การแยกประเภทกรณีทดสอบด้วย Finite State Machine แบบกลุ่มลักษณะ

Test Case Classification using Category-Partition Finite State Machine

เพ็ญพิชา ศุพผลา อุมาพร ลีลานันทกุล นุชชากร งามเสาวรส พีระพนธ์ โสพัศสถิตย์

Advanced Virtual and Intelligent Computing (AVIC) Center

ภาควิชาคณิตศาสตร์ คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

Email: xuixui5@hotmail.com, amp_le@hotmail.com

บทคัดย่อ

การทดสอบเป็นกิจกรรมสำคัญในวัฏจักรของการพัฒนาซอฟต์แวร์ ซึ่งเป็นสิ่งที่ผู้ทดสอบและผู้พัฒนาถูกคาดหวังว่าจะต้องดำเนินการก่อนส่งมอบซอฟต์แวร์ที่ปราศจากข้อผิดพลาด การรับประกันเช่นนั้นด้วยการทดสอบทุกกรณีเป็นสิ่งที่ทำไม่ได้ในทางปฏิบัติ จึงมีความพยายามที่จะสร้างกรณีทดสอบที่น่าเชื่อถือ อันหมายถึงครอบคลุมได้อย่างทั่วถึง โดยทั่วไปมักจะใช้ข้อมูลจำนวนมากในการสร้างกรณีทดสอบด้วยค่าใช้จ่ายสูง และมักกลายเป็นอุปสรรคของการทดสอบ บทความนี้เสนอขั้นตอนวิธีเชิงปฏิบัติง่ายๆ ซึ่งจะช่วยลดขนาดของกรณีทดสอบ แนวคิดหลัก อาศัยการแยกแยะความสัมพันธ์ข้อมูลออกเป็นประเภทตามข้อกำหนดและข้อจำกัดของโปรแกรม เพื่อสร้าง finite state machine เส้นทางทั้งหมดที่ได้จากจุดเริ่มต้นไปจนถึงจุดสิ้นสุด จะแทนกรอบทดสอบทั้งหมด ซึ่งวิธีนี้ลดปริมาณกรณีศึกษาลงได้มากเมื่อเทียบกับวิธีทั่วไป

Abstract

Testing is an essential activity in software development process. Testers and developers alike are facing a formidable expectation of delivery bug-free software. Certifying bug-free with exhaustive test is commonly known to be impossible. Numerous efforts have been attempted to arrive at a plausible test scenario wherein thorough coverage can be attained. Conventional approaches usually require large amount of test data (or input domain) to generate necessary test cases at premium expenses which, in many cases, end up to be a recalcitrant test process. This paper proposes a straightforward, yet practical algorithmic method to reduce all relevant test cases. The central idea rests upon identifying the relationships among category partition of input specifications and program constraints that subsequently are employed to construct a finite state machine. As such, all paths connecting the start and end states represent the required test cases. Reduction on the number of generated test frames based on the proposed method in comparison with conventional approaches proves to be quite significant.

คำสำคัญ: Black-box testing, category-partition, pairwise testing, software component, test case generation

1. บทนำ

การพัฒนาซอฟต์แวร์หรือเรียกในกรอบที่แคบลงว่าโปรแกรมนั้น มักจะหนีไม่พ้นอุปสรรคที่สำคัญอย่างหนึ่งคือ ข้อผิดพลาด วิธีที่ผู้พัฒนาจะรู้ว่าโปรแกรมมีข้อผิดพลาด ก็ต้องทดสอบโดยเลือกกรณีทดสอบ (test case) เมื่อโปรแกรมมีความซับซ้อนมากขึ้นและมีขนาดใหญ่ขึ้นจนกลายเป็นระบบโปรแกรมและซอฟต์แวร์ใน

ที่สุด กรณีทดสอบก็มีจำนวนมากขึ้นเป็นเงาตามตัว นักวิชาการได้ศึกษาวิจัยกระบวนการทดสอบในปริมาณมากๆ ด้วยวิธีทางสถิติ (statistical testing) [4] ด้วยการสุ่มข้อมูลนำเข้า (random inputs) เพื่อจะได้ test profile และ test size ของข้อมูลที่จะใช้ทดสอบ โดยพิจารณา structural functionality หรือ black-box ของซอฟต์แวร์ และใช้เกณฑ์ที่กำหนดจาก behavior model ตาม specification ของซอฟต์แวร์นั้น

