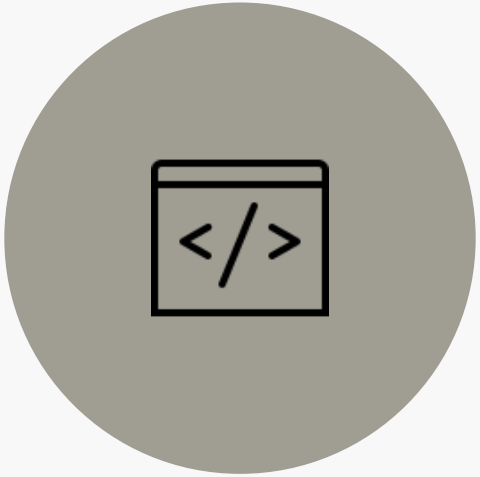
SOA Testing Strategy

06/08 - Web Service Dev & SOA

Adenda

What's On the menu? - Week 6

I: Presentation Stuffs



II: SOA Testing by Dev. Level

III: SOA Testing by Functionality

I: Presentation Stuffs

06/08 - Web Service Dev & SOA

Presentation Organisation

Both within 20 minutes

Course-Organised

- **12 Mins:** Slides
- **8 Mins:** Live Demo

Self-Organised

- **Declare the Presentation Organisation** @ The Very First Slide.

Presentation Order

Starts @ 1PM. No lecture neet week. The first group will be notified by the email. Feel free to come to the room earlier for preparation.

First Hour : 4 Groups that opted for **reservation**

- First Come, First Serve = Considers the time of **the last** submission. Submit earliest to become the first.

Then: 4 Groups that opted for **random**:

1. Shikono
2. HoneyLemon
3. docker compose down
4. Super Laika

Source: a python seeded random script (attached in GO EDU).

Note: During the presentation day. **Free to not attend** the other group's presentation (Attendance = Presentation). **Free to leave** the room after the presentation.

Presentation Checklist

For your system:

- ❑ **From Week 3:** How to compose the system to meet the problem statement requirements (i.e. Secure/Security & Adaptability/Correctness)? Using mediator-based and/or peer-to-peer topologies? How and why?
- ❑ **From Week 4:** Using REST, SOAP, ESB or a combination of the above to meet the problem statement requirements? How? Why and why not?
- ❑ **From Week 5:** How to deploy the system to achieve the problem statement requirement (i.e. self-recovery/reachability)? Why and why not?
- ❑ **From Week 6:** How to design a test strategy to achieve the problem statement requirement (i.e. secure/security)?

Extra Marks for:

- ❑ Mentioning **Week 2 contents** (e.g. async/sync, coupling, stateful/stateless)
- ❑ Analyse the system **beyond** the questions above (e.g. compose the system to meet self-recovery/reachability, etc.)

Presentation Submission

Two Ways: Go EDU & Email.

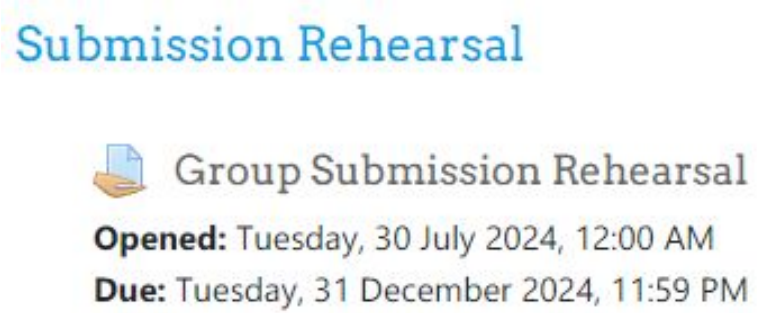
- **Primary:** Go EDU (Available Now)

Group Submission: Ensure you are in the right group.
5 PM Deadline before the Presentation Day: Save time for the 1st group.

Presentation Submission

Two Ways: Go EDU & Email.

- **Primary:** Go EDU (Available Now)
  - There is **a rehearsal** submission to try. Won't grade it. Cannot test for you (my user role is Teacher). Let me know any issues.



Presentation Submission

Two Ways: Go EDU & Email.

- **Back-Up:** Email
  - Subject: [6622][(Team Member)] Presentation Submission
  - **Beware:** Email can delay (as we have experienced during the class).

