

Integrációs tesztelés, rendszerteresztelés, validációs tesztelés

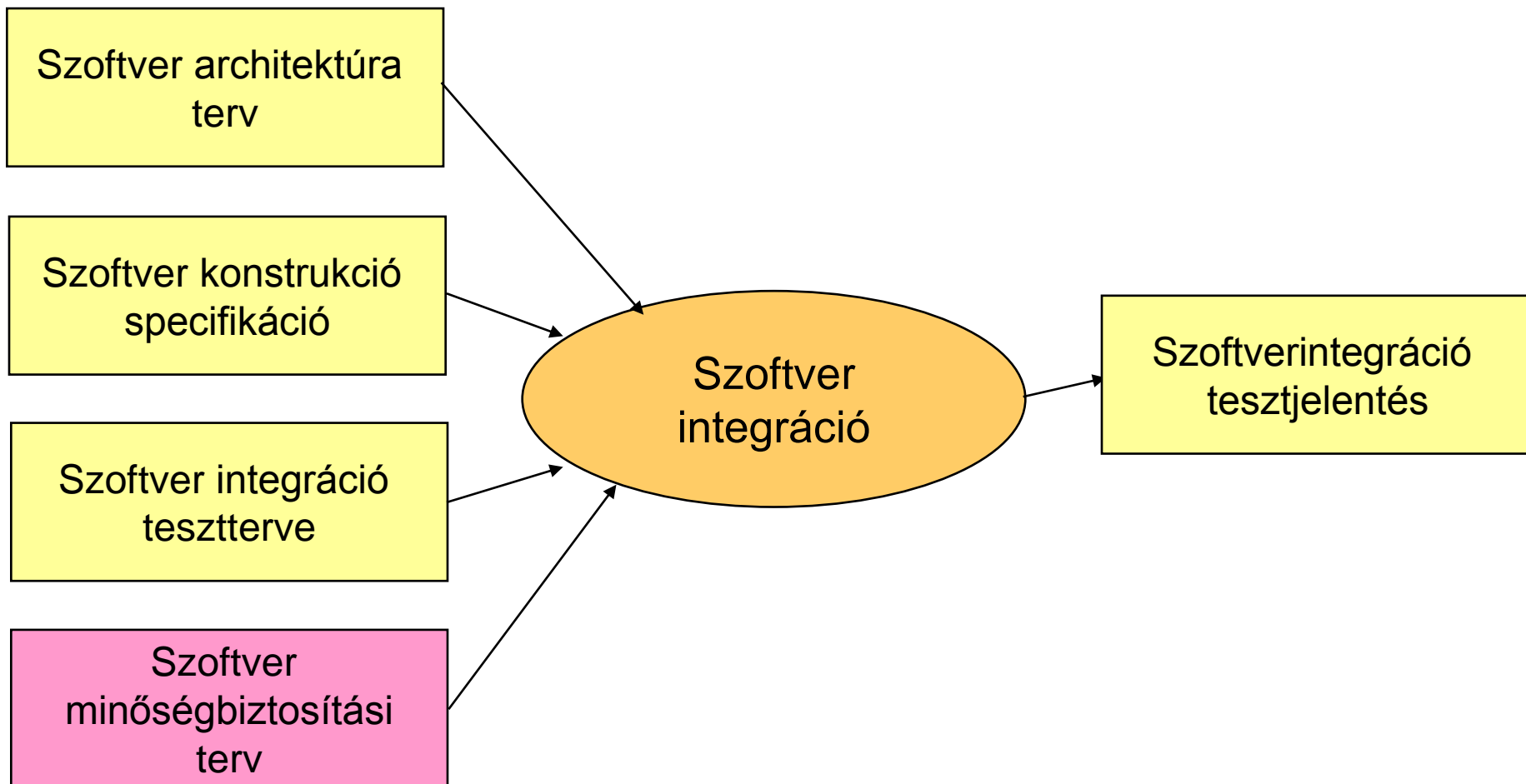
Majzik István

Budapesti Műszaki és Gazdaságtudományi Egyetem

Méréstechnika és Információs Rendszerek Tanszék

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

Szoftver integráció

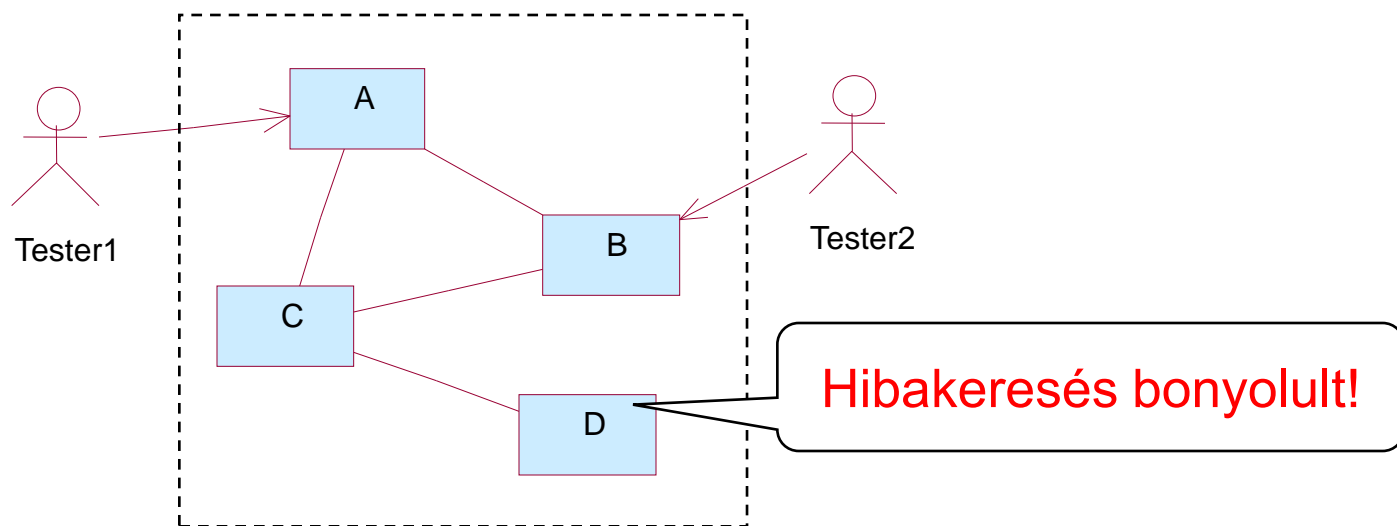


Szoftvermodulok integrációja

- Cél:
 - Modulok fokozatos kombinációja annak érdekében, hogy az összeillesztett elemek együttműködését tesztelni lehessen.
 - Szisztematikus módszer: Minden interakció (interfész használat) tesztje
 - Használati foratókönyvek alapján
- Tipikus módszerek a szabványokban:
 - Statikus elemzés
 - Dinamikus elemzés és tesztelés
 - Ekvivalencia-osztályok és bemeneti adatfelosztás szerinti teszt
 - Tesztesetek a határérték elemzésből
 - Tesztesetek hibabecslésből és hibakeresésből
 - Valószínűségi tesztelés

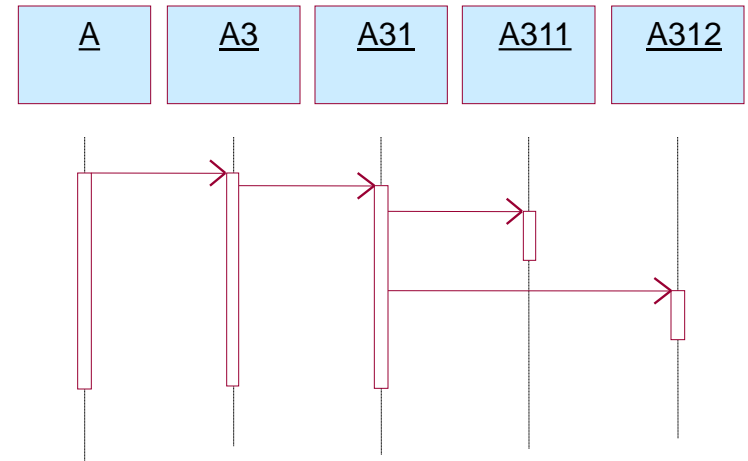
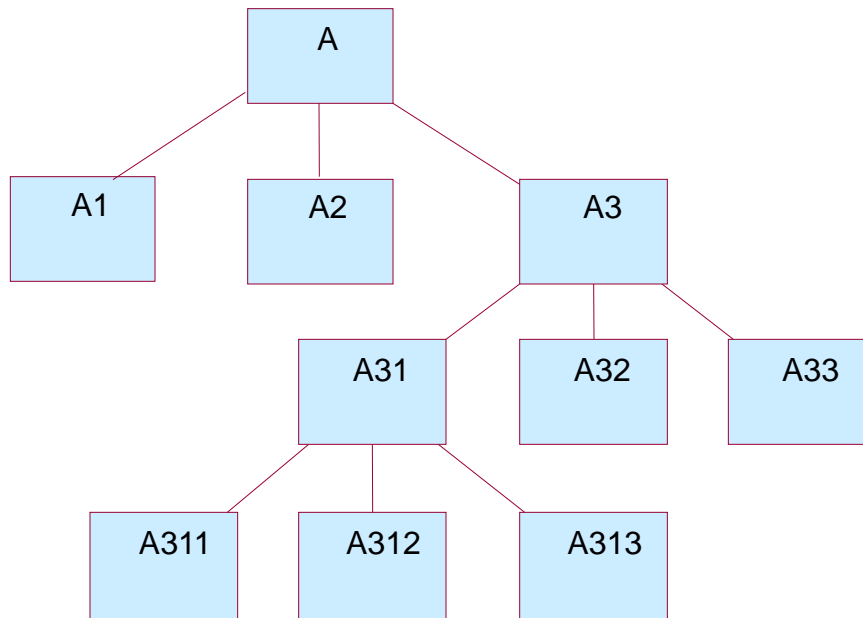
“Big bang” integrációs tesztelés

- Teljes rendszer tesztelése a **külső interfészek**en keresztül
- Külső teszt végrehajtó
- Rendszer funkcionális specifikációja alapján történik
- Kis rendszerek esetén alkalmazható



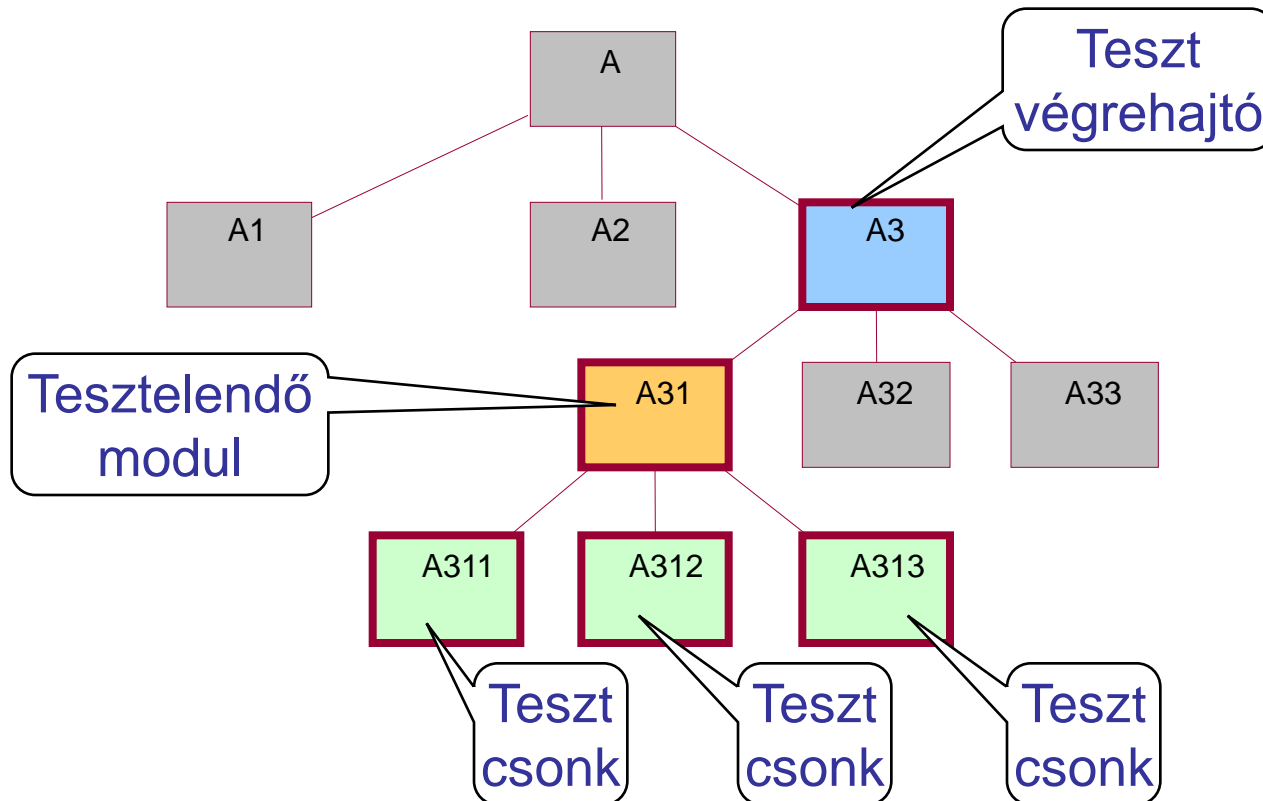
Inkrementális integráció és tesztelés

- Nagy rendszerek esetén megkönnyíti a hibakeresést
- Modul hívási hierarchia (ideális eset):