อย่างไรก็ตาม ข้อผิดพลาดในซอฟต์แวร์ยังเป็นสิ่งที่ผู้พัฒนาไม่สามารถป้องกันหรือกำจัดให้หมดสิ้นไป จึงมีการประเมินวิธีทดสอบก่อนที่ซอฟต์แวร์ให้ได้คุณภาพตามที่กำหนด ก่อนที่จะถูกส่งมอบแก่ผู้ใช้ ความจำเป็นของการสร้าง test coverage ตาม dependability theory [3] การเลือกกรณีทดสอบจากกลุ่มข้อมูลจำนวนมาก และสร้างกรณีทดสอบเหล่านั้นโดยอัตโนมัติ ซึ่งมีข้อมูลพอเพียงกับความต้องการ ทำให้มีการกำหนดแนวทางในการพิจารณาว่าคุณสมบัติ เกณฑ์สิ้นสุด (stopping rules) และตัววัด (measurement) ใดที่จะใช้สร้างกรณีทดสอบที่พอเพียงกับ statement, path, branch coverage [6] ในทางปฏิบัติ การทดสอบมักกระทำภายใต้ข้อจำกัดด้านเงินทุนและเวลา ดังนั้น จึงต้องมีการวางแผนการทดสอบอย่างดีและเลือกกรณีทดสอบที่ดี เพื่อให้ได้จำนวนกรณีทดสอบน้อย แต่สามารถพบข้อผิดพลาดได้จำนวนมาก

บทความนี้นำเสนอวิธีการลดกรณีทดสอบ โดยใช้วิธี Category-Partition Method ประยุกต์รวมกับ Pairwise Method ซึ่งมีสาระสำคัญของบทความมีดังนี้ ขั้นแรกจะกล่าวถึงกระบวนการสร้างกรณีทดสอบ โดยวิธี Category-Partition Method จากนั้นใช้วิธีการ Pairwise Method เพื่อสร้างกรอบทดสอบ (test frame) ซึ่งนำไปสร้างกรณีทดสอบ พร้อมทั้งแสดงตัวอย่างการสร้างกรอบทดสอบ และบทสรุปของการประยุกต์ขั้นตอนวิธีกับปัญหาการทดสอบ

2. กระบวนการสร้างกรณีทดสอบ

การทดสอบแบบ black-box ในยุคแรกมักใช้วิธีสร้างข้อมูลทดสอบแบบสุ่ม (random test data generation) แต่จะต้องสร้างกรณีทดสอบจำนวนมากเพื่อให้ครอบคลุมกรณีต่างๆครบถ้วน หรือ Pairwise Method [8] ถึงจะช่วยลดกรณีทดสอบที่ซ้ำซ้อนลง แต่พารามิเตอร์ที่ใช้ต้องเป็นอิสระต่อกัน

การแบ่งกลุ่มข้อมูลเพื่อสร้างกรณีทดสอบเป็นอีกเทคนิคหนึ่งที่ช่วยลดปริมาณข้อมูลที่ต้องพิจารณา รวมทั้งจำนวนตัวแทนข้อมูล บทความนี้พัฒนาขั้นตอนวิธีจากการทดสอบแบบ Pairwise Method ประยุกต์รวมกับเทคนิค Category-Partition Method [1, 2] ซึ่งเป็นวิธีเป็นการสร้างกรณีทดสอบจากข้อกำหนดคุณลักษณะของโปรแกรม (specification-based testing) มาช่วยในการแบ่งกลุ่มความสัมพันธ์ของขอบเขตของข้อมูลนำเข้าให้พอเพียงกับความครอบคลุม (test coverage) ตามที่กำหนด จึงเป็นการลดจำนวนกรณีทดสอบให้น้อยลง โดยใช้ตัวแทนกลุ่ม (category หรือ class partition) แต่สามารถครอบคลุมได้เท่ากับการใช้กรณีทดสอบจำนวนที่มากกว่าในวิธีอื่น ซึ่งอาจเลือกมาจากกรณีตัวแทนที่ซ้ำซ้อนจากกลุ่มเดียวกัน

ในการวิเคราะห์หาความสัมพันธ์เพื่อแบ่งประเภท (category) ของขอบเขตข้อมูลนำเข้าสำหรับโปรแกรมที่ต้องการทดสอบให้ครอบคลุมในทุกๆ กรณีทดสอบ โดยไม่เป็นการเสียเวลากระบวนการทดสอบ คุ้มค่ากับเวลาที่ต้องการความรวดเร็ว จะพิจารณาขอบเขตของข้อมูลนำเข้าที่เป็นไปได้ และเงื่อนไขต่างๆของสภาวะแวดล้อมที่เกี่ยวข้องในการทดสอบโปรแกรม เพื่อทำการแบ่งชนิดของขอบเขตของข้อมูลนำเข้าและสภาวะแวดล้อมออกเป็นชุดๆ เรียกว่า “parameter” ซึ่งแต่ละชุดจะถูกแบ่งออกเป็น “choice” โดยพิจารณาจากเงื่อนไขในการทำงานของโปรแกรมที่ต้องการทดสอบ แล้วกำหนดความสัมพันธ์ของพารามิเตอร์เหล่านั้นจากคุณลักษณะ (attribute) ให้กับแต่ละ choice รวมถึงเงื่อนไขการเข้าถึง choice มาสร้าง finite state machine (FSM ซึ่งต่อจากนี้จะเรียกสั้นๆ ว่า state machine) เพื่อหาเส้นทางเดินที่

