# Speciális tesztelési feladatok

Majzik István
Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék
http://www.mit.bme.hu/

# Objektum-orientált rendszerek tesztelése

# 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…

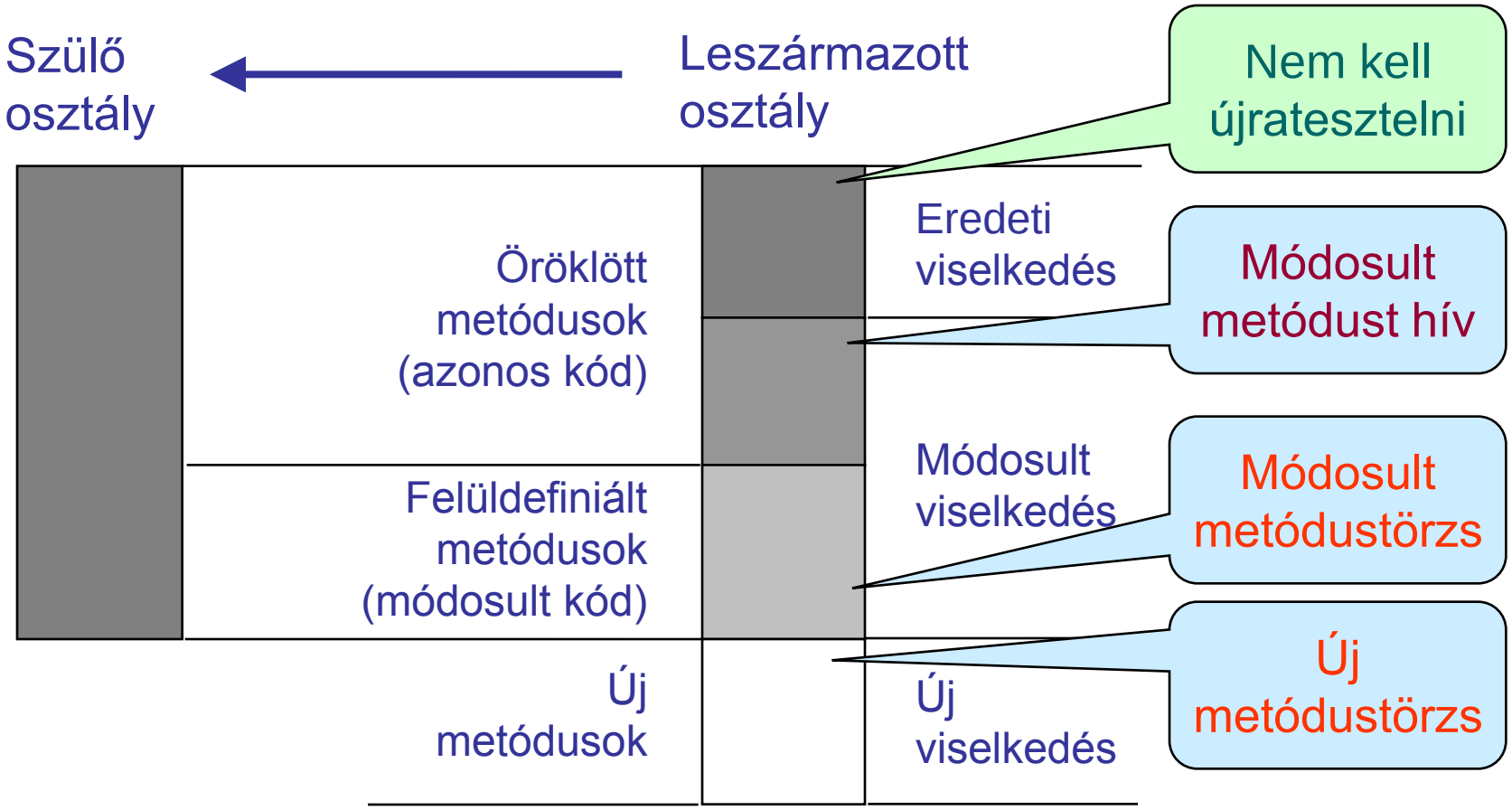
# Nem példányosítható osztályok

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

# Öröklés

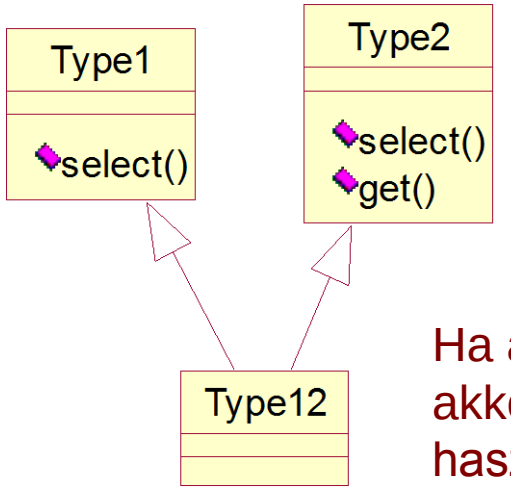
- Intuíció:
  - Öröklött metódusokat nem kell tesztelni (az ősben tesztelve volt)
- Általánosan nem alkalmazható megközelítés:
  - Felüldefiniálás
  - Más környezet, más hívott metódusok
- Öröklés típusai a viselkedés szempontjából:
  - Szigorú: Megtartja a szülő viselkedését + kiegészítheti
    - Az új metódusok megváltoztathatják az állapotot!
  - Szintaktikai: Interfész marad, viselkedés átdefiniálható
    - Átdefiniált metódusok újratesztelendők + 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 örökölt 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)

- Polimorf paraméterű metódus tesztelése:
  - Minden lehetséges kötési kombinációt tesztelni kell
  - Rögzített osztályhierarchia esetén működik
- Hasonló problémák:
  - Heterogén konténerek
  - Virtuális metódusok (C++): leszármazottakban implementálandó

