# *Design principles in Test Suite Architecture*

InSTA 2015

(International workshop on Software Test Architecture)

Graz, Austria 2015/4/13(Mon)

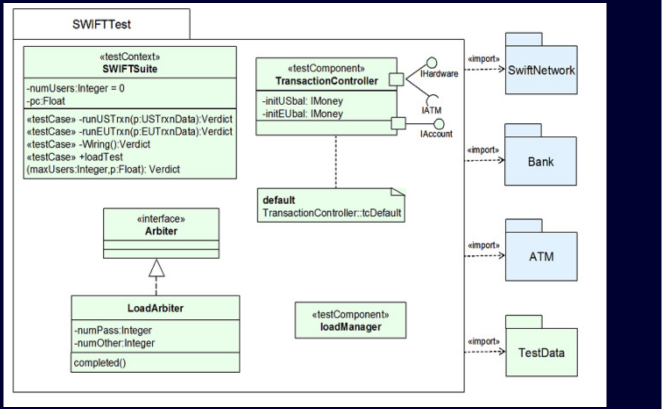
Nishi, Yasuharu

The University of Electro-Communications, Japan

# Software Test Engineering Process

- As software has got huge and complicated,
  test cases (= test suite) also get huge and complicated
  - such as
    - a test project with over 100,000 test cases
    - over 10 test levels
    - various test types such as load, configuration and security
  - You have to develop huge and complicated test suite systematically
- But technologies on test planning or test strategy
  are just immature
  - Engineering work and management work for test development  are confused
- It is necessary to define software test engineering process
  to develop huge and complicated test suite systematically

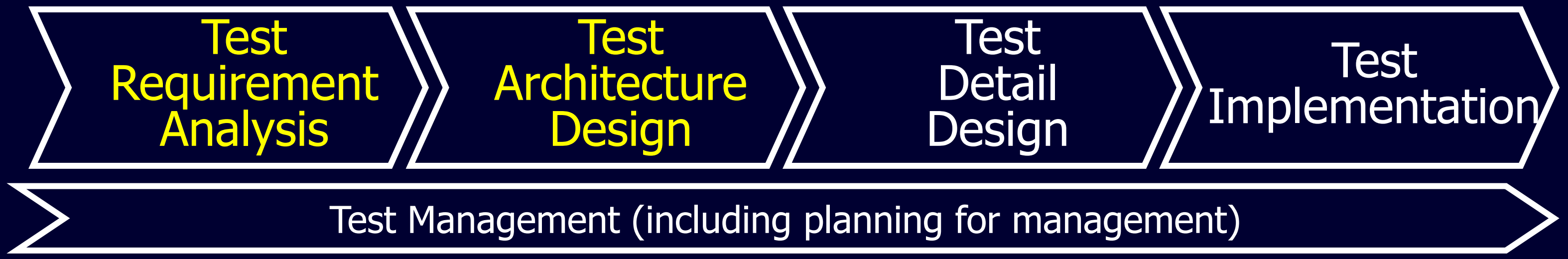
# Test “system” architecture and test “suite” architecture



- UTP defines ‘test architecture’ as test “system” architecture
  - UTP: UML Test Profile
  - Architecture for software development has two types as software architecture and system architecture
    - » Software architecture focuses on software inside
    - » System architecture focuses on execution environment
  - The concept of ‘test architecture’ of UTP focuses not only on architecture of test suite but rather on execution environment including automation
    - » In other words,
      UTP mainly focuses on Test “system” architecture
      but we should also research on Test “suite” architecture
  - The concept of Test Architecture in this presentation is test “suite” architecture

# VSTeP

- VSTeP(Viewpoint-based Software Test Engineering Process) is a generic test engineering process model focusing on test viewpoint
  - You can stress upper phase of test engineering process such as test requirement analysis and test architecture design which tend to be negligent
  - VSTeP drives you to good test suite, good review for test design, accumulation of knowledge and experience on testing
  - Reuse and improvement will be easy because you can do reverse-engineering of your past (unorganized) test suite
  - NGT (Notation for Generic Testing) is a made-in-Japan notation for Test Requirement Analysis and Test Architecture Design
    - » Modeling skill like object-oriented design is essentially necessary



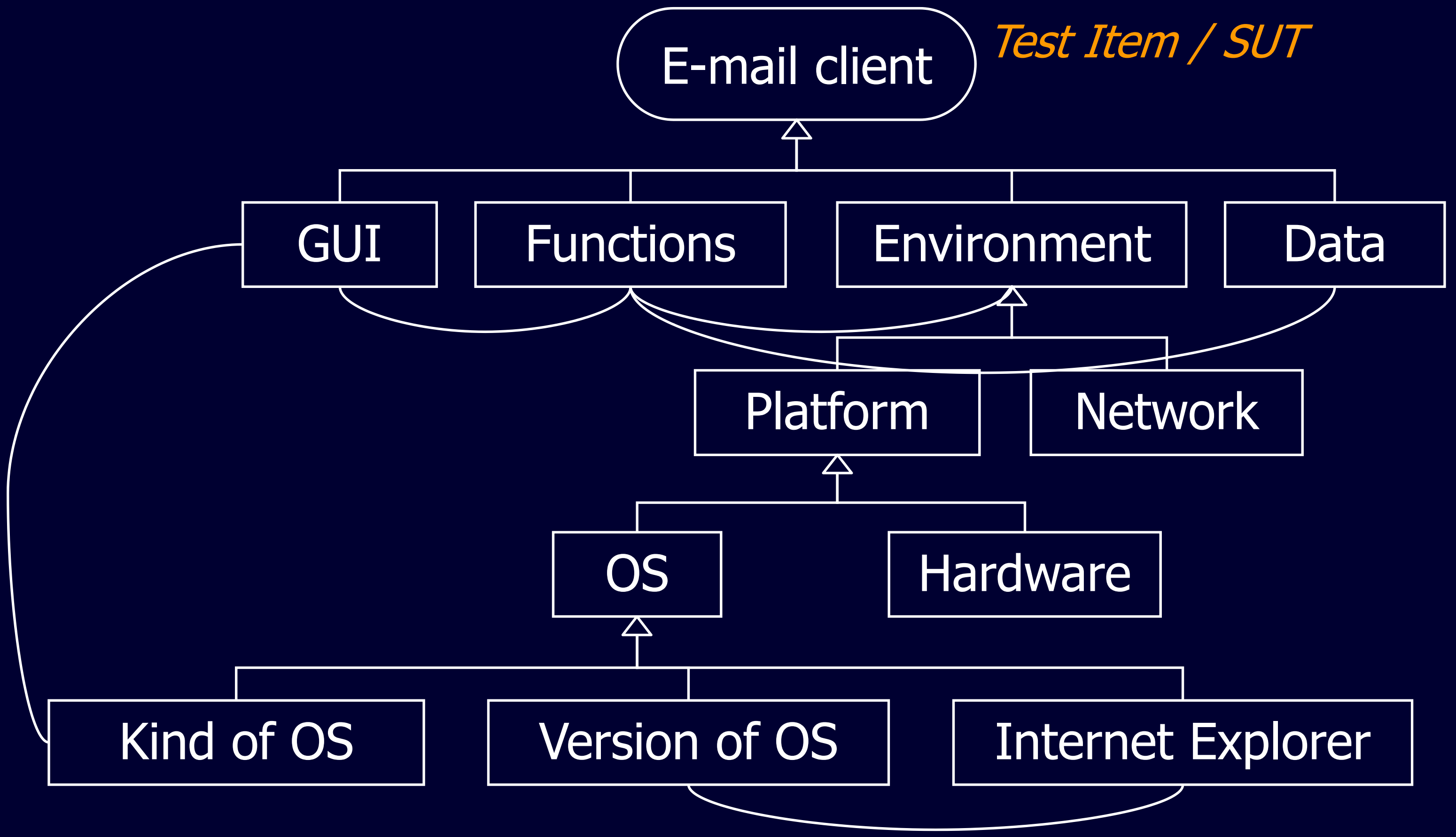
Test Requirement Analysis → Test Architecture Design → Test Detail Design → Test Implementation

