Szoftvertechnológia

2018 Március 21

6. előadás

Szoftverprototípus készítése

KADA ZSOLT
INFORMATIKAI ÜGYVEZETŐ IGAZGATÓ
GIRO ZRT.

PÁZMÁNY PÉTER KATOLIKUS EGYETEM - SZÉKELY ISTVÁN JOGTUDOMÁNYI INTÉZETE
INFORMÁCIÓS TECHNOLÓGIAI ÉS BIONIKAI KAR - KUTATÓ KAR

Tartalom

A prototípuskészítés szerepe
Prototípus készítése

- A **prototípuskészítés** általában a követelménytervezés része, a követelmények feltárásának és validációjának eszköze.
- A **prototípus** a szoftverrendszer kezdeti verziója, amely alkalmas a rendszer vagy egyes funkciók koncepciójának bemutatására és kipróbálására.
- Korábban úgy tekintették, hogy a prototípus alacsonyabb rendű a kívánt rendszernél.
- Ma a prototípus- és a normál rendszer közti határ fokozatosan elmosódik és sok rendszert az evolúciós modell alapján készítenek.

A prototípuskészítés szerepe
A prototípusok alkalmazása

- A felhasználó nem látja előre, hogyan fogja használni az új rendszert.
- A prototípus elsődleges célja az, hogy segítse a felhasználókat a rendszerkövetelmények megértésében:
  - **A követelmények feltárása:** a prototípussal a felhasználók megtapasztalhatják, hogyan fogja a rendszer a munkájukat támogatni.
  - **A követelmények validálása**: a prototípus felfedheti a félreértéseket, hibákat és hiányosságokat a követelményekben.
- A prototípus csökkenti a követelményekkel kapcsolatos kockázatokat.

A prototípuskészítés szerepe
A prototípuskészítés előnyei

- Segít felismerni a szoftver felhasználója és készítője közti félreértéseket. → A rendszer jobban **illeszkedik a felhasználó igényeihez.**
- Kiderülhet, hogy hiányzik valamely szolgáltatás vagy ellentmondások vannak a szolgáltatások között. → **A rendszer használhatóbb lesz.**
- A szoftverfolyamat elején már egy – legalábbis részben, modellként – működő rendszer áll rendelkezésre. → Gyorsabban **elkészül a rendszer.**
- A prototípus felhasználható a rendszerspecifikáció alapjaként. → **Javul a tervezés minősége.**
- Támogathatja a felhasználók képzését és a rendszertesztet is.
- A fejlesztéshez kevesebb erőforrásra van szükség. → **Költségcsökkentés.**

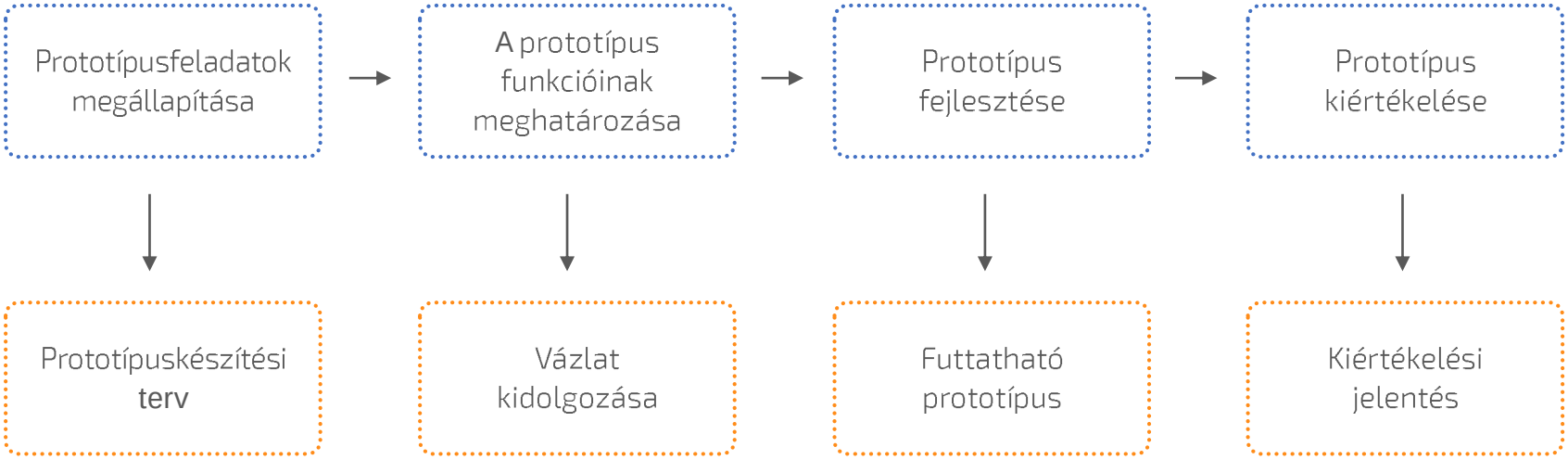
A prototípuskészítés szerepe
A prototípuskészítés veszélyei

- Eldobható prototípust végleges rendszerként használnak

  (Sérülhet mind a teljesítmény, mind a funkcionalitás, mind a megbízhatóság)

- A gyors fejlesztésből és az iterációból fakadó hibák:
  - Lassabb válaszidő
  - Bonyolultabb rendszerstruktúra

A prototípuskészítés szerepe
A prototípuskészítés folyamata



Tartalom

A prototípuskészítés módjai
A prototípuskészítés helye a szoftverfolyamatban

- **Evolúciós prototípus készítése:**
  - Célja egy működő rendszer átadása a megrendelőnek *(esetleg korlátozott funkcionalitással)*.
  - A legfontosabb követelmények implementálásával egyszerű rendszer készül, amelyet újabb követelmények feltárásával fokozatosan egészítenek ki új funkciókkal.
  - Az ***Agilis fejlesztés*** alapvető módszere.
  - Weblapfejlesztésben és e-business alkalmazásokban is jól használható.
- **Eldobható prototípus készítése:**
  - Célja a rendszerkövetelmények feltárása és validálása.
  - A nem teljesen megértett követelmények megvalósítása és bemutatása segíti a feltárást. A követelményspecifikáció elkészülte után nem használható fel.

Evolúciós prototípus készítése

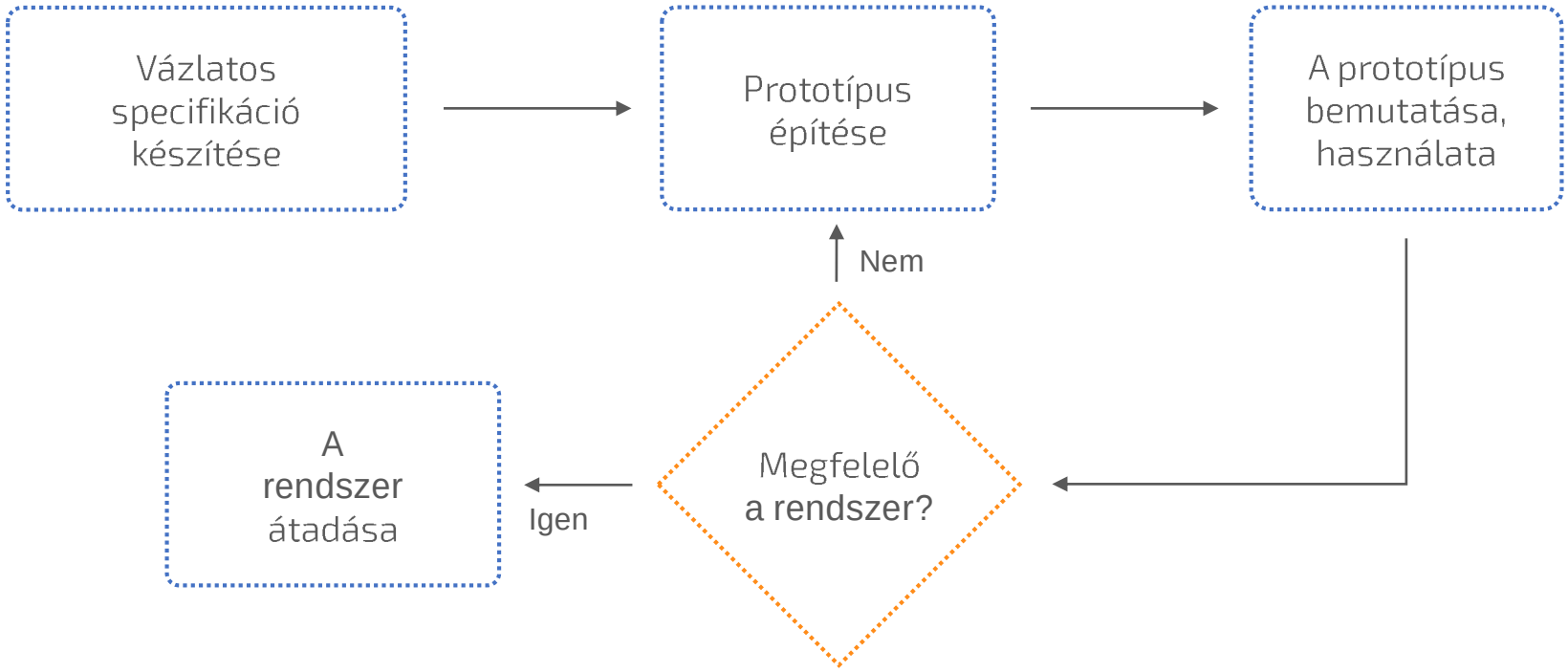
PÁZMÁNY PÉTER KATOLIKUS EGYETEM - SZÉCHY TERÉZVÁROSI INTÉZMÉNY
INFORMÁCIÓS TECHNOLÓGIAI ÉS BIONIKAI KAR - KUTATÓ KAR

Evolúciós prototípus készítése

- Olyan rendszereknél célszerű alkalmazni, ahol nem készíthető el előre a végleges specifikáció. Ilyenek általában az intenzív felhasználói interfész-használatot igénylő rendszerek.
- Nincs részletes rendszerspecifikáció, sokszor a részletes követelménydokumentum is hiányzik.
- A felhasználó már a rendszer fejlesztése közben jelezheti, hogy milyen irányban kívánja folytatni a fejlesztést.
- A fejlesztéshez gyors, iterálható fejlesztői eszközökre és módszerekre van szükség.
- Mivel nem készül követelményspecifikáció, a validáció is csak a rendszer *(vagyis a prototípus)* bemutatásával történhet.

Evolúciós prototípus készítése

Az evolúciós prototípuskészítés folyamata



Evolúciós prototípus készítése
Az evolúciós prototípuskészítés jellemzői

- A specifikáció, a tervezés és az implementáció részben átlapolható.
- A rendszer **inkrementumok sorozataként** fejlődik *(ld. **Agilis módszerek**)*, és kerül a felhasználóhoz, vagyis a felhasználó kulcsfigurái minden inkrementum tervezésében és értékelésében részt vesznek.
- Gyors fejlesztői eszközök és technikák alkalmazhatók *(CASE eszközök, 4GL, folyamatmodellező nyelvek: BPMN-Business Process Modeling Notation, WebServices).*
- A felhasználói felületek GUI fejlesztői eszközökkel készíthetők.