เป็นไปได้ใน state machine นี้ แต่ละเส้นทางที่เดินจากสถานะเริ่มต้น (start state) ไปจนถึงสถานะสุดท้าย (final state) จะแทนกรอบของข้อมูลซึ่งนำไปสร้างกรณีทดสอบ โดยมีรายละเอียดขั้นตอนดังนี้

ขั้นที่ 1 : การวิเคราะห์ข้อกำหนดต่างๆ ในการทำงานของโปรแกรมที่จะทดสอบ

- วิเคราะห์ขอบเขตของข้อมูลนำเข้าและสภาวะแวดล้อมที่เป็นไปได้สำหรับโปรแกรมที่นำมาทดสอบ แล้วแบ่งออกเป็นกลุ่มๆ เรียกว่า “parameter” หลังจากนั้นพิจารณาแต่ละพารามิเตอร์ เพื่อแบ่งเป็น choice ตามเงื่อนไขของโปรแกรมที่นำมาทดสอบ หลักในการพิจารณาพารามิเตอร์ มีดังนี้
  - พารามิเตอร์ในขอบเขตข้อมูลนำเข้า
    - พิจารณาคุณลักษณะของพารามิเตอร์ที่ใช้กับโปรแกรม
    - พิจารณาลักษณะเฉพาะที่ต้องการของพารามิเตอร์จากโปรแกรม
  - พารามิเตอร์ในสภาวะแวดล้อม
    - พิจารณาเงื่อนไขพารามิเตอร์ต่างๆ ที่มีผลกับการทำงานของโปรแกรม
    - พิจารณาลักษณะเฉพาะของพารามิเตอร์ที่ส่งผลกระทบต่อการทำงานของโปรแกรม
- รวบรวมพารามิเตอร์ของขอบเขตของข้อมูลนำเข้าและสภาวะแวดล้อมต่างๆ ที่เป็นไปได้ กำหนด choice ของพารามิเตอร์ตามลักษณะเฉพาะของพารามิเตอร์นั้นๆ

ขั้นที่ 2 : การกำหนดความสัมพันธ์ของ choice

- พิจารณาความสัมพันธ์ของพารามิเตอร์ในขั้นที่ 1 โดยการกำหนดคุณลักษณะ เบื้องต้นให้กับ choice ของพารามิเตอร์สภาวะแวดล้อมแต่ละกลุ่ม และเงื่อนไขในการเข้าถึง choice ที่เป็นไปได้ ซึ่งแสดงความสัมพันธ์ ระหว่าง choice จากนั้นพิจารณาว่า choice ใดบ้างที่เป็นสาเหตุให้เกิดความผิดพลาด (ในการทดลองจะใช้สัญลักษณ์ [error]) ในโปรแกรม กำกับหมายเหตุความผิดพลาดให้กับ choice นั้น จากนั้นเรียงลำดับความสำคัญของ choice

ขั้นที่ 3 : การสร้าง state machine จากความสัมพันธ์ของ choice

- นำความสัมพันธ์ของ choice ที่ได้ มาสร้าง state machine โดยกำหนดโหนดแรกเป็นสถานะเริ่มต้น และโหนดใบ (leaf node) เป็นสถานะสุดท้าย ส่วนโหนดที่เหลือแทนสถานะของข้อมูลที่เป็นไปได้ เงื่อนไขการเข้าถึง choice เป็นตัวกำหนดความสัมพันธ์ระหว่าง choice การเปลี่ยนสถานะของ state เป็นตัวกำหนดเส้นทางที่เป็นไปได้ ระหว่าง category สถานะเริ่มต้นจะชี้ไปยังโหนดของ choice ใน category กลุ่มแรก และโหนดใบทุกอันชี้ไปยังสถานะสุดท้ายใน state macthine
- พิจารณาทิศทางการชี้ของโหนดหนึ่งไปยังโหนดถัดไปตามเงื่อนไขในการเข้าถึง choice แล้วส่งคุณลักษณะของโหนดนั้นไปยังโหนดถัดไป พร้อมทั้งพิจารณาว่าในแต่ละโหนดมีคุณลักษณะที่ขัดแย้งกันอยู่กี่กรณี ถ้าโหนดใดมีคุณลักษณะที่ขัดแย้งกันจะทำการแตกโหนดนั้นออกตามจำนวนกรณีที่ขัดแย้งกัน
- พิจารณาทิศทางการชี้ของโหนดหนึ่งไปยังโหนดถัดไปตามเงื่อนไขในการเข้าถึง choice ซึ่งเป็นการเปลี่ยนสถานะของโหนดนั้นไปยังโหนดถัดไป

