Chương 5

Kỹ thuật kiểm thử hộp đen

5.1 Tổng quát về kiểm thử hộp đen

Đối tượng được kiểm thử là 1 thành phần phần mềm (TPPM). TPPM có thể là 1 hàm chức năng, 1 module chức năng, 1 phân hệ chức năng… Nói chung, chiến lược kiểm thử hộp đen thích hợp cho mọi cấp độ kiểm thử từ kiểm thử đơn vị, kiểm thử tích hợp, kiểm thử hệ thống, kiếm thử độ chấp nhận của người dùng.

Kiểm thử hộp đen (black-box testing) là chiến lược kiểm thử TPPM dựa vào thông tin duy nhất là các đặc tả về yêu cầu chức năng của TPPM tương ứng.

Đây là chiến lược kiểm thử theo góc nhìn từ ngoài vào, các người tham gia kiểm thử hộp đen không cần có kiến thức nào về thông tin hiện thực TPPM cần kiểm thử (mã nguồn của thành phần phần mềm, thuật giải được dùng, các dữ liệu được xử lý…).

Qui trình kiểm thử hộp đen tổng quát gồm các bước chính :

- Phân tích đặc tả về các yêu cầu chức năng mà TPPM cần thực hiện.
- Dùng 1 kỹ thuật định nghĩa các testcase xác định (sẽ giới thiệu sau) để định nghĩa các testcase. Định nghĩa mỗi testcase là xác định 3 thông tin sau :
  - Giá trị dữ liệu nhập để TPPM xử lý (hoặc hợp lệ hoặc không hợp lệ).
  - Trạng thái của TPPM cần có để thực hiện testcase.
  - Giá trị dữ liệu xuất mà TPPM phải tạo được.
- Kiểm thử các testcase đã định nghĩa.
- So sánh kết quả thu được với kết quả kỳ vọng trong từng testcase, từ đó lập báo cáo về kết quả kiểm thử.

Vì chiến lược kiểm thử hộp đen thích hợp cho mọi mức độ kiểm thử nên nhiều người đã nghiên cứu tìm hiểu và đưa ra nhiều kỹ thuật kiểm thử khác nhau, chúng ta sẽ chọn ra 8 kỹ thuật có nhiều ưu điểm nhất và được dùng phổ biến nhất, đó là :

1. Kỹ thuật phân lớp tương đương (Equivalence Class Partitioning).
2. Kỹ thuật phân tích các giá trị biên (Boundary value analysis).
3. Kỹ thuật dùng các bảng quyết định (Decision Tables)
4. Kỹ thuật kiểm thử các bộ n thần kỳ (Pairwise)
5. Kỹ thuật dùng bảng chuyển trạng thái (State Transition)
6. Kỹ thật phân tích vùng miền (domain analysis)
7. Kỹ thuật dựa trên đặc tả Use Case (Use case)
8. Kỹ thuật dùng lược đồ quan hệ nhân quả (Cause-Effect Diagram)

5.2 Kỹ thuật phân lớp tương đương

Tinh thần của kỹ thuật này là cố gắng phân các testcase ra thành nhiều nhóm (họ) khác nhau : các testcase trong mỗi họ sẽ kích hoạt TPPM thực hiện cùng 1 hành vi. Mỗi nhóm testcase thỏa mãn tiêu chuẩn trên được gọi là 1 lớp tương đương, ta chỉ cần xác định 1 testcase đại diện cho nhóm và dùng testcase này để kiểm thử TPPM. Như vậy ta đã giảm rất nhiều testcase cần định nghĩa và kiểm thử, nhưng chất lượng kiểm thử không bị giảm sút bao nhiêu so với vét cạn. Điều này là dựa vào kỳ vọng rất hợp lý sau đây :

- Nếu 1 testcase trong lớp tương đương nào đó gây lỗi TPPM thì các testcase trong lớp này cũng sẽ gây lỗi như vậy.

- Nếu 1 testcase trong lớp tương đương nào đó không gây lỗi TPPM thì các testcase trong lớp này cũng sẽ không gây lỗi.

Vấn đề kế tiếp là có cần định nghĩa các lớp tương đương đại diện các testcase chứa các giá trị không hợp lệ theo đặc tả hay không ? Điều này phụ thuộc vào tinh thần kiểm thử :

- Nếu ta dùng tinh thần kiểm thử theo hợp đồng (Testing-by-Contract) thì không cần định nghĩa các lớp tương đương đại diện các testcase chứa các giá trị không hợp lệ theo đặc tả vì không cần thiết.
- Còn nếu ta dùng tinh thần kiểm thử phòng vệ (Defensive Testing), nghĩa là kiểm thử hoàn hảo, thì phải định nghĩa các lớp tương đương đại diện các testcase chứa các giá trị không hợp lệ theo đặc tả để xem TPPM phản ứng như thế nào với những testcase này.