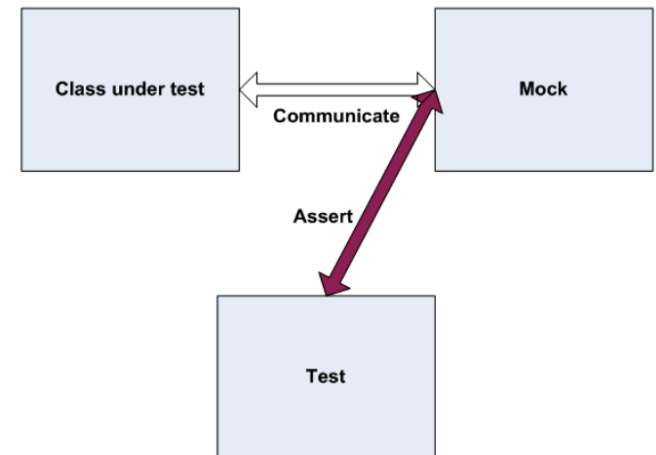
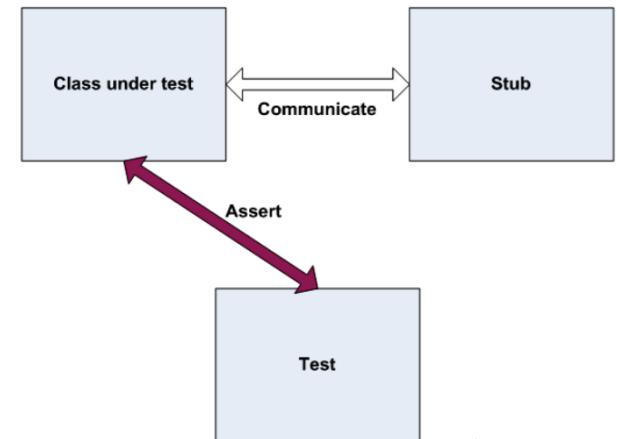
Modulok izolációs tesztelése a modul hierarchiában

- Modulok egyenként, elszigetelten teszteltek
- Teszt végrehajtó és kihelyettesítések szükségesek



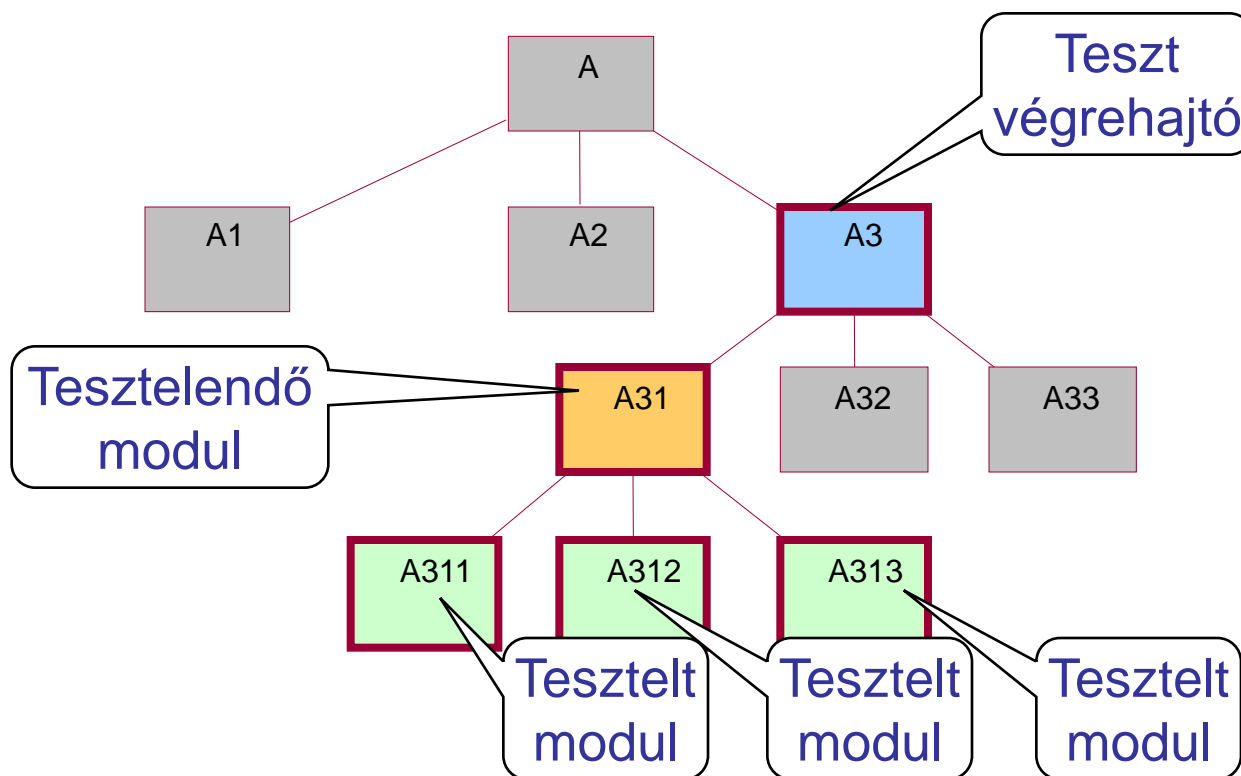
Általános probléma: Függőségek kezelése

- Sokféle technika a kihelyettesítésre
 - Ld. „isolation frameworks” (pl. Mockito, JMock, ...)
 - Összefoglaló név: **Test double**
- Stub (csonk)
 - Rögzített válaszok adott hívásokra
 - SUT **állapotának** ellenőrzésére
- Mock
 - Elvárt és ellenőrzött viselkedés
 - SUT **interakcióinak** ellenőrzésére (hányszor, milyen paraméterű hívás ...)
- Dummy
 - Nem használt („kitöltő”) objektum
- Fake
 - Működő, de nem az „éles” objektum



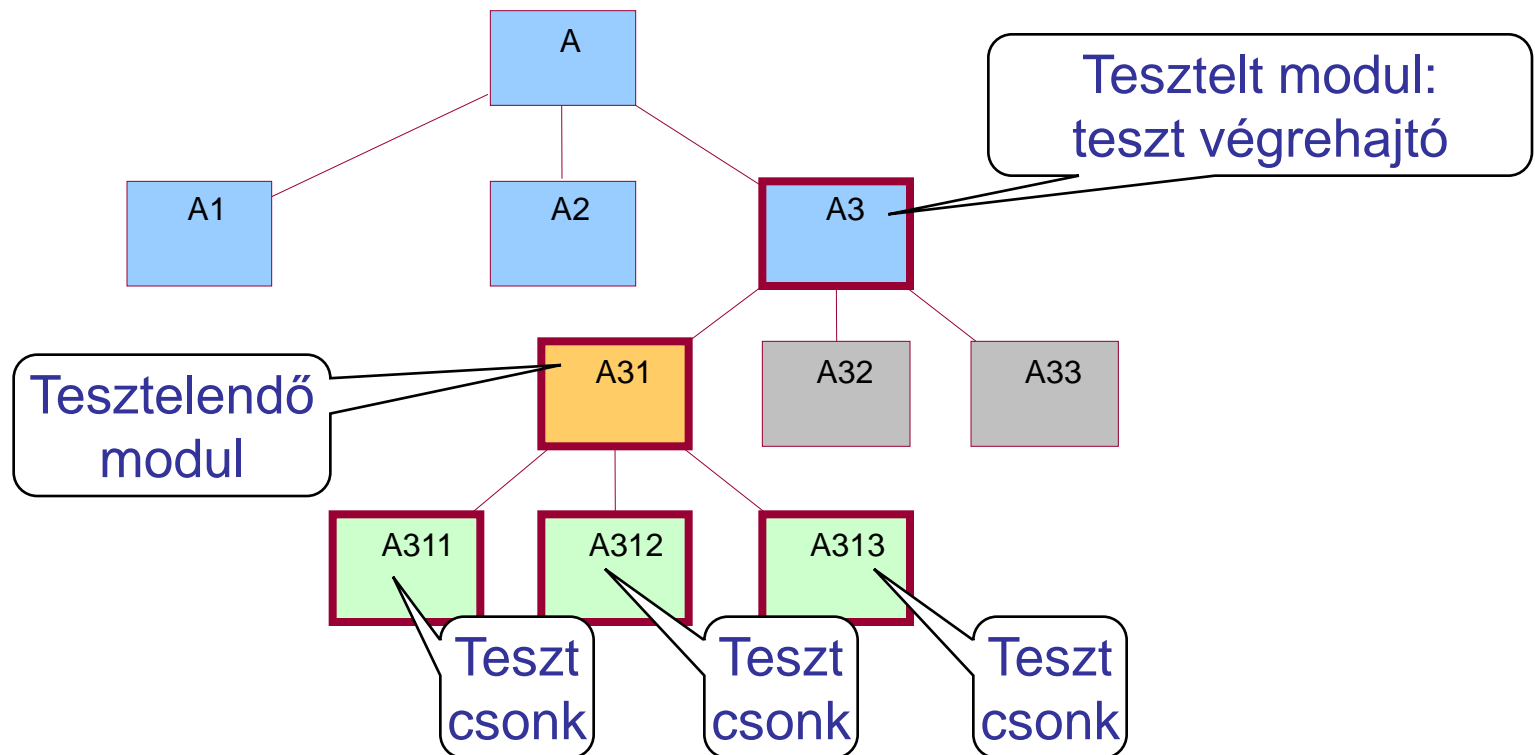
Alulról felfelé történő integrációs tesztelés

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



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



Összehasonlítás

	Alulról felfelé történő integrációs tesztelés	Felülről lefelé történő integrációs tesztelés
Előnyök	<ul style="list-style-type: none">• Integráció orientált, konstruktívabb• Könnyebb megfigyelni és irányítani a tesztek	<ul style="list-style-type: none">• Követelmény-orientált, ellenőrzéshez jól illeszkedő• Hamar összeállhat egy demonstrálható „alkalmazás váz”
Hátrányok	<ul style="list-style-type: none">• A rendszer maga csak a legvégén áll össze	<ul style="list-style-type: none">• Csonkok készítése általában nehezebb• Teszt bemenetek távol lehetnek az épp integrálandó modultól

Funkcionális integráció

- **Motiváció:**
 - Sok rendszerszintű funkció
 - Prioritások lehetnek ezek között kritikusság szempontjából (→ tesztelés ütemezése)
- **Alapötlet:**
 - Egy-egy rendszerszintű funkció mentén, fentről lefelé
- **Felülről lefelé történő integráció speciális esete**
 - Teszt csonkokra van szükség
 - Követelmény-orientált (“fentről” tesztelünk)
 - Legfelső szint van egyre „részletesebben” tesztelve
 - Modul módosítás: Alatta lévők tesztelését módosítja

Futtató rendszer integrációja

- Motiváció:

Nehéz csonkokat írni a futtató rendszerhez

– Pl. OS, RT-OS, taszk ütemezés ...