Test Management (including planning for management)

# Detail phase of VSTeP

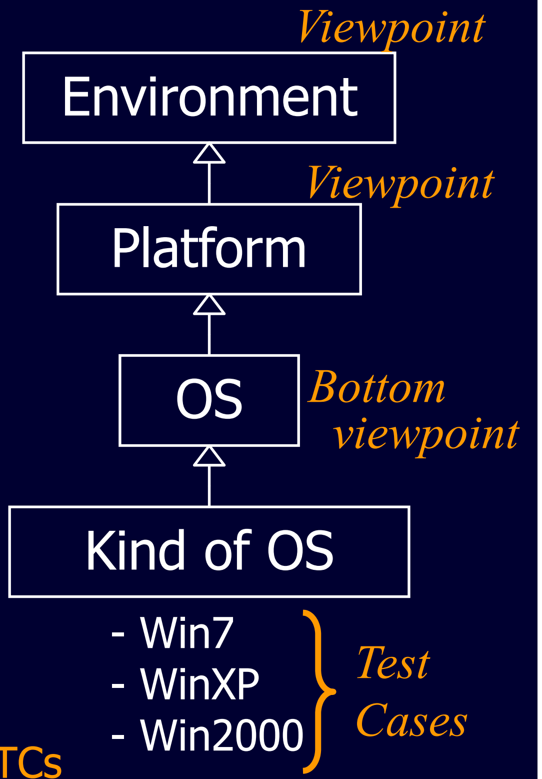
- TRA: Test Requirement Analysis
  - To make a test requirement model
    - » To extract, organize and understand test requirement
    - » To create a test requirement model which consists of test viewpoints, i.e. to create a viewpoint diagram
- TAD: Test Architecture Design
  - To make a test architecture model
    - » To re-organize test viewpoints into test containers as test types, levels and cycles for making test smooth
    - » To assemble test viewpoints into test frames which is template for TDD
- TDD: Test Detail Design
  - To make test cases
    - » To set values in detail into test frames or test viewpoints
- TI: Test Implementation
  - To make test scripts
    - » To add detail information necessary to execution to test cases
    - » To combine simple test scripts into a compound test script for making execution efficient

# Example of part of viewpoint diagram drawn for TRA



# What is test viewpoint: abstract test case

- Test cases has test values
  - ex) parameter: Kind of OS, values: Win7, WinXP, Win2000
  - Test parameters are also called as test conditions and test values are also called as test coverage items
  - Test cases consists of test values
- Viewpoints are abstract test cases
  - Bottom viewpoints means test parameters
  - Viewpoints don’t express any test values or test cases
  - Viewpoints can have hierarchical structure like classification trees or class diagrams
  - Viewpoints can be extracted from test conditions, test items and quality characteristics such as load, configuration and performance
  - Ideally viewpoints should indicate an INTENTION of a test case
    - » Viewpoint diagram can be a repository of intentions of TCs



# Various test viewpoints

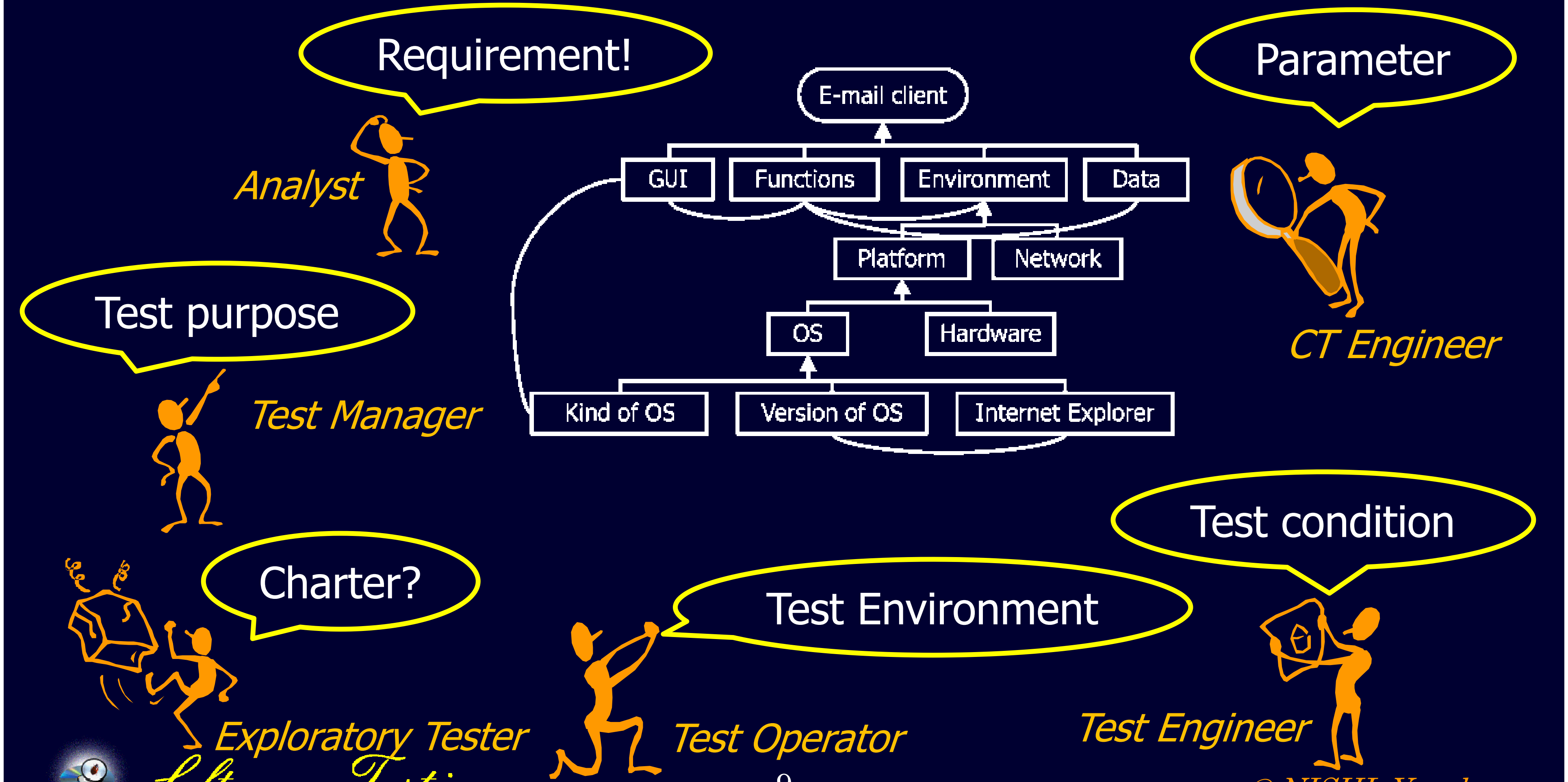
- Test viewpoint is a point where test engineers focus an attention for grasping a big picture of test design
  - Test viewpoint is abstraction and source of test cases
- Types of test viewpoints depend on organizations and/or test engineers

OS

- What should be exhausted:
  - » Specs, functions, data etc.
  - » Test conditions
- Characteristics which should be achieved
  - » Quality characteristics, non functional requirements etc.
- Parts of test items
  - » Funcs, Subsystems, modules etc.
- Bugs
  - » Errors and failures, bug patterns, weak points of test items etc.
- Customer usage
  - » Business, lifestyle etc.
- Other parts of systems than software
  - » Hardware units, hardware failures etc.
- Test types
  - » Load test, configuration test etc.
- Test levels
  - » Component test, system test etc.
- Lists and/or diagrams developed until software testing
  - » Use cases, State transition diagrams etc.

