

# Speciális tesztelési feladatok

Majzik István

Budapesti Műszaki és Gazdaságtudományi Egyetem

Méréstechnika és Információs Rendszerek Tanszék

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

# Tartalom

- Objektum-orientált rendszerek tesztelése
- Felhasználói felületek tesztelése
- Hibakezelés tesztelése hibainjektálással
- Robusztusság tesztelés

# OO rendszerek tesztelése

Problémák a „hagyományos” módszerek esetén:

- Információrejtés:
  - Láthatóság, hozzáférhetőség problémás
- Öröklődés:
  - Megváltozott környezet az öröklött metódusok esetén
  - Inkrementális tesztelés szükséges
- Polimorfizmus:
  - Eldönthetőség kérdéses
- Fedettségi mértékszámok használata
  - „Kódsor fedettség” az öröklődés miatt nem alkalmas

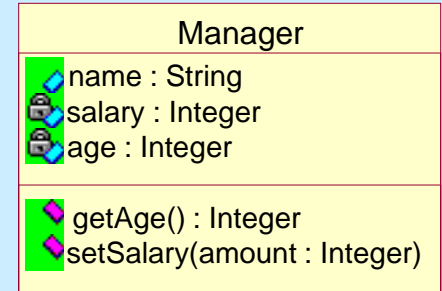
# Információrejtés

- Jellegzetes probléma:
  - Az objektumok állapota csak a publikus metódusokon keresztül hozzáférhető
- Megoldások:
  - Teszt pontok: extra (lekérdező) metódusok hozzáadása
    - Beavatkozó
  - Leszármazott osztály: extra metódusok beillesztése itt
    - Nem beavatkozó, de nem mindenhez férhet hozzá (ld. 'private' C++ esetén)
  - Nyelv-specifikus megoldások, wrapper osztályok
    - friend (C++), child unit (Ada), inspector (Eiffel)
    - ellenőrizetlen típuskonverzió publikussá tett osztályokba...

# Információrejtés

- Jellegzetes probléma:

- Az objektumok állapota csak a publikus metódusokon keresztül hozzáférhető



- Megoldások:

- Teszt pontok: extra (lekérdező) metódusok hozzáadása

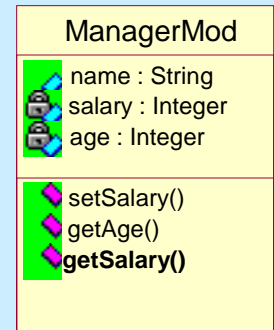
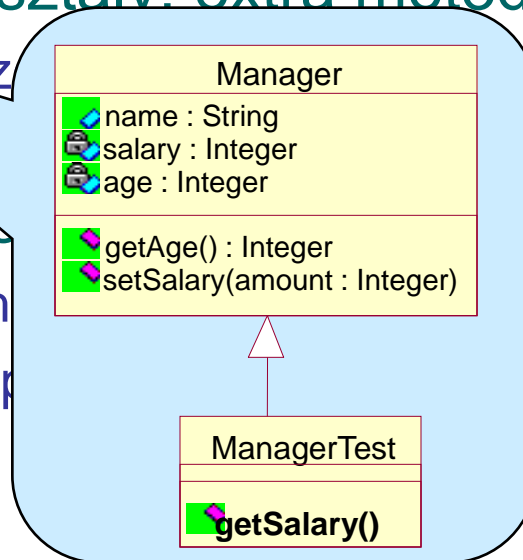
- Beavatkozó

- Leszármazott osztály: extra metódusok

- Nem beavatkozó (ld. 'private')

- Nyelv-specifikus

- friend (C++), ch... (Eiffel)
- ellenőrizetlen típusú osztályokba...



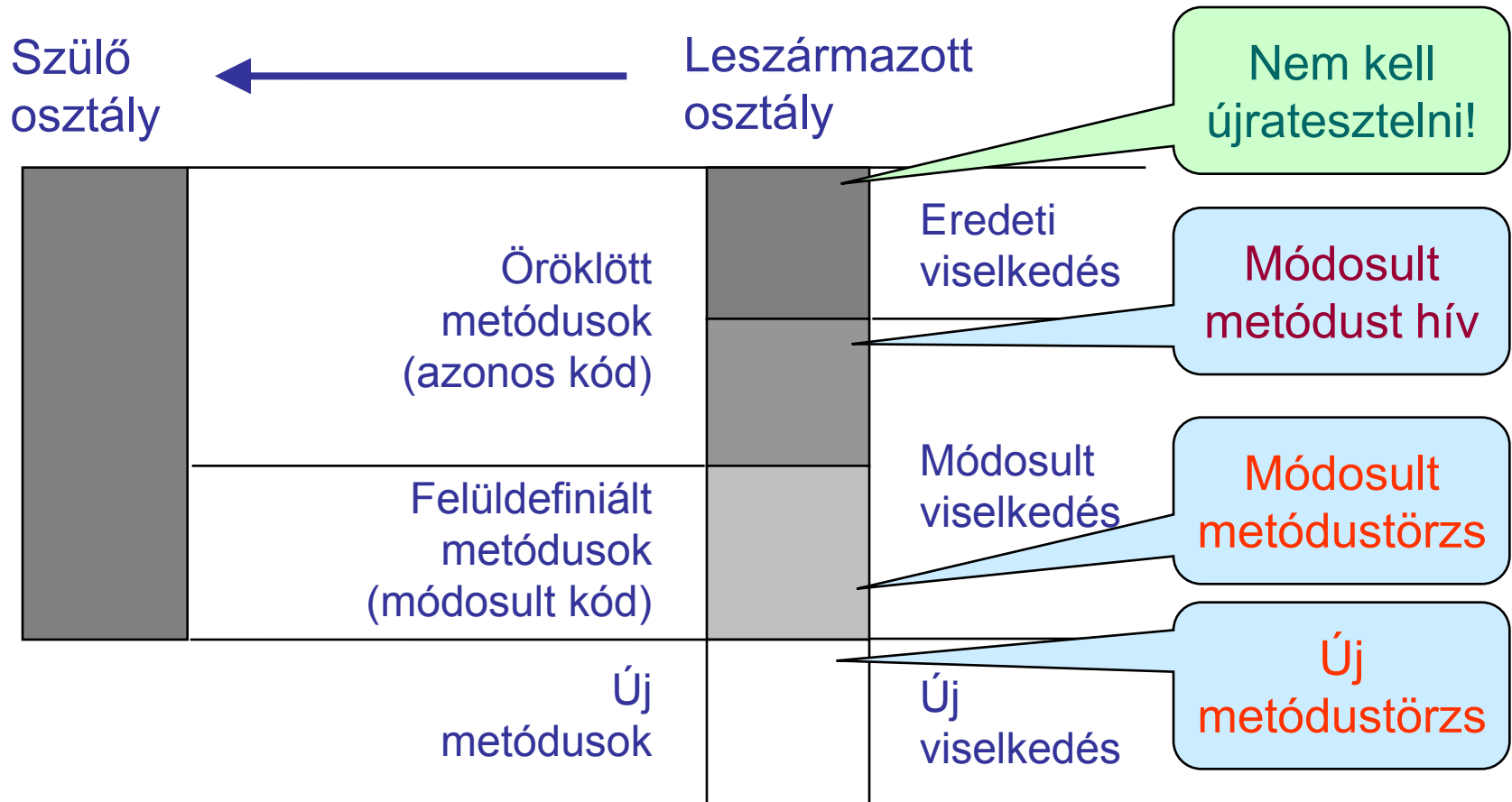
# Nem példányosítható osztályok

- **Probléma:**
  - Absztrakt osztályok (nemdefiniált implementáció)
  - Parametrikus osztályok (felparaméterezhető)
- **Megoldás:**
  - Önmagukban nem tesztelhetők
  - Funkcionális tesztelés  
„leszármazottakon” keresztül

# Öröklés

- **Intuíció:**
  - Öröklött metódusokat nem kell tesztelni (az ősből tesztelve volt)
- **Általánosan nem alkalmazható megközelítés:**
  - Felüldefiniálás
  - Más környezet, más hívott metódusok
- **Öröklés típusai a viselkedés szempontjából:**
  - **Szigorú:** Megtartja a szülő viselkedését + kiegészítheti
    - Az új metódusok megváltoztathatják az állapotot!
  - **Szintaktikai:** Interfész marad, viselkedés átdefiniálható
    - Átdefiniált metódusok újratestelendők + a hívók is!
  - **Implementációs:** Nem mindent örököl
    - Hivatkozások a nem örököltekre megváltoznak

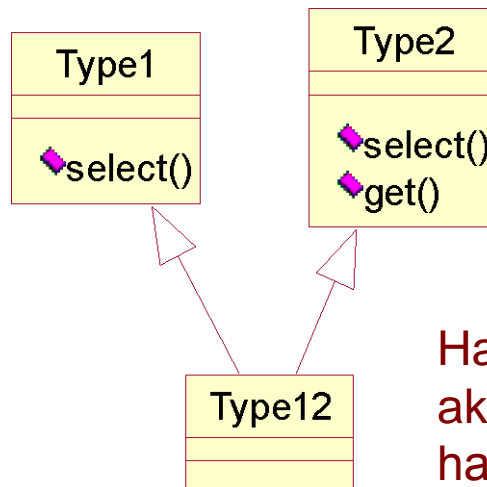
# Újratesztelés öröklés esetén





# Többszörös öröklődés

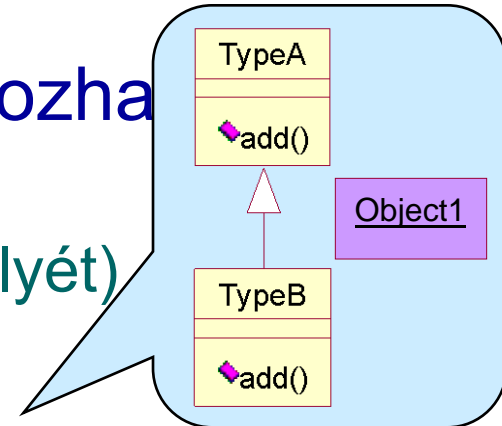
- Az előbbi problémák fokozottan jelentkeznek
- Homográf műveletek: Azonos név és profil
  - Fel kell oldani a kétértelműséget (fordító vagy explicit módon a programozó)
  - Egyik homográf művelet a másikat „felülírja”, még a másik osztályból öröklött műveletekben is!



Ha a Type1-beli select() az „erősebb”, akkor a Type2-ből öröklött get() is azt használja Type12-ben.

# Polimorfizmus

- Egy változó vagy referencia különböző osztályok példányait hordozhatja
  - Öröklés által korlátozva (a leszármazott foglalhatja el a szülő helyét)
- Dinamikus kötés:
  - Polimorf metódus esetén futás közben derül ki, milyen kódrészlet fog lefutni (pl. szülő, vagy a leszármazott átdefiniált metódusa?)
- Polimorf metódus tesztelése:
  - „Minimális funkcionalitás” feltételezése, ami megmarad az átdefiniálás során is?
  - Implicit elágazás (az előélet figyelembe vétele)



# OO teszt fedettségi mértékszámok

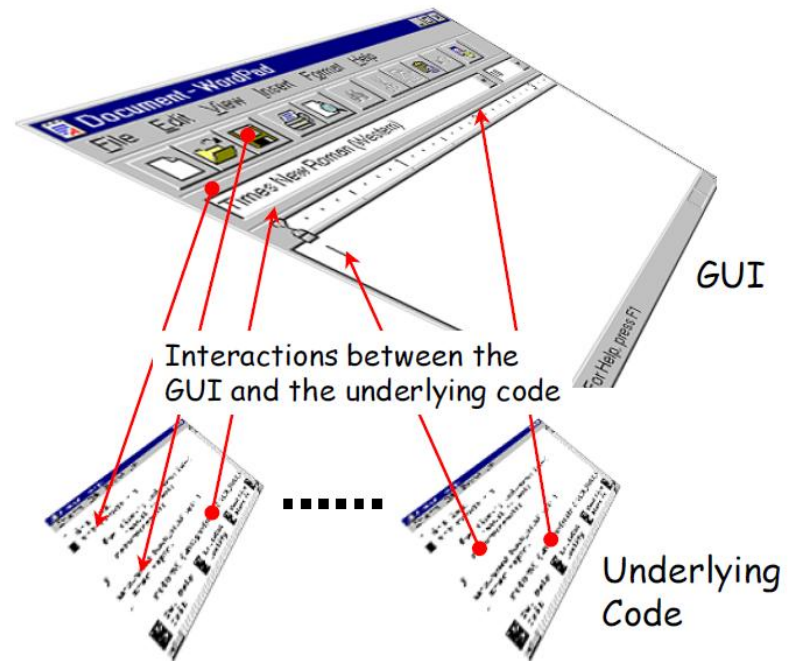
- Környezetfüggő mértékszámok szükségesek
  - „Forrás sorok” fedettsége hamis eredményt adna
  - Ha egy metódus a szülőben tesztelt, még a leszármazottban is tesztelni kell (pl. átdefiniált metódusokat használ)
- Környezetfüggő mértékszámok:
  - Öröklésfüggő fedettség: környezet = osztály
    - A metódus minden (örökölt) osztályban külön tesztelendő
  - Állapotfüggő fedettség: környezet = objektum állapot
    - A metódushívás minden állapotban külön tesztelendő
  - Szálfüggő fedettség: környezet = szál
    - A metódusok minden szálaban külön tesztelendők

# Tartalom

- Objektum-orientált rendszerek tesztelése
- Felhasználói felületek tesztelése
- Hibakezelés tesztelése hibainjektálással
- Robusztusság tesztelés

# GUI jellegzetességek

- Felhasználói utasítások fogadása, eredmény megjelenítése
  - Kommunikáció grafikus elemeken keresztül
  - A program „logikájához” nem ad hozzá
- Eseményvezérelt működésű
  - Manipuláció főként egérrel
- Implementációk
  - GUI toolkit (Qt, GTK+, Swing, WinForms, ...)
  - Webes GUI



# Mit tesztelünk?

- Funkció engedélyezett/tiltott
- Láthatóság
- Fókusz
- Méretezés
- Elhelyezés
- Keretrendszer eseményei
- Adatvalidáció
- Frissesség
- Konzisztencia
- ...

