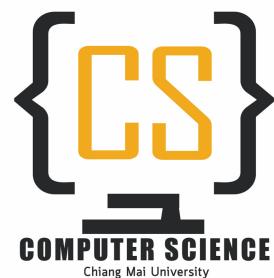


CS 361 – Software Engineering

Testing Techniques

Kamonphop Srisopha
Churee Techawut



Faculty of Science, Chiang Mai University
คณะวิทยาศาสตร์ มหาวิทยาลัยเชียงใหม่

Agenda

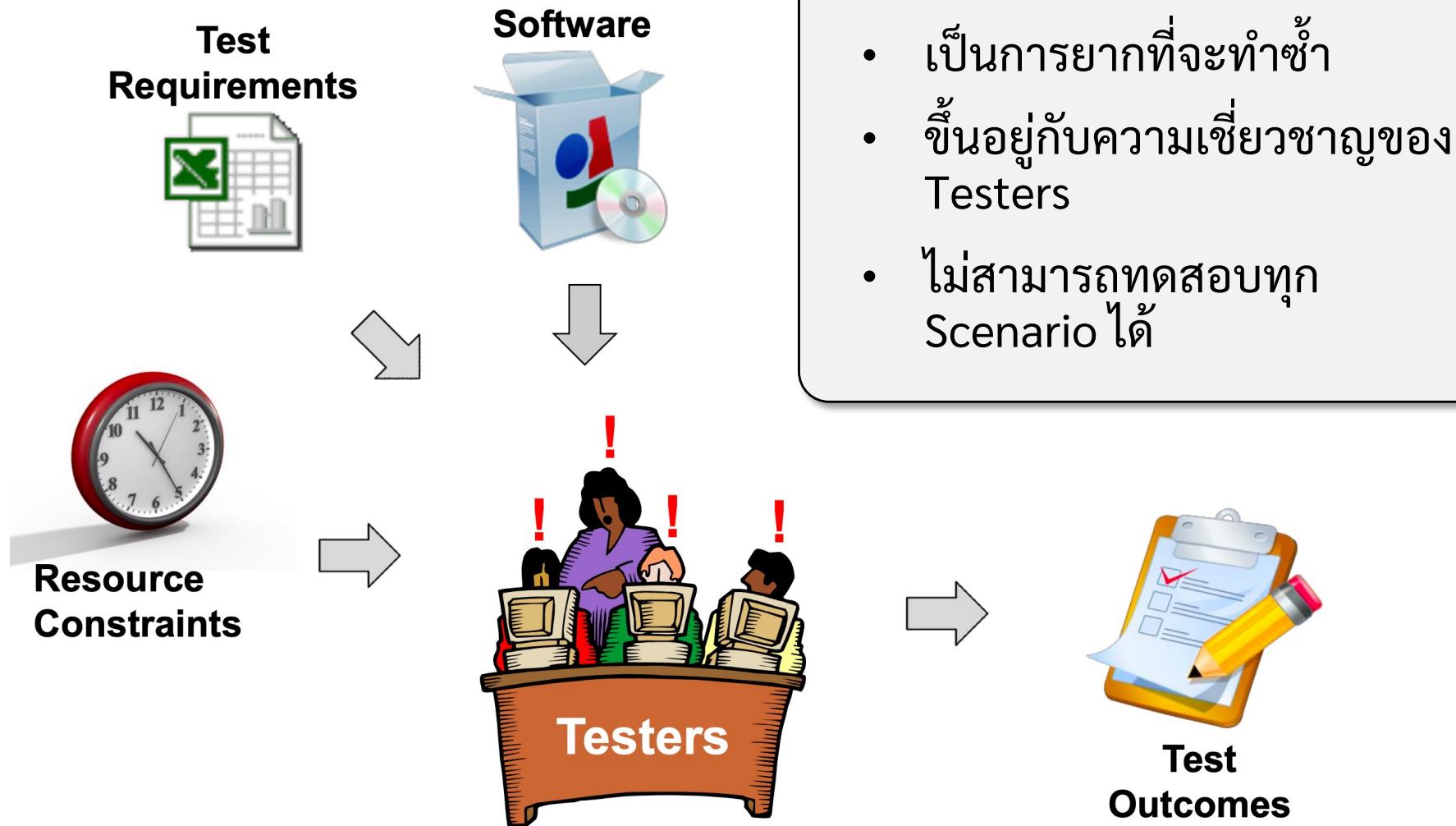
- Automated UI Testing with Selenium
- Fault Localization with Tarantula
- Testing your tests with Mutation Testing
- Behavior Driven Development with Cucumber

Automated UI Testing with Selenium

About Selenium

Selenium is a suite of tools for automating web browsers.

Manual Testing



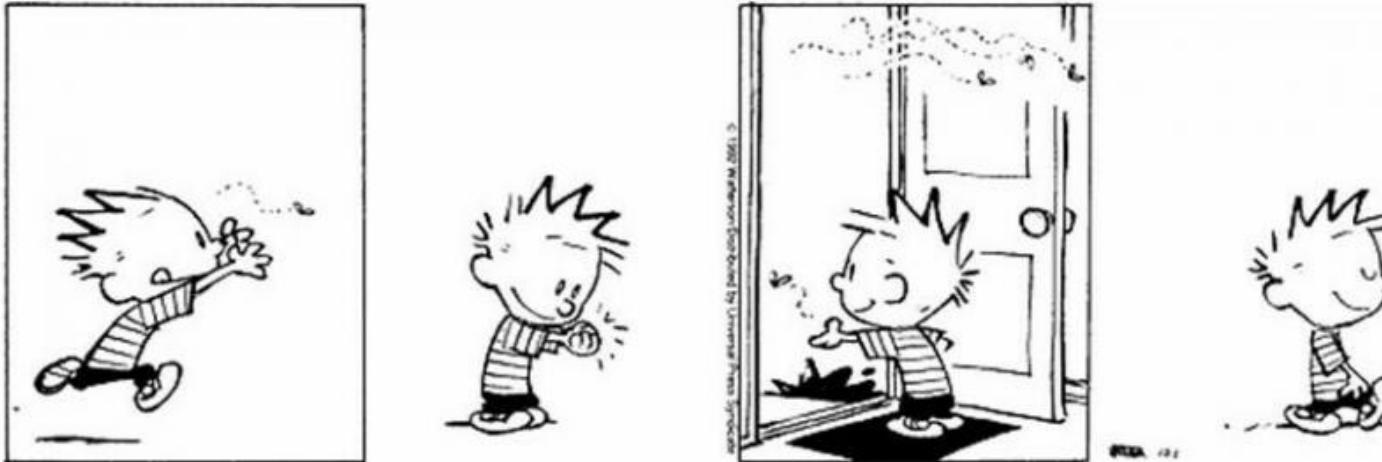
Automated Testing

- กระบวนการทดสอบซอฟต์แวร์อัตโนมัติ
- ในกระบวนการทดสอบแบบนี้ Tester จะเขียน test script (สคริปต์ทดสอบ) และรันมันผ่านเครื่องมือทดสอบ (testing tool) เพื่อทดสอบซอฟต์แวร์
- มักจะใช้เพื่อช่วยในการทดสอบแบบ Regression Testing
- ต้องอาศัยความเข้าใจในการสร้าง test script ภายใต้ Automation Framework ที่จะนำมาใช้
- ทำให้สามารถทดสอบช้าๆได้โดยไม่ใช้แรงงานและลดต้นทุน ด้านเวลา

Disadvantage of Automated Testing

- **Initial cost investment:** มี cost (แรงเงิน แรงเวลา) เบื้องต้นสูง (อาจไม่เหมาะสมกับทีมขนาดสั้นๆ หรือโครงการเล็ก)
- **Complexity:** เขียน Test script ที่สำหรับระบบที่ซับซ้อนอาจทำได้ยาก และบาง scenario อาจแปลงมาเป็น test script ไม่ได้
- **Maintenance:** มี cost ในการบำรุงรักษา (maintain) test scripts
 - **Intended Change** เช่นถ้า Feature ถูกแก้ไข หรือถูกเปลี่ยนไปแบบ ใจตามความต้องการที่เปลี่ยนไป, test script ที่สร้างมาก่อนหน้าอาจ report ว่าโค้ดตรงนั้น Fail (แต่จริงๆแล้ว มันต้อง pass) เพราะฉะนั้นก็ต้องเสียเวลามานั่งเขียน Test Script กันใหม่
 - ทดสอบ Usability และ/หรือ User Experience ได้ยาก

What is Regression Testing?