Thí dụ ta cần kiểm thử 1 TPPM "quản lý nguồn nhân lực" với đặc tả chức năng như sau : mỗi lần nhận 1 hồ sơ xin việc, TPPM sẽ ra quyết định dựa vào tuổi ứng viên theo bảng sau :

Tuổi ứng viên	Kết quả
0-16	Không thuê
16-18	Thuê dạng bán thời gian
18-55	Thuê toàn thời gian
55-99	Không thuê

Lưu ý rằng bảng đặc tả chức năng phía trên có lỗi ở các giá trị đầu và/hoặc cuối trong từng luật, và giả sử chúng ta chưa phát hiện lỗi này. Chúng ta sẽ thấy bằng cách nào sẽ phát hiện dễ dàng lỗi này.

Phân tích đặc tả chức năng của TPPM cần kiểm thử của slide trước, ta thấy có 4 lớp tương đương, mỗi lớp chứa các testcase ứng với 1 chế độ xử lý của TPPM : không thuê vì quá trẻ, thuê dạng bán thời gian, thuê toàn thời gian, không thuê vì quá già.

Ứng với mỗi lớp tương đương, ta định nghĩa 1 testcase đại diện, thí dụ ta chọn 4 testcase sau :

1. Testcase 1 : {Input : 2 tuổi, Output : không thuê}
2. Testcase 2 : {Input : 17 tuổi, Output : thuê bán thời gian}
3. Testcase 3 : {Input : 35 tuổi, Output : thuê toàn thời gian}
4. Testcase 4 : {Input : 90 tuổi, Output : không thuê}

Trong thí dụ trên, thay vì phải kiểm thử vét cạn 100 testcase, ta chỉ kiểm thử 4 testcase → chí phí giảm rất lớn, nhưng chất lượng kiểm thử hy vọng không bị giảm sút là bao.

Tại sao chúng ta hy vọng chất lượng kiểm thử dùng lớp tương đương không giảm sút nhiều ? Hãy xét đoạn code mà những người lập trình bình thường sẽ viết khi xử lý TPPM cần kiểm thử của slide trước :

```
if (applicantAge >= 0 && applicantAge <=16) qd ="NO";
if (applicantAge >= 16 && applicantAge <=18) qd ="PART";
if (applicantAge >= 18 && applicantAge <=55) qd ="FULL";
if (applicantAge >= 55 && applicantAge <=99) qd ="NO";
```

Ở góc nhìn kiểm thử hộp trắng, nếu dùng 4 testcase đại diện của 4 lớp tương đương, ta sẽ kiểm thử được ở phủ cấp 3, cấp phủ rất tốt vì đã kiểm thử 100% các lệnh mã nguồn, 100% các nhánh quyết định.

Tuy nhiên nếu người lập trình hiện thực như sau (rất cá biệt vì đây là người lập trình rất yếu tay nghề) :

```
if (applicantAge == 0) qd ="NO";
…
if (applicantAge == 16) qd ="PART";
…
if (applicantAge == 53) qd ="FULL";
…
if (applicantAge == 99) qd ="NO";
```

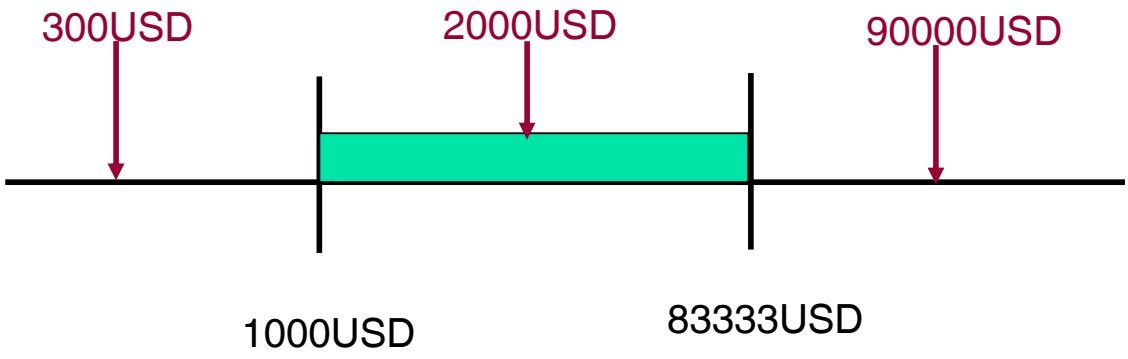
Thì nếu dùng 4 testcase đại diện của 4 lớp tương đương, ta mới kiểm thử được 4/100 lệnh mã nguồn của TPPM, mức độ phủ này chưa thể nói lên gì nhiều về TPPM!

Làm sao chọn testcase đại diện cho lớp tương đương ? Điều này phụ thuộc vào kiểu dữ liệu nhập. Ta hãy lần lượt xét 1 số kiểu dữ liệu nhập phổ biến.