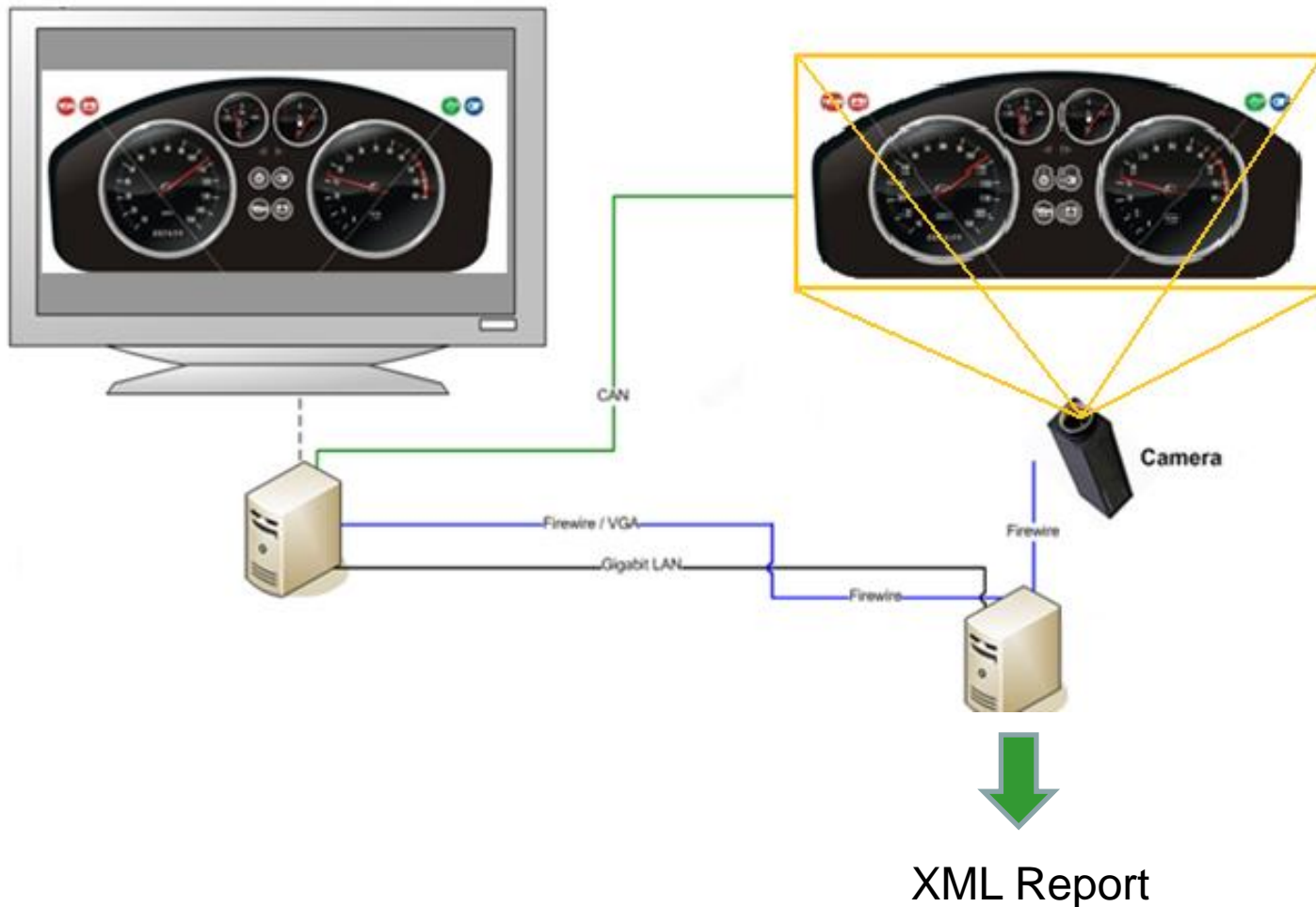
# Tesztelési nehézségek

- Informális követelmények
  - Minták, ergonómiai ajánlások, konvenciók
- Nagyszámú bemenet, nagy állapottér
  - Ugyanabban a kontextusban sokféle esemény
  - Ugyanaz az esemény sokféle kontextusban
  - Váratlan események
- Bonyolult GUI funkciók (toolkit mint „fekete doboz”)
  - Rejtett, nem dokumentált funkciók
- Nehéz teszt végrehajtás
  - Egérkattintások reprodukálása
- Nehéz kiértékelés
  - Grafikus felület változásai és háttér működés azonosítása

# Példa: Teszt környezet felhasználói felülethez

Prototípus (specifikáció, PC)

Implementáció (mikrokontroller)





# Teszt típusok

- Ellenőrző lista
  - Jól „néz ki” az ablak? Könnyen hozzáférhetőek az információk?
- Állapot alapú (belső állapotok)
  - Ha ezt megnyomom, akkor elérhetővé válik / tiltott lesz a másik funkció?
- Integrációs (több modulra)
  - Ha ezt megnyomom, megnyílik a link a böngészőben?
- Kommunikációs (modulok között)
  - Ha ezt megnyomom, végrehajtódik a másik modulban a parancs?
- Szinkronizációs
  - Ha ezt a nézetet megváltoztatom, megváltozik a másik is?
- Terheléses
  - Milyen gyakran kattinthatok rá?
- Kompatibilitást vizsgáló
  - Más alkalmazásokat nem befolyásol?
  - Más platformon is ilyen jól működik?

# Szisztematikus tesztelés előfeltételei

- GUI modell felvétele

- Előnyök:

- Teszt fedettség definiálható
    - Automatikus tesztgenerálásra is lehetőséget ad

- Két tipikus GUI modell:

- Operátor alapú GUI modell
    - Állapotgép alapú GUI modell

- Teszt módszer rögzítése

- Előnyök

- GUI modellhez való illeszkedés biztosítható
    - Ad-hoc megoldásoknál jobban kézben tartható

- Két példa:

- Scenario alapú tesztelés: (Leggyakoribb) használat tesztelése
    - Kombinatorikus tesztelés: Teljes fedésre ad lehetőséget

# I. Operátor alapú GUI modell

- Program objektumok
  - Háttérműködés elemeihez kötött (pl. szövegrészek, fájlok, ...)
- Események
  - Menü események (MM)
    - Műveletek kibontása (pl. File/Save)
  - Fókusz kiterjesztő/kisajátító események (FKE)
    - Munkaablakok (pl. eszköztárak) megjelenítése, új ablak nyitása
  - Rendszerkapcsolati események (RKE)
    - Program objektumok (háttér állapot) megváltoztatása
- Operátorok
  - Rendszerkapcsolati operátorok: (MM,FKE)\*RKE
    - Program objektumok befolyásolása (pl. Edit/Cut és hatása)
  - Felületi menü/fókusz operátorok: MM, FKE kombinálása
    - Új ablak nyitása egy művelet hatására (pl. File/Open... esetén)
  - Összetett (absztrakt) operátorok: Elemi operátorok sorozata
    - Pl. fájl kiírása adott könyvtárba

# Scenario alapú tesztelés

## 1. Teszt cél meghatározás

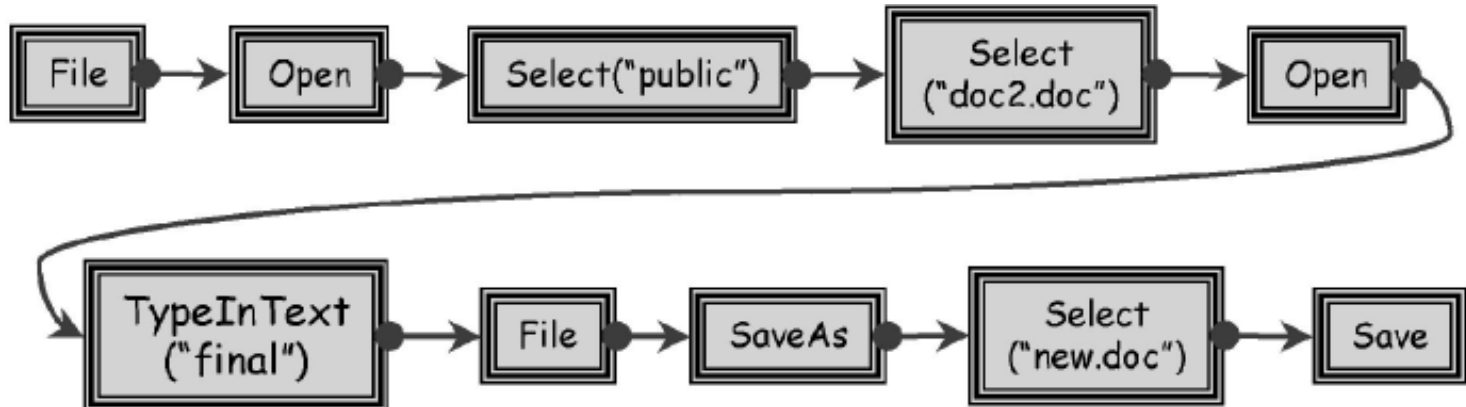
- Operátorok felmérése
- Objektumok felmérése
- Jellegzetes használat (kiindulási állapot, célállapot) meghatározása

## 2. Operátor szekvencia konstruálása

- Jellegzetes (legvalószínűbb) operátorsorozatok lefedése

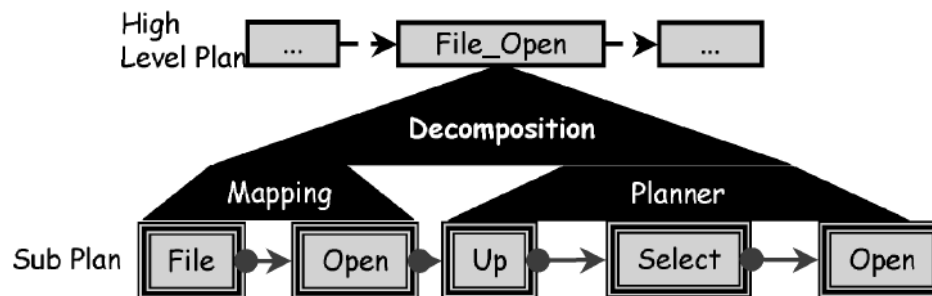
## 3. Konkrét esemény szekvenciára való leképezés

- Tesztesetek generálása
- Összetett operátorok automatikusan is leképezhetők



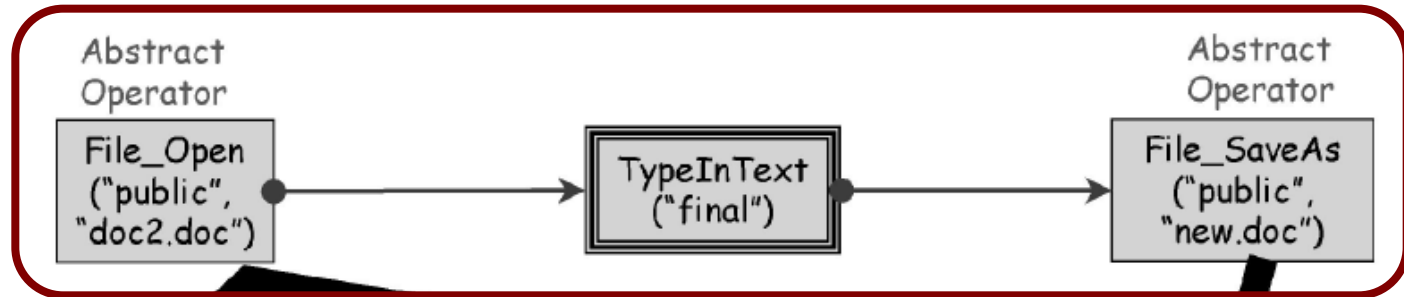
# Scenario generálás tervkészítő segítségével

- A tervkészítés (planner) probléma elemei a GUI teszt generáláshoz:
  - Kezdőállapot: Kiindulási GUI és rendszerállapot (objektumok állapota)
  - Célállapot: Elérendő GUI és rendszerállapot
  - Operátorok (előfeltételek és hatások): GUI események alapján
    - Szabad változókat tartalmazhatnak, hierarchikusak lehetnek
  - Objektumok (lehetnek az operátorok változói): Program objektumok
- Megoldás: Terv (plan): Célállapot elérése a kezdőállapotból
  - Operátor példányok halmaza
  - Részleges rendezési reláció az operátorok között: sorrendi kötöttség
  - Ok-okozati kapcsolatok az operátorok között: hatások és feltételek kötése
  - Operátorok változóinak behelyettesítése: konkrét objektumok
- A terv teljes sorrendezéssel teszt szekvenciaként használható
  - Linearizálás

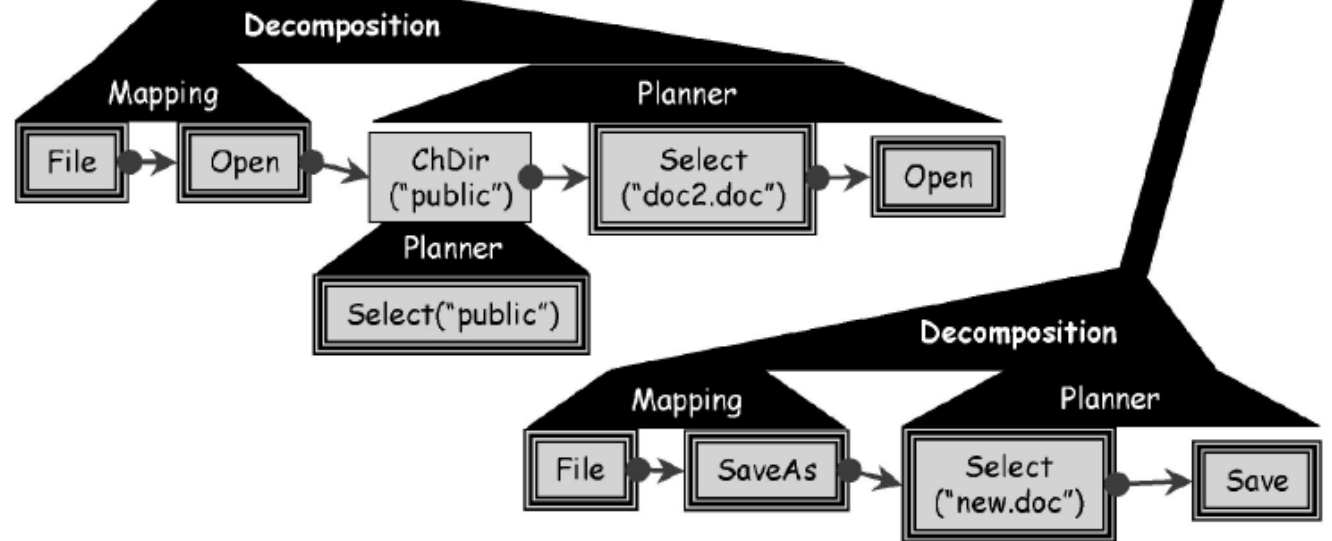


# Példa egy leképzésre

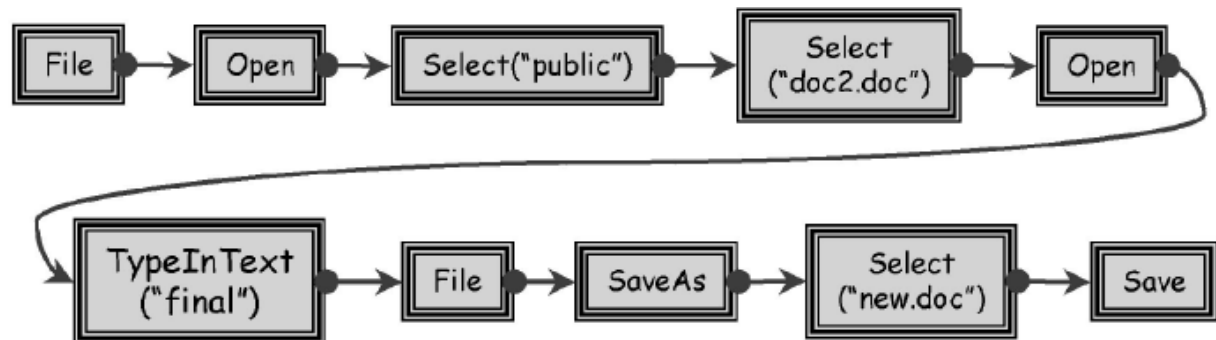
Kiindulás:



Leképzés:

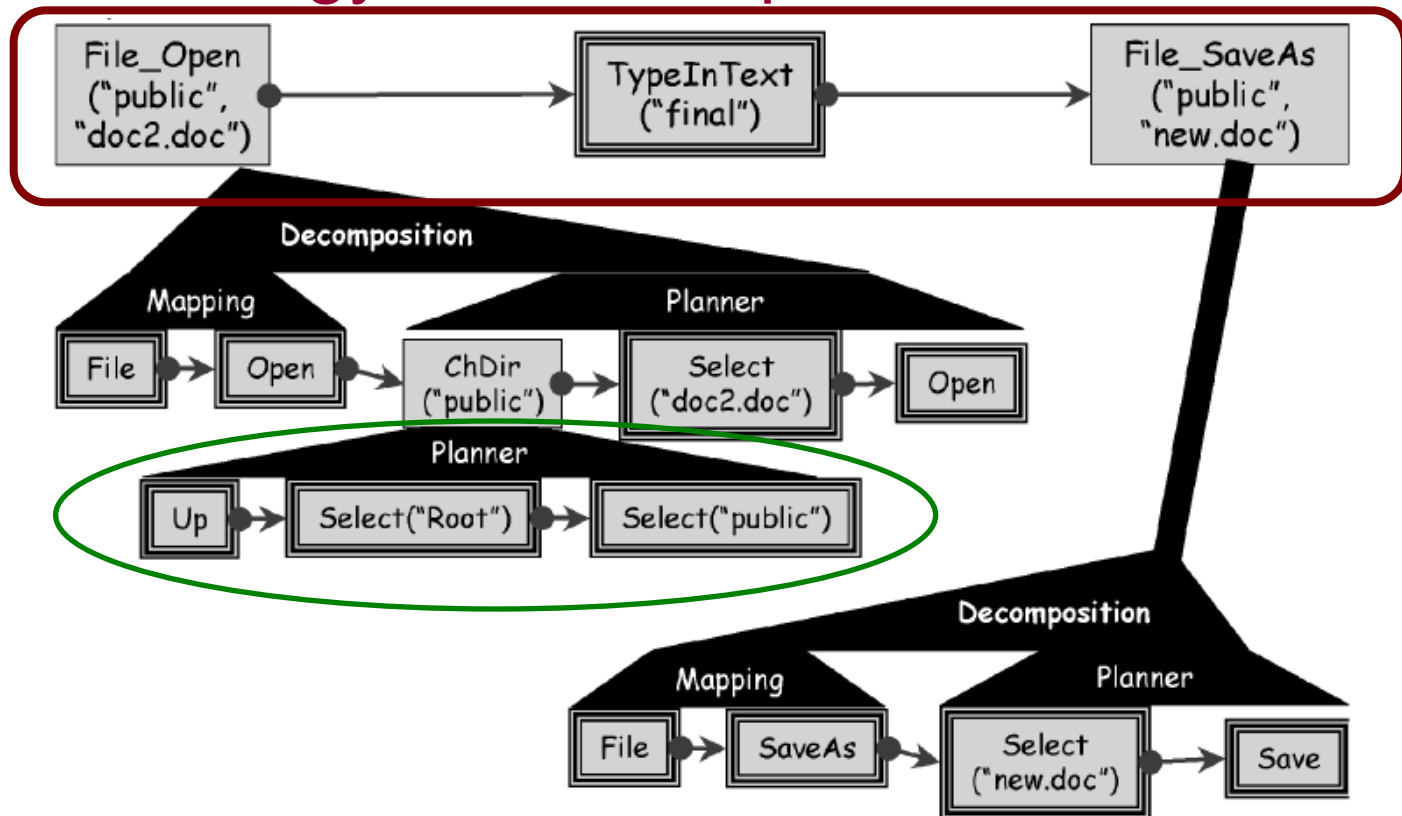


Eredmény:

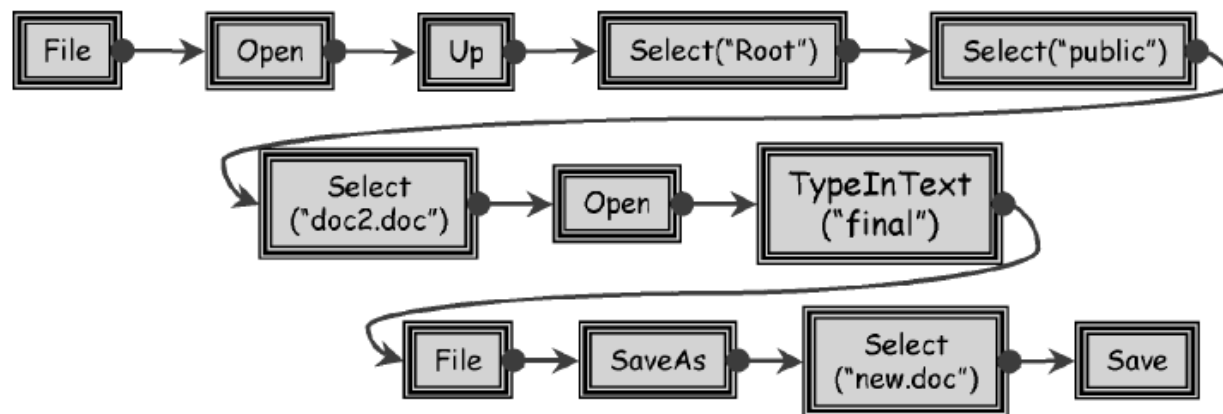


# Példa egy másik leképzésre

Leképzés:

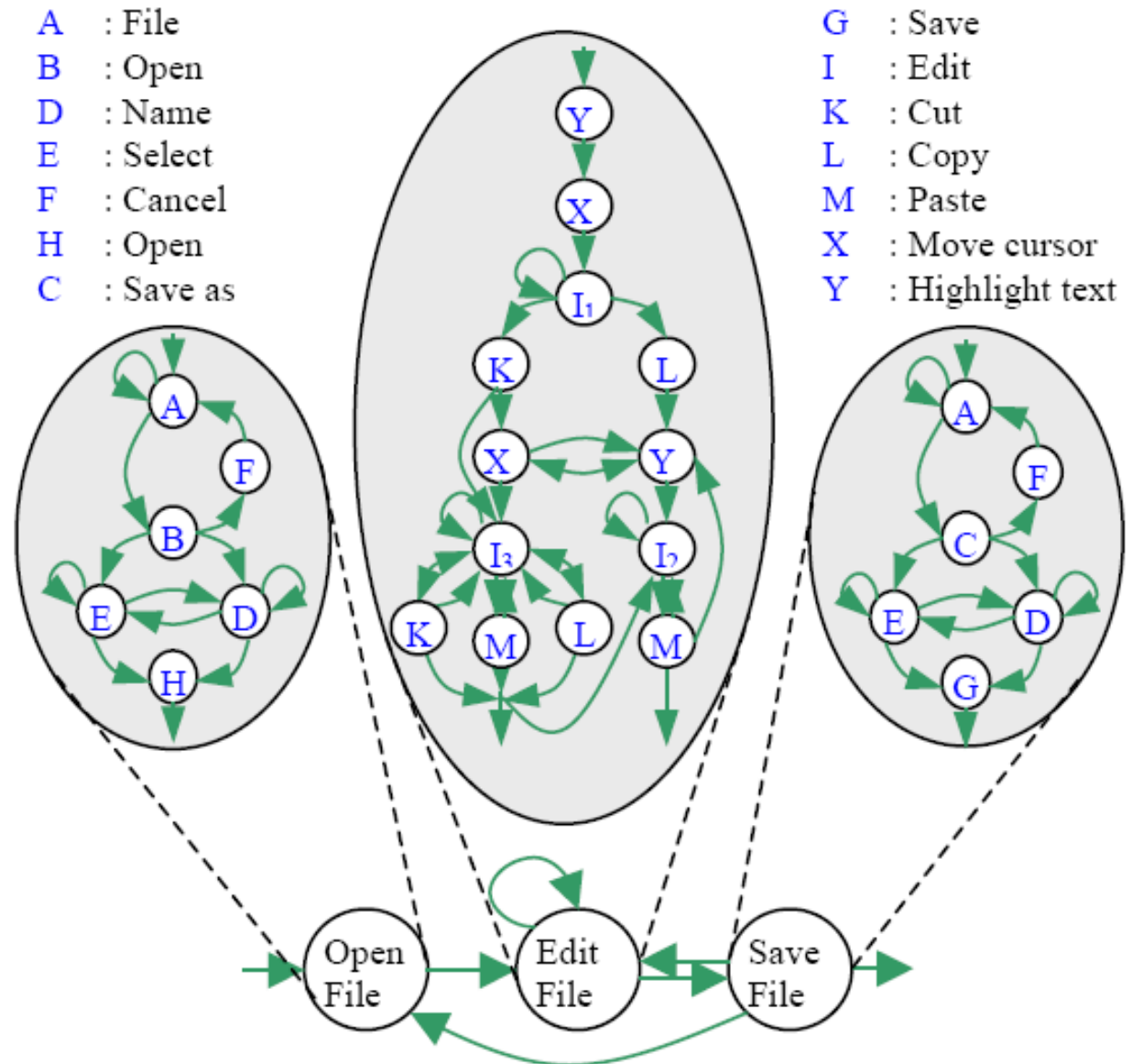


Eredmény:



## II. Állapotgép alapú GUI modell

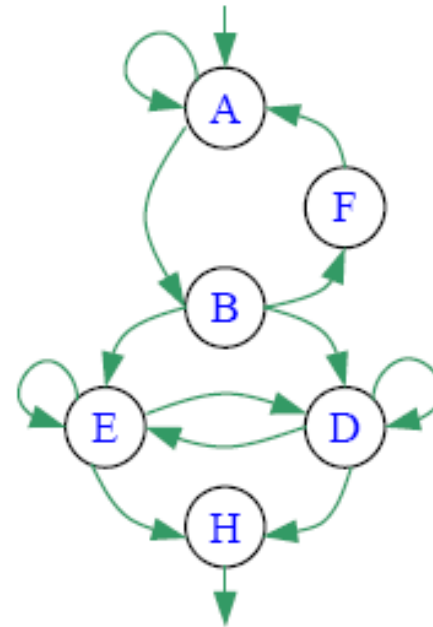
- GUI mint automata
- Esemény folyam (elemi műveletek)





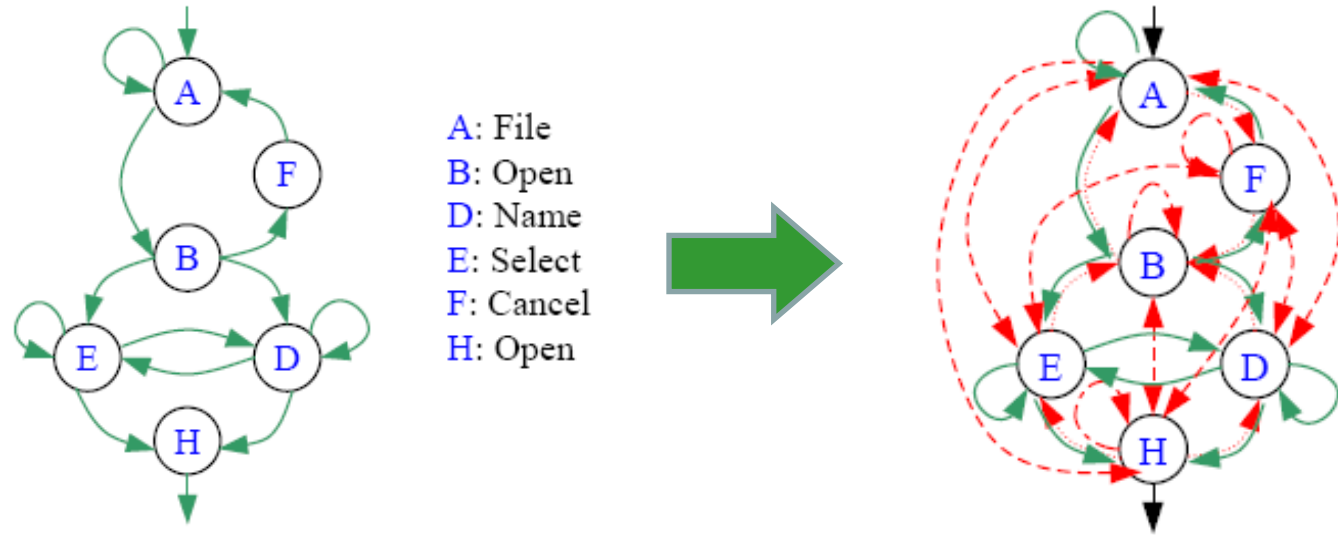
# A normál működés tesztelése

- Tesztelés fedettségi kritériumok alapján:
  - Átmenetek tesztelése
  - Átmenet párok tesztelése
  - Átmenet sorozatok tesztelése
    - Részleges bejárás
    - Teljes bejárás
- Valószínűségi tesztelés
  - Legvalószínűbb bejárásokat előre kell venni
  - Markov modell használható



A: File  
B: Open  
D: Name  
E: Select  
F: Cancel  
H: Open

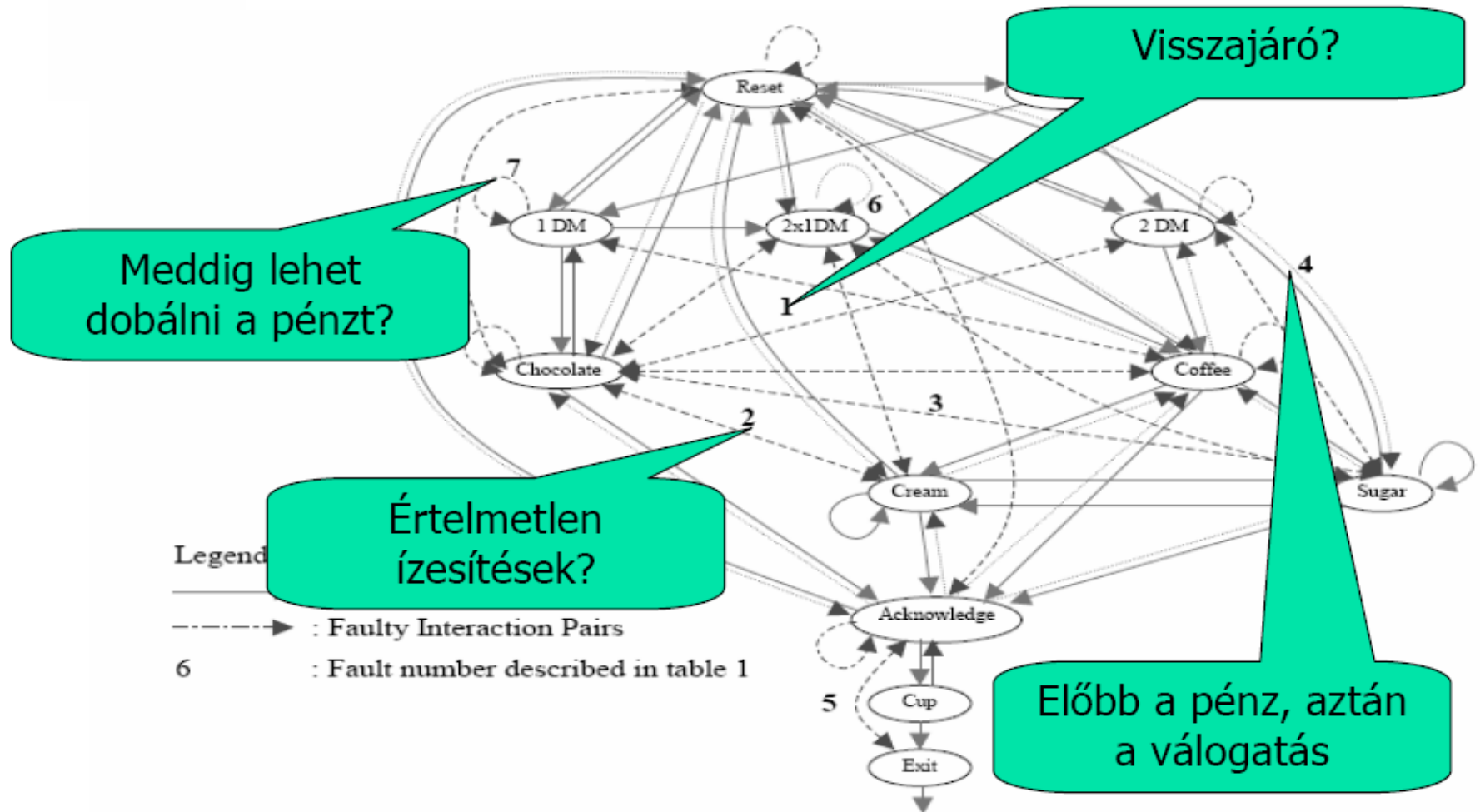
# A nem megengedett működés tesztelése



- Az állapotgép kiterjesztése „helytelen” átmenetekkel:
  - Sorrend megfordítása
  - Plusz önhurkok felvétele
  - Új sorrendi kapcsolatok felvétele (teljessé tétel)
- Helytelen átmenetek tesztelése
  1. Normál átmenetekkel a tesztelendő helytelen átmenetig
  2. Helytelen átmenet végrehajtásának kísérlete
  3. Elvárt hibajelzés vagy nem lehetséges végrehajtás ellenőrzése

