการทดสอบเช่นแบ่งเป็น Unit Tests, Regression Tests, Integration Tests และ Smoke Tests หรืออาจจัดกลุ่มโดยแบ่งตามระบบปฏิบัติการ หรือจัดตามคุณสมบัติอื่น ๆ ที่สะดวกในการใช้งานก็ได้
- ***รายชื่อเครื่องที่ใช้ในระบบ (Machine Lists)*** เป็นรายชื่อเครื่องในการทดสอบด้วยตัวกลุ่มทดสอบหนึ่ง ๆ ซึ่งแต่ละเครื่องอาจใช้ชื่อ Domain Name Service (DNS) สำหรับการเข้าถึงเครื่องนั้น ๆ นอกจากนี้ชื่อที่ถูกตั้งขึ้นมาสำหรับสภาพแวดล้อมในการทดสอบจะสามารถบอกได้ถึงประเภทของระบบปฏิบัติการ เวอร์ชัน หรือคอมไพเลอร์ที่ใช้ด้วย

3.2.2 เอกสารในการทดสอบ (Test Document)

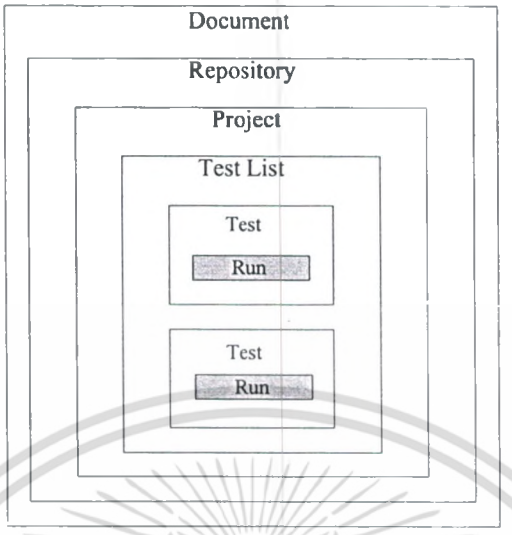
เอกสารในการทดสอบรูปแบบ Extensible Markup Language (XML) จะเป็นตัวช่วยให้ข้อมูลของการทดสอบสามารถแปลงเป็นรูปแบบของไฟล์ Script หรือรูปแบบที่เหมาะสำหรับจัดเก็บและเรียกใช้จากคลังข้อมูลได้ง่าย ที่สำคัญคือการใช้เอกสารแบบ เอกสารของการทดสอบแบบเอกซ์เอ็มแอล นี้ยังช่วยให้ทำการกระจายข้อมูลไปทางเว็บง่ายขึ้น ส่งผลให้ผู้ใช้สามารถเข้าถึงได้ด้วยบราวเซอร์ในการเรียกดูข้อมูลหรือผลของการทดสอบ จึงอำนวยความสะดวกให้สามารถทำงานได้เป็นวงกว้างมากกว่าเดิมที่เป็นการเข้าถึงคลังข้อมูลโดยตรง

ในเอกสาร เอกสารของการทดสอบแบบเอกซ์เอ็มแอล นี้จะประกอบไปด้วยข้อมูลทั้งหมดที่เกี่ยวกับโครงสร้างที่ได้อธิบายไว้ในส่วนของคลังข้อมูลก่อนหน้านี้แล้ว นอกจากนั้นเพื่อให้เอกสารมีรูปแบบที่เป็นมาตรฐานในการสร้างก็จะมีการกำหนดข้อตกลงในการเขียนข้อมูลในเอกสารนี้ด้วยการให้นิยามของข้อมูลแต่ละชนิดในเอกสารที่เรียกว่า Data Type Definition: DTD หรืออาจใช้ XML schemas แต่สำหรับ DTD จะมีการใช้งานที่แพร่หลายมากกว่าในปัจจุบันแม้ว่า XML schemas จะมีความยืดหยุ่นมากกว่าก็ตาม

ส่วนประกอบพื้นฐาน (Basic Elements) ในเอกสารของการทดสอบประกอบไปด้วย

- เอกสาร (Documents)
- คลังข้อมูล (Repository) ซึ่งอาจมีหลายที่ขึ้นอยู่กับการจัดรูปแบบองค์กร
- โครงการ (Project) ซึ่งเป็นไปได้ที่จะแตกต่างกันไปตามแผนก หรือกลุ่มการทำงาน
- รายการที่จะทดสอบ (Test List)
- ตัวทดสอบ (Tests)
- ตัวทดสอบที่ทำไปแล้ว (Runs)
- ลำดับขั้นการทดสอบ (Steps) ซึ่งอาจอยู่ในตัวทดสอบ หรือ ตัวทดสอบที่ทำไปแล้ว

นอกจากที่กล่าวแล้วอาจมีส่วนประกอบเสริมด้วยเพื่ออธิบายความหมายต่าง ๆ ได้แก่ ค่าคาดหวัง (Expected Result), ค่าจริงที่ได้ (Actual Result) และ ค่าสถานะ (Status) ในที่นี้สามารถดูตัวอย่าง DTD ของเอกสารการทดสอบในภาคผนวก ก. ซึ่งมีจะมีโครงสร้างเป็นไปตามรูปที่ 3.2



รูปที่ 3.2 ตัวอย่างโครงสร้าง DTD

สำหรับการใช้งานจริงอาจเป็นที่โต้แย้งกันได้ระหว่างการใช้ XML Elements หรือ XML Attributes ในเอกสารของการทดสอบแบบเอกซ์เอ็มแอล ซึ่งที่จริงแล้วขึ้นอยู่กับความเหมาะสมและความถนัด แต่ในที่นี้มีความเห็นว่าจะใช้ XML Elements มากกว่าเนื่องจากผู้ออกแบบเชื่อว่าการเขียนโปรแกรมที่เกี่ยวกับเอกสารในยุคนี้สามารถเข้าถึง Element ได้มากกว่า ตัวอย่างที่เห็นได้ง่ายที่สุดก็คือ STATUS element ของ RUN เนื่องจากเราจะยังไม่ทราบสถานะของการทดสอบ RUN จนกว่าการทดสอบจะสิ้นสุดลง ซึ่งการสร้าง XML tags ในขณะที่การประมวลผลไปเราจะสามารถเพิ่ม STATUS element ตามหลัง STEP tags ได้เพื่อเป็นค่าสถานะของ RUN แต่ถ้าเขียนแบบใช้ Attribute ก็จะต้องเตรียมส่วนข้อมูลสำรอง (buffer) ของ RUN tags และ STEP tags ไว้จนกระทั่งการทดสอบสิ้นสุดลงเราสามารถจึงจะใส่ค่า STATUS attribute ใน RUN tag นั้น ๆ ได้ ดังนั้นจะเห็นได้ว่าข้อมูลเดียวกันอาจทำได้หลายแบบ ดังตัวอย่างที่ได้ยกมา

สังเกตได้ว่าเราควรคำนึงถึงการทำ Script ที่เข้าใจง่าย และควรเขียน DTD เพื่อความสะดวกในการตรวจสอบเอกสารของการทดสอบแบบเอกซ์เอ็มแอล ทั้งนี้ทั้งนั้นก็ต้องคำนึงถึงความเป็นไปได้ในการนำไปใช้จริงด้วย

3.2.3 การเชื่อมต่อเอกซ์เอ็มแอล กับคลังข้อมูลในการทดสอบ (XML Interface to Repository)

ส่วนการทำงานหนึ่งที่สำคัญก็คือการเชื่อมต่อที่เป็นกลไกในการคัดลอกข้อมูล (Extracting) จากคลังข้อมูลให้เป็นรูปแบบของ เอกสารของการทดสอบแบบเอกซ์เอ็มแอล และทำกลไกในการแทรกข้อมูลจากผลของการทดสอบที่อยู่ในเอกสารของการทดสอบแบบเอกซ์เอ็มแอล กลับไปยังคลังข้อมูล ซึ่งการเชื่อมต่อนี้มีประโยชน์มากไม่ใช่เพียงในหลักการทำงานของยูนิเวอร์แซลเทสต์ไดรเวอร์เท่านั้นแต่ยังสามารถนำไปใช้ในกรณีที่เป็นการทำทดสอบซอฟต์แวร์แบบธรรมดาที่ไม่เป็นแบบอัตโนมัติ ในการเก็บข้อมูลของการทดสอบเข้าคลังข้อมูลอีกด้วย

คลังข้อมูลที่มีขายอยู่ในตลาดมักจะมีส่วนเชื่อมต่อกับการเขียนโปรแกรม (Application Programming Interfaces: APIs) ซึ่งสามารถนำมาใช้สร้างการเชื่อมต่อที่ทำการอ่านเขียนข้อมูล และดูแลรักษาเอกสารเอกซ์เอ็มแอล ซึ่งในปัจจุบันก็มีพวกฐานข้อมูลเอกซ์เอ็มแอล เช่น ฐานข้อมูลของ Oracle บางเวอร์ชันที่ไม่จำเป็นต้องสร้างการเชื่อมต่อดังกล่าว หรืออาจใช้พวก SQL procedure เป็นอีกทางเลือกหนึ่งในการทำก็ได้

ในที่นี้ผู้ออกแบบได้ใช้ SAX-based parser ซึ่งเป็น API อย่างง่ายสำหรับเอกซ์เอ็มแอล และสามารถหาได้ทั่วไปจาก World Wide Web ซึ่งเราจะนำมารวมเข้ากับส่วนการทำงานที่ต้องเชื่อมต่อ API กับคลังข้อมูล โดยที่ผู้ออกแบบได้แนะนำให้จะอยู่ในประเภท SAX event-oriented parser มากกว่าแบบอื่นเพราะว่าไฟล์เอกซ์เอ็มแอล ของผลการทดสอบนั้นอาจมีขนาดใหญ่และรูปแบบของ event based ก็ทำให้ไม่จำเป็นต้องสร้าง object model ทั้งหมดไว้ในหน่วยความจำซึ่งใช้ในการ Parse แบบ DOM (Document Object Model)