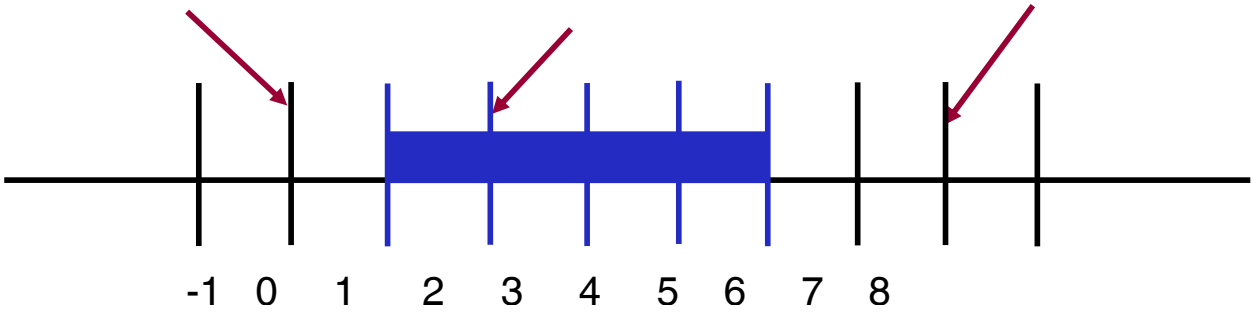
Thí dụ ta cần kiểm thử 1 TPPM "xét đơn cầm cố nhà" với đặc tả chức năng như sau : mỗi lần nhận 1 đơn xin cầm cố, TPPM sẽ ra quyết định chấp thuận nếu 4 điều kiện sau đều thỏa mãn :

1. Thu nhập hàng tháng của đương đơn nằm trong khoảng từ 1000$ đến 83333$.
2. số nhà xin cầm cố từ 1 đến 5.
3. Đương đơn phải là cá nhân, không được là hội, công ty hay người được ủy nhiệm (partnership, trust, corporation).
4. Loại nhà cầm cố phải là loại nhà cố định (single family, condo, townhouse), không xét loại nhà di động (treehouse, duplex, mobile home).

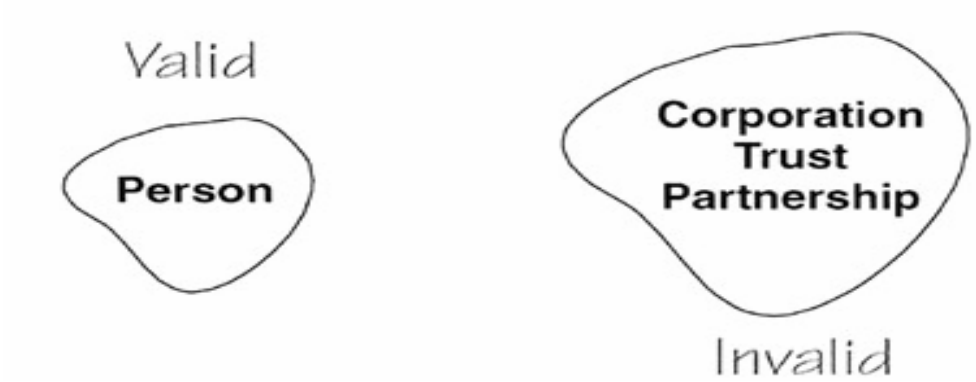
1. Nếu lớp tương đương được xác định bởi các dữ liệu nhập là số thực liên tục, thì ta chọn 1 testcase đại diện có giá trị nhập hợp lệ nằm trong khoảng liên tục các giá trị hợp lệ, và nếu muốn, 2 testcase miêu tả giá trị không hợp lệ nằm phía dưới và phía trên khoảng trị hợp lệ (số testcase cho mỗi lớp tương đương là từ 1 tới 3).



2. Nếu lớp tương đương được xác định bởi các dữ liệu nhập là số nguyên liên tục, trong trường hợp này ta chọn 1 testcase đại diện có giá trị nhập hợp lệ nằm trong khoảng liên tục các giá trị hợp lệ, và nếu muốn, 2 testcase miêu tả giá trị không hợp lệ nằm phía dưới và phía trên khoảng trị hợp lệ (số testcase cho mỗi lớp tương đương là từ 1 tới 3).