# Állapotgép példa: Kávéautomata

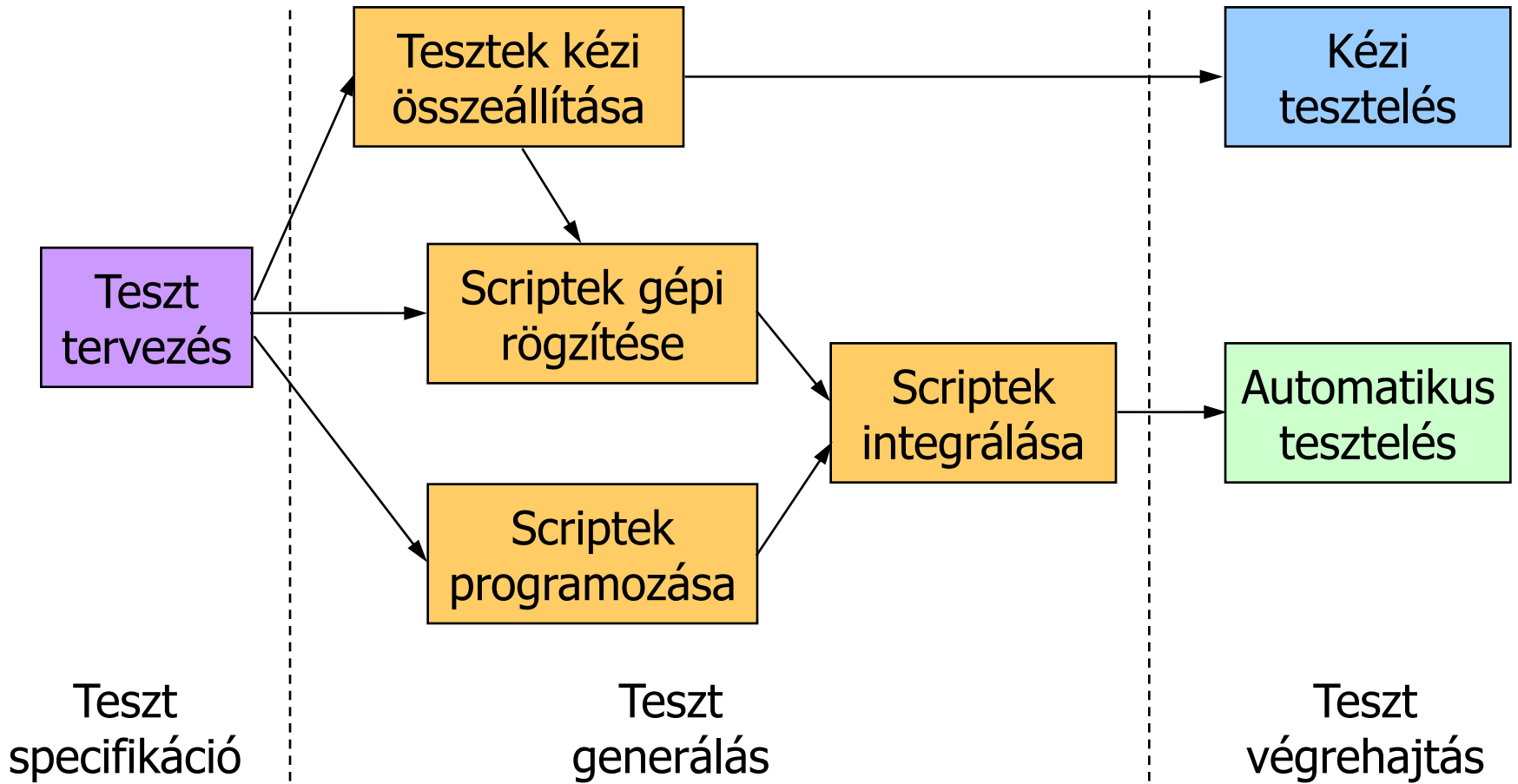
Felderíthető gyakorlati problémák:



# Milyen teszt típusok automatizálhatók?

- **Használhatósági** (nehezen automatizálható)
  - Jól „néz ki” az ablak?
- **Állapot alapú** (automatizálható)
  - Ha ezt megnyomom, akkor elérhetővé válik / tiltott lesz a másik funkció?
- **Integrációs** (az egyszerűbb esetekre automatizálható)
  - Ha ezt a gombot megnyomom, megnyílik a link a böngészőben?
- **Kommunikációs** (az egyszerűbb esetekre automatizálható)
  - Ha ezt a gombot megnyomom, megtörténik az interakció?
- **Szinkronizációs** (nehezen automatizálható)
  - Ha ezt a nézetet megváltoztatom, megváltozik a másik is?
- **Terheléses** (automatizálható)
  - Milyen gyakran kattinthatok rá?
- **Kompatibilitást vizsgáló** (az egyszerűbb esetekre automatizálható)
  - Más alkalmazásokat nem befolyásol?
  - Más platformon is ilyen jól működik?

# Kézi és automatikus tesztelés



# Automatizálási lehetőségek

- **Teszt generálás: Script rögzítés („record”)**
  - Felhasználói interakciók felvétele
  - Felvett script módosítható, többszörözhető
- **Teszt végrehajtás: Script lejátszás („playback”)**
  - Párhuzamosítható
  - Regressziós teszteléshez alkalmazható
- **Eredmény ellenőrzés:**
  - „Szöveg összevetés”: Csak karakteres felületekhez
  - „Képfeldolgozás”: Grafikus felületekhez
    - Widget alapú
    - Bitmap alapú

# Teszt automaták (példák)

Eszköz	Környezet	Licenz
Abbot	Java	CPL
Squish	Java + Web	Commercial (Eval)
SilkTest (Borland)	Multi	Commercial (Eval)
IBM RFT	Multi	Commercial (Eval)
BadBoy	Web	FFNP
JUnit Forms	.Net	BSD
QuickTest (HP)	Java + Windows	Commercial (Eval)
Ranorex	.Net+Web	Mixed
GUIDancer	Java	Commercial (Demo)
GTT	Java	GPL
Jemmy	Java	SPL
JFCUnit	Java	LGPL
Marathon	Java	LGPL
UISpec4J	Java	CPL
QF-Test	Java	Commercial (Eval)
Selenium	Web	Apache 2.0
WET	Web	BSD
Sahi	Web	Apache 2.0

# Példa: Rational Robot



- IBM Rational Functional Tester környezet
- Automatizált feladatok GUI komponensekhez:
  - Teszt szekvencia rögzítése („record”)
  - Teszt értékeléshez referencia (verifikációs pont) kijelölése
    - Menü, ablak, régió, clipboard, fájl, szöveg szintű elemek
    - Image mask megadható kép összehasonlításhoz
  - Teszt script mentése (módosítható SQABasic scriptek)
- Kiindulási információ:
  - (Grafikus) felhasználói felület felderítése, objektumok azonosítása
  - Object mapping: Felhasználói objektumokhoz
- Adatkészlet (data pool) megadható teszt sorozatokhoz
- Felhasználás:
  - Rögzített teszt szekvenciák lejátszása
  - Módosított szekvenciák lejátszása
  - Regressziós tesztelés



# Példa: Selenium



- Selenium IDE: Böngészőn keresztül történő tesztelés webes felületű alkalmazásokhoz
  - Rögzíti a felhasználói interakciókat
    - Módosítás: Szerkesztés, töréspontok
    - Mentés: Ruby, JavaScript, HTML
    - Kód generálás (Java JUnit)
  - Ezek teszteléshez újra lejátszhatók
    - Teszt bemenet: URL megnyitás, kattintás, szövegbevitel, ...
    - Teszt kimenet (assertion): Widget eltűnés, megjelenés, szöveg megjelenés,...
- Selenium Remote Control:
  - A tesztek több böngészőben futtathatók
    - Szerver komponens: Böngészők indítása, HTTP proxy funkció
    - Kliens könyvtár tesztek írásához: Java, PHP, Perl, Python, Ruby nyelvekhez
- Selenium Grid:
  - A tesztek több szerveren futtathatók a párhuzamos tesztelés érdekében
  - Selenium Hub: Több Remote Controlhoz

loginBookstore - Selenium IDE 1.0.7 \*

Base URL: http://localhost:8080/

Fast Slow

Command	Target	Value
open	/bookstore/Login.jsp	
type	userName	user
type	password	user1
clickAndWait	//input[@value='Login']	
assertTextPresent	Welcome, user!	

Command: assertTextPresent |

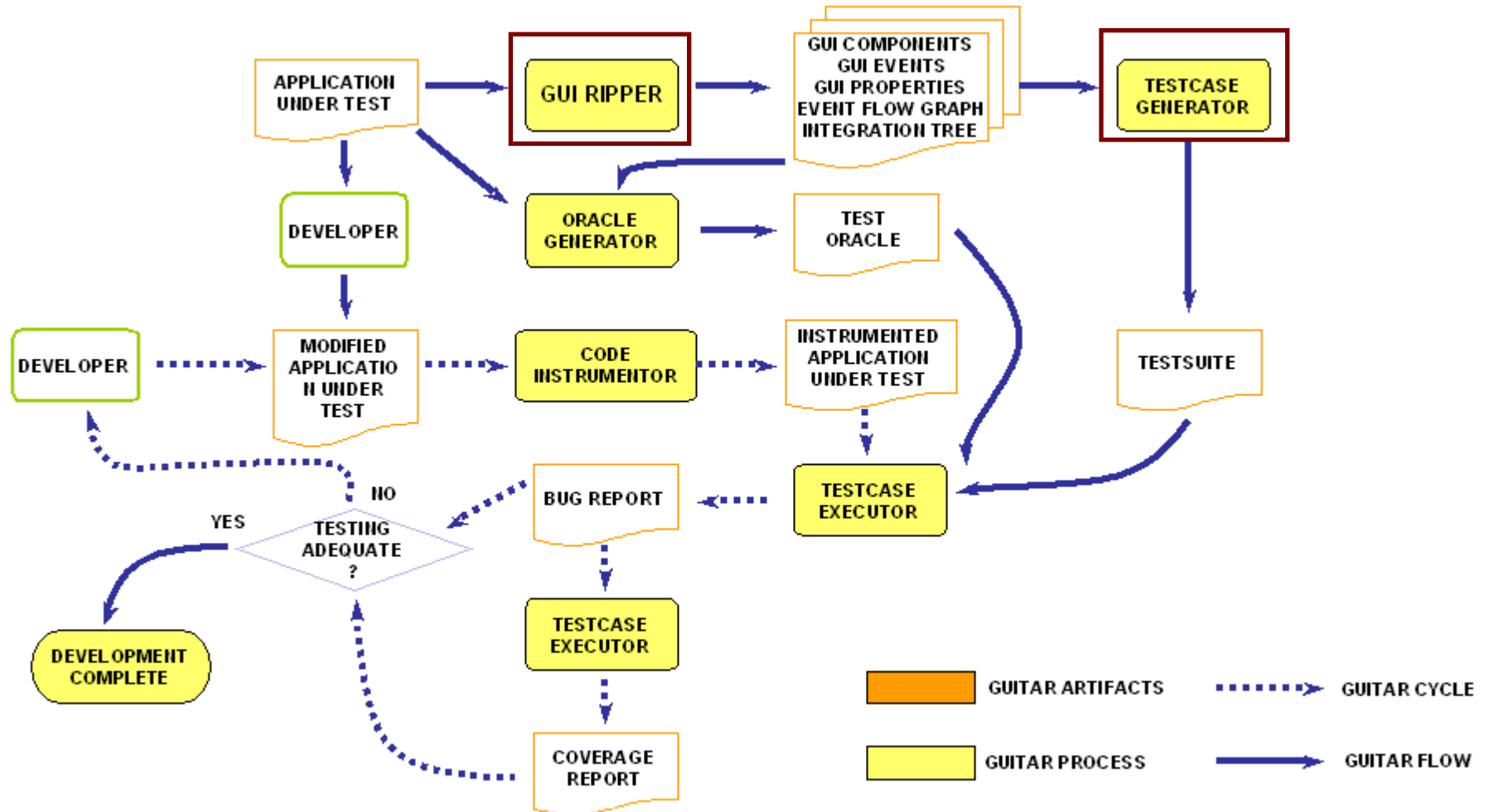
Target: Welcome, user! Find

Value:

Log Reference UI-Element Rollup Info Clear

```
[info] Executing: |type | userName | user |
[info] Executing: |type | password | user1 |
[info] Executing: |clickAndWait |
//input[@value='Login'] | |
[info] Executing: |assertTextPresent | Welcome, user! |
[error] false
```

# Példa: GUITAR - GUI Testing FrAmewoRk



# Tartalom

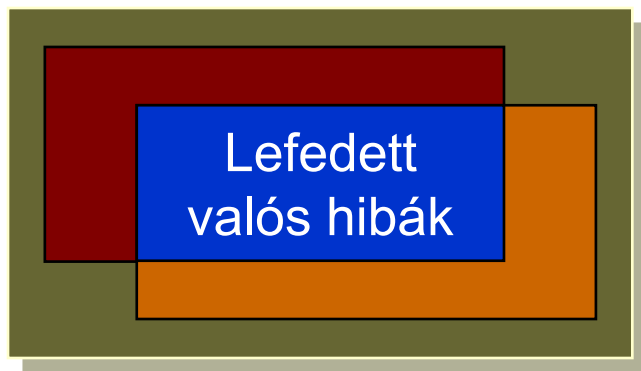
- Objektum-orientált rendszerek tesztelése
- Felhasználói felületek tesztelése
- Hibakezelés tesztelése hibainjektálással
- Robusztusság tesztelés

# Hibainjektálás: Célkitűzések

- Biztonságkritikus rendszerek
  - Hibakezelés tesztelése is szükséges
    - Fail stop rendszerek: Hibadetektálás
    - Fail operational rendszerek: Hibatűrés
- Megvalósítási lehetőségek
  - Valós hibák hatásának megfigyelése (naplózás): Véletlen hibák esetén nehézségek
    - Hosszú idejű működtetés szükséges,
    - vagy nagyszámú megfigyelés szükséges
  - Hibainjektálás: Valóságban várható hibák bevitele
    - Prototípuson vagy modellen elvégezhető
    - Valós hibák „gyorsított módon” (stressz teszteléshez hasonló)