Evolúciós prototípus készítése

Az evolúciós prototípuskészítés előnyei

Felgyorsul a rendszerfejlesztés

A gyors fejlesztés, az új rendszer sürgős használatbavétele gyakran fontosabb, mint a követelmények részletes, következetes feltárása vagy a hosszú távú karbantarthatóság.

Növelhető a felhasználó elkötelezettsége

A felhasználók bevonása a rendszerfolyamatba azt eredményezi, hogy a rendszer nagyobb valószínűséggel felel meg az elvárásoknak és a használatba vételkor a felhasználók már ismerik azt és tudják alkalmazni.

Evolúciós prototípus készítése

Az evolúciós prototípuskészítés hátrányai

Vezetési problémák

A hagyományos vezetési módszerek a vízesés modellre alkalmazhatók. Az új technológiák alkalmazásához speciális ismeretekre, esetleg más munkatársakra van szükség.

Karbantartási problémák

A folytonos változások a prototípus szerkezetének sérülését okozhatják, a dokumentáció hiánya és a speciális fejlesztő eszközök a karbantartást veszélyeztetik.

Szerződéskötési problémák

A fix áras szerződéshez előre ismerni kell a rendszer vázlatos követelményeit és tervét.
A ráfordításalapú szerződést pedig a megrendelő általában nem fogadja el.

Evolúciós prototípus készítése
A prototípus mint specifikáció

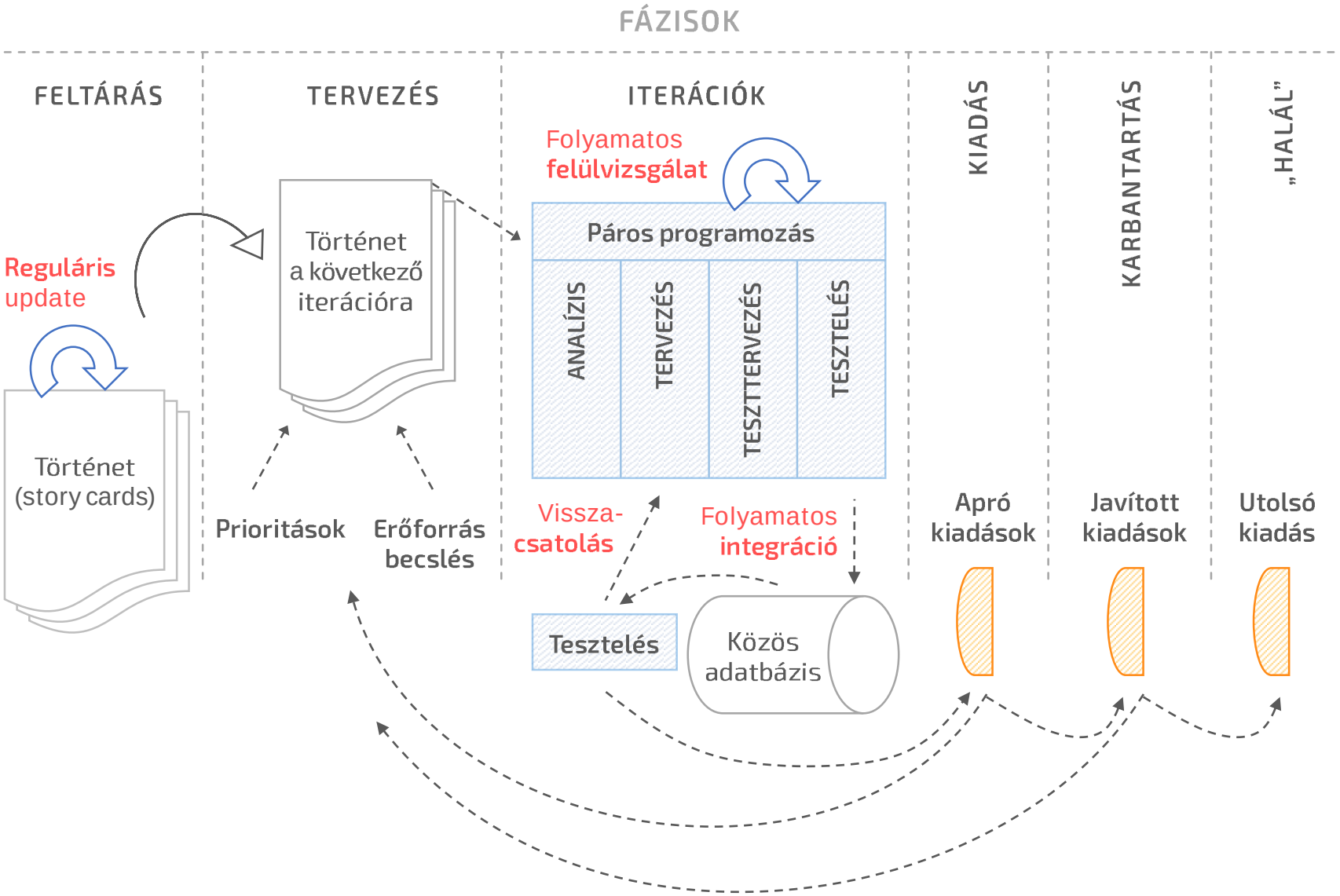
- Egyes követelményeket *(mint pl. a biztonságkritikus funkciókat)* nem lehet a prototípusba beépíteni, így nem fognak szerepelni a specifikációban.
- Egy implementáció nem lehet egy szerződés jogi melléklete, ráadásul menet közben folyton változik.
- A nem-funkcionális követelmények nem tesztelhetők teljes mértékben.

Evolúciós prototípus készítése
Inkrementális fejlesztés

- Az átfogó architektúra és rendszerkörnyezet megteremtése után a rendszer inkrementumokban készül és jut el a felhasználókhoz.
- Minden inkrementumhoz készülhet specifikáció és dokumentáció.
- A leszállított inkrementumokat a felhasználók tanulmányozhatják, így azok prototípusként használhatók.
- Az inkrementális fejlesztés tartalmazza a prototípuskészítés sok előnyét, miközben a folyamat jobban vezethető és a rendszer struktúrája is kézbentartható.
- Az agilis módszerek *(agile methods)* alapja.

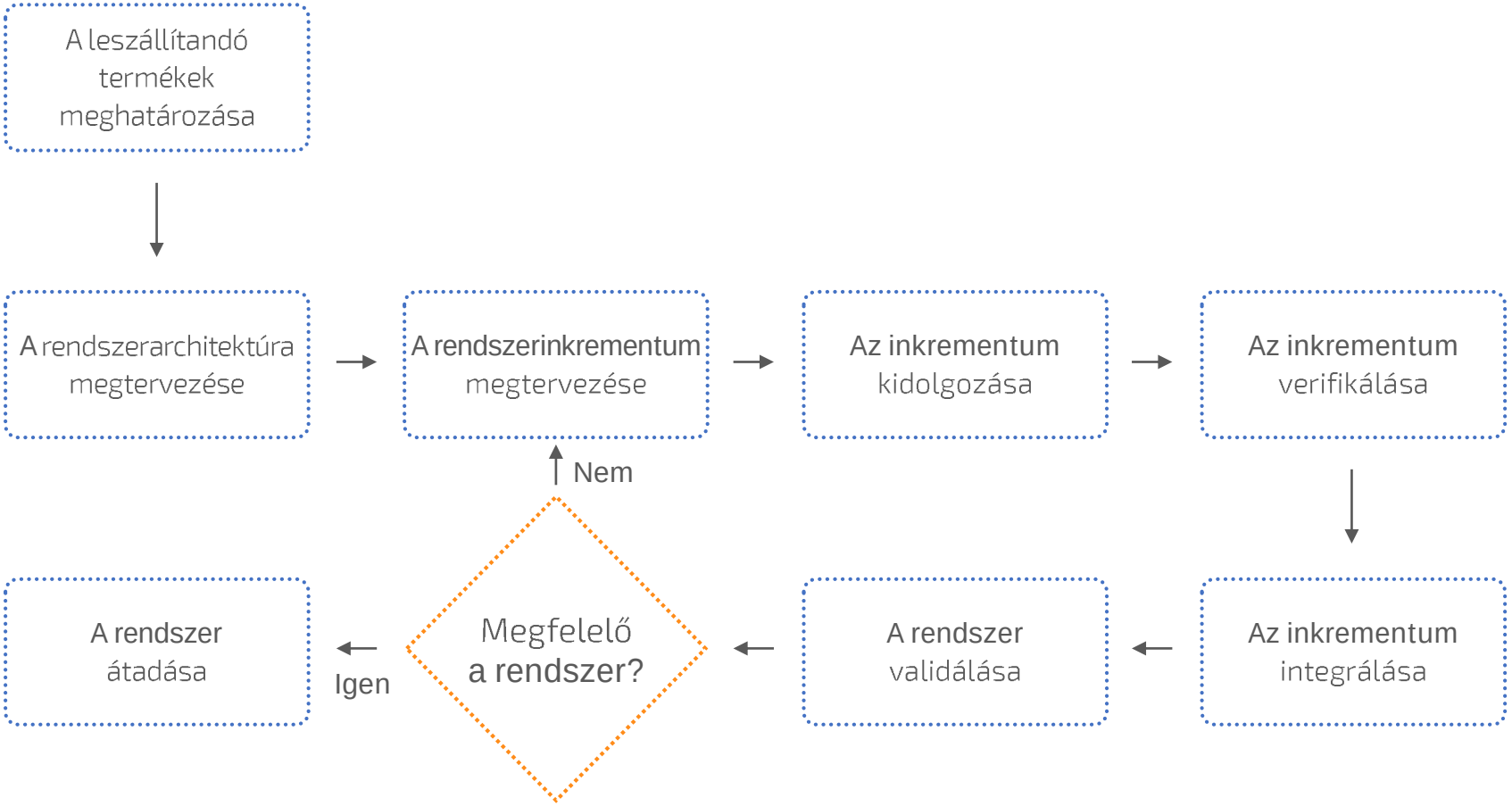
Evolúciós prototípus készítése

Példa: extrém programozás (XP)



