

Tesztelés a fejlesztés különböző fázisaiban

Majzik István, Micskei Zoltán

<http://www.inf.mit.bme.hu/>

1

Tartalom

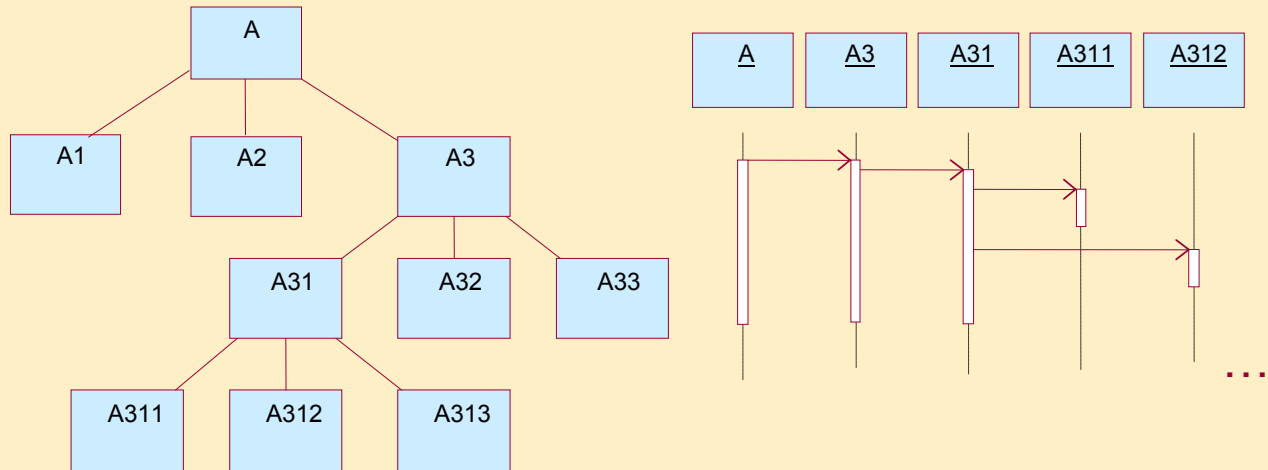
- Modul / unit tesztelés
- Integrációs tesztelés
- Rendszer tesztelés
- Elfogadás tesztelés

- Tesztek leírása: U2TP

2

Modul / unit tesztelés

- Modul / Unit:
 - Logikailag egy egységként kezelhető elem
 - Jól meghatározott interfésszel rendelkezik
 - OO fejlesztés: Osztály (csomag, komponens is lehet)
- Modul hívási hierarchia (ideális eset):



3

Miért van szükség modultesztelésre?

- Cél: Hibák gyors kijavítása a fejlesztés után (legalacsonyabb szinten)
 - Integrációs fázis hatékonyabb, ha a modulok teszteltek
 - A modul fejlesztője által leggyorsabb a javítás
- Modulok külön-külön tesztelhetők
 - Komplexitás kézbe tartható
 - Hibák helye egyszerűbben felderíthető, javítás olcsóbb
 - Párhuzamosítható folyamat
- Unit teszt jellegzetességek
 - A kód egy-egy speciális funkcióját ellenőrzi
 - „Szerződés” ellenőrzése a modul működéséről
 - Példaként szolgálhat az egység használatához
 - (Nem feltétlenül automatikus)

4

Unit teszt keretrendszerek

- Unit tesztek: Gyakran kell futtatni
 - Fejlesztés közben, kód változásakor (pl. refactoring)
 - Környezet változása esetén
 - Épp ezért gyorsnak kell lennie
- Jó eszköztámogatás
 - Keretrendszerek (JUnit, xUnit, TestNG, ...)
 - Támogatás az IDE-kben (Eclipse, Netbeans, ...)
- Általában egyszerű funkcionalitású eszközök
 - Tesztek és ellenőrzések megadása
 - Tesztek futtatása
 - Eredmény megjelenítése (red-green)

5

Egy egyszerű JUnit teszt

```
public class ListTests{  
    List list;    // SUT  
  
    @Before public void setUp(){  
        list = new List();  
    }  
  
    @Test public void add_EmptyList_Success(){  
        list.Add(1);  
        assertEquals(1, list.getSize());  
    }  
}
```

The diagram illustrates the components of a JUnit test class. Three yellow callout boxes with pointers identify specific parts of the code:

- Test előkészítés (fixtures)**: Points to the `@Before` annotation on the `setUp()` method.
- SUT meghívása**: Points to the `list.Add(1);` line inside the `add_EmptyList_Success()` method.
- Ellenőrzés**: Points to the `assertEquals(1, list.getSize());` line inside the `add_EmptyList_Success()` method.