เป้าหมายของ **Regression testing** คือ เพื่อทดสอบว่าสิ่งที่เพิ่มเติม หรือแก้ไขเข้าไปในซอฟต์แวร์ไม่ได้ไปกระทบและก่อปัญหาให้กับส่วน อื่นที่เคยทดสอบไปแล้ว

ส่วนใหญ่ที่ทำกันคือ รัน test suite ทั้งหมดซ้ำอีกรอบ
 เพราะฉะนั้น automated testing จะมีประโยชน์มากในจุดนี้
(เพื่อให้สามารถทดสอบซ้ำได้โดยไม่มี cost มาก) **นักศึกษาเห็นปัญหาอะไรมั้ย?**

What is Selenium?



- Free (open-source) **Automated Testing Framework**
- ริเริ่มด้วย Jason Huggins (2004)
- ช่วยในการทดสอบ web app บน browsers หรือ platforms ต่างๆ
- ใช้เพื่อสร้าง automated test script
- Support หลายภาษา เช่น Java, Python, C# (.NET)
- Support หลาย Browser เช่น Chrome, Firefox, Safari
- Active community (updated ล่าสุด August 2023)
- Selenium Conference (ล่าสุดจัดที่ Chicago, Illinois เมื่อ March 28, 2023)
- Selenium เป็น suite of tools หรือประกอบด้วยหลาย tools เข้าด้วยกัน



Three Main Selenium Testing Tools



Selenium IDE



Selenium WebDriver



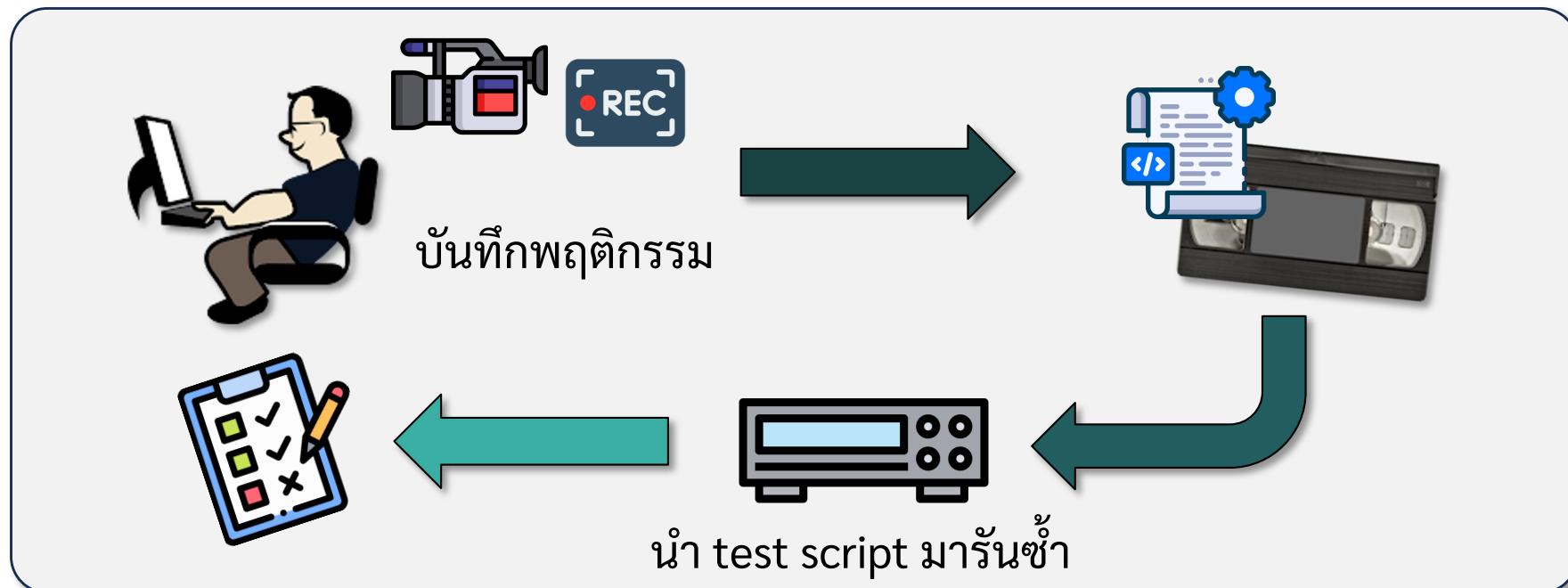
Selenium Grid

Selenium IDE – Record and Playback

บันทึกปฏิสัมพันธ์และลำดับเหตุการณ์ที่เกิดขึ้นระหว่างผู้ใช้กับระบบ (ผ่าน คีย์บอร์ดและเมาส์) แปลงให้อยู่ในรูปของสคริปต์ (script) และสามารถนำ script นั้นๆ มาเล่นซ้ำๆ ได้ในภายหลัง ผ่านเครื่องมือการทดสอบอัตโนมัติ

Record and Playback Process

สร้าง test script



Selenium IDE – Supported Browsers

The screenshot shows the Selenium IDE interface. At the top, there's a toolbar with various icons. Below it is a search bar and a URL field set to "https://www.google.com". The main area displays a test case named "TestCase1*" with five steps listed in a table:

	Command	Target	Value
1	open	/	
2	set window si ze	1474x1296	
3	edit content	css=body > div:nt h-child(1)	1 <p>1</p>
4	select frame	index=0	
5	click	css=.rr4y5c	

Below the table, there are four input fields: Command, Target, Value, and Description. At the bottom, there are tabs for "Log" (which is selected) and "Reference". A red box highlights the "Log" tab, and another red box highlights the first two steps in the table. A red arrow points from the text "มี feature ให้ playback ได้บน IDE" to the "Log" tab.

มี feature ให้ playback ได้บน IDE

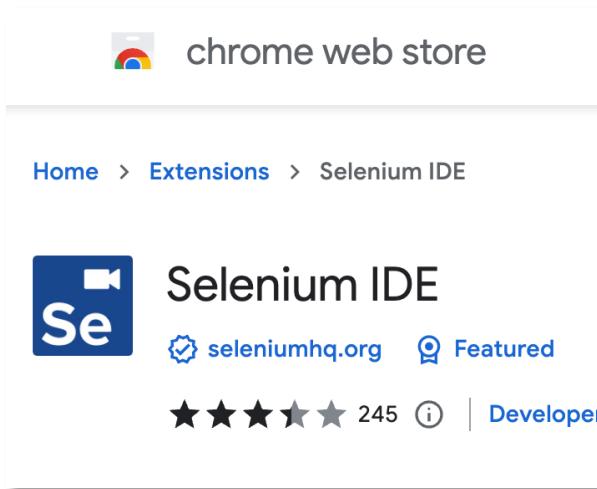
ระบบจะบันทึกว่า user กด ตรงไหนอะไรยังไงเป็น Step ๆ

- บันทึกปฏิสัมพันธ์และลำดับเหตุการณ์ที่เกิดขึ้นระหว่างผู้ใช้กับระบบ (Web app) ผ่านคีย์บอร์ดและเมาส์ และสามารถสร้าง test script ออกมาให้ได้
- เป็น Plugin ให้ใช้ได้ทั้งบน Chrome และ Firefox
- Test Script สามารถเล่นซ้ำได้บน Plugin หรือจะแก้ Test Script เองก็ได้
- สามารถ Export Script ที่เอาไปใช้กับ C# NUnit, Java Junit, JavaScript Mocha, Python pytest, Ruby Rspec เพื่อเล่นซ้ำได้

Selenium IDE Browser Plugins

Firefox

Chrome



 Firefox Browser
ADD-ONS

[Extensions](#) [Themes](#) [More... ▾](#)



Selenium IDE

by [Selenium](#)

⚠ This add-on is not actively monitored for security by Mozilla. Make sure you trust it before installing.

[Learn more](#)

Selenium IDE is an integrated development environment for Selenium tests. It is implemented as a Firefox extension, and allows you to record, edit, and debug tests.