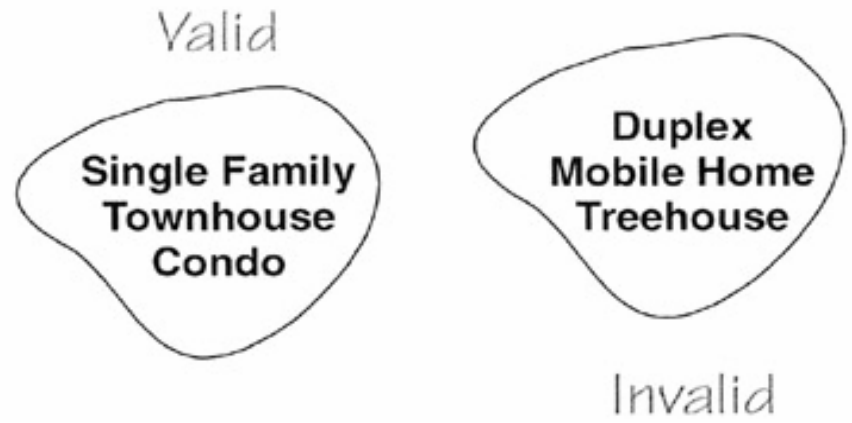


3. Nếu lớp tương đương được xác định bởi các dữ liệu dạng liệt kê rời rạc và không có mối quan hệ lẫn nhau gồm 1 trị hợp lệ và nhiều trị không hợp lệ. Trong trường hợp này ta chọn 1 testcase có giá trị nhập hợp lệ và nếu muốn, 2 testcase miêu tả 2 giá trị không hợp lệ nào đó, nhưng cho dù chọn 2 testcase nào cũng không đại diện tốt cho các trường hợp không hợp lệ còn lại (số testcase cho mỗi lớp tương đương là từ 1 tới 3).



4. Nếu lớp tương đương được xác định bởi các dữ liệu dạng liệt kê rời rạc và không có mối quan hệ lẫn nhau gồm n trị hợp lệ và m trị không hợp lệ. Trong trường hợp này ta chọn 1 testcase có giá trị nhập hợp lệ nào đó và nếu muốn, 2 testcase miêu tả 2 giá trị không hợp lệ nào đó, nhưng cho dù chọn các testcase nào cũng

không đại diện tốt cho các trường hợp hợp lệ và không hợp lệ còn lại (số testcase cho mỗi lớp tương đương là từ 1 tới 3).



Khi TPPM cần kiểm thử nhận nhiều dữ liệu nhập (thí dụ TPPM xét đơn cầm cố nhà ở slide trước có 4 loại dữ liệu nhập), ta định nghĩa các testcase độc lập cho các dữ liệu hay testcase dựa trên tổng hợp các dữ liệu nhập ?

Nếu định nghĩa các testcase độc lập trên từng loại dữ liệu nhập, số lượng testcase cần kiểm thử sẽ nhiều. Trong TPPM xét đơn cầm cố nhà, ta phải xử lý ít nhất là 3 testcase cho từng loại dữ liệu * 4 loại dữ liệu = 12 testcase.

Để giảm thiểu số lượng testcase nhưng vẫn đảm bảo chất lượng kiểm thử, người ta đề nghị chọn tescase như sau :

- 1 testcase cho tổ hợp các giá trị hợp lệ.
- n testcase cho tổ hợp các giá trị trong đó có 1 giá trị không hợp lệ, các giá trị còn lại hợp lệ, nên thay đổi các giá trị hợp lệ trong tổ hợp cho mỗi testcase.

Thí dụ TPPM xét đơn cầm cố nhà ở slide trước có 4 loại dữ liệu nhập), ta định nghĩa các testcase dựa trên tổ hợp các dữ liệu nhập như sau :

Thu nhập /tháng	Số lượng nhà	Đương đơn	Kiểu nhà	Kết quả
5.000	2	Person	Condo	Valid
100	1	Person	Single	Invalid

1.342	0	Person	Condo	Invalid
5.432	3	Corporation	Townhouse	Invalid
10.000	2	Person	Treehouse	Invalid

5.3 Kỹ thuật phân tích các giá trị ở biên

Kỹ thuật kiểm thử phân lớp tương đương là kỹ thuật cơ bản nhất, nó còn gợi ý chúng ta đến 1 kỹ thuật kiểm thử khác : phân tích các giá trị ở biên.

Kinh nghiệm trong cuộc sống đời thường cũng như trong lập trình các giải thuật lặp cho chúng ta biết rằng lỗi thường nằm ở biên (đầu hay cuối) của 1 khoảng liên tục nào đó (lớp tương đương). Do đó ta sẽ tập trung tạo các testcase ứng với những giá trị ở biên này.

Thí dụ xét đặc tả TPPM “quản lý nguồn nhân lực” ở slide 8, ta thấy đặc tả các luật đều bị lỗi ở các biên, thí dụ luật 1 qui định không thuê những người có tuổi từ 0 – 16, còn luật 2 qui định sẽ thuê bán thời gian những người từ 16-18 tuổi. Vậy người 16 tuổi được xử lý như thế nào bởi hệ thống ? Đã có nhập nhằng và mâu thuẩn trong các luật. Lỗi này do nắm bắt yêu cầu phần mềm sai.

Giả sử ta đã chỉnh sửa lại yêu cầu phần mềm như sau :

Tuổi ứng viên	Kết quả
0-15	Không thuê
16-17	Thuê dạng bán thời gian
18-54	Thuê toàn thời gian
55-99	Không thuê

Và đoạn code hiện thực sau :

```
if (0 < applicantAge && applicantAge < 15) kq ="NO";
if (16 < applicantAge && applicantAge <17) kq ="PART";
if (18 < applicantAge && applicantAge <54) kq ="FULL";
if (55 < applicantAge && applicantAge <99) kq ="NO";
```

Đoạn code hiện thực ở slide trước bị lỗi ở các gía trị biên (đúng ra là phải dùng điều kiện <= chứ không phải là <). Lỗi này thuộc về người hiện thực chương trình.