6

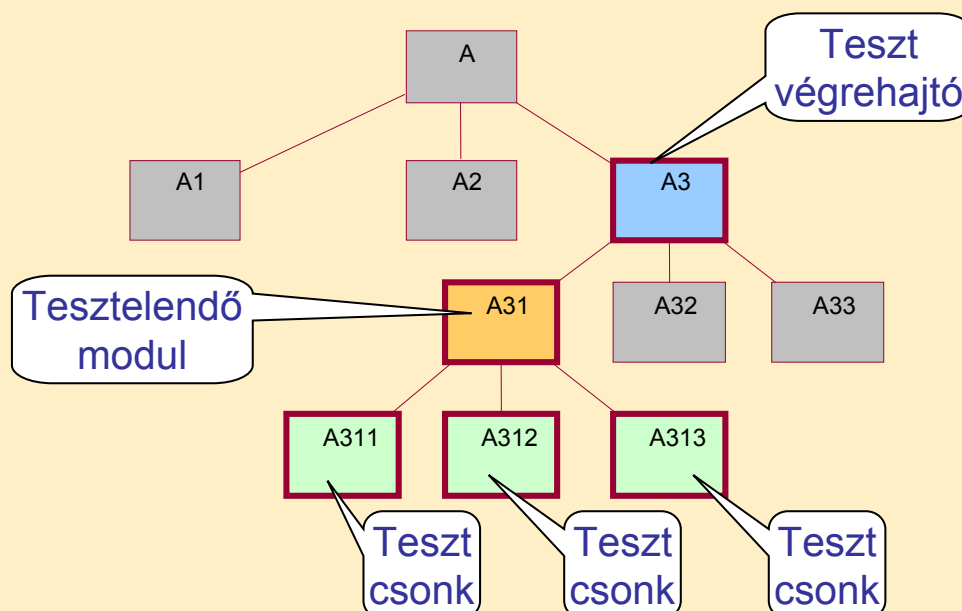
Jó unit teszt jellegzetessége

- Egyszerű, megbízható
 - Nincs benne bonyolult logika (pl. ciklusok)
- Egy teszt egy dolgot vizsgál
- Ez is jó minőségű kód
 - Duplikáció elkerülése
 - Jól áttekinthető, módosítható
- Tesztek függetlenek egymástól
- Ellenőrzések nincsenek túlspecifikálva
- ...

7

Modulok izolációs tesztelése

- Modulok egyenként, elszigetelten tesztelve
- Teszt végrehajtó és teszt csomópontok szükségesek



8

Probléma: Függőségek kezelése

- Mi tekinthető függőségnek?
 - Bármilyen, amivel
 - a SUT együttműködik,
 - de nem hozzá tartozik (nem közvetlenül befolyásolja)
- Példák:
 - Másik modul
 - Fájrendszer hívás
 - Dátum lekérdezése
 - ...

9

Példa: Nehezen tesztelhető

```
public class PriceService{  
    private DataAccess da = new DataAccess();  
  
    public int getPrice(String product)  
        throws ProductNotFoundException {  
        int p = this.da.getProdPrice(product);  
        if (p == null)  
            throw ProductNotFoundException;  
  
        return p;  
    }  
}
```

Nehezen
helyettesíthető,
ha pl. speciális
teszt szükséges

10

Példa: SUT tesztelhetővé tétele

```
public class PriceService{
    private IDataAccess da;

    public PriceService(IDataAccess da){
        this.da = da;
    }

    public int getPrice(String product)
        throws ProductNotFoundException {
        int p = this.da.getProdPrice(product);
        if (p == null)
            throw ProductNotFoundException;

        return p;
    }
}
```

Implementáció
átadása pl. a
konstruktorban

11

Példa: Unit tesztek a SUT-hoz

```
public class PriceServiceTests{
    @Before init(){
        DataAccessStub das = new DataAccessStub();
        das.Add("A100", 50);
        PriceService ps = new PriceService(das);
    }

    @Test public void testSuccessfulPriceQuery(){
        int p = ps.getPrice("A100");
        assertEquals(50, p);
    }

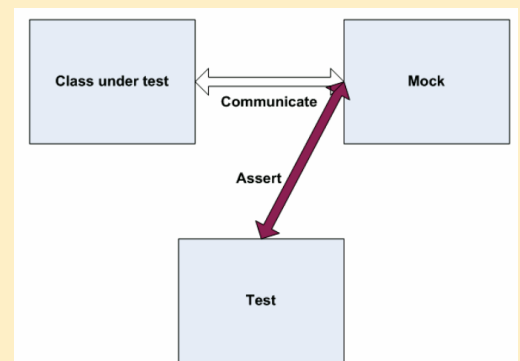
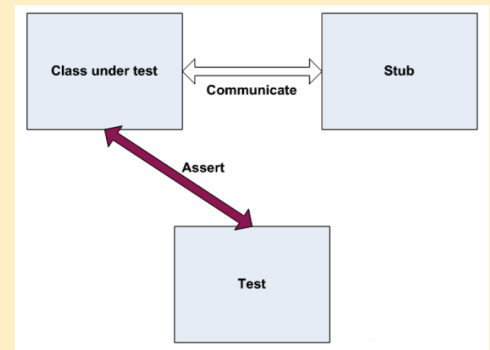
    @Test(expected = ProductNotFoundException.class)
    public void testNotExistingProduct(){
        ps.getPrice("notExists");
    }
}
```

Tesztben egy csonk
használata

12

Stub, Mock, Dummy, Fake, ... objektumok

- Sokféle technika a helyettesítésre
 - Eltérő elnevezések (ld. xUnit Patterns)
 - Összefoglaló név: **Test double**
 - Helyettesítő elem teszteléshez
- Stub (csonk)
 - SUT **állapotának** ellenőrzésére
 - Rögzített válaszok adott hívásokra
- Mock
 - SUT **interakcióinak** ellenőrzésére
 - Elvárt és ellenőrzött viselkedés
- Dummy
 - Nem használt („kitöltő”) objektum
- Fake
 - Működő, de nem az „éles” objektum



Isolation frameworks

- Csonkok és mockok kézi elkészítése nehézkes
 - Sok manuális munka
 - Karbantartása időigényes lehet
- Keretrendszerek:
 - Osztály/interfész leírás alapján
 - Dinamikus csonkok vagy mockok készítése
- Példák:
 - JMock, Mockito, Rhino Mocks, Typemock...

Példa: Mock használata (Mockito)

```
public class PriceServiceTests{  
    @Before init(){  
        DataAccess mockDA = mock(DataAccess.class);  
        PriceService ps = new PriceService(mockDA);  
    }  
  
    @Test public void testSuccessfulPriceQuery(){  
        when(mockDA.getProdPrice(„A100“)).thenReturn(50);  
  
        int p = ps.getPrice(„A100“);  
  
        verify(mockDA, times(1)).getProdPrice(„A100“);  
    }  
  
    ...  
}
```

Megfelelő típusú
test double kérése

„Működési szabályok”
megadása

Milyen működést kellett
megfigyelnie

15

Megéri ilyen unit tesztek készíteni?