ขั้นที่ 4 : การสร้างกรอบทดสอบจากเส้นทางใน state machine

- การเลือกเส้นทางจะพิจารณาโดยอาศัยเทคนิคของ Pairwise โดยพิจารณาระหว่าง category ทีละคู่ และตัดเส้นทางที่ซ้ำกันออกไป ดังนั้นเส้นทางที่ได้จะเป็นเส้นทางที่ไม่ซ้ำ และเป็นอิสระต่อกัน
- แต่ละเส้นทางเดินใน state machine แทนลักษณะของกรอบข้อมูลที่จะนำมาสร้างกรณีทดสอบ

ขั้นที่ 5 : ขั้นตอนการสร้างกรณีทดสอบ

เนื่องจากแต่ละกรอบทดสอบเป็นการกำหนดเงื่อนไขของการสร้างกรณีทดสอบ ดังนั้นแต่ละกรอบ

ทดสอบสามารถสร้างกรณีทดสอบได้อย่างน้อยหนึ่งกรณี

3. ตัวอย่างการสร้างกรอบทดสอบ

บทความนี้จะแสดงขั้นตอนการหากรณีทดสอบของโปรแกรมการค้นหารายชื่อหนังสือที่มีราคาน้อยกว่าหรือเท่ากับราคาที่กำหนด พร้อมทั้งแสดงรายชื่อหนังสือและราคาของหนังสือที่ทำการค้นหาได้ ซึ่งมีคุณลักษณะจำเพาะในการทำงานของโปรแกรมดังต่อไปนี้

Command: search

Syntax: search <title> <price>

Function:

คำสั่ง search ในที่นี้เป็นการค้นหารายชื่อหนังสือจาก keyword ซึ่งต้องมีราคาน้อยกว่าที่ผู้ใช้กำหนด พร้อมทั้งแสดงราคาของหนังสือที่ค้นหามาได้

<title> เป็น keyword ในการค้นหารายชื่อหนังสือ โดยที่ <title> ต้องขึ้นต้นและลงท้ายด้วย quote (“) และถ้าใน <title> จำเป็นต้องมี quote ก็ให้แทน quote (“) ด้วย quote 2 ตัวติดกัน (“”)

<price> เป็นราคาสูงสุดของการค้นหารายการหนังสือที่มีค่ามากกว่าศูนย์ และถ้าไม่มีการกำหนดค่า price ก็จะทำการค้นหาหนังสือทั้งหมดจาก keyword <title> ซึ่งมีราคาเท่าใดก็ได้

ตัวอย่าง:

- search “computer” 1000
  แสดงรายชื่อหนังสือพร้อมทั้งราคาของหนังสือ ที่มีคำว่า **computer** เป็นชื่อหัวข้อ และรายการหนังสือนี้เป็นรายชื่อหนังสือที่มีราคาน้อยกว่าหรือเท่ากับ 1000
- search “computer network” 1000
  แสดงรายชื่อหนังสือพร้อมทั้งราคาของหนังสือที่มีคำว่า computer network เป็นชื่อหัวข้อ และรายการหนังสือนี้เป็นรายชื่อหนังสือที่มีราคาน้อยกว่าหรือเท่ากับ 1000
- search “computer “” network”
  แสดงรายชื่อหนังสือที่มีคำว่า computer “ network เป็นชื่อหัวข้อ พร้อมทั้งราคาของหนังสือ

ขั้นที่ 1 : การวิเคราะห์ข้อกำหนดต่างๆ ในการทำงานของโปรแกรมที่จะทดสอบ

จากขอบเขตข้อมูลนำเข้า และสภาวะแวดล้อมของโปรแกรมการค้นหารายชื่อหนังสือ สิ่งแรกคือการวิเคราะห์ถึงสิ่งที่จะส่งผลกระทบต่อการทำงานของโปรแกรมโดยพิจารณาจากข้อกำหนดการทำงานของโปรแกรมข้างต้น คือ **Syntax** และ **Function** การทำงานของโปรแกรม การค้นหารายชื่อหนังสือ จะได้พารามิเตอร์ 2 ตัว คือ <title> และ <price> ซึ่งสามารถแยกพิจารณาลักษณะของพารามิเตอร์ตาม **syntax** ได้ดังนี้