# Why “viewpoint” ?

- The word “viewpoint” is independent of roles



E-mail client

GUI, Functions, Environment, Data

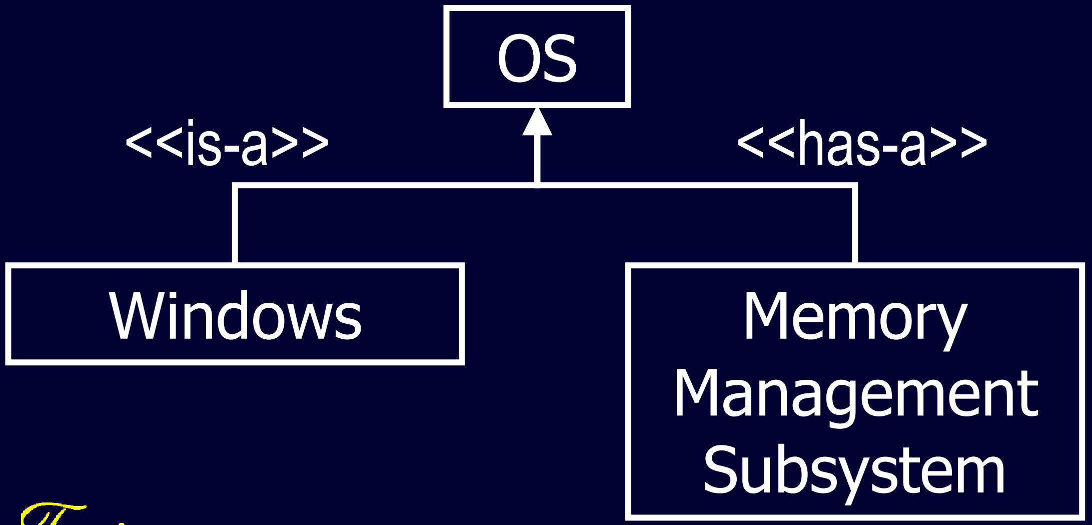
Platform, Network

OS, Hardware

Kind of OS, Version of OS, Internet Explorer

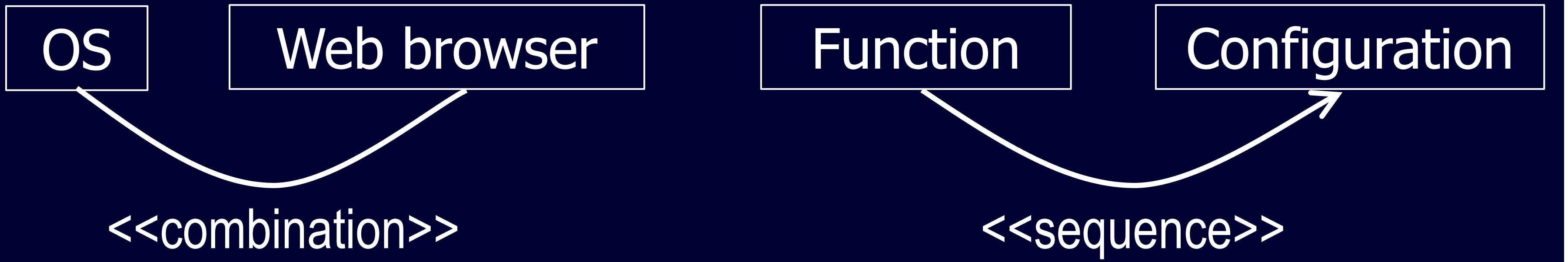
# Types of Hierarchical relationship

- Test viewpoints have two fundamental relationships
  - Hierarchy relationships and Interaction relationships
  - Types of relationships can be expressed as "<<stereotype>>"
- Hierarchical relationships can bear several meanings
  - is-a relationship: inheritance
  - has-a relationship: possession
  - There may be other hierarchical relationships
    - object-attribute and cause-effect is example



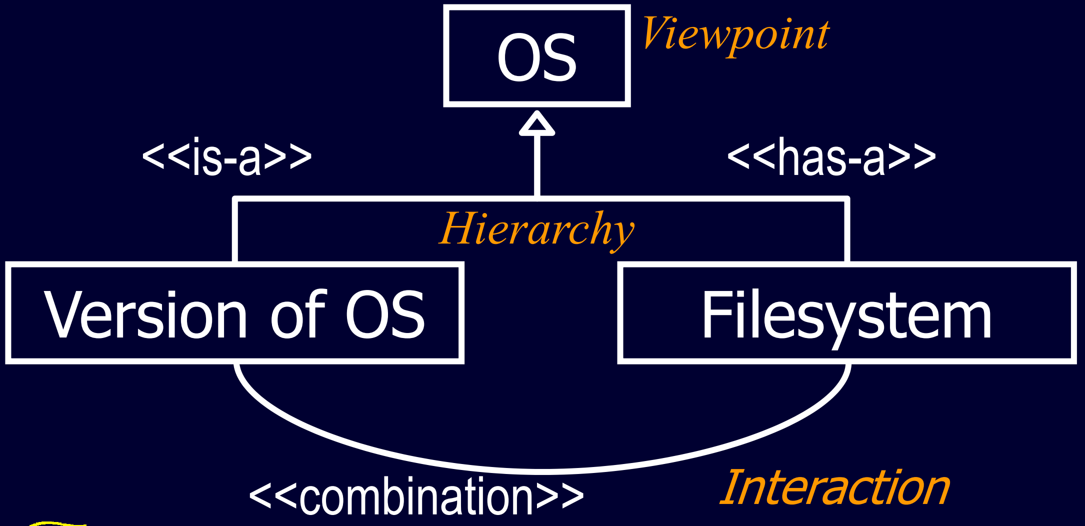
# Interactive relationships of viewpoints

- Viewpoints can relate each other with interactive relationships
  - Non-hierarchical relationships are necessary: Interactive relationships
  - They can also bear several meanings: combination, sequential etc.
  - Lines without arrowhead represent "combinatorial relationships"
  - Arrows with an open head represent "sequential relationships"
  - Relationships can represent their meanings with <<stereotype>>
  - In this workshop interactive relationships without stereotypes represent combinatorial relationship