- Egy példa pilot projekt számai:

Stage	Team without tests	Team with tests
Implementation (coding)	7 days	14 days
Integration	7 days	2 days
Testing and bug fixing	Testing, 3 days Fixing, 3 days Testing, 3 days Fixing, 2 days Testing, 1 day Total: 12 days	Testing, 3 days Fixing, 1 day Testing, 1 day Fixing, 1 day Testing, 1 day Total: 8 days
Overall release time	26 days	24 days
Bugs found in production	71	11

Forrás: Roy Osherove, The Art of Unit Testing, 2009.

16

Olvasnivalók

- Martin Fowler: Mocks Aren't Stubs, 2007,
 - URL: <http://martinfowler.com/articles/mocksArentStubs.html>
- Roy Oshero: The Art of Unit Testing: With Examples in .Net.
 - Manning Publications; 1st edition (June 3, 2009),
 - URL: <http://artofunittesting.com/>
- Brett L. Schuchert: Mockito.LoginServiceExample
 - URL: <http://schuchert.wikispaces.com/Mockito.LoginServiceExample>

17

Tartalom

- Modul / Unit tesztelés
- Integrációs tesztelés
- Rendszer tesztelés
- Elfogadás tesztelés
- Tesztek leírása: U2TP

19

Integrációs tesztelés

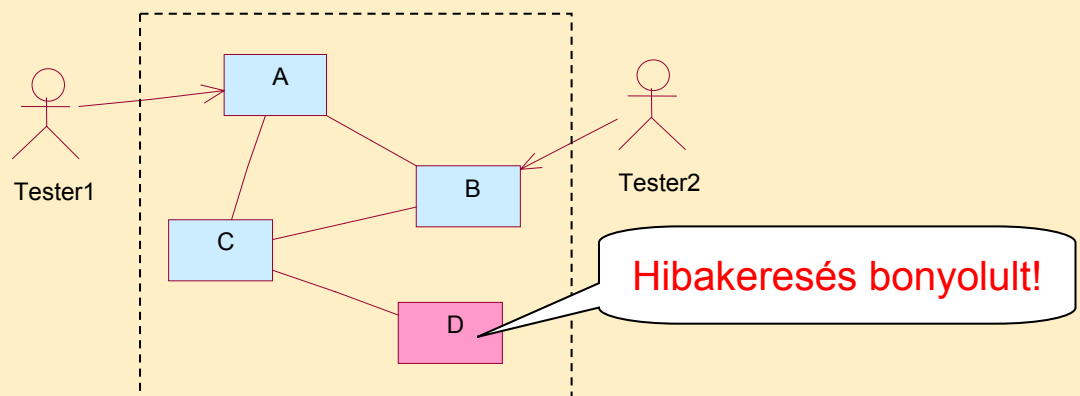
Modulok együttműködésének ellenőrzése

- Kapcsolódási interfészek tesztelése
 - A rendszer annak ellenére hibás lehet, hogy minden modul egyenként hibátlan!
- Módszerek:
 - Funkcionális tesztelés: **Forgatókönyvek** tesztje
 - Ez sokszor a specifikáció része (scenario)
 - (Strukturális tesztelés csak modulszinten!)
- Megközelítés:
 - “Big bang”: minden modult egyszerre integrálni
 - **Inkrementális**: egyenként összerakni a modulokat

20

“Big bang” integrációs tesztelés

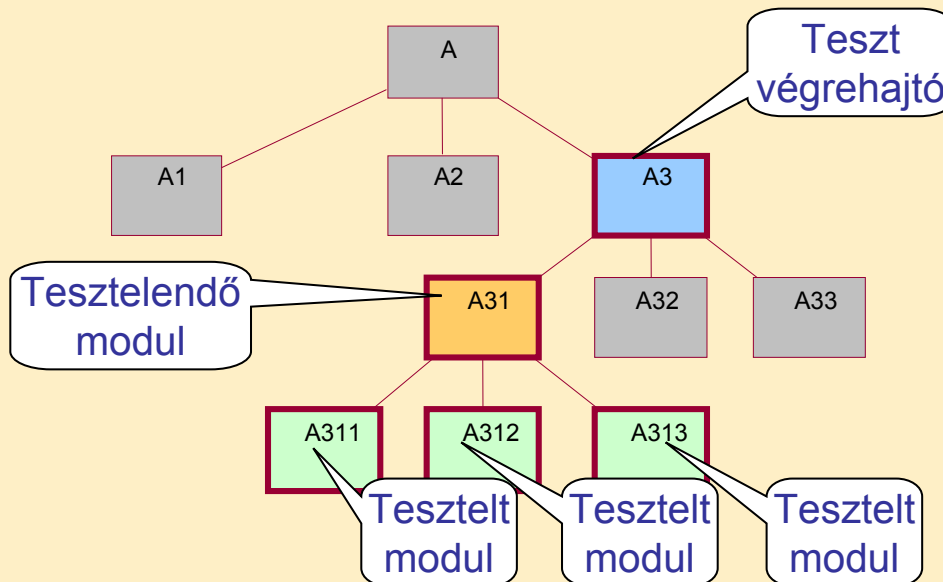
- Tesztelés a **külső interfészek**en keresztül
- Külső teszt végrehajtó
- Rendszer funkcionális specifikáció alapján történik
- Modul módosítás: Újratesztelés
- Kis rendszerek esetén alkalmazható