Evolúciós prototípus készítése

Az inkrementális fejlesztési folyamat



Eldobható prototípus készítése

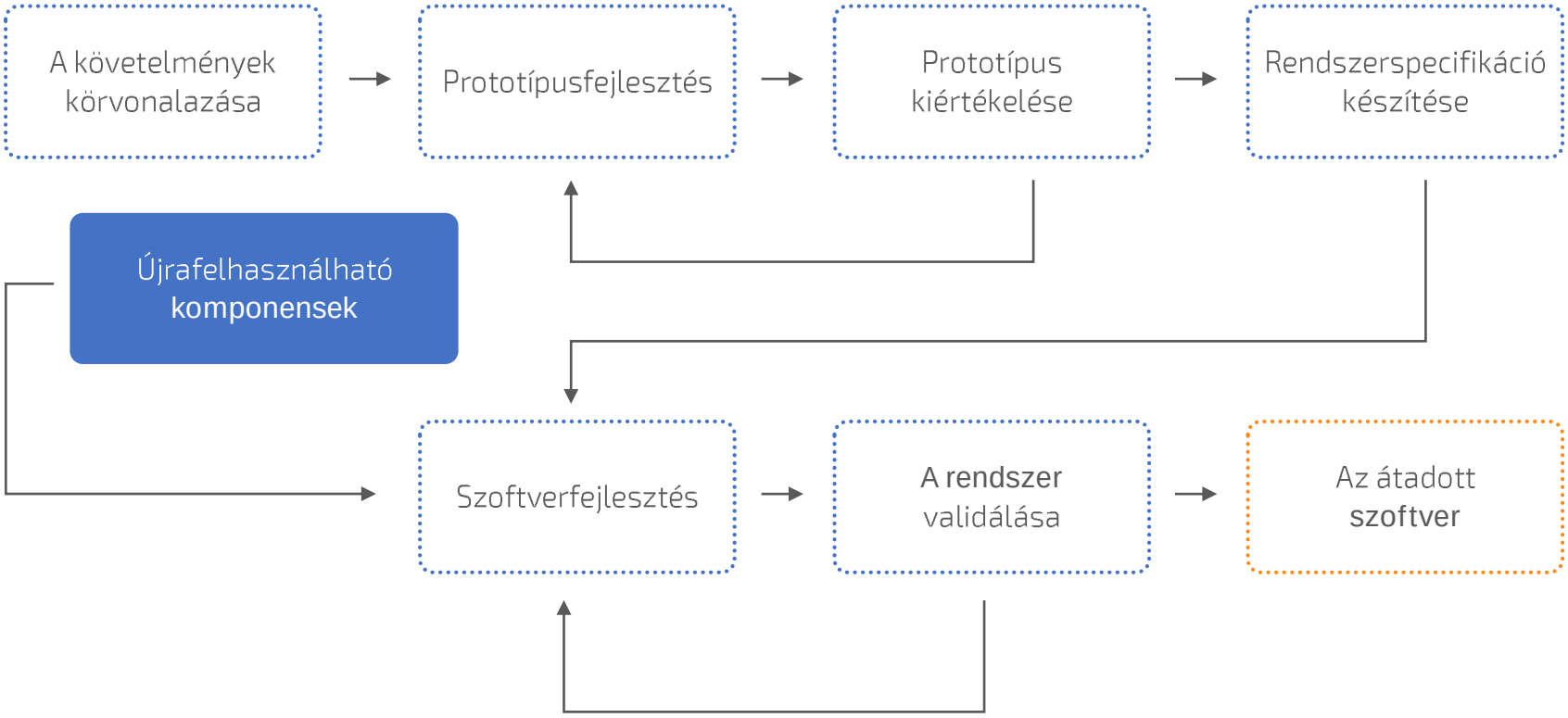
PÁZMÁNY PÉTER KATOLIKUS EGYETEM - SZÉCHY TUDOMÁNYEGYETEMI INTÉZMÉNY
INFORMÁCIÓS TECHNOLÓGIAI ÉS BIONIKAI KAR - KUTATÓ KAR

Eldobható prototípus készítése

- Célja a követelményspecifikációból fakadó kockázat csökkentése.
- A prototípust egy kezdeti specifikáció alapján készítik, validálásra átadják a felhasználónak, majd eldobják.
- Az eldobható prototípus nem tekinthető végleges rendszernek, mert:
  - A kivételek, hibák kezelése általában hiányzik.
  - Több rendszertulajdonság kimaradhat a prototípusból.
  - Nem készül specifikáció a hosszú távú karbantartásra.
  - A prototípus még nem a megfelelő struktúra szerint épül.

Eldobható prototípus készítése

Az eldobható prototípuskészítés folyamata



Eldobható prototípus készítése
A prototípus átadása

- A vezetők gyakran nyomást gyakorolnak a fejlesztőkre, hogy egy működő *eldobható prototípust végleges rendszerként* adjanak át.
- Ez nagyon veszélyes, mert:
  - Az eldobható prototípus nem alakítható úgy, hogy a nem-funkcionális követelményeknek *(teljesítmény, megbízhatóság, skálázhatóság, stb.)* eleget tegyen.
  - A prototípus rendszerint dokumentálatlan marad, mert a cél a gyors elkészítés és bemutatás.
  - A változtatások miatt a rendszer struktúrája általában romlik a fejlesztés során.
  - A prototípus készítésekor az általános szervezeti szabványokat nem tartják be *(minőségbiztosítás, technológiai fegyelem, projektdokumentálás).*

Tartalom

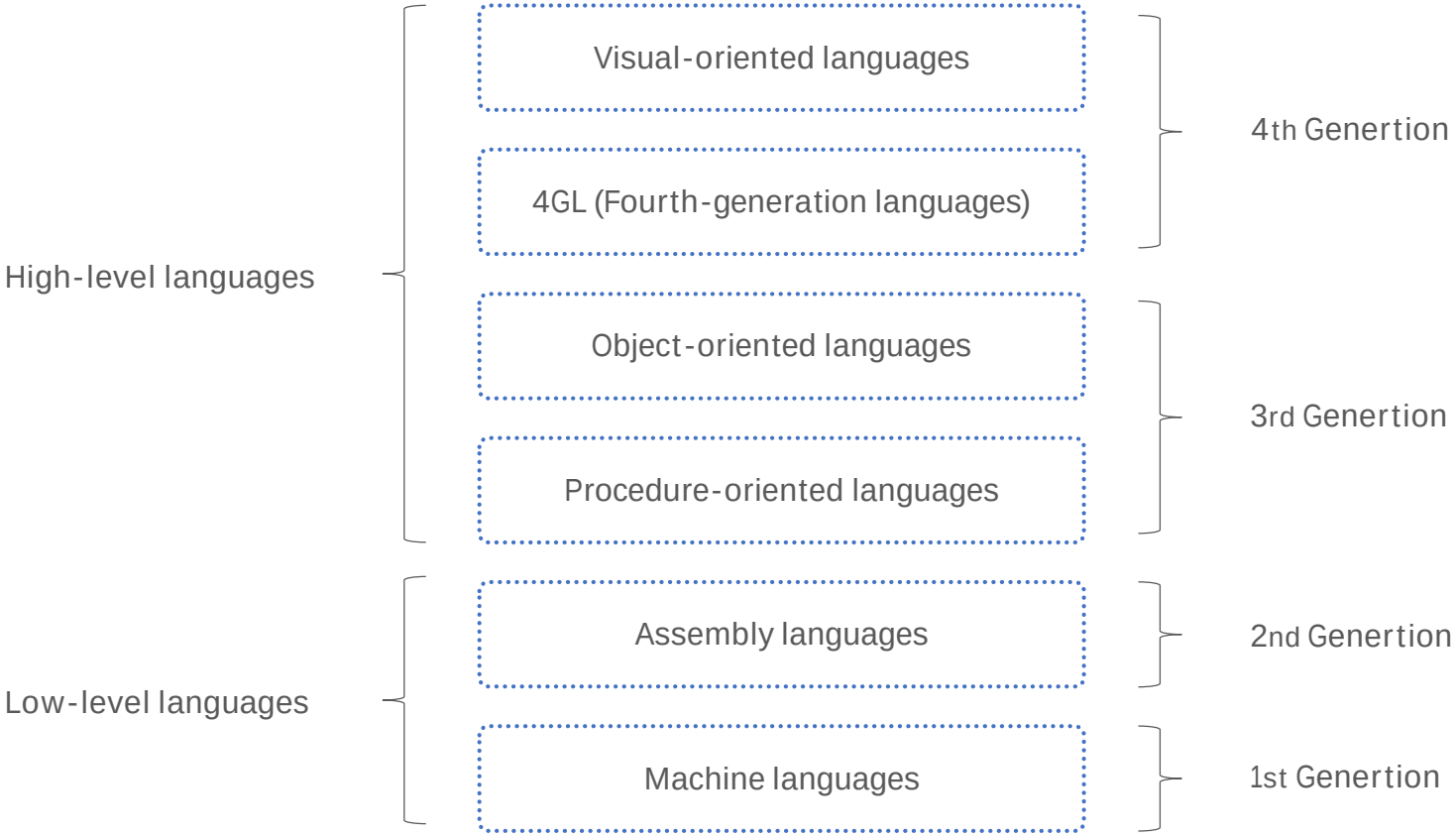
Gyors prototípuskészítési technikák

- A gyors prototípuskészítéshez az alábbi technikák alkalmazhatók:
  - Fejlesztés dinamikus, magas szintű nyelven,
  - Adatbázis-programozás,
  - Grafikus modellek és fejlesztő eszközök,
  - Komponensek és alkalmazások összeépítése.
- A gyakorlatban ezeket együttesen alkalmazzák.
- A legtöbb prototípuskészítő eszköz tartalmazza a *vizuális programozás támogatását*, ahol grafikus szimbólumok reprezentálják a függvényeket, adatokat, feldolgozószkripteket, akár összetett üzleti funkciókat. Az eszköz a rendszer vizuális reprezentációjából generálja a végrehajtható programot.

Magas szintű nyelvek

Magas szintű nyelvek

A programozási nyelvek fejlődése



Magas szintű nyelvek
Fejlesztés magas szintű nyelven

- Olyan nyelvek, amelyek hatékony futási idejű adatkezelő eszközöket tartalmaznak.
- Korábban a nagy rendszerek fejlesztéséhez nem használtak ilyeneket, mert nagy teljesítményű futtató rendszereket igényelnek, ami növeli a tárigényt, csökkenti a futási teljesítményt.
- Némely nyelv olyan integrált támogató környezetet tartalmaz, amely felhasználható a gyors prototípuskészítéshez.
- A magas szintű nyelvek többsége fejlett felhasználói interfész-fejlesztő képességekkel rendelkezik.

Magas szintű nyelvek
3.-4. generációs nyelvek

- 3GL
  - C, C++, C#, Java
  - egy 3GL utasítás kb. 5-10 assembly utasítás

- 3GL és 4GL között
  - Python, Ruby

- 4GL
  - ORACLE ADF, LabVIEW, ORACLE Reports, SQL
  - Számos 3GL nyelvhez van kiegészítő *library* 4GL szintű funkcionalitással
  - Egy 4GL utasítás kb. 30-40 assembly utasítás

Magas szintű nyelvek
4GL jellemzői

- Tartalmaz:
  - GUI tervezés
  - Vizuális fejlesztőkörnyezet
  - A felhasználó is megértheti, sőt programozhat is
  - A programkészítés termelékenységét növeli (van ellenkező tapasztalat is)
- De:
  - Több erőforrást igényel
  - Nehezen menedzselhető
  - Szegényes a tervezési módszertani támogatás
  - Több száz 4GL nyelv létezik, nem mindegyik jó mindenre

- James Martin: prototípuskészítésre, iteratív tervezésre, adatkezelésre használható.

