

CS 361 – Software Engineering

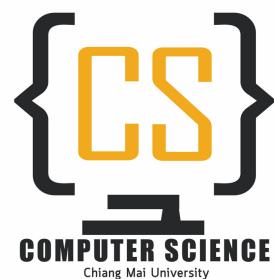
Software Testing I

ภาคต่อ

(อัพเดทจากสไลด์ก่อนหน้า)

Kamonphop Srisopha

Churee Techawut



Faculty of Science, Chiang Mai University
คณะวิทยาศาสตร์ มหาวิทยาลัยเชียงใหม่

How people react differently to software “failures”



- **User**
- **Developer**
- **Manager**
- **Tester**

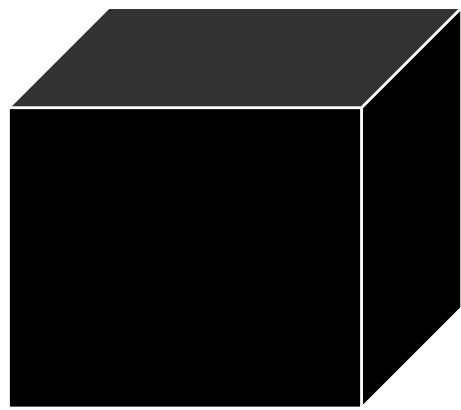


What Users Want!

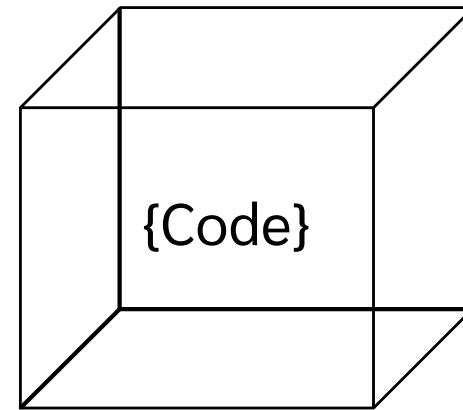


General Approach To Testing

How Should Test Cases Be Selected?

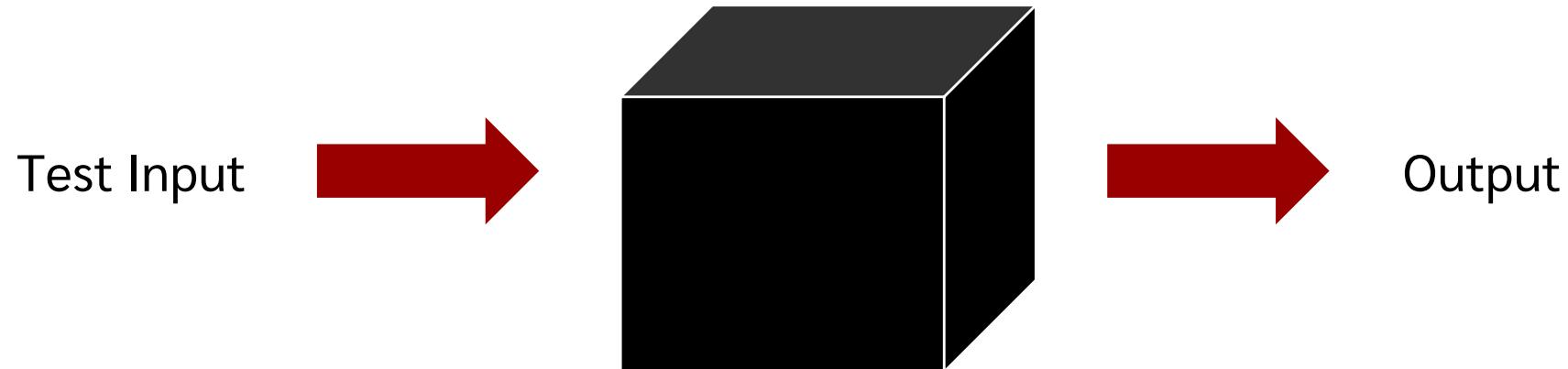


Black-box Testing
(Test Case สร้างขึ้นมาโดยอ้างอิง
specification)



White-box Testing
(Test Case สร้างขึ้นมาโดยอ้างอิง จาก
โครงสร้างภายในของ code
(structure elements of the code))

Black-box Testing



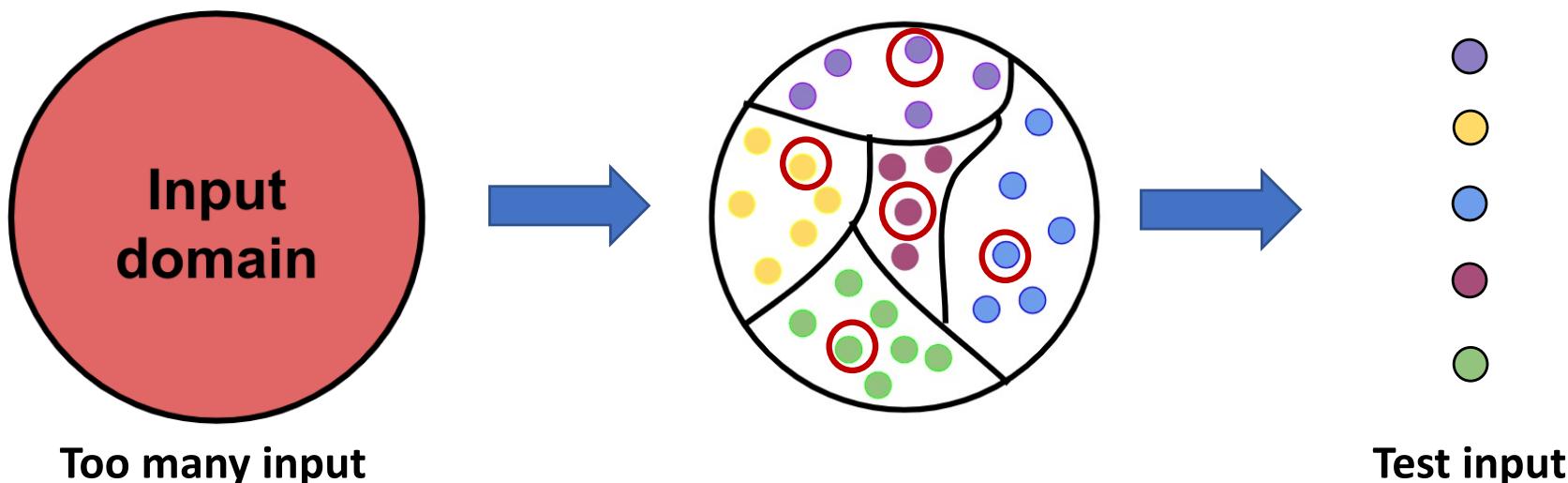
Black-box Testing
(Based on a **specification**)