Cách đơn giản và hiệu quả để phát hiện lỗi ở slide trước là thanh tra mã nguồn (code inspection), mà ta sẽ đề cập trong chương 7. Ở đây ta chỉ trình bày kỹ thuật kiểm thử dựa trên các giá trị biên để phát hiện các lỗi này.

Ý tưởng của kỹ thuật kiểm thử dựa trên các trị biên là chỉ định nghĩa các testcase ứng với các giá trị ngay trên biên hay lân cận biên của từng lớp tương đương. Do đó kỹ thuật này chỉ thích hợp với các lớp tương đương xác định bởi các giá trị liên tục (số nguyên, số thực), chứ nó không thích hợp với lớp tương đương được xác định bởi các giá trị liệt kê mà không có mối quan hệ lẫn nhau.

Qui trình cụ thể để thực hiện kiểm thử dựa trên các giá trị ở biên :

- Nhận dạng các lớp tương đương dựa trên đặc tả về yêu cầu chức năng của TPPM.
- Nhận dạng 2 biên của mỗi lớp tương đương.
- Tạo các testcase cho mỗi biên của mỗi lớp tương đương :

  - 1 testcase cho giá trị biên.

  - 1 testcase ngay dưới biên.

  - 1 testcase ngay trên biên.
- Ý nghĩa ngay trên và ngay dưới biên phụ thuộc vào đơn vị đo lường cụ thể :

  - nếu là số nguyên, ngay trên và ngay dưới lệch biên 1 đơn vị.
  - nếu đơn vị tính là “$ và cent” thì ngay dưới của biên 5$ là 4.99$, ngay trên là 5.01$.

- nếu đơn vị là $ thì ngay dưới của 5$ là 4$, ngay trên 5$ là 6$.

Thí dụ dựa vào đặc tả của TPPM “quản lý nguồn nhân lực” :

Tuổi ứng viên	Kết quả
0-15	Không thuê
16-17	Thuê dạng bán thời gian
18-54	Thuê toàn thời gian
55-99	Không thuê

Ta sẽ định nghĩa các testcase tương ứng với các tuổi sau : {-1,0,1}, {14,15,16}, {15,16,17}, {16,17,18}, {17,18,19}, {53,54,55}, {54,55,56}, {98,99,100}.

Có nhiều testcase trùng nhau, nếu loại bỏ các testcase trùng nhau, ta còn : -1, 0, 1, 14, 15, 16, 17, 18, 19, 53, 54, 55, 56, 98, 99, 100 (16 testcase so với hàng trăm testcase nếu vẹt cạn).

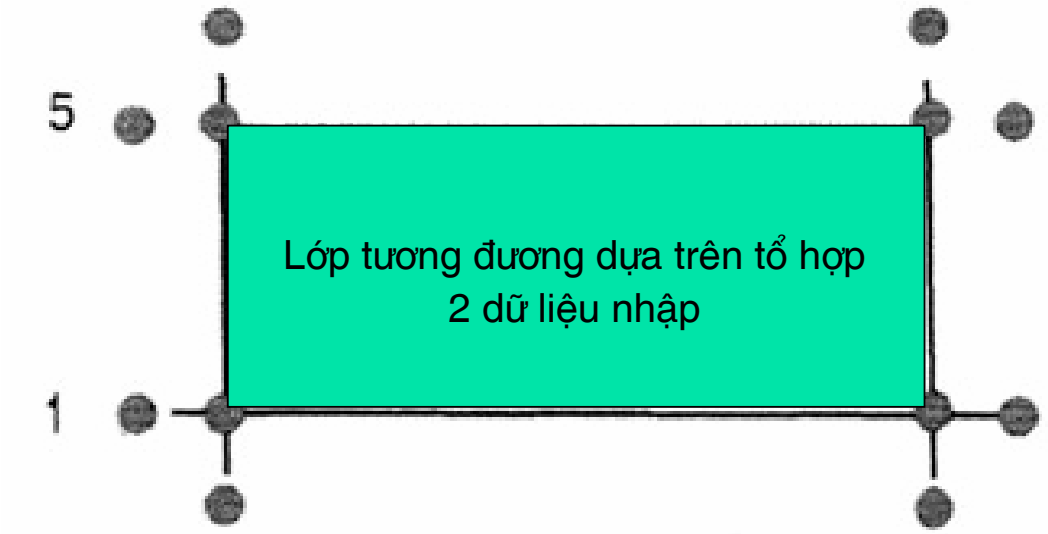
Khi TPPM cần kiểm thử nhận nhiều dữ liệu nhập (thí dụ TPPM xét đơn cầm cố nhà ở slide trước có 4 loại dữ liệu nhập), ta định nghĩa các testcase độc lập cho các dữ liệu hay testcase dựa trên tổng hợp các dữ liệu nhập ?

Nếu định nghĩa các testcase độc lập trên từng loại dữ liệu nhập, số lượng testcase cần kiểm thử sẽ nhiều. Trong TPPM xét đơn cầm cố nhà, ta phải xử lý ít nhất là 6 testcase cho từng loại dữ liệu * 4 loại dữ liệu = 24 testcase.