สรุปคือสำหรับการเรียกใช้ข้อมูลการทดสอบขึ้นมา (Get Tests) ส่วนการเชื่อมต่อเอกซ์เอ็มแอล จะถูกใช้เพื่อทำการคัดลอกข้อมูลรายการการทดสอบ (Test Lists) และรายการเครื่องที่จะทดสอบ (Machine Lists) จากคลังข้อมูล โดยจะมีเอกสารของการทดสอบแบบเอกซ์เอ็มแอลสำหรับนำเข้า (input) ไปยังตัวเชื่อมต่อกับคลังข้อมูลเพื่อให้ได้รายการการทดสอบที่เรียกว่า “Get Test XML” ซึ่งอาจจะมีการแทรกคำสั่งที่ระบุได้ว่าข้อมูลอะไรบ้างที่จะต้องมีในเอกเอกสารของการทดสอบแบบเอกซ์เอ็มแอล ของการทดสอบที่จะนำไปสร้างเป็นเอกสาร Script สำหรับทดสอบต่อไป

3.2.4 Style Sheets ในการแปลงข้อมูลของการทดสอบ (Test Transformation Style Sheets)

การที่จะแปลงข้อมูลพารามิเตอร์ของการทดสอบในเอกสารของการทดสอบแบบเอกซ์เอ็มแอล เป็น Shell Script จริง ๆ อาจใช้การเขียน Extensible Style Sheet Language (XSL) ช่วย ซึ่ง XSL Style Sheet จะระบุถึง XML schemas ที่เป็นรูปแบบการแสดงข้อมูลในเอกสารเอกซ์เอ็มแอล โดยจะมีการอธิบายว่าจะแปลงกลุ่มข้อมูลต่าง ๆ ของเอกซ์เอ็มแอล อย่างไรให้ออกมาในรูปแบบตามที่ต้องการได้

ตัวอย่างที่เห็นได้ชัดคือการนำเสนอข้อมูลด้วย Elements ต่าง ๆ ของ HTML (Hyper-Text Markup Language) ที่มีความสัมพันธ์เกี่ยวเนื่องกับ Elements ของ เอกสารเอกซ์เอ็มแอล กล่าวคือจะสร้าง Style Sheet หนึ่งขึ้นมาที่สามารถอ้างอิงคู่กับเอกสารการทดสอบเพื่อให้แสดงผลบนบราวเซอร์ง่ายขึ้น ซึ่งในการใช้งานอาจทำโดยเรียกไฟล์ Style Sheet รวมเข้ากับไฟล์เอกซ์เอ็มแอลโดยตรงหรืออาจใช้ JavaScript หรือ ASP (Active Server Pages) ช่วยในการโหลดไฟล์นั้น Style Sheet ให้มาทำงานร่วมกับเอกซ์เอ็มแอล ก็ได้เพื่อให้ได้ไฟล์ในรูปแบบ HTML

แต่ที่เป็นประโยชน์ที่สุดสำหรับการทำงานของยูนิเวอร์แซลเทสต์ไดรเวอร์ก็คือนำไปใช้ในการสร้างข้อความในรูปของ Shell Script ซึ่งในการประมวลผล Element ในเอกสารของการทดสอบแบบเอกซ์เอ็มแอล ของการทดสอบ Style Sheet ตัวหลักมีการบอกว่าจะประมวลผลอย่างไร

นอกจากนี้อาจทำการแยก Style Sheet ออกมาสำหรับแต่ละการทดสอบประเภทย่อย (Test Subtype) และทำการรวม Style Sheet เหล่านั้นไว้กับ Style Sheet ตัวหลัก กล่าวคือ Style Sheet ย่อย ๆ เหล่านี้จะเป็นตัวบอกว่าจะแปลง Test Elements และ Step Elements ในเอกสารของการทดสอบแบบเอกซ์เอ็มแอล ได้อย่างไร ซึ่ง Style Sheet ยังนำใช้สำหรับการแปลงข้อมูลเกี่ยวกับรายการเครื่อง (Machine List) ที่ใช้ในการทดสอบอีกด้วย

3.2.5 ส่วนประมวลผลรูปแบบ (Style Processor)

ในการแปลงภาษาในรูปแบบ Tree ของเอกซ์เอ็มแอลให้เป็นข้อความรูปแบบอื่นเช่น HTML หรือเอกซ์เอ็มแอล ในรูปแบบอื่นนั้นทาง W3C ได้แนะนำตัวประมวลผลที่มีประสิทธิภาพ ได้แก่ XSLT (the Extensible Style Sheet Language Transformations) และ XPath (the XML Path Language) ซึ่งตัวประมวลผลตามที่ทาง W3C แนะนำสามารถทำงานได้ดีแต่บางตัวประมวลผลอาจทำการคอมไพล์ได้ไม่สมบูรณ์ในตัวเอง อย่างเช่นในตัวประมวลผลแบบ XSLT ซึ่งมักต้องใช้

ร่วมกับภาษาจาวาดังนั้นจึงจำเป็นต้องมี JRE (Java Runtime Engine) หรือตัวที่ทำการประมวลผลเกี่ยวกับภาษาจาวาด้วย

สำหรับเงื่อนไขในการเลือกตัวประมวลเราอาจให้ความสำคัญกับระบบปฏิบัติการที่จะนำตัวประมวลผลนั้นไปทำงานไม่ว่าจะเป็น UNIX หรือ Windows แต่จริงแล้วมันก็ไม่ได้ทำให้เกิดการทำงานที่ต่างกันแต่อย่างใด เพราะกระบวนการควบคุมในการสร้าง Script ก็ยังคงสร้าง Script ที่เหมาะสมได้จากเครื่องแม่ (host) ซึ่งจะสร้าง Script ด้วยการแตกข้อมูลจากคลังข้อมูลของการทดสอบโดยตรง หรือไม่ก็อาจใช้การแปลงข้อมูลด้วย XSLT ช่วย

การทำงานในส่วนนี้ไม่ซับซ้อนมากนัก กล่าวคือตัวประมวลผลจะอ่านไฟล์เอกซ์เอ็มแอล ที่เป็นข้อมูลนำเข้า (input) ซึ่งประกอบไปด้วยตัวทดสอบ ต่าง ๆ และอ่านไฟล์ Style Sheet ซึ่งเป็นไปได้ว่าจะมี Style Sheet ของการทดสอบประเภทย่อย ๆ รวมอยู่ด้วย จากนั้นก็จะประมวลผลสร้างไฟล์ที่มี Shell Script แทรกอยู่ด้วยขึ้นมา หรือที่เรียกว่า Script ของการทดสอบ

นอกจากนั้นส่วนนี้ยังจัดการเกี่ยวกับรายการเครื่อง (Machine List) ที่ใช้ในการทดสอบซึ่งรายการจะถูกระบุไว้ในไฟล์เอกซ์เอ็มแอล โดยจะสร้างเป็นไฟล์รายการในรูปแบบ ASCII สำหรับใช้ในการทดสอบในตอนเริ่มทดสอบด้วย Script บนเครื่องที่กำหนด

3.2.6 ตัวช่วยในการสร้าง Script (Script Building Utilities)

ในการเขียน Style Sheets เพื่อใช้เป็นรูปแบบในการสร้าง Script สำหรับการทดสอบถ้ามี Utility Scripts ด้วยจะเป็นประโยชน์ในการจัดการเกี่ยวกับรายละเอียดของ XML tags, จัดการเกี่ยวกับ Test Mode และจัดการเกี่ยวกับการสร้าง Script ซึ่งตัวอย่างรายการของ Utilities มีดังตารางที่ 3.1

Script	Description
add_command	Add a simple command to the script
add_step	Add a testing step to the script
create_test_script	Create the test script
daily_testing.ksh	Initiate the testing following detection of the Build
detect_new_build.ksh	Monitor the inbox for a new Build
killzombies	Terminates all user's processes whose parent PID=1
logevent	Write a date/time stamped record
print_end_tag_driver	Print Document XML end tag
print_end_tag_project	Print Project XML end tag
print_end_tag_repository	Print Repository XML end tag
print_end_tag_run	Print Run XML end tag
print_end_tag_step	Print Step XML end tag
print_end_tag_test	Print Test XML end tag
print_end_tag_testlist	Print Testlist XML end tag
print_start_tag_driver	Print Document XML start tag
print_ start _tag_project	Print Project XML start tag
print_ start _tag_repository	Print Repository XML start tag
print_ start _tag_run	Print Run XML start tag
print_ start _tag_step	Print Step XML start tag
print_ start _tag_test	Print Test XML start tag
print_ start _tag_testlist	Print Testlist XML start tag
run_test_script	Run the generated script
set_test_mode	Set Stop or Continue or Fail Mode
time_out	Kill a given process if it still exists after a time value
waitfor.ksh	Wait for a given file to be created
waitforgz.ksh	Wait for a compressed file to be valid

ตารางที่ 3.1 ตัวอย่างรายการ Utilitiesในการสร้าง Script

ในแต่ละ Tags ของเอกสารของการทดสอบแบบเอกซ์เอ็มแอล จะมีการนำ Utilities เกี่ยวกับการสร้าง Start tag และ End tag มาใช้ตัวอย่างเช่น ใน XSL Style Sheet จะมีการเขียน

```
.print_start_tag_test name folder description
```