- The program is considered a “black-box”
- Test cases are based on the **system specification**

Black-box Testing – An Introduction

Equivalence Partitioning

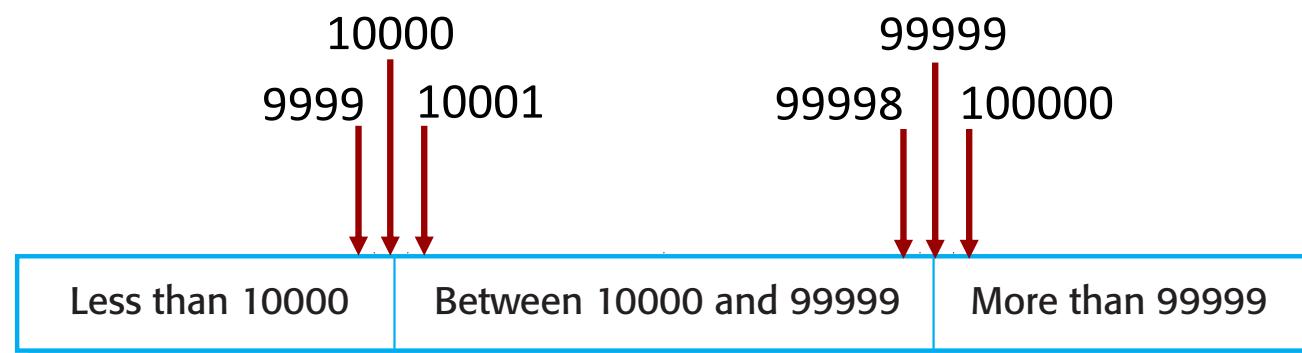
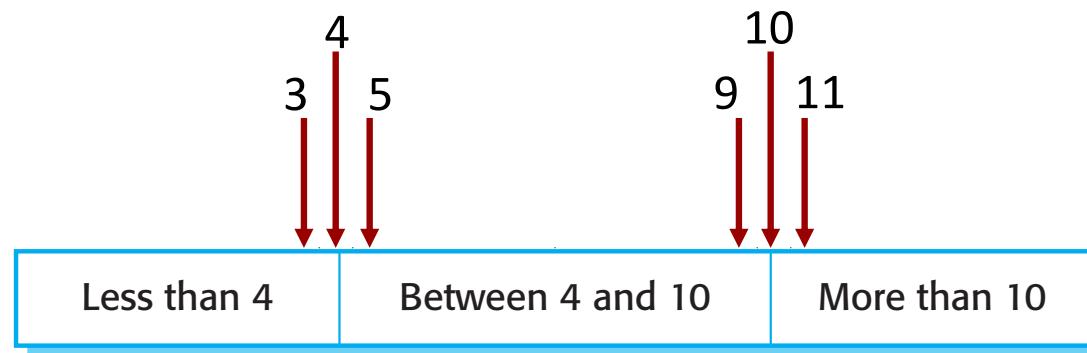
Idea: จาก specification แบ่ง input data ออกเป็นหลายๆ คลาสที่ในแต่ละคลาสมีสมาชิกที่มีลักษณะเหมือนกันและคิดว่าเมื่อนำไปทดสอบกับระบบแล้วได้ผลออกมาเหมือนกัน **ให้เลือก 1 สมาชิกในแต่ละคลาสมาระบุเป็น test case**



Black-box Testing – An Introduction

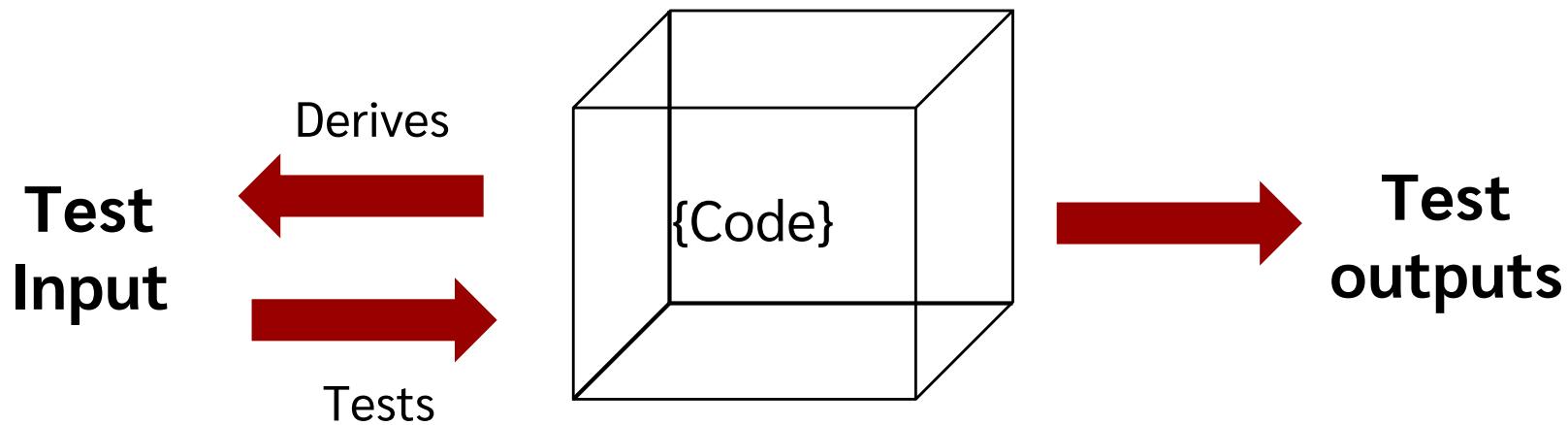
Boundary Conditions

Idea: ข้อผิดพลาดมักจะเกิดขึ้นใกล้ๆ กับขอบเขตของ class \Rightarrow ให้นำ test inputs ที่อยู่ใกล้ๆ กับขอบเขตมาทดสอบด้วย.