Congratulations

To 3 ***Undisputed*** A & B+ bound groups (based on Week 5 Exercise)

- **Very minimal** (2-3 sentences) to improve. Primary just need to **maintain this standard** for the presentations & exams **to get the desired grade.**
- **Undisputed:** Decent sized answer + cover most (if not all) aspects of assessment criteria <u>in details.</u>

Congratulations

To 3 ***Undisputed*** A & B+ bound groups (based on Week 5 Exercise)

- Observation: Each group has **a unique way** to answer. This is what I wish for; Becoming **the best of you** (not the best of me).
- For the rest: **Jury is still out**. Let's start practise /w this week's exercise!

II: SOA Testing by Dev.Level

06/08 - Web Service Dev & SOA

Testing Based on Dev. Level

TL;DR: Testing = Rehearse the system. Ensuring that it works as intended.

Unit Testing

Function/Method-level testing.

Integration Testing

Interservice/ Intercomponent-level testing.

System Testing

Use Case-level testing**.**

Unit Testing

Example Test Case:

```
def is_even(number):
    return number % 2 == 0
```

Unit Test Case 1

Target: is_even(*number*)

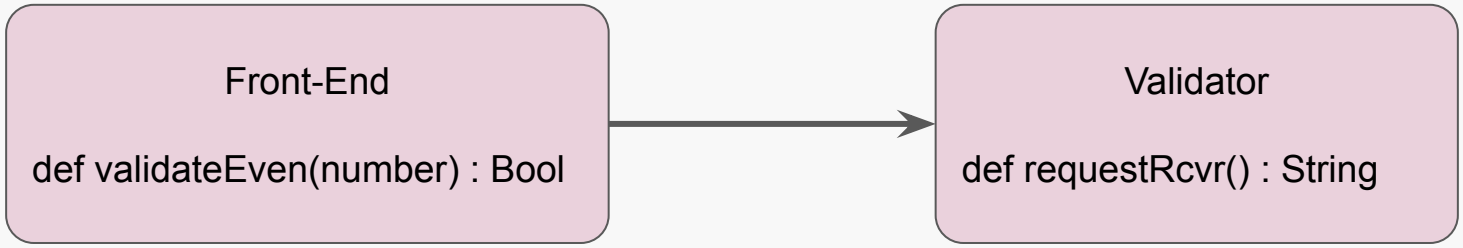
Input: number = 32 (i.e. is_even(32))

Pass: Output == 0
Fail: Output != 0

Note: Revised test cases' Pass/Fail conditions. They are different from the earlier version.

Integration Testing

Example Test Case (Outbound):



Integration Test Case 1

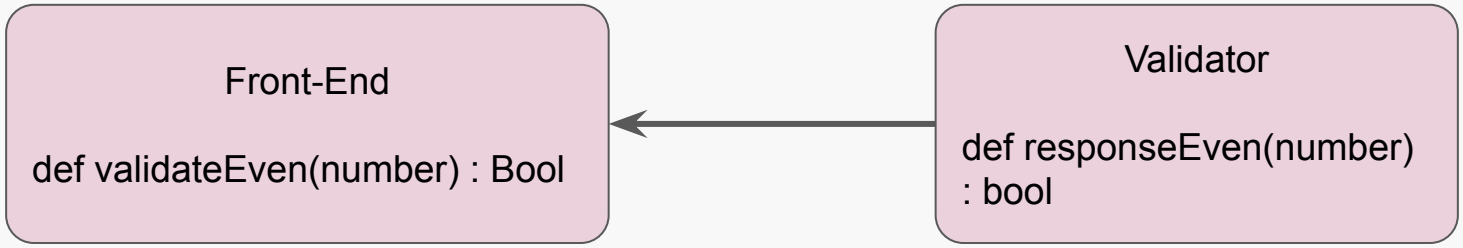
Target: validateEven@Front-End + requestRcvr@Validator

Input: number = 32@Front-End (i.e. validateEven(32))

Pass: requestRcvr() == “validateEven(32)”
Fail: requestRcvr() != “validateEven(32)”

Integration Testing

Example Test Case (Inbound):



Integration Test Case 2

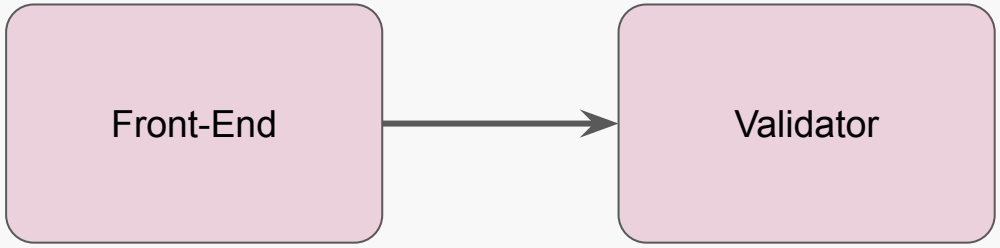
Target: responseEven@Validator + validateEven@Front-End

Input: number = 32@Validator (i.e. responseEven(32))

Pass: validateEven(32) == true
Fail: validateEven(32) != true

System Testing

Example Test Case:



System Test Case 1

Input:
1. Typing ‘32’ at a Front-end tablet
2. Tap ‘Send’.