- Stratégia:

1. Alkalmazás modulok integrációja és tesztelése felülről lefelé, a futtató rendszer szintjéig

2. Futtató rendszer alulról felfelé történő tesztelése

- Funkciók izolációs tesztelése

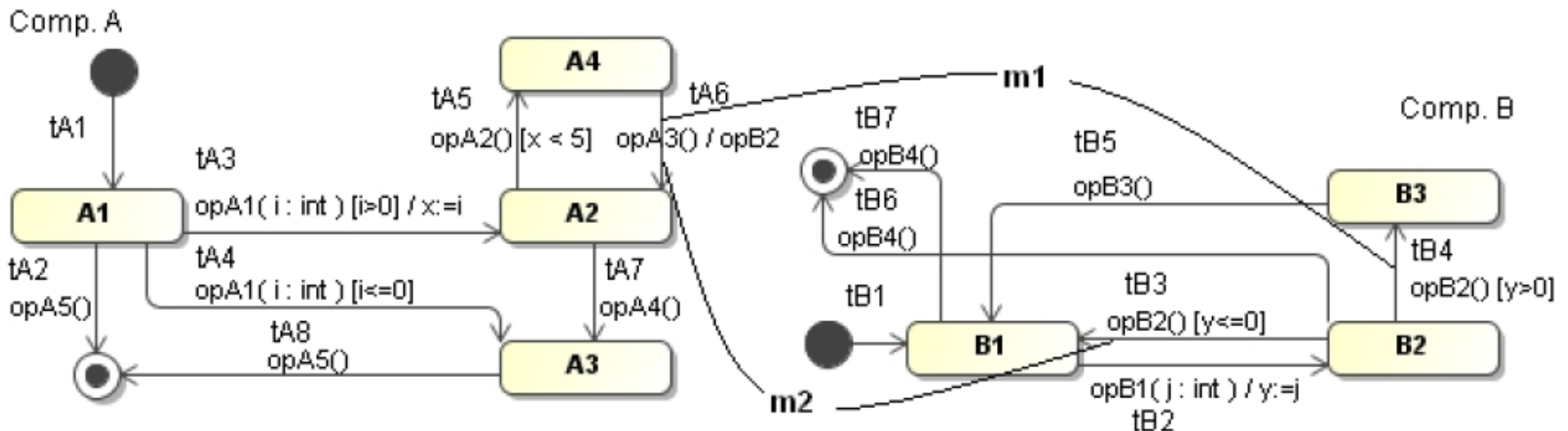
- „Big bang” tesztelés

az alkalmazás hierarchia legalsó rétegével integrálva

3. Alkalmazás és futtatórendszer integrációja, a felülről lefelé történő tesztelés befejezése

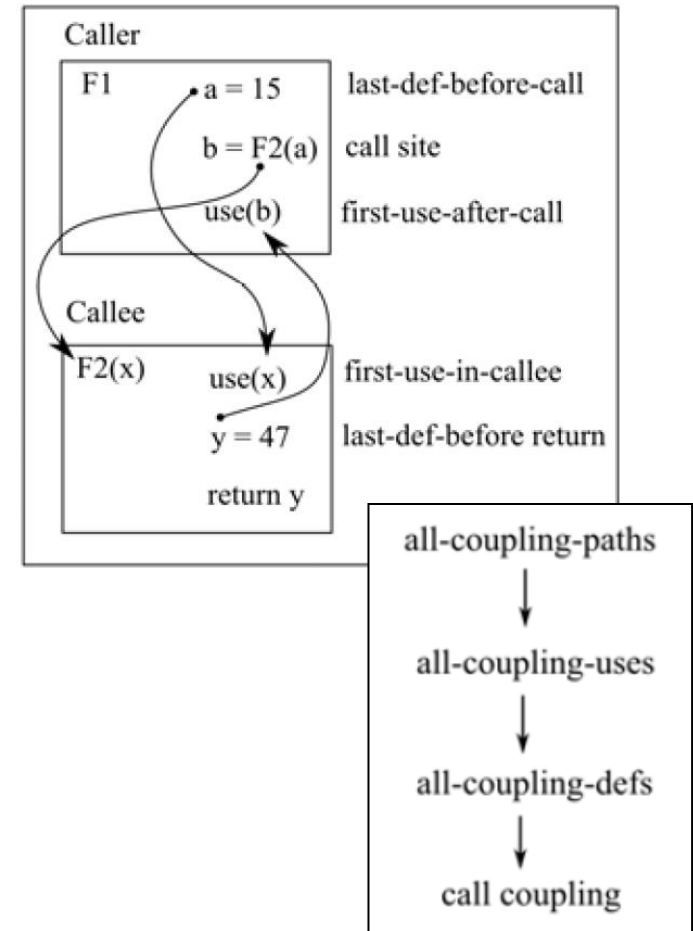
Fedettségi mértékek: Állapot alapú megközelítés

- Cél: Komponensek közötti interakciók „fedése”
- Állapot alapú mértékek:
 - Interfész (funkcióhívások) fedettsége
 - Interfész funkciók fedettsége a **hívó** illetve a **hívott** minden releváns **állapotára** (állapot partíciójára)
 - Kiterjesztés: Minden **trigger** illetve **feltétel** mellett

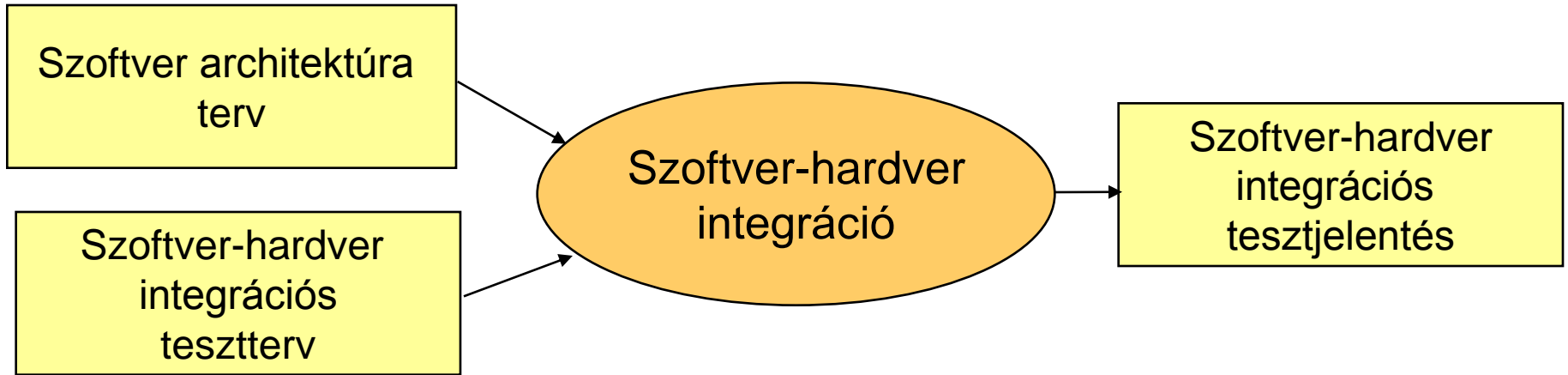


Fedettségi mértékek: Adatfolyam alapú megközelítés

- Adatfolyam alapú mértékek:
 - Minden csatolási útvonal (coupling path) fedése
 - Def-use címkék használata kiterjesztése a hívásokra (call és return)
 - All-coupling-defs, all-coupling-uses, all-coupling-paths a csatolási útvonalakra
- Interfészek robusztussága
 - Ld. külön a robusztusság teszteléshez kapcsolódva



Szoftver-hardver integráció



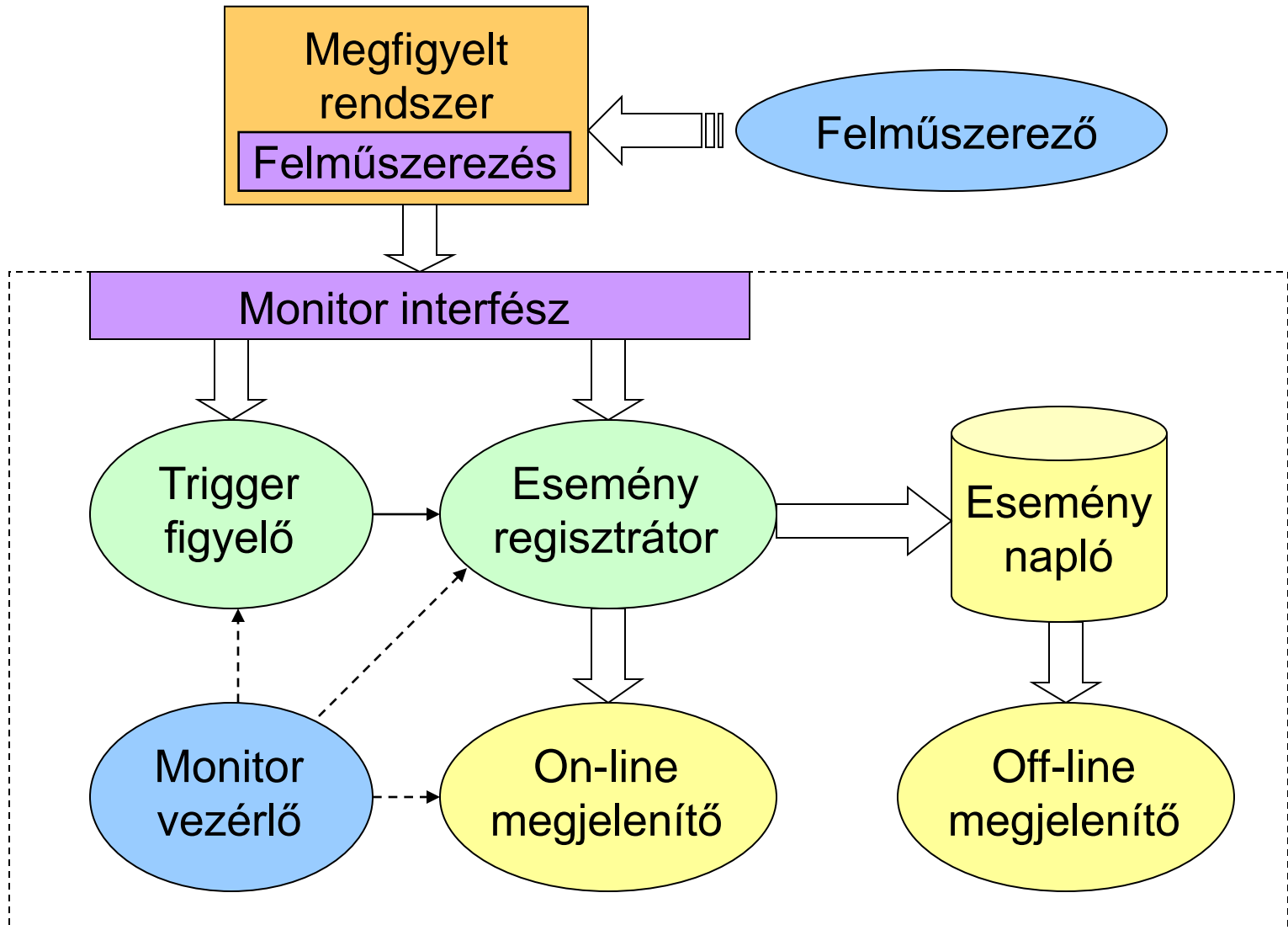
Szoftver-hardver integráció

- Célok: Szoftver és hardver „összekapcsolása”
 - Együttműködésük megfelelőségének bizonyítása
 - Funkcionalitás és teljesítmény szempontjából (pl. elosztott rendszerben)
 - Hibahatások szempontjából (pl. biztonságkritikus rendszerben)
 - Szoftver-hardver integrációs teszterv alapján
- Tipikus módszerek a szabványokban:
 - Funkcionális és „fekete doboz” tesztelés
 - Ekvivalencia-osztályok és bemeneti adatfelosztás teszt
 - Határérték elemzés
 - Tesztesetek ok-okozati diagramokból
 - Folyamatszimuláció
 - Prototípus készítés
 - Teljesítménytesztelés
 - Válaszidő- és memóriakikötések tesztelése
 - Teljesítmény követelmények tesztelése
 - Lavina- / stressz tesztelés