Để giảm thiểu số lượng testcase nhưng vẫn đảm bảo chất lượng kiểm thử, người ta đề nghị chọn tescase như sau :

- 1 số testcase cho các tổ hợp các giá trị biên.
- 1 số testcase cho các tổ hợp các giá trị ngay dưới và ngay trên biên.

TPPM xét đơn cầm cố nhà ở slide trước có 2 dữ liệu nhập liên tục là thu nhập hàng tháng và số lượng nhà. Tổng hợp 2 loại dữ liệu này theo góc nhìn đồ họa trực quan, ta thấy cần định nghĩa các testcase cho các trường hợp sau :



Thu nhập /tháng	Số lượng nhà	Kết quả	Diễn giải
$1,000	1	Valid	Min Thu nhập, min SoLN
$83,333	1	Valid	Max Thu nhập, min SoLN
$1,000	5	Valid	Min Thu nhập, max SoLN
$83,333	5	Valid	Max Thu nhập, max SoLN
$1,000	0	Invalid	Min Thu nhập, below min SoLN
$1,000	6	Invalid	Min Thu nhập, above max SoLN
$83,333	0	Invalid	Max Thu nhập, below min SoLN
$83,333	6	Invalid	Max Thu nhập, above max SoLN
$999	1	Invalid	Below min Thu nhập, min SoLN
$83,334	1	Invalid	Above max Thu nhập, min SoLN
$999	5	Invalid	Below min Thu nhập, max SoLN
$83,334	5	Invalid	Above max Thu nhập, max SoLN

5.4 Kỹ thuật dùng bảng quyết định (decision table)

Bảng quyết định là 1 công cụ rất hữu ích để đặc tả các yêu cầu phần mềm hoặc để đặc tả bảng thiết kế hệ thống phần mềm. Nó miêu tả các qui tắc nghiệp vụ phức tạp mà phần mềm phải thực hiện dưới dạng dễ đọc và dễ kiểm soát :

	Rule 1	Rule 2	…	Rule p
Conditions				
Condition-1				
…				
Condition-m				

Actions				
Action-1				
…				
Action-n				

Condition-1 tới Condition-m miêu tả m điều kiện dữ liệu nhập khác nhau có thể có. Action-1 tới Action-n miêu tả n hoạt động khác nhau mà hệ thống có thể thực hiện phụ thuộc vào tổ hợp điều kiện dữ liệu nhập nào. Mỗi cột miêu tả 1 luật cụ thể : tổ hợp điều kiện nhập cụ thể và các hoạt động cụ thể cần thực hiện.

Lưu ý các hoạt động cần thực hiện không phụ thuộc vào thứ tự các điều kiện nhập, nó chỉ phụ thuộc vào giá trị các điều kiện nhập.

Tương tự, các hoạt động cần thực hiện không phụ thuộc vào trạng thái hiện hành của TPPM, chúng cũng không phụ thuộc vào các điều kiện nhập đã có trước đó.

Chúng ta sẽ lấy 1 thí dụ cụ thể để làm rõ bảng quyết định. Giả sử TPPM cần kiểm thử là phân hệ chức năng nhỏ của công ty bảo hiểm : nó sẽ khuyến mãi cho những chủ xe (cũng là tài xế) nếu họ thỏa ít nhất 1 trong 2 điều kiện : đã lập gia đình / là sinh viên giỏi. Mỗi dữ liệu nhập là 1 giá trị luận lý, nên bảng quyết định chỉ cần có 4 cột, miêu tả 4 luật khác nhau :

	Rule 1	**Rule 2**	**Rule 3**	**Rule 4**
Conditions				
Married?	Yes	Yes	No	No
Good Student?	Yes	No	Yes	No
Actions				
Discount ($)	60	25	50	0

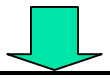
Qui trình cụ thể để thực hiện kiểm thử dùng bảng quyết định :

1. Tìm bảng quyết định từ đặc tả về yêu cầu chức năng của TPPM hay từ bảng thiết kế TPPM. Nếu chưa có thì xây dựng nó dựa vào đặc tả về yêu cầu chức năng hay dựa vào bảng thiết kế TPPM.

2. Từ bảng quyết định chuyển thành bảng các testcase trong đó mỗi cột miêu tả 1 luật được chuyển thành 1 đến n cột miêu tả các testcase tương ứng với luật đó :

   - nếu điều kiện nhập là trị luận lý thì mỗi cột luật được chuyển thành 1 cột testcase.
   - nếu điều kiện nhập là 1 lớp tương đương (nhiều giá trị liên tục) thì mỗi cột luật được chuyển thành nhiều testcase dựa trên kỹ thuật lớp tương đương hay kỹ thuật giá trị biên.