21

Alulról felfelé történő integrációs tesztelés

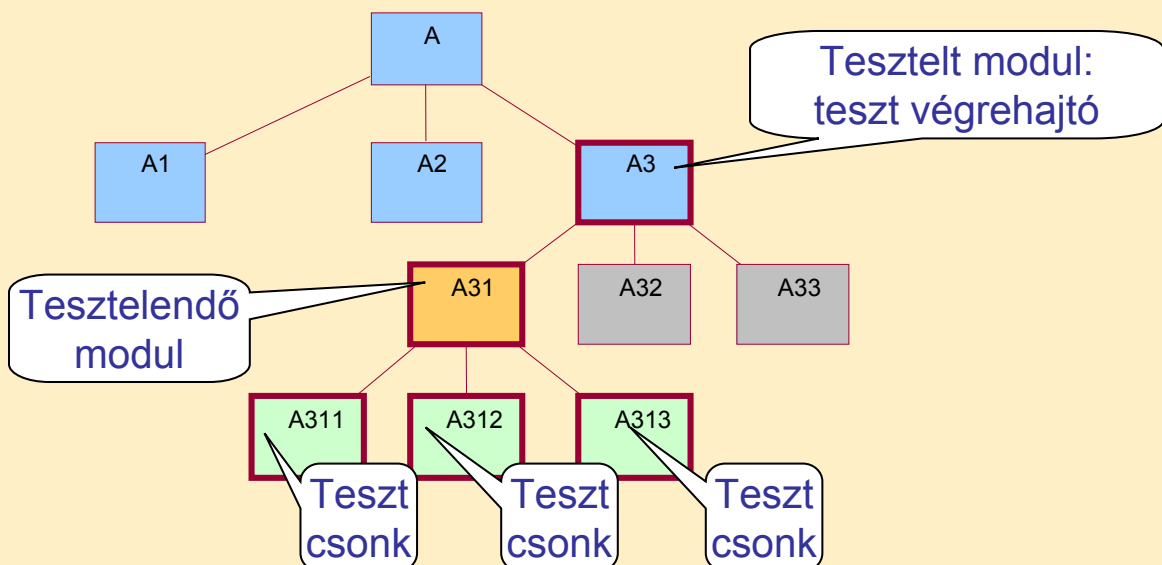
- Tesztelendő modul a már **teszteltek**et használja
- **Teszt végrehajtó** szükséges
- Integrációval párhuzamosan megtehető
- Modul módosítás: felette lévők tesztjére hatással van



22

Felülről lefelé történő integrációs tesztelés

- Modulok a **hívó modulokból** kerülnek tesztelésre
- **Csonkok** helyettesítése tesztelendő modulokkal
- Erősen követelmény-orientált ("fentről" tesztelünk)
- Modul módosítás: alatta lévők tesztelését módosítja



23

Felülről lefelé vs. alulról felfelé

- Felülről lefelé

- + Követelmény-orientált, ellenőrzéshez jól illeszkedő
- + Hamar összeállhat egy demonstrálható „szkeleton”
- Csonkok készítése általában nehezebb
- Teszt bemenetek távol lehetnek az épp integrálandó modultól

- Alulról felfelé

- + Integráció orientált, konstruktívabb
- + Könnyebb megfigyelni és irányítani a teszteket
- A rendszer maga csak a legvégén áll össze

24

Futtató rendszer integrációja

- Motiváció:

Nehéz csonkokat írni a futtató rendszerhez

– Pl. OS, J2EE konténer, ...

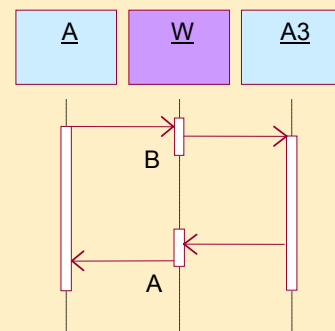
- Stratégia:

1. Alkalmazás modulok integrációja **felülről lefelé**, a futtató rendszer szintjéig
2. Futtató rendszer **alulról felfelé** történő tesztelése
 - Funkciók izolációs tesztje (ha szükséges)
 - „Big bang” tesztelés az alkalmazás hierarchia legalsó rétegével
3. Alkalmazás és futtatórendszer **integrációja**, a felülről lefelé történő tesztelés befejezése

25

Eszközök az integrációs teszteléshez

- Wrapper (csomagoló) kódrészletek
 - „before” wrapper: A hívás végrehajtása előtt
 - Paraméterek vizsgálata
 - Hívási szekvencia ellenőrzése
 - „after” wrapper: A hívás végrehajtása után
 - Visszaadott érték ellenőrzése vagy módosítása
 - „replace” wrapper: A hívott helyettesítése
 - Teszt csonk megvalósítás



26

Tartalom

- Modul / Unit tesztelés
- Integrációs tesztelés
- Rendszer tesztelés
- Elfogadás tesztelés
- Tesztek leírása: U2TP

27

Rendszertesztelés

Tesztelés a rendszerszintű specifikáció alapján

- Jellemzők:
 - Hardver-szoftver integráció után végezhető
 - Funkcionális tesztek +
nem-funkcionális jellemzők tesztje is!
- Kiemelhető:
 - Adat integritás vizsgálata
 - Felhasználói profil figyelembe vétele (terhelés)
 - Rendszer alkalmazhatósági korlátok megállapítása
(erőforrás-használat, telítődés)
 - Hibahatások vizsgálata

28

Rendszerteszt típusok



29

Tartalom

- Modul / Unit tesztelés
- Integrációs tesztelés
- Rendszer tesztelés
- Elfogadás tesztelés
- Tesztek leírása: U2TP

30

Elfogadás tesztelése (validációs tesztelés)

- Cél: Valóságos környezet hatásának tesztelése
 - Felhasználói elvárások figyelembe vétele:
Nem specifikált felhasználói elvárások is megjelennek
 - Váratlan eseményekre való reagálás:
Kis valószínűségű eseménykombinációk is megjelennek
- Időzítemi szempontok
 - Időzítések megfigyelése az adott környezetben
 - Korlátok ellenőrzése: Valósídejú monitorozás
- Környezeti szimuláció
 - Adott szituációk valóságban nem tesztelhetők
(pl. védelmi rendszerek)
 - Szimulátorokat is validálni kell

