# SOA összefoglaló

## *1. Szolgáltatásorientált architektúra*

### *Vállalati IT*

Üzleti elvárások jellemzői: üzleti agilitás (rugalmasság), szervezeti határokat átívelő megoldások, hatékonyabb működés, monitorozás
IT környezet jellemzői: eltérő interfészek, -platformok, -komm. protokollok, rendelkezésre állás, gyakori változások
IT szükségletek: újrafelhasználás, folyamat konzisztencia elősegítése, szabványos integráció, változások gyors implementációja, jó és központosított menedzsment, függőségek csökkentése, rendszerfelügyelet, SLA betartás

### *SOA*

Szolgáltatás orientált architektúra: sok definíció, mindenkinek mást jelent:

- Az üzlet szempontjából: az üzletet, illetve az ügymenetet szolgáltatások halmazként tudjuk leképezni, s könnyen elérhetővé tesszük ügyfeleink és partnereink számára.
- Architektúra szempontjából: egy architektúrális stílus, amelynek fő jellemzője a laza csatolás, az újrafelhasználás, az egyszerű és összetett szolgáltatások kombinációja. 3 fő eleme: szolgáltató, igénybevevő és a leíró.
- Megvalósítás szempontjából: fejlesztési modell, mely szabványokra épül, megfelelő módszertani és eszköztámogatással bír és korszerű technológiákat alkalmaz.
- Üzemeltetés szempontjából: szolgáltatás szint megállapodások (SLA-k), melyek biztosítják az igénybevevő és a szolgáltató közötti üzleti prioritásoknak megfelelő QoS betartását.

A szolgáltatásorientált architektúra újrafelhasználható komponensekre, modulokra, szolgáltatásokra támaszkodik, amelyek az alkalmazásoktól és a futtató platformoktól is függetlenül meghívhatók. Szolgáltatás: olyan ismételhető funkció, amely meghívásakor elvégez valamilyen meghatározott tevékenységet (pl. operációs rendszer funkció, saját fejlesztésű üzleti logika/művelet vagy „dobozos" alkalmazás egy modulja, stb.). A SOA négy fő jellemzője:

- Komponens alapú: szabványos szolgáltatási interfészekre támaszkodik az alkalmazások és az erőforrások felé
- Együttműködő: könnyű információ cserét tesz lehetővé az alkalmazások és az erőforrások között
- Moduláris: építőkocka-elvű felépítést az üzleti folyamatok és az infrastruktúra terén egyaránt
- Skálázható: fokozatos kiterjesztés.

Kapcsolódó fogalmak:

- Implementáció: maga a kód, egy-vagy több komponens, pl. JEE alkalmazás
- Interfész: metódus paraméterekkel, egy komponens több interfészt is implementálhat, protokoll-független
- Végpont: interfészt protokollhoz, fizikai címhez köt

SOA alapelvek:

1. Újrafelhasználás (ktsg és időmegtakarítás, redundancia kiküszöbölése)
2. Szolgáltatások zártsága (impl részletek elrejtése)
3. Laza csatolás (dimenziók: hely, platform, nyelv, formátum, protokoll, idő)
4. Erős kohézió (egy szolgáltatás metódusai között, ami tartósságot, jó újrafelhasználhatóságot biztosít)
5. Szolgáltatás-granularitás (egyszerű szolgáltatások (metódusok) vs sok szolgáltatást hívó összetett szolgáltatások)

SOA fogalmak:

- Szolgáltatás: újrahasznosítható, elemi üzleti funkció
- Szolgáltatás orientáció: az üzleti folyamatok elemi szolgáltatásokból történő felépítése
- Szolgáltatás orientált architektúra: olyan IT architektúrális modell, mely támogatja a szolgáltatás orientációt
- Kompozit alkalmazás: elemi szolgáltatásokból építkező összetett alkalmazás

SOA absztrakciós rétegek:

1. Vállalati/szervezeti szint
2. Folyamat szint
3. Szolgáltatás szint
4. Komponens szint
5. Objektum szint

fel: absztrakció iránya, le: újrafelhasználás iránya

SOA rétegei:

1. Fogyasztók
2. Üzleti folyamatok (kompozíció, koreografálás, üzleti állapotgépek)
3. Szolgáltatások (elemi és összetett)
4. Szolgáltatás komponensek
5. Háttérrendszerek

Nem funkcionális követelmények IT megoldásokkal szemben:

- Üzleti követelmények: működési intervallumok, jogi követelmények, ipari szabványok
- Technikai adottságok: környezeti adottságok, technikai modell, korlátozott hozzáférés, szakértelem hiánya
- Minőségi elvárások: teljesítmény, skálázhatóság, tranzakcionális integritás, biztonsági követelmények
- Nem futásidejű elvárások: szolgáltatásmenedszment, verziókezelés, disaster-recovery

# 2. XML webszolgáltatások

XML: szabvány, amely szintaxist definiál jelölő nyelvek készítéséhez jól-formált, ha … érvényes: ha megfelel egy előre definiált sémának, sémák pl: DTD, XDR, XSD. XML feldolgozás: XSLT, Document streaming (pull parsing, push parsing), DOM, XML binding (objektum-XML leképezés)
XML webszolgáltatás: olyan hálózati alkalmazás vagy komponens, amely a SOAP protokoll és a WSDL nyelv felhasználásával, XML dokumentumok formájában kommunikál a külvilággal Jellemzők:platformfüggetlen, laza csatolás, önleíró, lehet szinkron/aszinkron. A webszolgáltatások szabványai:

- XML: adatreprezentációs réteg
- SOAP: szállítási réteg (XML alapú kommunikációs protokoll, header + body) SOAP = XML + HTTP
- WSDL: leírja, hogy az adott webszolgáltatás milyen műveleteket kínál fel, definiálja a típusokat és műveletek meghívásának módját. WSDL felépítése:
  - típusok: adattípusok definiálása XSD segítségével
  - üzenetek: egy művelet által használt adatokat írja le (paraméterek, visszatérési érték)
  - port típusok: a műveleteket tartalmazza (one-way, notification, request-response, solicit-response)
  - kötések: megadja, hogy az egyes műveletek milyen konkrét protokollon érhetőek el
  - elérési pontok: megadja, hol érhető el a szolgáltatás a binding-ban megadott protokollon
- UDDI: szolgáltatások megkeresésére. Egy directory szolgáltatás, amely webszolgáltatásokról tárol információt és SOAP-ot használ kommunikációra, az adatokat elosztott adatbázisban replikálva tárolja, egy alkalmazás akár futásidőben is kereshet benne webszolgáltatást (API-t nyújt).

# 3. WS-* szabványok

A webszolgáltatások olyan aspektusaira adnak specifikációt, amelyeket az alap szabványok nem tárgyalnak. *Kivétel* ez alól a WS-I, amely a meglévő szabványok használatát pontosítja az interoperabilitás érdekében.

### *WS-I*

Új szabványokat a WS-I nem definiál, csak a meglévőkhöz ad best practice-eket és teszt eszközöket. Tisztázza a nem egyértelmű kérdéseket és ezeket Profileok formájában rögzíti. Érdemes alkalmazni az együttműködés érdekében. Fontosabb Profile-ok:

- Basic Profile: alapvető szabványok (SOAP, stb) együttes használatára ad javaslatot, amivel biztosítható az eltérő rendszerek együttműködése (pl. WSDL style attribútumok használatának tisztázása)
- Attachments Profile
- Simple SOAP Binding Profile
- Basic Security Profile: WS-Security használatához ad best practice-eket

### *WS-Security*

Üzenet szintű biztonság előnyösebb a transzport szintűvel szemben, mert biztonságos több szolgáltató esetén is, az üzenettitkossága, integritása biztosítható nem megbízható csatornák és résztvevők esetén is, az aláírás alkalmazás szinten is látszik, tetszőleges része aláírható az üzenetnek stb. Az üzenetszintű biztonság megteremtésének eszköze WS-Security. Definiálja, hogy hogyan kell a titkosítást és aláírásokat használni biztonságos SOAP üzenetben (ehhez felhasznál két W3C szabványt: XML Digital Signature, XML Encryption, amelyek azt adják meg, hogy az XML-ben hogyan kell aláírást és titkosítást használni). WS állapotmentessége miatt minden üzenetben biztonsági token utazik, ez lehet: Username Token (usr /pw), bináris (valamibe kódolva), XML token (biztonsági tartományok közötti autentikációra), EncryptedData Token (titkosított)

### *WS-Addressing*

A webszolgáltatások végpontjainak és a köztük váltott üzenetek azonosítására ad megoldást, a szállítási protokolltól függetlenül. 2 fő rész:

- Végpont referenciák: A fizikai címtől független címzési paramétereket tartalmazza
- Üzenet információs fejlécek: To, From, Reply To, stb.