Magas szintű nyelvek
4GL fejlesztői környezetek

- A nagy adatbázisok és alkalmazási rendszerek szállítói kifejlesztették 4GL fejlesztői környezeteiket is.
- Ezek grafikus segítséget nyújtanak:
  - nemcsak az adatbázis tervezéséhez, kezeléséhez
  - az alkalmazás felhasználói felületeinek tervezéséhez,
  - report készítéséhez, stb.
- Továbbá támogatják:
  - az üzleti folyamatok tervezését,
  - az alkalmazásintegrációt
    (régi rendszerekkel, partnerek rendszereivel)
  - a vezetéstámogató eszközöket
    (Business Intelligence Beans)

Magas szintű nyelvek
Prototípuskészítő nyelvek

- Szempontok az alkalmas nyelv kiválasztásához:
  - **Az alkalmazási terület** jellege

    Természetes nyelvű feldolgozáshoz a Lisp vagy a Prolog alkalmasabb.
  - A **felhasználói interakció** jellege

    Az intenzív Web alapú felhasználói interakció kidolgozására a Java, a SmallTalk vagy a szkriptnyelvek több eszközt kínálnak.
  - **A támogatási környezet**

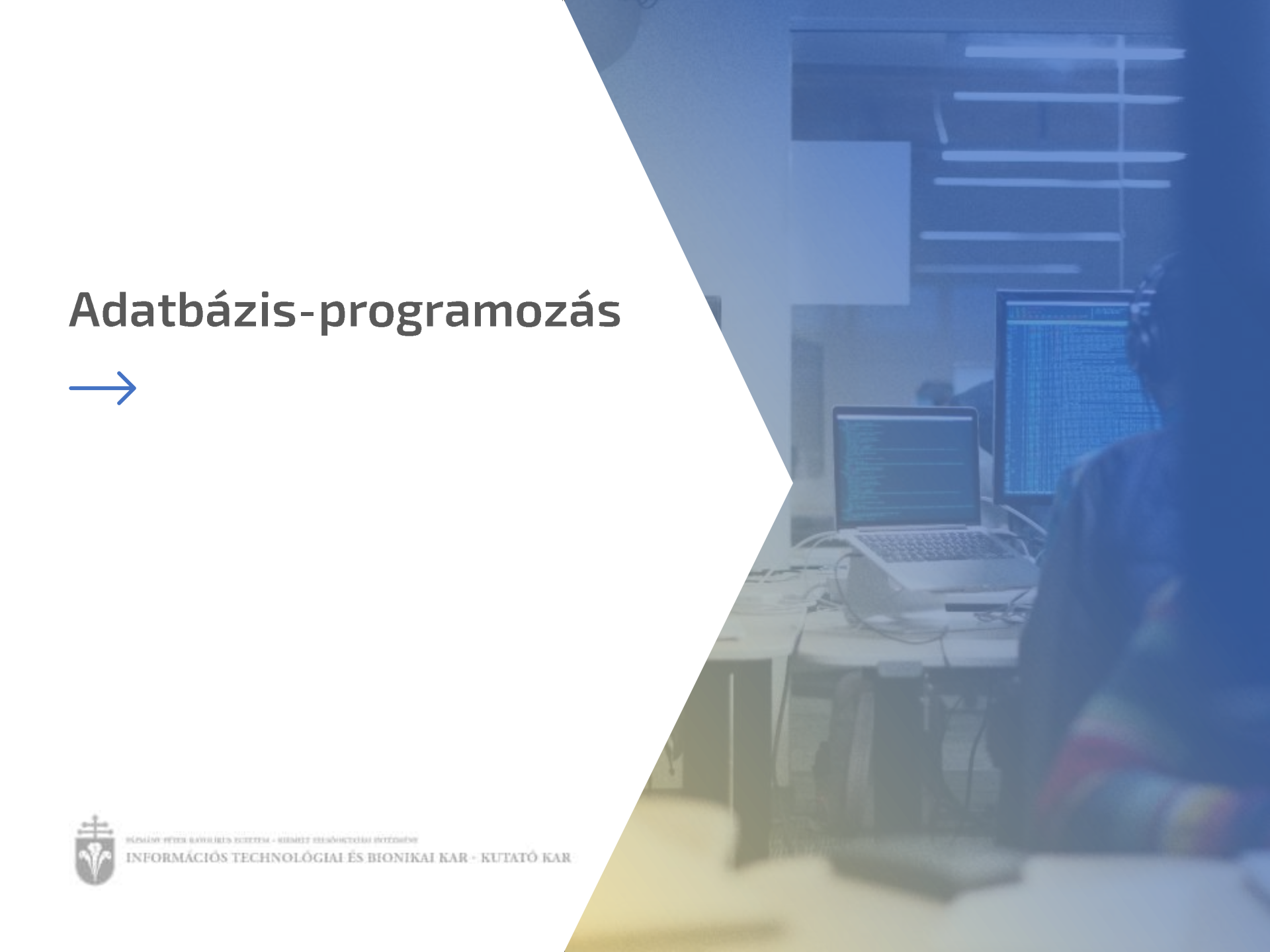
    A nyelvvel együtt sokféle eszköz és komponens segíti a prototípuskészítést.
- A rendszerprototípus különböző részei különböző nyelven programozhatók.
- A különböző nyelven írt részek közötti kapcsolatot kommunikációs keretrendszer teremtheti meg.

Magas szintű nyelvek
Prototípuskészítő nyelvek

NYELV	TÍPUS	ALKALMAZÁSI TERÜLET
SmallTalk	Objetumorientált	Interaktív rendszerek
Java	Objetumorientált	Interaktív rendszerek
Prolog	Logikai	Szimbolikus feldolgozás
Lisp	Listaalapú	Szimbolikus feldolgozás

- Nagy rendszerek prototípusának készítésekor nincs egyetlen ideális nyelv. A rendszer egyes részei az igényeknek megfelelő leginkább alkalmas nyelven készülhetnek.
- Mivel a nyelvek eltérő egyedfogalmakat használhatnak, a részek közti adat- és vezérlési kapcsolatok sok addicionális kódot igényelhetnek.

Adatbázis-programozás

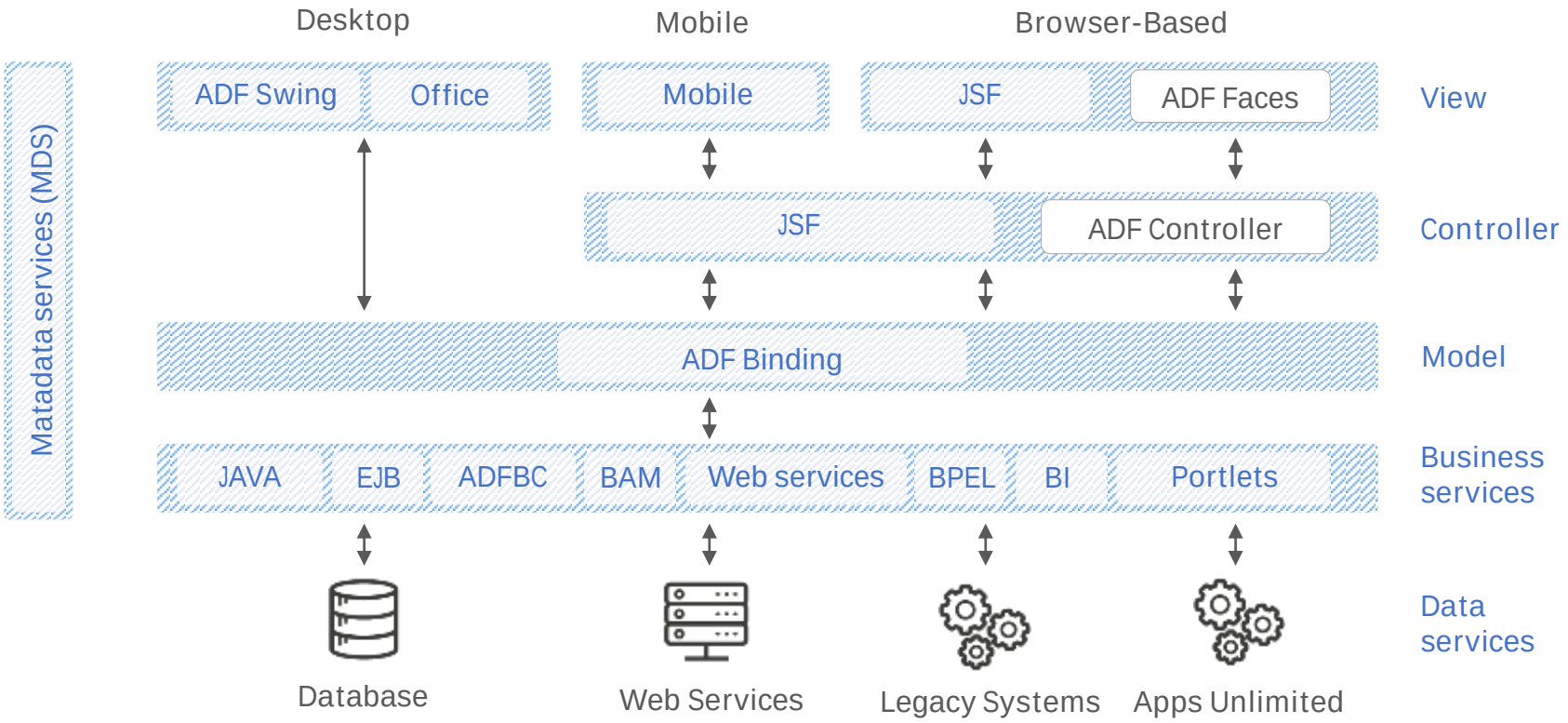


PÁZMÁNY PÉTER KATOLIKUS EGYETEM - SZÉKELY TUDOMÁNYTÉKAI INTÉZMÉNY
INFORMÁCIÓS TECHNOLÓGIAI ÉS BIONIKAI KAR - KUTATÓ KAR

Adatbázis-programozás

- Az evolúciós fejlesztés az adatbázison alapuló üzleti alkalmazások területén általánosan alkalmazott technika.
- A kereskedelmi adatbázis-kezelő rendszerek olyan 4GL fejlesztő eszközöket tartalmaznak, amelyek támogatják a lekérdezést/adatkezelést *(SQL)*, táblázatkezelést, jelentésgenerálást, felhasználói felületek tervezését, stb.
- Az adatfeldolgozási alkalmazásokban sok közös feladat van: adatbázis-manipulációk *(keresés, frissítés, rendezés, report készítése, stb.)*, egyszerű műveletek, űrlapkezelés, stb. Egy 4GL-ben ezeket általánosítják.
- Gyakran integrálhatók CASE eszközökkel is. Ezek generálhatnak SQL-t vagy alacsonyabb szintű kódot.