ซึ่งจะทำให้ได้โค้ดดังนี้

```
echo "  <TEST"
echo "          name = \"$1\""
echo "          folder = \"$2\""
echo "  >"
echo "  <DESCRIPTION><![CDATA[$3]]></DESCRIPTION>"
```

ในส่วนที่เกี่ยวกับการจัดการ Test mode จะเป็นตัวบอกว่าหากการประมวลผลลำดับขั้นการทดสอบ (Steps) ล้มเหลวควรจะประมวลผล Script ต่อหรือไม่ โดยจะมีการกำหนดสถานะของ Test mode ไว้

นอกจากเกี่ยวกับ Test mode แล้ว Utilities ที่สำคัญต่อการทำงานในการเริ่มต้นสร้าง Script ได้แก่

- add_command: ใช้เพื่อใส่ Shell Command ใน Script ที่สร้างขึ้น เช่นพวก Change Directory command หรือ cd เป็นต้น
- add_step: ใช้เพื่อใส่ command สำหรับการทดสอบใน Script เช่นคำสั่งให้ประมวลผล หรือให้เก็บและแยกกลุ่มข้อผิดพลาด (error output) สำหรับ "Actual Result" tag ในเอกสารการทดสอบ แล้วนำไปเปรียบเทียบกับค่า "Expected Value" แล้วจึงนำผลการเปรียบเทียบไปใส่ใน Status tag ของ Run เป็นต้น

3.2 สภาพแวดล้อมของการทดสอบ

เครื่องคอมพิวเตอร์ที่จะทดสอบจะมีการเชื่อมต่อผ่านระบบเครือข่ายทั้งหมด ดังนั้นจึงมีเครื่องที่คอยควบคุม (Controlling machine) ที่ทำหน้าที่เริ่มทดสอบ แม้ว่าตามหลักการทำงานยูนิเวอร์แซลเทสต์ไดรเวอร์จะมีการติดตั้งระบบในทุกเครื่องก็ตามแต่ในการทำงานจริงยังคงต้องสร้างการติดต่อสื่อสารระหว่างเครื่องให้ค่อนข้างยืดหยุ่นด้วย เนื่องจากผู้ใช้สามารถเข้าถึงทุก ๆ เครื่องได้ซึ่งได้มีการนำ File Server มาใช้ในการจัดเก็บ Home Directory ของผู้ใช้ เพื่อให้สามารถเข้าถึงได้จากทุกเครื่องในแบบ NFS automount ดังนั้นเครื่องที่ใช้ระบบปฏิบัติการ Windows อยู่จะต้องมีการเข้าถึง Home Directory ด้วยการทำ drive mapping ไปยังเครื่องที่ใช้ Samba Software ในการเข้าถึงระบบ File system ของ UNIX โดยที่ Script ทั้งหมดที่ใช้ในการประมวลผลจะเป็น Script เดียวกันที่เขียนด้วย Korn shell ทั้งสองระบบ และเพื่อที่จะทำให้สามารถเข้าถึงแต่ละเครื่องที่ใช้ทดสอบจะต้องมีการกำหนดค่าต่าง ๆ ในแบบ remote shell (remsh) command จากเครื่องที่ควบคุมการทดสอบด้วย

ในทางตรงกันข้ามการติดต่อจากเครื่อง UNIX ไปยังเครื่องที่ใช้ระบบปฏิบัติการ Windows สามารถทำโดยใช้ utility ที่จัดการการเข้าถึงแบบ remote logon จากเครื่อง UNIX ไปยังเครื่อง Windows ซึ่ง utility นี้จะกำหนดค่าในการเรียกใช้สภาพแวดล้อม Cygwin bash shell ใน UNIX เอง (เฉพาะ UNIX ที่สนับสนุน Red Hat) ซึ่ง Script แบบนี้จะเขียนด้วย bash shell

3.3 การประยุกต์ใช้การทดสอบให้ทำงานร่วมไปกับการพัฒนาระบบและการรวมระบบ (Interface to Build and Packaging)

เพื่อให้ยูนิเวอร์แซลเทสต์ไดรเวอร์มีประสิทธิภาพมากที่สุดอาจเพิ่มความสามารถในการตรวจสอบว่าโปรแกรมที่จะต้องทดสอบนั้นมีการพัฒนาหรือรวมกลุ่มเสร็จแล้วด้วย ซึ่งจะช่วยให้ทำการทดสอบได้เองช่วงเวลาใดก็ได้เมื่อโปรแกรมพร้อมที่จะทดสอบ

ส่วนมากการนำการทดสอบมารวมกับกระบวนการพัฒนาโปรแกรมมักจะทำบนเครื่อง UNIX เพราะง่ายต่อการใช้ Home Directory หรือ Directory ในการพัฒนาร่วมกัน และมีความสามารถในการจัดลำดับการทำงานที่ดี

ในการทำเช่นนี้จะต้องมีส่วนที่ทำการตรวจสอบ "Inbox" ซึ่งเป็นที่ที่กำหนดไว้เพื่อเก็บไฟล์ที่บอกได้ว่ามีการพัฒนาหรือการรวมกลุ่มที่เสร็จและพร้อมที่จะทำการทดสอบ (Trigger files) โดยไฟล์ดังกล่าวถูกสร้างมากจากฝ่ายพัฒนาโปรแกรม ดังนั้นจะต้องมีการร่วมมือกับฝ่ายพัฒนาโปรแกรมในการเพิ่มเงื่อนไข logic ในกลุ่ม procedures ที่พัฒนาอยู่เพื่อให้มีการสร้างไฟล์ดังกล่าวไว้ในที่ที่กำหนดไว้ และเมื่อมีการตรวจพบ Trigger file ก็จะมีการเรียกใช้ Script ของการทดสอบในเรื่องนั้น ๆ เพื่อแสดงรายละเอียดเกี่ยวกับงานของการทดสอบในครั้งก่อน ๆ เช่นผลการทดสอบหรือ logs ให้กับฝ่ายที่ควบคุมการทดสอบเห็น

จากนั้นกระบวนการทดสอบก็จะเริ่มขึ้นโดยการเอา Script มาประมวลผลเพื่อเอาข้อมูลการทดสอบเช่น ตัวทดสอบ และรายการเครื่องที่ใช้ในการทดสอบ ซึ่งข้อมูลดังกล่าวจะมีการประมวลผลผ่าน XSL Translator เพื่อสร้าง Test Script และ Machine List

นอกจากนี้สำหรับทุก ๆ เครื่องในรายการแล้ว Script จะถูกนำไปใช้เริ่มทำงานบนเครื่องนั้น ๆ ด้วยการทำ Remote Logon โดยจะมีการรอรับโปรแกรมที่พัฒนาเสร็จที่จะต้องทำงานในระบบปฏิบัติการบนเครื่องนั้น ๆ และเมื่อ Test Script มีการประมวลผลบนเครื่องนั้นเสร็จก็จะได้ Logs Files และ XML Files ที่มีรายละเอียดของการทดสอบและผลการทดสอบ ทั้ง Logs และผลการทดสอบจะมีการแสดงผลบนเว็บแบบ online และผลการทดสอบในรูปแบบเอกซ์เอ็มแอล จะถูกนำผ่านกระบวนการแปลงข้อมูลเพื่อเก็บในคลังข้อมูลอีกที

3.4 ข้อดีและความเสี่ยงที่พึงระวัง

3.5.1 ข้อดี

- สามารถพัฒนาเพื่อทดสอบ Command line ตามScenario ที่ต้องการได้
- ส่งเสริมการใช้คลังข้อมูลกลางในการเก็บกรณีทดสอบ, การประมวลผลกรณีทดสอบ และรายงาน
- สามารถสร้างการทดสอบที่ซ้ำ ๆ กันได้โดยไม่ต้องใช้คนทำงานนั้นซ้ำ กันบ่อย ๆ
- เป็นการประมวลผลและทำรายงานแบบอัตโนมัติทั้งหมด
- สามารถทำงานในขณะได้บนหลาย ๆ ระบบปฏิบัติการด้วยการทำงานแบบอัตโนมัติ
- สคริปต์ที่ใช้ทดสอบบำรุงรักษาง่ายและเป็นสคริปต์กลางที่เรียกใช้ได้ทั่วไป
- มีการแสดงผลการทดสอบในรูปแบบเว็บ และบันทึกในคลังข้อมูล
- ฝ่ายทดสอบโปรแกรมสามารถเพิ่มตัวทดสอบได้ง่ายโดยไม่ต้องพึ่งการสนับสนุนจากฝ่ายอื่นมากนัก

3.5.2 ความเสี่ยง

มีความเสี่ยงเกี่ยวกับเรื่องเทคโนโลยีซึ่งขึ้นกับประสบการณ์ขององค์กรการทำงานเกี่ยวกับเอกซ์เอ็มแอล ซึ่งในระยะเริ่มแรกของการพัฒนา XSL จะค่อนข้างซับซ้อนมากกว่าการทำ สคริปต์สำหรับการทดสอบหนึ่ง ๆ เพราะต้องสร้างให้สามารถใช้งานได้ครอบคลุมการทดสอบทุกรูปแบบด้วย

บทที่ 4
การออกแบบแฟ้มเชื่อมประสานสำหรับการทดสอบดีเอ็มแอลฐานข้อมูล

4.1 ความสำคัญของการทดสอบฐานข้อมูล

ฐานข้อมูลมีบทบาทในการทำงานของเกือบทุกองค์กรสมัยใหม่ โดยระบบการจัดการฐานข้อมูลทำให้องค์กรสามารถเข้าถึงข้อมูลจำนวนมากได้อย่างมีประสิทธิภาพ รวมทั้งการควบคุมความถูกต้องให้กับข้อมูล และลดความจำเป็นในการเข้าใจรายละเอียดกลไกในการจัดเก็บ และค้นคืน ปัจจัยสำคัญของระบบการจัดการฐานข้อมูลจะต้องรับประกันอัลกอริธึม และโครงสร้างฐานข้อมูล โดยลักษณะความถูกต้องของระบบฐานข้อมูลมีดังต่อไปนี้ (Chays et al. 2000)

1. โปรแกรมประยุกต์กระทำตามที่กำหนดหรือไม่ ?
2. โครงสร้างฐานข้อมูลที่ถูกต้องสะท้อนถึงข้อมูลจริงที่เป็นแบบจำลองขององค์กรหรือไม่ ?
3. ข้อมูลในฐานข้อมูลถูกต้องหรือไม่ ?
4. ปกป้องความปลอดภัย และความเป็นส่วนตัวอย่างเหมาะสมหรือไม่ ?
5. ระบบการจัดการฐานข้อมูลกระทำการเพิ่ม ลบ และปรับปรุงของข้อมูลทั้งหมดอย่างถูกต้องหรือไม่ ?