Gyakran használt technológia: Monitorozás

- Megfigyelendő:
 - Időadatok, kommunikáció, szinkronizáció, ...
 - Memóriafooglalás, CPU terhelés, ...
- Alapfeladatok:
 - Információ hozzáférés: **Felműszerezés**
 - Szoftver úton: Extra utasítások beszúrása
 - Hardver úton: Csatlakozás a rendszer buszra, kimenetekre
 - Információ szűrés: **Triggerelés**
 - Szoftver úton: Extra utasítások megoldják
 - Hardver úton: Busz forgalom, jelek figyelése
 - Információ tárolás: **Regisztrálás**
 - Szoftver úton: Naplózás szoftver modulból
 - Hardver úton: Jelek mintavételezése („logikai analizátor”)

Általános monitorozási séma



Példa: Előfeldolgozás (felműszerezés)

```
for (i=0; i<9; i++) {  
    V(sm1);  
    if (i==a) {  
        P(sm1);  
    } else {  
        P(sm2);  
    }  
}
```

Elő-
feldolgozó

```
for (i=0; i<9; i++) {  
    EV(V1); V(sm1);  
    if (i==a) {  
        EV(P1); P(sm1);  
    } else {  
        EV(P2): P(sm2);  
    }  
}
```

```
void switch_up() {  
    if( gear == 5 ) {  
        error();  
        return;  
    }  
    set_gear( gear+1 );  
}
```

```
char _TABLE[ NUM_BLOCK / 8 ];  
void switch_up() {  
    if( gear == 5 ) {  
        error();  
        _TABLE[17/8] |= (1<<(17%8));  
        return;  
    }  
    set_gear( gear+1 );  
    _TABLE[18/8] |= (1<<(18%8));  
}
```

Monitorozás problémái

- **Beavatkozás:**

Ha a rendszer erőforrásait használjuk, akkor megváltoztatjuk a rendszer viselkedését

- Példa: Időzítések, eseménysorrend eltérő lesz
- Megoldás: Hardver monitorozás, korrekció, bennhagyás

- **Szemantikai hézag:**

A megfigyelt információ különbözik a szükséges információtól

- Példa: Processzor buszjelekből szemafor műveletekre következtetni
- Megoldás: Rugalmas (szoftver) triggerelés vagy off-line analízis

- **Globális jellemzők származtatása:**

Elosztott rendszerek esetén a lokális információból globális jellemzők

- Példa: Körkörös várakozás detektálása
- Megoldás: Központi monitor, lokális monitorok szinkronizálása

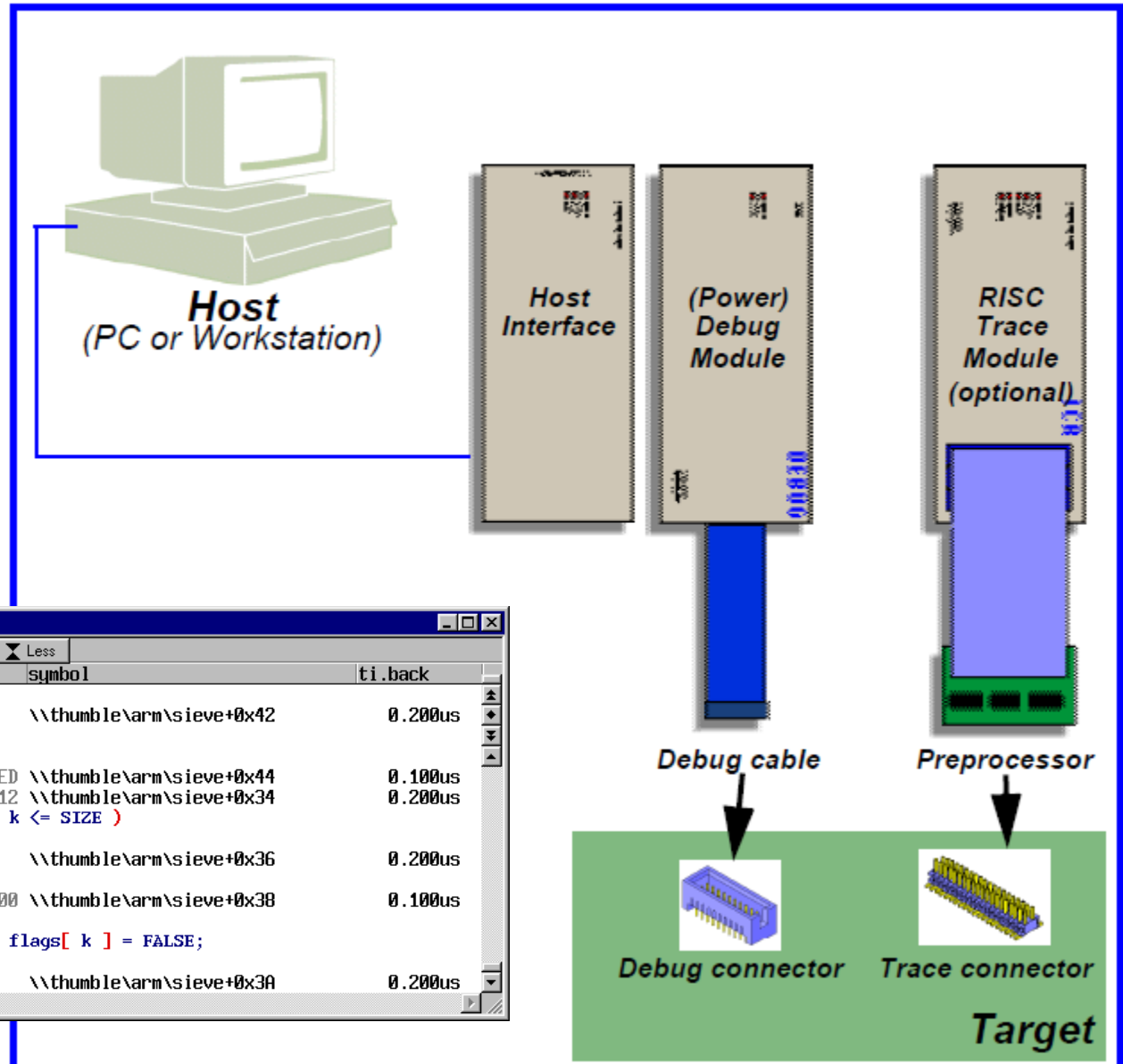
Általános megoldások áttekintése

Monitorozás típusa	Felműszerezés	Triggerelés	Regisztrálás
Szoftver (rugalmas, beavatkozó)	Trigger utasítások (aut.) beszúrása	Trigger utasítások közvetlenül	Alkalmazás, vagy monitor processz
Hardver (nem beavatkozó, de specifikus)	Közvetlen csatlakozás	Hardver komparátor	Lokális vagy távoli puffer
Hibrid (rugalmas, kevésbé beavatkozó)	Trigger utasítások (aut.) beszúrása	Trigger utasítások közvetlenül	Hardver (lokális vagy távoli puffer)

Példa: Hardver monitorozás

Lauterbach TRACE32

- Bus trace, program trace
- 512 Kframes trace memory
- 94 channels
- 36 bit time stamp
25 ns resolution



B::TRace.List

record	run	address	cycle	d.l	symbol	ti.back
-00000023	f	add r3,r3,r7				
696		T:00001BBE	fetch	3101	\\thumble\arm\sieve+0x42	0.200us
-00000022	f	b 0x1BB0				
692		T:00001BC0	fetch	E7ED	\\thumble\arm\sieve+0x44	0.100us
-00000021	f	T:00001BB0	fetch	2B12	\\thumble\arm\sieve+0x34	0.200us
692					while (k <= SIZE)	
-00000020	f	cmp r3,#0x12				
693		T:00001BB2	fetch	DC04	\\thumble\arm\sieve+0x36	0.200us
-00000019	f	bgt 0x1BBE				
693		T:00001BB4	fetch	2400	\\thumble\arm\sieve+0x38	0.100us
694					{	
-00000018	f	mov r4,#0x0				
694		T:00001BB6	fetch	4805	\\thumble\arm\sieve+0x3A	0.200us
-00000018					flags[k] = FALSE;	
-00000018					}	

Példa: Szoftver monitorozás

- **Profilerek:**
 - Függvényhívási hierarchia felderítése, hívási gyakoriság hozzárendelése
 - Az egyes függvényekben eltöltött idő hozzárendelése
→ mit érdemes újraírni, optimalizálni
 - Annotált forrás: programsorok végrehajtási gyakorisága
 - Célkörnyezetben futtatva használhatók az eredmények
- **Legegyszerűbb példa: gprof**
 - Program felműszerezése: speciális fordítás
`cc -pg prog.c -o prog` (opció; `gcrt0.o`, `libc_p.a`)
 - Futás közbeni adatgyűjtés: adat file-ok (`gmon.out`)
 - Off-line kiértékelés: `gprof prog gmon.out`

Példa: gprof jelentés: Flat profile

% time	cumulative seconds	self seconds	calls	self us/call	total us/call	name
33.65	0.71	0.71	4	177500.00	177500.00	RELERR
28.91	1.32	0.61	4	152500.00	152500.00	RELFINE
13.27	1.60	0.28	60	4666.67	4666.67	RELCOARSE
10.90	1.83	0.23	32	7187.50	7187.50	INTERPOL
6.16	1.96	0.13	4	32500.00	32500.00	RESFINE
2.84	2.02	0.06	28	2142.86	2142.86	RESCOARSE
1.90	2.06	0.04	1	40000.00	40000.00	write_res
1.42	2.09	0.03	1	30000.00	30000.00	INIT
0.95	2.11	0.02	8	2500.00	2500.00	PUTZERO
0.00	2.11	0.00	1	0.00	0.00	OUTPUT

- **Time:** Percentage of the total running time of program used by this function
- **Cumulative seconds:** Sum of the seconds accounted for by this function and those listed above it
- **Self seconds:** The number of seconds accounted for by this function alone
- **Self us/call:** The average number of microseconds spent in this function per call
- **Total us/call:** The average number of microseconds spent in this function and its descendents per call

Példa: gprof jelentés: Call graph

index	% time	self	children	called	name
[1]	100.0	0.00	2.11		main [1]
		0.71	0.00	4/4	RELERR [2]
		0.61	0.00	4/4	RELFINE [3]
		0.28	0.00	60/60	RELCOARSE [4]
		0.23	0.00	32/32	INTERPOL [5]
		0.13	0.00	4/4	RESFINE [6]
		0.06	0.00	28/28	RESCOARSE [7]
		0.04	0.00	1/1	write_res [8]
		0.03	0.00	1/1	INIT [9]
		0.02	0.00	8/8	PUTZERO [10]

		0.71	0.00	4/4	main [1]
[2]	33.6	0.71	0.00	4	RELERR [2]