[Add to Firefox](#)

Benefits of Selenium IDE

- ใช้งานไม่ยาก และง่ายต่อการ install (ผ่าน Browser Plugins)
- ไม่ต้องมีพื้นฐานด้านคอมพิวเตอร์/programming
- การทดสอบตรงกับพฤติกรรมของผู้ใช้อย่างมาก
- สามารถแก้ไข test script ได้ทันทีบน IDE
- มี feature ให้ playback test script ได้บน IDE
- playback feature สามารถบอกได้ว่า step ไหนผิด หรือไม่ตรงกับ ตอนที่เคยบันทึกไว้
- สามารถทำ assertion ได้บน IDE

Drawbacks of Selenium IDE

- บาง Browser ไม่สามารถใช้ตัว IDE ได้
- บาง action ไม่สามารถบันทึกแล้วมาเล่นซ้ำได้ เช่น captcha, two-factor authentication, เป็นต้น
- Capture space ใหญ่มาก (เป็นไปไม่ได้ที่จะบันทึกทุกสิ่งทุกอย่างทุกกรรมกิจ เพื่อมาทดสอบระบบ)
- สิ่งต่างๆที่ Complex ไม่สามารถบันทึกได้ (เช่น loop, conditionals - if statements)
- อาจมีปัญหากับ HTML elements บางอย่างเช่น iframe

Other Record and Playback Testing Tools

- Katalon – Web, API, mobile, Desktop (Windows) apps
- Espresso – Android apps
- XC(UI)Test – iOS apps
- Cypress.io – For JavaScript Web Apps



Selenium Webdriver

- สร้างขึ้นมาเพื่อช่วยอุด Drawbacks ของ Selenium IDE
- เป็น Library/API ที่ใช้ในการ Drive (ขับเคลื่อน) Web Browser ให้ทำตามที่เราต้องการ (ใช้ได้กับหลายค่าย เช่น Chrome, Firefox, Safari, Opera)
- Support หลาย programming language ทำให้เราสร้างโปรแกรมหรือโค้ดที่มาทดสอบ web app เราได้หลากหลายและซับซ้อนขึ้นได้ (เช่น Error/Exception Handling, Conditionals, Database Accessing, Logging เป็นต้น)
- เมื่อ Export Test Script จาก Selenium IDE ตัว script ก็ใช้ webdriver เป็นส่วนประกอบ

Selenium Webdriver

Coding Example

Using
Webdriver

```
1  from selenium import webdriver
2  from selenium.webdriver.common.by import By
3
4
5  def test_eight_components():
6      driver = webdriver.Chrome()
7
8      driver.get("https://www.selenium.dev/selenium/web/web-form.html")
9
10     title = driver.title
11     assert title == "Web form"
12
13     driver.implicitly_wait(0.5)
14
15     text_box = driver.find_element(by=By.NAME, value="my-text")
16     submit_button = driver.find_element(by=By.CSS_SELECTOR, value="button")
17
18     text_box.send_keys("Selenium")
19     submit_button.click()
20
21     message = driver.find_element(by=By.ID, value="message")
22     value = message.text
23     assert value == "Received!"
24
25     driver.quit()
```

Test Script From Selenium IDE - Pytest

```
1 # Generated by Selenium IDE
2 import pytest
3 import time
4 import json
5 from selenium import webdriver
6 from selenium.webdriver.common.by import By
7 from selenium.webdriver.common.action_chains import ActionChains
8 from selenium.webdriver.support import expected_conditions
9 from selenium.webdriver.support.wait import WebDriverWait
10 from selenium.webdriver.common.keys import Keys
11 from selenium.webdriver.common.desired_capabilities import DesiredCapabilities
12
13 class TestUntitled():
14     def setup_method(self, method):
15         self.driver = webdriver.Chrome()
16         self.vars = {}
17
18     def teardown_method(self, method):
19         self.driver.quit()
20
21     def test_untitled(self):
22         # Test name: Untitled
23         # Step # | name | target | value
24         # 1 | open | /selenium/web/web-form.html |
25         self.driver.get("https://www.selenium.dev/selenium/web/web-form.html")
26         # 2 | setWindowSize | 1474x1296 |
27         self.driver.set_window_size(1474, 1296)
28         # 3 | click | id=my-text-id |
29         self.driver.find_element(By.ID, "my-text-id").click()
30         # 4 | type | id=my-text-id | Selenium
31         self.driver.find_element(By.ID, "my-text-id").send_keys("Selenium")
32         # 5 | click | css=.d-flex |
33         self.driver.find_element(By.CSS_SELECTOR, ".d-flex").click()
34         # 6 | click | css=.btn |
35         self.driver.find_element(By.CSS_SELECTOR, ".btn").click()
```

import module
ที่เกี่ยวข้องและ
สร้าง function
setup และ
teardown ให้
อัตโนมัติ

แต่ต้องไปใส่
assert เอง

τετηνανός Φault

Fault Localization with Tarantula

Black box testing
In test case norm
reg specification
requirement
white box
In norm Tarantula
ROS Code



Fault Localization

เทคนิคต่างๆที่นักศึกษาได้เรียนมานั้นถึงตอนนี้ล้วนแต่เป็นเทคนิคในการคิด Test Cases (เช่น 2 techniques จาก Blackbox Testing และ หลายๆ Coverage Criteria จาก Whitebox testing)

แล้วเทคนิคการหา Fault ล่ะมีมั้ย?



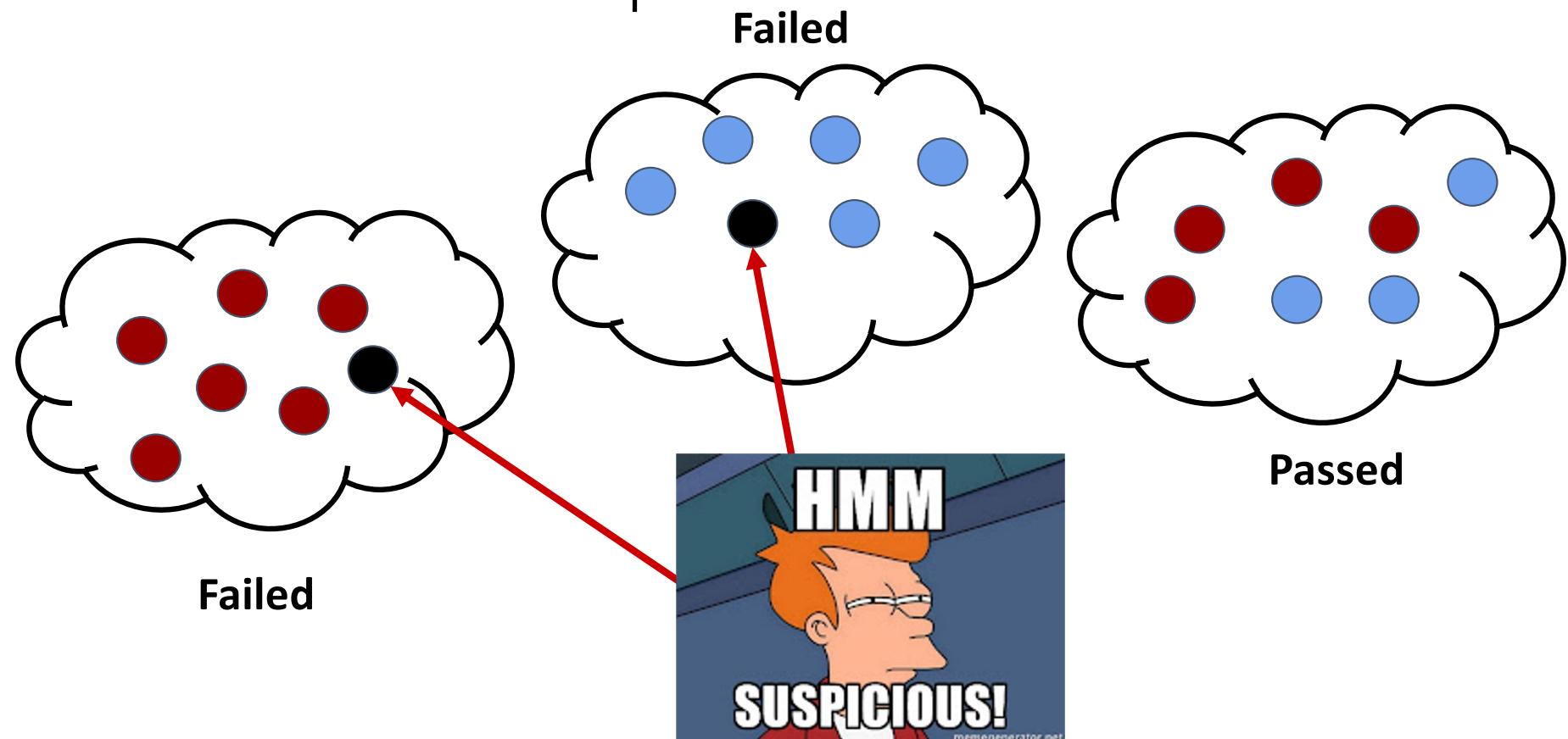
เทคนิคอะไรบ้างที่นักศึกษาใช้ในการหา Fault?



ข้อดีข้อเสียของเทคนิคเหล่านั้น?

The Tarantula* Technique

Key Idea: Statement ที่ถูกใช้งานโดย Test Case ที่ Fail ป່ອຍໆນີ້
ຄວາມນ່າສົກສໍຍວ່າຈະເປັນ Fault/Bug ມາກກວ່າ Statement ທີ່ຖືກໃຊ້ງານ
ໂດຍ Test Case ທີ່ Pass ປ່ອຍໆ



* Jones, James A., and Mary Jean Harrold. "Empirical evaluation of the tarantula automatic fault-localization technique." *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*. ACM, 2005.