เราจะทราบได้อย่างไรว่าระบบฐานข้อมูลของเรามีลักษณะความถูกต้องตามที่ได้กล่าวมาแล้วข้างบน นั้นคือเราจะต้องนำเทคนิคต่างๆ ในการทดสอบซอฟท์แวร์ คือ การพัฒนาอย่างมีระบบในการเลือกเทสต์เคสที่เป็นอินพุตหนึ่งชุดที่ประกอบด้วย อินพุต(ข้อมูล) สิ่งแวดล้อม และ เอาต์พุตที่คาดหวัง ให้มีขนาดเล็กเพียงพอเหมาะสมในการทดสอบให้ครอบคลุมสถานการณ์ต่างๆ ที่มีความเป็นไปได้ซึ่งจะได้ผลลัพธ์เป็นเอาต์พุต

ซึ่งในการศึกษาครั้งนี้เราจะเน้นไปที่การทดสอบระบบการจัดการฐานข้อมูลว่ามีความถูกต้องในการกระทำการเข้าถึงและแก้ไขข้อมูล ซึ่งเป็นกลุ่มคำสั่งในภาษา SQL ที่เรียกว่า ภาษาการจัดการฐานข้อมูล(ดีเอ็มแอล) โดยเราจะนำบางส่วนของขั้นตอนการทดสอบซอฟท์แวร์ นำมาใช้ในการทดสอบดีเอ็มแอลของฐานข้อมูล นั้นคือขั้นตอนการเตรียมเทสต์ดาต้า ซึ่งมีอยู่ด้วยกันสองวิธี คือ เทสต์ไดร์เวอร์ และ เทสต์สตับ แต่เราจะสนใจในการนำไปใช้เทสต์ไดร์เวอร์ ซึ่งเป็นซอฟท์แวร์โมดูลหรือโปรแกรมประยุกต์ที่ใช้ทำให้เกิดชุดทดสอบที่จัดให้ อินพุตทดสอบ(ข้อมูล) ควบคุม และดูแลการดำเนินการ โดยมีการกำหนดรูปแบบแฟ้มเป็น XML เพื่อเป็นมาตรฐานในการแลกเปลี่ยนชุดทดสอบ และออกแบบเทสต์ไดร์เวอร์ให้มีลักษณะการติดต่อกับผู้ใช้ผ่านส่วนประสานกราฟิก (GUI)

4.2 รูปแบบภาษาดีเอ็มแอล

4.2.1 ฐานข้อมูลเชิงสัมพันธ์ และ SQL

ฐานข้อมูลเชิงสัมพันธ์เป็นการใช้แบบจำลองข้อมูลเชิงสัมพันธ์ที่ซึ่งมองข้อมูลเป็นกลุ่มของความสัมพันธ์ ซึ่งความสัมพันธ์คือตาราง โดยในแต่ละตารางประกอบด้วยแถว (Row) ที่แทนข้อมูลเกี่ยวกับหนึ่งรายการ และแต่ละสดมภ์ (Column) แทนคุณลักษณะต่างๆ ของข้อมูล โดยโครงร่างความสัมพันธ์เป็น R(A1, … , An) เป็นชื่อความสัมพันธ์(ชื่อตาราง) ตามด้วยชื่อของคุณลักษณะ (ชื่อสดมภ์) แต่ละสดมภ์จะมีชื่อ Ai และโดเมน(ชนิด) dom(Ai) ซึ่งโครงร่างความสัมพันธ์อธิบายถึงโครงสร้างของข้อมูล ในขณะที่ความสัมพันธ์อธิบายถึงสถานะของข้อมูลในขณะเวลานั้น โดยโครงร่างความสัมพันธ์จะถูกกำหนดในช่วงเวลาออกแบบฐานข้อมูล และมีการเปลี่ยนแปลงน้อยที่สุด อย่างไรก็ตามสถานะความสัมพันธ์ถูกแก้ไขอย่างคงที่ในการส่งผลต่อการเปลี่ยนแปลงในข้อมูลจริงที่เป็นแบบจำลอง แต่ในบางครั้งสถานะความสัมพันธ์อาจจะเปลี่ยนแปลงเมื่อมีการเข้ามาใหม่ในฐานข้อมูล หรือเมื่อมีการลบ หรือแก้ไขโครงร่างฐานข้อมูลเชิงสัมพันธ์เป็นเซตของโครงร่างความสัมพันธ์ด้วยกันกับเซตของข้อบีบบังคับความถูกต้อง และสถานะฐานข้อมูลเชิงสัมพันธ์เป็นเซตของสถานะความสัมพันธ์ของการให้ความสัมพันธ์ เช่นข้อบีบบังคับความถูกต้อง ซึ่งมีชนิดต่างๆของข้อบีบบังคับ ดังนี้

1.กำหนดข้อบีบบังคับโดเมน

2.กำหนดข้อบีบบังคับที่มีเอกลักษณ์

3.คุณลักษณะ A สามารถเป็นค่าพิเศษ(ว่างเปล่า)ที่เรียกว่า NULL โดยการกำหนดข้อบีบบังคับ not-NULL คือให้คุณลักษณะไม่สามารถให้มีค่าเป็น NULL ได้

4.ข้อบีบบังคับความถูกต้องอ้างอิง (Referential integrity)

5.ข้อบีบบังคับความถูกต้องซีแมนติค เช่น เงินเดือนหัวหน้าแผนกจะต้องสูงกว่าสมาชิกอื่นๆในแผนก

SQL (Structured Query Language) เป็นภาษาที่ใช้สำหรับการจัดการ และดึงข้อมูลจากฐานข้อมูล ซึ่งฐานข้อมูลที่จะใช้งานภาษา SQL ได้ต้องเป็นฐานข้อมูลเชิงสัมพันธ์ ซึ่ง SQL ไม่เพียงเป็นแต่เป็นภาษาที่ใช้สำหรับการคิวรีเท่านั้นแต่ยังนำมาใช้ควบคุมการทำงานของ DBMS ได้ด้วย เช่น การใช้คำสั่ง SQL เพื่อสร้างตารางในฐานข้อมูล หรือแม้แต่การอนุญาตให้ผู้ใช้แต่ละคนมีสิทธิในการใช้ฐานข้อมูลเป็นต้น โดยสามารถแบ่งกลุ่มภาษา SQL ได้ 3 กลุ่มดังนี้

1. ภาษาการจัดการฐานข้อมูล (Data Manipulation Language-DML) เป็นกลุ่มคำสั่งในภาษา SQL ที่ใช้สำหรับการเข้าถึงข้อมูลและการแก้ไขข้อมูล เช่น SELECT ใช้เพื่อคิวรีหาข้อมูล, INSERT ใช้เพื่อเพิ่มข้อมูล, DELETE ใช้เพื่อลบ และ UPDATE ใช้เพื่ออัพเดตลงในตารางข้อมูล

2. ภาษาการกำหนดข้อมูล (Data Definition Language-DDL) เป็นกลุ่มคำสั่งในภาษา SQL ที่ใช้กำหนดออบเจ็กต์ฐานข้อมูล เช่น CREATE ใช้เพื่อเพิ่มตารางลงในฐานข้อมูล DROP ใช้เพื่อลบตารางออก และ ALTER ใช้เพื่อเปลี่ยนแปลงโครงสร้างตารางในฐานข้อมูล

3. ภาษาการควบคุมข้อมูล (Data Control Language-DCL) เป็นกลุ่มคำสั่งในภาษา SQL ที่ควบคุมความปลอดภัยข้อมูลเช่น GRANT ใช้เพื่อให้สิทธิในการใช้ฐานข้อมูล และ REVOKE ใช้เพื่อยกเลิกสิทธิในการใช้ฐานข้อมูล

เนื่องจากเราสนใจการทดสอบดีเอ็มแอลของฐานข้อมูล จึงนำเสนอรูปแบบเฉพาะภาษาการจัดการฐานเท่านั้น

SELECT [DISTINCT] <column(s)>
FROM <table(s)>
[WHERE <condition>]
[GROUP BY <column(s)>] [HAVING <condition>]
[ORDER BY <column(s)> [ASC| DECS]]

INSERT INTO <table_name>
VALUES(v_column1, v_column2, …) หรือ
INSERT INTO <table_name> (col_name1, col_name2)
VALUES(v_column1, v_column2, …)

DELETE FROM <table_name>
WHERE <condition>

UPDATE <table_name> SET <column_name1> = <new_value1> , <column_name2> = <new_value2> WHERE <condition>