Tetszőleges Message Exchange Pattern kialakítható vele, csak one-way WSDL műveleteket használva (míg a WSDL csak 4-félét definiál)

### *WS-ReliableMessaging*
Feladata a transzportrétegben bekövetkezett hibák orvosolása alkalmazás szinten, a transzportrétegtől függetlenül. Pl: üzenetkimaradás, üzenetsokszorozódás, üzenetek sorrendcseréje. 4 WS Policy elemet definiál: AtMostOnce (pontosan egyszer küldik el), AtLeastOnce (legalább egyszer megérkezik), ExactlyOnce (pontosan egyszer érkezik meg), InOrder (az előző 3 bármelyikével kombinálható). Résztvevők:

- Megbízható forrás: kezdeményezi a kommunikációt, ő jelzi a végét is, megfelelő mezőket helyez el a SOAP header-ben, ha kell, újraküld
- Megbízható fogadó: válaszol (kezdeményezésre és lezárásra), visszajelez a kapott üzenetről, duplikátumokat eldobja, esetleg késlelteti az üzeneteket a sorrendhelyesség érdekében.

A működés alapja a szekvencia: ez egy egyirányú csatorna, egyedi azonosítóval. A küldő hozza létre (CreatreSequence üzenet), a fogadó erre CreateSequenceResponse-szal válaszol). Üzenetekben sequence id és MessageNumber megy azonosítás célából, a fogadó SequenceAck elemekkel nyugtáz. Lezárása: CloseSequence majd TerminateSequence üzenettel. Hibakezelés: az üzenet tartalmazza a hibakódot, az okot és a részletes leírást (pl.CreateSequenceRefused, SequenceFault, stb.).

### *WS-Trust*
WS-Security által nem tárgyalt feladatokkal foglalkozik pl.: biztonsági tokenek kiadása, megújítása, validálása, érvénytelenítése. A WS-Secure Conversation használja fel.

### *WS-Secure Conversation*
Feladatai: biztonsági kontextus megosztása és session kulcs meghatározása (szimetrikus titkosítás). Egy biztonsági session-nek felel a Security Context Token (SCT) felel meg, amely háromféle módon hozható létre:

1. 3. fél Security Token Service-től kérhető
2. az egyik fél határozza meg
3. a két fél közösen egyezik meg

### *WS-Coordination*
Framework-öt biztosít különálló szolgáltatások együttműködésére, a műveletek és a résztvevők csoportosításával. Alapja a koordinációs kontextus és a koordinátor (aki kordinál: aktiválás, regisztrálás, koordinációs protokoll).

### *WS-AtomicTransaction*
Alapvetően a 2 fázisú kommitot írja le WS üzeneteken keresztül a WS-Coordination-t felhasználva. Koordinációs protokollok:

- Completion: kezdeményezi a koordinációt, tranzakció befejezését
- 2 fázisú kommit: lehet volatile 2PC (nem perzisztens komponensek pl. cache), durable 2PC (adatbázisok)

### *WS-Transactions*
3 szabvány együttes neve:

- WS-Coordination
- WS-AtomicTransaction
- WS-BusinessActivity: a hosszan tartó műveleteket támogatja, visszagördítés helyett kompenzációval (ellentétes művelet végrehajtásával). Két protokollt definiál: BusinessAgreementWithParticipantCompletion és BusinessAgreementWithCoordinatorCompletion, előbbi esetben a résztvevő tudja, mikor fejeződik be a művelete, utóbbiban a koordinátorra hagyatkoznak.

### *WS-Policy*
Szolgáltatások meghívására vonatkozó szabályokat lehet vele deklarálni. A segítségével a WS-* szabványok által támogatott middleware szolgáltatásokat közzétehetjük a WSDL-ben, így ezek tekintetében is önleíró lesz a webszolgáltatás. Alapeleme az assertion. pl: Messaging assertions, Security assertions, Reliable Messaging assertions. Összekapcsolhatóak és egymásba ágyazhatók, ekkor definiálni kell, hogy mit kell teljesíteni (pl. minden gyermeket, csak egyet, legalább egyet). WS-PolicyAttachment: A Policy-t rendeli a Web Service entitáshoz, ez történhet WSDL-ben vagy UDDI-ban.

## *4. Biztonsági megfontolások SOA környezetben*

Biztonsági követelmények: azonosítás, autorizáció, autentikáció, bizalmasság, titoktartás, integritás, auditálhatóság, letagadhatatlanság, bizalom, federáció stb.
Nagyvállalati biztonság: általában többrétegű:

1. Külső réteg: tűzfalak, VPN, antivírus
2. Kontroll réteg: ki léphet be? mihez férhet hozzá?
3. Audit réteg: biztonsági szabványok előírásai, jelentések, kockázatelemezés

Védendő objektumok:

- erőforrás objektumok (fájl, szolgáltatás, weboldal, nyomtató)
- hierarchikus tér
- konténer objektumok (védett objektumok csoportosításai)

Autorizációs követelmények kikényszerítése történhet:

- Hozzáférésvezérlési listákkal (ACL): egy adott objektumhoz rendelve megadja, milyen műveleteket végezhetnek rajta el az egyes felhasználók, csoportok
- Védett objektumok házirendjeivel (POP): környezeti feltételektől függő hozzáférésvédelem (pl. időpont)
- Autorizációs szabályok megadásával (AuthRules): összetett feltételek, a konkrét kérés paramétereinek figyelembevételével

Jogosultság életciklusa:

1. alkalmazott felvétele a beosztásba,
2. jogosultság változása (állandó vagy egyedi jelleggel),
3. jogosultság törlése

SOA biztonság jellegzetességek:

- nem elég a külső védelem (firewall pl.)
- alkalmazások összekötése MOM-on keresztül (így nem csak hagyományos bejelentkezés van, hanem rendszerek közti is)
- Problémák: autentikáció a MOM felé, MOM üzenetek titkosítása hálózati szinten, biztonsági hitelesítők átadása.
- Kihívások: szolgáltatások dinamikus kiválasztása, platform és nyelv független megoldások kellenek, laza csatolás csak akkor teljes, ha a biztonságra is igaz.
- Megvalósítása: WS-* szabványok használhatóak az egyes szolgáltatások biztosítására (WS-Security, WS-Trust, Ws-SecureComversation). Ezek főleg az autentikációval foglalkoznak, autorizációra a XACML használható, auditra meg saját loggolás. Lazán csatolható biztonsági szolgáltatások kialakítására is szükség van.

## Federáció

Ugyanaz a felhasználó több rendszerben különböző identitásokkal rendelkezhet (ezeket a rendszereket ráadásul mások is üzemeltethetik). Cél: cross-domain single sign-on.
Identitás federáció: paradigma olyan technológiákat, szabványokat és használati eseteket ír le, melyek lehetővé teszik a felhasználói identitás biztonságos hordozhatóságát az önálló biztonsági rendszerekkel védett üzleti rendszerek között. Megkülönbözteti a Service Provider-t (SP) és Identity Provider-t (IdP), ez utóbbi végzi az autentikációt, másik nyújtja a szolgáltatást. SP-hez tartozó lokális account és az IdP által tárolt összekapcsolására több megoldás létezik (attribútum, fix azonosító, pszeudoním azononosító, stb.)
Egy federációs szabvány: SAML.

- SAML forgatókönyv a federációra:
    1. hozzáférés kérése SP-nél
    2. SP hitelesítési kérést állít ki és átirányít egy IdP-hez
    3. IdP hitelesít
    4. IdP assertiont ad
    5. IdP visszairányítja a usert az SP-hez
    6. SP elfogad
    7. SP lokális munkamenetet indít az assertion alapján
- SAML Assertion: a hiteletsítés tényét igazolja, 3 részből áll: Authentication Statement (aznosítás ténye) + Attribute Statement (ki azonosította magát) + Authorization Decision Statement (azonosítás következményei)
- SAML logout:
    1. LogoutRequest vizsgálata
    2. Munkamenetért felelős entitások értesítése
    3. Minden entitásnak küldeni kell egy kijelentkeztetési kérést
    4. Aktuális session megszüntetése
    5. Ha siker -> LogoutResponse küldése sikeres státusszal
    6. Ha nem -> státusz: PartialLogout

Nem funkcionális követelmények a biztonságban:

- Teljesítménykérdések: HW gyorsítók, szimm titkosítás, amit lehet transzport szinten
- Menedzsment: házirendje kiépítése, monitoringja

# 5. SOA Inventory minták

### *SOA tervezési minták*
A klasszikus OO tervezési mintákhoz hasonlóan gyakran előforduló problémákra adnak újrahasznosítható megoldást. A SOA egy architekturális paradigma így a SOA tervezési minták is architekturális szintűek. A megoldásra gyakran könnyű rájönni. Gyakran az alkalmazás és a hatások összegyűjtése az igazán hasznos.

## Alapvető Inventory minták

Nem az egyes szolgáltatásokkal foglalkoznak, hanem azzal, hogyan kezeljük, szervezzük szolgáltatásoknak egy összetartozó halmazát.