- <title> มีรูปแบบการเปิดและปิดของ quote อย่างไร
- ความยาวของ keyword ที่ทำการค้นหามีลักษณะอย่างไร
- ใน keyword มี quote อยู่หรือไม่
- ใน keyword มีช่องว่าง (blank) หรือไม่
- <price> มีค่าหรือไม่อย่างไร

และจาก **Function** จะได้สภาวะแวดล้อมที่เกี่ยวข้องคือเอาต์พุตโดยพิจารณาลักษณะของสภาวะแวดล้อมคือ

- การแสดงผลของการหารายชื่อหนังสือ

หลังจากนั้นจะวิเคราะห์ลักษณะเฉพาะของพารามิเตอร์ต่างๆ ที่เป็นไปได้จากการทำงาน สามารถกำหนด choice ต่างๆ ของพารามิเตอร์ได้ดังนี้

Parameter of input domain:

<title> pattern:

Choice1<A1>: quote correct

Choice2<A2>: improperly quote

Choice3<A3>: not quote

<title> size:

Choice1<B1>:non-empty

Choice2<B2>: empty

Embedded quotes:

Choice1<C1>: no embedded quote

Choice2<C2>: one embedded quote

Choice3<C3>: several embedded quotes

Embedded blank:

Choice1<D1>: no embedded blank

Choice2<D2>: one embedded blank

Choice3<D3>: several embedded blanks

Boundary <price>:

Choice1<E1>: more than or equals to zero

Choice2<E2>: omitted

Choice3<E3>: less than zero

Parameter of environment

Display <title> and <price> to search for matches

Choice1<F1>: exactly one

Choice2<F2>: more than one

Choice3<F3>: none

ขั้นที่ 2 : การกำหนดความสัมพันธ์ของ choice

เพื่อไม่ให้เกิด combination ที่เป็นไปไม่ได้ จึงมีการกำหนดคุณลักษณะเบื้องต้นให้กับ choice เงื่อนไขในการเข้าถึง choice และกำกับหมายเหตุความผิดพลาดของ choice ให้กับพารามิเตอร์ ดังนี้

Parameters

<title> pattern:

Choice1<A1>: quote correct [attribute Quote]

Choice2<A2>: improperly quote [error]

Choice3<A3>: not quote [error]

<title> size:

Choice1<B1>:non-empty [if Quote] [attribute NonEmpty]

Choice2<B2>: empty [if Quote] [attribute Empty]

Embedded quotes:

Choice1<C1>: no embedded quote

Choice2<C2>: one embedded quote [if NonEmpty]

Choice3<C3>: several embedded quotes [if NonEmpty]

Embedded blank:

Choice1<D1>: no embedded blank

Choice2<D2>: one embedded blank [if NonEmpty]

Choice3<D3>: several embedded blanks [if NonEmpty]

Boundary <price>:

Choice1<E1>: more than or equals to zero

Choice2<E2>: omitted

Choice3<E3>: less than zero [error]

Environment

Display <title> and <price> to search for matches

Choice1<F1>: exactly one

Choice2<F2>: more than one

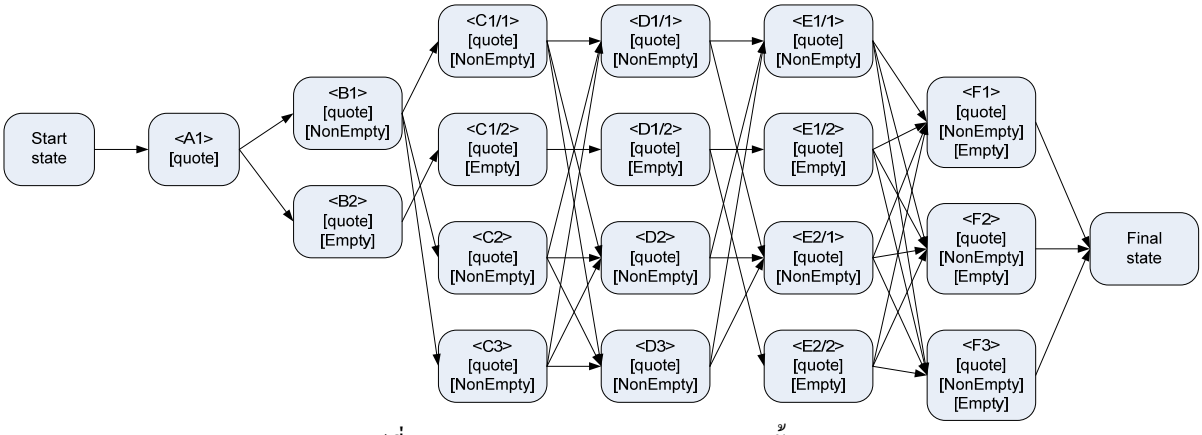
Choice3<F3>: none

จะเห็นว่า choice ที่ไม่ได้กำหนดประเภทของคุณลักษณะของเงื่อนไขในการเข้าถึงนั้นคือ choice ที่สามารถเข้าถึงได้ด้วยคุณลักษณะใดๆ ของประเภทคุณลักษณะนั้น