# Hibainjektálás tervezése

- Hibamodell:
  - Egy kép a rendszerben esetlegesen előforduló hibákról
- Hibamodell jósága
  - Feltételezett és valós hibamodell viszonya
  - **Redundáns elemek**: növelik a költséget
  - **Nem lefedett hibák**: rontják a tesztelés minőségét



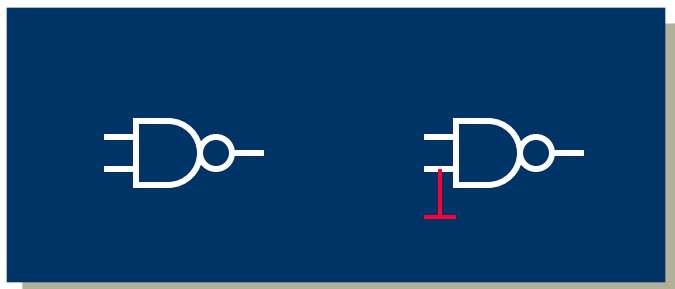
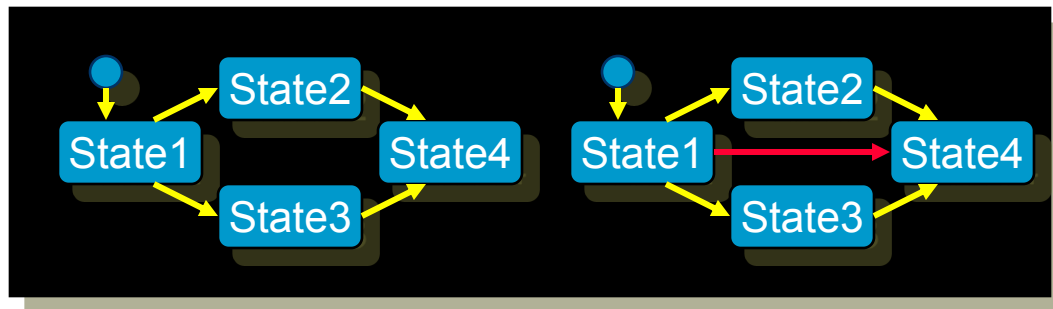
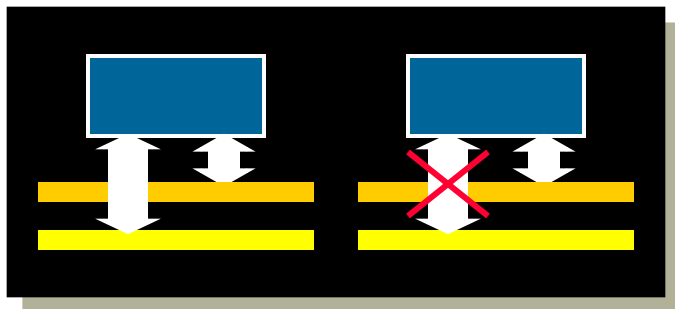
# Hibainjektálás: Módszerek

- **Hardver hibák injektálása:**
  - Hardver úton: Precíz (valós hibaokhoz közeli), de drága
    - Jelek közvetlen módosítása (tápfeszültség is)
    - Besugárzás (nehéz-ion, neutron), hőmérsékleti hatások
  - Szoftver úton: Olcsóbb, de kevésbé valóságos
    - **Hatások** szoftver emulációja: CPU regiszterek, memória módosítása
- **Szoftver hibák injektálása:**
  - Forráskód mutációja (pl. gyakori programozói hibák)
- **Modell alapú injektálás: Szimulációval vizsgálható**
  - Hardver: VHDL, Verilog modellek alapján
  - Szoftver: UML, állapotgép modellek

# Hibainjektáló eszközök

- **Hardver hibák:**
  - Közvetlen hozzáférés jelekhez: RIFLE, GOOFI
  - Besugárzás (nehéz-ion, neutron): speciális kamrák
  - Hatások szoftver emulációja: FIAT, FERRARI, FTAPE
- **Szoftver hibák (fault/error):**
  - Alacsony szintű emuláció: DOCTOR, Xception
  - Kód mutációs eszközök: FINE, DEFINE, G-SWFIT
  - Protokoll rétegek hibái: ORCHESTRA, Neko, WS-FIT
- **Modell alapú injektálás:**
  - VHDL, HDL szint: FOCUS, MEFISTO
  - Komponens szint (CPU, diszk, memória): DEPEND

# A hibák hatásainak emulációja



```
if (EVENT1 == event)
    state = STATE1;
if (EVENT2 == event)
    state = STATE2;
```

Red arrows on the right side of the code block show a flow from the second 'if' statement back to the first, indicating a loop or a return to a previous state.

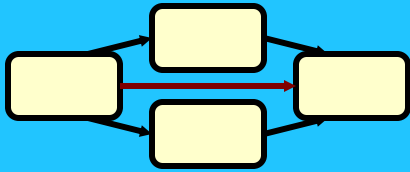


```
cmp eax, 1
je LABEL1
cmp eax, 2
je LABEL2
```

Red arrows on the right side of the code block show a flow from the second 'cmp' statement back to the first, indicating a loop or a return to a previous state.



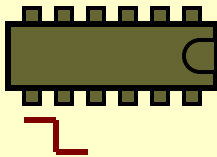
# A hibainjektálás valóságghűsége



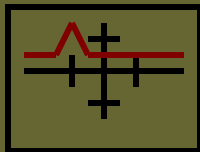
Hibaszimuláció a forráskódban  
vagy modell szinten a fejlesztőeszközben



Regisztartalom módosítása,  
memóriakép felülírása



Síneken haladó vagy áramkörök  
lábán megjelenő jelek módosítása



Radioaktív sugárzás, ioninjektálás,  
tápfeszültség zavarása, hőmérséklet

Ár,  
valóságghűség

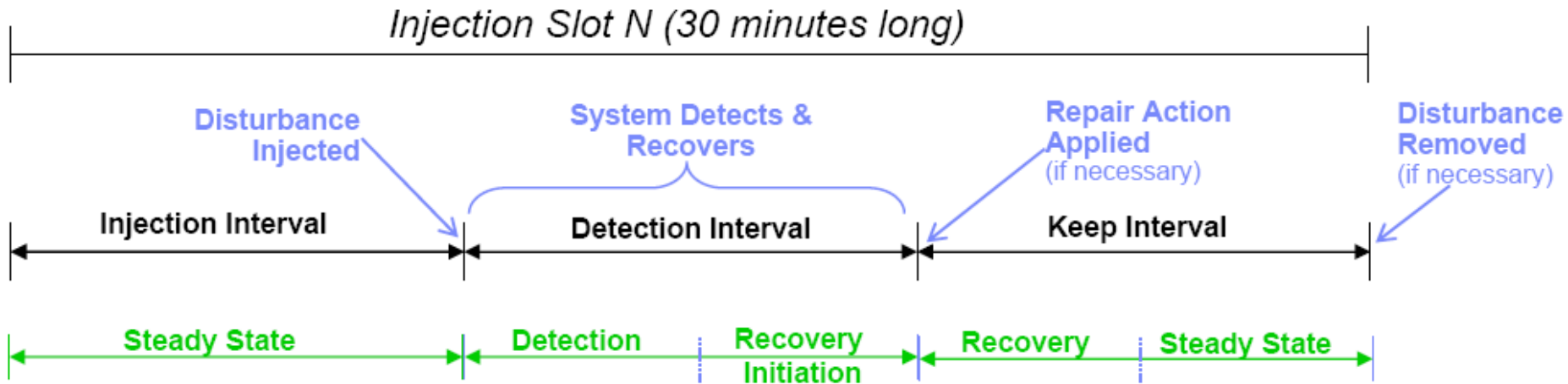


# Hibainjektálási lehetőségek (összefoglalás)

- **Hardver alapú hibainjektálás**
  - Hibaok injektálása közelíthető
    - Besugárzás, elektromágneses zavarok, hőmérséklet
    - Tápfeszültség túske, jelvezetékek leragadása, összeragadása, ...
  - Rugalmatlan, drága, de valóságghűsége jobb
- **Szoftver alapú hibainjektálás**
  - Hibaállapot injektálása (hardver hibaok hatása)
    - Processzor regisztertartalom, memóriatartalom, fájlok, üzenetek,...
  - Mutáció bevitele (vezérlés, adatkezelés)
  - Rugalmas, olcsóbb, de valóságghűsége kisebb
- **Modell alapú hibainjektálás**
  - Tipikusan a komponens szintű hibajelenségek modellezése
    - Funkciók, interakciók megváltozása
  - Tervezési fázisban legkorábban végrehajtható, de a valóságghűség itt is kérdéses (legmagasabb absztrakciós szintű)

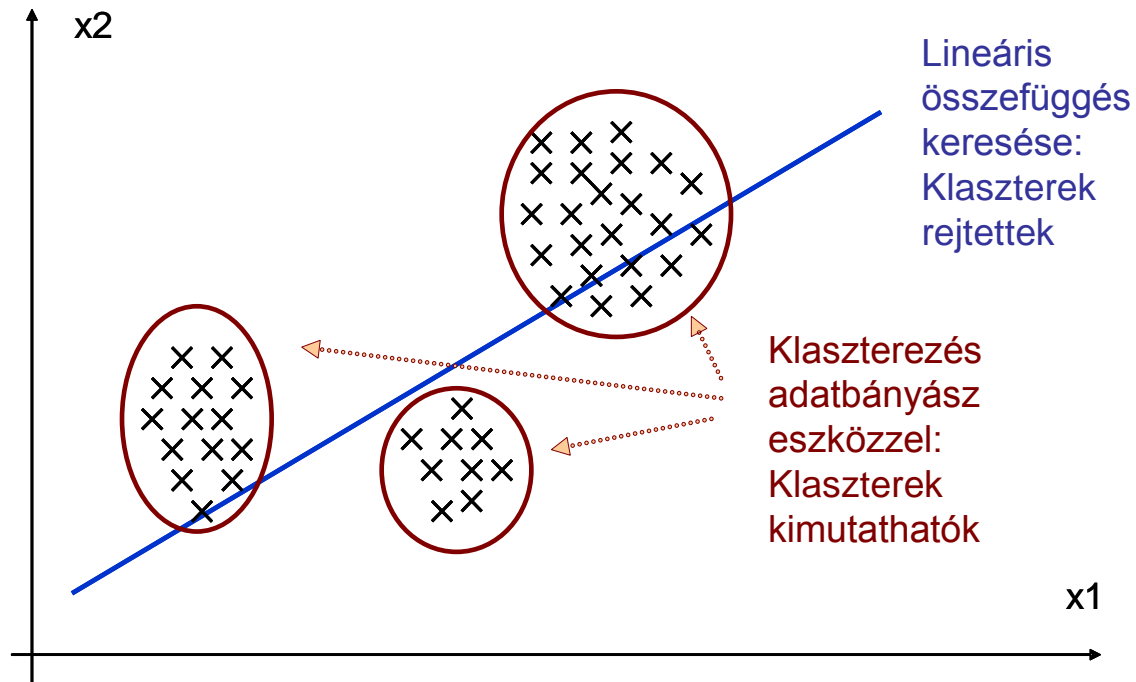
# Hibainjektálási kísérletek végrehajtása

- Példa: Alkalmazás szerver tesztelése
  - IBM Autonomous Computing
- Hibaterhelés:
  - Támadások, operátori hibák
  - Leállások, újraindulások
- Hibainjektálási intervallumok meghatározása:



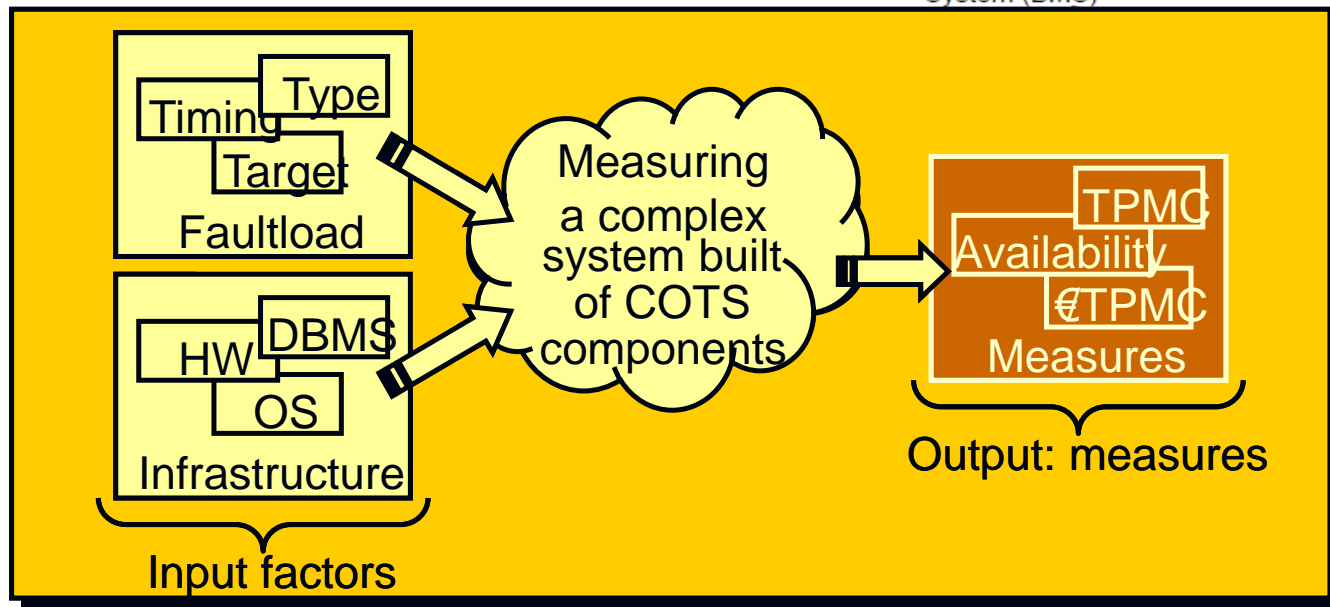
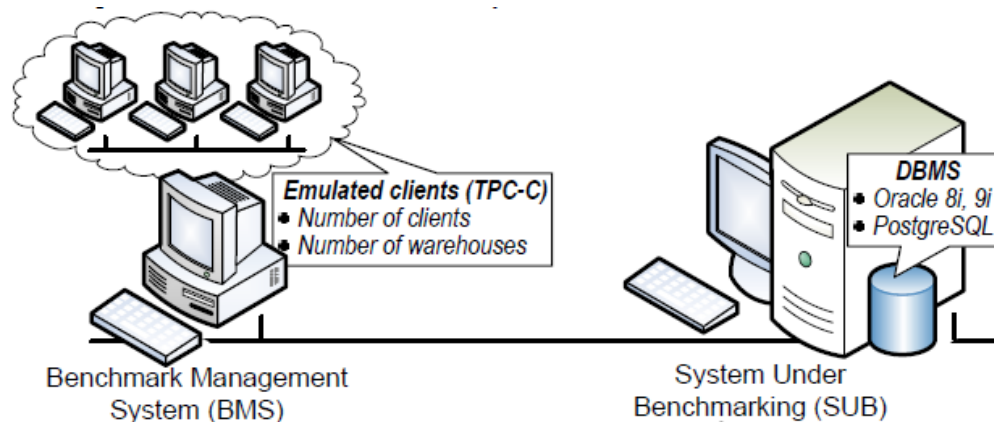
# Eredmények kiértékelése