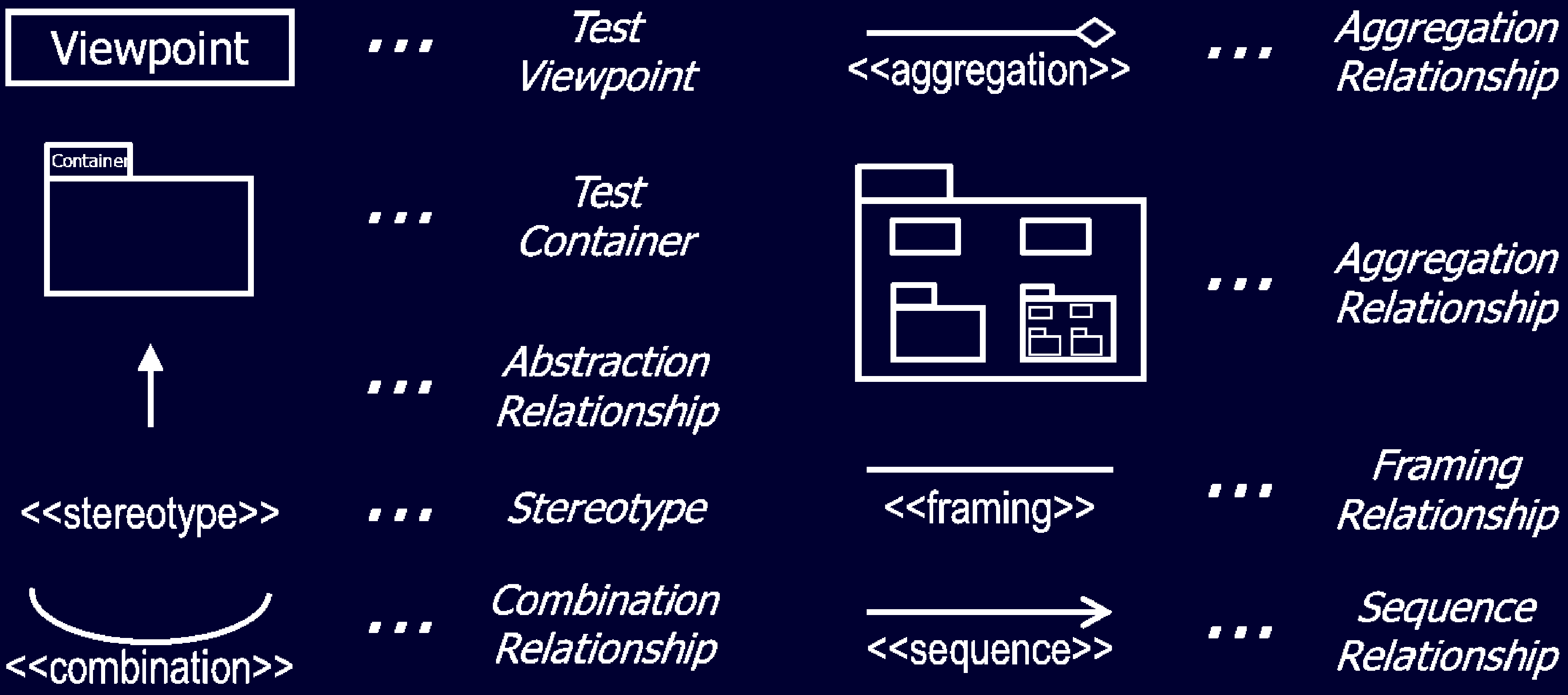


# Relationships of viewpoints

- Test viewpoints have two fundamental relationships
  - Hierarchy relationships
    - Detail a viewpoint step by step to reach test coverage item with a straight line
    - Have several types such as is-a, has-a, cause-effect, object-attribute
  - Interaction relationships
    - Connect test viewpoints to test combination of viewpoints with a curved line
    - Have several types such as combination (needs combinatorial testing) etc.
- Types of relationships can be expressed as “**<<stereotype>>**”



# Notation of viewpoint diagram in NGT



Viewpoint … *Test Viewpoint*

Container … *Test Container*

… *Abstraction Relationship*

<<stereotype>> … *Stereotype*

<<combination>> … *Combination Relationship*

<<aggregation>> … *Aggregation Relationship*

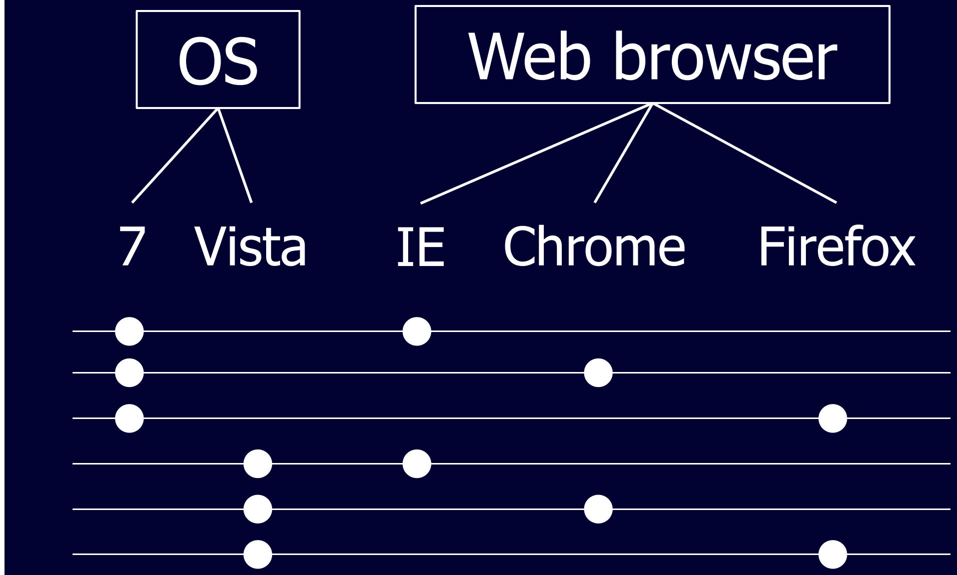
… *Aggregation Relationship*

<<framing>> … *Framing Relationship*

<<sequence>> … *Sequence Relationship*

# Viewpoint diagram is simple enough

- Viewpoint diagram is simple enough to make a TRA/TAD model
  - More simple than classification tree

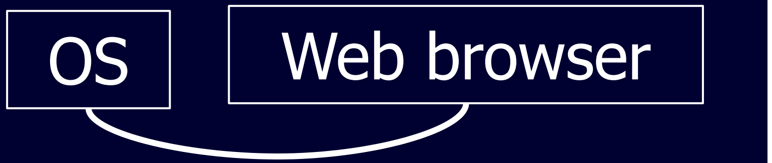


OS

Web browser

| OS | | Web browser | | |
|---|---|---|---|---|
| 7 | Vista | IE | Chrome | Firefox |
| ● | | ● | | |
| ● | | | ● | |
| ● | | | | ● |
| | ● | ● | | |
| | ● | | ● | |
| | ● | | | ● |