Adatbázis-programozás
Oracle Application Development Framework



- JEE platformon *(pl. JSF könyvtár, 150 üzleti mintával)*
  Eclipse interfész, integrált biztonsági funkciók
  http://www.oracle.com/technetwork/developer-tools/index.html

Felhasználói felületek prototípusai

INFORMÁCIÓS TECHNOLÓGIAI ÉS BIONIKAI KAR - KUTATÓ KAR

Felhasználói felületek prototípusai

- A felhasználót be kell vonni a felhasználói felületek tervezésébe, a fejlesztő nem erőltetheti rá saját elképzeléseit.
- A prototípusok készítése segít a felhasználók bevonásában.
- A felhasználói interfész fejlesztése a rendszerfejlesztési költségek növekvő hányadát adja.
- Az interfészgenerátorok segítenek abban, hogy a felhasználók gyorsan véleményt alkossanak a felületekről. A felületek specifikációjából jól strukturált programot generálnak.
- A web alapú felhasználói interfészek készítésére jól ismert weblaptervező-eszközök léteznek.

Komponensek és alkalmazások integrálása

INFORMÁCIÓS TECHNOLÓGIAI ÉS BIONIKAI KAR - KUTATÓ KAR

Komponensek és alkalmazások integrálása

- A prototípus gyorsabban összeállítható újrafelhasználható komponensekből, amelyek összeépítését kompozíciós mechanizmus támogatja.
- A kompozíciós mechanizmusnak szabványos vezérlő és kommunikációs eszközöket kell tartalmaznia a komponensek integrálásához.
- A rendszerspecifikáció készítésekor figyelembe kell venni, milyen újrafelhasználható komponensek állnak rendelkezésre. Ez a követelményekkel kapcsolatban kompromisszumokat követelhet.

Komponensek és alkalmazások integrálása
Prototípuskészítés újrafelhasználással

- **Az alkalmazás szintjén:**

  Teljes alkalmazási rendszerek integrálása a prototípusba úgy, hogy egymás funkcióit megosztva használják.

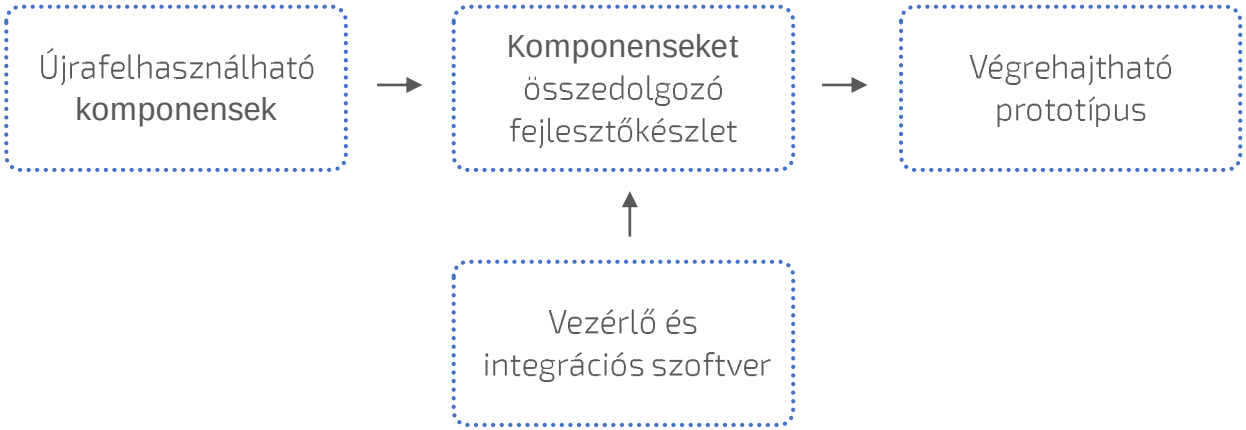
  (Például szabványos szövegszerkesztő alkalmazása a szövegszerkesztésre.)

- **A komponensek szintjén:**

  Az egyedi komponensek integrálása egy szabványos keretrendszerbe.

  (Például egy szkriptnyelv, mint a Visual Basic, vagy olyan integrációs keretrendszer, amely CORBA szabványon alapul, vagy .net, JavaBean futtatásra alkalmas.)

Komponensek és alkalmazások integrálása
Újrafelhasználható komponensek alkalmazása



- **Komponenskönyvtárak**:

  Nem megoldott a komponensek tulajdonságainak, interfészeinek szabványos nyilvántartása.

- **Új dimenzió:** webServices

  Az igénybevett szolgáltatás nem komponens formájában épül be a rendszerbe, hanem weben elérhető szolgáltatásként.

  WSDL – Web Service Definition Language *(W3C szabvány)*

Komponensek és alkalmazások integrálása
Alkalmazások újrafelhasználása

- Ha az alkalmazás szkriptkészítő és konverziós eszközöket is tartalmaz, a prototípus funkciói kidolgozhatók.
- Például: az MsOffice makrók alkalmazásával az egyes alkalmazások integrációja összetett dokumentumokkal kialakítható.
- Az újrafelhasználható alkalmazás összes funkciója rendelkezésre áll.
- Akkor működik jól, ha a felhasználók ismerik az alkalmazásokat. Ha nem, a sok fölöslegesnek látszó funkció zavaróvá válhat.

Komponensek és alkalmazások integrálása
A Service Oriented Architecture - SOA

- A szolgáltatásközpontú architektúra lényege, hogy nem fejlesztik ki minden újabb alkalmazáshoz az azonos funkciókat, hanem az interneten keresztül, szolgáltatásként veszik azokat igénybe.
- Az OO komponensalapú fejlesztés hátránya, hogy szabványok híján nehéz az idegen forrásból származó komponenseket a saját rendszerbe integrálni.
- Ehelyett a SOA szerint a kívánt funkciót a weben elérhető szolgáltatásként veszik igénybe.
- A Web Services szabványai *(pl. WSDL a szolgáltatások leírására, stb.)* megkönnyítik ezen szolgáltatások igénybevételét.

Komponensek és alkalmazások integrálása
A Service Oriented Architecture - SOA

- A SOA három komponense:
  - **Szolgáltató (Service** Provider):

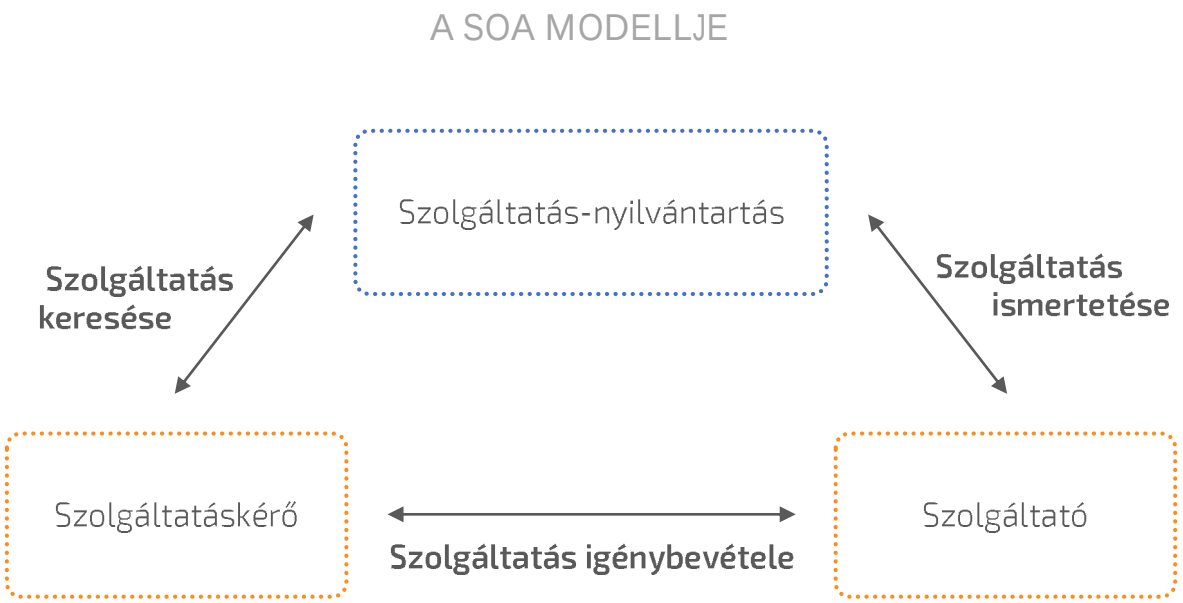
    A szolgáltatás tulajdonosa, felelős azért, hogy szolgáltatása a leírásoknak megfelelően működjön.
  - **Szolgáltatáskérő** (Service Requestor):

    Egy igény bevehető szolgáltatást keres a nyilvántartásban, a feladatából származó követelményeknek megfelelően.
  - **Szolgáltatás-nyilvántartás** (Service Registry):

    Egy (megosztott) szolgáltatástárház, ahol a szolgáltatáskérő megtalálhatja a kívánságainak megfelelő szolgáltatást.

    A SOA esetén ez nem feltétlenül szükséges, ha az igénylő már ismeri a szolgáltatót.

Komponensek és alkalmazások integrálása
A Service Oriented Architecture - SOA

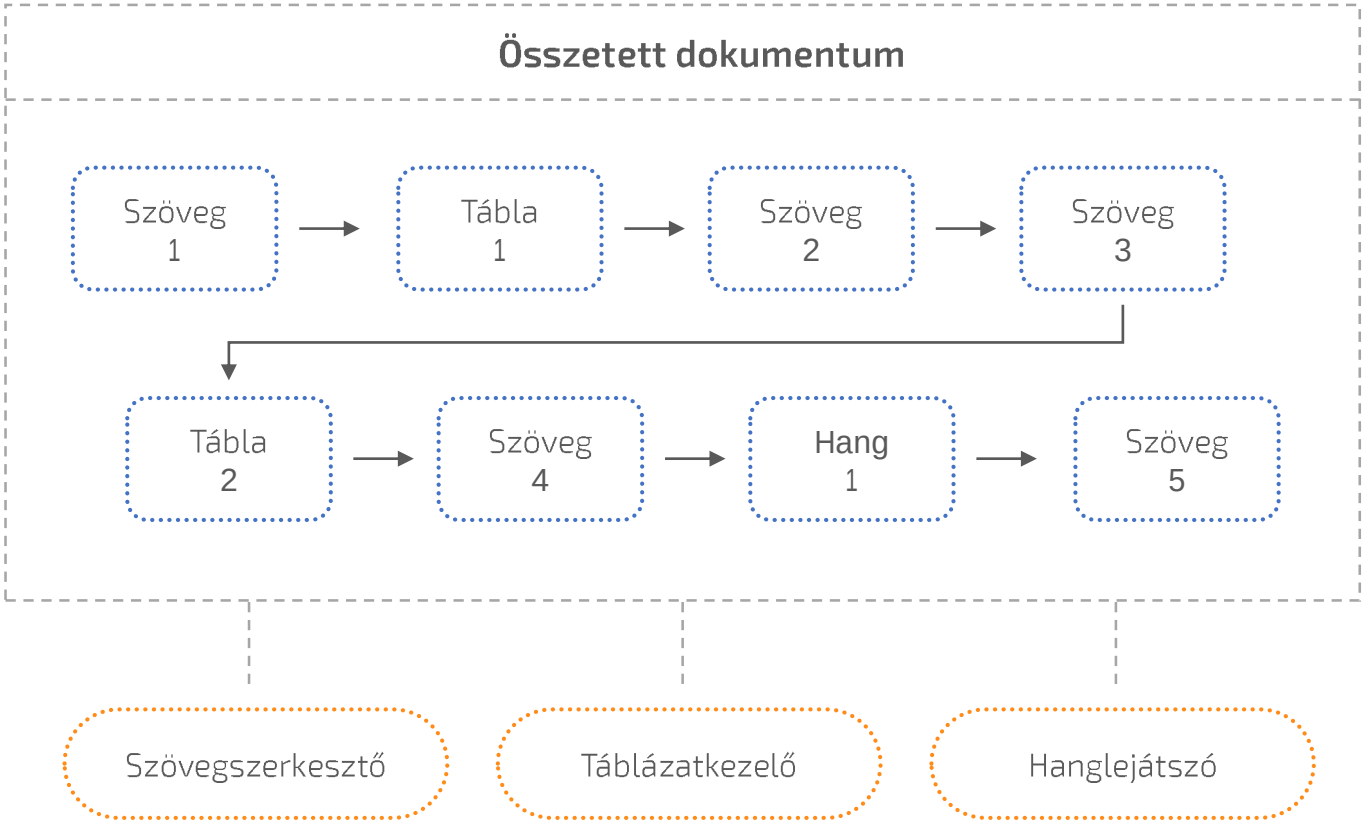


A SOA-ban nagy szerepet kap az üzenetküldő köztes réteg *(middleware)*.