ขั้นที่ 3 : การสร้าง state machine จากความสัมพันธ์ของ choice

กำหนดโหนดแรกเป็นสถานะเริ่มต้น โหนดใบเป็นสถานะสุดท้าย และสร้างโหนดเพื่อแทน choice ของพารามิเตอร์ โดยกำหนดคุณลักษณะของ choice ให้กับแต่ละโหนด แล้วพิจารณาเงื่อนไขการเข้าถึง choice เพื่อสร้างเส้นทางเชื่อมแต่ละโหนด

ผลลัพธ์ของ state machine ที่ได้แสดงในรูปที่ 1



รูปที่ 1 state machine แสดงกรอบทดสอบทั้งหมด

ขั้นที่ 4 : การสร้างกรอบทดสอบจากเส้นทางใน state machine

แต่ละเส้นทางใน state machine แทนลักษณะของข้อมูลนำเข้าที่จะนำมาทดสอบ ซึ่งเส้นทางในการเปลี่ยนแต่ละสถานะนั้น ถ้ามีการเดินผ่านแล้ว ถือว่าเส้นทางนั้นได้ทำการทดสอบแล้ว เส้นทางเดินจาก state machine ที่ได้ในขั้นที่ 3 มีทั้งสิ้น 15 เส้นทาง คือ

1. {A1,B1,C1/1,D1/1,E1/1,F1}
2. {A1,B2,C1/2,D1/2,E1/2,F1}
3. {A1,B1,C2,D1/1,E2/1,F1}
4. {A1,B1,C3,D1/1,E1/1,F2}
5. {A1,B1,C1/1,D2,E1/1,F3}
6. {A1,B1,C1/1,D3,E1/1,F2}
7. {A1,B1,C2,D2,E2/1,F2}
8. {A1,B1,C2,D3,E2/1,F3}
9. {A1,B1,C3,D2,E2/1,F3}
10. {A1,B1,C3,D3,E1/1,F1}
11. {A1,B2,C1/2,D1/2,E2/2,F1}
12. {A1,B2,C1/2,D1/2,E1/2,F2}
13. {A1,B2,C1/2,D1/2,E1/2,F3}
14. {A1,B2,C1/2,D1/2,E2/2,F2}
15. {A1,B2,C1/2,D1/2,E2/2,F3}

จาก combination ของโปรแกรมค้นหารายชื่อหนังสือโดยตรง จะได้ลักษณะของข้อมูลนำเข้าเพื่อใช้ทดสอบ 1*2*3*3*2*3 หรือเท่ากับ 108 กรณี แต่ถ้าใช้ Category-Partition Method โดยกำหนด คุณลักษณะต่างๆ ให้กับพารามิเตอร์และสภาวะแวดล้อม แล้วนำมาสร้าง state machine เพื่อหาลักษณะของข้อมูลนำเข้า จะได้ข้อมูลนำเข้า 15 กรณี ทำให้จำนวนข้อมูลนำเข้าที่จะใช้ทดสอบลดลง 86.11%

4. ผลการทดลอง

ในการทดลองนี้ได้ทำการทดลองกับโปรแกรม 3 ชุดคือ ชุดที่ 1 ใช้ตัวอย่างโปรแกรมการค้นหารายชื่อหนังสือที่มีราคาน้อยกว่าหรือเท่ากับราคาที่กำหนด พร้อมทั้งแสดงรายชื่อหนังสือและราคาของหนังสือที่ทำการค้นหา ชุดที่ 2 เป็นโปรแกรมการค้นหารหัสของเรือที่ยังไม่มีคนจองของนักแล่นเรือ ชุดที่ 3 เป็นโปรแกรมตรวจสอบเวลาที่รถไฟออกจากสถานีหัวลำโพงไปยังจังหวัดต่างๆ

ผลการทดลองหาตัวแทนที่จะแทนลักษณะของข้อมูลนำเข้าที่จะนำมาทดสอบทั้ง 4 แบบ โดยไม่รวม [error] ในการสร้างกรณีทดสอบ เทียบกับวิธีรวมโดยตรงจากชุดทดลอง 3 ชุด สรุปได้ดังตารางที่ 1