### *Enterprise Inventory*
Probléma: a vállalaton belül különböző fejlesztési projektek állítanak elő szolgáltatásokat. Az eltérő célok, platformok és prioritások miatt olyan architekturális különbségek alakulhatnak ki az egyes projektek között, amelyek meggátolják a projekthatárokon átível funkcionalitások megvalósítását (elszigetelt silók alakulnak ki).
Megoldás: legyen a vállalaton belül egyetlen szolgáltatás inventory (leltár) és minden projekt ugyanazon előírt standardeket követve, ugyanazon infrastruktúrát igénybe véve fejlesszen szolgáltatásokat
Alkalmazás: ahol javasolt: kis/közepes szervezet (elegendő erőforrással), közepes/nagy szervezet (erősen kontrollált IT-val), új szervezet (korábbi alkalmazások nélkül)
Hatások: nagy mennyiségű tervezés a szolgáltatásjelöltek azonosítására és modellezésére

### *Domain Inventory*
Probléma: különféle (főleg szervezeti) okok akadályozzák az egységes szolgáltatás inventory kialakítását.
Megoldás: több szolgáltatás inventory kialakítása a vállalaton belül, jól definiált (pl. földrajzi/szervezeti) területek mentén. Az egyes inventorykon belül más szabályozást és standardizálást írhatunk elő a szolgáltatásokra
Alkalmazás: ajánlott, ha a vállalaton belül nincs egységes adatmodell, és nem is reális ennek kialakítása vagy, ha több IT osztály van a vállalaton belül, amelyet nem lehet centralizálni
Hatások: az enterprise inventory előnyei csak kisebb hatókörben érvényesülnek, megjelenhetnek redundáns szolgáltatások

### *Service Normalization*
Probléma: funkcionalitásukban részben átfedő szolgáltatások, melyek eltérő fejlődésük miatt ugyanazt más módon valósítják meg.
Megoldás: az inventory szolgáltatásait együttesen modellezve alakítsuk ki a konkrét szolgáltatásokat, átfedések elkerülésével
Hatás: hosszabb analízis a szolgáltatások fejlesztése előtt, a normalizáltság fenntartása később is többlet governance erőfeszítést kíván

### *Logic Centralization*
Probléma: az egyes projektek nem használják föl a meglévő szolgáltatásokat, így az új, redundáns részeket tartalmazó szolgáltatások denormalizálják az inventoryt
Megoldás: a governance írja elő és kényszerítse ki a szolgáltatások „hivatalos végpontjának” használatát

### *Canonical Protocol*
Probléma: a különböző protokollokat támogató szolgáltatások együttműködése nem megoldható, vagy csak teljesítmény overheadet jelentő protokoll-konverzióval
Megoldás: egyetlen kanonikus protokollt válasszunk ki, mindenhol azt próbáljuk használni.
Alkalmazás: pl. XML web szolgáltatás (protokoll elemek verziója is legyen egységes: WS-*, WS-I Basic Profile)
Hatások: a kiválasztott protokoll korlátaival együtt kell élnünk és protokoll gyártói támogatásának megszűnése kockázatot jelent

### *Canonical Schema*
Probléma: az eltérő adatmodellekkel dolgozó szolgáltatások közti együttműködést nehezíti az adatmodellek közti transzformáció megvalósítása és futási idejű overheadje
Megoldás: közös adatmodell kialakítása az inventoryn belül
Alkalmazás: web szolgáltatások esetén XSD sémával

### *Service Layers*
Probléma: az inventory belső struktúra nélkül nagy eséllyel fog inkonzisztens, redundáns elemeket tartalmazni

Megoldás: kettő vagy több logikai szolgáltatásréteg kialakítása, hasonló funkcionalitás alapján (Task, Entity, Utility rétegek). Az egyes rétegek fejlesztése, kormányzása különböző teamekre bízható
Hatások: általában iteratív analízis szükséges a rétegek azonosításához, struktúra későbbi átalakítását erősen megnehezíti

#### *Cross-Domain Utility Layer*
Amikor a Utility Layer több inventory-ra közös. Függőséget vezet be az inventory-k között.

## Logikai rétegződési minták

Azt tárgyalják, milyen rétegeket lehet kialakítani (lásd Service Layers):
1. Utility Abstraction (pl. naplózás, értesítés)
2. Entity Abstraction (újrafelhasználható műveletek üzleti entitásokon)
3. Process (Task) Abstraction (konkrét üzleti használati esetek megvalósítása)

A három minta együttes alkalmazása a *Three-Layer Inventory* kombinált tervezési minta

## Inventory központosítási minták

Az inventory bizonyos aspektusainak fizikai központosításával foglalkoznak.

#### *Process Centralization*
Probléma: ha egy komplex üzleti folyamatot a többihez hasonló szolgáltatásként implementálunk, bizonyos funkcionalitásokat (pl. állapotkezelés) nekünk kell újra meg újra implementálni
Megoldás: az üzleti folyamat menedzselését bízzuk folyamatmotorra, amely számos szolgáltatást nyújt.
Hatások: bevezetése hosszú és költséges lehet

#### *Schema Centralization*
Probléma: a szolgáltatások közös adatsémái redundánsan szerepelhetnek a szolgáltatásszerződésekben
Megoldás: fizikailag különítsük el az adatsémákat, a szolgáltatásszerződések csak ezekre hivatkozzanak
Alkalmazás: pl. WSDL, XSD-ket importál
Hatás: a központosított sémától sok szolgáltatás függ

#### *Policy Centralization*
Probléma: a szolgáltatáshoz tartozó tranzakciós, biztonsági, QoS követelményeket megfogalmazó közös policyk redundáns módon szerepelhetnek
Megoldás: A közös policykat egy helyen tároljuk, a szolgáltatások ezekre csak hivatkoznak
Alkalmazás: WSDL, WS-Policy, WSPolicyAttachments, ESB termékek is támogathatják, de gyakran nem teljes mértékben, gyártóspecifikus megoldásokkal

#### *Rules Centralization*
Probléma: több szolgáltatás által használt üzleti szabályok redundáns implementációja
Megoldás: rule engine-ek alkalmazása. Főbb előnyök: redundancia megszüntetése, lehetőség a szabályok futási ideje módosítására
Hatások: az üzleti logika decentralizáltabb lesz. Lehet, hogy nem integrált a rule engine, hanem külön termék.

## Inventory megvalósítási minták

#### *Dual Protocols*
Probléma: a kanonikus protokoll nem felel meg minden igénynek
Megoldás: alternatív protokoll engedélyezése az elsődleges mellett
Alkalmazás: pl. XML-WS plusz bináris (pl. RMI, Remoting)
Hatások: szükséges lehet protokollkonverzió, nagyobb ráfordítást igényel a konfiguráció menedzsment

#### *State Repository*
Probléma: hosszan futó üzleti folyamatok állapota fölöslegesen van a memóriában a teljes futás alatt
Megoldás: az állapot kimentése
Alkalmazás: a folyamatnak dedikált adatbázis, jellemzően a folyamatot futtató gépen, folyamatmotorok is nyújthatnak támogatást
Hatás: az állapot mentés és visszatöltés overheadjével számolni kell, ennek az esetleges kézi menedzselése plusz fejlesztést igényel

#### *Stateful Services*
Probléma: üzleti folyamatoknak vagy más összetett szolgáltatásoknak nagyméretű állapotinformációt kell kezelniük, amely hosszú ideig a memóriában maradhat
Megoldás: vezessünk be állapottal rendelkező utility szolgáltatásokat, amelyek (részben) átveszik az állapotkezelés feladatát
Alkalmazás: saját szolgáltatás, vagy WSCoordination implementáció
Hatások: az állapotkezelő szolgáltatások skálázhatóságára különösen ügyelni kell

#### *Inventory Endpoint*
Probléma: az inventory több szolgáltatását az inventory határain kívülre is publikálni kell, ami biztonsági, adatvédelmi, interoperabilitási problémákat vet fel
Megoldás: egy kitüntetett szolgáltatás legyen a belépési pont az inventory felé, ami a szükséges mediációs logikát tartalmazza
Alkalmazás: akár többet is bevezethetünk, különféle külső klienscsoportoknak, protokollkonverziót is megvalósíthat
Hatás: az inventoryn belüli változásokat elrejti a külső hívók elől plusz szolgáltatásszerződés menedzselésével jár

### Inventory governance minták

#### *Canonical Expression*
Legyenek a szolgáltatások által felkínált műveletekre elnevezési konvenciók.

#### *Metadata Centralization*
Vezessünk be szolgáltatástárat, hogy a különböző fejlesztő csapatok ne fejlesszenek redundáns szolgáltatásokat

#### *Canonical Versioning*
Vezessünk be a teljes inventoryra érvényes verziózási stratégiát, a verziószám fejezze ki, hogy kompatibilis-e az új verzió a régivel (pl. csak új műveletet adtunk hozzá)

## *6. Szolgáltatástervezési minták*

### Alapvető szolgáltatás minták

#### *Functional Decomposition*
Az üzleti funkcionalitás felosztása kisebb funkcionális egységekre

#### *Service Encapsulation*
Az elszigetelt, silószerű alkalmazások funkcionalitását publikáljuk szolgáltatások formájában

### Szolgáltatás megvalósítási minták

#### *Service Facade*
Probléma: a szolgáltatáslogika és a kliens felé nyújtott szerződés szoros csatolása számos problémát vet fel: a kliensekkel való együttműködés módja befolyásolja a logikát, több szerződés támogatásakor bonyolódik a logika, a logika módosítása a kliensek felé nyújtott szerződés módosítását vonhatja maga után.
Megoldás: a szerződés és a szolgáltatáslogika közé iktassunk be egy facade szolgáltatást
Alkalmazás: nem a „core” üzleti logikát végzi, hanem pl: relay: egyszerű továbbhívás, broker logika: be/kimenet transzformáció, viselkedéskorrekció: a logika megváltozását fedi el meglévő kliensek elől
Hatások: plusz egy réteg költsége fejlesztésben, futási időben (bár ez minimálisra csökkenthető, ha egy gépen van a core logikával)

#### *Redundant Implementation*
Probléma: a más szolgáltatások által intenzíven használt szolgáltatások meghibásodása single point of failure-t jelentenek
Megoldás: a kritikus szolgáltatások több példányban fussanak
Alkalmazás: különböző példányok egyszerre aktívak és különböző klienscsoportokat szolgálnak ki, vagy egy példány aktív és csak hiba esetén veszi át a helyét a backup
Hatások: megnövekedett infrastruktúraigény, governance erőfeszítés

#### *Service Data Replication*
Probléma: a szolgáltatások által használt adatbázis túlterhelődhet, kieshet

Megoldás: a szolgáltatások a megosztott adatbázis replikáit használják
Alkalmazás: read-only szolgáltatásnál elég az egyirányú replikáció, agnosztikus [1] szolgáltatásoknál érdemes megfontolni
Hatások: megnövekedett infrastruktúra, licenszköltségek

#### *Partial Validation*
Probléma: jellemzően agnosztikus szolgáltatások, a jobb újrafelhasználás érdekében sokféle, nagyméretű adatot adnak vissza. Ezek teljes validálása a hívó oldalon fölösleges overhead.
Megoldás: a hívó csak a ténylegesen használt adatokat validálja.