Összefoglalás: Tesztelési feladatok

1. Modul/unit tesztelés

- Izolációs tesztelés

2. Integrációs tesztelés

- „Big bang” tesztelés
- Top-down (felülről-lefelé) tesztelés
- Bottom-up (alulról-felfelé) tesztelés
- Futtató környezet integrációja

3. Rendszertesztelés

- Teljes rendszer együttes tesztelése

4. Validációs tesztelés

- Felhasználói elvárások tesztelése
- Környezeti szimuláció

32

Tartalom

- Modul / Unit tesztelés
- Integrációs tesztelés
- Rendszer tesztelés
- Elfogadás tesztelés
- Tesztek leírása: U2TP

33

U2TP: UML 2 Testing Profile (OMG, 2004)

- Able to capture all needed information for **functional black-box testing** (specification of test artifacts)
 - Mapping rules to TTCN-3, JUnit
- **Language** (notation) and **not a method** (how to test)

Packages (concept groups):

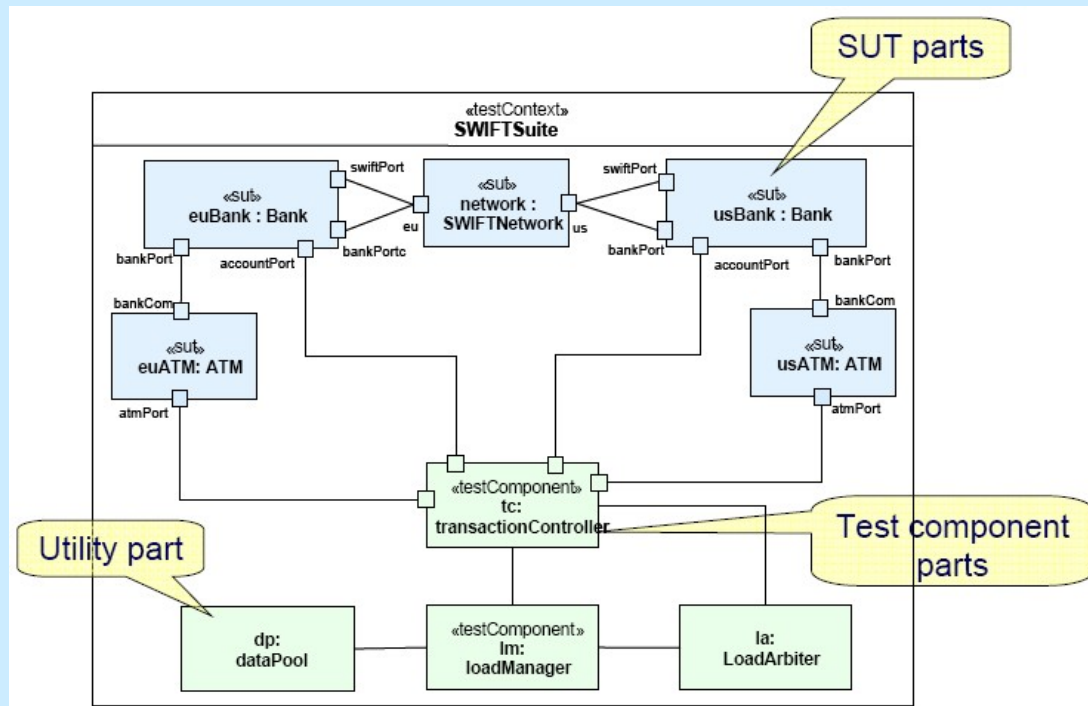
- **Test Architecture**
 - Elements and relationship involved in test
 - Importing the UML design model of the SUT
- **Test Data**
 - Structures and values to be processed in a test
- **Test Behavior**
 - Observations and activities during testing
- **Time Concepts**
 - Timer (start, stop, read, timeout), TimeZone (synchronized)

U2TP Test Architecture package

Identification of main components:

- **SUT: System Under Test**
 - Characterized by interfaces to control and observation
 - System, subsystem, component, class, object
- **Test Component:** part of the test system (e.g., simulator)
 - Realizes the behavior of a test case
(Test Stimulus, Test Observation, Validation Action, Log Action)
- **Test Context:** collaboration of test architecture elements
 - Initial test configuration (test components)
 - Test control (decision on execution, e.g., if a test fails)
- **Scheduler:** controls the execution of test components
 - Creation and destruction of test components
- **Arbiter:** calculation of final test results
 - E.g., threshold on the basis of test component verdicts