Pass: Displayed “Yes, this number is even.” on the screen.
Fail: Displayed “Sorry, this number is odd.” on the screen.

III: SOA Testing by Functionality

06/08 - Web Service Dev & SOA

Testing Based on Functionality

TL;DR: Testing = Rehearse the system. Ensuring that it works as intended.

Functionality Test

“Hard” Requirement: **Break** the system’s functionality when failed (e.g. previous examples).

Non-Functionality Test

“Soft” Requirement: Not break the system’s functionality. But **undesirable** when failed.

Non-Functionality

Examples:

Robustness Test

- Related to **Recoverability & Availability** of the System.

Service Level Agreement (SLA) Test

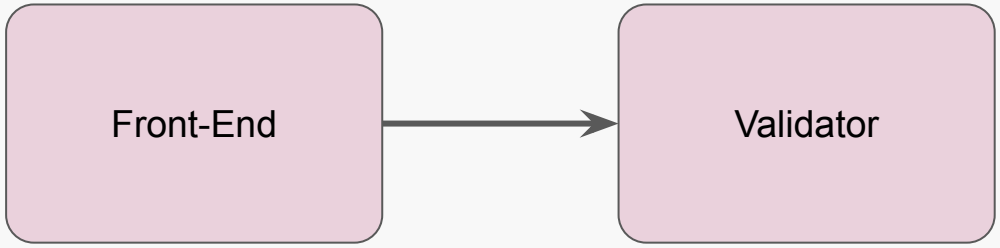
- Related to **Performance & Availability** of the System.

Dependability Test

- Related to **Security** of the System.

Robustness Test

Example Test Case:



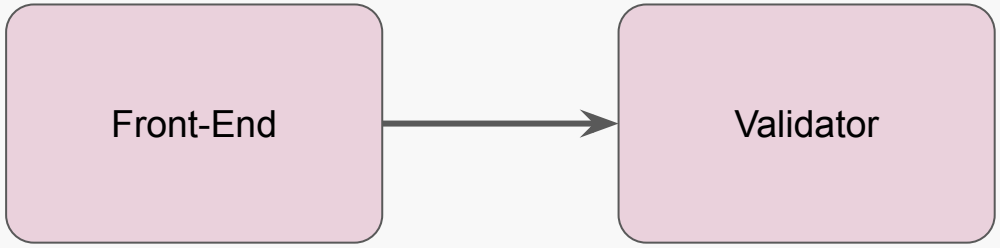
Non-Functional System Test Case 1
Prerequisite: Validator is down.

Input:
1. Typing ‘32’ at a Front-end tablet
2. Tap ‘Send’.

Pass Conditions:
1. Validator **is recovered within 2 minutes**.
2. Front-End displayed “Yes, this number is even.” on the screen.

SLA Test

Example Test Case:



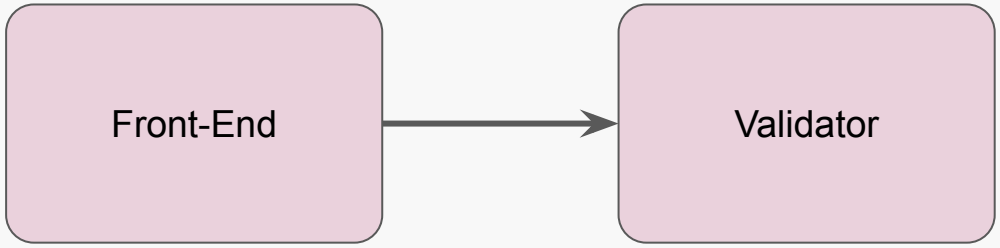
Non-Functional System Test Case 2
Prerequisite: Validator is down.

Input:
1. Typing ‘32’ at a Front-end tablet
2. Tap ‘Send’.

Pass Conditions:
1. Validator is recovered.
2. Front-End displayed “Yes, this number is even.” on the screen.
3. **The whole process takes less than 3 minutes.**

Dependability Test

Example Test Case:



Non-Functional System Test Case 3
Prerequisite: Validator is down.

Input:
1. Typing ‘32/0’ at a Front-end tablet
(Attempt to break the sys with the division of 0)
2. Tap ‘Send’.