White-box Testing

Overview:



Test case เขียนขึ้นมาโดยอ้างอิงจาก โครงสร้างภายในของโค้ด
structure elements of the code

มักจะเรียกมันว่า “Structural Testing” เพื่อทดสอบ
 เช่น ว่าโค้ดทุกบรรทัดถูกรันหรือไม่, ทุก Boolean
 expression ถูกทดสอบหรือไม่ เป็นต้น

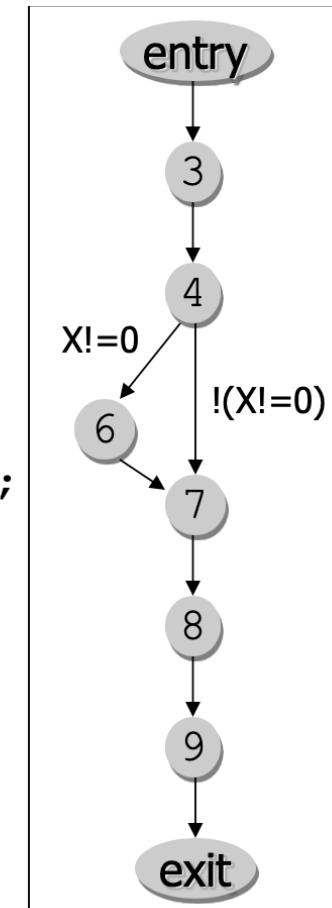
Why Structure Elements of the Code?

Idea: เงื่อนไขที่จำเป็นในการตรวจเจอ fault คือโค้ดที่เป็น fault นั้นต้องถูกรันใช้งาน (ถ้าไม่รันมัน ก็ไม่อาจรู้ได้ว่ามันเป็น fault)

วิธีการ

- แปลงโค้ดเป็นกราฟ
- หา test case ที่มาทดสอบ
โครงสร้างของกราฟ
เพื่อให้ได้ค่าครอบคลุม
(coverage) ที่ต้องการ

```
1. void main() {  
2.     float x, y;  
3.     read(x);  
4.     read(y);  
5.     if (x!=0)  
6.         x = x+10;  
7.     y = y/x;  
8.     write(x);  
9.     write(y);  
10. }
```



White-box Coverage Criteria

Statement Coverage

Basic Condition Coverage

Branch Coverage

Branch and Basic
Condition
Coverage

Modified
Condition/Decision
Coverage

Multiple Condition
Coverage

Statement Coverage (White-box Testing)

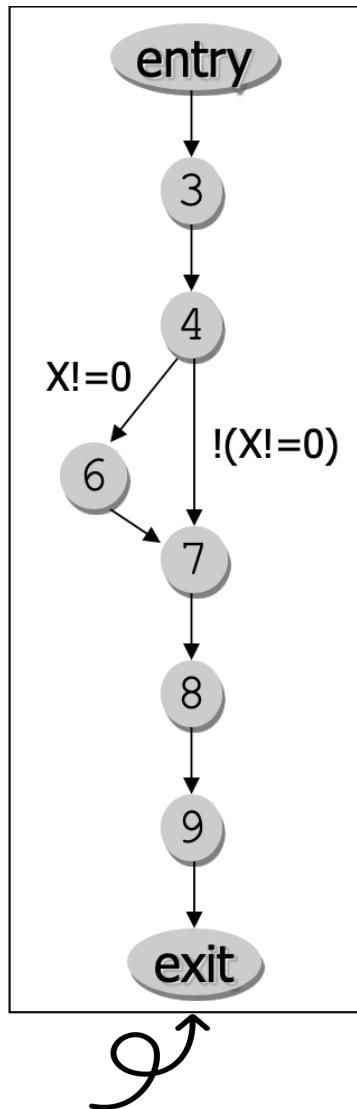
- **Statement Coverage** คือ ค่าการครอบคลุมของ statement (โค้ดในแต่ละบรรทัดหรือคำสั่ง) ที่ Test Case ได้รันมั่น
- คิดเป็น % โดยหาจาก

$$\text{Statement Coverage} = \frac{\text{จำนวน Statement ที่ถูกรัน}}{\text{จำนวน Statement ทั้งหมด}} \times 100$$

- **เป้าหมาย:** เรายังจะหา test suite ที่ทำให้ **ทุก Statement** ถูกรันอย่างน้อย 1 ครั้ง (เพื่อให้ได้ 100% Statement Coverage)

Statement Coverage Example

```
1. void main() {  
2.     float x, y;  
3.     read(x);  
4.     read(y);  
5.     if (x!=0)  
6.         x = x+10;  
7.     y = y/x;  
8.     write(x);  
9.     write(y);  
10. }
```



ในกราฟ Statement = Node (วงกลม)
โค้ดนี้มีทั้งหมด 8 Statements

Test Case อะไรบางที่เมื่อนำมา
ทดสอบกับ main() แล้วได้
100% Statement Coverage
(แต่ละ statement ถูกรันอย่าง
น้อย 1 ครั้ง)?