แต่เนื่องจากระบบเทสต์ไดร์เวอร์กุยจำเป็นต้องมีการสร้าง และลบตารางจึงจำเป็นต้องใช้ภาษาการกำหนดข้อมูล (ดีดีแอล) เข้ามาเกี่ยวข้องด้วยคือ

CREATE TABLE <table name>
<column name1><data type>[CONSTRAINT<constraint name>
<integrity constraint>]
… <column nameN><data type>[CONSTRAINT<constraint name>

<integrity constraint>]

DROP TABLE <table name>

4.2.2 การแลกเปลี่ยนเอกสารการทดสอบระหว่างองค์กรด้วย XML Schema

มาตรฐาน XML(eXtensible Markup Language) เป็นภาษาที่ใช้ในการสร้างเอกสารที่มีการกำหนดความหมายของข้อมูลได้รับการออกแบบให้ใช้งานได้ดีบนเว็บ โดยเฉพาะการแลกเปลี่ยนข้อมูลระหว่างโปรแกรมประยุกต์ แต่ก็สามารถทำงานได้ดีแม้สภาพแวดล้อมการทำงานที่ไม่ใช่เว็บ ซึ่งจุดประสงค์หลักของ XML มีดังนี้

- XML ควรสนับสนุนโปรแกรมประยุกต์ได้หลากหลาย ทั้งในรูปแบบเป็นเครื่องมือสำหรับการสร้าง การแสดงผลข้อมูล และแม้แต่การจัดเก็บข้อมูล เพื่อให้โปรแกรมประยุกต์ต่างๆ นำ XML มาใช้งาน
- ความสามารถเสริมที่มีอยู่ใน XML ต้องมีน้อยที่สุด หรือไม่ควรมีเลย เนื่องจากจะทำให้ทั้งเอกสารและโปรแกรมประมวลผลมีความเข้ากันได้ยาก เช่น โปรแกรมประมวลผลหนึ่งได้รับการออกแบบมาให้สามารถอ่านและประมวลผลเอกสาร XML ได้ด้วยความสามารถเสริมตัวหนึ่ง แต่ว่าเอกสารที่สร้างด้วยความสามารถเสริมอีกตัวหนึ่งอาจไม่สามารถทำงานได้ถูกต้องด้วยโปรแกรมประมวลผลนี้
- การออกแบบ XML จะต้องเป็นรูปแบบที่สั้นและกระทัดรัด

การสร้างเอกสาร XML จะต้องคำนึงถึงการมีลักษณะ "Well Formed" คือเอกสารตัวนั้นต้องทำตามหลักของภาษา XML ทุกประการ เช่น เมื่อมี tag เปิด ต้องมี tag ปิดเสมอ และจะต้องตามลำดับตัวอักษรเล็กใหญ่มีความแตกต่างกันเป็นต้น แต่การ Well Formed อย่างเดียวยังไม่พอเนื่องจากเอกสารที่สร้างขึ้นนี้มีความจำเป็นต้องมีการแลกเปลี่ยนไปยังองค์กรอื่น จึงต้องมีการกำหนดข้อตกลงในหลักภาษาให้สอดคล้องเข้าใจความหมายตรงกัน ซึ่งขึ้นกับว่าจะเลือกใช้ DTD (Data Type Definition) หรือ XML Schemas

DTD (Data Type Definition) สามารถกำหนดได้เอง หรืออาจจะใช้ DTD ที่องค์กรอื่นได้มีการกำหนดไว้แล้วก็ได้ซึ่งจะเป็นการกำหนดว่า element หนึ่งๆ นั้นมีส่วนประกอบอะไรบ้างตัวอย่างในรูปที่ 4.1

```
1 <?xml version = “1.0”>
2 <!DOCTYPE memo [
3 <!ELEMENT memo (to,from,subject?,body)>
4 <!ELEMENT to (#PCDATA)>
5 <!ELEMENT from (#PCDATA)>
6 <!ELEMENT subject (#PCDATA)>
7 <!ELEMENT body (#PCDATA)>
8 ]>
```

รูปที่ 4.1 Data Type Definition

จากรูปที่ 4.1 บรรทัดที่ 2 DOCTYPE เป็นตัวย่อของคำว่า Document Type ในบรรทัดที่ 3 เป็นการนิยามว่า element memo นี้ประกอบด้วย element ย่อย 4 ตัวคือ to, from, subject, body ซึ่ง element subject จะปรากฏในเอกสาร XML 1 ครั้งหรือว่าไม่ปรากฏเลย ส่วนบรรทัดที่ 4-7 เป็นการนิยามว่า element นั้นๆเป็นชนิด PCDATA

ซึ่ง DTD ตามรูปที่1. นั้นสามารถรวมอยู่ในเอกสาร XML หรือแยกจากเอกสาร XML แล้วใช้อ้างถึงก็ได้ แต่การแยกจากเอกสาร XML เป็นวิธีที่สะดวกกว่าเนื่องจากเมื่อมีการแก้ไข DTD ก็จะแก้ไขแค่เพียงที่เดียว

วิธีการในการจัดการรูปแบบเอกสาร XML ไม่ได้มีเพียง DTD แต่มีวิธีการจัดการรูปแบบอีกหนึ่งคือXML Schemas ดังตัวอย่างในรูปที่ 4.2

```
1 <?xml version=”1.0”>
2 <Schema name=”memo” xmlns=”urn:Schemas-microsoft-com:xml-
data”xmlns:dt=”urn:schemas-microsoft-com:datatypes”>
3 <ElementType name=”to” content=”textOnly” />
4 <ElementType name=”from” content=”textOnly” />
5 <ElementType name=”subject”content=”textOnly” />
6 <ElementType name=”body” content=”textOnly” />
7 <ElementType name=”memo” content=”textOnly” />
```

```
8<element type="to"minOccurs="1" maxOccurs="1"/>
9  <element type= "from" minOccurs= "1" maxOccurs= "1" />
10  <element type= "subject"minOccurs= "0"maxOccurs= "1" />
11  <element type= "body" minOccurs= "1" maxOccurs= "1" />
12  </ElementType>
```

รูปที่ 4.2 XML Schema

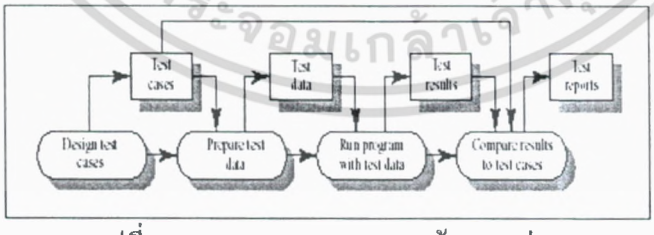
XML Schema ตามรูปที่ 4.2 นั้นมีความหมายเหมือนกับ DTD ในรูปที่ 4.1 แต่ยังไม่ถือว่าเป็นมาตรฐานที่เป็นทางการเหมือนกับ DTD

4.3. การทดสอบซอฟต์แวร์ด้วยแฟ้มเชื่อมประสานด้วย XML

การทดสอบหมายถึงการตรวจสอบซอฟต์แวร์ โดยการทดสอบโปรแกรมจะทำการเลือกอินพุต และทำการตรวจสอบหาข้อผิดพลาดของซอฟต์แวร์ โดยมีวัตถุประสงค์หลักของการทดสอบซอฟต์แวร์ คือการพัฒนาอย่างมีระบบในการเลือกกลุ่มทดสอบให้มีขนาดเล็กเพียงพอเหมาะสมในการทดสอบให้ครอบคลุมสถานการณ์ต่างๆ ที่มีความเป็นไปได้(เทคนิคในการเลือกเทสต์เคสน้อยที่สุดให้เจอข้อผิดพลาดมากที่สุด เช่น Domain Testing และ Category-partition Testing เป็นต้น) ซึ่งจะได้ผลลัพธ์เป็นเอาต์พุต โดยกลุ่มทดสอบคือ อินพุตหนึ่งชุดที่ประกอบด้วยพารามิเตอร์สองตัวคือ (อินพุต และสิ่งแวดล้อม) และเอาต์พุตที่คาดหวังจะได้ สามารถเขียนกลุ่มทดสอบเป็นเซตดังนี้

Test cases = {input, expected output, environment}

ในการทดสอบซอฟท์แวร์มีกระบวนการทดสอบข้อบกพร่องสามารถแสดงได้ดังรูปที่ 4.3



รูปที่ 4.3 กระบวนการทดสอบข้อบกพร่อง

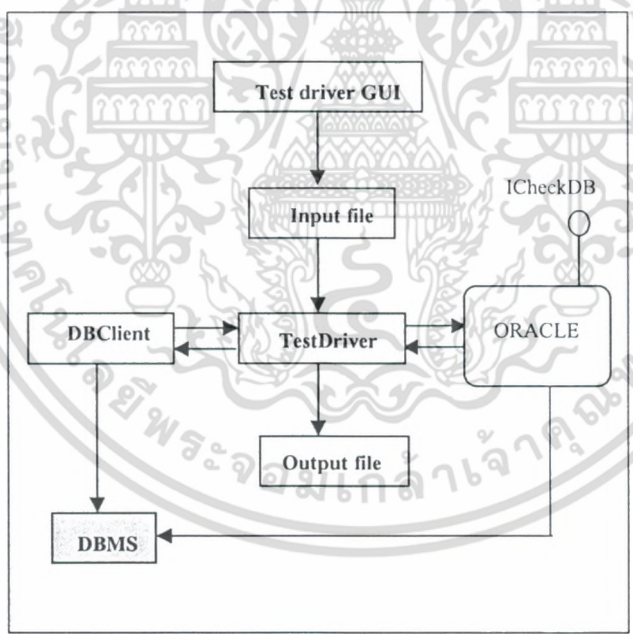
ซึ่งในกระบวนการเตรียมเทสต์คาต้ามีอยู่ด้วยกันสองวิธี คือ เทสต์ไดร์เวอร์ เป็นซอฟท์แวร์โมดูล หรือโปรแกรมประยุกต์ที่ใช้ทำให้เกิด ชุดทดสอบที่จัดให้อินพุตทดสอบ(ข้อมูล) ควบคุมและดูแลการดำเนินการ โดยเทสต์ไดร์เวอร์จะทำการป้อนข้อมูลจำนวนมากสำหรับเทสต์เคสแทนคีย์บอร์ด และเมาส์ของระบบที่ทดสอบ และเทสต์สตับ เป็นองค์ประกอบซอฟท์แวร์ที่เลียนแบบ