#### *UI Mediator*
Probléma: bonyolultabb szolgáltatások, folyamatok nagyobb válaszideje alatt a felhasználó sokáig nem kap visszajelzést felhasználói felületen
Megoldás: egy közbeiktatott szolgáltatás legyen felelős a folyamatos visszajelzésért
Alkalmazás: a mediator vagy egy szolgáltatás, vagy egy szolgáltatás ügynök (lásd Service Agent minta)
Hatások: plusz egy réteg fejlesztése

## Szolgáltatás biztonsági minták

#### *Exception Shielding*
A kivételek ne tartalmazzanak érzékeny információkat (pl. személyes adatokat, stacktrace-t), ezeket szűrjük ki a válaszból
Alkalmazás: implementáció nem célszerű külön szolgáltatásként, hanem a core logika része, a csomagolt kivételeket is ellenőriznie kell, teljesen infó mehet a logba, a hívó pedig a kivételben tartalmazott ID alapján tud a helpdeskhez fordulni
Hatások: nagyobb feldolgozási idő, de csak kivételeknél

#### *Message screening*
A szolgáltatások tartalmazzanak olyan logikát, amely kiszűri a rosszindulatú üzeneteket
Alkalmazás: üzenet méret ellenőrzése, rosszindulatú tartalom pl. SOAP headerben és bodyban is lehet, titkosított tartalom ellenőrzése csak dekriptálás után lehetséges
Hatás: a kimeríti ellenőrzés sok erőforrást emészthet fel, a nem elég alapos ellenőrzés hamis biztonságérzetet adhat

#### *Trusted Subsystem*
Probléma: a szolgáltatás által használt erőforrásokat (pl. adatbázist) más alkalmazások is elérik, esetleg nem megfelelő jogosultságellenőrzéssel
Megoldás: csak a szolgáltatáson keresztül lehessen elérni az ilyen háttérrendszereket, amely felé a szolgáltatás saját azonosítóval jelentkezzen be.
Hatás: a szolgáltatás válik a támadók főcélpontjává

#### *Service Perimeter Guard*
Külső kliensek által hívható szolgáltatásokat tegyük a demilitarizált zónába
Alkalmazás: Message Screeninggel, Exception Shieldinggel, Brokered Authenticationnel kombinálható
Hatás: a támadások fő célpontja, és teljesítményben szűk keresztmetszet

## Szolgáltatás szerződési minták

#### *Decoupled Contract*
Probléma: ha a szolgáltatásszerződés konkrét komponenstechnológiához kötődik, a szerződésen keresztül végeredményben a kliensek is ahhoz fognak.
Megoldás: A szolgáltatásszerződés fizikailag különüljön el az implementációtól
Alkalmazás: a WSDL-t ne az implementációból generáljuk, mert szoros lesz a csatolás

#### *Contract Centralization*
A szolgáltatáslogikát csak a hivatalos elérési ponton keresztül érjük el, így elkerülve a szoros csatolást
Hatások: teljesítményproblémák, gyorsabb fejlesztés érdekében a fejlesztők hajlamosak megkerülni

#### *Contract Denormalization*
Probléma: a szigorúan normalizált szerződések teljesítmény szempontból nem mindig megfelelőek

[1] az agnosztikus kb. azt jelenti, hogy a hívótól függetlenül működik a szolgáltatás,emiatt újrafelhasználható, stb.

Megoldás: több művelet verziót is kínáljon fel a szolgáltatás, különböző részletezettséggel, akár redundáns funkcionalitással is
Hatás: nagyobb, nehezebben menedzselhető szerződések (+ esetleg adatmodellek)

#### *Concurrent Contracts*
Probléma: egy szolgáltatásszerződésben leírt funkcionalitáshalmaznak csak egy részét akarjuk bizonyos kliensek felé publikálni
Megoldás: több szerződés létrehozása ugyanazon üzleti logikához
Alkalmazás: Facade-del célszerű megoldani
Hatás: több szerződés több governance erőfeszítés

#### *Validation Abstraction*
Probléma: a validáció nagy részét már a WSDL által tartalmazott XSD-ben deklaráljuk és a validációs követelmények változása új szerződésverzióval jár
Megoldás: a validáció várhatóan változékony részét inkább a szolgáltatáslogikában vagy külön szolgáltatásban, vagy Service Agenttel implementáljuk
Hatás: a szerződés nem tartalmazza központosítva a validációs kényszereket, plusz implementáció

## Legacy rendszereket egységbe záró minták

#### *Legacy Wrapper*
Probléma: korábbi rendszert elfedő szolgáltatás szerződésében megjelenik az adott rendszerre specifikus részlet, szoros csatolást eredményezve a régi rendszerhez
Megoldás: a kliensek felé felkínált szerződésből távolítsuk el ezeket a részleteket, akár egy új wrapper szolgáltatás bevezetésével
Alkalmazás: kerülendő pl. korrelációs ID-k, hibakódok, audit infók
Hatás: több szerződés, teljes elfedés általában úgysem lehetséges

#### *Multi-Channel Endpoint*
Probléma: több silószerű korábbi rendszer épült ki különféle csatornák kezelésére (pl. telebank, bankfiók, internebank). Mindegyik saját adatreprezentációval, kommunikációs megoldásokkal.
Megoldás: egységes szolgáltatás, amely minden csatornáról megszólítható
Alkalmazás: pl. SOAP headerben lehet csatornaspecifikus információkat átadni, gyakran ESB, BPM termékekre építve a komplex routing/workflow/broker logika miatt
Hatás: szűk keresztmetszet lehet, új termékek (pl. ESB) beszerzését vonhatja maga után

#### *File Gateway*
Probléma: korábbi rendszerek nagy mennyiségű adatot fájlok formájában bocsátanak ki (pl. CSV, CWF). A szolgáltatásoknak kell ezeket parzolni, pollozni
Megoldás: fedjük el a fájlkezelés részleteit egy gateway logikával
Alkalmazás: a gateway általában egy utility komponens, amely tipikus szolgáltatásai:

- polling, konfigurálható gyakorisággal
- parzolás, az egyszerre beolvasható adatmennyiség konfigurálható
- adatok transzformációja
- a szolgáltatás meghívása, adatok átadása
- fájlok törlése, archiválása

## Szolgálatás governance minták

#### *Compatible Change*
Probléma: egy már publikált szolgáltatásszerződés módosítása a meglévő kliensek módosítását követelheti meg
Megoldás: visszafelé kompatibilis módosításokat hajtsuk végre a szerződésen
Alkalmazás: új művelet a WSDL-ben, új port típus hozzáadása, opcionális policy Assertion hozzáadása, stb
Hatás: lehet, hogy csak szerződés duplikálással alkalmazható

#### *Version Identification*
A szolgáltatásszerződés tartalmazzon verzió információt.
Alkalmazás: célszerű, ha utal a kompatibilitásra, nem kompatibilis verzióváltásnál akár névterekbe is bekerülhet a verziószám
Hatás: egységesíteni kell a verziózási konvenciókat

#### *Termination Notification*
Probléma: egy szolgáltatás előre tervezett visszavonásáról maga a szolgáltatás értesítse a hozzáforduló klienseket
Alkalmazás: programatikusan ellenőrizhető, teljes szolgáltatás vagy egyes műveletek visszavonása
Hatás: nincs rá szabvány a governance tudja kikényszeríteni

#### *Service Decomposition*
Nagyra nőtt szolgáltatások szétbontása kisebbekre.
Hatás: Mivel a már használt szolgáltatások esetén a kliensek módosításával jár, célszerű már a modellezési fázisban gondolni rá

#### *Proxy Capability*
Probléma: a Service Decomposition érinti a klienseket
Megoldás: az új szolgáltatás mellett maradjon meg a régi is, amely az újat hívja meg
Alkalmazás: Facade-dal kombinálva gyakori, a régit meg kell jelölni eltávolításra
Hatás: denormalizáltság, a proxy nem garantálja ugyanazt a működést, megbízhatóságot és teljesítményt, mint a régi szolgáltatás

#### *Decomposed Capability*
Probléma: a jövőben előreláthatóan nagyobbra növő szolgáltatást nem mindig tudunk már induláskor kisebbekre szedni (pl. mert nincs meg az infrastruktúra). A jövőbeli dekompozíciót gátolhatja egy nem kellően átgondolt interfész.
Megoldás: a későbbi dekompozíciót szem előtt tartva finombabb szemcsézettségű műveleteket vegyünk fel
Hatás: hosszabb előzetes modellezés, nagyobb szolgáltatás, túl finom szemcsézettség negatívan hat a teljesítményre, tényleges dekompozíciónál szükség lehet a Proxy Capability-re.