*Classification Tree*



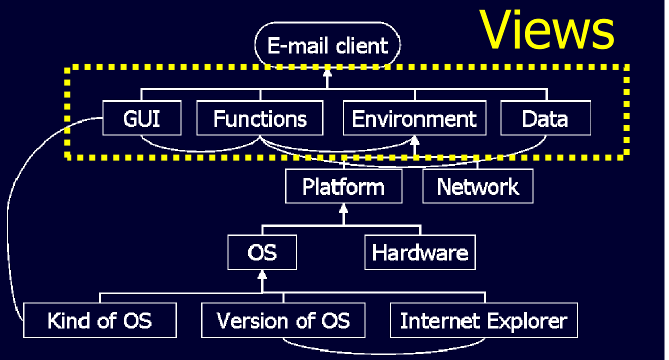
OS

Web browser

*Viewpoint diagram*

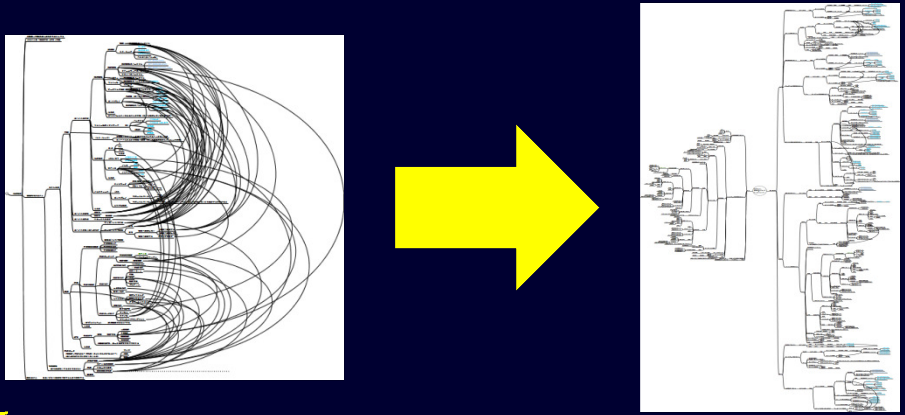
# TRA: Test requirement analysis

- To extract, organize and understand test requirements
  - Requirements from customers to achieve
    - Functional requirement, non-functional requirement, business goals etc.
  - Constraints to achieve requirement from customers
    - Requirement of test project management such as efforts, costs etc.
    - Test tools and/or methods directly requested by customer especially
  - Information of current quality of the test item
    - Ex) bugs which were detected in prior reviews
- To create a test requirement model on viewpoint diagram
  - Extract test viewpoints from test requirements
  - Detail test viewpoints and connect parent viewpoint and child viewpoints
  - Extract interaction relationships and connect viewpoints
  - Top-level viewpoints are most important for grasping a big picture, called "View"



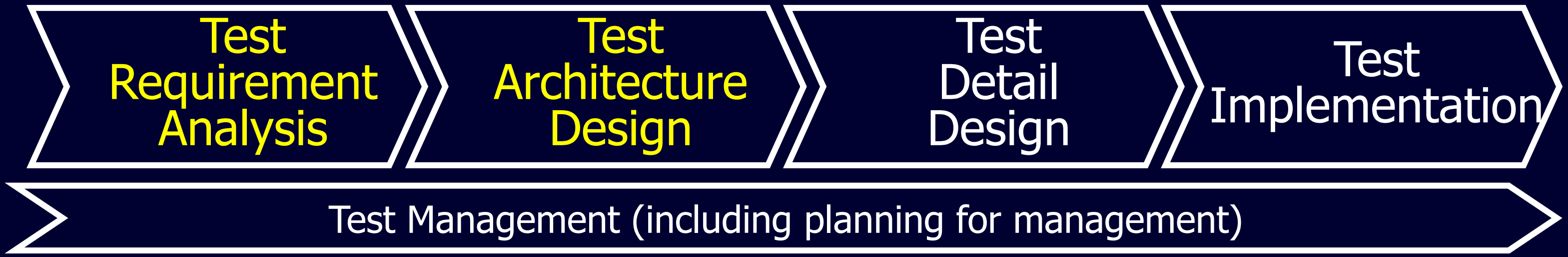
# Refinement of a test requirement model

- You can refine a test requirement model to make it clear and easy to understand
  - To detail viewpoints step by step to exhaust / list all test conditions
  - To move, divide or rename viewpoints if necessary
  - To check non MECE viewpoints in each layer and re-organize them as MECE
    - MECE: Mutually Exclusive and Collectively Exhaustive
  - To check whether brotherhood viewpoints have the same stereotypes of hierarchy connections
  - To check whether interactions would be better to change viewpoints



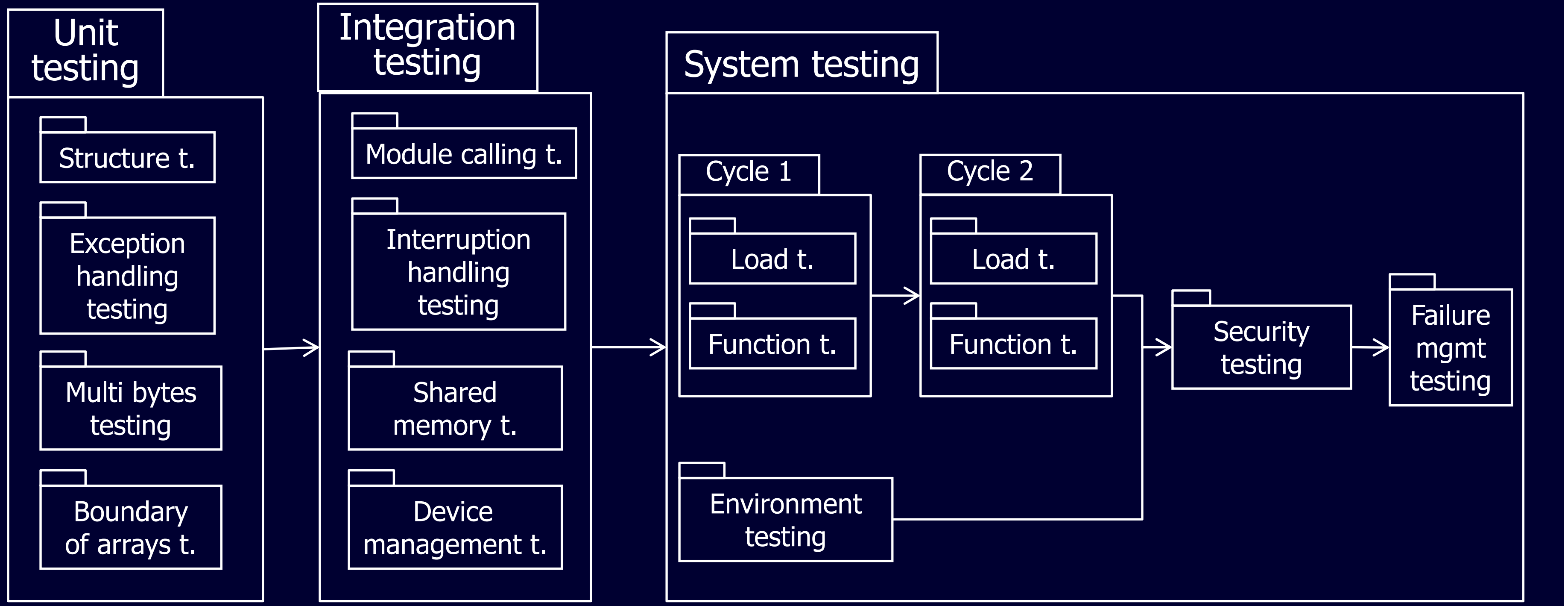
# VSTeP

- VSTeP(Viewpoint-based Software Test Engineering Process) is a generic test engineering process model focusing on test viewpoint
  - You can stress upper phase of test engineering process such as test requirement analysis and test architecture design which tend to be negligent
  - VSTeP drives you to good test suite, good review for test design, accumulation of knowledge and experience on testing
  - Reuse and improvement will be easy because you can do reverse-engineering of your past (unorganized) test suite
  - NGT (Notation for Generic Testing) is a made-in-Japan notation for Test Requirement Analysis and Test Architecture Design
    - » Modeling skill like object-oriented design is essentially necessary


Test Requirement Analysis
Test Architecture Design
Test Detail Design
Test Implementation
Test Management (including planning for management)

# TAD: Test Architectural Design using Test Containers

- Test architecture is a big picture of test suite
  - It is easy to grasp a big picture in test container level for large and complicated testing
  - Several viewpoints can be packed into a “test container”
  - Test containers can be test levels, test types and test cycles



Unit testing

Structure t.