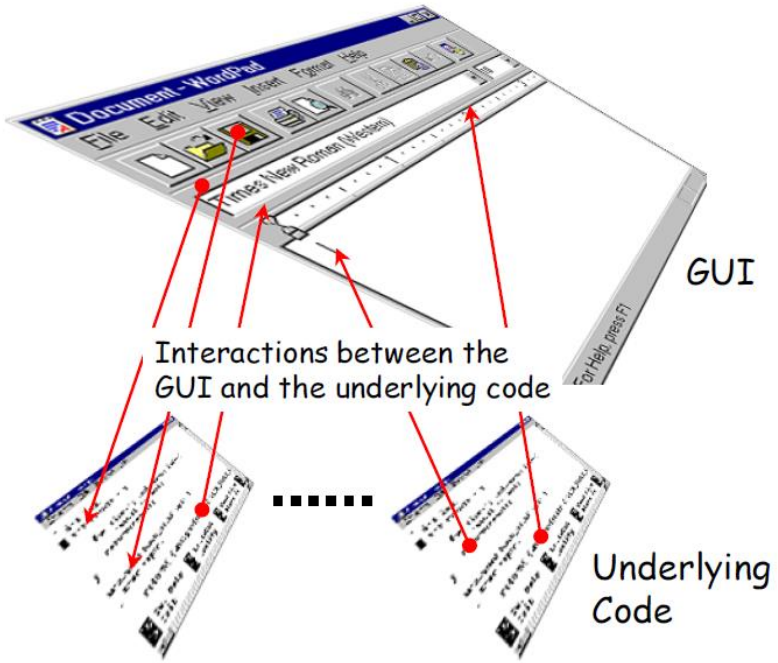
# OO teszt fedettségi mértékszámok

- 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 szálban külön tesztelendők

# Felhasználói felületek tesztelése

# GUI jellegzetességek

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



# 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

# Szisztematikus tesztelés előfeltételei

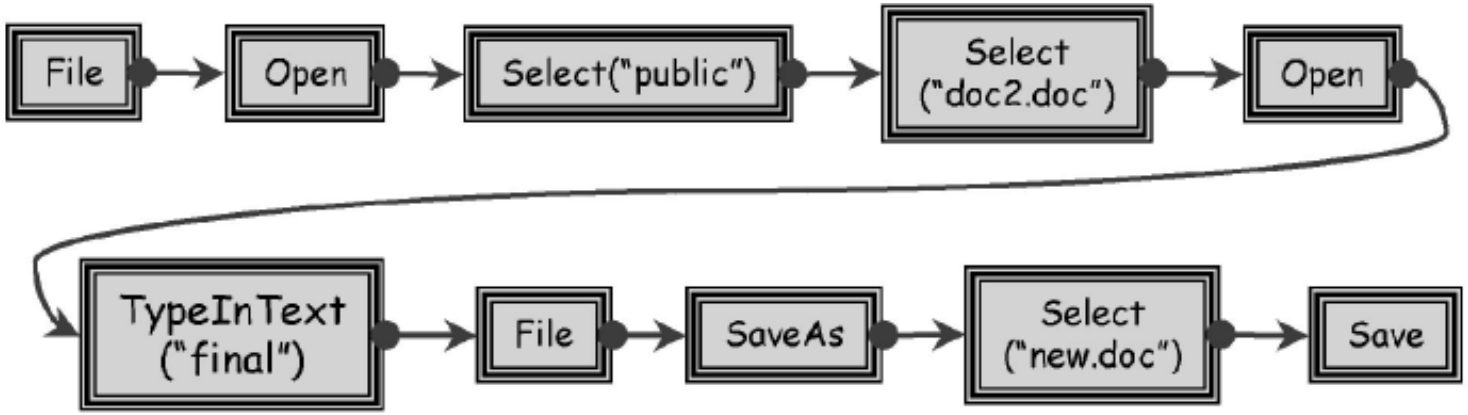
- GUI modell felvétele
  - Előnyök:
    - Teszt fedettség definiálható
    - Automatikus tesztgenerálásra is lehetőséget ad
  - Két tipikus GUI modell:
    - Operátor alapú GUI modell
    - Állapotgép alapú GUI modell
- Meghatározott teszt módszer rögzítése
  - Előnyök
    - GUI modellhez való illeszkedés biztosítható
    - Ad-hoc megoldásoknál jobban kézbentartható
  - 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

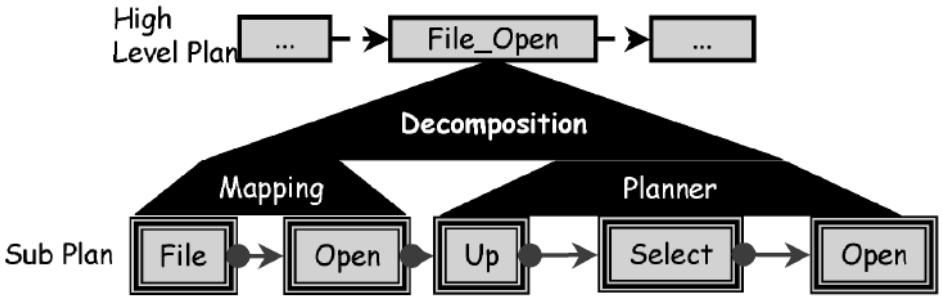
# Scenario alapú tesztelés

1. Teszt cél meghatározás
   - Operátorok felmérése
   - Objektumok felmérése
   - Jellegzetes használat (kiindulási állapot, célállapot) meghatározása
2. Operátor szekvencia konstruálása
   - Jellegzetes operátorsorozat (összetett operátorokkal)
3. Konkrét eseményszekvenciára való leképzés
   - Tesztesetek generálása
   - Összetett operátorok automatikusan is leképezhetők



# Eseményszekvenciára való leképzés tervkészítővel

- 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 (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