	Rule 1	**Rule 2**	**Rule 3**	**Rule 4**
Conditions				
Cond 1	Yes	Yes	No	No
Cond 2	Yes	No	Yes	No
Actions				
Action-1	60	25	50	0



	TC1	**TC2**	**TC3**	**TC4**
Conditions				
Married?	Yes	Yes	No	No
Good Student?	Yes	No	Yes	No
Actions				
Discount ($)	60	25	50	0

	Rule 1	**Rule 2**	**Rule 3**	**Rule 4**
Conditions				
Cond 1	0–1	1–10	10–100	100–1000
Cond 2	<5	5	6 or 7	>7
Actions				
Action-1	Do X	Do Y	Do X	Do Z

	TC1	TC2	TC3	TC4
Conditions				
Cond 1	0	5	50	500
Cond 2	3	5	7	10
Actions				
Action-1	Do X	Do Y	Do X	Do Z

5.5 Kỹ thuật kiểm thử các bộ n thần kỳ (pairwise)

Chúng ta hãy kiểm thử 1 website với đặc tả yêu cầu như sau :

1. Phải chạy tốt trên 8 trình duyệt khác nhau : Internet Explorer 5.0, 5.5, and 6.0, Netscape 6.0, 6.1, and 7.0, Mozilla 1.1, and Opera 7.
2. Phải chạy tốt ở 3 chế độ plug-ins : RealPlayer, MediaPlayer, none.
3. Phải chạy tốt trên 6 HĐH máy client : Windows 95, 98, ME, NT, 2000, and XP.
4. Phải chạy tốt trên 3 web server khác nhau : IIS, Apache, and WebLogic.
5. Phải chạy tốt trên 3 HĐH máy server : Windows NT, 2000, and Linux.

Như vậy để kiểm thử đầy đủ đặc tả yêu cầu, ta phải kiểm thử website trên 8*3*6*3*3 = 1296 cấu hình khác nhau.

Một thí dụ khác : hãy kiểm thử 1 hệ thống xử lý dữ liệu ngân hàng có đặc tả yêu cầu như sau :

1. Phải xử lý tốt 4 loại khách hàng là consumers, very important consumers, businesses, and non-profits.
2. Phải xử lý tốt 5 loại tài khoản : checking, savings, mortgages, consumer loans, and commercial loans.

3. Phải chạy tốt ở 6 bang khác nhau với chế độ khác nhau : California, Nevada, Utah, Idaho, Arizona, and New Mexico.

Như vậy để kiểm thử đầy đủ đặc tả yêu cầu, ta phải kiểm thử hệ thống xử lý dữ liệu ngân hàng trên 4*5*6 = 120 cấu hình khác nhau.

Một thí dụ khác : hãy kiểm thử 1 phần mềm hướng đối tượng có chi tiết thiết kế như sau : Đối tượng class A sẽ gởi thông điệp đến đối tượng class X dùng tham số là đối tượng class P.

1. Có 3 class con của A là B, C, D.
2. Có 4 class con của P là Q, R, S, T.
3. Có 2 class con của X là Y, Z.

Như vậy để kiểm thử đầy đủ đặc tả thiết kế, ta phải kiểm thử phần mềm trên với 4*5*3 = 60 trường hợp khác nhau.

Thông qua 3 thí dụ kiểm thử vừa giới thiệu, ta thấy số lượng kiểm thử là rất lớn, thường ta không có đủ tài nguyên về con người, về trang thiết bị và thời gian để kiểm thử hết tất cả. Buộc lòng ta phải tìm tập con các trường hợp cần kiểm thử, nhưng làm sao xác định được tập con cần kiểm thử thỏa điều kiện là số lượng càng ít càng tốt mà chất lượng kiểm thử không bị suy giảm nhiều.

Thật ra có nhiều chiến lược kiểm thử khác nhau :

1. Không kiểm thử bộ n nào cả (vì khiếp sợ số lượng quá lớn).
2. Kiểm thử tất cả bộ n, như vậy sẽ delay dự án quá lâu và làm mất thị trường.
3. Kiểm thử 1 hay 2 bộ n và hy vọng là đủ.
4. Chọn các bộ n đã kiểm thử rồi (cho project khác).
5. Chọn các bộ n dễ dàng tạo ra và kiểm thử nhất, mà không để ý chất lượng của chúng ra sao.

6. Tạo tất cả bộ n và chọn 1 ít bộ n đầu tiên trong danh sách.
7. Tạo tất cả bộ n và chọn 1 ít bộ n theo cơ chế ngẫu nhiên.
8. Chọn 1 tập con đủ nhỏ bộ n mà tạo được điều kỳ diệu là chất lượng kiểm thử không bị giảm sút. Bằng cách nào, nếu có ?