Suspiciousness Level of a Statement

ใช้ข้อมูลจาก Statement ที่ถูกใช้งานและผลของการทดสอบของแต่ละ test case (passed/failed) เพื่อมาหาค่า Suspiciousness (ค่าความน่าสงสัย) ของแต่ละ Statement

$$suspiciousness(X) = \frac{\frac{failed(X)}{total\ failed}}{\frac{passed(X)}{total\ passed} + \frac{failed(X)}{total\ failed}}$$

- $failed(X)$ = จำนวน Test Case ที่ไม่ผ่านที่ใช้งาน Statement X
- $total\ failed$ = จำนวน Test Case ทั้งหมดที่ไม่ผ่าน
- $passed(X)$ = จำนวน Test Case ที่ผ่านที่ใช้งาน Statement X
- $total\ passed$ = จำนวน Test Case ทั้งหมดที่ผ่าน



- ค่า Suspiciousness
- ใกล้ 0 หมายถึง ไม่น่าสงสัยเลย
 - ใกล้ 1 หมายถึง น่าสงสัยมาก

Example:

สมมุติว่าใน Test Suite มีทั้งหมด 6 Test cases
มี 5 test cases ผ่าน (passed), 1 test case ไม่ผ่าน (failed)

ถ้า Statement X ถูกใช้งานโดย 4 test cases (3 ผ่าน, 1 ไม่ผ่าน)

$$\text{Suspiciousness (X)} = \frac{\frac{1}{1}}{\frac{3}{5} + \frac{1}{1}} = 0.63$$

$$\frac{\frac{1}{1}}{\frac{2}{3} + \frac{1}{1}} = ?$$

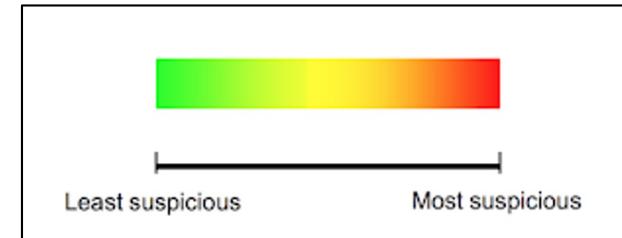


ถ้า Statement Y ถูกใช้งานโดย 3 test cases (2 ผ่าน, 1 ไม่ผ่าน)
ค่า Suspiciousness (Y) จะเป็นเท่าไหร่?

Coding Example

		Test Cases	
mid()	{	3,3,5	
int	x,y,z,m;	1,2,3	
1:	read("Enter 3 numbers:",x,y,z);	3,2,1	
2:	m = z;	5,5,5	
3:	if (y<z)	5,3,4	
4:	if (x<y)	2,1,3	
5:	m = y;		
6:	else if (x<z)		
7:	m = y; // *** bug ***		
8:	else		
9:	if (x>y)		
10:	m = y;		
11:	else if (x>z)		
12:	m = x;		
13:	print("Middle number is:",m);		
	}		Pass/Fail Status

Color-Coded Visualization

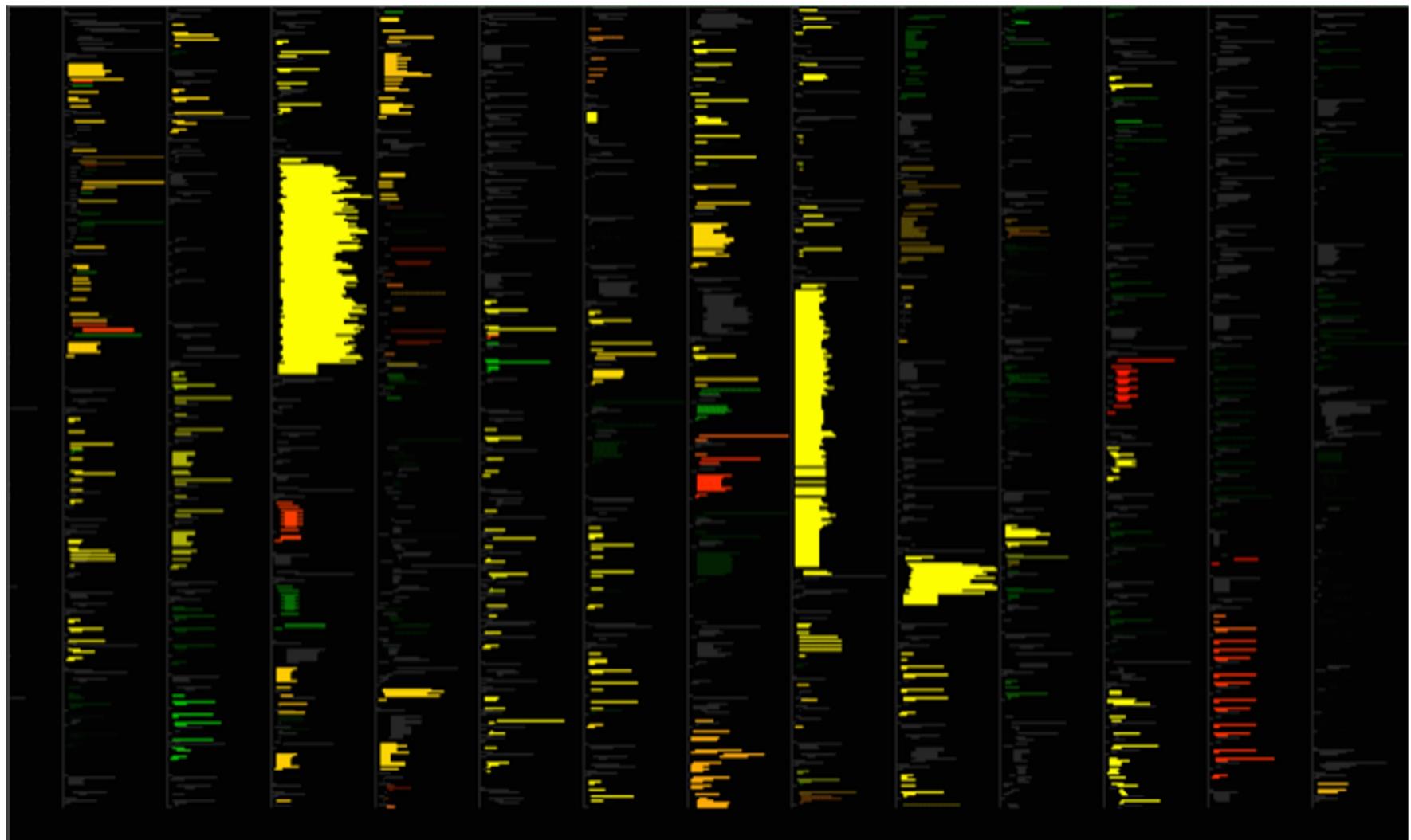


```

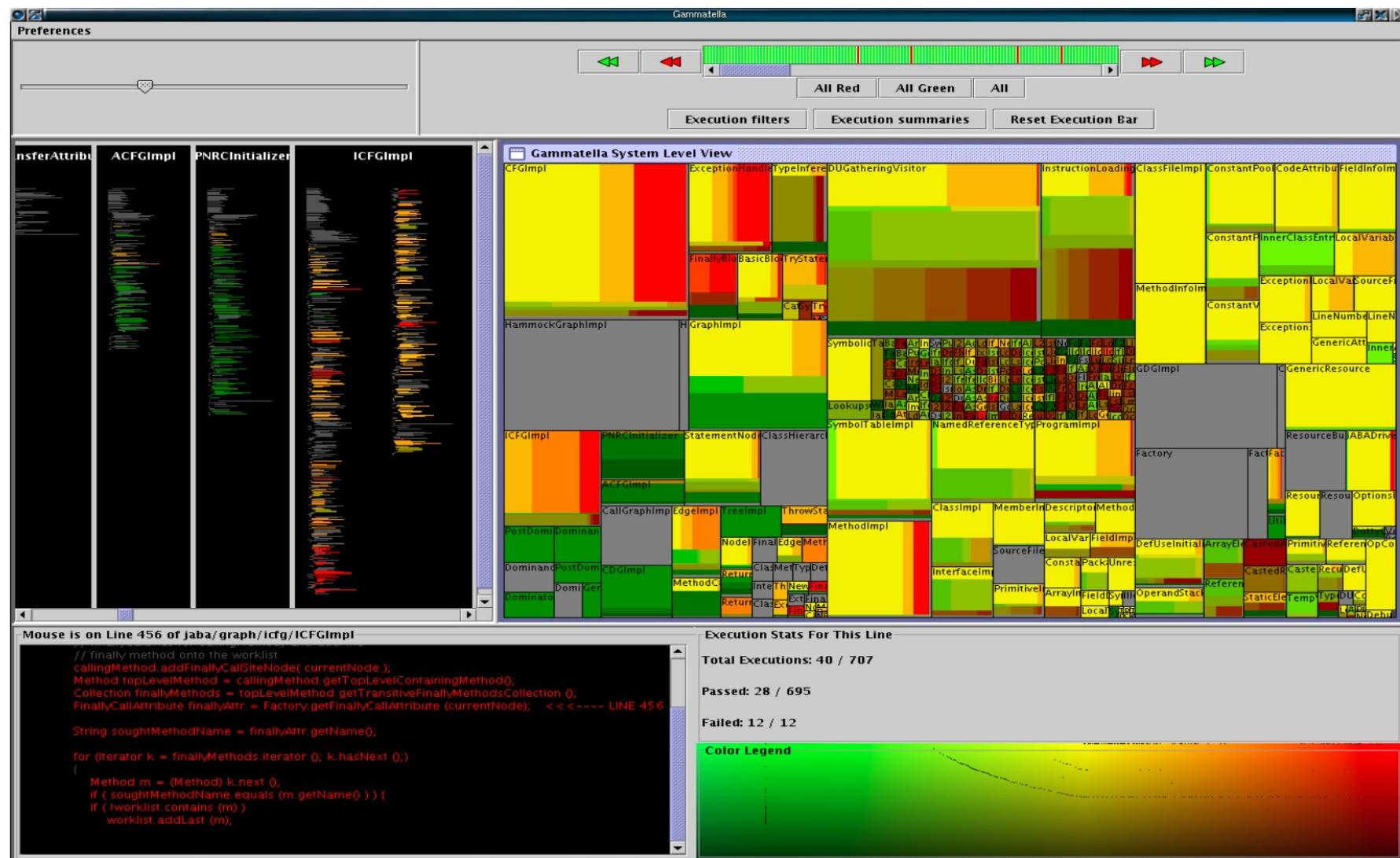
        mid() {
            int x,y,z,m;
1:         read("Enter 3 numbers:",x,y,z);
2:         m = z;
3:         if (y<z)
4:             if (x<y)
5:                 m = y;
6:             else if (x<z)
7:                 m = y;
8:             else
9:                 if (x>y)
10:                    m = y;
11:             else if (x>z)
12:                 m = x;
13:         print("Middle number is:",m);
}
    
```