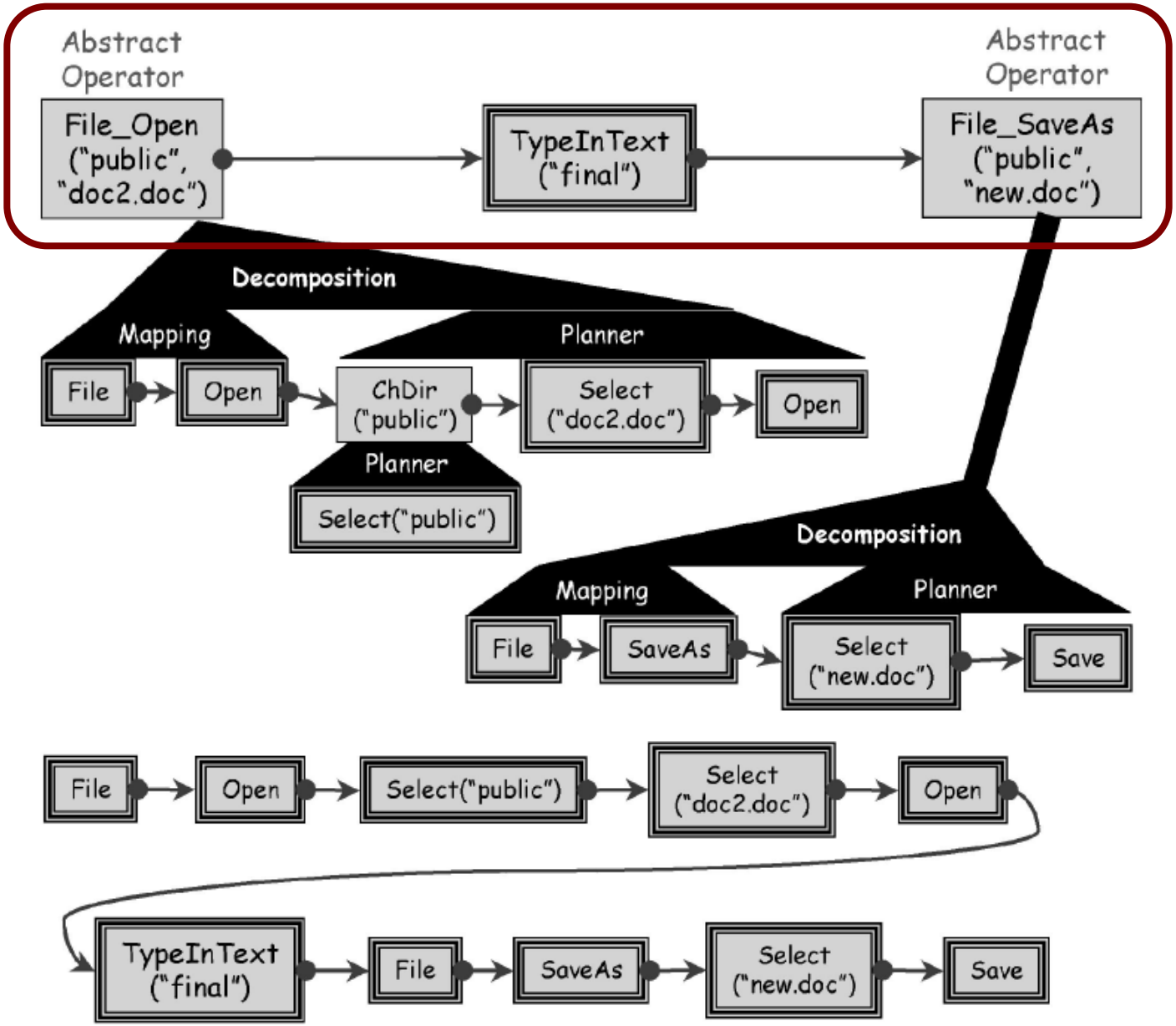


# Példa egy leképzésre

Kiindulás:

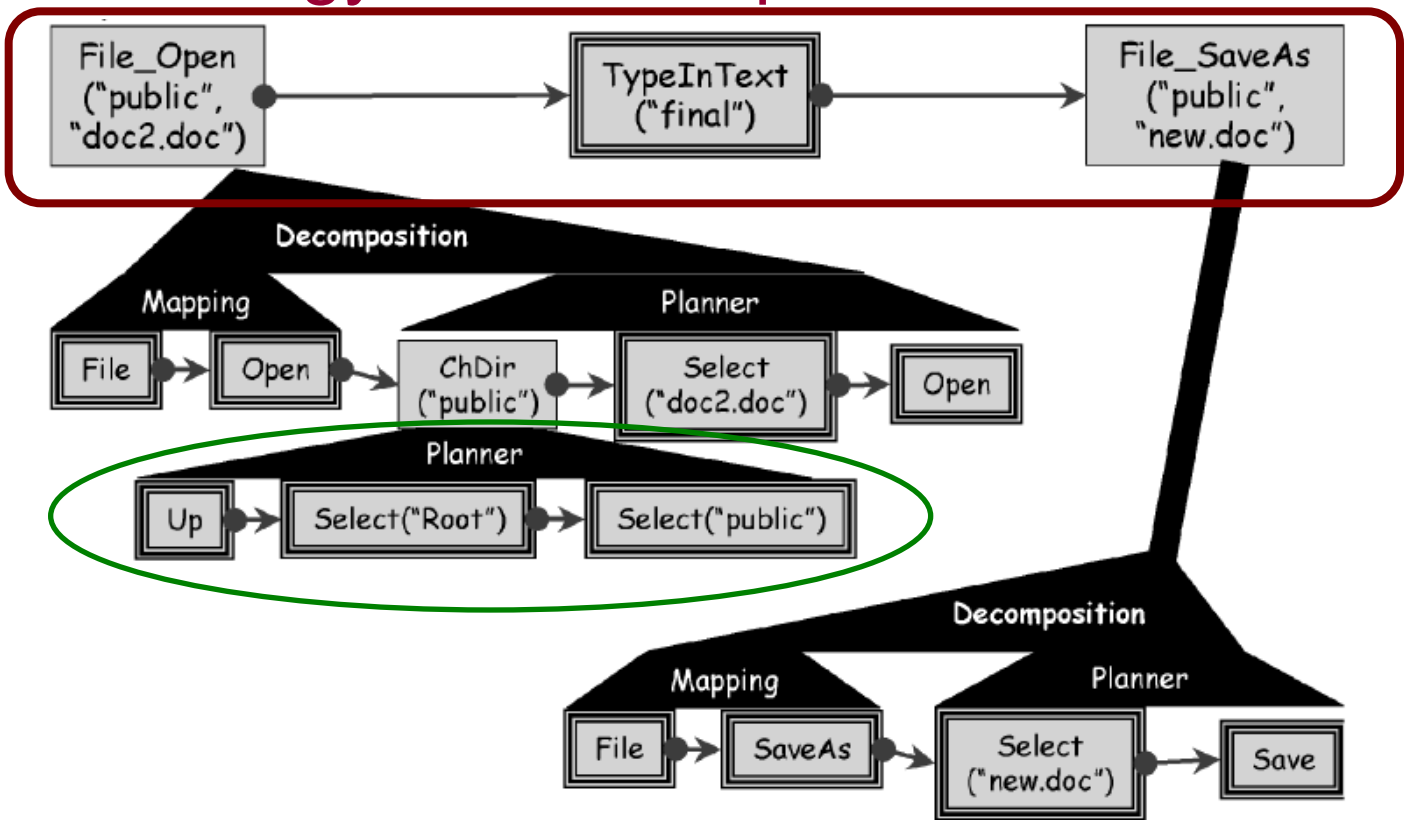
Leképzés:

Eredmény:

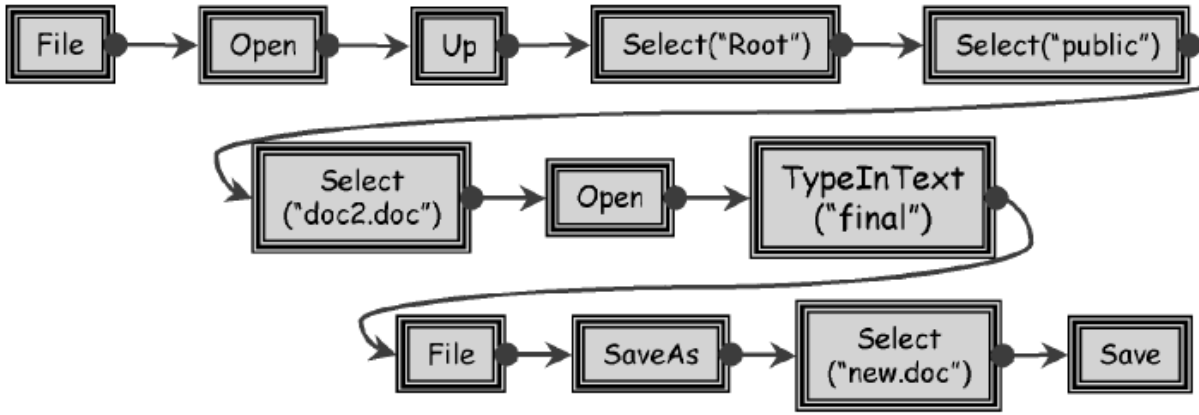


# Példa egy másik leképzésre

Leképzés:

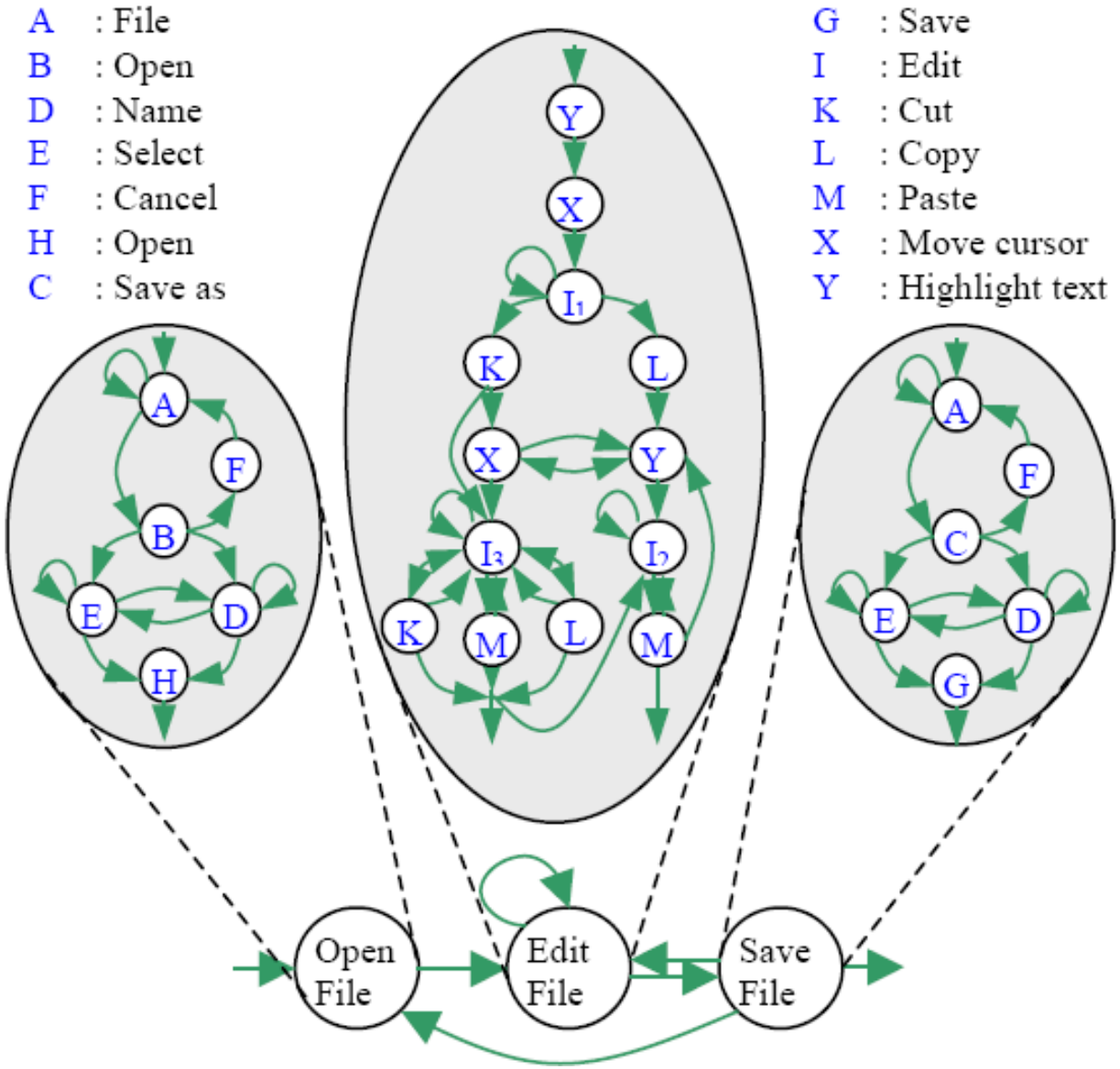



Eredmény:



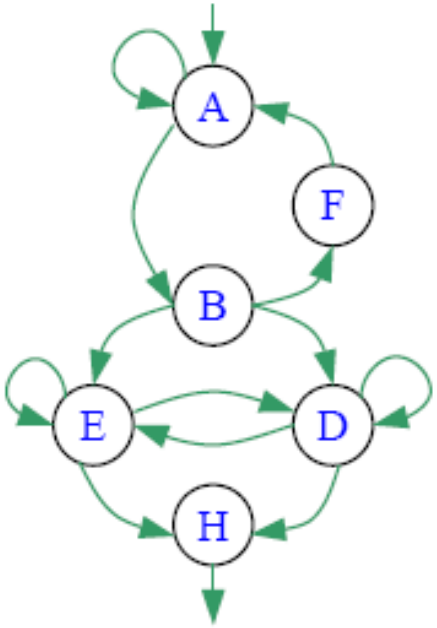
# II. Állapotgép alapú GUI modell

- GUI mint auto-mata
- Esemény folyam (elemi műve-letek)