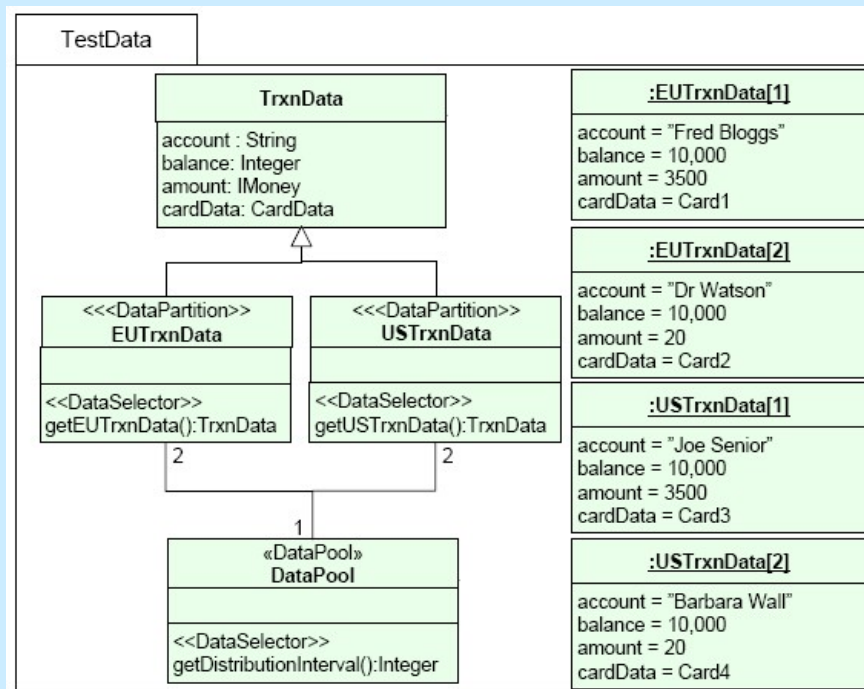
U2TP Test Architecture example



U2TP Test Data package

- Identification of **types and values** for test (sent and received data)
 - **Wildcards** (* or ?)
 - **Test Parameter**
 - Stimulus and observation
 - **Argument**
 - Concrete physical value
 - **Data Partition: Equivalence class** for a given type
 - Class of physical values, e.g., valid names
 - **Data Selector: Retrieving data out of a data pool**
 - Operating on contained values or value sets
 - **Templates**