#### *Distributed Capability*
Nagy terhelésre számító műveletek implementációját fizikailag különítsük el, pl. Service Facade alkalmazásával
Hatás: több szerződés, teljes elfedés általában úgysem lehetséges

# *8. Szolgáltatáskompozíciós minták*

## Képességkompozíciós minták

#### *Capability Composition*
Egyszerűbb szolgáltatások segítségével építsünk összetettebb, üzletileg értelmezhető szolgáltatásokat
Hatás: kommunikációs overhead a monolitikus megoldáshoz képest, függőség a szolgáltatások között

#### *Capability Recomposition*
Az agnosztikus szolgáltatásokat több összetett szolgáltatásban is használjuk fel
Hatás: ezen szolgáltatások menedzselése bonyolultabb lesz (teljesítmény, biztonság, verziókezelés), különösen fontos a szolgáltatás felelősének tisztázása

## Messaging alapú tervezési minták

#### *Service Messaging*
Eltérő rendelkezésre állású rendszerek közti kommunikációra használjunk aszinkron üzenetkezelő rendszereket (MOM: Message Oriented Middleware), amely biztosítja az üzenetek biztonságos, tranzakcionális kézbesítését. A többi messaging alapú tervezési minta erre épül.

#### *Messaging Metadata*
Probléma: szinkron kommunikációnál a perzisztens kapcsolathoz állapot, kontextus rendelhető. Messaging esetén ez nem lehetséges.
Megoldás: az állapotot az üzenetben adjuk át.
Alkalmazás: üzenet fejlécében elhelyezhető propertyk, vagy WS-* szabványok által definiált SOAP hederekben
Hatás: kisebb memóriaigény a kapcsolat hiánya miatt, de hosszabb feldolgozási idő az üzenet parszolása miatt

#### *Service Agent*
Probléma: komplex üzleti folyamatok sok szolgáltatást használnak, ami bonyolultabb és a sok hívás miatt nagyobb kommunikációs overheadet eredményez.
Megoldás: a jól definiálható események hatására aktiválódó funkcionalitást olyan alkalmazások implementálják, amelyek ezekre az eseményekre iratkoznak fel. Így kevesebb explicit szolgáltatáshívásra lesz szükség.

Alkalmazás: a service agentnek nincs szolgáltatásszerződése, csak megszakítja az üzeneteket, tipikusan agnosztikus funkciókra: naplózás, autentikáció, terheléskiegyenlítés
Hatás: plusz egy réteg menedzselése, túlzott használat esetén nehéz követni egy esemény hatását

#### *Intermediate Routing*
Probléma: fejlesztési időben nem tudjuk, hova kell célba juttatni az üzenetet, vagy az eredeti célpont nem érhető el
Megoldás: futási időben, egy router logika határozza meg a célpontot
Alkalmazás: gyakran service agent-tel, routing típusok: content-based routing, load balancing, 1:1 Routing. A routing döntés konfigurálhatósága érdekében célszerű rule engine-t alkalmazni
Hatás: a dinamikusan módosított szabályokat tesztelni kell, teljesítmény overhead

#### *State Messaging*
Probléma: a szolgáltatás állapotot tárol, ami a megnövekedett memóriaigény miatt skálázhatósági problémát jelent
Megoldás: utazzon az állapot az üzenetekben
Alkalmazás: egy- vagy kétirányú lehet (ha a kliens sem tárol állapotot), pl. WS-Addressing egyirányú, kétirányú: saját message headerekkel
Hatás: kétirányú esetben az üzenet elvesztése az állapot elvesztésével jár, érzékeny adatokat titkosítani kell, az állapot nagy méretével nő a szükséges sávszélesség és feldolgozási idő

#### *Service Callback*
Az aszinkron üzenetek tartalmazzák azt a címet, ahová a küldő a korrelációs információt tartalmazó választ várja
Alkalmazás: a szükséges információk megadhatók pl. WS-Addressinggel, gyakran aszinkron üzenetsorba érkezik a válasz (mert a küldő pl. valami máson kezdett dolgozni)
Hatások: nincs azonnali visszajelzés

#### *Service Instance Routing*
Probléma: a válaszüzeneteknek egy specifikus, állapottal rendelkező szolgáltatáspéldányhoz kell visszatalálniuk. Ehhez valamilyen példányazonosítót kell feldolgoznunk a szolgáltatáslogikában.
Megoldás: az üzenetkezelő infrastruktúra tudja értelmezni a példányazonosítókat, automatikusan a megfelelő példány kapja az üzenetet
Alkalmazás: kliens oldalon is támogatás kell, általában WS-Addressing kiterjesztés, BPEL pl. támogatja
Hatás: nagy a kísértés sok állapotot tárolni a szolgáltatásban

#### *Reliable Messaging*
Az üzenetek garantált célba juttatására használjunk alkalmazás szintű protokollt (pl. WS-ReliableMessaging), ha a szállítási réteg nem garantálja.
Hatás: overhead

#### *Event-Driven Messaging*
Probléma: valamilyen esemény bekövetkezését pollozva lekérdezni nem hatékony
Megoldás: az esemény kiváltójánál iratkozzunk fel, kérjünk értesítést
Alkalmazás: MOM vagy ESB támogatásával

## Kompozíció megvalósítási minták

#### *Agnostic Sub-Controller*
Probléma: eredetileg nem-agnosztikus logika egy részéről (jellemzően több entitás szolgáltatást összefogó kompozícióról) kiderül, hogy újrafelhasználásra alkalmas lenne, de addigra már egy nagyobb nem-agnosztikus logikával van egységbe zárva
Megoldás: szervezzük ki az újrafelhasználó logikát egy új, agnosztikus szolgáltatás, vagy az eredeti szolgáltatás egy új képességének formájában
Alkalmazás: ha új képesség: nem tisztán nem-agnosztikus lesz a task layer, ha új szolgáltatás: jellemzően entity layerben
Hatás: megzavarhatja a rétegződési elveket, csak biztosan újrahasználható logika esetén javasolt

#### *Composition Autonomy*
Probléma: az elosztott környezetben futó összetett szolgáltatások kevésbé autonómok
Megoldás: a kompozícióban résztvevő szolgáltatásokat izolált környezetbe telepítjük
Alkalmazás: Redundant Implementation, vagy Service Data Replication bevethető a megvalósítás során
Hatás: az izolált környezet kialakítása költséget jelent

#### *Atomic Service Transaction*
Probléma: különféle hibák miatt félbemaradt műveletek inkonzisztens állapotban hagyhatják az erőforrásokat

Megoldás: tranzakciókezelés
Alkalmazás: több tranzakciós erőforrást érintő elosztott tranzakció kétfázisú kommittal vezényelheti le a tranzakció koordinátora, XML webszolgáltatások esetén a WS-Coordinationre épülő WS-AtomicTransaction szabványosítja
Hatás: nagyobb erőforrásigény

#### *Compensating Service Transaction*
Probléma: hosszan tartó tranzakciók sokáig tartanak fogva erőforrásokat, rontva ezzel a skálázhatóságot
Megoldás: tranzakciók helyett kompenzáló szolgáltatásokat alkalmazzunk, mely visszagördülés esetén az inverz műveletet hajtja végre.
Alkalmazás: pl. a BPEL készen támogatja
Hatás: több művelet a szolgáltatásszerződésekben

### Szolgáltatásinterakciós biztonsági minták

#### *Data Confidentiality*
Probléma: az adatok bizalmasságát biztosítani kell nem megbízható köztes résztvevők esetén is
Megoldás: transzport rétegbeli titkosítás helyett üzenet szinten titkosítsunk
Alkalmazás: WS-Security (XML-Encryption)
Hatás: teljesítmény overhead (főleg aszimm.), szimmetrikus kulcsokat időnként cserélni kell, a titkosítás csak a bizalmasságot garantálja, a hitelességet nem (más is küldhette az üzenetet)

#### *Data Origin Authentication*
Probléma: a közbülső érintett csomópontok miatt nem lehetünk benne biztosak, az eredeti küldő eredeti üzenetét olvassuk-e
Megoldás: digitális aláírással biztosítsuk az üzenetek hitelességét
Alkalmazás: WS-Security (XML-Digital Signature)
Hatás: teljesítmény overhead,

#### *Direct Authentication*
A hívó azonosítása az üzenetben elhelyezett hitelesítővel

#### *Brokered Authentication*
Probléma: single sign-on követelmény, vagy a hívó és a szolgáltatás nem bíznak meg egymásban
Megoldás: külön autentikációs broker bevonása
Alkalmazás:X.509 PKI, Kerberos, WS-Trust, SAML
Hatás: single point of failure, a hitelesítők érvényességi ideje, esetleges visszavonása