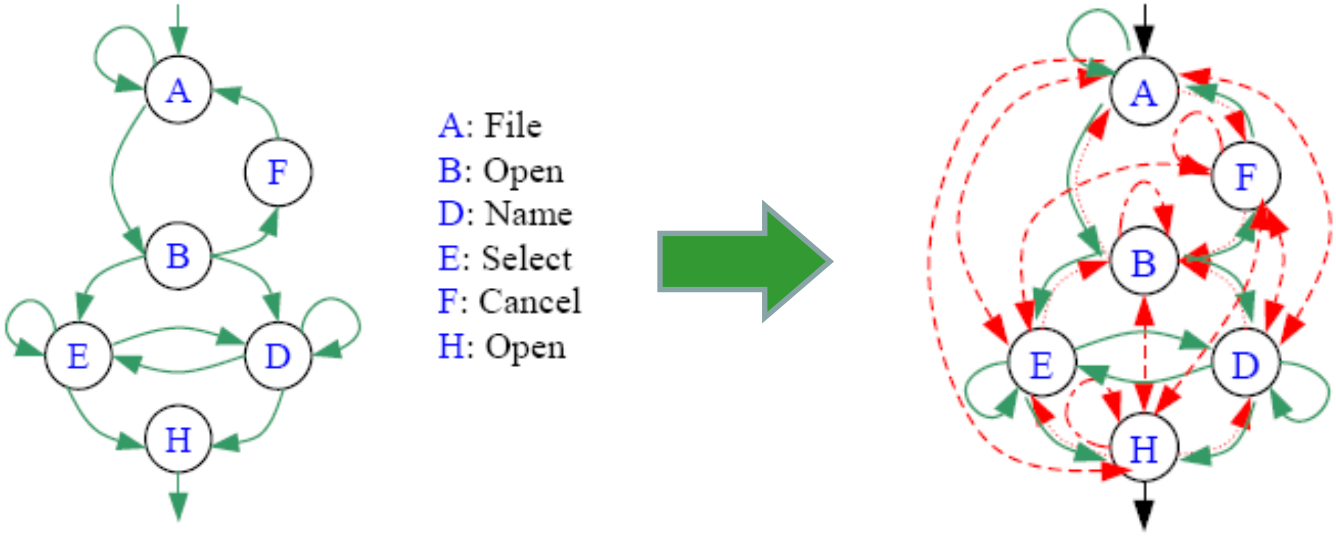
# A normál működés tesztelése

- 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



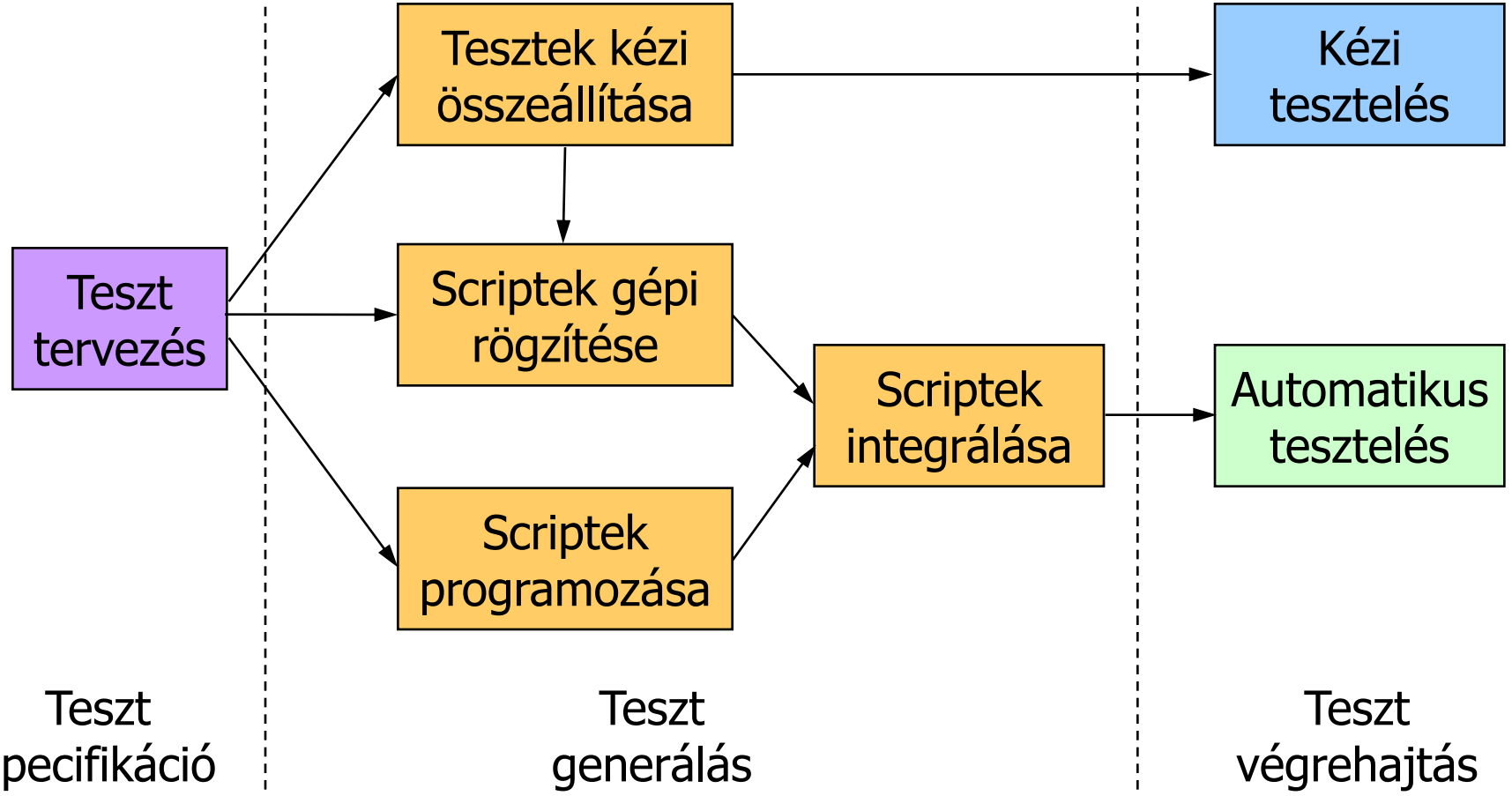

A: File
B: Open
D: Name
E: Select
F: Cancel
H: Open

# 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

# 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)

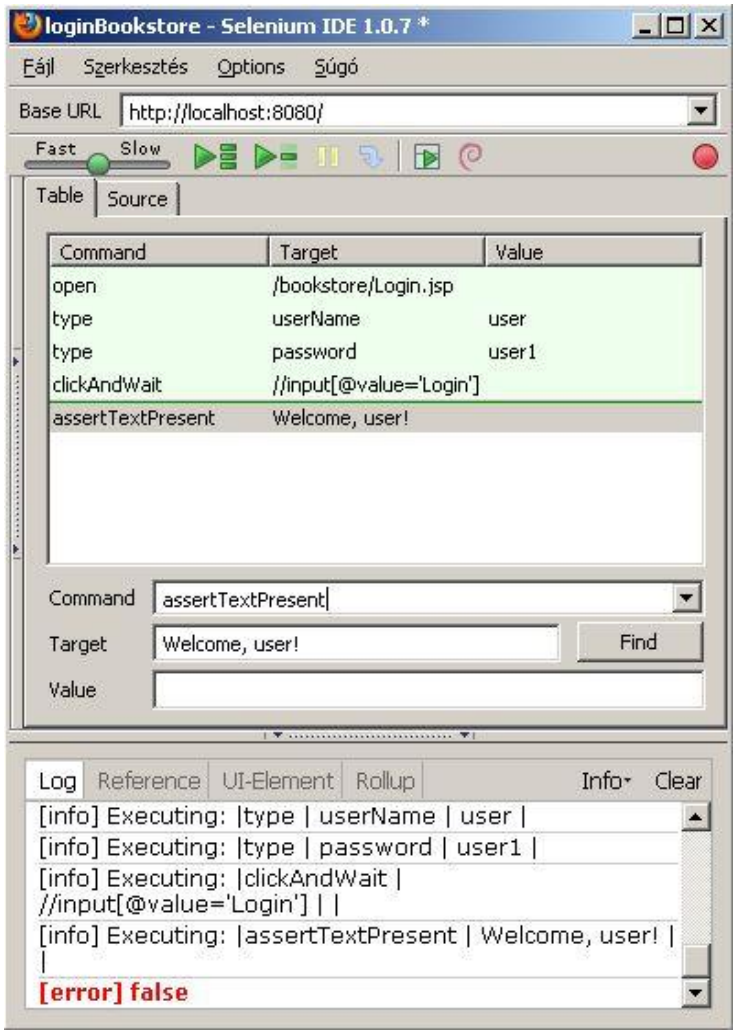
| Eszköz | Környezet | Licensz |
|---|---|---|
| Abbot | Java | CPL |
| Squish | Java + Web | Commercial (Eval) |
| SilkTest (Borland) | Multi | Commercial (Eval) |
| IBM RFT | Multi | Commercial (Eval) |
| BadBoy | Web | FFNP |
| NUnit Forms | .Net | BSD |
| QuickTest (HP) | Java + Windows | Commercial (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