Có 2 phương pháp xác định được tập con các bộ n thần kỳ cần kiểm thử :

1. Dùng ma trận trực giao (orthogonal array).
2. Dùng tiện ích Allpairs.

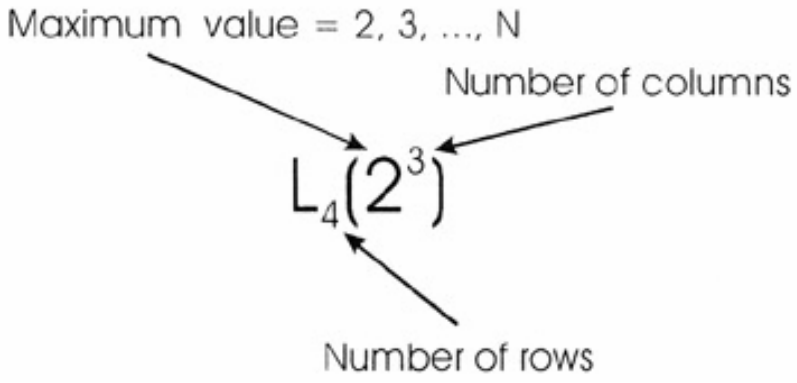
Ma trận trực giao là 1 bảng 2 chiều gồm n hàng * m cột các giá trị số nguyên với 1 đặc tính rất đặc biệt là : 2 cột bất kỳ trong ma trận đều chứa đúng các bộ đôi xác định trước.

Thí dụ, xét ma trận trực giao sau 4 hàng * 3 cột sau :

	1	2	3
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

Ta thấy 2 cột bất kỳ trong ma trận đều chứa đúng 4 bộ đôi sau đây (1,1), (1,2), (2,1), (2,2).

Ta miêu tả 1 ma trận trực giao như sau :



Số lượng các giá trị trong các cột không nhất thiết phải giống nhau, chúng ta có thể trộn nhiều loại cột chứa phạm vi giá trị khác nhau. Thí dụ ta dùng ký hiệu L18(21,37) để miêu tả ma trận trực giao có 18 hàng, mỗi hàng chứa 1 cột 2 giá trị và 7 cột chứa 3 giá trị.

Qui trình kiểm thử dùng các bộ n thần kỳ dựa vào ma trận trực giao gồm các bước chính :

1. Nhận dạng các biến dữ liệu. Thí dụ bài toán kiểm thử website trong các slide trước chứa 5 biến là Browser, Plug-in, Client operating system, Web Server, và Server operating system.
2. Xác định các giá trị cho từng biến. Thí dụ biến Browser có 8 giá trị lần lượt là : IE 5.0, IE 5.5, IE 6.0, Netscape 6.0, 6.1, 7.0, Mozilla 1.1, và Opera 7.
3. Tìm ma trận trực giao (trên mạng hay trong thư viện cá nhân) có mỗi cột cho mỗi biến, và số giá trị trong cột tương ứng với số giá trị của biến tương ứng. Thí dụ ta cần tìm ma trận trực giao Lx (816133) cho bài toán kiểm thử website.
4. Nếu không có ma trận thỏa mãn chính xác yêu cầu thì chọn ma trận trực giao có số cột tương đương, nhưng số lượng giá trị trong từng cột lớn hơn 1 chút cũng được. Cụ thể chưa ai tạo ra được ma trận trực giao có kích thước Lx (816133), do đó ta có thể tìm ma trận trực giao với kích thước L64(8243).
5. Ánh xạ bài toán kiểm thử vào ma trận trực giao bằng cách thay thế giá trị số bằng giá trị ngữ nghĩa. Thí dụ ta dùng cột 3 chứa 4 giá trị 1-4 để miêu tả biến Web Server, trong trường hợp này ta thế các giá trị 1-4 thành 4 giá trị ngữ nghĩa là IIS, Apache, WebLogic, Not used.
6. Xây dựng các test cases và kiểm thử chúng.

Thay vì dùng ma trận trực giao để nhận dạng các testcase, ta có thể dùng 1 thuật giải để sinh tự động các testcase.

Thí dụ James Bach đã cung cấp 1 tool tạo tự động tất cả các tổ hợp bộ n, bạn có thể download tool này từ địa chỉ http://www.satisfice.com và cái tiện ích này vào máy.

Để chuẩn bị dữ liệu nhập cho tiện ích, ta có thể Excel định nghĩa các biến dữ liệu cùng tập các giá trị nhập có thể có của từng biến theo dạng cột/hàng, sau đó lưu kết quả dạng văn bản thô *.txt. Chạy tiện ích theo dạng hàng lệnh như sau :

```
Allpairs input.txt > output.txt
```

Dùng 1 trình soạn trhảo văn bản mở file output.txt và xem kết quả.

5.6 Kết chương

Chương này đã giới thiệu những vấn đề tổng quát về kiểm thử hộp đen 1 TPPM.

Chúng ta đã giới thiệu chi tiết cụ thể về 4 kỹ thuật kiểm thử hộp đen được dùng phổ biến là kỹ thuật phân lớp tương đương, kỹ thuật phân tích các giá trị ở biên, kỹ thuật dùng bảng quyết định, và kỹ thuật kiểm thử các bộ n thần kỳ.

Ứng với mỗi kỹ thuật kiểm thử, chúng ta cũng đã giới thiệu 1 thí dụ cụ thể để demo qui trình thực hiện kỹ thuật kiểm thử tương ứng.