- Nagy mennyiségű adat
- Feldolgozás:
  - OLAP: On-Line Analytical Processing
  - Adatbányászat: „Rejtett” összefüggések felderítése



# Példa: DBMS hibainjektálás és helyreállítás

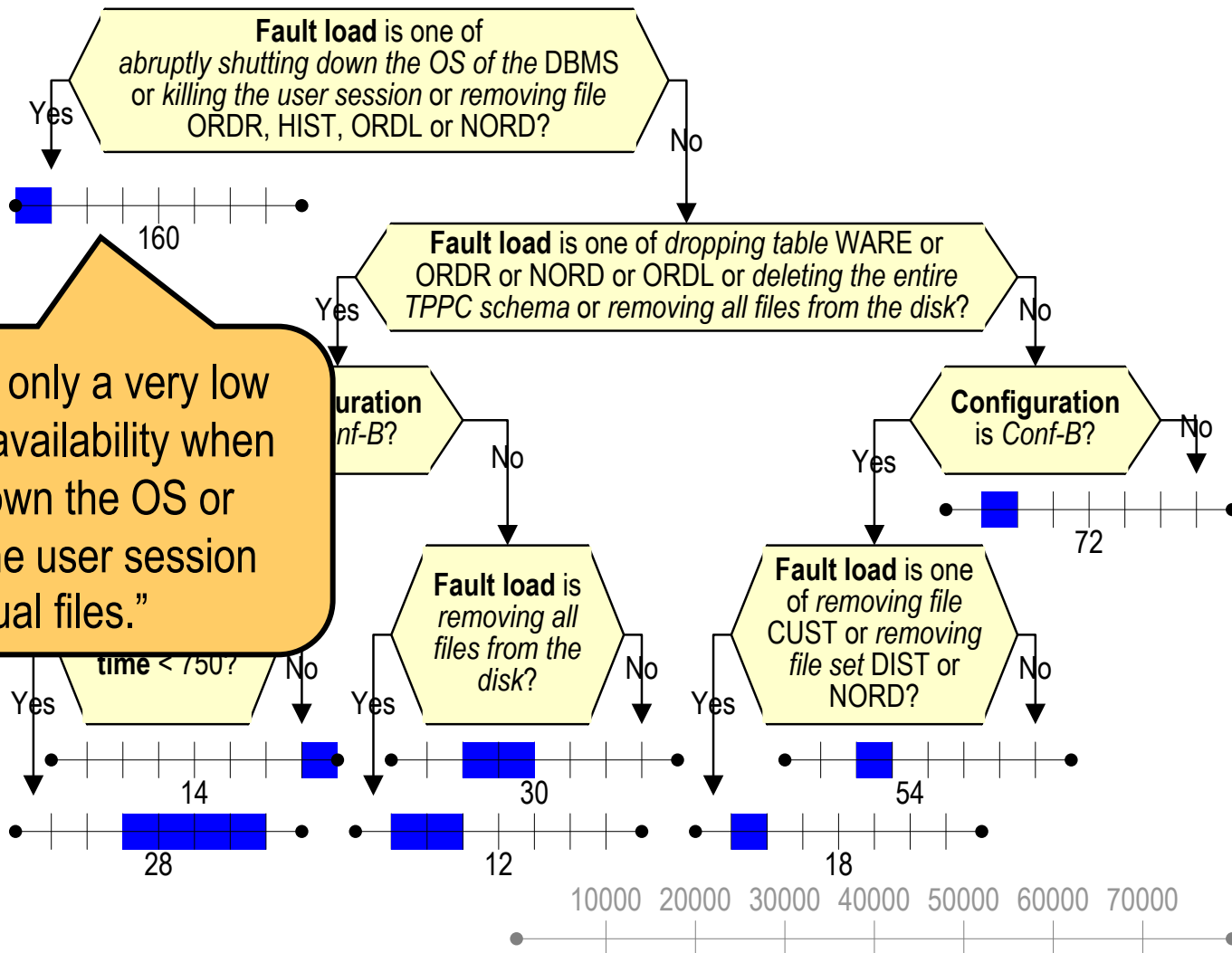
- Mi határozza meg a kiesés idejét?
  - Hiba(típus)
  - Konfiguráció
  - ...



- | Operating system  | Operator faults  |
|---|--|
| <ul style="list-style-type: none"><li>• Windows 2000,</li><li>• Windows XP</li><li>• SuSE Linux 7.3</li></ul> | <ul style="list-style-type: none"><li>• Kill user session</li><li>• Remove data file</li><li>• Drop table, ...</li></ul> |

# Példa: DBMS hibainjektálás és helyreállítás

- Osztályozás a kiesés ideje alapján, döntési fa mutatja a paramétereket



“Clients experience only a very low degradation of the availability when abruptly shutting down the OS or the DBMS, killing the user session or removing individual files.”

## Egy más célú technika: Hibabeültetés

- Hibák direkt bevitele, majd tesztelés indítása
- Becslés:

$$\frac{\text{Megtalált beültetett hibák sz.}}{\text{Összes beültetett hibák száma}} \approx \frac{\text{Megtalált valódi hibák sz.}}{\text{Összes valódi hiba száma}}$$

- Sok feltétel teljesülése esetén jó becslő csak!
  - Beültetett hibák reprezentatívak,  
eloszlásuk megfelel a valódi hibákénak ...

# Tartalom

- Objektum-orientált rendszerek tesztelése
- Felhasználói felületek tesztelése
- Hibakezelés tesztelése hibainjektálással
- **Robusztusság tesztelés**



# Definíciók

## Robusztusság (IEEE Std 610.12.1990):

- „The degree to which a system operates correctly in the presence of
  - exceptional inputs or
  - stressful environmental conditions”
- Annak jellemzője, mennyire helyesen működik a rendszer
  - rendkívüli bemenetek vagy
  - nagy igénybevételt jelentő környezeti feltételek mellett.

## Robusztusság hiba:

- Helytelen (nem elvárt) működés rendkívüli bemenetek és környezeti feltételek esetén

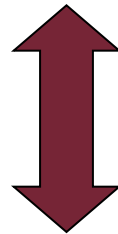
## Robusztusság tesztelés:

- A robusztusság hibák aktiválása a tesztelés során

# Robusztusság tesztelés

## Funkcionális tesztelés

- Specifikációnak **megfelelő** működés vizsgálata
- **Érvényes** bemenet / elvárt kimenet



## Robusztusság tesztelés

- Specifikációtól **eltérő** működés(képtelenség) vizsgálata
- **Extrém vagy hibás** bemenet / elvárt kezelés, hibajelzés

# Bemenetek robusztusság teszteléshez

- **Véletlen bemenetek**
  - Van esélye robusztusság hiba aktiválásának
  - Egyszerű teszt adat generálás, de kis hatékonyság
- **Típus-specifikus bemenetek**
  - Típustól függően előre kijelölt extrém értékek
  - Komplex kombinációk lehetségesek
- **Objektumok mint bemenetek**
  - NULL érték használható extrém értéként
  - Létrehozáshoz extrém paraméterek megadása
- **Scenario mutációval generált bemenetek**
  - Az extrém bemenetek állapottól függően adhatók ki
  - Sorrendi, kihagyási, időzítési hibák is definiálhatók

# Bemenetek generálása típus-specifikus szabályokkal

Type	Parameter Mutation
<b>String</b>	Replace by null value
	Replace by empty string
	Replace by predefined string
	Replace by string with nonprintable characters
	Add nonprintable characters to the string
	Replace by alphanumeric string
	Add characters to overflow max size
<b>Number</b>	...
<b>List</b>	...
<b>Date</b>	...
<b>Boolean</b>	...

# Munkaterhelés (workload) a tesztelés során

- Valódi terhelés
  - Pl. regisztrált scenariók visszajátszása
- Realisztikus terhelés
  - Jellegzetes scenariók generálása
  - Jobban hordozható, kézbentartható
- Szintetikus terhelés
  - Jellemző használat: Tervezett, névleges terhelés
  - Túlterhelés



# Robusztusság teszt kimenetek értékelése

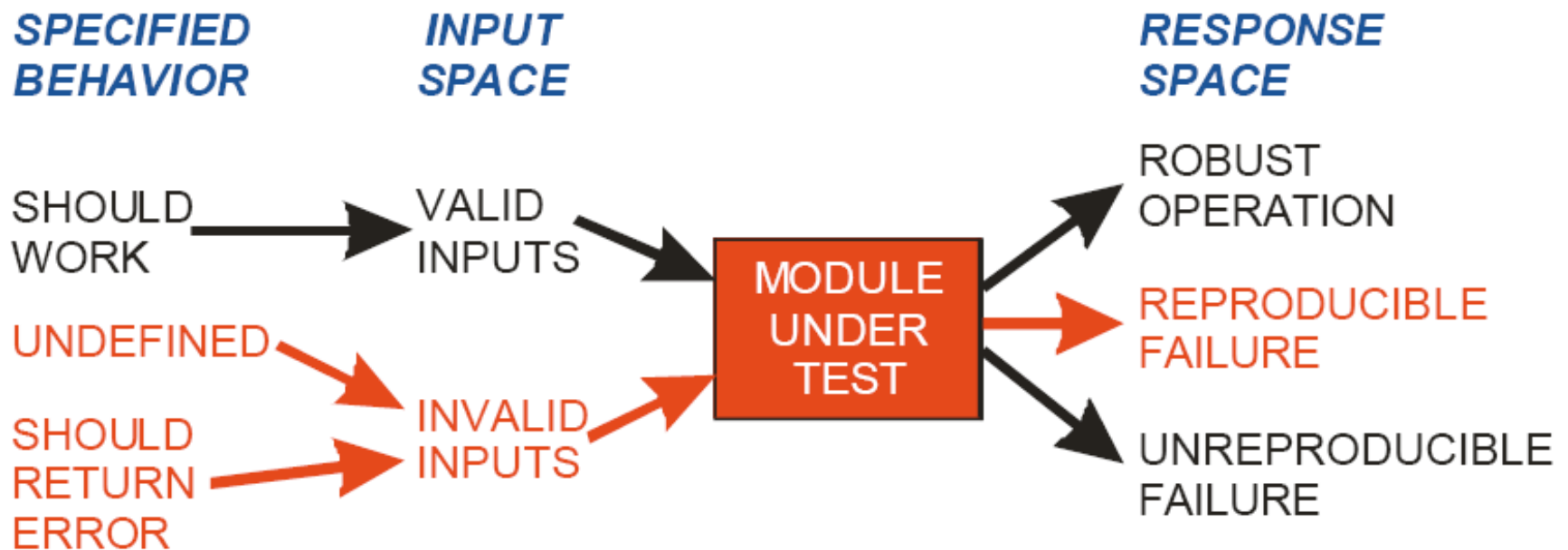
- Specifikációtól eltérő működés
  - Sokszor nincs előírt érték
  - Elvárt eredmények egyszerűsített kezelése szükséges
- Klasszikus kategóriák: **CRASH**
  - **Catastrophic:** A teljes rendszer összeomlik / újraindul
  - **Restart:** Az adott alkalmazás újraindulása
  - **Abort:** Az adott alkalmazás leáll
  - **Silent:** Hibajelzés nélküli érvénytelen művelet
  - **Hindering:** Érvénytelen hibakód
- Nem robusztusság hiba:
  - Érvényes hibakód visszaadása

# Jellegzetes eszközök

- Hardver hibainjektáló eszközök
  - **Közvetett robusztusság tesztelés:** Környezet komponenseibe injektált hibák **hatása** vizsgálható a tesztelt komponensre
  - Belső hibák injektálása nem robusztusság tesztelés!
- Kombinatorikus robusztusság tesztelő eszközök:
  - **Fuzz:** Véletlenszerű bemenetek konzolos alkalmazásokhoz
  - **Ballista:** Típus-specifikus tesztelés POSIX, CORBA hívásokhoz
  - **JCrasher:** Extrém objektumok Java alkalmazásokhoz
- Forráskód mutációs eszközök
  - Funkcionális teszt szekvenciák mutálhatók
  - Szekvencia vagy hívási paraméter megváltoztatása
- Benchmark eszközök
  - **Benchmark:** Reprezentatív, megismételhető vizsgálat
  - **DBench:** Szolgáltatásbiztonság benchmarkok

# Típus-specifikus tesztelés: Ballista – alapelvek

## API tesztelés





# Típus-specifikus tesztelés: Ballista – bemenetek

- Szélső és extrém értékek beállítása

Data type	Substitution values					
Pvoid	NULL	0xFFFFFFFF	1	0xFFFF	-1	Random
Integer	0	1	MAX INT	MIN INT	0.5	
Boolean	0	0xFF (Max)	1	-1	0.5	
String	Empty	Large (> 200)	Far (+ 1000)			

# Típus-specifikus tesztelés: Ballista – kombinációk

API

Sfseek (Sfio\_t \*theFile, int pos)

TESTING OBJECTS

Sfio\_t\*

Int

ORTHOGONAL  
PROPERTIES

File State	Buffer Type	Flags	IntValue
OPN_READ	MAPPED	STRING	MAXINT
OPN_WRITE	<b>BUFFERED</b>	READ	MININT
OPN_RW	NON_BUFFERED	WRITE	<b>ZERO</b>
<b>CLOSED</b>		APPEND	ONE
DELETED		LINE	NEGONE
		SHARE	2
		PUBLIC	4
		<b>MALLOC</b>	8
		STATIC	16
		IOCHECK	32
		BUFCNST	64
		WHOLE	...
		MALLOC_STATIC	