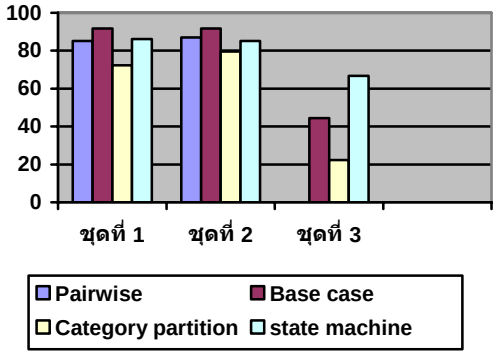
ตารางที่ 1 จำนวนกรณีทดสอบของแต่ละวิธี

วิธี	Pair wise	Base case	Category-partition	FSM	Combination
ชุดที่1	16	9	30	15	108
ชุดที่ 2	14	9	22	16	108
ชุดที่ 3	9	5	7	3	9

ตารางที่ 2 เปรียบเทียบอัตราการลดลงของตัวแทนลักษณะของข้อมูลนำเข้าที่นำมาทดสอบกับกลุ่มข้อมูลทั้งหมด (combination จากตารางที่ 1) และกราฟเปรียบเทียบในรูปที่ 2

ตารางที่ 2 เปอร์เซ็นต์กรณีทดสอบเทียบกับ combination

วิธี	Pairwise (%)	Base case (%)	Category-partition (%)	FSM (%)
ชุดที่1	85.19	91.67	72.22	86.11
ชุดที่ 2	87.04	91.67	79.63	85.19
ชุดที่ 3	0	44.44	22.22	66.67



รูปที่ 2 กราฟเปรียบเทียบอัตราการลดลงของตัวแทน

ตารางที่ 3 แสดงอัตราการลดลงของกรณีทดสอบของแต่ละวิธี

ตารางที่ 3 เปอร์เซ็นต์การลดลงของกรณีทดสอบ

วิธี	Pairwise (%)	Base case (%)	Category-partition (%)	FSM (%)
เปอร์เซ็นต์	57.41	75.93	58.02	79.32

5. บทสรุปและงานที่จะทำในอนาคต

วิธีจำแนกกรณีทดสอบโดยใช้ state machine ในการแบ่งตามลักษณะของขอบเขตข้อมูลนำเข้า ให้ผลไม่ด้อยไปกว่าวิธีอื่นๆ เช่น Base case [1], Category-Partition Method หรือดีกว่า Pariwise และวิธีรวมโดยตรง (combination) สำหรับกรณีพื้นฐานนั้นจะใช้ได้ดีกับโปรแกรมที่มีกรณีปกติหลักเพียงกรณีเดียว แต่จะไม่ครอบคลุมโปรแกรมที่มีกรณีปกติมากกว่าหนึ่งกรณี ดังเช่นในกรณีทดสอบชุดที่ 1 และ 2 ตัวเลขที่ได้เป็นตัวเลขที่ต่ำก็จริง แต่ไม่ครอบคลุมทุกกรณี เมื่อเทียบกับวิธีที่นำเสนอซึ่งแม้ว่าจะได้ตัวเลขที่สูงกว่า แต่ครอบคลุมกรณีทดสอบได้มากกว่า

ประโยชน์ที่ได้รับจากวิธีที่นำเสนอคือ (1) ความครอบคลุมเท่ากับ category-partition เพราะใช้หลักการแบ่งกลุ่มแบบเดียวกัน แต่ประยุกต์วิธี pairwise ในการสร้างกรอบทดลอง ทำให้ตัดกรณีที่ซ้ำกันออกไปได้ ผลลัพธ์ของจำนวนกรณีทดสอบจึงน้อยลง (2) วิธี pairwise และกรณีพื้นฐานจะตัด choice ที่ถูกเลือกแล้วออกโดยไม่พิจารณาความสัมพันธ์ของข้อมูล ทำให้กลุ่มที่ควรถูกทดสอบบางกลุ่มหายไป และบางกลุ่มอาจจะเป็นข้อมูลที่ไม่มีในความเป็นจริง แต่วิธีที่นำเสนอพิจารณาความสัมพันธ์ของข้อมูลนำเข้า เพื่อกำหนดเส้นทางของ state machine ยิ่งข้อมูลมีความสัมพันธ์ระหว่างกันมากเท่าใด ก็ยิ่งครอบคลุมได้มากเท่านั้น (3) วิธีที่นำเสนอสามารถแปลงเป็นตารางเมทริกซ์โดยตรง ซึ่งเหมาะสำหรับการนำไปประยุกต์ในเครื่องมือสร้างกรณีทดสอบแบบอัตโนมัติ