Komponensek és alkalmazások integrálása
Összetett dokumentumok

- Egyes alkalmazásokból prototípus fejleszthető összetett dokumentumok alkalmazásával.
- Az összetett dokumentum *(pl. egy táblázat)* több különböző aktív elemet tartalmazhat, amelyek mindegyikéhez egy-egy alkalmazás tartozik. Amikor a felhasználó az adott elemet kiválasztja, aktiválódik a hozzá tartozó alkalmazás.
- Az összetett dokumentum integráló szerepet játszik a különböző alkalmazások között.

Komponensek és alkalmazások integrálása
Összetett dokumentumok



Tartalom

Tervezés újrafelhasználással
A szoftver újrafelhasználása

- A legtöbb mérnöki tervezési tevékenység komponensek újrafelhasználásán alapul. A terveket más rendszerekben már kipróbált, szabványos, kisebb-nagyobb komponens újrafelhasználására alapozzák *(csavaroktól a hajtóművekig).*
- A szoftverfejlesztés hagyományosan az eredeti fejlesztésen alapul, de a minőség javítása, a költségek és a fejlesztési idő csökkentése érdekében mindinkább előtérbe kerül a szoftver komponensek újrafelhasználása.
- Ehhez olyan tervezési módszereket kell alkalmazni, amely a szisztematikus újrafelhasználáson alapul.

Tervezés újrafelhasználással
Újrafelhasználáson alapuló szoftverfejlesztés

- **Alkalmazási rendszerek újrafelhasználása**
  - Teljes alkalmazási rendszerek újrafelhasználása:
  - Beépítve más rendszerekbe, vagy
  - Speciális felhasználói igényeket kiszolgáló alkalmazáscsaládok kifejlesztése.
- **Komponensek újrafelhasználása**
  - Különböző méretű *(objektum – alrendszer)* komponensek beépítése új rendszerekbe. *(pl. driverek, interfészmodulok, stb.)*
- **Függvények újrafelhasználása**
  - Egyszerű, jól definiált tevékenységet végző komponensek újrafelhasználása. *(pl. szabványos könyvtárak)*

Tervezés újrafelhasználással
Az újrafelhasználás előnyei

- **Javuló megbízhatóság**
  A komponenseket már több működő rendszerben kipróbálták.

- Alacsonyabb **projektkockázat**
  A komponensek ára és adaptálási költsége pontosabban tervezhető.

- **A szaktudás jobb kihasználása**
  A **speciális szaktudás a komponensben testesül meg**, nem szükséges minden projekthez külön alkalmazni.

- **Szabványosság**
  A szabványoknak való megfelelést a komponensek garantálják *(interfészek, kommunikációs és GUI szabványok)*

- Gyorsabb **fejlesztés**
  Egy rendszer kifejlesztése gyorsabb, ha kevesebb eredeti fejlesztést igényel.

Tervezés újrafelhasználással
Az újrafelhasználás hátrányai

- **Növekvő karbantartási költségek**

  A komponens forráskódja és tervezési dokumentációja hiányában növekszik a karbantartás költsége.

- **Az eszköztámogatás hiánya**

  A CASE eszközök gyakran nem támogatják az újrafelhasználást.

- A ***„nem mi találtuk ki”*** **jelenség**

  Egy teljes rendszer kidolgozása nagyobb szakmai kihívás.

- **A komponenskönyvtárak karbantartása**

  Sokba kerül a komponenskönyvtárak feltöltése és folyamatos karbantartása.

- **Az újrafelhasználható komponensek megtalálása és adaptálása**

  Még nem fejlődtek ki a komponensek megtalálását és adaptálását segítő általános technikák.

Tervezés újrafelhasználással
Kritikus követelmények

- Meg kell találni a megfelelő újrafelhasználható komponenseket. Ehhez katalógusokra és nyilvántartásokra, alkalmas keresőmechanizmusokra van szükség.
- Az újrafelhasználónak bíznia kell abban, hogy a komponens a leírtaknak megfelelően és megbízhatóan működik.
- A komponenseknek olyan dokumentációval kell rendelkezniük, amely érthető, teljes, aktuális és hivatkozik a korábbi felhasználásokra is *(referenciák)*.

Újrafelhasználás programgenerátorral

PÁZMÁNY PÉTER KATOLIKUS EGYETEM - RÉSZLET TUDOMÁNYTERÜLETI INTÉZMÉNYE
INFORMÁCIÓS TECHNOLÓGIAI ÉS BIONIKAI KAR - KUTATÓ KAR

Újrafelhasználás programgenerátorral

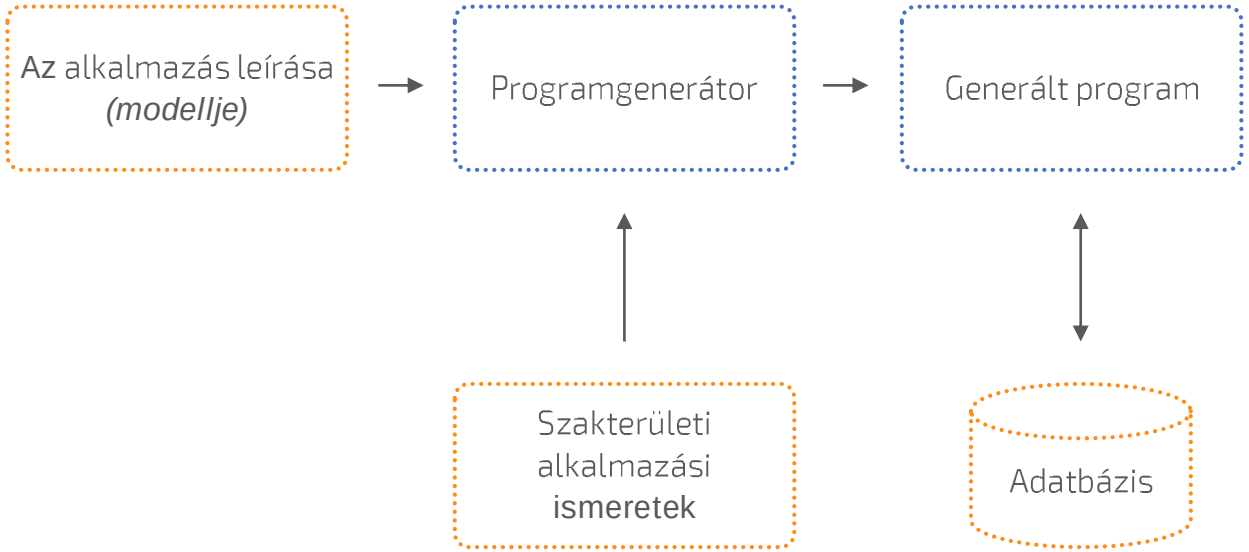
- A generátoralapú újrafelhasználás akkor lehetséges, ha egy programgenerátor tartalmazza egy szakterület alapvető ismeretanyagát. *(pl. adatfeldolgozás)*
- Az ilyen programgenerátorok tartalmazzák a szabványos algoritmusokat és függvényeket, és ezek paraméterezését követően a generátor automatikusan előállítja a programot.
- A szakterületre kidolgozott nyelven vagy újabban grafikus eszközökkel lehet elkészíteni a rendszer modelljét.

(Ebben az esetben elsősorban a szakterületi tudás újrafelhasználásáról van szó.)

Újrafelhasználás programgenerátorral
A programgenerátorok típusai

- A generátorok fajtái:
  - **Alkalmazásgenerátorok** - üzleti adatfeldolgozó rendszerek készítésére.
  - **Szintaktikus elemzők** - a programozási nyelvek értelmezésére.
  - **CASE eszközökben lévő kódgenerátorok** – egy szoftvertervből a tervezett rendszer implementációját állítják elő.
- A generátoralapú újrafelhasználás igen költséghatékony, de viszonylag kevés szakterülethez léteznek ilyen rendszerek *(üzleti adatfeldolgozás, eBusiness, stb.)*
- Ezzel a módszerrel könnyebben állíthatók elő az alkalmazások, mint a komponensalapú módszerrel.

Újrafelhasználás programgenerátorral
A generátor alapú újrafelhasználás folyamata



Komponensalapú fejlesztés

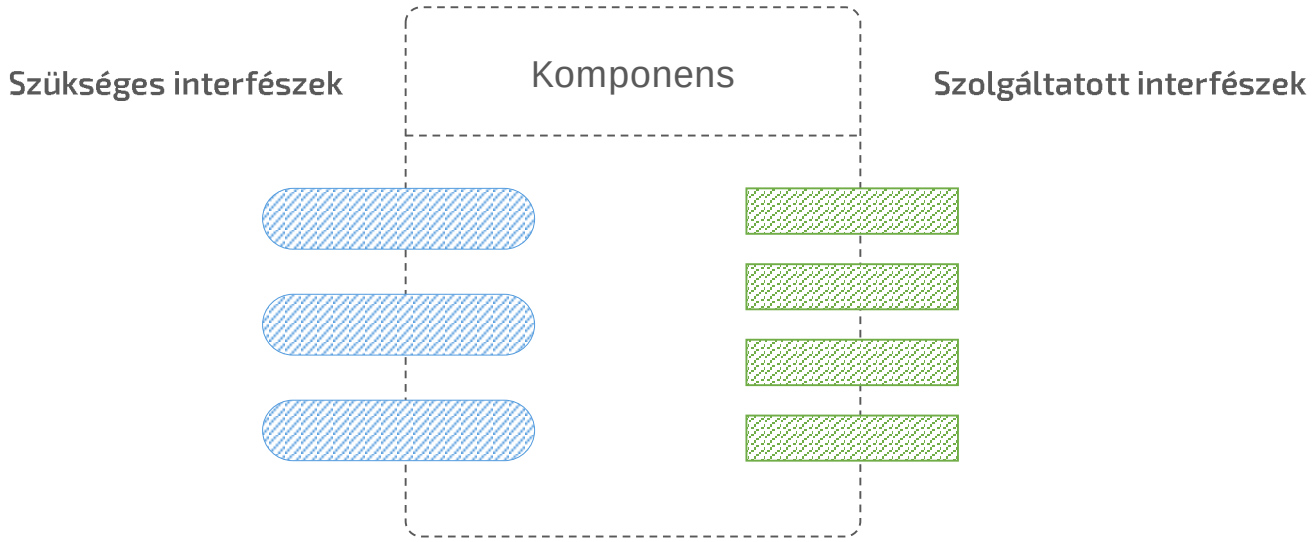
PÁZMÁNY PÉTER KATOLIKUS EGYETEM - SZÉCHY TUDOMÁNYTERÜLETI INTÉZMÉNY
INFORMÁCIÓS TECHNOLÓGIAI ÉS BIONIKAI KAR - KUTATÓ KAR

Komponensalapú fejlesztés

- A komponensalapú szoftverfejlesztés *(CBSE – Component Based Software Engineering)* az újrafelhasználáson alapul.
- Kialakulásának oka az, hogy az objektumorientált fejlesztés nem igazán támogatja az újrafelhasználást, mert:
  - Az egyedi objektumosztályok túl részletesek és specifikusak és csak a folyamat késői fázisában kapcsolódnak az alkalmazáshoz.
  - Nem alakult ki olyan piac, ahol az egyes szakterületek objektumosztályaihoz lehetne hozzájutni.
- A komponensek az objektumosztályoknál sokkal absztraktabbak és különálló szolgáltatásoknak tekinthetők.