ข้อคิดที่ได้จากการตัวอย่าง

Branch Coverage (White-box Testing)

- **Branch Coverage** คือ ค่าการครอบคลุมของ Branch (เงื่อนไขใน if statement ถูกเรียกใช้งานทั้ง true และ false) ที่ Test Case ได้รันมัน
- คิดเป็น % โดยหาจาก

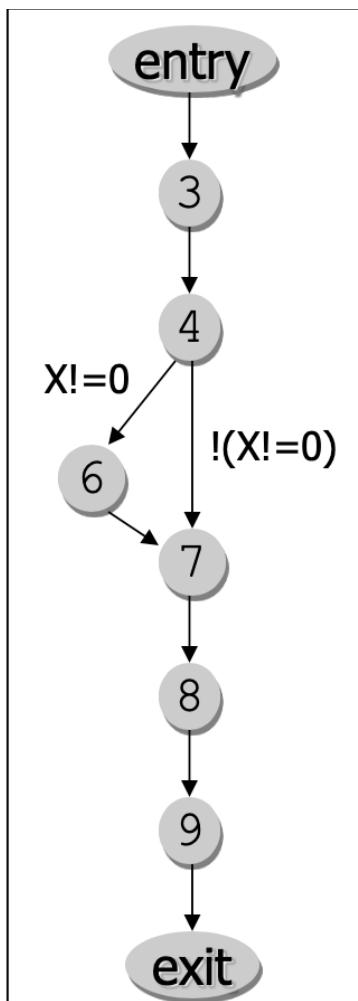
$$\text{Branch Coverage} = \frac{\text{จำนวน Branch ที่ถูกรัน}}{\text{จำนวน Branch ทั้งหมด}} \times 100$$



- **เป้าหมาย:** เาระบุว่า test suite ที่ทำให้ **ทุก Branch** ถูกรันอย่างน้อย 1 ครั้ง (เพื่อให้ได้ 100% Branch Coverage)

Branch Coverage Example #1

```
1. void main() {  
2.     float x, y;  
3.     read(x);  
4.     read(y);  
5.     if (x!=0)  
6.         x = x+10;  
7.     y = y/x;  
8.     write(x);  
9.     write(y);  
10. }
```

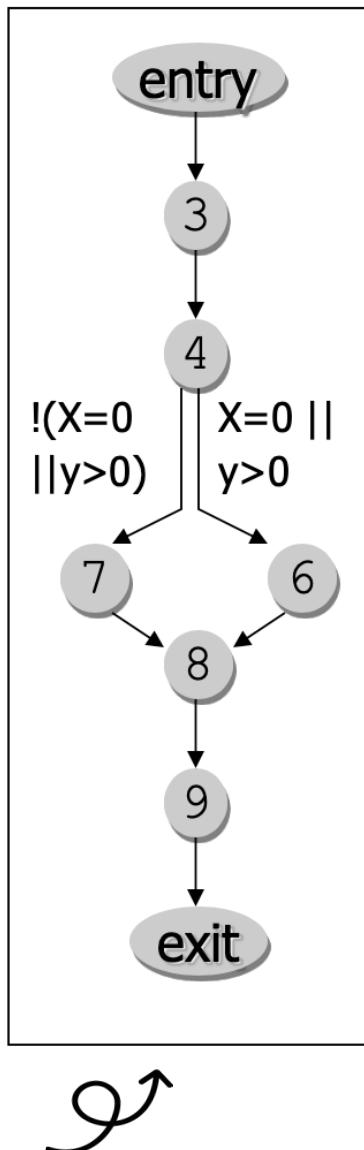


↗
Test Case อะไรบางที่เมื่อนำมา
ทดสอบกับ main() แล้วได้
100% Branch Coverage
(แต่ละ branch ถูกรันอย่างน้อย 1
ครั้ง)?

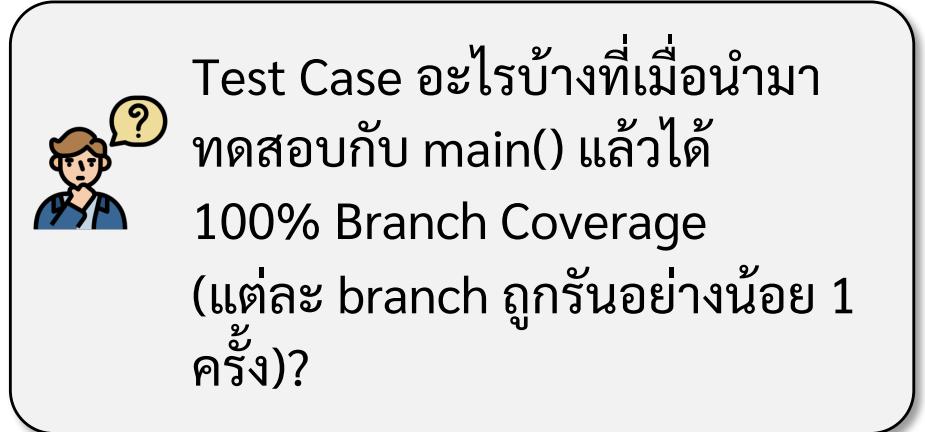
ในกราฟมีทั้งหมด 2 Branches 4-6-7 และ 4-7

Branch Coverage Example #2

```
1. void main() {  
2.     float x, y;  
3.     read(x);  
4.     read(y);  
5.     if (x==0) || (y>0)  
6.         y = y/x;  
7.     else x = y+2/x;  
8.     write(x);  
9.     write(y);  
10.}
```