		Test Cases					
		3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3
Pass/Fail Status	{	P	P	P	P	P	F
		●	●	●	●	●	●
		●	●	●	●	●	●
		●	●	●	●	●	●
			●				
				●			
					●		
						●	●
						●	●
							●

File/Module-level View



System-level View



Many Variants of This Approach

Tarantula^[1]
$$S(s) = \frac{\text{failed}(s)/\text{totalfailed}}{\text{failed}(s)/\text{totalfailed} + \text{passed}(s)/\text{totalpassed}}$$

Ochiai^[2]
$$S(s) = \frac{\text{failed}(s)}{\sqrt{\text{totalfailed} \cdot (\text{failed}(s) + \text{passed}(s))}}$$

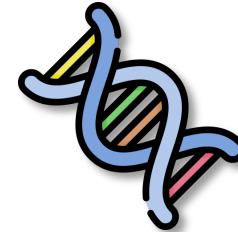
D*[3]
$$S(s) = \frac{\text{failed}(s)^*}{\text{passed}(s) + (\text{totalfailed} - \text{failed}(s))}$$

[1] J. Jones, M. J. Harrold, and J. Stasko. (2002) Visualization of test information to assist fault localization.

[2] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund. (2006) An evaluation of similarity coefficients for software fault localization.

[3] W. E. Wong, V. Debroy, R. Gao, and Y. Li. (2014) The DStar method for effective software fault localization.

Testing your tests with Mutation Testing



Mutation Testing

≡ ղղօՆՄՐՈՂԱՎ ԲԿԵՐ

ເນື່ອ test ຂອບດົກ ເນື່ອ ດາວໂຫລນ test cases

Key Idea: สร้าง mutant และนำมันมาทดสอบกับ Test Suite ที่มีอยู่ เพื่อจะได้ดูว่า Test Suite นั้นดีแค่ไหน (ตรวจสอบ mutant มั้ย)

Mutant: โค้ดกล้ายพันธุ์จากการแก้ไขเล็กๆ (เช่นเปลี่ยน operator จากอย่างหนึ่งเป็นอีกอย่างหนึ่ง,เปลี่ยนค่าตั้งต้นของ variable สักอันหนึ่ง) เรียกอีกอย่างคือการใส่ bug ลงไปใส่โค้ด

କେବଳ test case
କିମ୍ବା କେବୁ?

```
# source code  
x += y
```

mutation

• 14 •

$x \neq y$

```
# source code  
x in [1, 2, 3]
```

mutation

α not in $\{\pm, \sim, \circ, \cdot\}$

```
# source code  
x = True
```

mutations

X None

Common Types of Mutations

- **Statement Mutations**
 - เช่น ลบ else block, elif block, ลบ $x = x + 1$
- **Logical Mutations**
 - เช่น `if (a == b and c > d)` เป็น `if (a == b or c > d)`
- **Relational Mutations**
 - เช่น จาก `<` เป็น `<=`, จาก `==` เป็น `!=`
- **Arithmetics Mutations**
 - เช่น จาก `+` เป็น `-`, จาก `*` เป็น `/`
- **Method Mutations**
 - เช่น จาก `startsWith()` เป็น `endsWith()`
- **Constant Mutations**
 - เช่น จาก `PI = 3.14` เป็น `PI = 3.1415`
- และอื่นๆ (เช่น Variable mutations, Unary Operator Mutations, Exception mutations)



Mutation ที่ดีในแต่ละประเภท
ควรจะสร้างมาเพื่อเลียนแบบ
ความผิดพลาดที่เกิดขึ้นได้ในการ
เขียนโปรแกรมจริงๆ (หรืออีก
อย่างคือต้อง Realistic) และแต่
ละภาษาอาจมีชนิดที่มีเฉพาะแต่
ในภาษานั้นๆ

Mutation Testing Steps

ขั้นตอนในการใช้ Mutation Testing

1. ทดสอบ Program P กับ Test Suite ที่มีอยู่
2. สร้าง Set ของ Mutant ชนิดต่างๆสำหรับ P
3. นำแต่ละ Mutant ใน Set ข้อ 2 มาทดสอบกับ Test Suite
4. วิเคราะห์ผลลัพธ์ (*Interpreting Results*)
5. สรุปและรายงานผล (*Mutation Score*)

ควรหยุดสร้าง mutant
เมื่อไหร่?

วิเคราะห์ผลลัพธ์อย่างไร?

ผลที่สรุปออกมาเป็น
อย่างไร?

Interpreting Test Results

เมื่อเรา运行 Test Suite มาเร็ว กับ Program P ที่มี Mutant หนึ่งในสามสิ่งนี้ จะเกิดขึ้น (หมายเหตุ: assume ว่า Test Suite รัน Program P ผ่าน หมด)

- อย่างน้อย 1 ใน test cases ไม่ผ่าน

- หมายถึง Mutant นั้นๆ **ถูกฆ่า** โดย Test case
- สรุปได้ว่า test suite นี้ cover ส่วนที่ถูกแก้ไปโดย Mutant

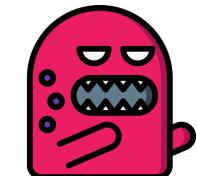
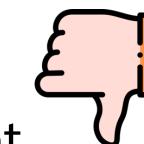
๑



- ทุก test cases ใน test suite ผ่านหมด

- หมายถึง Mutant นั้นๆ **ไม่ถูกฆ่า (รอดชีวิต)**
- สรุปได้ว่า test suite นี้ไม่ได้ cover ส่วนที่ถูกแก้ไปโดย Mutant