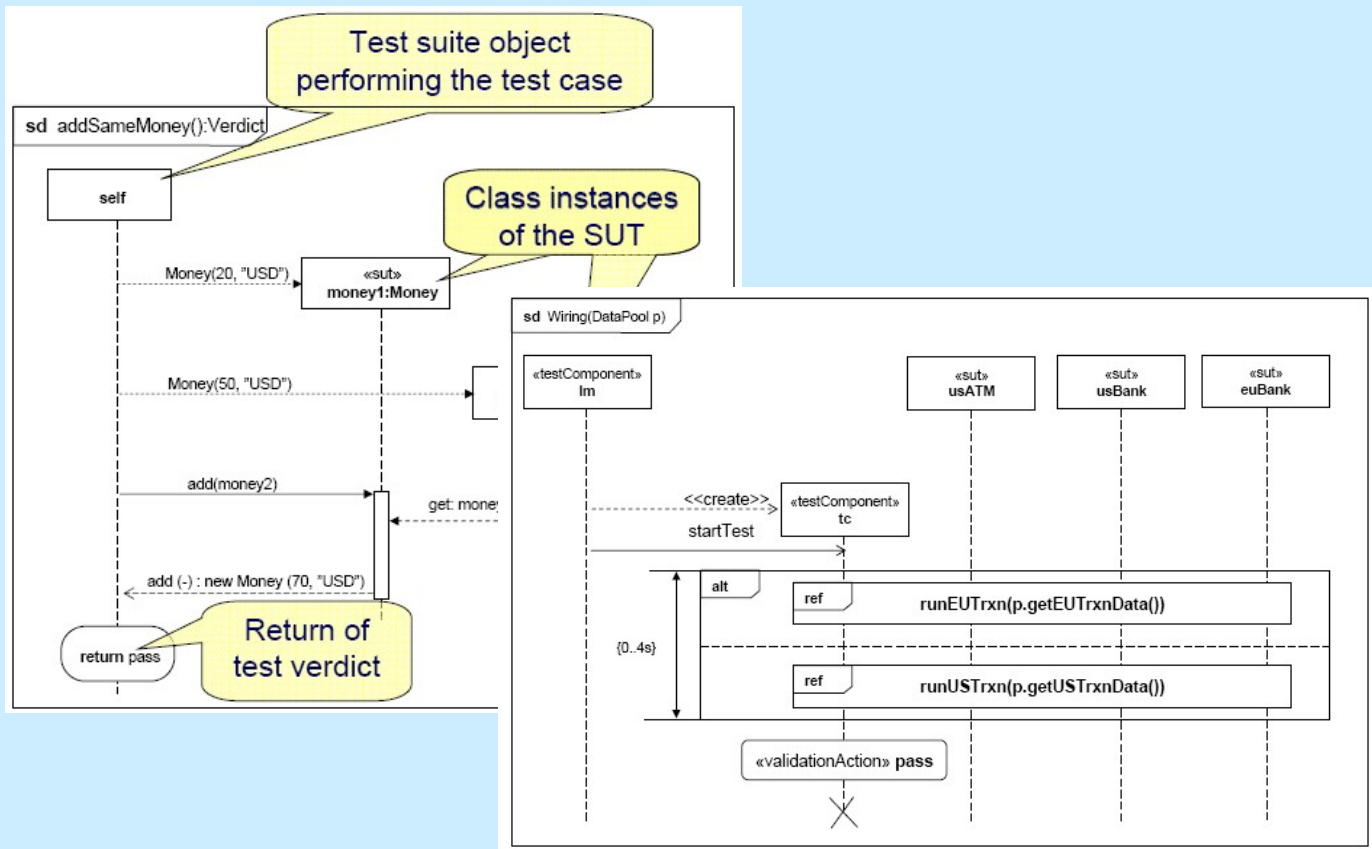
U2TP Test Data example



U2TP Test Behavior package

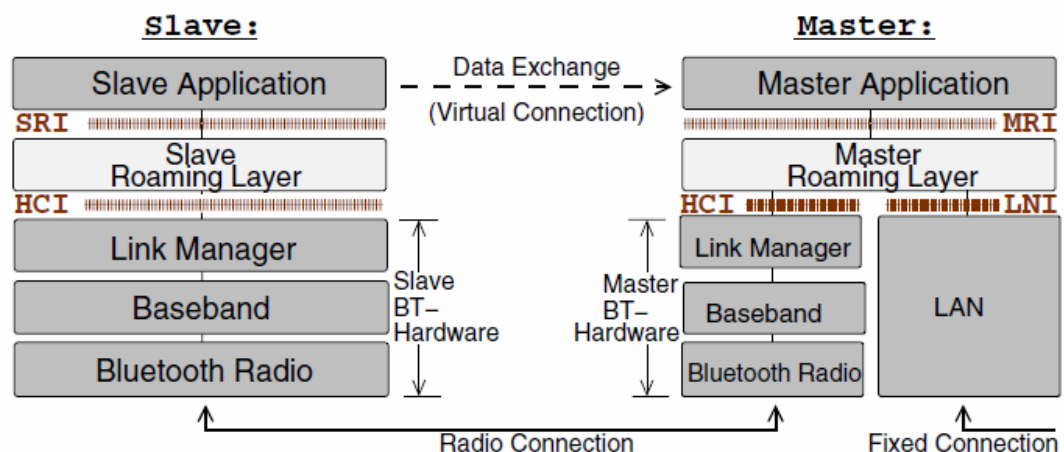
- Specification of **default/expected** behavior
- Identification of behavioral elements:
 - **Test Stimulus**: test data sent to SUT
 - **Test Observation**: reactions from the SUT
 - **Verdict**: pass, fail, error, inconclusive values
 - **Actions**: Validation Action (inform Arbiter), Log Action
- Test Case: Specifies one case to test the SUT
 - **Test Objective**: named element
 - **Test Trace**: result of test execution
 - Messages exchanged
 - **Verdict**

U2TP Test Behavior example



Mintapélda: Bluetooth roaming

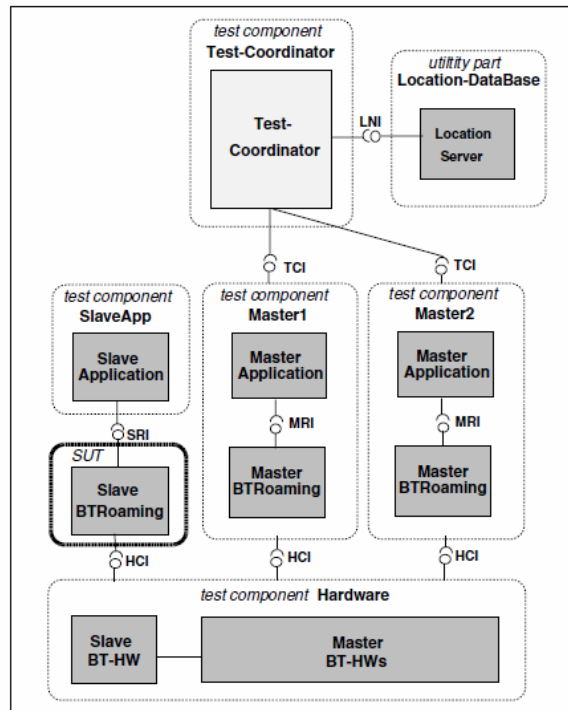
Tesztelendő rendszer:



Teszt cél:

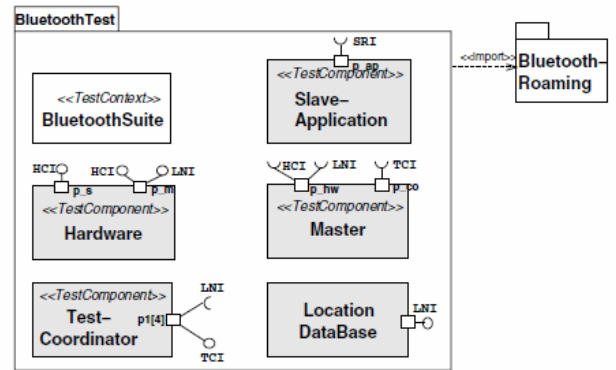
- Slave Roaming Layer funkciói
 - Link minőségének figyelése
 - Kapcsolat létesítése másik masterrel

Komponensek szerepei

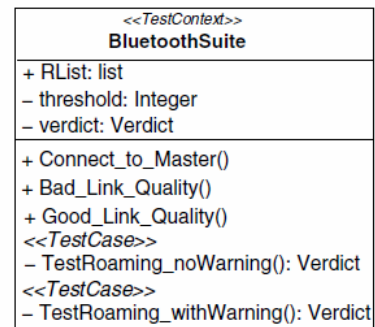


 . System Under Test (SUT)
 . Test Component with new class
 . Test Components with existing classes