เนื่องจากตัวอย่างที่ใช้ในการทดลองยังมีขนาดเล็ก การวิเคราะห์เส้นทางด้วย state machine จึงไม่ประสบปัญหาแต่อย่างใด เพราะเส้นทางทั้งหมดแทนกรณีที่ต้องทดสอบทุกๆกรณี สิ่งที่จะต้องพิจารณาในอนาคตคือการประยุกต์วิธีที่นำเสนอกับโปรแกรมหรือซอฟต์แวร์ที่มีขอบเขตของข้อมูลนำเข้าขนาดใหญ่ซ้อนเหลื่อมกัน แต่ไม่ว่าขอบเขตของข้อมูลนำเข้าจะมีขนาดและลักษณะใด หากสามารถแบ่งกลุ่มข้อมูลได้โดยวิธีใดวิธีหนึ่ง (ซึ่งมีงานวิจัยเกี่ยวกับการแบ่งกลุ่มข้อมูลมากมาย) การสร้าง state machine เพื่อจำแนกกรณีทดสอบจึงไม่ใช่เรื่องยากในทางทฤษฎี สิ่งสำคัญที่ต้องคำนึงถึงในทางทฤษฎีคือการพิสูจน์ความพอเพียงของข้อมูล (data adequacy) [5] ที่ใช้เป็นตัวแทนในกรณีทดสอบขนาดเล็กที่สุดของโดเมนข้อมูลนำเข้า โดยพิจารณาจาก predicate, dependability theory, measurement เพื่อหาเกณฑ์สิ้นสุด

ของการกำหนดเงื่อนไขในการสร้าง state machine สำหรับกรณีทดสอบที่เป็นไปได้ทั้งหมด นั่นหมายถึงการนำเอาวิธีทาง white-box มาช่วยในการวิเคราะห์ ซึ่งในทางปฏิบัติ โดยเฉพาะจากมุมมองเชิงพาณิชย์ เป็นสิ่งที่ไม่คุ้มค่าการลงทุน ทั้งในแง่เงินลงทุนและเวลาที่ใช้ในการพัฒนาซอฟต์แวร์ให้ออกสู่ตลาดเร็วที่สุด การชั่งน้ำหนักระหว่างความเป็นไปได้เชิงวิจัยกับการประยุกต์ใช้จริงในอุตสาหกรรม จึงเป็นความท้าทายที่ต้องศึกษาวิจัยและพัฒนาให้บรรลุวัตถุประสงค์ของคำว่า “วิศวกรรมซอฟต์แวร์”

เอกสารอ้างอิง

[1] P. Ammnann and J. Offutt. “Using Formal Methods To Derive Test Frames In Category-Partition Testing”, Proceedings of the Ninth Annual Conference on Computer Assurance, 1994--COMPASS '94', Safety, Reliability, Fault Tolerance, Concurrency and Real Time, Security, June 27-July 1, 1994, pp. 69-79.

[2] Thomas J. Ostrand and Marc J. Balcer. “The Category-Partition Method for Specifying and Generating Functional Tests”, Communications of the ACM, Volume 31, Issue 6, June 1988, pp. 676-686.

[3] D. Hamlet. “Foundations of Software Testing: Dependability Theory”, ACM SIGSOFT Software Engineering Notes , Proceedings of the 2nd ACM SIGSOFT Symposium on Foundations of Software Engineering SIGSOFT '94, December 1994, Volume 19 Issue 5, pp. 128-139.

[4] P. Thevenod-Fosse and H. Waeselynck. “STATEMATE Applied to Statistical Software Testing”, ACM SIGSOFT Software Engineering Notes , Proceedings of the 1993 ACM SIGSOFT International Symposium on Software Testing and Analysis ISSTA '93, Volume 18 Issue 3, pp. 99-109.

[5] E.J. Weyuker. “The Evaluation of Program-based Software Test Data Adequacy Criteria”, Communications of the ACM, June 1988, Vol. 31, No. 6, pp. 668-675.

[6] H. Zhu, P.A.V. Hall, and J.H.R. May. “Software Unit Test Coverage and Adequacy”, ACM Computing Surveys, Vol. 29, No. 4, December 1997, pp. 366-427.

[7] Ian Sommerville. Software Engineering, 6th Edition, Addison-Wesley, 2001.

[8] Kuo-Chung Tai and Yu Lei, “A Test Generation Strategy for Pairwise Testing”, IEEE Transactions on Software Engineering, Vol. 28, No. 1, January 2002, pp. 1-2.