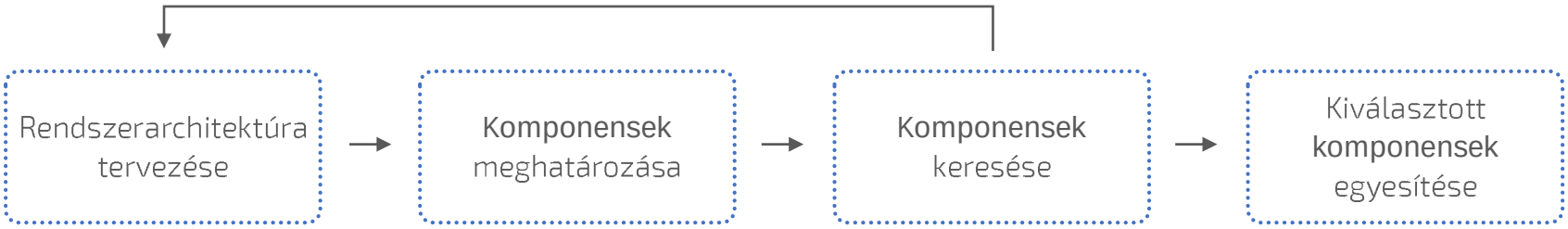
Komponensalapú fejlesztés
A komponensek interfészei

- **Szolgáltatott interfészek** – a komponens által szolgáltatott interfészek.
- **Szükséges interfészek** – azok az interfészek, amelyeket a komponenst használó rendszernek vagy környezetének kell biztosítania.



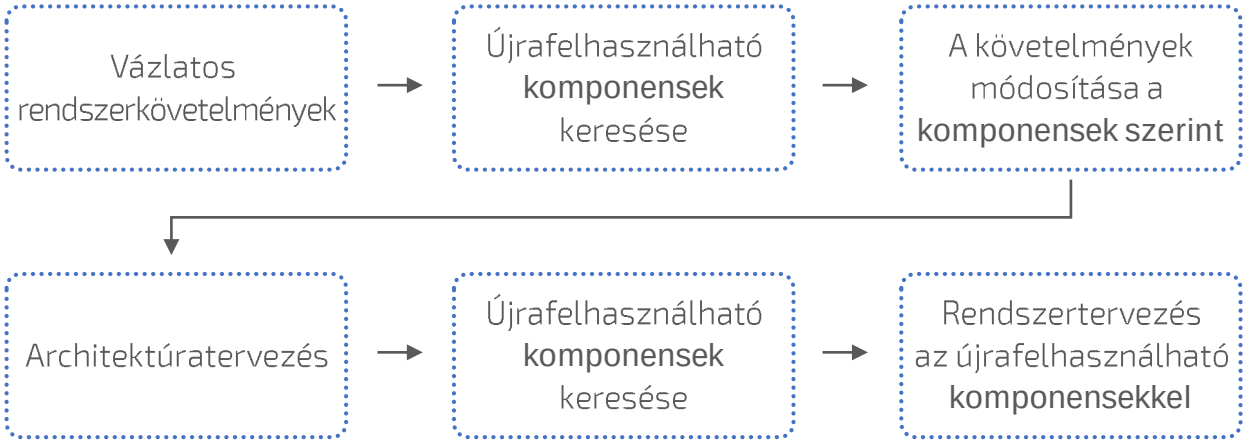
Komponensalapú fejlesztés
A komponensalapú fejlesztés folyamata

- A komponensalapú fejlesztés beilleszthető a szabályos szoftverfolyamatba, ha beépítjük abba az újrafelhasználással kapcsolatos tevékenységeket:
- Komponensek specifikálása,
- Komponensek megtalálása,
- A tervek *(esetleg a követelmények)* módosítása a meglelt komponensek tulajdonságainak megfelelően.
- Ez az **alkalmazkodó újrafelhasználás**



Komponensalapú fejlesztés

Fejlesztés komponensek újrafelhasználásával



- A rendszer megvalósítása történhet prototípuskészítéssel vagy inkrementális módon.
- A legtöbb programozási nyelvben hivatkozhatunk könyvtárban tárolt komponensekre
- Leggyakrabban script nyelvet használnak a komponensek integrálására.

Komponensalapú fejlesztés
Hátrányok, nehézségek

- A komponensek inkompatibilitása miatt a költség- és időmegtakarítás a vártnál kevesebb lehet *(integrációs munkák).*
- Nehézséget okozhat a komponensek megtalálása *(nincsenek szabványok a komponensek tulajdonságainak leírására, hiányoznak az egységes komponenskönyvtárak).*
- A követelmények változását követő evolúció lehetetlen, ha a komponensek nem cserélhetők.
- A karbantartás nehezebb.
- Mindezek ellenére *(a fejlesztési idő és a szoftverek élettartamának csökkenése)* sokszor megéri az újrafelhasználható komponensek alkalmazása.

Komponensalapú fejlesztés
Alkalmazás-keretrendszerek

PÁZMÁNY PÉTER KATOLIKUS EGYETEM - RÉSZLET TUDOMÁNYTERÜLET INTÉZMÉNYE
INFORMÁCIÓS TECHNOLÓGIAI ÉS BIONIKAI KAR - KUTATÓ KAR

Komponensalapú fejlesztés
Alkalmazás-keretrendszerek

- A keretrendszer absztrakt és konkrét osztályok gyűjteményéből és a köztük lévő interfészekből álló alrendszerterv.
- A keretrendszerek úgy implementálhatók, hogy a terv részeit komponensek hozzáadásával egészítjük ki.
- Általában viszonylag nagy, újrafelhasználható egységek, de nem önálló alkalmazások.
- Az alkalmazások több keretrendszer integrálásával hozhatók létre.

Komponensalapú fejlesztés

Alkalmazás-keretrendszerek

A KERETRENDSZEREK CSOPORTOSÍTÁSA

- **A rendszer infrastruktúrájának keretrendszerei**
  - A rendszer infrastrukturális alapjainak *(kommunikáció, felhasználói felületek, titkosítás, stb.)* fejlesztését támogatják.
- **Köztes, integrációs keretrendszerek**
  - komponensek közti kommunikációt és információcserét támogató szabványok és osztályok. *(Ilyen például a CORBA, JavaBean, JMI, COM, DCOM, .NET, stb.)*
- **Vállalati alkalmazások keretrendszerei**
  - Az egyes speciális szakterületi alkalmazások fejlesztését támogatják. A szakterületi tudást tartalmazzák *(pl. pénzügy, telekommunikáció).*

Komponensalapú fejlesztés

Alkalmazás-keretrendszerek

A KERETRENDSZEREK KIBŐVÍTÉSE

- A keretrendszer általános struktúra, amely a konkrét alkalmazás létrehozásakor konkrét osztályokkal kibővíthető.
- A keretrendszer kibővítése az alábbiakat jelenti:
  - A keretrendszer absztrakt osztályainak kiegészítése konkrét osztályokkal.
  - Műveletek hozzáadása, amelyek meghívhatók a keretrendszer által kezelt események bekövetkezésekor.
- A keretrendszerek hátránya a bonyolultság. Sok időt igényel az effektív használatukhoz szükséges megismerésük.

Komponensalapú fejlesztés

„Polcról levehető" termékek

PÁZMÁNY PÉTER KATOLIKUS EGYETEM - RÉSZLET TUDOMÁNYTERÜLET INTÉZMÉNYE
INFORMÁCIÓS TECHNOLÓGIAI ÉS BIONIKAI KAR - KUTATÓ KAR

Komponensalapú fejlesztés

„Polcról levehető" termékek

- A *„polcról levehető"*, COTS – Commercial Off-The-Shelf rendszerek általában komplett alkalmazási rendszerek, amelyek APIval rendelkeznek.
- Legtöbbször rendszerszoftvertermékek az egyszerű komponenseknél nagyobb funkcionalitással.
- Nagy rendszerek építésekor gyakran használt stratégia a COTS termékek integrálása, különösen a gyors fejlesztést kívánó eCommerce, eBusiness rendszerek körében. A fejlesztési idő nagyságrendekkel csökkenthető.

Komponensalapú fejlesztés

„Polcról levehető" termékek

COTS integrációs nehézségek

- A funkcionalitás és a teljesítmény nem tartható kézben:
  - A COTS rendszerek sokszor kevésbé effektívek, mint azt a reklámokban ígérik.
- A COTS rendszerek együttműködése bizonytalan
  - A különböző COTS rendszerek eltérő feltételezésekkel készültek *(pl. sorkezelés)*, ezért az integráció nehéz lehet.
- Az evolúció ellenőrizhetetlen
  - A szállító és nem a felhasználó határozza meg.
- A COTS termékek támogatása
  - A szállító által biztosított támogatás gyakran nem terjed ki a rendszer teljes élettartamára.

Komponensalapú fejlesztés
Újrafelhasználásra fejlesztett komponensek

PÁZMÁNY PÉTER KATOLIKUS EGYETEM - SZÉCHY TUDOMÁNYTERÜLETI INTÉZMÉNY
INFORMÁCIÓS TECHNOLÓGIAI ÉS BIONIKAI KAR - KUTATÓ KAR

Komponensalapú fejlesztés

Újrafelhasználásra fejlesztett komponensek

KOMPONENSEK **FEJLESZTÉSE ÚJRAFELHASZNÁLÁSRA**

- Az újrafelhasználható komponensek meglévő komponensek általánosításával hozhatók létre. Ehhez nagy tapasztalat kell.
- A komponens újrafelhasználhatóságának jellemzői:
  - Stabil szakterületi absztrakciókra támaszkodik.
  - El kell rejtenie az állapotait és műveleteket kell biztosítania a az állapotához való hozzáférésre.
  - Amennyire lehetséges, függetlennek és önállónak kell lennie.
  - A hibakezelést interfészeken keresztül kell megoldania.
- Az újrafelhasználhatóság és a használhatóság ellentmondása:
  - Minél általánosabb interfésszel rendelkezik, annál inkább újrafelhasználható, de annál bonyolultabb, vagyis kevésbé használható.