Overview

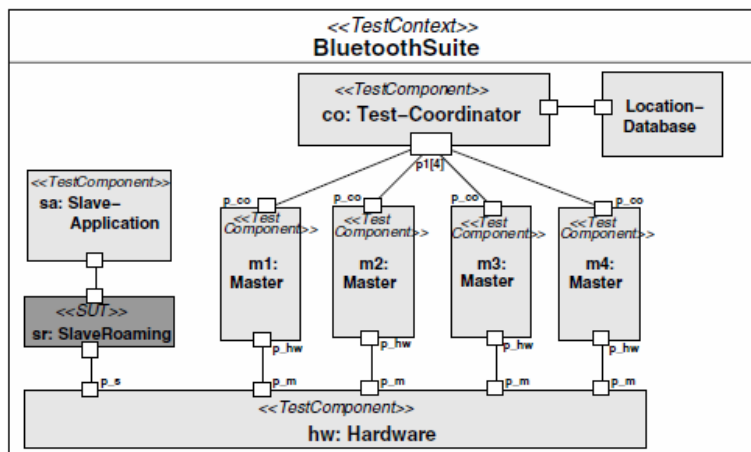


Test package

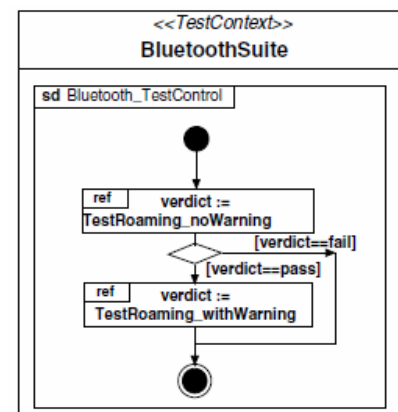


Test context

Teszt konfiguráció és teszt vezérlés



Test configuration

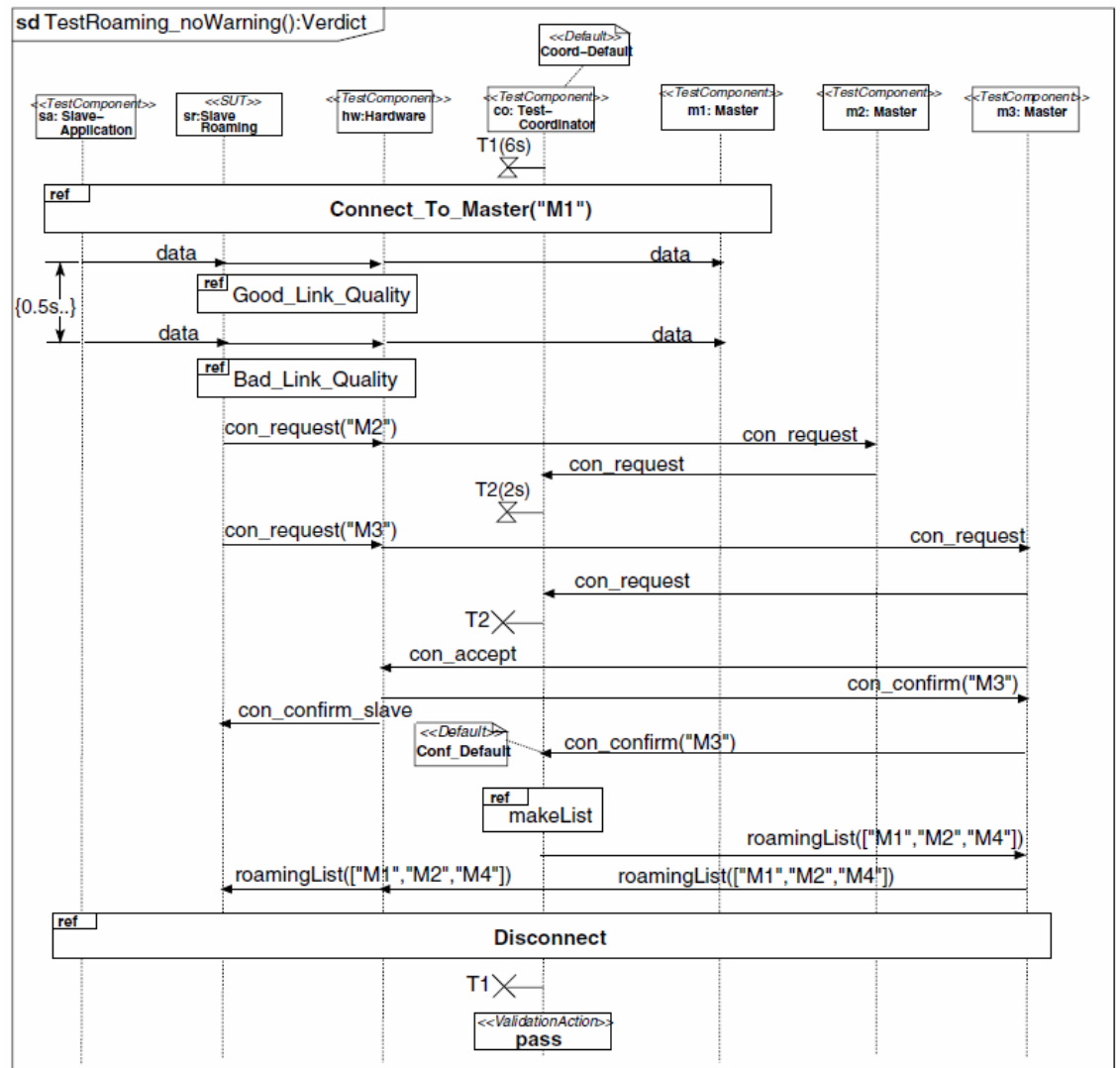


Test control

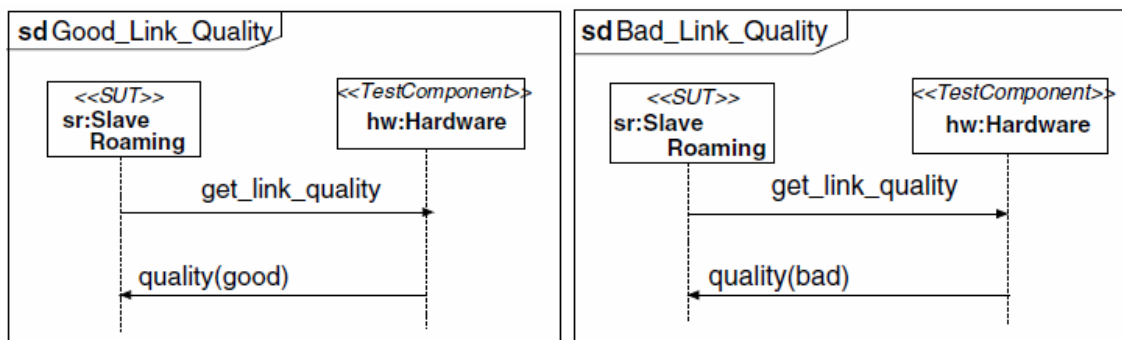
Test scenario

Test case implementation
(see Blue-
ToothSuite)

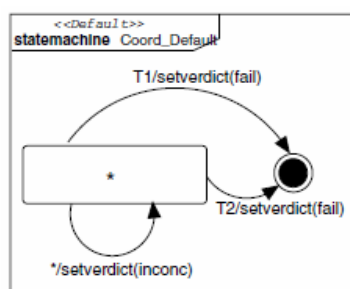
- References
- Timers
- Defaults



Néhány hivatkozott részlet



Sequence diagrams



Default behaviours specified
to catch the observations
that lead to verdicts

- Here: Processing timer events