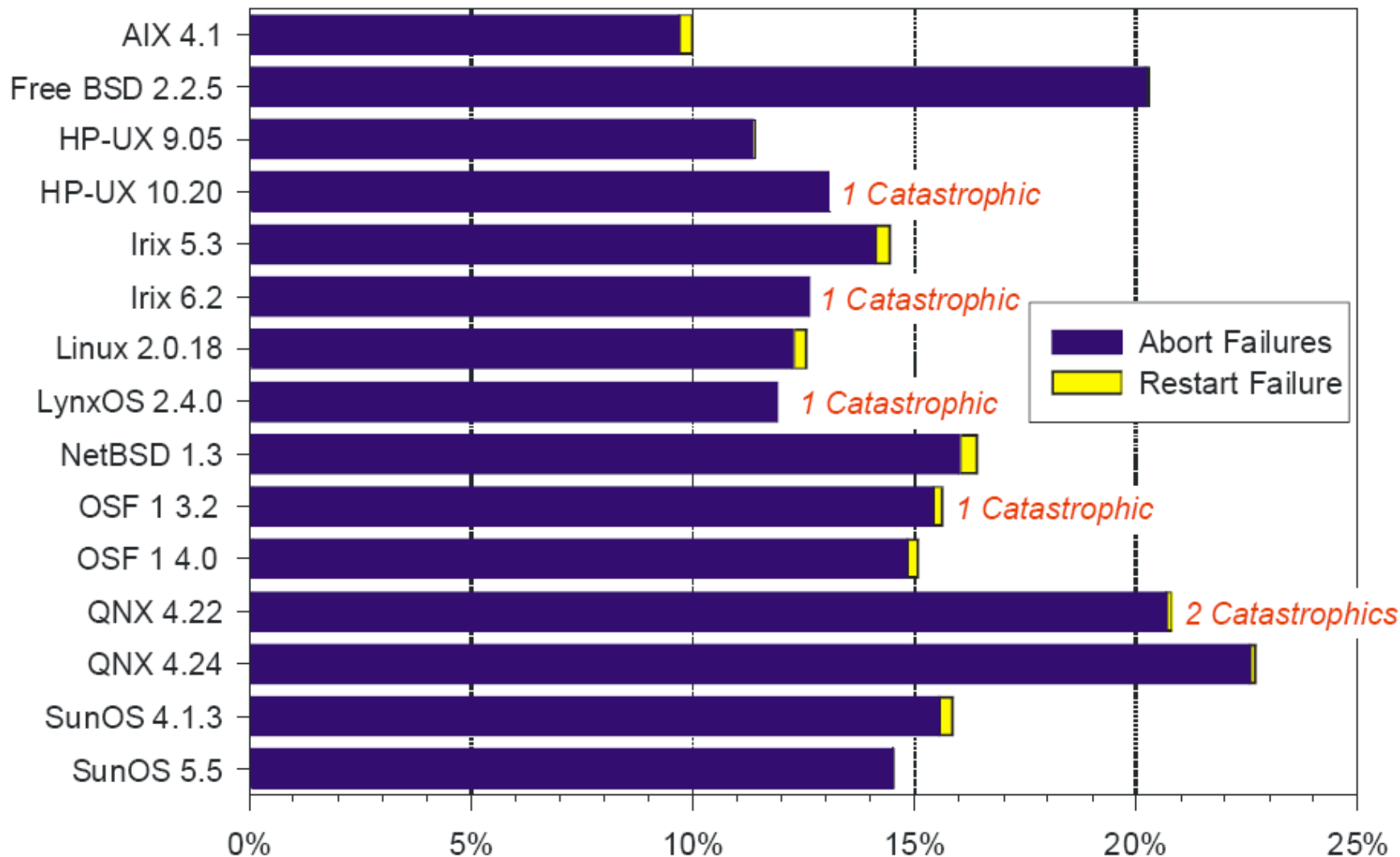
TEST  
VALUES

TEST CASE

Sfseek (Sfio\_t \*theFile=( Composite Value), int pos=0)

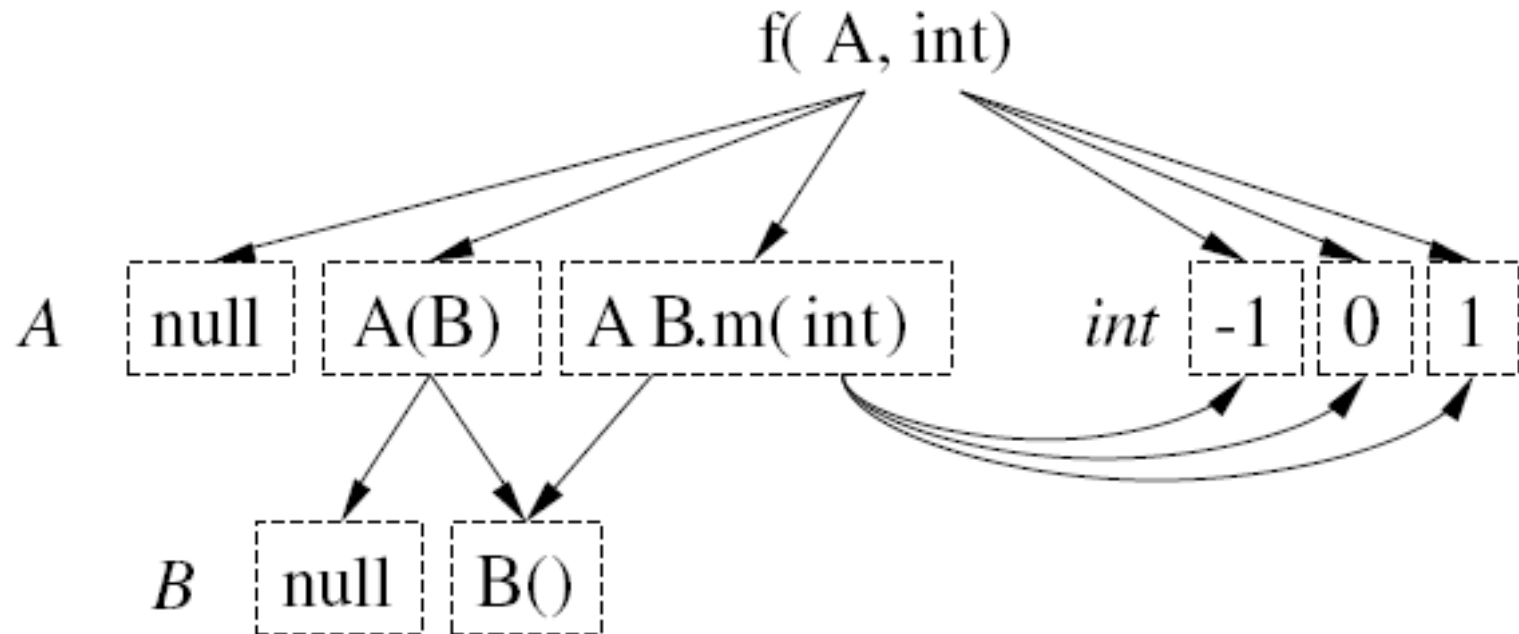
# Típus-specifikus tesztelés: Ballista – eredmények

Ballista Robustness Tests for 233 Posix Function Calls

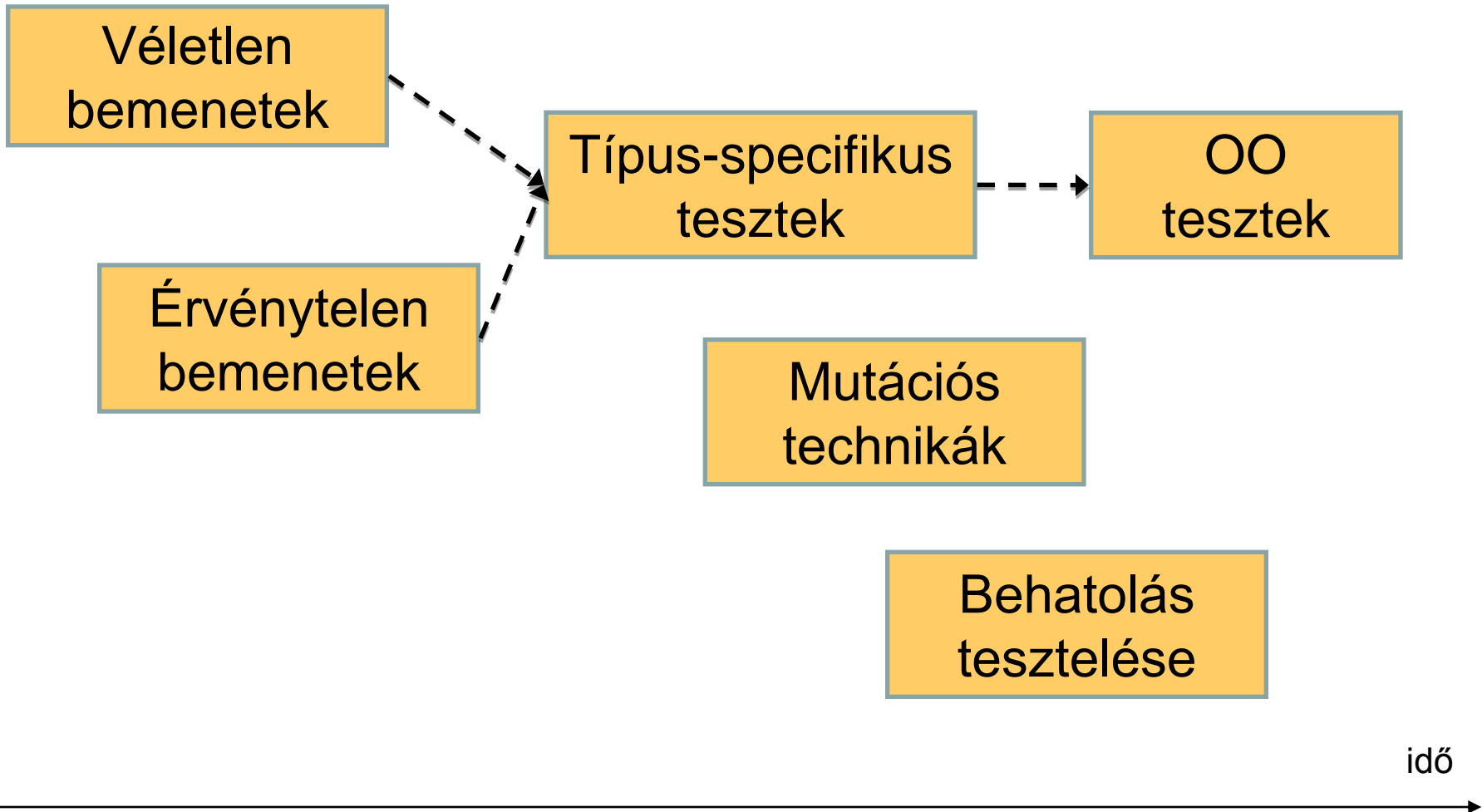


# OO programok tesztelése: JCrasher – paraméterek

- Hogyan generáljunk adott típusú (extrém) objektumot?



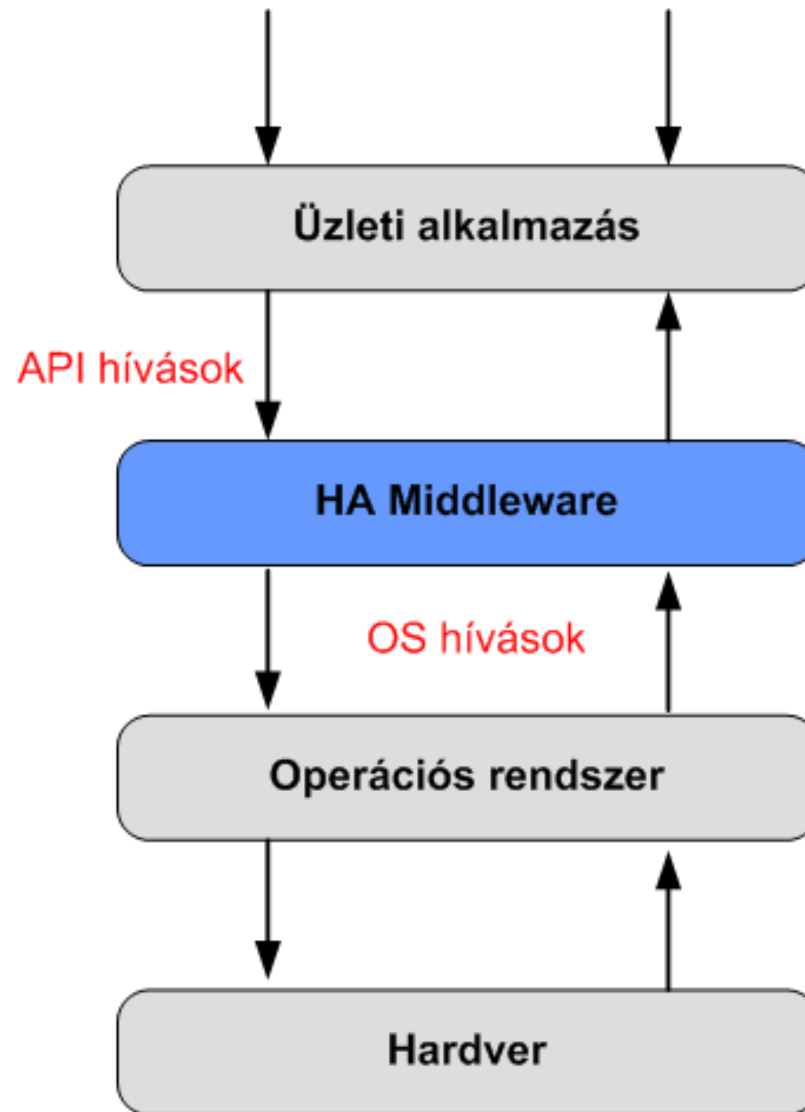
# A módszerek fejlődése



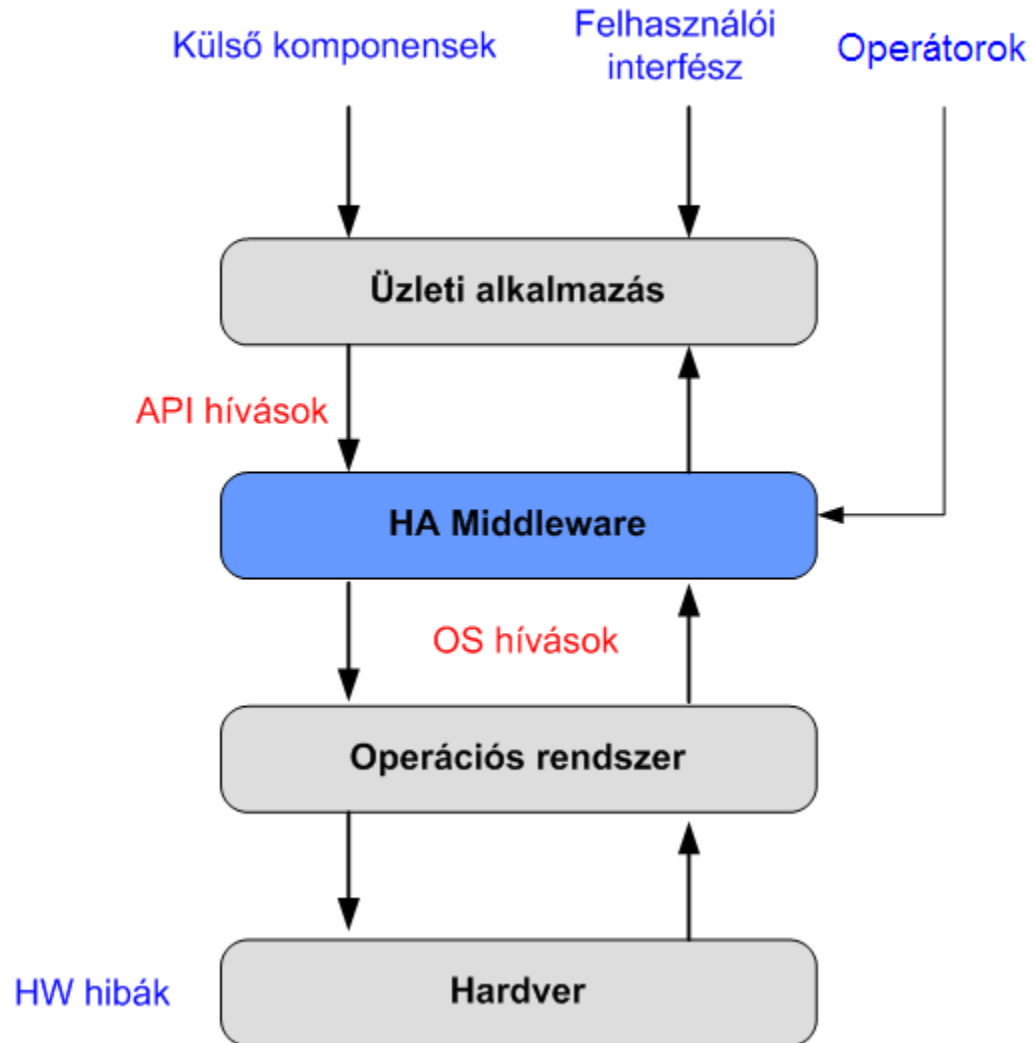
# Mintapélda: HA köztesréteg tesztelése

- HA köztesréteg: Nagy rendelkezésreállítás biztosítása
  - Komponens hibadetektálás, helyreállítás, újraindítás, failover
  - Szabványos operációs rendszerek felett
  - Elosztott alkalmazások
- Service Availability Forum: HA köztesréteg specifikáció
  - AIS: Redundancia menedzselési funkciók (AMF)
  - Szabványos interfészek (C nyelvű)
- Miért kritikus a robusztusság hiba?
  - Futtatott komponensek hibáira kell felkészülni
  - Egy hibás komponens a köztesréteg robusztusság hibájának aktiválásával az **egész rendszer** működését befolyásolhatja
- Robusztusság tesztelés
  - Közös interfész alapján összehasonlítható implementációk
  - Nagyszámú hibaforrás és hibamód → **automatikus** tesztelés