Komponensalapú fejlesztés

Újrafelhasználásra fejlesztett komponensek

ÚJRAFELHASZNÁLHATÓ KOMPONENSEK FEJLESZTÉSE

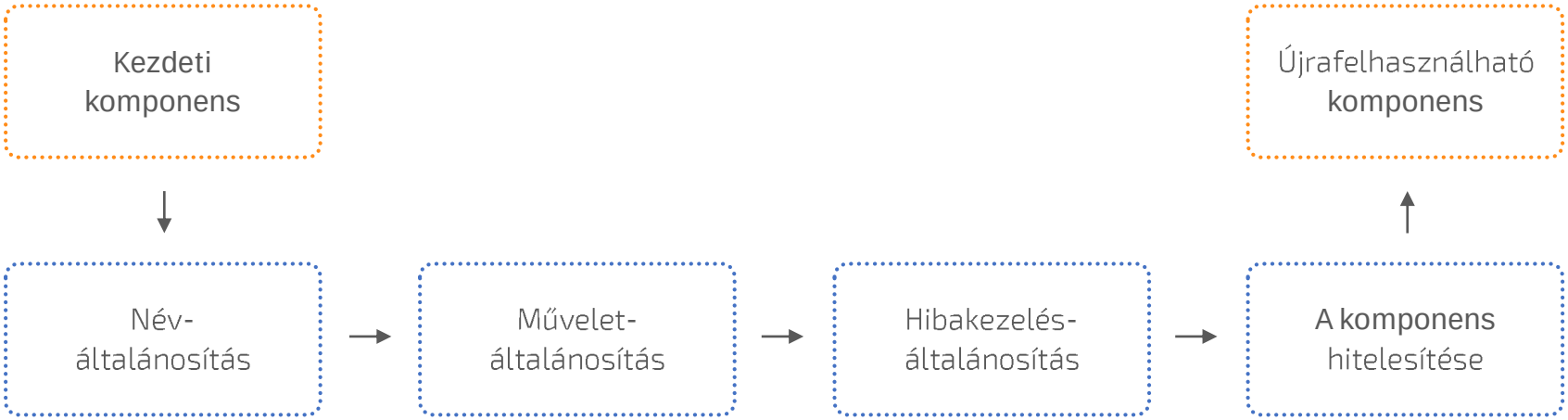
- Az újrafelhasználható komponensek fejlesztési költségei többszörösen meghaladják az egyszerű, specifikus komponensek költségeit. Ezt egyetlen projekt költségeiből nem lehet fedezni, ezért kialakultak olyan szoftverfejlesztő vállalatok, amelyek specializálódtak az újrafelhasználható komponensek fejlesztésére.
- Az általános komponensek kevésbé effektívek, több erőforrást használnak és végrehajtási idejük hosszabb, mint a specifikus komponenseké.

Komponensalapú fejlesztés

Újrafelhasználásra fejlesztett komponensek

SPECIFIKUS KOMPONENS KITERJESZTÉSE

Egy specifikus komponens újrafelhasználhatóvá tétele az alábbi folyamattal végezhető:



Alkalmazáscsaládok

Alkalmazáscsaládok

- Az alkalmazáscsalád az alkalmazási rendszerek olyan termékcsaládja vagy termékvonulata, amelyek egy közös, szakterület-specifikus architektúrára épülnek *(„közös mag”).*
- Az alkalmazáscsaládnak ezt a közös magját minden esetben újra felhasználják, amikor egy új alkalmazást fejlesztenek ki.
- Mindegyik specifikus alkalmazás különbözik a többitől, miután a közös mag más komponensekkel egészül ki.

Alkalmazáscsaládok

Az alkalmazáscsaládok specializációja

- **Platformspecializáció**
  - Az alkalmazás egyes verziói különböző platformokra készülnek *(pl. Windows, Solaris, Linux, ...)*, de a funkcionalitás azonos.
- **Konfigurációs specializáció**
  - Az alkalmazás egyes verziói különböző perifériákkal képesek együttműködni. *(A perifériákat kezelő komponensek különböznek.)*
- **Funkcionális specializáció**
  - Az alkalmazás egyes verziói eltérő funkcionális követelmények kielégítésére készülnek. *(A funkcionális komponensek különböznek.)*

Alkalmazáscsaládok

Állóeszköz nyilvántartó rendszer – I.

Alkalmazáscsaládok

Példa: Állóeszköz nyilvántartó rendszer – II.

- **Adatbázisszint**
  - A nyilvántartott tárgyak, eszközök részletes adatait, mennyiségét, stb. tárolja és kezeli.
- **I/O leírások szintje**
  - Az adatbázis struktúrájának, az input és output formátumoknak leírásai.
- **Funkcionális szint**
  - A lekérdezések és adatkezelések funkcióit tartalmazza.
- **Interfészek szintje**
  - Felhasználói és programinterfészek

Alkalmazáscsaládok

Alkalmazáscsaládok architektúrái

- Az alkalmazáscsalád specializálása az alábbi megfontolásokkal biztosítható:
  - Az architektúrát úgy kell strukturálni, hogy jól különválasztható és módosítható alrendszerekből álljon.
  - Az architektúrában a kezelendő egyedeket és azok leírását szintén külön kell választani. Az egyedekhez csak magasabb szinten, a leírásokon keresztül szabad hozzáférést biztosítani.
- A fentiekkel megoldható, hogy az alkalmazás speciális alkalmazási igényeknek megfelelő kialakításához csak jól körülhatárolt komponensek cseréjére legyen szükség.

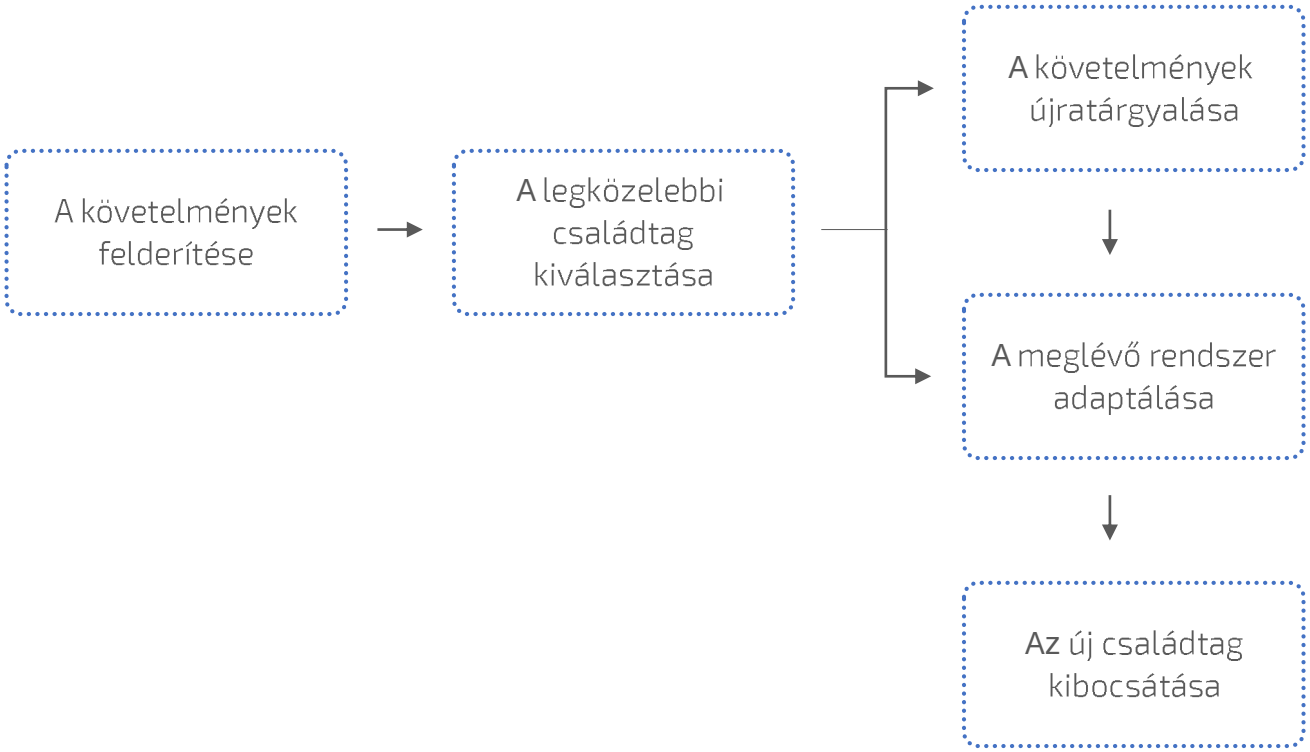
Alkalmazáscsaládok

Alkalmazáscsaládok tagjainak fejlesztések

- **A követelmények felderítése**
  - A család meglévő tagjai prototípusként használhatók
- **Kiindulásként a követelményekhez legközelebb álló családtag kiválasztása**
  - Meg kell találni az adott követelményekhez legközelebb lévő, már létező családtagot.
- **A követelmények újratárgyalása**
  - A követelményeket a meglévő rendszerhez minél közelebb kell hozni.
- **A meglévő rendszer adaptálása**
  - Új modulok kifejlesztése, változtatások a kiválasztott meglévő családtag moduljain.
- **Az új családtag kibocsátása**
  - Az új családtag dokumentálása, a bevezetést, telepítést támogató funkciók megtervezése.

Alkalmazáscsaládok

Az új családtag fejlesztésének folyamata



Köszönöm a figyelmet!

Az előadásról

Az előadás a Pázmány Péter Katolikus Egyetem Információs Technológiai és Bionikai **Karán meghirdetett A szoftvertechnológia** alapjai **című tárgy tananyagát mutatja be**.

Kada Zsolt a GIRO Zrt. informatikai ügyvezető igazgatója. Mérnöki képesítéseit a Torinói Műszaki Egyetemen és a Pázmány Péter Katolikus Egyetemen szerezte. Pályafutását Torinóban kutató fejlesztőként kezdte a Telecom Italia és a Politecnico di Torino közös projektjein. A pénzügyi szférában dolgozott mind banki (Erste Bank), mind beszállítói oldalon (IND). A közigazgatásban a Közigazgatási és Elektronikus Közszolgáltatások Központi Hivatalának (KEKKH) IT fejlesztési főosztályát vezette.

Kapcsolódó források

- **Vető István, A szoftvertechnológia alapjai diasor**
- Ian Sommerville, Szoftverrendszerek **fejlesztése**
  - 17. fejezet, Gyors szoftverfejlesztés
  - 18. fejezet, Szoftver-újrafelhasználás