๒

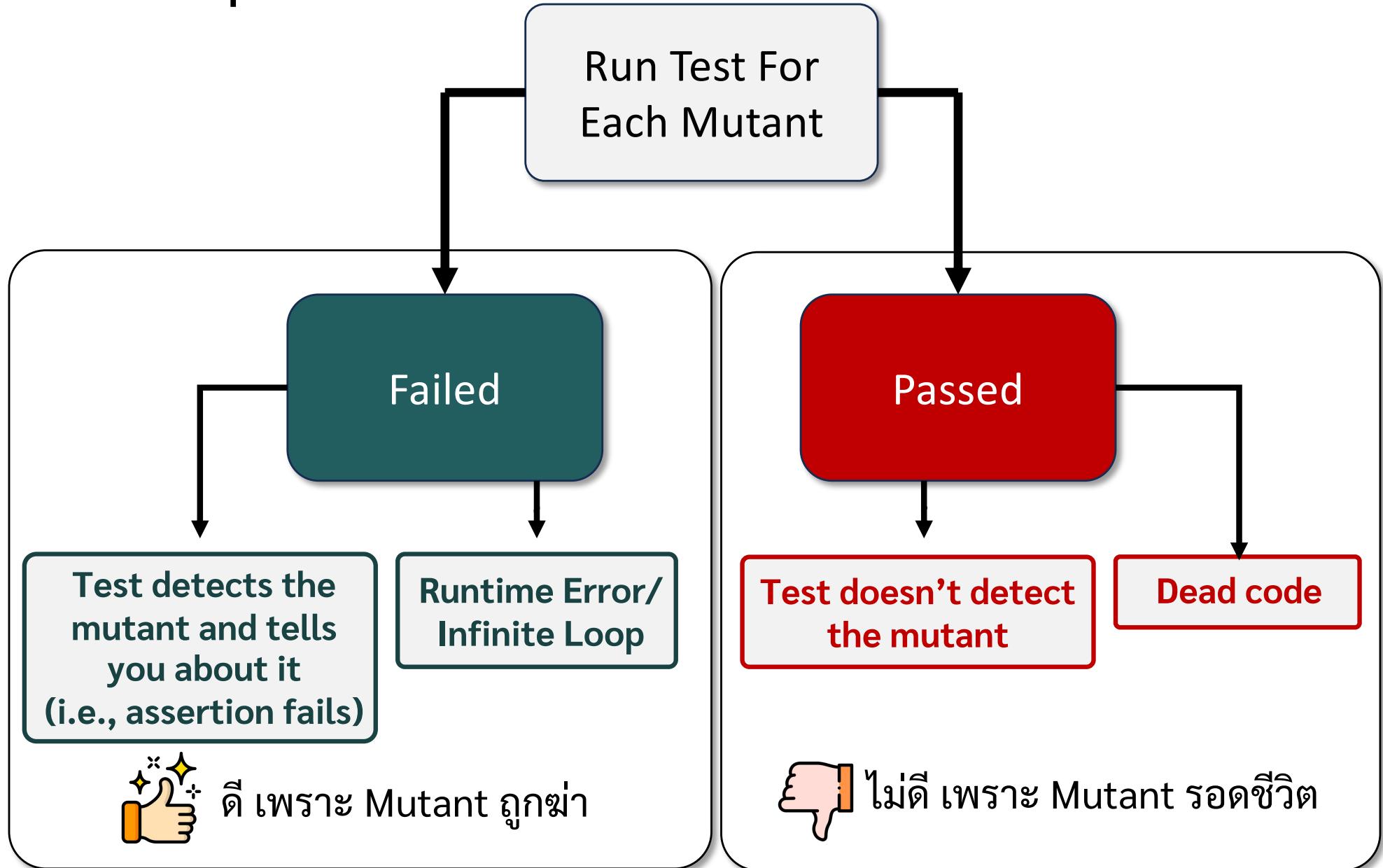


- ได้ Infinite loop/ runtime error

- ให้ถือว่า Mutant **ถูกฆ่า** เพราะ Test Suite ทำให้ Program ทำงานไม่ถูกต้อง (แต่ก็ต้องตรวจสอบเองให้แน่ใจว่าเกิดจากอะไรก็ได้)

๓ + ต้องตรวจสอบเพิ่ม

Interpretation



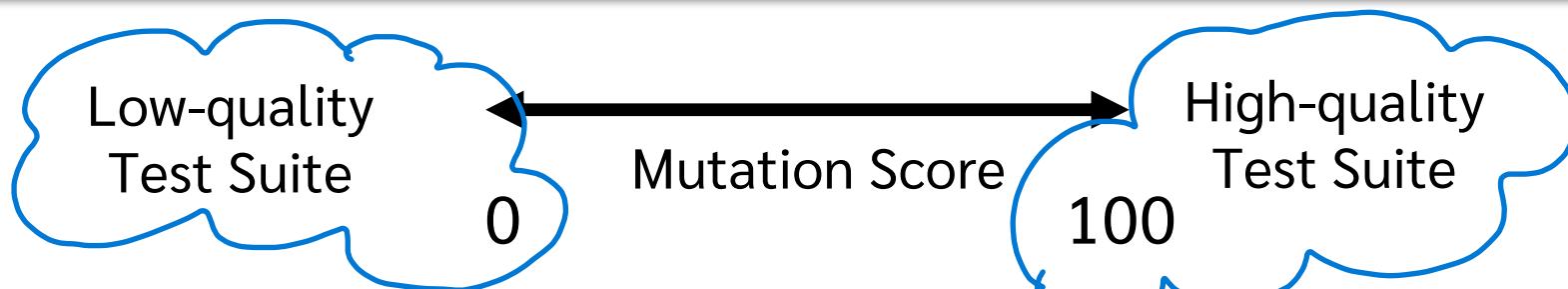
Mutation Testing Score

เป้าหมายคือต้องการฆ่า mutant ให้ได้มากที่สุด



โดยความประสิทวิภาคของ Test Suite จะวัดได้จาก Mutation Score ดังนี้

$$\text{Mutation score} = \frac{\text{Number of mutants killed}}{\text{Number of mutants created}} \times 100$$



Mutants Killed = มี test case อย่างน้อย 1 test case ที่ Fail สำหรับ mutant นั้นๆ
หมายเหตุ: 100% Mutation Score อาจเป็นไปไม่ได้ เพราะ ... (slide ถัดไป)

Equivalent Mutants - Challenge

សំណើលាងក្រុងការប្រើប្រាស់កម្មវិធីរបស់ខ្លួន និងការប្រើប្រាស់កម្មវិធីរបស់ខ្លួន

ในบางครั้งการกลยุทธ์ที่เกิดขึ้นอาจทำให้โปรแกรมมีฟังก์ชันที่เหมือนกับโปรแกรมดั้งเดิม หรือพูดอีกนัยหนึ่งก็คือไม่มี Test Case ที่เป็นไปได้ที่จะแยกแยะ mutant จากโปรแกรมดั้งเดิม แม้ว่า source code จะต่างกัน เราจะเรียก mutant เหล่านี้เรียกว่า **Equivalent Mutants**

```
index = 0
while True:
    do_stuff( )
    index = index + 1
    if index == 10:
        break
```

١٢

నీస్సు MStart \geq 10
గొంతుకుట్టు

```
if index == 10:
```

VS

```
if index >= 10:
```

Other Challenges

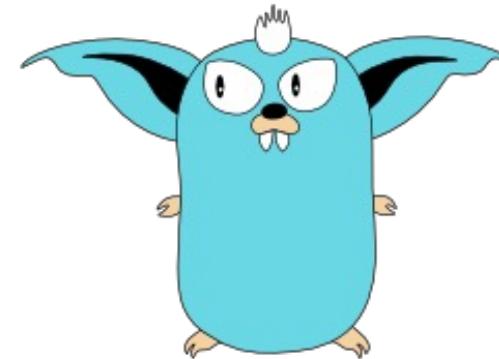
- จำนวน Mutant ที่เพิ่มมากขึ้นเมื่อ Code มีความซับซ้อนขึ้น (Scalability Issue) – **ปัญหาที่สำคัญ (ไม่มีคำตอบตายตัว)** = **ควรหยุดสร้าง mutant เมื่อไหร่ดี**
- ใช้ทรัพยากรามากเพื่อมาทดสอบทุก Mutant
 - เวลาที่ใช้ทดสอบ = **จำนวน Mutants x จำนวน Tests**
- คุณภาพของ Mutant ที่บางที่ mutant ก็ไม่สมเหตุสมผล (realistic) ทำให้เสียเวลาในการทดสอบ
- ยังจำเป็นต้องใช้แรงงานคนเยอะ (Manual Effort) เช่น ตรวจว่าเป็น Equivalent Mutants หรือไม่ หรือ ตรวจสอบ case ที่ได้ผลลัพธ์เป็น Runtime Error/Infinite Loop
- ต้องใช้ Tool มาช่วยในการสร้าง Mutant** แต่ก็มี Tool ไม่มากที่มาช่วยตรงจุดนี้ได้และอาจยังไม่ Support บางภาษา

Mutation Testing Libraries

- **Mutant** – Ruby
- **VisualMutator** – .NET/C#
- **Pitest** – Java/ Kotlin
- **Infection** – PHP
- **MutPy and Mutatest** – Python 3.x
- **Gremlins and Go-mutesting** – Go
- **Stryker Mutator** – JavaScript, .NET/C#, and Scala