# Hibamodell: Elsődleges források

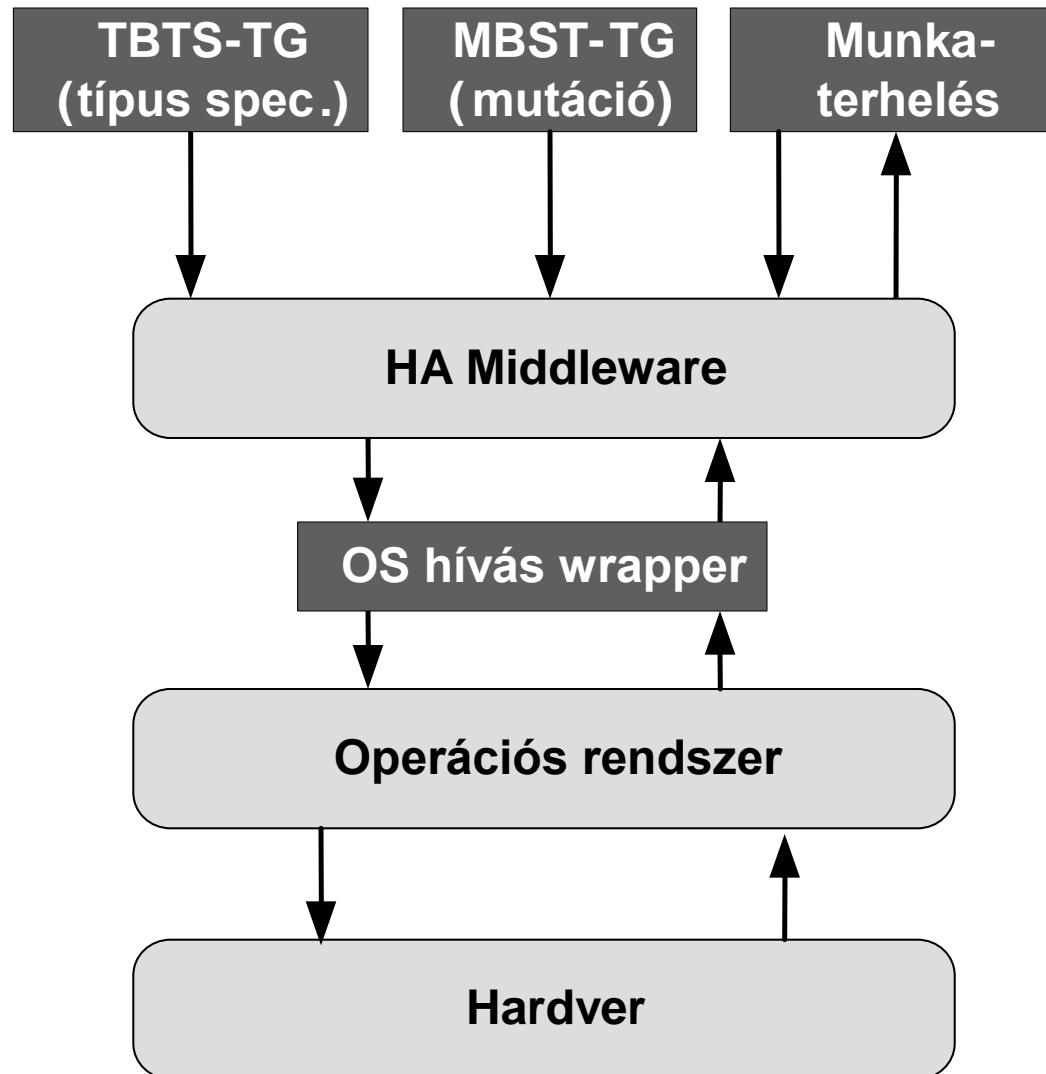


# Hibamodell: Másodlagos források

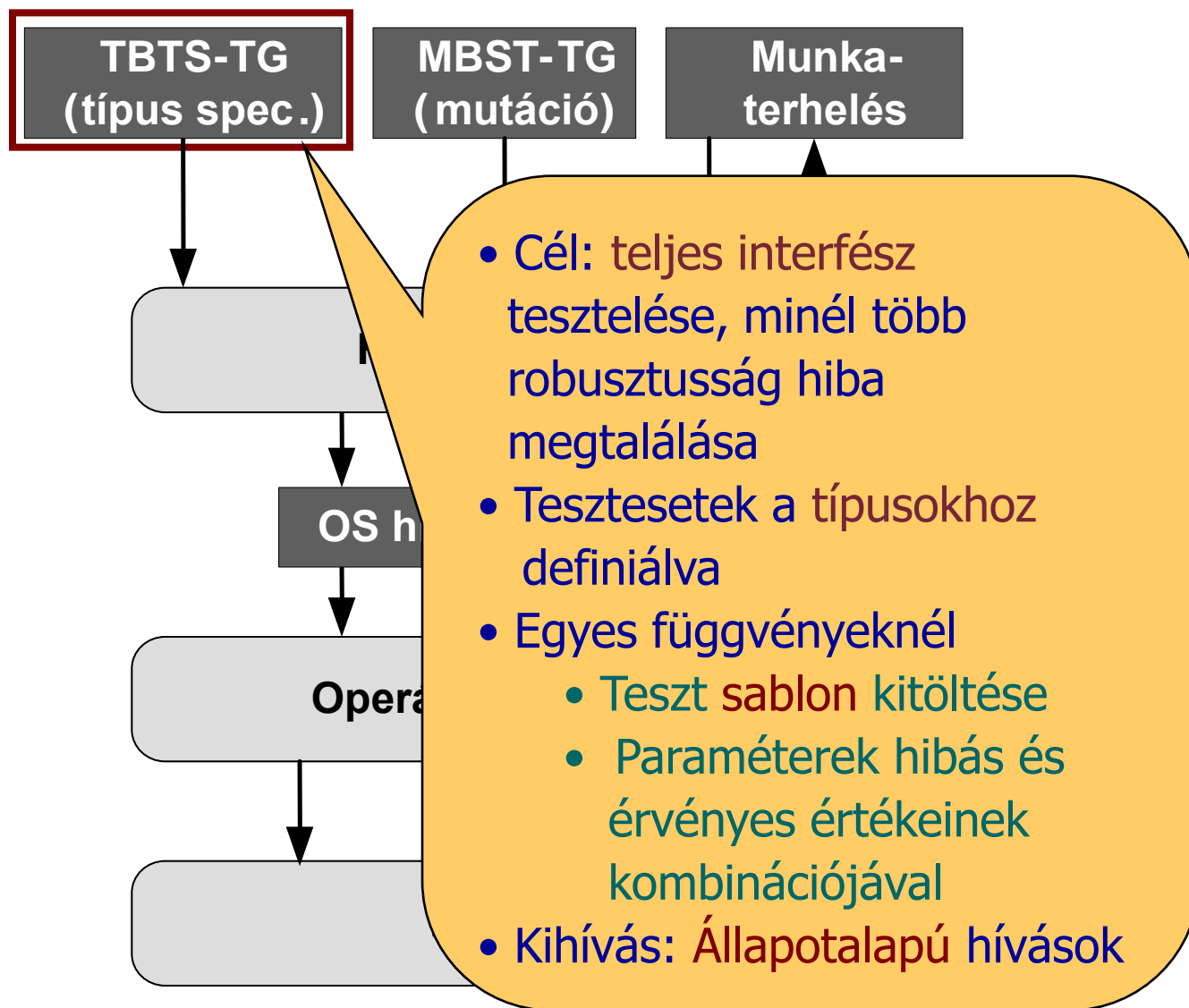




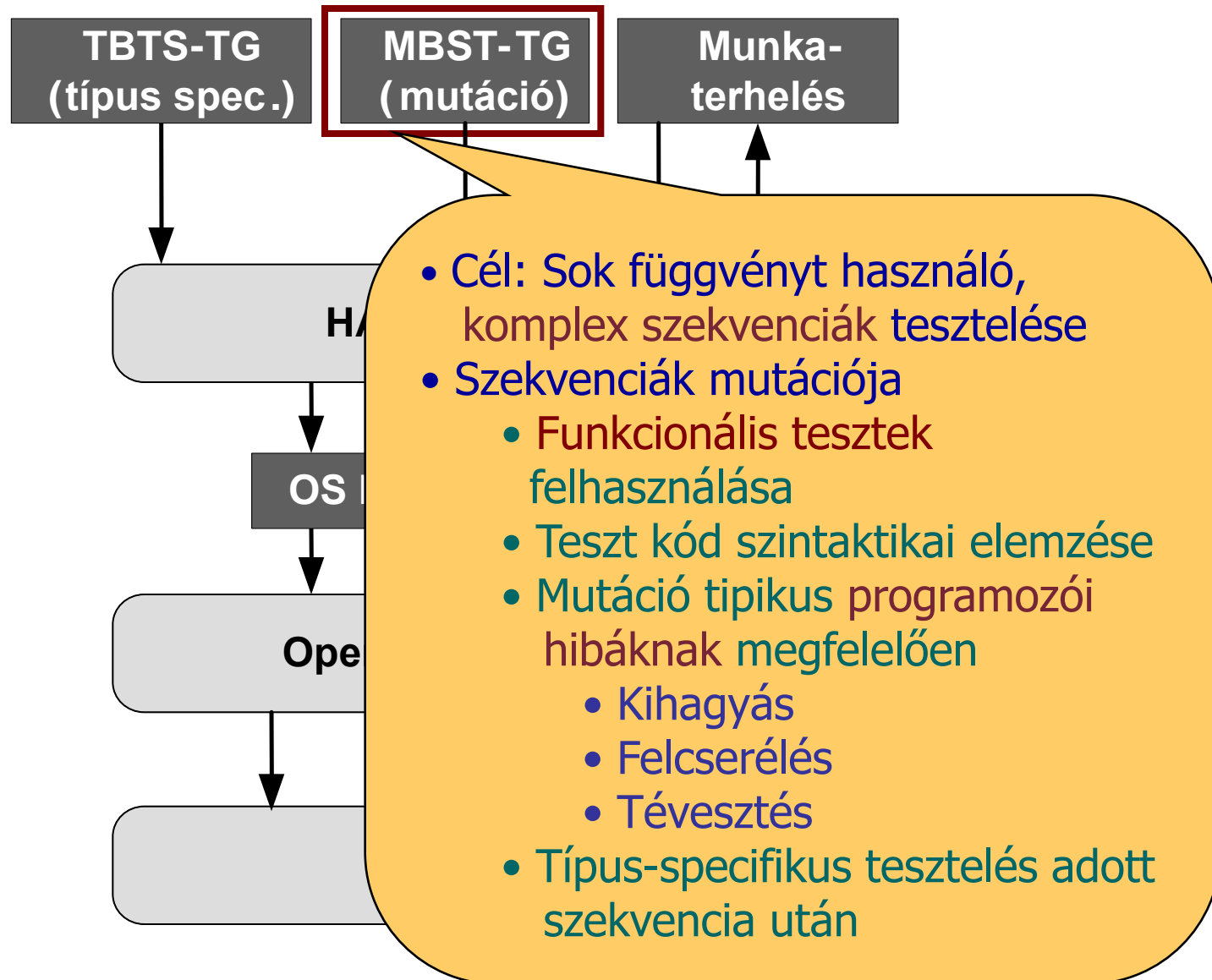
# Az elkészült teszteszközök



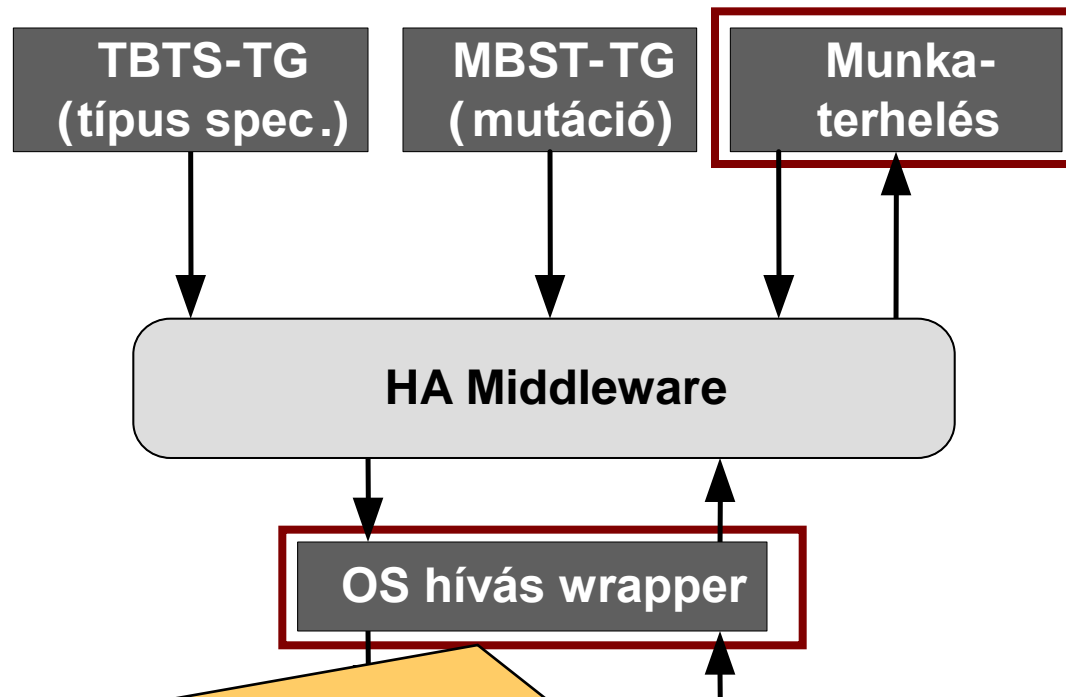
# Az elkészült teszteszközök



# Az elkészült teszteszközök



# Az elkészült teszteszközök



- Cél: Környezeti hatások tesztelése
- Munkaterhelés biztosítása
  - A köztesréteg adott rendszerhívásokat használjon
- Rendszerhívások „eltérítése”
  - Várakoztatás
  - Visszatérési érték megváltoztatása

# Eredmények áttekintése

SAFE4TRY  
robosztusabb

Típus specifikus	openais-0.80.1	openais-trunk	SAFE4TRY
hibátlan lefutás	24568	26019	29663
segmentation fault	1100	1468	0
timeout	467	2178	2
Mutáció			
Hibás / összes	30 / 92	28 / 92	1 / 92
OS wrapper			
Nem volt hiba	6	5	5
Alkalmazás hiba	0	2	1
Köztes réteg hiba	3	2	3

Rendszerhívások hibájára  
mindegyik érzékeny