Pass Conditions:
1. Front-End displayed “Yes, this number is even.” on the screen (i.e. **The front-end request is sanitised to be 32 at Validator**).

Testing Based on Functionality

In SOA (or any real-world) system:

Functionality

May not enough to satisfy the customer.

Non-Functionality

Can **change customer' minds.** (e.g. Android vs iOS, Switch vs PS vs Xbox,)

IRL

Requires the combination between testings /w different level & functionality.

Functionality

May not enough to satisfy the customer.

Non-Functionality

Can **change customer' minds.** (e.g. Android vs iOS, Switch vs PS vs Xbox,)

Combinatorial Explosion (1)

A Classic SW Testing Problem ...

Toy example: A system of 2 services, one function per service, each function receive a boolean input & has 1 if-else based return:

- No. of unit test cases per service = **2** (1 for if, 1 for else).
- No. of intrg. test, assuming the network connection is perfect:
  2 (Service A) * 2 (Service B) * 2 (Inbound + Outbound) = **8** to cover all possibility.

What if we considers more realistic network connection (e.g. 4 - online, partial loss, delayed, no connection), No. of intrg. test will be:

2*2*2 (From the above)*4 = **32** cases to cover all possibility.

Note: This is just a toy example. How about the real-world system? (No. of service, no. of function per service, no. of if-else, range of inputs, etc.)

Combinatorial Explosion (2)

... Meets A Classic Mitigation Approach

- Too many test cases /w so little time & efforts.
- This is a classic SE problem /w ongoing research.

- Many mitigation approaches are available (automated test, CI, etc.). **No perfect solution (more detail will be on another course).**

- Classic approach that **works for all systems**: Make a test strategy, by **prioritising** the most important test cases **(manually)**....

Running Example:

Milk's Restaurant - Test Strategy (1) - Functionality

- Order Mgmt. **is the most important service:**
  - Missing Order = Angry Customer & Reputation Loss
  - Delayed Order = Angry Customer, Reputation Loss & Financial Loss (Customer May Reject Finished Order)

Test Strategy to Prevent Missing Order

- Unit Test Cases on Order Mgmt:
  - From Front-End Requests:
    - Ensure that the order request is received.
    - Ensure that the order cancel request is received.
    - Ensure that the order revision request is received.
    - Ensure that the order status tracking request is received.
  - To Queue Mgmt:
    - Ensure that the order is forwarded successfully.

Running Example:

Milk's Restaurant - Test Strategy (1) - Functionality

Test Strategy to Prevent Missing Order

- Integration Test Cases:
  - Inbound/Outbound between Front-End and Order Mgmt.
  - Inbound/Outbound between Order Mgmt and Queue Mgmt.

- System Test Cases:
  - Customer order food.
  - Chef cook food.

Running Example:

Milk's Restaurant - Test Strategy (2) - Non-Functional

Test Strategy to Prevent Delayed Order

- Unit Test Cases on Order Mgmt:
  - From Front-End Requests:
    - Ensure that the order request is received **within 2 mins.**
    - Ensure that the order cancel request is received **within 2 mins.**
    - Ensure that the order revision request is received **within 2 mins.**
    - Ensure that the order status tracking request is received **within 2 mins.**
  - To Queue Mgmt:
    - Ensure that the order is forwarded successfully **within 2 mins.**

Running Example:

Milk's Restaurant - Test Strategy (2) - Non-Functional

Test Strategy to Prevent Missing Order

- Integration Test Cases:
  - Inbound/Outbound **+ Timeout Retry** between Front-End and Order Mgmt.
  - Inbound/Outbound **+ Timeout Retry** between Order Mgmt and Queue Mgmt.

- System Test Cases:
  - Customer order food, reaching Order Mgmt **within 5 minutes (in any network conditions)**.
  - Chef cook food **within 5 minutes (in any network conditions)**.

Group Exercise

06/08 - Web Service Dev & SOA

Group Exercise -Week 6

1. Based on the problem statement, how would you design a test strategy to achieve the following requirement?
   a. For IoT: Security
   b. For Metaverse: Secure
   c. For DIY projects: Secure/Security

Elaborate why your test strategy can make you achieve the above requirement.

Tip: *Try to use this exercise as a writing practice for the exam & presentation so you can get in-class feedback to practice by your own later.*

Send To:
suwichak.fu(at)kmitl.ac.th

Subject:
[6622][(Team Name)][IoT/Metaverse] Group Exercise Submission

Example:
[6622][Magneto][IoT] Group Exercise Submission