ในกราฟมีทั้งหมด 2 Branches: 4-7-8 และ 4-6-8

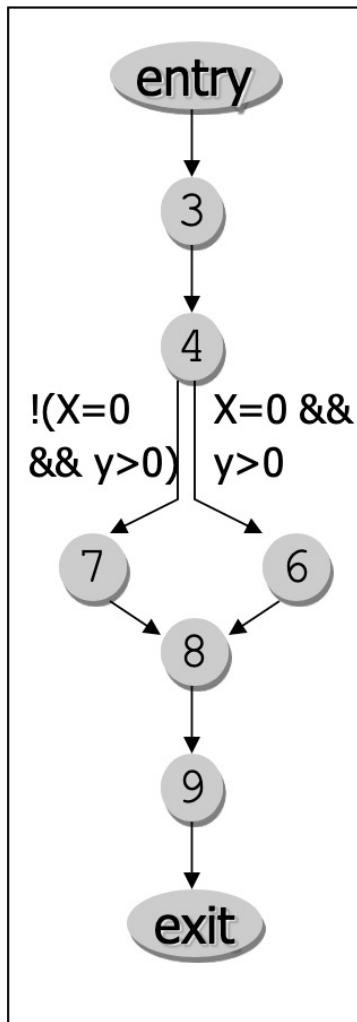


ข้อคิดที่ได้จากการตัวอย่าง

Basic Condition Coverage

เป้าหมาย: เรากำหนด test suite ที่ทำให้ทุก Boolean term (expression) ได้ผลเป็น True หรือ False อย่างน้อยอย่างละครั้ง

```
1. void main() {  
2.     float x, y;  
3.     read(x);  
4.     read(y);  
5.     if (x==0) && (y>0)  
6.         y = y/x;  
7.     else x = y+2;  
8.     write(x);  
9.     write(y);  
10.}
```



Test Case อะไรบางที่เมื่อนำมาทดสอบกับ main() แล้วได้ 100% Basic Condition Coverage

Test case #1:

- $X = 0$
- $Y = -2$

(T) && (F)

Test case #2:

- $X = 2$
- $Y = 3$

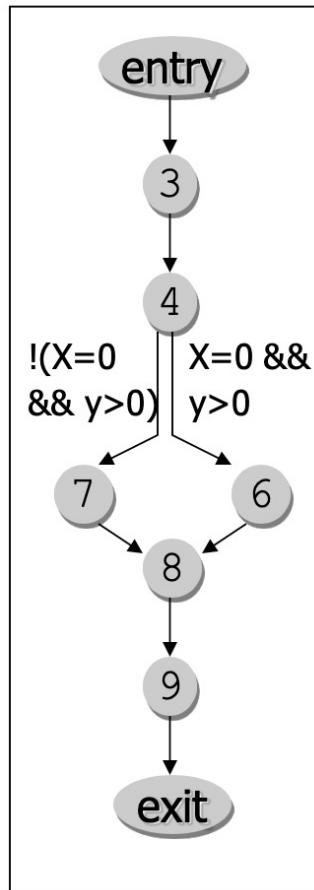
(F) && (T)

สอง Test cases ข้างบนนี้ทำให้เราได้ 100% Basic Condition Coverage เนื่องจาก แต่ละ expression เป็น T/F อย่างน้อย 1 ครั้ง **แต่เราไม่ได้ cover ทุก branch**

Branch and Basic Condition Coverage

เป้าหมาย: เรากำหนดให้ test suite ที่ทำให้ **ทุก Boolean term (expression)** ได้ผลเป็น True หรือ False อย่างน้อย 1 ครั้ง และ **ผล (decision)** รวมต้องออกมาเป็น True หรือ False อย่างน้อย 1 ครั้ง

```
1. void main() {  
2.     float x, y;  
3.     read(x);  
4.     read(y);  
5.     if (x==0) && (y>0)  
6.         y = y/x;  
7.     else x = y+2;  
8.     write(x);  
9.     write(y);  
10.}
```



Test case #1:

- X = 0
- Y = -2

Test case #2:

- X = 2
- Y = 3



100% Basic Condition Coverage



% Branch Coverage

100% Branch and Basic Condition, But...

If ((a || b) && c)

a	b	c	Outcome
T	F	T	T
F	T	F	F

- ถึงแม้ว่าในตัวอย่างข้างต้นใช้ 2 test cases ก็สามารถทำให้เราได้ **100% Branch and Basic Condition Coverage**
- เราจะเห็นว่าบาง basic condition ไม่ส่งผลโดยตรงต่อผลลัพธ์ เช่น ในกรณีแรก T F T ได้ T, condition b ไม่ได้ส่งผลต่อผลลัพธ์ (b จะเป็น T/F ก็ไม่มีผล)
- ในกรณีที่สอง F T F ในส่วนของ condition a และ b จริงๆ ไม่ได้ส่งผลต่อผลลัพธ์ เพราะ c เป็น False (a/b จะเป็น T/F ก็ไม่มีผล)
- นั่นหมายถึงบางเงื่อนไข (condition) ถูก **ปิดบัง** อยู่

Multiple Condition Coverage

- **เป้าหมาย:** เรากำหนดว่า test suite ที่ทดสอบทุก combination ของ basic condition
- จัดเป็น theoretical เพราะสิ่นเปลี่ยนสูงและเพราะไม่ practical ถ้าจะทดสอบทุก combination

```
if ( ( ( ( a || b ) && c ) || d ) && e )
```



ต้องใช้ทั้งหมดกี่ test case ถึงจะทำให้เราได้
100% Multiple Condition Coverage?

Modified Condition/Decision Coverage (MCDC)

เป้าหมาย: ให้หา test suite ที่แต่ละเงื่อนไขในการตัดสินใจ (Boolean term expression) ถูกแสดงว่าส่งผลต่อการตัดสินใจโดยรวม (decision) อย่างอิสระจากเงื่อนไขอื่น ๆ

- ใช้กรณีทดสอบน้อยกว่าและไม่สิ้นเปลืองเท่า MCC แต่มีประสิทธิภาพมากกว่า BBCC (เนื่องจาก BBCC ไม่ได้กำหนดว่าต้องหา test suite ที่แต่ละเงื่อนไขส่งผลโดยตรงต่อผลลัพธ์)
- เป็นระดับของ coverage ที่ถูกกำหนดไว้เป็นมาตรฐานขั้นต่ำในการทดสอบซอฟต์แวร์ที่เป็น safety-critical และ ซอฟต์แวร์ทางการบิน เช่น ของ US Federal Aviation Administration (FAA)

MCDC Example # 1

เป้าหมาย: ให้หา test suite ที่แต่ละเงื่อนไขในการตัดสินใจ (Boolean term expression) ถูกแสดงว่าส่งผลต่อการตัดสินใจโดยรวม (decision) อย่างอิสระจากเงื่อนไขอื่น ๆ