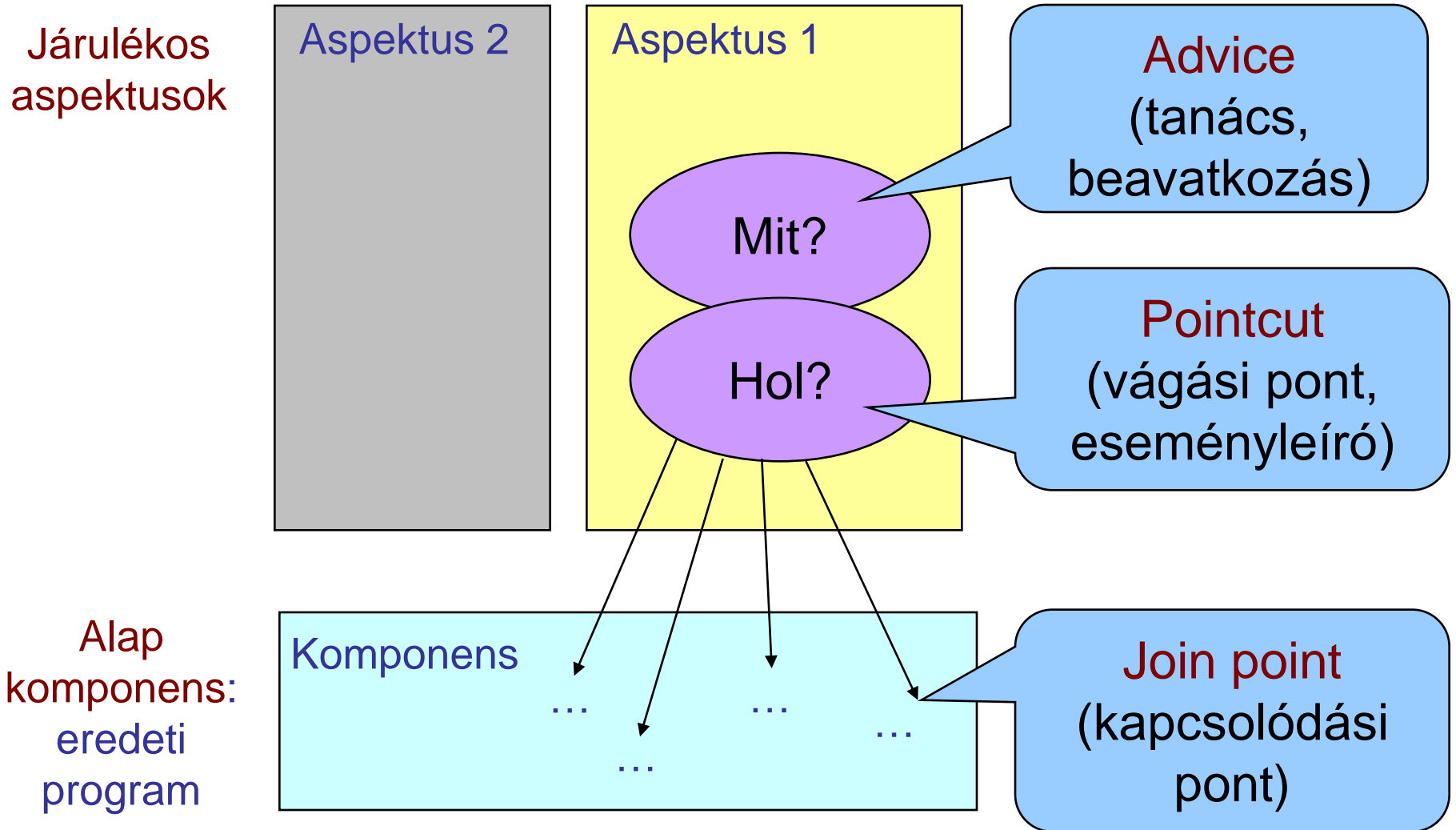
		0.61	0.00	4/4	main [1]
[3]	28.9	0.61	0.00	4	RELFINE [3]

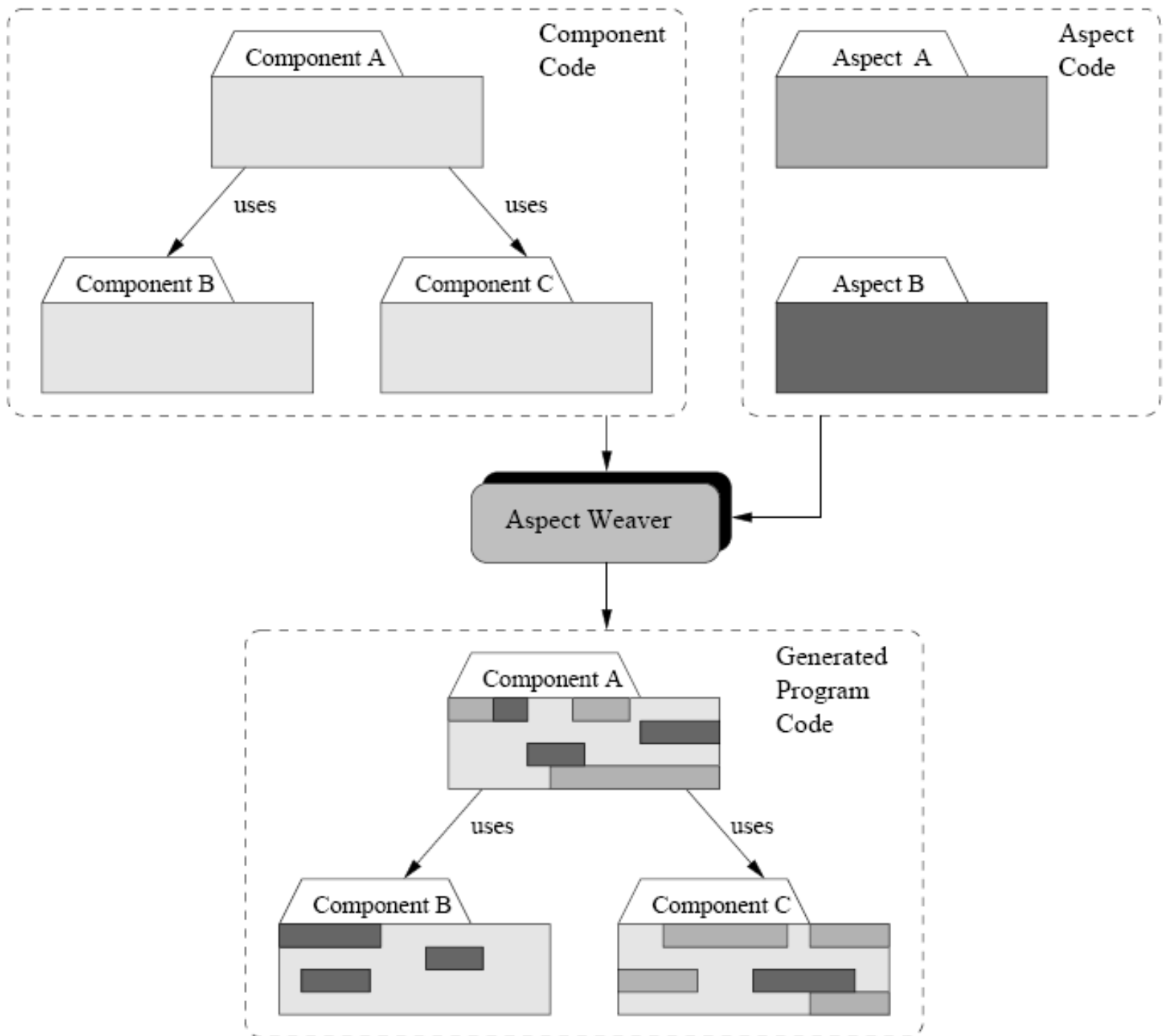
- **Self:** Total amount of time spent in this function
- **Children:** Total amount of time propagated into this function by its children
- **Called:** Number of times the function was called

Példa: Monitorozás megvalósítása AOP-vel

- A monitorozás a teljes szoftvert „átszövő” beavatkozás
 - Naplózás, nyomkövetés, hibakezelés is ilyen
 - Sok helyen kellene beavatkozni, módosítani
- Beavatkozások modularizálása ún. aspektusokban
 - Megadható a beavatkozás helye (reguláris kifejezés)
 - Vágási pont (pointcut): hová kell kapcsolódnunk?
 - kapcsolódási pont (join point): aspektus interakció helye
pl. `send* ()` - minden `send` kezdetű metódus
 - Újrahasználható beavatkozás (viselkedés)
 - Beavatkozás (advice): mit kell ott csinálni?
Pl. naplófájlba írás a `send* ()` végrehajtásáról
- Fordítás során:
 - Aspektus és eredeti kód összefésülése (felműszerezés)

Példa: Aspektus-orientált programozás





Példa: Aspektus-orientált minta

Eredeti forráskód:

```
class Controller {  
    public int send(int msg) {  
        <<sending message>>  
    }  
}
```

A send() hívást szeretnénk regisztrálni

Vágási pont definiálás loggedCall névvel

Aspektus kód:

```
public aspect SimpleLog {  
    pointcut loggedCall(int msg):  
        execution(public int Server.send*(..) && args(msg);  
  
    before(int input): loggedCall(msg) {  
        <<log request(msg)>>  
    }  
    after() throwing exception (Exception e): loggedCall() {  
        <<log exception>>  
    }  
}
```

Vágási pont (**trigger**):
send() végrehajtása

Vágási pont előtt avatkozunk be

Beavatkozás (**regisztrálás**)