### Transzformációs minták

#### *Data Model Transformation*
Canonical Schema hiányában szükséges a szolgáltatások adatmodelljeinek megfeleltetése
Alkalmazás: pl. XSLT, csak végszükség esetén
Hatás: plusz fejlesztés (ha nem is az implementáció, de az XSLT megírása), futási idejű overhead

#### *Data Format Transformation*
Adatformátumok közti transzformációt újrahasznosítható módon valósítsuk meg
Alkalmazás: külön szolgáltatás , a szolgáltatások része, service agent
Hatás: saját implementáció esetén plusz fejlesztés, teljesítmény overhead

#### *Protocol Bridging*
Az eltérő protokollok vagy protokollverziók közti váltást ne pont-pont alapon oldjuk meg, hanem egy broker logikával.
Hatások: lehet, hogy az új protokollon keresztül nem lehetséges az üzenet pontos szemantikájának átvitele, overhead

## *8. Kombinált minták*

#### *Orchestration*
Üzleti folyamatok automatizálása, akár hosszan tartó folyamatok támogatása, humán interakciókkal. Tipikus megvalósítás: WS-BPEL kompatibilis workflow engine.
Orchestration = Process Abstraction + Process Centralization + State Reporsitory + Compensating Service Transaction
Kibővített feature-ök: Atomic Service Transaction, Rules Centralization, Data Model Transformation

### *ESB*
ESB = Asynchronous Queuing + Service Broker + Intermediate Rouing + Data Model Transformation + Data Format Transformtion + Protocol Bridging
Kibővített feature-ök:Reliable Messaging, Policy Centralization, Event-Drive Messaging, Rules Centralization

### *Service Broker*
Service Broker = Data Model Transformation + Data Format Transfrmation + Protocol Bridging

### *Canonical Schema Bus*
ESB-vel túl nagy a kísértés, teljesen eltérő szolgáltatásokat integráni. Ezt próbálja orvosolni.
Canonichal Schema Bus = Enterprise Service Bus + Decouplced Contract + Contract Cetralization + Canocial Schema

### *Official Endpoint*
Official Endpont = Logic Centralization + Contract Centralization

### *Federated Endpoint Layer*
Federated Endpoint Layer = Official Endpoint + Service Normalization + Canonical Protocol + Canonical Schema + Canonical Expression

### *Three-Layer Inventory*
Three-Layer inventory = Utility Abstraction + Entity Abstraction + Process Abstraction

# *9. Enterprise Service Bus (ESB)*

ESB: alkalmazás szintű integráció SOA-ban, elsősorban Architektúrális minta (pattern). Számos módon implementálható: centralizáltan, elosztottan. Szabványokon alapuló integrációt tesz lehetővé lazán csatolt alkalmazások és szolgáltatások között. Az ESB-n belüli mediáció (közvetítés) a szolgáltatáshívások/üzenetek/események intelligens feldolgozását teszi lehetővé. Az alkalmazások végpontjain, illetve a bus infrastruktúrában:

- Transzformáció, szolgáltatás kiválasztás, tartalom-alapú irányítás
- Naplózás, mérés, monitorozás
- Autonóm viselkedés (rendszer és üzleti események detektálása), optimalizáció

ESB feladatai:

- Hívó és szolgáltató összeköttetése (lazán csatolt módon)
- Szolgáltatás virtualizáció: routing (elfedi a szolgáltatót), protocol conversion (elfedi a hívás módját), transformation (elfedi az interfészt)
- Aspektusorientált kapcsolatot tesz lehetővé (biztonság, menedzsment, naplózás, audit, stb.)

ESB képességei:

- Kommunikáció (címzés, protokollok, szinkron/aszinkron)
- Integráció (elterjedt háttérrendszerek elérése)
- Biztonság (autentikáció, autorizáció, letagadhatatlanság, bizalmasság, szabványok)
- Üzenetkezelés (transzformációk, tartalom alapú logika)
- QoS (tranzakciókezelés, garantált kézbesítés)
- Szolgáltatásszint (átbocsátóképesség, válaszidő, rendelkezésreállás, skálázhatóság)
- Menedzsment (naplózás, monitorozás, mérés)

ESB:

1. Kommunikációs protokollok és interakciós minták
    - Kapcsolat képessége a hívókkal és szolgáltatókkal (QoS támogatás)
    - Interakciós minták támogatása (pl.:request/reply, onew-way, pub/sub)
    - Tipikus elvárások: HTTP(SOAP/HTTP, XML/HTTP), MQ (sOAP/JMS/MQ, XML/MQ, text/MQ), Adapterek legacy rendszerekhez, WS-I, WS-Security
2. Üzenetmodellek és metamodellek
    - Üzenet modellek: hívó és szolgáltató között áramló adatstruktúrák leírása, kezelése
    - Meta-modellek: üzenetek leírására szolgál, pl. XML Schema, az ESB-k legalább egyet támogatnak
    - Tartalom-modellek írják le a konkrét üzenet típusokat (szintén XML schema), az ESB-k több tartalom-modellt támogatnak (iparág-specifikus és vállalat specifikus modelleket, adattípusok és gyengés típusos modellek)
3. Mediation-flows és mediációs minták
    - a hívó és a szolgáltató között áramló üzenetek feldolgozása (nagy granularitás - összetett mediációs logikák, újrahasznosíthatók, Mediation Pattern-ekből épülnek fel)

- Mediation Pattern-ek alkotják a mediációs folyamatok lépéseit (kisebb granulraitás – elemi lépések, intenzíven újrafelhasználhatók, ESB temékek mediációs primitíveknek nevezik). ESB Mediációs primitívek: Request / Response, Request / Multi Response, Event Propagation , Protocol Switch, Transform, Enrich, Route, Distribute, Monitor, Correlate, Canonical Adapter, Transform – Log – Route, Gateway

### *ESB Aspektusok*

- Szemantikus interfész
  - Üzleti funkciók és az általuk használt adatok jelentése
  - Általában szoros csatolással az implementációban
- Nyelv és platform
  - Szolgáltatások megvalósításához használt programozási nyelv és a futtatást végző környezet
  - Laza csatolás megvalósítása (nyelv- és platformfüggetlen): interfészdefiníciók (WSDL, XSD, IDL), adatreprezentáció (XML), protokollok (SOAP, IIOP, MQ), ESB
- Adatformátum
  - A szolgáltatások által használt struktúrák és formátumok
  - Cél: deklaratív csatolás vagy transzformáció
  - Eszköz: ESB által nyújtott lehetőségek: XSLT, adapterek
  - Fontos kérdés: karakterkódolás, tömörítés
- Protokoll és hely
  - Protokoll és cím, amin keresztül a szolgáltatás elérhető
  - Deklaratív, transzformált vagy egyeztetett csatolással
  - Eszköz: szolgáltatástár a cím lekérésére, ESB
- Idő
  - Rendelkezésre állásbeli különbséget, nem várt leállások esetén se legyen baj
  - Cél: laza csatolással vagy transzformálva
  - Eszköz: aszinkron üzenetkezelő rendszerek (MOM), ESB (eleve tartalmazhat MOM-ot), üzenetek korrelációja megoldandó
- Garantált üzenettovábbítás
  - Deklaratív vagy egyeztetett módon
  - Eszköz: WS-ReliableMessaging, MOM
- Biztonság
  - Autentikáció, autorizáció, bizalmasság, integritás, letagadhatatlanság
  - Deklaratívan: WS-Security
  - Egyeztetve: WS-Policy
  - Transzformációval: köztes komponensekkel
- Verziókezelés
  - Verziók együttélése, megfelelő verziók összerendelése (probléma: nem valószínű, hogy az összes szolgáltatás hívó egy időben tud átállni az új verzióra)
  - Cél: egyeztetett vagy deklaratív csatolás
  - Eszköz: szolgáltatások elnevezési konvenciója, verzió alapú route ESB-ben, opcionális új paraméterek az interfészen
- Állapot
  - Lehet, hogy egy adott szolgáltatás feltételez valamilyen állapotot
  - Cél: deklaratív
  - Eszköz: WS-ResourceFramework, aszinkron üzenetek korrelációja, bejövő üzenetek összekötése futó folyamatokkal

Aspektusok csatolása az alkalmazások között:

- Szoros csatolással: szolgáltatáshívó és szolgáltatás kódjában közvetlenül manipulált (pl. üzleti adatmodell)
- Deklaratív módon: szolgáltatás interfészen megadva, de kódból nem látható (pl. tranzakcikezelés)
- Transzformációval : interfészen megjelenik, kódban manipulált, de módosítása esetén az infrastruktúra követi a változást (pl. adatformátumok)
- Egyeztetve: a hívó és a szolgáltató több lehetőséget kínál fel, az infrastruktúra talál egy közösen használhatót (p. WS-Secutiry algoritmusk)
- Lazán csatolva (szétcsatolt): egyik fél változása sem érinti a másik felet (pl. szolgáltatás helye)

### *Lehetséges ESB Topológiák*

- Egy ESB
  - cél: vállalaton belüli szolgáltatások virtualizációja, aspektusokkal való bővítése
- ESB-k közvetlen összekötése
  - Külön névterek és adminisztrációs tartományok

    - Névtérleképezés szükséges mindkét ESB-ben
    - Ha: mindkét ESB minden szolgáltatását akarjuk integrálni
- ESB-k összekötése közbeiktatott ESB-vel
    - Külön névterek és adminisztrációs tartományok
    - Névtérleképezés a közbülső ESB-ben
    - Ha: a szolgáltatásoknak csak egy részét akarjuk integrálni
- ESB-k összekötése federált ESB-vel
    - Külön névterek és adminisztrációs tartományok
    - Névtérleképezés a közbülső ESB-ben
    - Ha: csak a szolgáltatások egy részét akarjuk integrálni és el akarjuk fedni a több szolgáltatás-implementációt

# *10. Portálok*

Az adat és alkalmazás integráció nem minden esetben lehetséges, illetve néha túl nagy ráfordítást igényelne. Megoldás: felületi integráció: pl. portál rendszerek, IVR.