`if (a && b && c)`

Test case	a	b	c	outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False

MCDC Example # 2

If (a && (b || c))

	a	b	c	$a \wedge (b \vee c)$
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	F
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

Recap

If (a && (b || c))

- ต้องใช้กี่ test cases ที่จะทำให้เราได้ **100% Branch Coverage?**

- ต้องใช้กี่ test cases ที่จะทำให้เราได้ **100% Basic Condition Coverage?**

- ต้องใช้กี่ test cases ที่จะทำให้เราได้ **100% Branch and Basic Condition Coverage?**

- ต้องใช้กี่ test cases ที่จะทำให้เราได้ **100% Multiple Condition Coverage?**

- ต้องใช้กี่ test cases ที่จะทำให้เราได้ **100% Modified Condition/Decision Coverage?**

Subsumption Hierarchy

เกณฑ์ความครอบคลุมการทดสอบ A จะรวม เกณฑ์ความครอบคลุมการทดสอบ B อุปภายในตัวมันด้วย ก็ต่อเมื่อ หากใช้ test suite ทั้งหมดที่ได้ 100% A สำหรับโปรแกรม P ก็จะได้ 100% B สำหรับ P ด้วย
เรียกอีกอย่างว่า **Test Criteria A มีสมรรถภาพสูงกว่า B**

Test suite ที่ได้
100% coverage
crietria A

↓
subsumes

Test suite ที่ได้
100% coverage
crietria B

Multiple Condition
Coverage

Theoretical Criteria

Practical Criteria

Coverage Report Example (Java JSP):

Coverage Report - org.apache.jsp

Package	# Classes	Line Coverage	Branch Coverage
org.apache.jsp	9	56%  2035/3598 	28%  553/1932 
Classes in this Package		Line Coverage	Branch Coverage
AdminMenu.jsp		45%  160/349 	18%  36/200 
Default.jsp		62%  289/461 	29%  63/217 
DepsGrid.jsp		48%  182/373 	22%  44/199 
DepsRecord.jsp		58%  231/394 	34%  83/239 
EmpDetail.jsp		55%  248/447 	22%  50/221 
EmpsGrid.jsp		67%  299/443 	44%  101/225 
EmpsRecord.jsp		79%  412/518 	51%  135/264 
Header.jsp		24%  68/274 	3%  6/165 
Login.jsp		43%  146/339 	17%  35/202 

Class ต่างๆใน package

% coverage

Coverage Report Example (JS):

All files src/components

93.33% Statements 28/30 75% Branches 6/8 84.62% Functions 11/13 93.33% Lines 28/30

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File	Statements	Branches	Functions	Lines
AboutPage.js	50%	1/2	0%	50%
App.js	100%	3/3	100%	100%
FuelSavingsForm.js	100%	3/3	100%	100%
FuelSavingsPage.js	100%	7/7	100%	100%
FuelSavingsResults.js	100%	6/6	50%	100%
FuelSavingsTextInput.js	100%	4/4	100%	100%
HomePage.js	100%	2/2	100%	100%
NotFoundPage.js	50%	1/2	0%	50%
Root.js	100%	1/1	100%	100%

Using Cypress

Coverage Report Example (Python)

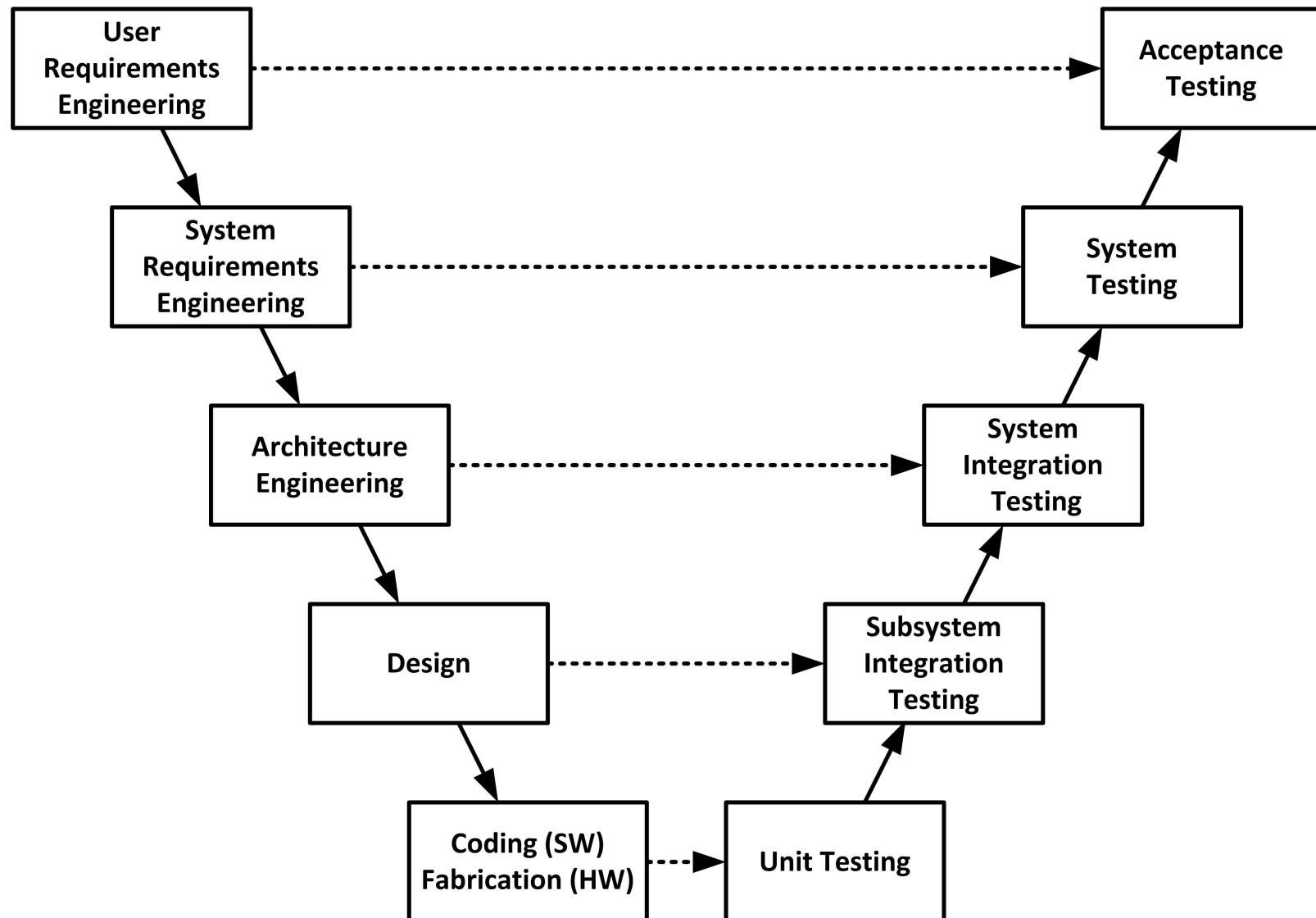
Cog coverage: 38.75%						
Module	statements	missing	excluded	branches	partial	coverage
cogapp/__init__.py	1	0	0	0	0	100.00%
cogapp/__main__.py	3	3	0	0	0	0.00%
cogapp/cogapp.py	500	224	1	210	30	49.01%
cogapp/makefiles.py	22	18	0	14	0	11.11%
cogapp/test_cogapp.py	845	591	2	24	1	29.57%
cogapp/test_makefiles.py	70	53	0	6	0	22.37%
cogapp/test_whiteutils.py	68	50	0	0	0	26.47%
cogapp/whiteutils.py	43	5	0	34	4	88.31%
Total	1552	944	3	288	35	38.75%