Infection



Gremlins



Stryker Mutator

Simple Example:

Source code:

```
def isPositive(num):  
    if num > 0:  
        return True  
    else:  
        return False
```



Unit Tests:

```
def testpositive():  
    assertTrue(isPositive(3))  
  
def testnegative():  
    assertFalse(isPositive(-3))
```



Source code with relational mutation:

↳ Mutation შრომისას დატვირთვა

Mutation

```
def isPositive(num):  
    if num >= 0:  
        return True  
    else:  
        return False
```

Unit tests + mutation tests



So you made mistake
let me discover it for you,
and prevent production issues

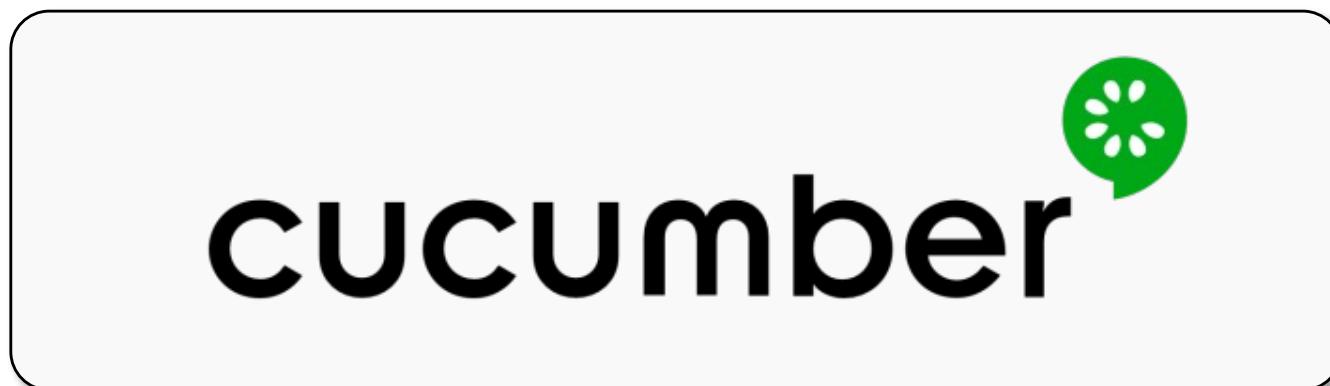
imgflip.com

Unit tests only



Why assertion `1==1` did
not discover production issues

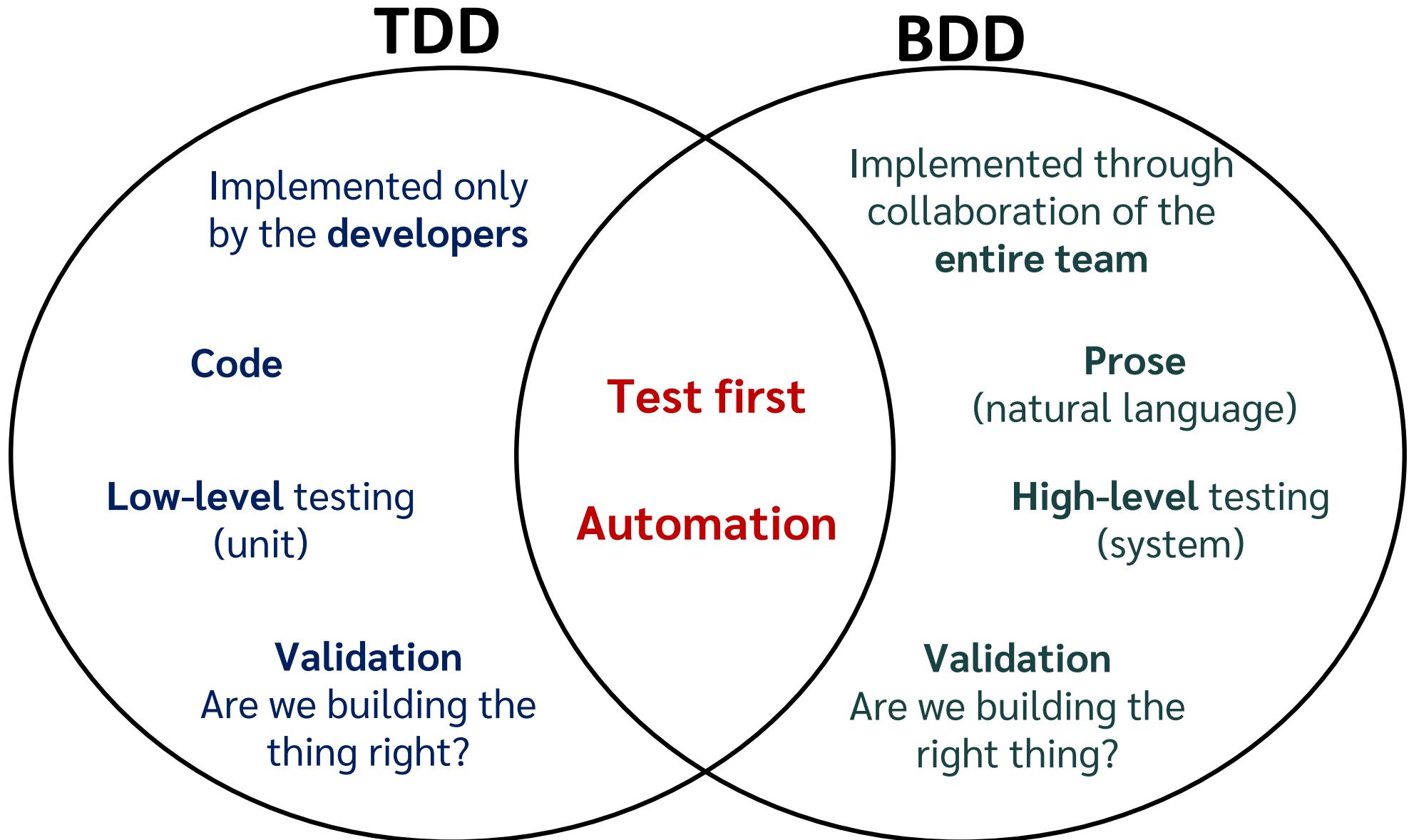
Behavior Driven Development (BDD) with Cucumber



Behavior Driven Development (BDD)

- การพัฒนาซอฟต์แวร์โดยการขับเคลื่อน (driven) ด้วย พฤติกรรม (Behaviour)
- **Collaboration and shared understanding:** เกิดขึ้นมาเพื่อให้นักพัฒนา, testers, business stakeholder ทำงานร่วมกัน โดยใช้ภาษาพูดในการ อธิบายพฤติกรรมของระบบ ทำให้ทุกคนเข้าใจตรงกันไม่ว่าจะเป็นคนที่ไม่มี ความรู้ด้านคอมฯ
- **Focus on behavior:** เกิดขึ้นมาเพื่อปิดข้อจำกัดของวิธี TDD (Test Driven Development) ที่ไม่ใช่ Programmer ก็เข้าใจได้ โดยเน้นไปที่พฤติกรรม ของตัวระบบ เพื่อให้ระบบสอดคล้องกับความต้องการและคาดหวังของผู้ใช้ มากขึ้น
- **Allow for Automation:** สามารถใช้ร่วมเครื่องมือการทดสอบอัตโนมัติได้ ทำให้สามารถตรวจสอบได้ตลอดการพัฒนา ว่าระบบตรงกับพฤติกรรมที่ คาดหวังหรือไม่

TDD vs BDD



User Story's Acceptance Criteria

- Acceptance Criteria ไว้ใช้อธิบายว่าระบบต้องทำงานอย่างไร แบบไหน ถึงจะส่งมอบและตรงตามความต้องการของลูกค้า (เกณฑ์)
- BDD นำแนวความคิดที่จะเอา Acceptance Criteria มาเขียน เป็น code ให้อยู่ในรูปที่ทุกคนในทีม (ไม่ใช่ dev เท่านั้น) สามารถเข้าใจได้
- **Cucumber** เป็นซอฟต์แวร์ยอดนิยมที่นำมาใช้สร้าง Test Script ตามหลัก BDD โดยใช้ Syntax แบบ Gherkin (Given, When, Then)

Domain-Specific Language for BDD (Gherkin)

As a new user, I want to be able to sign-up an account so that the system can remember me and my data

- **Given** I'm at the sign-up form
- **And** all these mandatory fields are entered
 - First Name
 - Last Name
 - Email
 - Password
- **When** I submit the form
- **Then** an account is created
- **And** account name is set as my Email