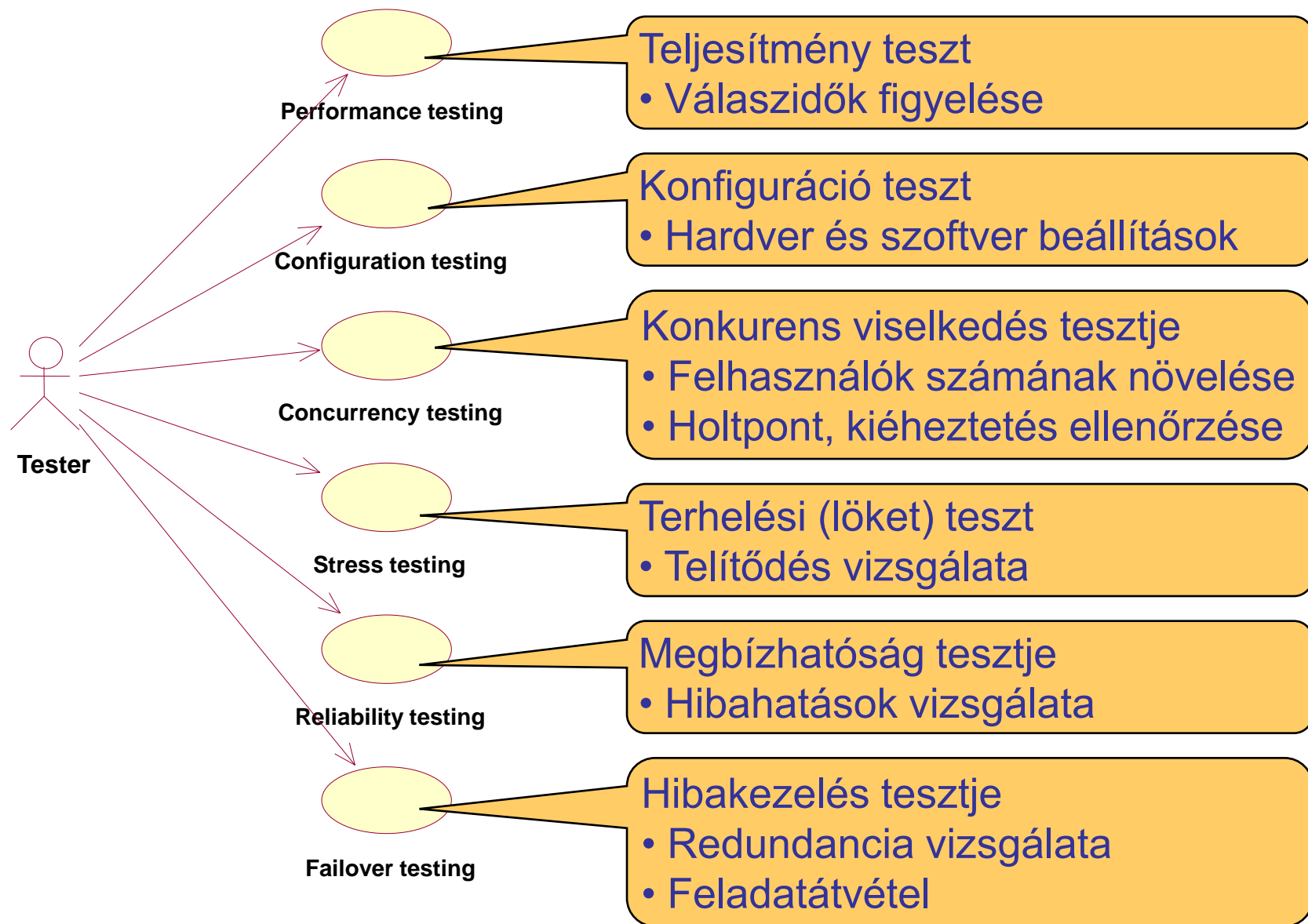
Kivétel dobásának regisztrálása

Rendszertesztelés

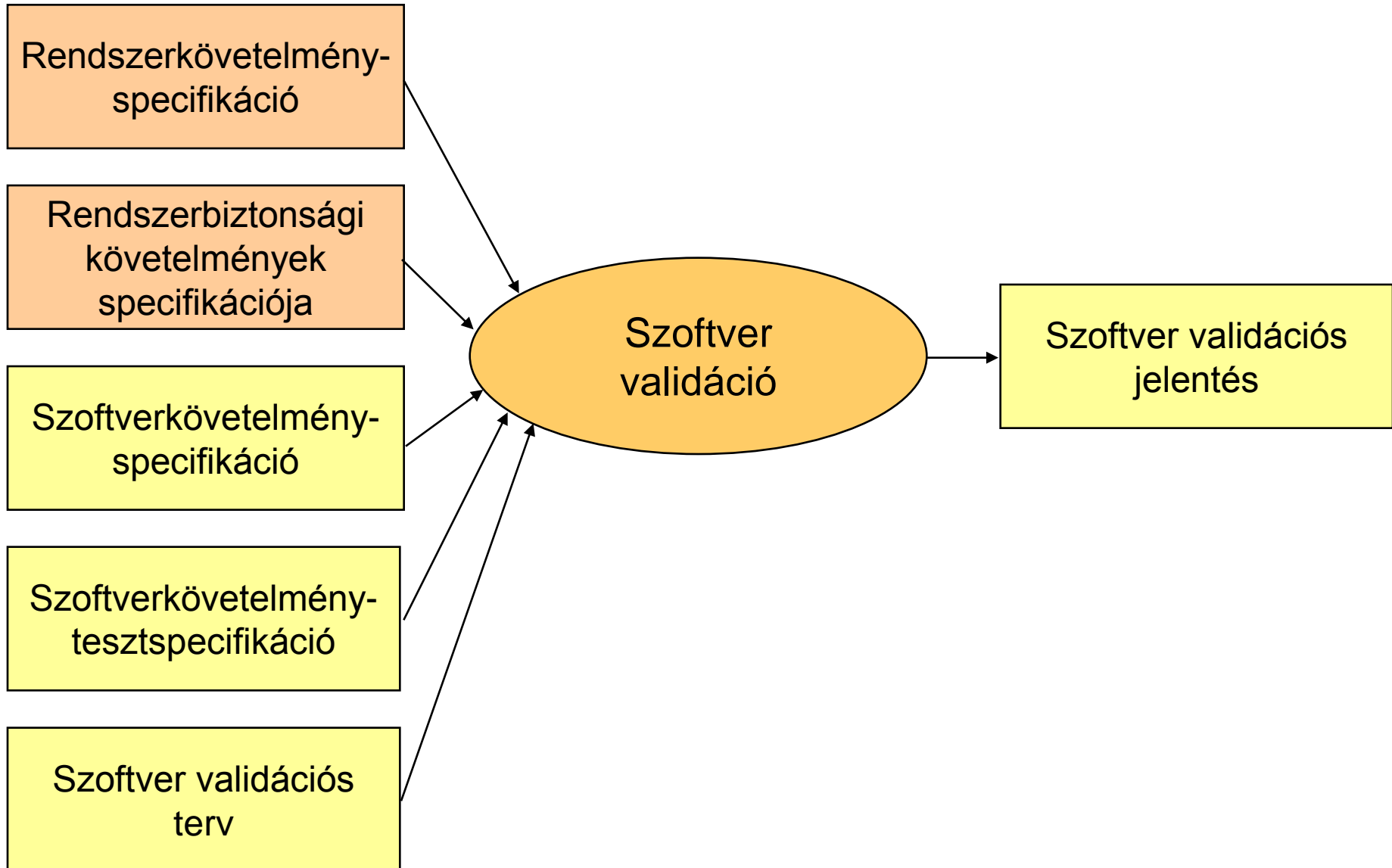
Tesztelés a rendszerszintű specifikáció alapján

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

Teszt típusok



Szoftver érvényesítés (validáció)



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
- Időzíítési szempontok
 - Időzítések megfigyelése az adott környezetben
 - Korlátok ellenőrzése: Valósídejű 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
- Funkcionális integráció

3. Rendszerteresztelés

- Teljes rendszer tesztelése

4. Validációs tesztelés

- Felhasználói elvárások tesztelése
- Környezeti szimuláció

A tesztek és a teszt környezet dokumentálása

Majzik István

Budapesti Műszaki és Gazdaságtudományi Egyetem

Méréstechnika és Információs Rendszerek Tanszék

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

Standard test documentation (IEEE 829:1998)

Standard for Software Test Documentation

Test planning:

- **Test Plan:** What is tested, by whom, how, in what time frame, to what quality
SPACEDIRT: Scope, People, Approach, Criteria, Environment, Deliverables, Incidentals, Risks, Tasks

Test specification:

- **Test Design Specifications:** Test conditions, expected outcome, what is a successful test
- **Test Case Specifications:** The specific test data (test suites)
- **Test Procedure Specifications:** What kind of physical set-up is required, how the tester runs the test, what steps need to be followed

Test reporting

- **Test Item Transmittal Report:** When specific tested items are passed from one stage of testing to another
- **Test Log:** What tests cases were run, by whom, in what order, and whether individual tests were passed or failed
- **Test Incident Report:** Details of test failure (when, why)
- **Test Summary Report:** Assessment about the quality of the system

Standard test documentation (IEEE 829:2008)

Standard for Software and System Test Documentation

Test planning:

- Master Test Plan (MTP): Overall test planning for multiple levels
- Level Test Plan (LTP): Scope, approach, resources, and schedule of the testing

Test design:

- Level Test Design (LTD): Test cases, the expected results, the test pass criteria
- Level Test Case (LTC): Specifying the test data for use in running the test cases
- Level Test Procedure (LTPr): How to run each test (preconditions and the steps)

Test reporting:

- Level Test Log (LTL): Record of relevant details about the execution
- Anomaly Report (AR): Events that occur during testing and require investigation
- Level Interim Test Status Report (LITSR): Summarize/evaluate interim results
- Level Test Report (LTR): Summarize/evaluate the results after test execution has finished for the specific test level
- Master Test Report (MTR): Summarize/evaluate the results of the levels

Modelling: UML 2 Testing Profile U2TP (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):

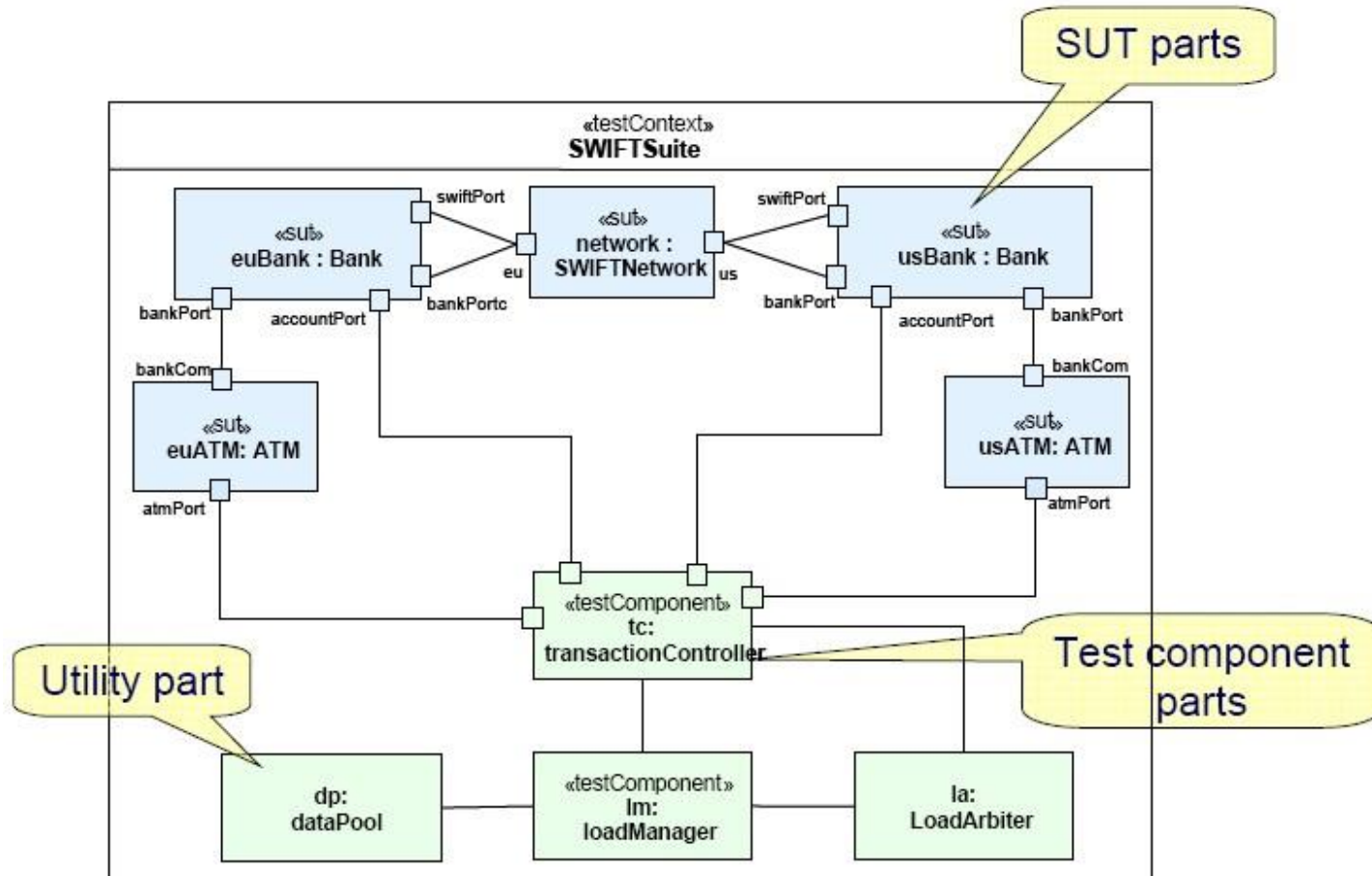
- **Test Architecture**
 - Elements and relationship involved in test
 - Importing the UML design model of the SUT
- **Test Data**
 - Structures and values to be processed in a test
- **Test Behavior**
 - Observations and activities during testing
- **Time Concepts**
 - Timer (start, stop, read, timeout),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