coverage.py v7.2.7, created at 2023-05-29 15:26 -0400

Using Python's Coverage Package

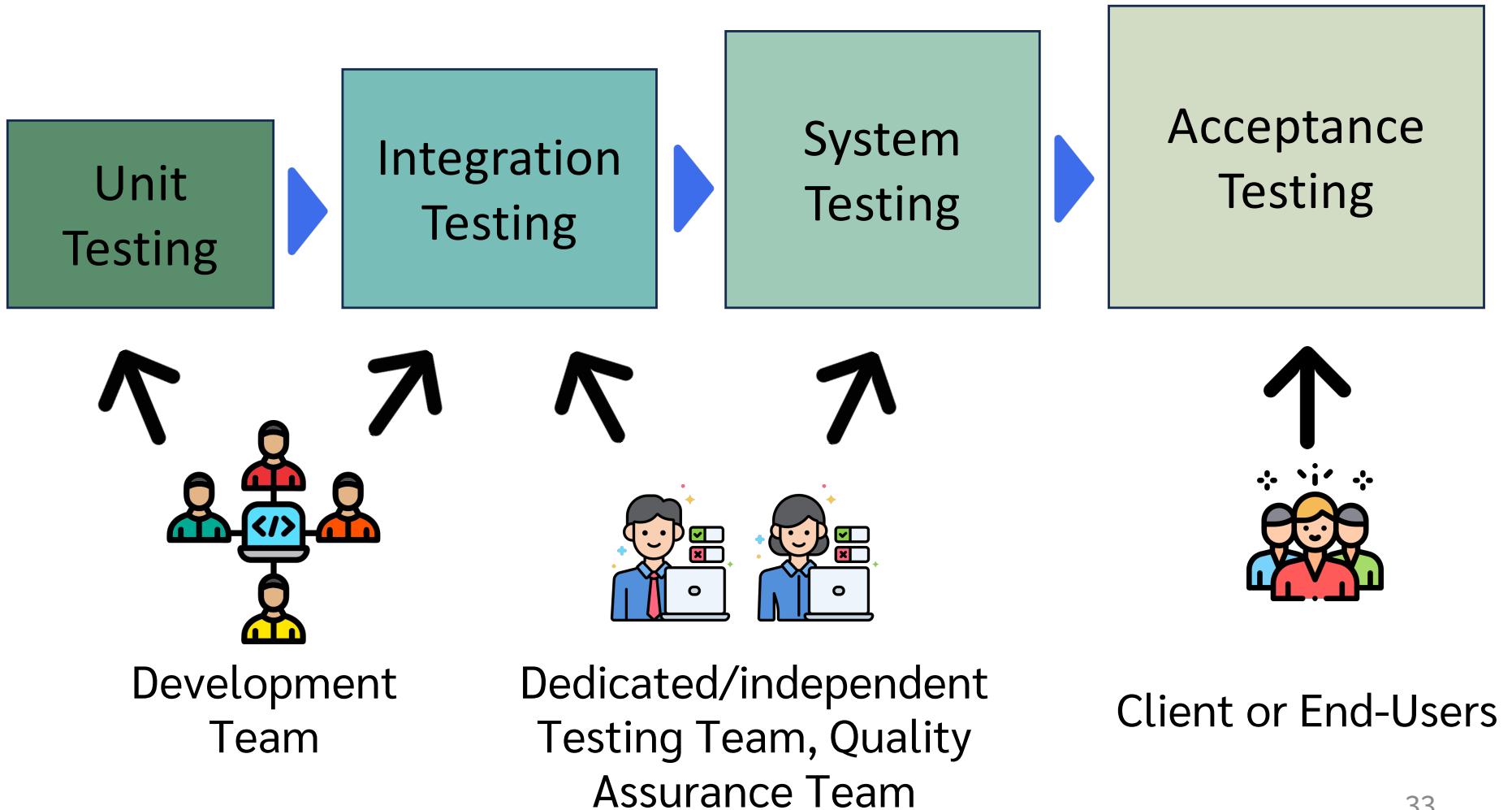
Levels of Testing

Levels of Testing (V-model)



Software Testing Levels

Who performs the tests?



Development Team Not Test Every Level?