หรือออบเจ็กต์ที่ใช้(ในระหว่างการพัฒนาและทดสอบ) ในการลอกเลียนพฤติกรรมขององค์ประกอบจริง โดยปกติเทสต์สตับจะจัดให้เอาต์พุตทดสอบ แต่ในที่นี้เราจะสนใจเทสต์ไดร์เวอร์วิธีเดียว

โดยระบบการเทสต์ไดร์เวอร์เป็นการใช้พื้นฐานบน JDBC ซึ่งใช้จาวาในการให้ SQL เรียกฐานข้อมูล โดยในช่วงการทำงานระบบการเทสต์ไดร์เวอร์ใช้ ODBC หรือ MySQL เพื่อส่งการเรียก SQL ให้ ORACLE(เป็นวิธีการผลิตผลลัพธ์การทำนายเปรียบเทียบกับผลลัพธ์จริงของซอฟท์แวร์ภายใต้การทดสอบ)

4.4. แฟ้มเชื่อมประสานของระบบเทสต์ไดร์เวอร์กุย

เทสต์ไดร์เวอร์กุยจะเป็นโปรแกรมที่ทำการช่วยให้ผู้ทดสอบสามารถสร้างแฟ้มอินพุตสำหรับเทสต์ไดร์เวอร์ได้ง่ายขึ้นผ่านทางการใช้แบบกราฟิก โดยจะทำการสร้างแฟ้มอินพุต และจะทำการเรียกเทสต์ไดร์เวอร์ให้ทำงานซึ่งจะได้ผลลัพธ์ออกมาเป็นแฟ้มเอาต์พุตซึ่ง ทั้งแฟ้มอินพุตและเอาต์พุตถูกกำหนดรูปแบบเป็น XML เพื่อเป็นมาตรฐานในการแลกเปลี่ยนชุดทดสอบ ซึ่งมีโครงสร้างและการทำงานของเทสต์ไดร์เวอร์กุยดังรูปที่ 4.4



รูปที่ 4.4 โครงสร้างและการทำงานของเทสต์ไดร์เวอร์กุย

จากรูปที่ 4.4 โครงสร้างและการทำงานของเทสต์ไดร์เวอร์กุย ประกอบด้วย สามองค์ประกอบที่ติดต่อกัน คือ

- องค์ประกอบแรก (Test driver GUI) เป็นส่วนติดต่อประสานกับผู้ใช้ เช่น ช่วยให้ผู้ใช้สามารถสร้างอินพุตไฟล์ขึ้นมาเพื่อใช้ในการทดสอบ เป็นต้น

- องค์ประกอบสอง (Test Driver) จะทำการอ่านแฟ้มอินพุตขึ้นมาโดยใช้คลาส Parser เสร็จแล้วจะทำการโหลด Oracle ขึ้นมาโดยจะนำค่าที่เกี่ยวกับการเชื่อมต่อที่อ่านมาจากแฟ้มอินพุตส่งไปให้ Oracle ผ่านทาง method prepare Connection จากอินเทอร์เฟส IcheckDB แล้วก็จะเรียก method beforeQuery เพื่อส่งคำสั่ง SQL ที่จะคิวรี ส่งไปให้ Oracle ก่อน จึงอ่านคำสั่ง SQL ส่งไปให้คลาส DBClient เพื่อทำการคิวรีไปที่ DBMS จากนั้น Test Driver จะรอ DBClient ส่งผลลัพธ์การคิวรี เมื่อได้รับเรียบร้อยจะเรียก method afterQuery ไปยัง Oracle เพื่อขอทราบผลลัพธ์การตรวจสอบความถูกต้องของ DBMS เมื่อ Oracle ส่งผลลัพธ์การตรวจสอบกลับมา ก็จะรวบรวมผลลัพธ์ของทุกชุดทดสอบแล้วเขียนลงยังเอาต์พุต

- องค์ประกอบสุดท้าย (ORACLE) โดยในช่วงแรกองค์ประกอบเทสต์ไดร์เวอร์จะทำการเรียก method prepare Connection ก่อนโดยจะส่งค่าต่างๆที่จำเป็นในการติดต่อฐานข้อมูลให้แก่ Oracle มีดังนี้ url (URL ของฐานข้อมูลที่ทำการทดสอบ) dbDriver (คลาส JDBC Driver ของ DBMS ที่เราจะทดสอบ) user และpassword (ชื่อผู้ใช้และรหัสผ่านที่ใช้สำหรับฐานข้อมูล) ช่วงที่สอง รับคำสั่ง SQL ที่จะทำคิวรีจากองค์ประกอบเทสต์ไดร์เวอร์ซึ่งจะเรียก method beforeQuery ก่อนจะทำคิวรีไปที่ DBMS ช่วงสุดท้าย องค์ประกอบเทสต์ไดร์เวอร์เรียก method afterQuery แล้ว oracle จะต้องทำการตรวจสอบความถูกต้องของ DBMS และคืนค่าเป็น true ถ้าการทดสอบถูกต้อง และเป็น false ถ้าการทดสอบได้ผลผิดพลาด

- DBClient เป็นส่วนหนึ่งขององค์ประกอบเทสต์ไดร์เวอร์คอยรับคำสั่ง SQL แล้วคิวรีไปยัง DBMS และรับผลลัพธ์การคิวรีจาก DBMS ไปให้องค์ประกอบเทสต์ไดร์เวอร์

- DBMS เป็นชนิดระบบการจัดการฐานข้อมูลของฐานข้อมูลที่เราทดสอบดีเอ็มแอล ซึ่งสนับสนุน ODBC และ MySQL

- แฟ้มอินพุต และเอาต์พุต จากที่กล่าวมาแล้วว่าทั้งแฟ้มอินพุตและเอาต์พุตในระบบเทสต์ไดร์เวอร์กุย ถูกกำหนดรูปแบบเป็น XML(eXtensible Markup Language) เพื่อเป็นมาตรฐานในการแลกเปลี่ยนชุดทดสอบ ซึ่งเราจะสนใจใช้วิธีการกำหนดรูปแบบชุดทดสอบและการกำหนดความหมายข้อมูลโดยใช้ DTD (Data type Definition) ซึ่งเราสามารถกำหนดรูปแบบได้เองโดยรูปแบบแฟ้มอินพุตแสดงดังรูปที่ 4.5 ส่วนแฟ้มเอาต์พุตแสดงดังรูปที่ 4.6

```
1 <?xml version = '1.0' encoding='UTF-8'?>
2 <!ELEMENT DBDriver (#PCDATA)>
3 <!ELEMENT DBPassword (#PCDATA)>
4 <!ELEMENT DBURL (#PCDATA)>
5 <!ELEMENT DBUser (#PCDATA)>
6 <!ELEMENT Oracle (#PCDATA)>
```

```
7 <!ELEMENT SQL (#PCDATA)>
8 <!ELEMENT TestCase (SQL)>
9 <!ELEMENT TestDriver (DBURL, DBDriver, DBUser, DBPassword, Oracle, Testset)>
10 <!ELEMENT TestSet (TestCase+)>
```

รูปที่ 4.5 แฟ้มอินพุต (Data Type Definition)

- Tag DBURL จะเป็นตัวระบุ URL ของ DBMS ที่เราต้องการจะทดสอบ

- Tag DBDriver เป็นชื่อ class JDBC Driver ของ DBMS ที่เราจะทดสอบ

- Tag DBUser, DBPassword เป็น username และ password สำหรับเชื่อมต่อกับ DBMS ถ้าไม่มีเว้นว่าง

- Tag Oracle คือชื่อคลาส Oracle ที่เป็นตัวตรวจสอบความถูกต้องของ DBMS

- Tag TestSet เป็นชุดของ TestCase

- Tag TestCase คือ Testcase ที่เราต้องการจะทดสอบโดย 1 sql command เท่ากับ 1 Test Case โดย TestCase ประกอบไปด้วย tag SQL จะเป็น SQL command

```
1 <?xml version = '1.0' encoding='UTF-8'?>
2 <!ELEMENT DBURL(#PCDATA)>
3 <!ELEMENT inputFile (#PCDATA)>
4 <!ELEMENT QueryResult (#PCDATA)>
5 <!ELEMENT SQL (#PCDATA)>
6 <!ELEMENT TestCase (SQL, QueryResult, TestResult)>
7 <!ELEMENT TestDriver (DBURL, InputFile, TestSet)>
8 <!ELEMENT TestResult (#PCDATA)>
```

รูปที่ 4.6 แฟ้มเอาต์พุต (Data Type Definition)