Exception handling testing

Multi bytes testing

Boundary of arrays t.

Integration testing

Module calling t.

Interruption handling testing

Shared memory t.

Device management t.

System testing

Cycle 1

Load t.

Function t.

Cycle 2

Load t.

Function t.

Environment testing

Security testing

Failure mgmt testing

# Test containers are tasks or not?

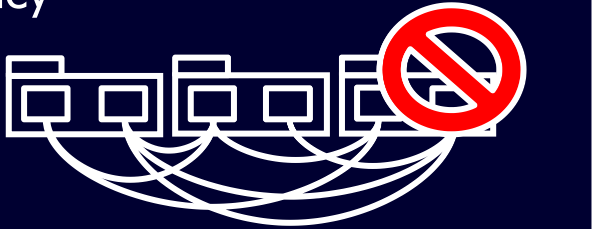
- ISTQB defines a test type as:
  - *a group of test activities aimed at testing a component or system focused on a specific test objective, i.e. functional test, usability test, regression test etc*.
- ISTQB defines a test level as:
  - *a group of test activities that are organized and managed together. A test level is linked to the responsibilities in a project. Examples of test levels are component test, integration test, system test and acceptance test*
- *ISO/IEC/IEEE 29119 defines a test sub-process as:*
  - *test management and dynamic (and static) test processes used to perform a specific test level (e.g. system testing, acceptance testing) or test type (e.g. usability testing, performance testing) normally within the context of an overall test process for a test project*
- *ISO/IEC/IEEE 29119 defines* a test level and test type as:
  - *a specific instantiation of a test sub-process*.

# Differences between UTP and NGT

- UTP has a broader scope while NGT focuses on just test suite architecture
  - UTP can describe test system architecture and test suite architecture
- UTP can potentially have a descriptive power as strong as NGT in test suite architecture
  - TestContext in UTP is similar to a test viewpoint or a test container in NGT
  - Concretion is necessary because TestContext is too generic
- There is no example on test suite architecture in UTP
  - I'm wondering UTP can't describe these or not:
    - » Hierarchy of TestContext
    - » Stereotype of combination
    - » Model on test container level
  - Even if so, I hope UTP will be updated to describe those
    - » NGT will go "UTP test suite architecture profile" ☺

# No Guides for good TAD

- Some characteristics, principles and patterns for software can be applied as guides for good TAD
  - “Quality Characteristics” for software are already available such as ISO/IEC 25000s
    - » Functional Suitability / Performance efficiency / Compatibility / Usability / Reliability / Security / Maintainability / Portability
  - Design principles and design patterns for software design are also major
    - » Coupling / Cohesion / Encapsulation / Responsibility
    - » Design patterns such as MVC, singleton
- This presentation introduces 10 design principles for Test Architecture
  - Coupling / Cohesion
  - Maintainability / Automatability
  - Circumstance consistency / Development consistency
  - Describability
  - Design direction / Design positiveness
  - Execution velocity consistency



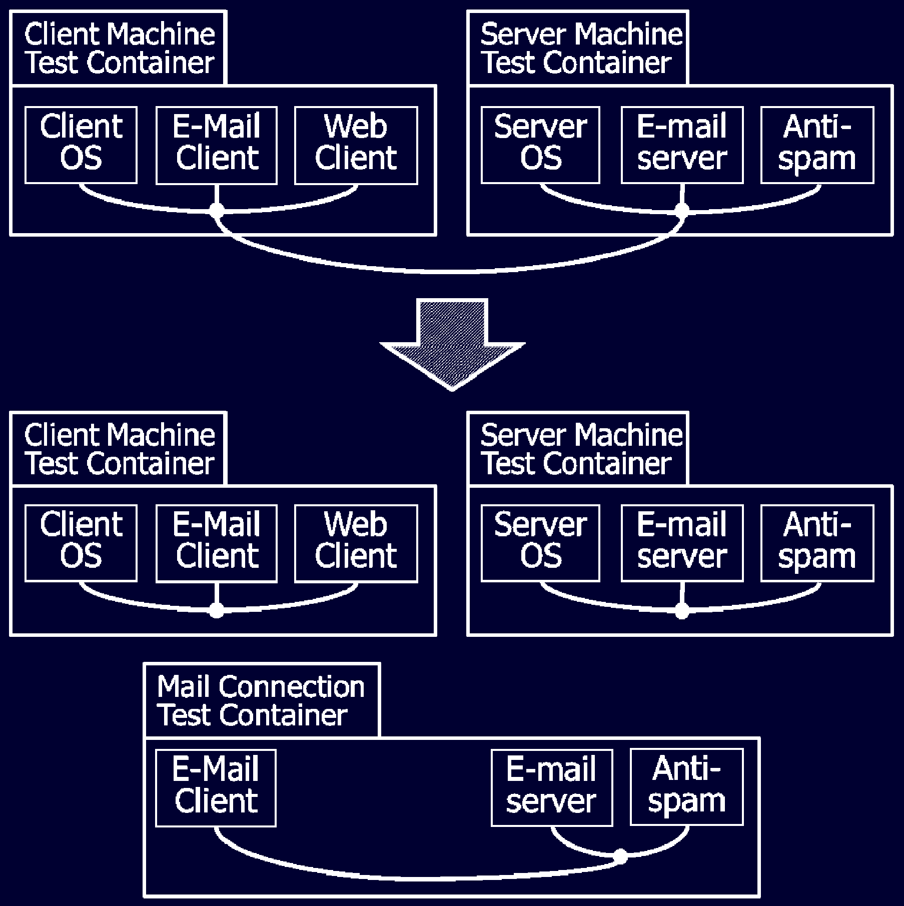
# 10 Design Principles for Test Architecture

1. Coupling
2. Cohesion
3. Maintainability
4. Automatability
5. Circumstance consistency
6. Development consistency
7. Describability
8. Design direction
9. Design positiveness
10. Execution velocity consistency

These are not "manual"