# Hibakezelés tesztelése hibainjektálással

# 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)

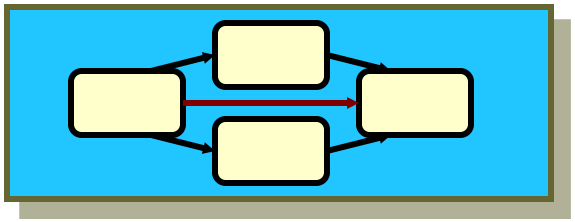
# Hibainjektálás: Módszerek

- 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 valósághű
    - 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

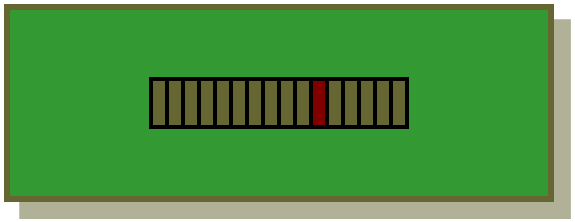
# Hibainjektáló eszközök

- 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

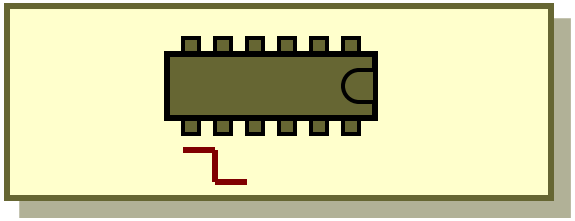
# A hibainjektálás valósághűsége



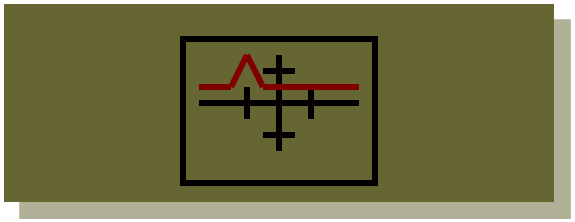
Hibaszimuláció a forráskódban vagy modell szinten a fejlesztőeszközben



Regisztertartalom módosítása, memóriakép felülírása



Síneken haladó vagy áramkörök lábán megjelenő jelek módosítása



Radioaktív sugárzás, ioninjektálás, tápfeszültség zavarása, hőmérséklet

Ár, valósághűség

# Egy más célú technika: Hibabeültetés

- Hibák direkt bevitele, majd tesztelés indítása
- Becslés:

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

- Tesztkészlet minősége – tesztelt alkalmazás minősége
- Sok feltétel teljesülése esetén jó becslő csak!
  - Beültetett hibák reprezentatívak, eloszlásuk megfelel a valódi hibákénak …

# 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

# 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 értékként
  - Létrehozáshoz extrém paraméterek megadása
- Scenario mutációval generált bemenetek
  - Az extrém bemenetek állapottól függően adhatók ki
  - Sorrendi, kihagyási, időzítési hibák is definiálhatók

# Munkaterhelés (workload) a tesztelés során

- Valódi terhelés
  - Pl. regisztrált scenariok visszajátszása
- Realisztikus terhelés
  - Jellegzetes scenariok 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
  - **C**atastrophic: A teljes rendszer összeomlik / újraindul
  - **R**estart: Az adott alkalmazás újraindulása
  - **A**bort: Az adott alkalmazás leáll
  - **S**ilent: Hibajelzés nélküli érvénytelen művelet
  - **H**indering: Érvénytelen hibakód
- Nem robusztusság hiba:
  - Érvényes hibakód visszaadása

# Jellegzetes eszközök

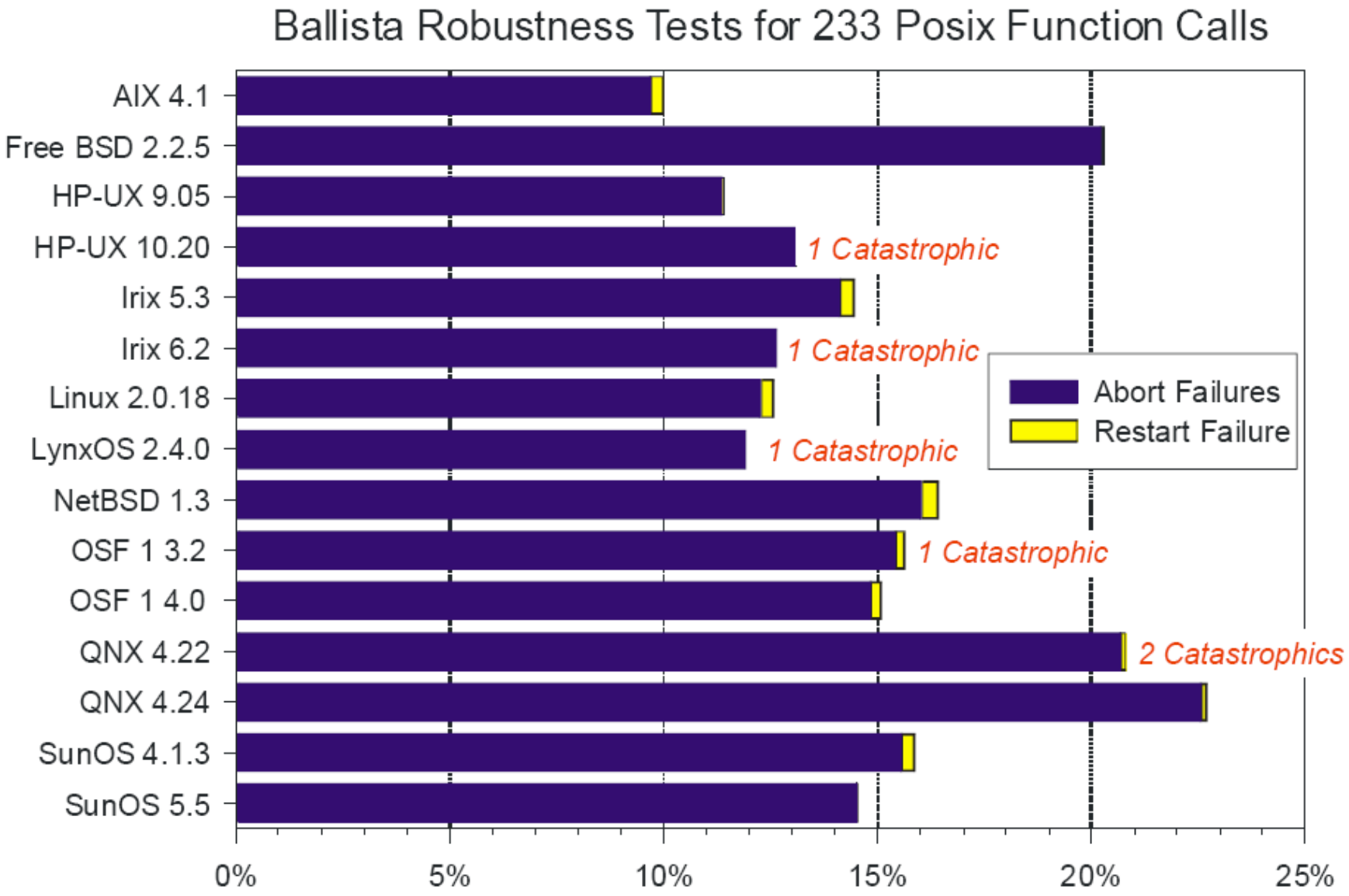
- Hardver hibainjektáló eszközök
  - Közvetett robusztusság tesztelés: Környezet komponenseibe injektált hibák hatása vizsgálható a tesztelt komponensre
  - Belső hibák injektálása nem robusztusság tesztelés!
- Kombinatorikus robusztusság tesztelő eszközök:
  - Fuzz*:* Véletlenszerű bemenetek konzolos 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 – bemenetek