- Tag DBURL จะเป็น URL ของ DBMS ที่เราทำการทดสอบได้ข้อมูลมาจากแฟ้มอินพุต

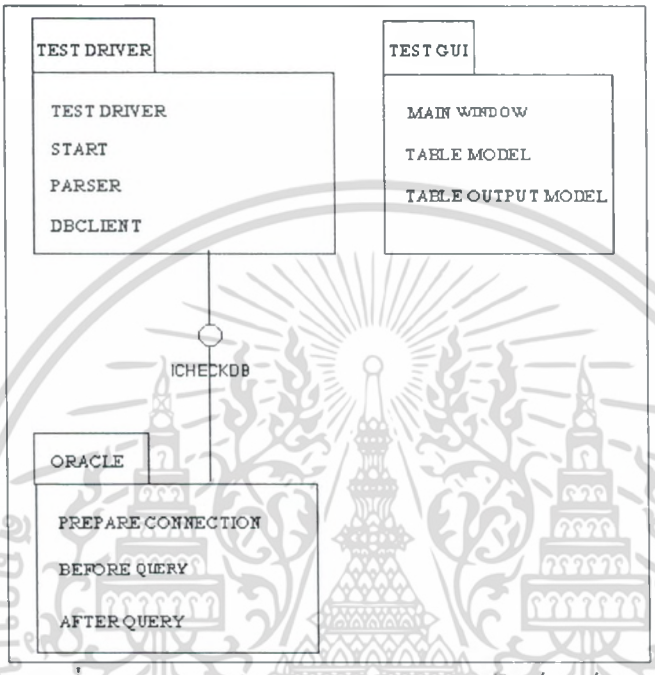
- Tag InputFile เป็นชื่อแฟ้มอินพุตที่ทำการทดสอบ

- Tag TestSet คือชุดของ TestCase ที่ทำการทดสอบ

- Tag TestCase จะเป็นผลลัพธ์ของแต่ละ Test Case โดยประกอบด้วย tag SQL คือ SQL command ที่เราทำการทดสอบ tag QueryResult จะเก็บผลลัพธ์จากการทำการ query ตามคำสั่งจาก

tag SQL และ tag TestResult จะเก็บผลจากการทดสอบ จาก Oracle จะให้ค่า true ถ้าการทดสอบแล้วปรากฏว่าผลลัพธ์ถูกต้องและ false เมื่อผลการทดสอบผิด

กรอบการทำงานของระบบเทสต์ไดร์เวอร์กุย ประกอบด้วยแพ็คเกจต่างๆ ได้แก่ TEST DRIVER, TEST GUI และ ORACLE โดยที่ภายในแต่ละแพ็คเกจต่างๆ จะประกอบด้วยคลาสต่างๆ ดังรูปที่ 4.7



รูปที่ 4.7 กรอบการทำงานของระบบเทสต์ไดร์เวอร์กุย

- START เป็นคลาสแรกที่ทำงานเมื่อระบบเทสต์ไดร์เวอร์กุยถูกเรียกใช้ โดยจะทำการไปเรียกใช้คลาส MAIN WINDOW ที่อยู่ในแพ็คเกจ TEST GUI
- TEST DRIVER
  - เป็นคลาสที่ทำการอ่านแฟ้มอินพุตโดยเรียกใช้คลาส PARSER
  - นำค่าต่างๆ เกี่ยวกับการเชื่อมต่อที่อ่านมาจากแฟ้มอินพุตส่งให้Oracleผ่านทางเมทธอด prepare Connection
  - ส่งคำสั่ง SQL ไปให้ Oracle โดยเรียกใช้จากเมทธอด beforeQuery
  - ส่งคำสั่ง SQL ไปให้คลาส DBCLIENT
  - เรียกเมทธอด afterQuery ไปยัง Oracle เพื่อขอทราบผลลัพธ์การตรวจสอบความถูกต้องของ DBMS
  - รวบรวมผลลัพธ์จากการคิวรี และการตรวจสอบเขียนลงยังแฟ้มเอาต์พุต
- PARSER อ่านค่าต่างๆ จากแฟ้มอินพุต

- DBCLIENT คิวรีคำสั่ง SQL ที่ได้ไปยัง DBMS และส่งผลลัพธ์การคิวรีไปยังคลาส TEST DRIVER

- ICHECKDB เป็นอินเตอร์เฟสระหว่างแพ็คเกจ TEST DRIVER กับ ORACLE

- MAIN WINDOW เป็นส่วนประสานกราฟิกกับผู้ใช้หลัก

- TABLE MODEL เป็นส่วนประสานกราฟิกกับผู้ใช้ที่แสดงในส่วนรายละเอียดแต่ละเทสต์เคสที่สร้างขึ้นมา

- TABLE OUTPUT MODEL เป็นส่วนประสานกราฟิกกับผู้ใช้ที่แสดงผลลัพธ์ โดยประกอบด้วยคำสั่ง SQL ผลลัพธ์ที่เกิดจากคิวรี และผลการทดสอบจากโปรแกรม

- prepareConnection เป็นเมทธอดที่ใช้ในการรับ และเตรียมค่าต่างๆ ในการเชื่อมต่อ

- beforeQuery เป็นเมทธอดที่ใช้ในการรับคำสั่ง SQL จาก TEST DRIVER

- afterQuery เป็นเมทธอดที่ใช้ทดสอบคำสั่ง SQL ที่ได้จาก เมทธอด beforeQuery เมื่อตรวจสอบเรียบร้อยจะคืนค่าออกมาเป็น true หรือ false ส่งกลับไปยังคลาส TEST DRIVER

ระบบเทสต์ไดร์เวอร์กุยสำหรับทดสอบดีเอ็มแอลของฐานข้อมูลช่วยให้ผู้ใช้สามารถสร้างชุดทดสอบต่างๆ ได้ง่ายและสะดวกซึ่งสามารถเก็บไว้ในรูปแบบ XML ซึ่งเป็นมาตรฐานในการแลกเปลี่ยนแฟ้ม โดยเราสามารถนำชุดทดสอบนี้นำไปใช้ทดสอบกับระบบการจัดการฐานข้อมูล (DBMSs) อื่นๆซึ่งเราไม่จำเป็นต้องไปนั่งคีย์หรือสร้างชุดทดสอบใหม่อีกครั้ง และอาจนำไปใช้กับเครื่องมือทดสอบอื่นๆได้(ต้องสนับสนุน XML) ซึ่งจะช่วยให้ผู้ใช้ประหยัดเวลาอย่างมาก นอกจากจะช่วยสร้างแฟ้มอินพุตแล้ว เมื่อทดสอบเสร็จระบบเทสต์ไดร์เวอร์กุยยังสนับสนุนในการรวบรวมผลลัพธ์การทดสอบจากชุดทดสอบต่างๆ มาเก็บในรูปแบบของ XML ด้วยซึ่งเราสามารถเก็บผลลัพธ์นี้เพื่อใช้ในการวิเคราะห์หาจุดผิดพลาด หรือเปรียบเทียบผลได้ในภายหลัง และยังมีมาตรฐานในการแลกเปลี่ยนแฟ้มอีกด้วย

บทที่ 5

สรุปผล และข้อเสนอแนะ

5.1 สรุปผล

งานวิจัยนี้ได้นำเสนอสถาปัตยกรรมของเครื่องมือทดสอบซอฟท์แวร์โดยอัตโนมัติ นั่นคือ สถาปัตยกรรมของยูนิเวอร์แซลเทสต์ไดร์เวอร์ ซึ่งเป็นสถาปัตยกรรมหนึ่งที่นำแฟ้มรูปแบบ XML มาใช้เป็นแฟ้มเชื่อมประสานระหว่างคลังข้อมูล โดยรูปแบบแฟ้มเชื่อมประสาน XML ได้ถูกกำหนดด้วย DTD หรือ XML Schema ก็ได้แต่ปัจจุบันนิยมใช้ XML Schema ในการกำหนดรูปแบบแฟ้ม XML

สถาปัตยกรรมที่ได้นำเสนอในงานวิจัยนี้เป็นรูปแบบสถาปัตยกรรมหนึ่งของเครื่องมือทดสอบซอฟท์แวร์โดยอัตโนมัติที่ทำการทดสอบด้วยเอกสารสคริปต์ ซึ่งเอกสารสคริปต์นี้สร้างจากแฟ้มเชื่อมประสาน XML ที่เป็นตัวกลางในการนำข้อมูลการทดสอบเข้าและออกจากคลังข้อมูล ดังนั้นจากหลักการทำงานของยูนิเวอร์แซลเทสต์ไดร์เวอร์จะเห็นได้ว่าแฟ้มเชื่อมประสาน XML สามารถเป็นรูปแบบข้อมูลกลางในการแลกเปลี่ยนข้อมูลหรือกรณีทดสอบที่บันทึกในคลังข้อมูลได้ โดยที่จากรูปแบบข้อมูลในแฟ้ม XML จะสามารถแปลงเป็นรูปแบบเอกสารสคริปต์ต่าง ๆ ได้โดยสะดวกผ่านขั้นตอนการแปลงรูปแบบแฟ้มเอกสาร XML ทั่วไป นอกจากนี้แฟ้มรูปแบบ XML ยังเป็นที่ยอมรับเป็นรูปแบบมาตรฐานในระบบเครือข่ายคอมพิวเตอร์โลกทำให้สามารถใช้ในการแลกเปลี่ยนกรณีทดสอบผ่านเครือข่ายคอมพิวเตอร์ได้อีกด้วย