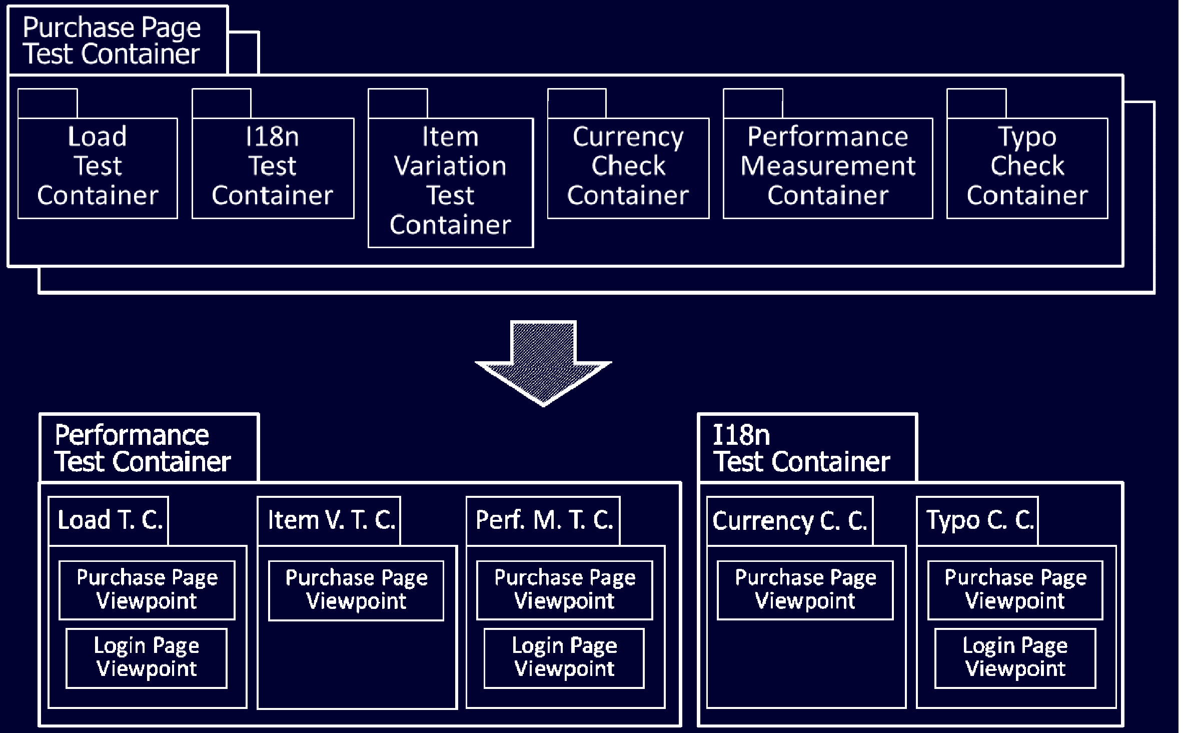
# 1. Coupling

- Test architect should reduce coupling
  - If relationships among test containers unnecessarily increase, test design will be more complicated and difficult to understand
  - Responsibilities or test objectives are properly assigned in the lower test architecture
  - Test designer can easily design combinatorial testing for each test containers in the lower test architecture



# 2. Cohesion

- Test architect should increase cohesion
  - If test types or viewpoints are disorderly grouped, test design will be more confusable and difficult to understand such as in the upper test architecture applying the page object pattern.
  - Responsibilities or test objectives are properly assigned in the lower test architecture
  - Test designer will not design currency check or typo check test together with load test


Purchase Page Test Container

Load Test Container

I18n Test Container

Item Variation Test Container

Currency Check Container

Performance Measurement Container

Typo Check Container

Performance Test Container

Load T. C.

Purchase Page Viewpoint

Login Page Viewpoint

Item V. T. C.

Purchase Page Viewpoint

Perf. M. T. C.

Purchase Page Viewpoint

Login Page Viewpoint

I18n Test Container

Currency C. C.

Purchase Page Viewpoint

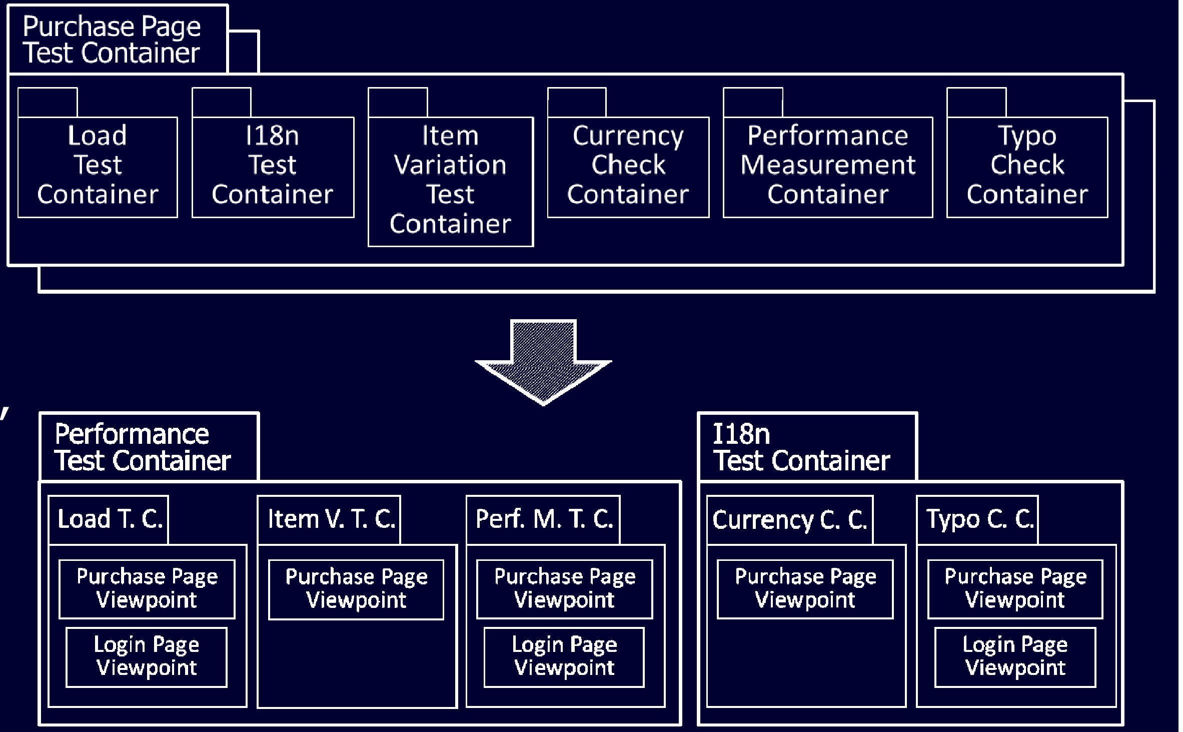
Typo C. C.