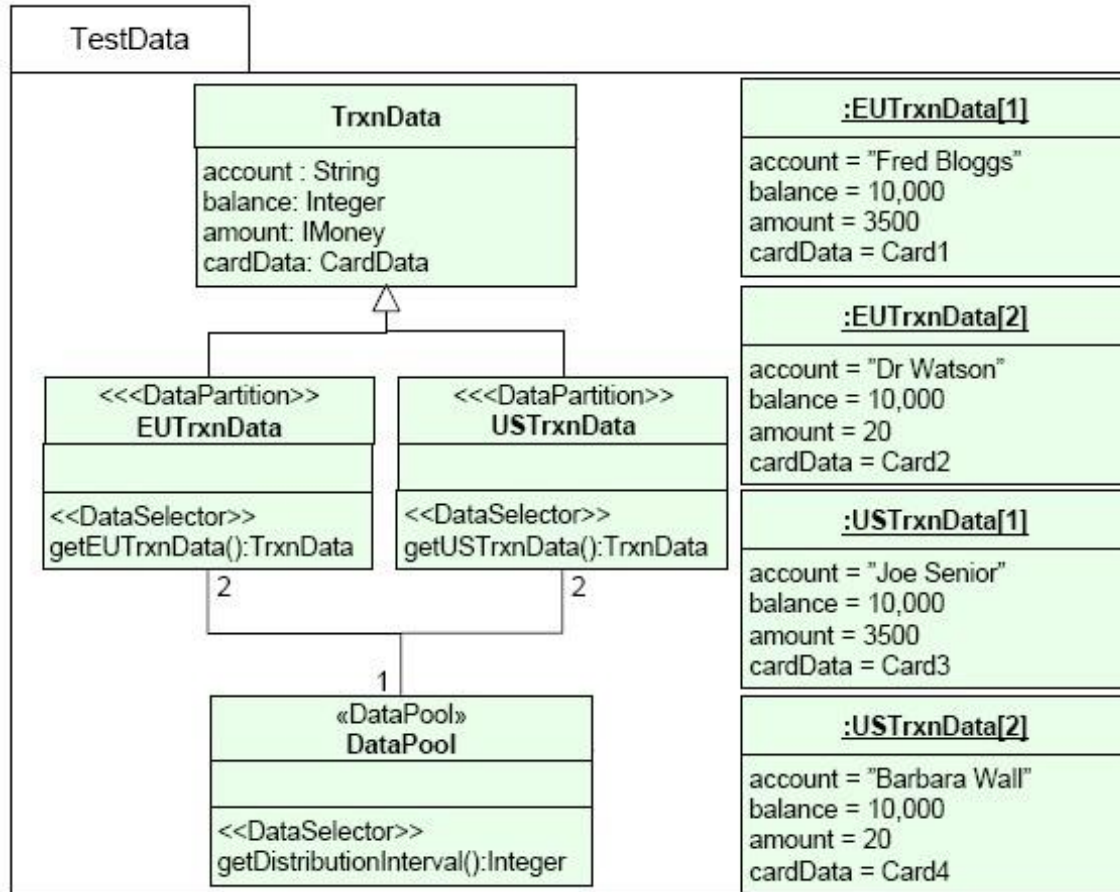
U2TP Test Architecture example



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**

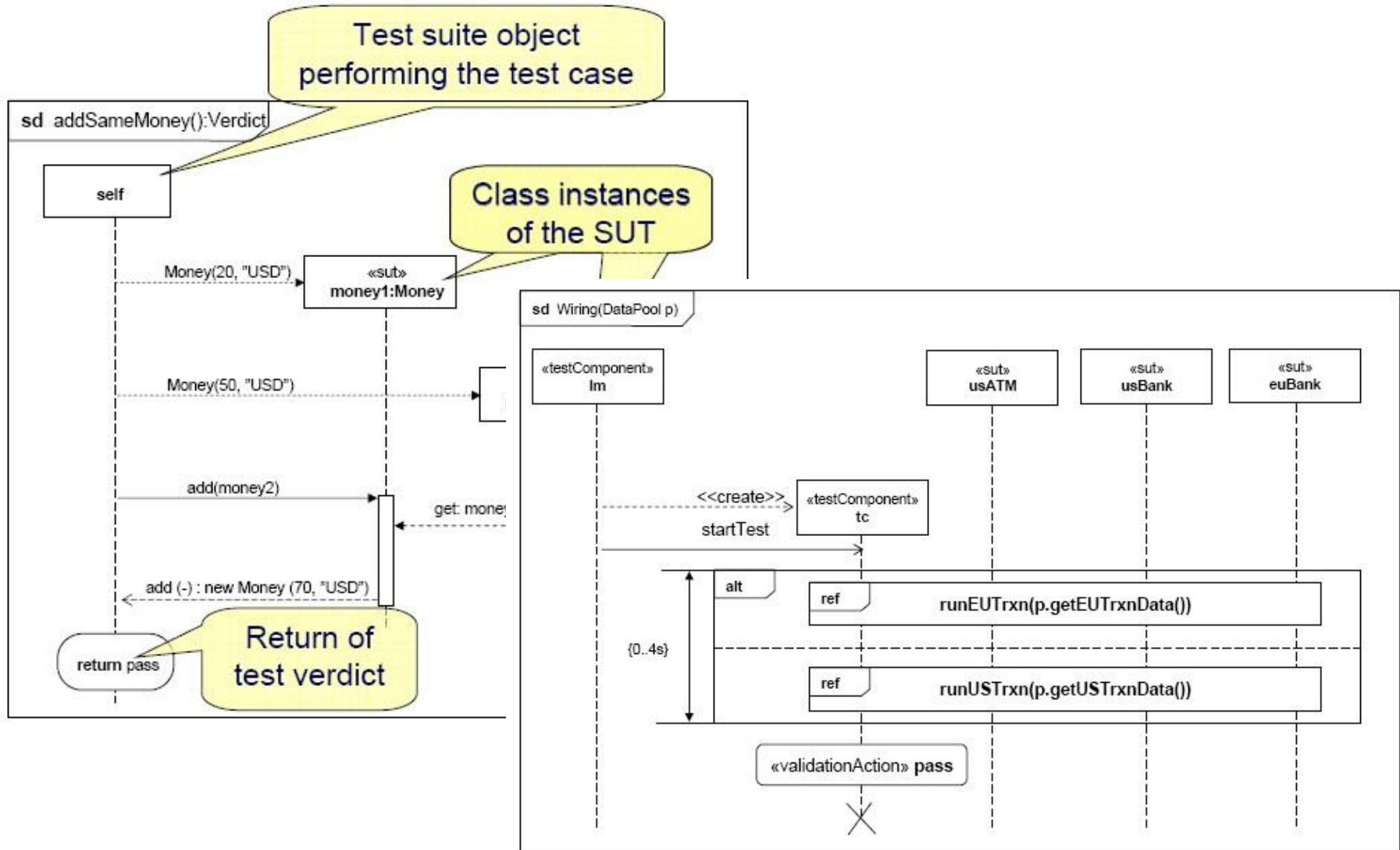
U2TP Test Data example



U2TP Test Behavior package

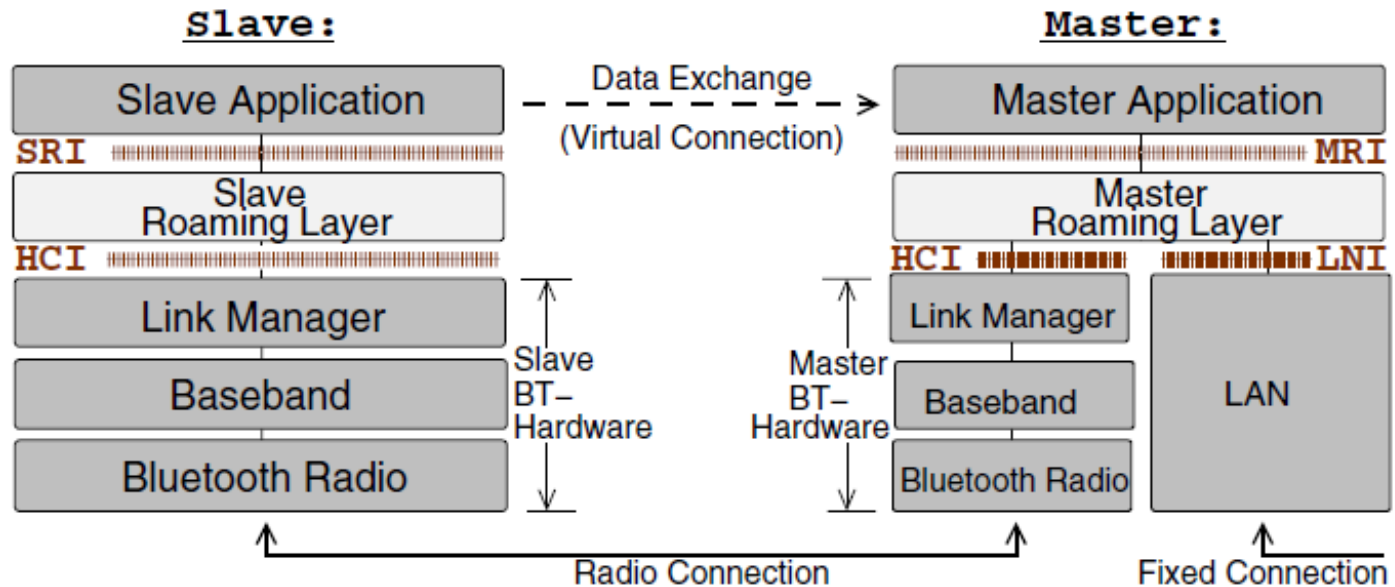
- Specification of default/expected behavior
- Identification of behavioral elements:
 - Test Stimulus: test data sent to SUT
 - Test Observation: reactions from the SUT
 - Verdict: pass, fail, error, inconclusive values
 - Actions: Validation Action (inform Arbiter), Log Action
- Test Case: Specifies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

U2TP Test Behavior example



Példa: BlueTooth roaming

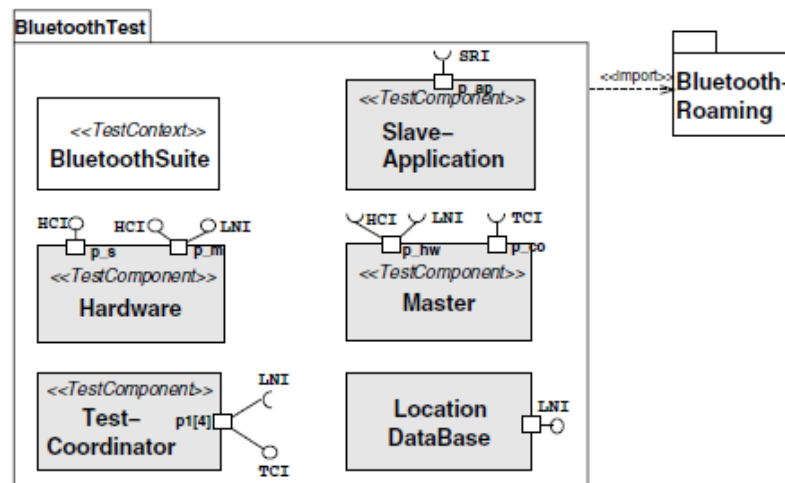
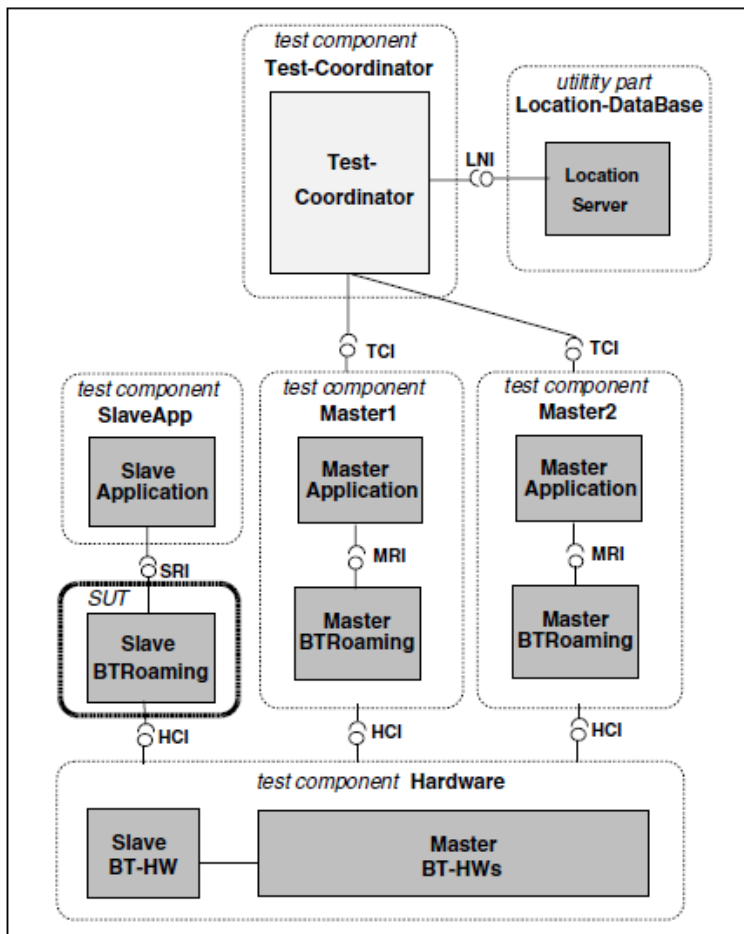
Tesztelendő rendszer:



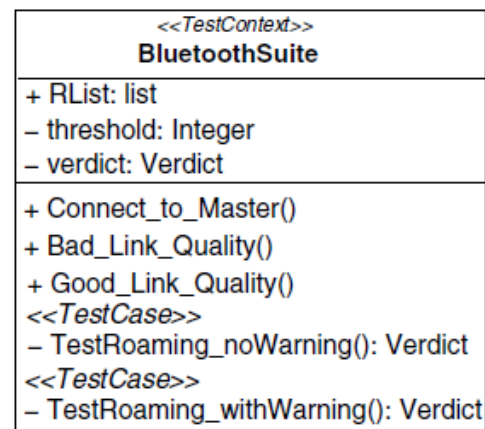
Teszt cél:

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

Példa: Komponensek szerepei



Test package

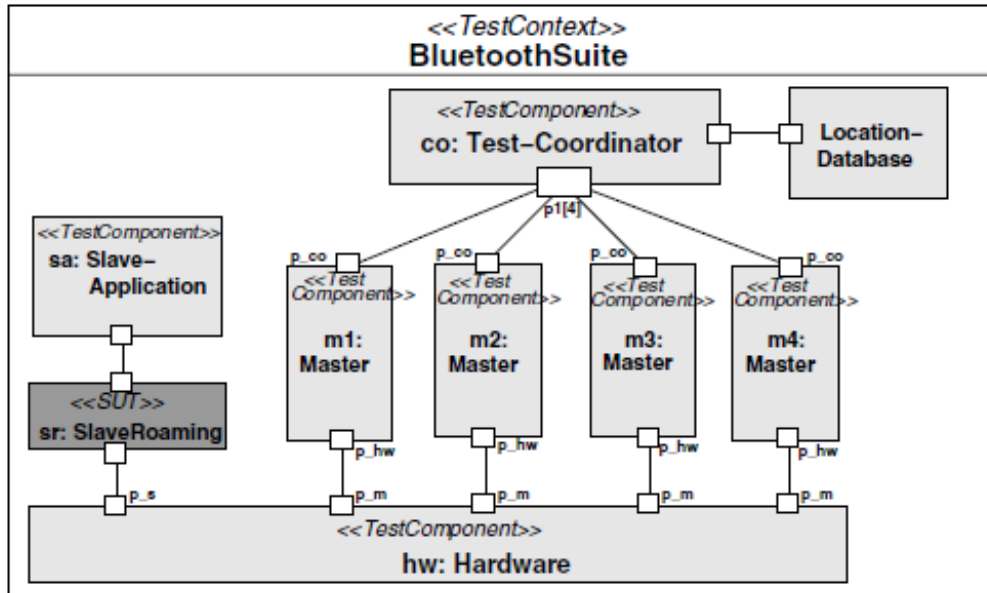


- . System Under Test (SUT)
- . Test Component with new class
- . Test Components with existing classes

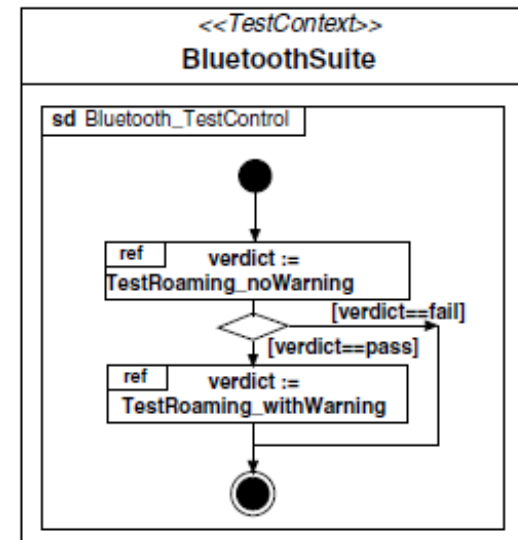
Overview

Test context

Példa: Teszt konfiguráció és teszt vezérlés



Test configuration

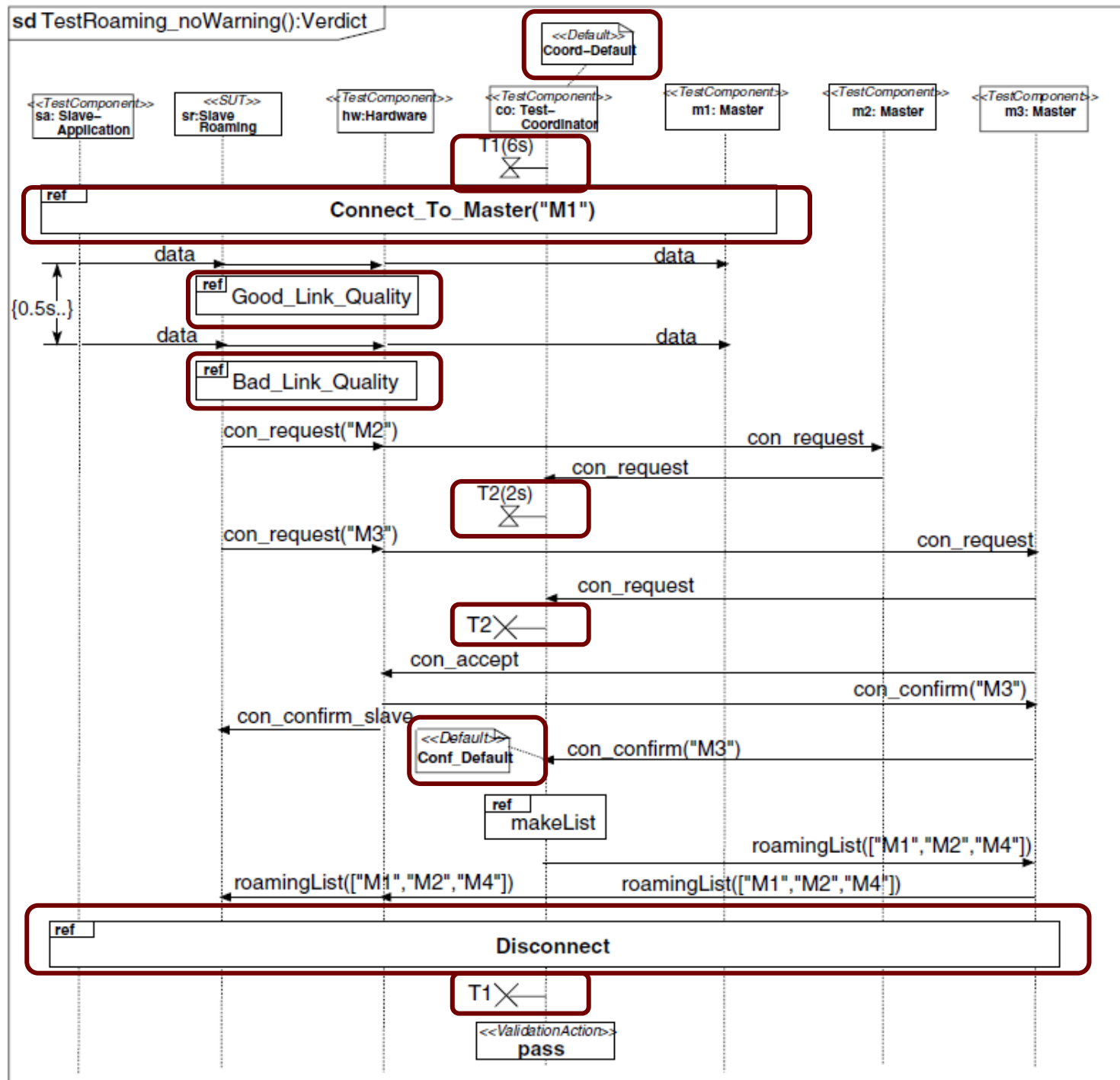


Test control

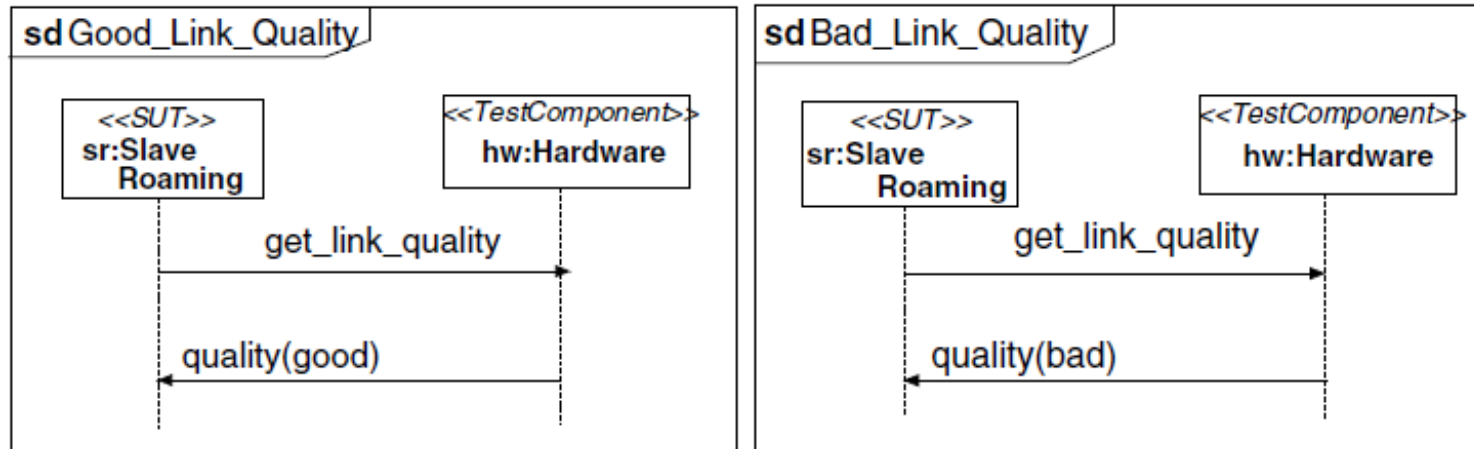
Példa: Teszt scenario

Test case
implemen-
tation
(see Blue-
ToothSuite)

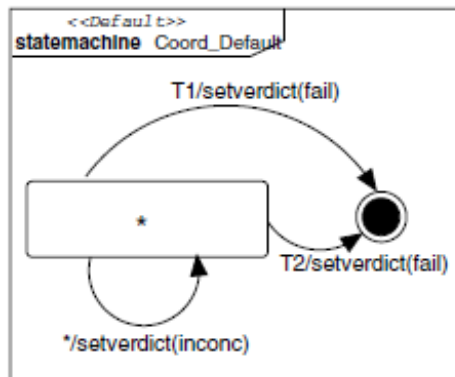
- References
- Timers
- Defaults



Példa: Néhány hivatkozott részlet



Sequence diagrams



- Default behaviours specified to catch the observations that lead to verdicts
- Here: Processing timer events at the TestCoordinator