รูปแบบแฟ้มเชื่อมประสาน XML ตามที่กล่าวในสถาปัตยกรรมยูนิเวอร์แซลเทสต์ไดรเวอร์เป็นรูปแบบที่ครอบคลุมสำหรับการทดสอบประเภทต่าง ๆ ไม่ว่าจะทำการทดสอบโดยอัตโนมัติหรือไม่ ซึ่งการกำหนดรูปแบบแฟ้มเชื่อมประสานนี้อาจมีรูปแบบที่ต่างกันออกไปได้อีก ขึ้นกับความเหมาะสมกับข้อมูลกรณีทดสอบในคลัง และข้อตกลงในการเชื่อมต่อระหว่างระบบหรือเครื่องมือที่ใช้แฟ้มเชื่อมประสานในการแลกเปลี่ยนกรณีทดสอบ แต่สำหรับการนำกรณีทดสอบจากแฟ้มเชื่อมประสานไปใช้ทดสอบนั้นสามารถทำได้หลากหลายวิธี ซึ่งการสร้างเป็นเอกสารสคริปต์ก็เป็นแนวหนึ่งดังที่ได้กล่าวในบทที่เกี่ยวกับสถาปัตยกรรมยูนิเวอร์แซลเทสต์ไดรเวอร์ เป็นต้น

กรณีศึกษาการออกแบบแฟ้มเชื่อมประสานสำหรับระบบทดสอบ DML ฐานข้อมูล พบว่าสามารถช่วยให้ผู้ใช้สามารถสร้างชุดทดสอบต่างๆ ได้ง่ายและสะดวกซึ่งสามารถเก็บไว้ในรูปแบบ XML ซึ่งเป็นมาตรฐานในการแลกเปลี่ยนแฟ้ม โดยเราสามารถนำชุดทดสอบนี้นำไปใช้ทดสอบกับระบบการจัดการฐานข้อมูล (DBMSs) อื่นๆซึ่งเราไม่จำเป็นต้องไปนั่งคีย์หรือสร้างชุดทดสอบใหม่อีกครั้ง ซึ่งเราสามารถเก็บผลลัพธ์นี้เพื่อใช้ในการวิเคราะห์หาจุดผิดพลาด หรือเปรียบเทียบผลได้ในภายหลัง และยังมีมาตรฐานในการแลกเปลี่ยนแฟ้มอีกด้วย

จากการศึกษาการทำคลังเก็บกรณีทดสอบเพื่อสามารถนำกรณีทดสอบมาใช้ซ้ำ สถาปัตยกรรมที่ได้นำเสนอนั้น มีความเป็นไปได้ที่นำไปสู่การพัฒนาเครื่องทดสอบซอฟท์แวร์แบบถดถอยโดยอัตโนมัติต่อไป

5.2 ข้อเสนอแนะ

สำหรับสถาปัตยกรรมยูนิเวอร์แซลเทสต์ไดรเวอร์ที่นำเสนอในงานวิจัยนี้เป็นเพียงแนวทางหนึ่งในการนำแฟ้มเชื่อมประสาน XML มาช่วยในการทดสอบซอฟท์แวร์ ซึ่งสถาปัตยกรรมนี้สามารถนำประยุกต์ใช้เพื่อพัฒนาเครื่องมือการทดสอบให้เป็นอัตโนมัติ โดยเครื่องมือดังกล่าวอาจรองรับการตรวจจับการเปลี่ยนแปลงกรณีทดสอบ (Trigger) ในคลังข้อมูลเพื่อทำการทดสอบซอฟท์แวร์สำหรับกรณีนั้น ๆ ใหม่ เป็นต้น นอกจากนั้นสถาปัตยกรรมยูนิเวอร์แซลเทสต์ไดรเวอร์ได้มีการออกแบบเพื่อการทดสอบซอฟท์แวร์โดยไม่จำกัดระบบปฏิบัติการ โดยสรุปแล้วสถาปัตยกรรมนี้เป็นตัวอย่างหนึ่งที่มีแนวคิดที่สามารถนำไปใช้พัฒนาระบบหรือเครื่องมือการทดสอบซอฟท์แวร์แบบอัตโนมัติ โดยใช้เอกสารสคริปต์ในการทดสอบที่ง่ายต่อการทำความเข้าใจและปรับปรุงแก้ไข และมีคลังกรณีทดสอบที่ช่วยให้ผู้ทดสอบไม่ต้องสร้างกรณีทดสอบใหม่เสมอ ๆ สำหรับการทดสอบแบบถดถอย แต่ในการนำไปพัฒนาเครื่องมือนั้นจะมีความเสี่ยงเกี่ยวกับเรื่องเทคโนโลยีซึ่งขึ้นกับประสบการณ์ขององค์กรการทำงานเกี่ยวกับเอกซ์เอ็มแอล ซึ่งในระยะเริ่มแรกของการออกแบบ XSL จะค่อนข้างซับซ้อนมากกว่าการทำสคริปต์สำหรับการทดสอบหนึ่ง ๆ เพราะต้องสร้างให้สามารถใช้งานได้ครอบคลุมการทดสอบทุกรูปแบบด้วย

ภาคผนวก ก.

ตัวอย่าง DTD ของเอกสารของการทดสอบ (UTD)

```
<!-- KMITL Test Document DTD - version 1.0 -->
<!--Copyright (c) 2003-2004, Infomation Technology @ KMITL.ac.th-->

<!ELEMENT BMCTD              (DESCRIPTION?, REPOSITORY? ) >
<!ELEMENT REPOSITORY         (PROJECT) * >
<!ELEMENT PROJECT            (DESCRIPTION?, (TESTLIST | FOLDER) * )
>
<!ELEMENT TESTLIST           (DESCRIPTION?, (TEST) * )  >
<!ELEMENT FOLDER             (FOLDER | TEST) *  >
<!ELEMENT TEST               (DESCRIPTION?, (RUN | STEP) * )  >
<!ELEMENT RUN                ( (STEP) * , STATUS? , DURATION?)  >
<!ELEMENT STEP               (DESCRIPTION? , EXPECTED? , ACTUAL? ,
STATUS?) >
<!ELEMENT DESCRIPTION        (#PCDATA)  >
<!ELEMENT EXPECTED           (#PCDATA)  >
<!ELEMENT ACTUAL             (#PCDATA)  >
<!ELEMENT STATUS             (#PCDATA)  >
<!ELEMENT DURATION           (#PCDATA)  >
<!ATTLIST BMCTD
        title                CDATA " (untitled) "
        author               CDATA " (unknown) "
        creation_date        CDATA " (unknown) "
        last_modified        CDATA " (unknown) "
        copyright            CDATA "Copyright (c) 2003-2004, Infomation
Technology @ KMITL.ac.th "
>
<!ATTLIST REPOSITORY
        name                 CDATA #REQUIRED
        settings             CDATA #IMPLIED
>
<!ATTLIST PROJECT
        name                 CDATA #REQUIRED
        settings             CDATA #IMPLIED
>
<!ATTLIST TESTLIST
        name                 CDATA #REQUIRED
        status               CDATA #IMPLIED
>
<!ATTLIST FOLDER
        name                 CDATA #REQUIRED
        folder               CDATA #IMPLIED
>
<!ATTLIST TEST
        name                 CDATA #REQUIRED
        number               CDATA #IMPLIED
```

บรรณานุกรม

Beizer, Boris. 1982. **Software Testing Techniques.** Van Nostrand Reinhold electrical/Computer Science and engineering series.

Beizer, Boris. 1995. **Black-Box Testing: Techniques for Functional Testing of Software and Systems.** John Wiley & Sons.

Castro, Elizabeth. 2001. **XML for the World Wide Web: Visual QuickStart Guide.** Peachpit Press

Cagle , Kurt. et al. 2001. **Professional XSL.** Wrox Press.

Cordrey, Gordon. 2002. "Universal Test Driver: A Data-Driven Framework for Script-Based Testing". **The Fourth International Conference on Practical Software Testing Techniques.** New Orleans, Louisiana, USA.

Chays, David., Vokolos, Filippos I., and Weyuker , Elaine J. 2000. "A Framework for Testing Database Applications" **Proc. International Symposium on Software Testing and Analysis (ISSTA2000)**, Portland, Or. USA.

Patton, Ron. 2000. **Software Testing.** Indiana: Sams Publishing.

Poston, Robert. 1994 a. "Moving from Manual to Automated Test Execution". **Comm. ACM**. 37, 9(Sept. 1994): 124-129.

Poston, Robert. 1994 b. "Automating Testing from Object Models".. **Comm. ACM**. 37, 9 (Sept. 1994): 48-58.

Mean, W. Scott. and Bodie, Michael A. 2001. **The Book of SAX: the Simple API for XML.** No Starch Press.

Walmsley, Priscilla. 2002. **Definitive XML Schema.** Prentice-Hall Inc.

```
        type                    CDATA "MANUAL"
        subtype                 CDATA #IMPLIED
        status                  ( Ready | Design ) "Ready"
        folder                  CDATA #IMPLIED
        responsible             CDATA " "
        creation_date           CDATA #IMPLIED
        timeout                 CDATA #IMPLIED
        attachment              CDATA #IMPLIED
        plan_scheduling_date    CDATA #IMPLIED
        plan_scheduling_time    CDATA #IMPLIED
        runtime_attachment      CDATA #IMPLIED
        extended_parameters     CDATA #IMPLIED
>
<!ATTLIST RUN
        name                    CDATA #REQUIRED
        tester_name             CDATA " "
        execution_date          CDATA #IMPLIED
        execution_time          CDATA #IMPLIED
        host_name               CDATA #IMPLIED
        path                    CDATA " "
>
<!ATTLIST STEP
        name                    CDATA #REQUIRED
        execution_date          CDATA #IMPLIED
        execution_time          CDATA #IMPLIED
        path                    CDATA #IMPLIED
        line_number             CDATA “0”
>
```