### *Típusai*

- Általános internet portálok
    - elérhetőség: internet
    - Fókusz: Tartalomcentrikus
    - Profilok: kereső, hírportál, közösségi, személyes portál
    - Domináns elemek: hírek, cikkek, fórumok, blogok, tartalomkereső
- Vállalati portálok
    - elérhetőség: intranet / extranet
    - Fókusz: Funkcióorientált
    - Profilok: B2C (pl. webáruház), B2B (pl. partner portál), B2E (pl. alkalmazotti intranet), Dashboard-ok
    - Domináns elemek: felület-integrált alkalmazások, adatbázisok, dokumentum tár, csoport munka támogatás, integrált kezelőfelület

### *Portál - Single point of interaction*

1. Minden információ egy helyen (csak ezzel az egy hellyel kell kapcsolatba lépni)
2. Bárhonnan, bármikor, bármilyen kliensről
3. Perszonalizáció, személyreszabott felület
4. Aggregáció (egységes felület)
5. Single sign-on (egyszeri bejelentkezés)
6. Keresés
7. Együttműködés

### *Portál szerver*

- Alkalmazás szerverre épülő megoldáscsomag
- Alapszolgáltatások: felület aggregáció, perszonalizáció, kliens-technológiák támogatása, skin-ezés
- További szolgáltatások: dokumentumtár, kollaborációs eszközök, RSS, wiki, blog, fórum, kész, testre szabható modulok, pl. hírek, tőzsde, naptár, leevlezés, to-do lista

## Portlet

UI komponensek, melyek a portál építőelemei . A portletek Java alapú webes komponensek, melyek a portlet container-ben futnak és dinamikus tartalmat állítanak elő. Példák portletek-re: email, ügyfél-alapadatok, ügyfél-portfólió, szabályzattár portlet. Portlet szabvány: JSR-168.

### *JSR-168 által leírt elemek:*

Életciklus:

1. init
2. render
3. processAction
4. destroy

Üzemmódok:

- View: a portlet alapfunkcionalitása
- Edit: beállítások módosítása
- Help: súgó megjelenítése

Ablak-állapotok:

- Normal: normál méret, más portletekkel megoszthatja az oldalfelületet
- Minimized: 0 vagy minimális tartalom
- Maximized: teljes oldalt betöltő megjelenítés

Definiált portlet funkciók, szolgáltatások: portlet perzisztencia, portlet kontextus, security, user profile, session mgmt, lokalizáció, cache-elés, URL generálás

### *Webservices For Remote Portlets*

Webservices For Remote Portlets (WSRP): webszolgáltatás szabványt definiál, mellyel távol futó, felületet generáló szolgáltatások integrálhatóak más felületi alkalmazásokba, illetve portálokba. Szereplők:

- Producer: gyártja/generálja a tartalmat (felületi szolgáltatást nyújtó komponens)
- Consumer: köztes rendszer, mely megjelenítés-orientált szolgáltatásokkal kommunikál a felhasználók nevében

Előnyei: megjelenítés és futtatás helye elkülönül, a szolgáltatás megjelenítést is nyújt, minimális fejlesztést igényel, interoperabilitás (java-.net között), széleskörű ipari támogatás

### *Portletek Granularitása lehet:*

- Komplett alkalmazás - 1 portlet
  - Portlet állapot = alkalmazás állapot, könnyen leképezhető
  - A portlet zárt egységet alkot
  - Korlátozott testreszabhatóság (modulok átrendezése, skinezés, stb az alkalmazásban valósul meg)
  - Korlátozott újrafelhasználás (komponensek csak portleten belül, funkció kiemelése plusz fejlesztést igényel)
- Portletekre osztott alkalmazás
  - Elosztott alkalmazás állapota = több portlet állapota együtt (inkonzisztencia léphet fel az oldalak közti navigáció, félbehagyott műveletek miatt)
  - Portletek között kommunikáció szükséges, akár oldalakon átívelő kommunikáció
  - Jól testreszabható (modulok átrendezése, skinezés, stb. portál szinten valósulhat meg)
  - Támogatja az újrafelhasználást (portlet szinten újrahasznosítható funkciók)

### *Portletek közötti kommunikáció megoldási lehetőségek:*

- Gyártó specifikus kommunikáció
  - üzenetküldés: processAction-ön belül vagy message feldolgozása során
  - üzenet címzése: név / portlet id alapján, előre konfigurálható összeköttetések alapján
  - üzenet tartalmára lehet megkötés: csak string vagy tetszőleges Java objektum
  - gyártó függő megoldás (rontja a hordozhatóságot)
- Közös állapottár / session
  - nem igényel keretrendszeri portlet-kommunikáció támogatást
  - Közös állapottár: gyártófüggetlen, hordozható megoldás (EJB, JDBC, elosztott cache alapú)
  - Közös session: alkalmazás-session scope vagy portál session-je
- URL paraméterek, cookie-k
  - nem szerveren belüli kommunikáció (request-response-on átívelő, csak string)
  - URL paraméterek: ha a keretrendszer nem rejti el, akkor minden portlet látja
  - Cookie: response előállítása során kerül beállításra (következő kliens request-nél olvashatóak ki)

### *Portlet 2.0*

- A portletek közti kommunikációt szabványosítja
- Standard cache-elési mechanizmusok
- AJAX támogatás (…)
- bővíthetőség filterek, listenerek segítségével

### *További integrációs alternatívák*

- Webalkalmazások integrációja (Laza csatolás, Web-clipping , WSRP, Felület portletesítése, Új UI fejlesztése)
- Nagyvállalati ERP rendszerek integrációja: gyakran kész portlet-felületeket szállítanak
- Host alkalmazások: host access megoldások (applet)
- Ablakozós / grafikus / vastag kliens alkalmazások
- Teljes újrafejlesztés
- Felület újrafejlesztése
- Terminal Server jellegű megoldások

### *Kollaborációs eszközök szolgáltatások portálokban*

1. Presence: felhasználói jelenlét / státusz figyelése
2. IM, Instant call: két vagy több résztvevő kommunikációja (a párbeszéd rögzíthető)

3. Web conferencing: két vagy több résztvevő előre egyeztetett online egyeztetése (szöveg, kép (webcam, screen-sharing), hang, whiteboard)
4. Csapathely: csapat / projekt centrikus felület (naptár, fórum, dokumentumok)
5. Blog, wiki szolgáltatások
6. Dokumentumtár: dokumentumok megosztása (verziózás, keresés, kategorizálás)
7. Taggelés (cloud tagging)

# *11. Üzleti folyamatmodellezés (BPM)*

Business Process Management (BPM): a menedzsment és IT határán húzódó tudományterület. Központjában üzleti folyamatok állnak, amelyek olyan ismétlődő tevékenység sorozatok, amik embereket, szervezeti egységeket, alkalmazásokat, dokumentumokat érintenek, pl. hitelbírálat (bank), új előfizetői végpont létesítése (telco). A BPM célja: az üzleti folyamatok fokozatos és folyamatos optimalizációja a: tervezés, megvalósítás, életbe léptetés, kontroll és monitorozás, analízis és visszacsatolás által.

***BPR – BPM összehasonlítása:***

| Business Process Reengineering | Business Process Management |
|---|---|
| menedzsment megközelítés | IT + menedzsment megközelítés |
| a meglévő folyamatot figyelmen kívül hagyja, teljesen újratervezi | a meglévő folyamatra építve, optimalizációs módosításokat és átalakításokat vezet be |
| egyszer, nagy hatékonyságú javulást ígér | iteratív megközelítés, folyamatos javulás |
| elsősorban konzultáció | konzultáció és eszközök |

Rugalmas IT kialakítása újrahasznosítható szolgáltatásokra és folyamatokra alapozva. Lépések:
1. Modellezés: követelményelemzés, modellezés és szimuláció, tervezés
2. Összeállítás: felépítés, tesztelés
3. Telepítés / Futtatás: integráció (személyek, folyamatok, információk)
4. Felügyelet: alkalmazások és szolgáltatások felügyelete, üzleti mérőszámok monitorozása

Egy példa, IBM termék család: Modellezés: WebSphere Business Modeler, Összeállítás: WebSphere Integration Developer, Telepítés: WebSphere Process Server, Felügyelet: WebSphere Business Monitor