Purchase Page Viewpoint

Login Page Viewpoint

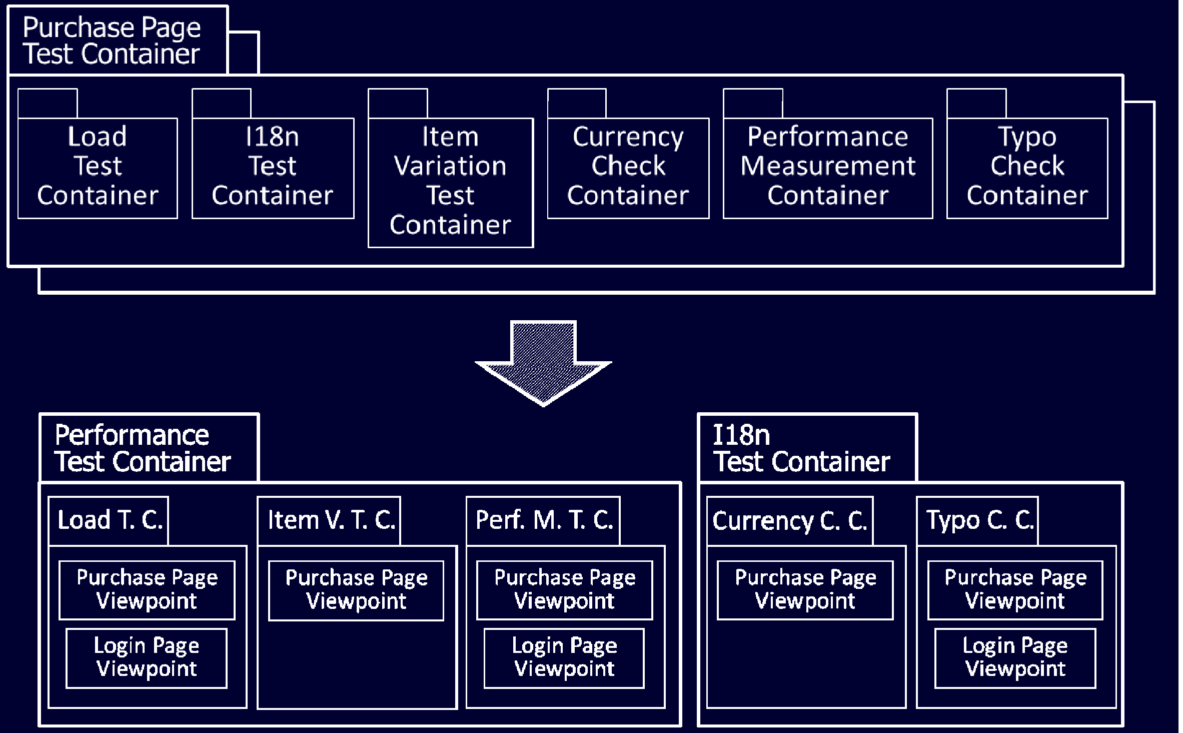
# 3. Maintainability

- Test architect should consider and increase maintainability
  - As test design itself needs frequent change, maintenance and enhancement like software, it could be better to separate unstable part and stable part
    - » Web applications often need performance enhancement and its test
  - Test designer can easily specify where to be changed
    - » It requires longer, wider and broader perspective
  - Test suite has its own "quality characterisitcs"
  - Good maintainability leads productine engineering of test suite



# 4. Automatability

- Test architect should consider and increase automatability
  - Automatable test viewpoints should be isolated into the same test container
    - » Performance test can be automated with test tools
    - » I18n test needs human check with various nationality
  - Test designer can easily isolate tests to be automated
    - » As without isolation efficiency of automation will be left low, managers will decide to invest no money into automation



# What is “test level”?

- “Test level” is a mysterious word…
  - Unit, integration, system and user acceptance are typical test levels
  - Test managers usually consider them as a given or common knowledge
    - E.g. according to a company standard or textbooks
    - But definition of test level is rather ambiguous…
  - Test architect has to design test levels for large-scale and complicated SUTs
    - Modern test standard doesn’t define specific test levels as a given
    - In agile development another kind of test levels might be possible
    - design of test levels should follow design principles in test architecture
- We need design principles for test-level-like containers

  5. Circumstance consistency
     - Test architect should identify and assemble test viewpoints which need specific environment into each test container
  6. Development consistency
     - Test architect should make test bases in a test container consisitent with the same development phase

# 7. Describability

- Test architect should isolate non-descriptive test viewpoints
  - test viewpoints should be arranged into test containers according to how detail each test viewpoint needs to be described
- Non-descriptive tests are so important as descriptive tests
  - Non-descriptive tests:
    - » Exploratory testing, user experience testing, penetration testing etc.
  - Non-descriptive tests essentially works by learning and creativity
- Even in non-descriptive tests, test viewpoints should be specified, designed and isolated
  - Charters for exploratory testing, market segments for user experience testing and threats for penetration testing are all test viewpoints and to be designed
  - If non-descriptive tests are mixed with descriptive tests, learning and creativity will be frustrated

# 8. Design Direction

- Test architect should balance the design directions
  - Design has generally two directions: forward design and backward design
    - » Also called deductive/inductive or direct/inverse design
    - » FD is a design after investigation of specifications which the designs are based on
    - » BD is a design after investigation of behaviours which the designs result in
    - » When a test case forms “if X is input, Y will be output”,
      FD considers X first and derives Y, while BD considers Y first and explores X
  - Forward test design derives test cases from test conditions
    - » E.g. functional testing and load testing
    - » Forward test design tends to be too simple or superficial
  - Backward test design derives test cases from expected results or checkpoints
    - » E.g. performance testing and usability testing
    - » Backward test design tends to be too difficult to design
      or to include unintended omissions

# 9. Design positiveness

- Test architect should balance positive design and negative design
  - Design has generally two opposite thinking manners: positive design and negative design
    - » Positive design is a design to accomplish all reqs or to cover all specifications
    - » Negative design is a design to avoid any problem or to detect bugs
  - Positive test design tends to be too exhaustive due to a lot of detail or combinatorial test cases, and are all for checking
  - Negative test design tends to be unable to assure any quality explicitly, and needs too much efforts due to exhaustive exploratory testing.

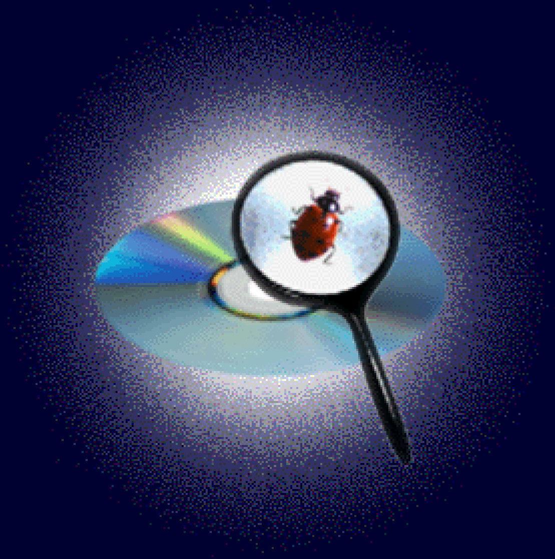
# 10. Execution velocity consistency

- Test architect should arrange “rhythm” of test team
  - Test team has some rhythm in execution of test cases
    - » Some sub teams can have quick rhythm while others can have slow rhythm
  - Disharmonious rhythms in the same team will frustrate the team members
    - » Unintended wait or unexpected rush tends to irritate them and to increase mistakes
  - Rhythms are derived from test execution velocity
    - » Testing good quality SUT makes quick rhythm with high motivation
    - » Testing poor quality SUT makes slow rhythm with deep consideration on exploring more bugs, specifying locations of bugs more accurately and writing better bug reports
  - Allocation of test containers according to execution velocity can lead good test architecture

# Conclusion

- Test (suite) Architecture Design is important for large-scale and complicated SUT
  - NGT is a notation more focusing on TAD than UTP
  - In NGT, test containers are fundamental component of test architecture
- No guides for good TAD
  - Quality characteristics, design principles and patterns can be applied to guides for good TAD
- This presentation have introduced 10 design principles for test architecture design
  - Coupling / Cohesion / Maintainability / Automatability / Circumstance consistency / Development consistency / Describability / Design direction / Design positiveness / Execution velocity consistency
- Quantitative research will be necessary for future

# Thank you for your kind attention



NISHI, Yasuharu
http://qualab.jp/
Yasuharu.Nishi@uec.ac.jp

# Profile – Dr. NISHI, Yasuharu

Assistant professor:
the University of Electro-Communications, Japan
(also providing consultancy service to industry on testing and TQM)
President:
Association of Software Test Engineering, Japan (ASTER)
President:
Japan Software Testing Qualifications Board (JSTQB)
National delegate:
ISO/IEC JTC1/SC7/WG26 Software testing
Founder:
Japan Symposium on Software Testing (JaSST)
Founder:
Testing Engineers' Forum (Japanese community on software testing)
Vice chair:
SQiP/Software Quality Committee of JUSE (promoting organization of TQM)
(SQiP has published the book of "SQuBOK: Software Quality Body of Knowledge" and is operating engineer certification on software quality)
Research interest:
Software testing, software quality/TQM , embedded software engineering, software process improvement, software project management, system safety