Effective Development and Practice of Automatic Testing based on TTCN3

Wang Shaofeng, Luo Fuliang, Yan Longguo and Zhang Yanwei
Content

- Background
- Performance
- Test Framework
- Automation Resources
- Conclusion
Difficulties and Challenges on Automatic Testing

Product requirement
Software becomes more complex

Tester requirement
Testing should be more simple

Agile requirement
Feedback should be quickly

How do Automation match Business?

• Hard for TSE&TEE to master Automation skills and join business testing.
• It takes so much time to test a version, but we need feedback quickly.
• TTCN3 can’t satisfy the test case linearization.
The Concept and Goal of TEP

- Effective performance.
- Concurrent performance.
- Testing resource detachable
- Satisfy the requirement for different tester (TE/AE/SE) to manage automation resource.

Simplification & Standardization
- Easy to learn & use, satisfy the requirement of business.

Syntax
- Automatization match Business

Resource
- Performance

TEP (Test Execute Platform)
more than a TTCN3 Execution Platform, also offers a workbench to make an effective management for testing resource. Support CI testing, regression testing etc.

Core value of TEP:
Make automation match business easier.
Content

- Background
- Performance
- Test Framework
- Automation Resources
- Conclusion
Performance

Requirement

- Business becomes more and more complex
- Testsuits become more and more huge (thousands of function/testcase)
- performance & stability always to be a top-level requirement

How to make an effective management for testing resource? How to upgrade the performance?
A Lightweight Executor

A lightweight executor based on language transform (TTCN3 to C++)

>> Easy to manage testing resource based on binary library

>> Easy to obtain concurrent performance
Concurrent Performance

![Diagram showing Concurrent Performance with Multi-Process and Multi-Thread]
Test Framework

- More demand for Automation Engineer

  Be familiar with a language (TTCN3)?
  knowledge on Application & Business detail?
  Some special testing skills?
  
  
  Actually, testing should be easy and simple.
  A standard way to generate effective test case?
Overview

Precondition & Postcondition
> most like other TFL

Teststep
> test framework body

Series & Parallel
> like thread and thread pool

Sync
> block semantic, like barrier.

More than a TFL,
also offer a running model like concurrent action
Test Framework

```javascript
var class_MS xPhone
precondition
{
}

teststep
{
    parallel
    {
        series
        {
            sync;
        }
    }
}

postcondition
{
}
```

Test framework base on class variable.

What’s class?
Why need a new concept: class

How to reuse a runs on function in other components?

Function func1() runs on Comp1 …

//Choice 1: using component compatibility
type component CompA
{
    Now CompA is compatibility with Com1
    some Comp1’s content;
}

//Choice 2: using component extends
type component CompB extends Comp1
{
    Now CompB is extends from Comp1
    .......
}

What an image When component relationship is very complex?

type component CompC extends Comp1, Comp2, Comp3, Comp4, Comp5 …
{
    Do you face any situation like CompC?
}

Implementation Detail

A new type more like OO: class (vs component)

Old style:

```haskell
type component CompC extends Comp1,Comp2,Comp3,Comp4,Comp5 ...... { ... }
```

New style:

```haskell
type class CompC
{
    Function Reference + Attribute;
}
```

//Function Reference different component
Function func1() runs on Comp1 return integer;
Function func5() runs on Comp5 return integer;

//Member Attribute
var charstring strVersion := “V1.0.0”;
var charstring strIP := “10.78.75.69”;

Usage: More like a component composite style

```haskell
var CompC myCompC;
myCompC.strIP := “10.78.75.100”;
myCompC.func1();
```
Requirement of Automation Resource

- **Concept**
  - Automation Resource
    - Testcase, ActionWord, Data(environment data), logic, class etc.

- **Defect**
  - Automation Resources is too complex to design.
  - Not a easy work for TSE/TEE/TAE, testing and business not always to be the same.

- **Requirement**
  - Automation Project need to be divided into Automation Resource independently.
    - TSE take charge of planning the Automation Resources(Class).
    - TEE take charge of developing Logics & Testcases.
    - TAE take charge of developing ActionWords.
What Automation Resources to be

Automation Project to be divided into resources independently。Less change when the version update.

SE&TE who understand business are suffice for Automation。
Testing Resource Management

Class mapping NE model. The functions & attributes mapping NE’s AWs & parameters. New Class can beParsed from AW-package automatically.

Easy Configuration. The TTCN3 templates instead of NE’s para templates which created by parsing asn.1 syntax.
Graphics Configuration of Class

The topological graphics mapping the hardware network.

Better reusable and shareable template.
The Main Workbench

Testcases Tree

Logics and ActionWords

Symbol View

Easy to learn and use
The Objects, which planed by SE at first, relative to the new Testcase will be created automatically.

Powerful running model
precondition / postcondition / parallel / series / sync etc.
Content

- Background
- Performance
- Test Framework
- Automation Resource
- Conclusion
The Benefits

Automation match Business

- High quality
  Actionword/Class based on binary library;
  Testcase based on framework;
  Less change while the version update.

- Less demand for customer
  Based on framework workbench
  Effective & Concurrent performance

- Effective management for Automation.
  Resource detachable & shareable.
  Based on binary library (not source code)
Thank You

www.huawei.com