- Szélső és extrém értékek beállítása

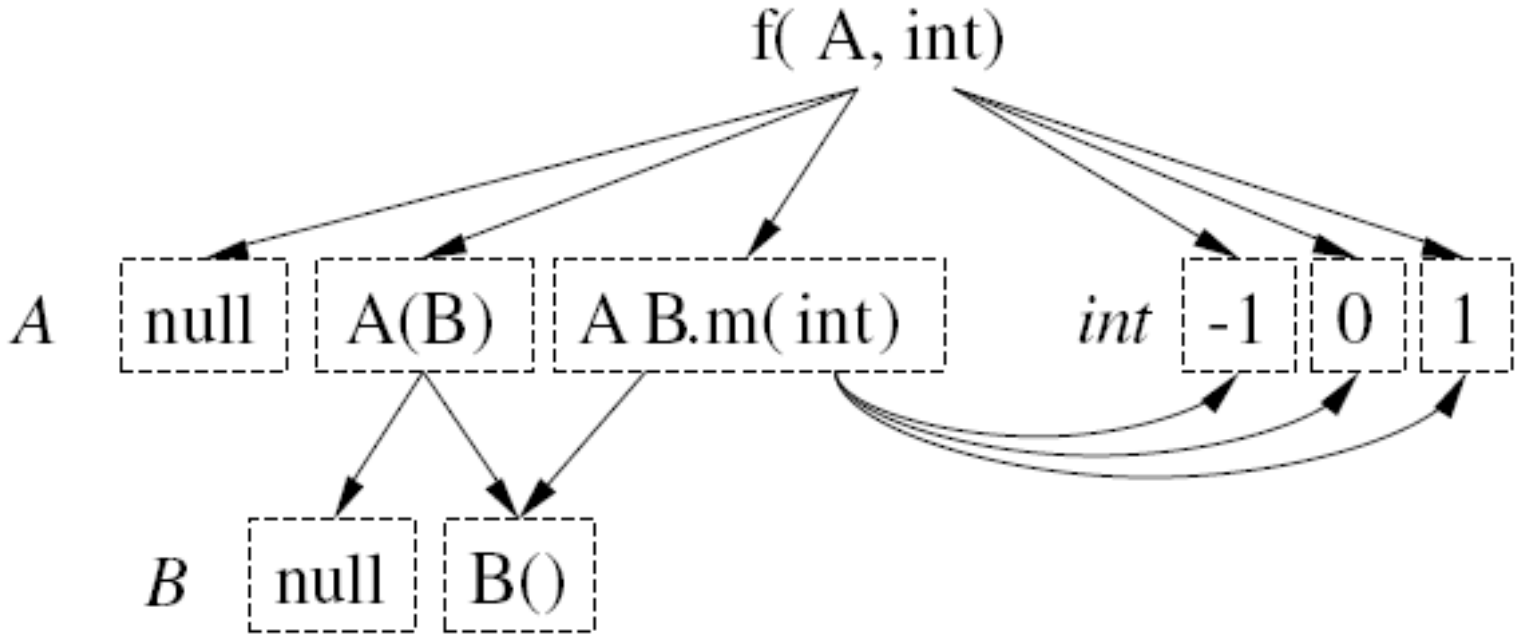
| Data type | Substitution values | | | | | |
|---|---|---|---|---|---|---|
| Pvoid | NULL | 0xFFFFFFFF | 1 | 0xFFFF | -1 | Random |
| Integer | 0 | 1 | MAX INT | MIN INT | 0.5 | |
| Boolean | 0 | 0xFF (Max) | 1 | -1 | 0.5 | |
| String | Empty | Large (> 200) | Far (+ 1000) | | | |

# Típus-specifikus tesztelés: Ballista – eredmények



# OO programok tesztelése: JCrasher – paraméterek

- Hogyan generáljunk adott típusú (extrém) objektumot?



# A módszerek fejlődése