เหตุผลหลักที่ Software Development Team มักจะมีทีม สำหรับทดสอบระบบโดยเฉพาะที่แยกต่างหาก เช่น ทีม Quality Assurance หรือ Independent Testers คือ



Bias and Familiarity

“เราเขียนโค้ดตรงนี้เอง ถูก/เวิร์คซั่ว ไม่ต้องทดสอบหรอก”

“มันเวิร์คบนเครื่องเราอ่ะ ไม่ต้องทดสอบหรอก เสียเวลา”

“เขียนโค้ดเองแล้วทดสอบเองแล้ว ผลคือผ่านหมดเลย”

Unit Testing

Does each unit work as specified?

- การทดสอบการทำงานของซอฟต์แวร์ในแต่ละหน่วยการทำงานที่เล็กที่สุดที่สามารถทดสอบได้ (มีการใช้งาน assert function และ สามารถเป็นได้ทั้ง White-box และ black-box)
- หน่วยในที่นี่อาจเป็น function, class, หรือ method หนึ่งๆ
- ผู้ทดสอบคือ **Development Team** ของระบบ เพราะเข้าใจแต่ละส่วนมากที่สุด
- อาจต้องมีการสร้าง **Mock** (Stub หรือ Driver) มาช่วยจำลองด้านต่างๆ เช่น ฐานข้อมูล, การเชื่อมต่อ network, การรับค่าข้อมูล เพื่อให้ unit ที่กำลังทดสอบสามารถทดสอบได้ (แต่ไม่ต้องไปต่อ service อื่นจริงๆ – no dependency)

Test Driver vs Stub

Driver และ **Stub** คือโมดูลหรือโปรแกรมจำลองบางหน่วยของระบบทำให้เราสามารถทดสอบโมดูลบางโมดูลได้หากมันต้องการค่าหรือข้อมูลอะไรบางอย่างหรือต้องถูกเรียกใช้งานบางอย่างจากโมดูลอื่น

Driver

โปรแกรมหรือโมดูลจำลองที่รับข้อมูลกรณีทดสอบแล้วส่งผ่านข้อมูลนั้นไปองค์ประกอบหรือโมดูลที่ถูกทดสอบในระดับที่ต่ำกว่า

เช่น ทดสอบ Stack ADT ที่มี method pop, push, และ print ตัว driver ก็คือโปรแกรมหรือโมดูลที่มารับค่าต่างๆ จาก user เพื่อให้ method พากนั้นถูกเรียกใช้งาน

Stub

โมดูลจำลองที่มี interface, API, method เหมือนโมดูลจริง แต่ไม่มีการประมวลผลข้อมูล อาจมีการตรวจ (verify) ข้อมูลอย่างง่าย ส่งค่าหรือให้ผลลัพธ์ด้วยค่าที่ถูกกำหนดไว้ เพื่อที่เราจะได้สามารถควบคุมพฤติกรรมของระบบส่วนด้านบนให้เป็นอย่างที่เราต้องการ เช่น โมดูลจำลองที่ให้ค่าเป็น true เมื่อเมื่อถูกเรียก

Integration Testing

Do the units work when combined?

- ทดสอบว่าเมื่อ unit ต่างๆมาร่วมกันมั้นยังสามารถทำงานได้ถูกต้องอยู่
- วิธีนี้จะทดสอบโปรแกรมโดยการเพิ่มจำนวนโมดูลขึ้นเรื่อยๆ
- จะเป็นการทดสอบโดยที่ unit ใน slide ก่อนหน้า มีการไปต่อ กับ Service อื่นๆที่มันจะต้องทำงานด้วยจริงๆ เช่น Network, Database, Other Service หรือ Party ต่างๆ
- จะเป็นการทดสอบด้วย Development Team หรือ ทีมทดสอบที่ไม่ใช่ ส่วนใน Dev เป็นทีมแยกออกจากต่างหาก
- รวมถึงการทดสอบตอนนำ increments ของแต่ละ sprint มารวมกัน

System Testing

Does the system work as a whole?
And how well?

- เป็นการทดสอบระบบโดยรวม (หรือทั้งระบบ) หลังจากที่ได้ทุกส่วนเชื่อมต่อกัน
กล้ายเป็นระบบแล้วหรือ integrate มาหมดแล้ว
- ระบบจะถูกทดสอบโดยทีมทดสอบที่ไม่ใช่ development team
- ทดสอบ functionality ที่ระบุใน specification (ตาม user story) หรือ requirements รวมทดสอบประสิทธิภาพของพาก non-functional requirements ร่วมด้วย เช่น
 - performance testing,
 - interface testing,
 - scalability testing
 - เป็นต้น

Acceptance Testing

Does the system meet the stakeholders' requirements and expectations?

- เป็นกระบวนการทดสอบในระดับท้ายสุดก่อนที่ระบบจะถูกปล่อยออกใช้งานจริง โดย client หรือ end-user
- Alpha (*in-house*) และ Beta (*few selected users* ไม่ใช่ dev) Testing ก็เป็นส่วนหนึ่ง
- กระบวนการคร่าวๆ คือ:
 - Dev จะต้องเตรียมข้อมูลทดสอบหรือ scenario ที่จะให้ผู้ใช้ทดสอบ
 - ผู้ใช้จะทำการทดสอบระบบตาม scenario โดยที่ dev ห้ามบอกผู้ใช้ว่าต้องกด ตรงนี้ไปตรงนี้เลือกตรงนี้
 - Dev ถาม feedback จากผู้ใช้หลังจากผู้ใช้ได้ทดสอบระบบ หรือสังเกตว่าเกิด ข้อผิดพลาดอะไรตรงไหนตอนผู้ใช้งานระบบบ้าง ให้บันทึกแล้วกลับไปแก้ไข
 - ถ้าข้อผิดพลาดมีน้อยหรืออยู่ในระดับที่ผู้ใช้พอใจ ก็สามารถปล่อยระบบออกใช้งานจริงได้

Summary of Levels of Testing

- **Unit Testing:** Ensures individual components/functions operate correctly.
- **Integration Testing:** Ensures combined components/modules function in harmony.
- **System Testing:** Ensures the entire application operates as expected.
- **Acceptance Testing:** Ensures the final product meets user or stakeholder expectations and requirements.



Questions ?