Given กำหนด
เงื่อนไขก่อนเริ่ม^{การทำงาน}

When เหตุการณ์
หรือกิจกรรมที่
เกิดขึ้น

Then ผลของ
เหตุการณ์หรือ^{กิจกรรม}

Cucumber

- ซอฟต์แวร์ยอดนิยมที่นำมาใช้สร้าง Test Case ตามหลัก Behavior-Driven Development (BDD)
- เครื่องมือการทดสอบอัตโนมัติที่เป็น Open Source และใช้งานได้ฟรี
- ใช้งานได้กับหลายภาษา Java, Ruby, JavaScript
- เขียน Test case โดยใช้ภาษา Gherkin (แต่ Dev ต้องเขียนไฟล์พ่วงที่ทำให้ cucumber เข้าใจภาษา Gherkin อีกที -- เรียกว่า step definitions)
- มี port ออกไปใช้กับภาษาอื่นๆ เช่น Python (Behave), C# (SpecFlow), Rust, PHP (Behat)



Two Parts of Cucumber

- Cucumber ประกอบด้วย 2 ส่วนหลักๆ
 - ส่วนของ **Feature File** หรือ file ที่รวม Test Case หลายๆ Case ไว้เพื่อทดสอบ Feature หนึ่งๆ ของระบบ เขียนอธิบายสิ่งที่จะทดสอบด้วยภาษาธรรมชาติ โดยใช้ Syntax แบบภาษา Gherkin (Given, Then, When)
 - ส่วนของ **Step Definition**: หรือ file ที่รวม parser (ตัวแปลความหมาย) ที่ทำให้ Cucumber เข้าใจความหมายและค่า parameter ต่างๆ ที่ปรากฏใน Given, Then, When ที่อยู่ในไฟล์ Feature ข้างบน

Important Keywords (Gherkin Language)

- **Feature:** ใช้เริ่มต้น test file เพื่ออธิบาย functionality ที่กำลังจะทดสอบ เพื่อ อธิบายสิ่งต่างๆ เช่น จุดประสงค์ของการทดสอบนี้เป็นต้น
- **Scenario:** ใช้เป็นตัวบอกรว่า test case นี้ทำอะไร (ใน 1 feature file ทดสอบมีได้ หลาย scenarios หรือหลาย test case)
- **Scenario Outline:** คล้ายๆ Scenario แต่ใช้ในกรณีที่ต้องมีการใส่ค่า variable แทน ค่า placeholder บางอย่าง
- **Given:** ใช้บอก condition ตั้งต้นของ test case
- **When:** ใช้บอกเหตุการณ์หรือกิจกรรมที่เกิดขึ้น
- **Then:** ใช้บอกผลกระทบหรือ outcome ของเหตุการณ์หรือกิจกรรมจาก When
- **And:** ใช้เชื่อมประโยคในกรณีที่ Given, When, Then มีมากกว่า 1 สิ่งที่คล้ายกัน
- **But:** คล้ายๆ And แต่ใช้ในกรณีที่มีข้อแตกต่าง
- **Examples:** ใช้กับ Scenario Outline เพื่อนำค่าในตารางมาใช้กับ placeholder
- **Step definition:** function/method definition

Example:

Feature:

```
# feature/hello_cucumber.feature
Feature: Hello Cucumber
As a product manager
I want our users to be greeted when they visit our site
So that they have a better experience

Scenario: User sees the welcome message
When I go to the homepage
Then I should see the welcome message
```

Step Definitions

```
When(/^I go to the homepage$/) do
  visit root_path
end

Then(/^I should see the welcome message$/) do
  expect(page).to have_content("Hello Cucumber")
end
```

Example:

Feature:

ใช้ Scenario Outline แทน scenario ในกรณีที่จะมีการแทนค่า placeholder จากในค่าในตาราง Examples

เมื่อ run, ตัวทดสอบจะนำค่าในตาราง Examples แต่ละตัวไปใส่ใน scenario ตรง placeholder ให้อัตโนมัติ

Feature: Is it Friday yet?

Everybody wants to know when it's Friday

Scenario Outline: Today is or is not Friday

Given today is "<day>"

When I ask whether it's Friday yet

Then I should be told "<answer>"

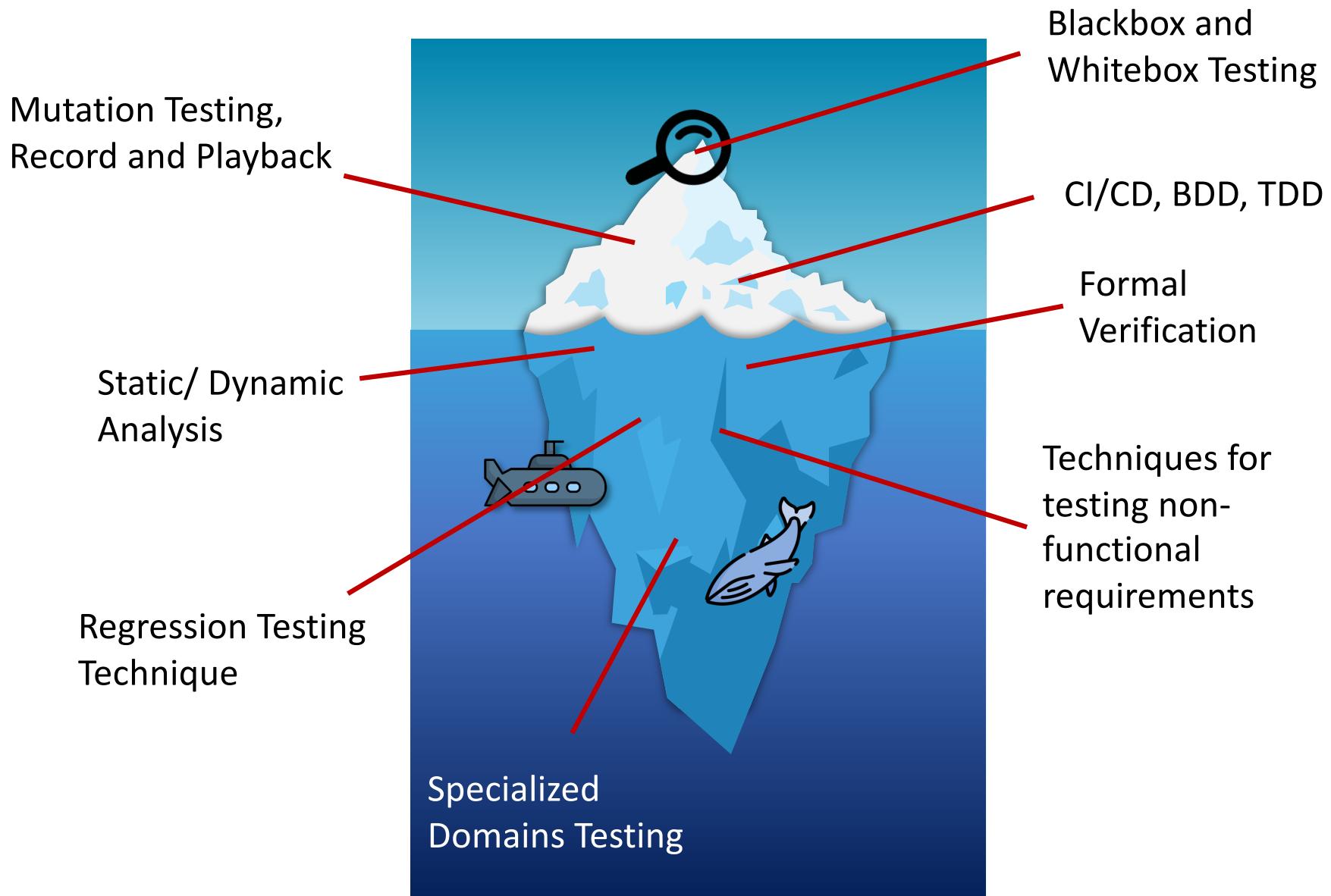
Examples:

day	answer
Friday	TGIF
Sunday	Nope
anything else!	Nope

Example:

```
class IsItFriday {  
    static String isItFriday(String today) {  
        return "Friday".equals(today) ? "TGIF" : "Nope";  
    }  
}  
  
public class Stepdefs {  
    private String today;  
    private String actualAnswer;  
  
    @Given("today is {string}")  
    public void today_is(String today) {  
        this.today = today;  
    }  
  
    @When("I ask whether it's Friday yet")  
    public void i_ask_whether_it_s_Friday_yet() {  
        actualAnswer = IsItFriday.isItFriday(today);  
    }  
  
    @Then("I should be told {string}")  
    public void i_should_be_told(String expectedAnswer) {  
        assertEquals(expectedAnswer, actualAnswer);  
    }  
}
```

Software Testing – Tip of the Iceberg



Questions ?