## Folyamatmodellezés

Modell: absztrakciós leírása, megragadása egy tárgynak, fogalomnak. Lehet fizikai (repülőgép, autó) vagy koncepcionális modell (pénzügyi modell). A modellezés a tervezés tipikus eszköze (kezelhetővé teszi a komplexitást, csökkenti a kockázatot, egyértelművé teszi a kommunikációt). A folyamat modellezés céljai:
- Meglévő folyamatok dokumentációja, megértése
- Szabályzatok, oktatás
- Szükséges erőforrások meghatározása (emberek, rendszerek, stb.)
- Analízis
- Változások, optimalizáció tervezése
- Meglévő és tervezett új folyamatok tesztelése, analízis
- Problémák azonosítása (pl. szűk keresztmetszetek)
- Implementáció készítése

Eszközei:
- Event-driven Process Chains (EPC)
  - Gráf alapú reprezentáció (elemek pl.:event, function, oranizational unit, resource, fork / join)
- Business Process Modeling Notation (BPMN)
  - Gráf alapú reprezentáció (elem csoportok: Flow objects, Conncting Objects, Swimlanes, Artifacts)
- Unified Modeling Language (UML)
  - Activity diagramok (elemek: initial activity, activity, decision, signal, fork / joinm final activity)
- Business Process Execution Language (BPEL)

Folyamatmodellhez kapcsolódó további modellek (tartalmilag szorosan kapcsolódnak a folyamatmodellhez, de eszköztámogatásuk eltérő):
- Információs / Adat modell (kimeneti/bemeneti adatstruktúrák, constraint-ek)
- Erőforrás modell (humán és gépi erőforrások, rendelkezésre állás, költség)
- Szervezeti modell (szervezeti hierarchia, földrajzi struktúra)
- Üzleti mérőszámok modellje (KPI – üzleti teljesítmény mutatók, mérőszámok, dimenziók)

## Folyamatoptimalizáció

- AS-IS folyamat modellezés: javítási, optimalizációs lehetőségek keresése és szűk keresztmetszetek, erőforrások kihasználtságának elemzése, hatékonyságnövelő változtatások azonosítása.
- WHAT-IF modell: AS-IS modellből kiindulva alternatív folyamatverziók előállítása
- AS-IS és WHAT-IF teljesítmény összehasonlítása (szimuláció és/vagy analízis révén)
- TO-BE modell: optimális változáshalmaz meghatározása

## Folyamatszimuláció

Szimuláció: folyamat teljesítményének előzetes felmérése, statisztikák generálása a folyamat végrehajtásairól, rámutathat potenciális javítási és optimalizációs lehetőségekre
Megismételhető futások: azonos szimulációs paraméterek, Eltérő folyamatverziók összehasonlítása
Szimuláció tipikus paraméterei (fix vagy eloszlással meghatározott):

- Folyamatindítások száma
- Lépések végrehajtási ideje
- Lépések költségvonzata
- Lépésből származó haszon / bevétel
- Elágazási valószínűségek

Folyamat szimuláció lépései:

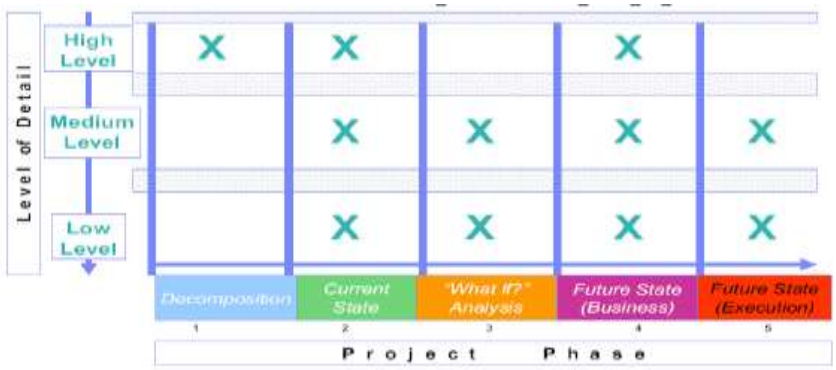
1.Előkészítés (modellezéssel párhuzamosan)
- Szerepkörök és költségek meghatározása
- Lépések végrehajtási idejének megadása
- Erőforrások terhelése és rendelkezésre állása
- Elágazási ágak valószínűségei

2. Szimuláció előkészítése
- Szimulációs paraméterek: szimuláció intervalluma (pl.: 1 hónap / év), folyamat indítások ütemezése, eloszlása, Erőforráspool-ok paraméterei (pl.: hány adatörgzítő dolgozik...)

3. Szimuláció futtatása
- Szimulált folyamatlefutások
- Futási eredmények, statisztikák gyűjtése

4. Analízis
- Statikus analízis: szimuláció nélkül, a modellből önmagában nyerhető infók, pl. tevékenysége lépései, ciklusok, bejárhatatlan utak, erőforráshasználat, stb.
- Dinamikus analízis: a szimulációk eredményén futtatott elemzések és kimutatások, pl. folyamat esetek költség / idő analízise, lépések lefutásának költség / idő analízise, erőforrás terheltség, stb.

## Folyamatoptimalizálás a gyakorlatban – projekt megközelítés




## A modellek szintjei (Level Of Detail)

1. High level:globális kép (térkép), koncepcionális vagy folyamvezérelt.
2. Medium level: részletes struktúra, az analízis céljához vagy a megfelelő kérdések felvetéséhez ad képet.
3. Low level: részletes folyamat minden érintett elemmel, a modell részletes tartalommal van feltöltve, hogy az analízis elvégezhető legyen.

### *Magas szintű modelleírás*

- Cél: összefoglalja és meghatározza a folyamat tulajdonosát, megmutatja a komplexitást, összefüggéseket,

- függőségeket, kijelöli a hatóköröket
- Célközönség: felső vezetés, projektmenedzserek, üzleti elemzők, területi szakértők, architectek

#### *Középső szinű modelleírás*
- Cél: A kritikus pontok feltérképezése és kiosztása további elemzésre, mérőszámok hozzárendelhetők, kiindulást ad a „What if" és „To be"modelleknek
- Célközönség: projektmenedzsment, területi szakértők, üzleti elemzők, architect

#### *Alacsony szintű modelleírás*
- Cél: nézetek előállítása, a hangsúly az értelmes analízis generálásán
- Célközönség: üzleti elemzők, projektmenedzsment, architect, területi szakértők

## Projekt fázisok (Project Phase)

#### *A jelenlegi állapot dekompozíciója*
- Cél: Megegyezés a releváns folyamatok struktúrájáról, Közös nézőpont kialakítása az összes érintett fél számára, Minden, a jelenlegi állapotot érintő analízis és a jövőbeli állapotdefiníció alapjának kialakítása, Komplexitás, kapcsolatok és függőségek feltárása
- Célközönség: Felső vezetés, projektmenedzsment, területi szakértők, üzleti elemzők, architect

#### *Jelenlegi állapot*
- Leírás: a jelenlegi állapot a projekt alapvető fázisa, amely nyújt:
    - egy kiinduló nézőpontot arról, hogy mi történik a jelenlegi ütemben
    - egy alapot minden jövőbeli döntéshez
    - kiindulópontot az implementációhoz
    - referenciát az elérni kívánt jövőbeli állapotok pénzügyi rendszerszintű és összehasonlító elemzéséhez
- Cél: az üzlet mostani állapotának leírása, Minden félreértés kiiktatása, az üzleti folyamatokkal kapcsolatban, További vizsgálandó területek meghatározása, Minden „What if?" analízis alapja
- Célközönség: minden résztvevő

#### *"What if?" analízis*
- A „What if?" analízis forgatókönyveket nyújt a folyamat javítására a jelenlegi állapot modelljéből kiindulva
- Leírás: különböző részletezettségű modellek gyűjteménye, melyek mindegyike bizonyos lehetséges változások kiválasztása esetén írják le az üzleti folyamatot, a modellek részletezettsége és tartalma specifikus is lehet
- Cél: az analízis (folyamat, rendszer, emberi erőforrás, struktúra) elsődleges eszköze, problémák izolálása és a lehetséges megoldások feltérképezése, a különféle változások hatásainak validálása és kvantifikálása
- Célközönség: folyamatelemzők, üzleti elemzők, területi szakértők, folyamattulajdonos és felső vezetés az eredmények áttekintésekor

#### *Jövőbeli állapot*
- Leírás: különböző részletezettsgű modellek gyűjteménye, melyek az üzlet számára legjobb vagy ajánlott megoldást írják le
- Cél: összehasonlító elemzés kifejlesztése a jelenlegi állapot és a kívánt megoldás között, Üzleti követelmények meghatározása, a megoldás tervezésének és megvalósításának alapja
- Célközönség: folyamatelemzők, üzleti elemzők, területi szakértők, folyamattulajdonos és felső vezetés az eredmények áttekintésekor, az architect ez alapján tervezi meg a következő fázist

#### *Modellezés lépései*
A modellezési folyamat minden fázisához 5 típusú tevékenység tartozik:
1. Tervezés (döntések, hatókör)
2. Konstrukció (fejlesztések)
3. Validáció (finomítás, szintaxis, szemantika, értelem, szabványok)
4. Analízis (izoláció, kvantifikáció, szimuláció, riportok)
5. Kommunikácó (nyomtatás, publikálás, eladás)

Ezek a lépések átlapolódhatnak, mert a modellek iteratív módon épülnek, pl. a modell analízise és alkalmazása után kiderülhet, hogy érvénytelenek az eredmények, ilyenkor új iteráció indul

# *12. Workflow rendszerek, BPEL*

## Workflow rendszerek

Workflow (szervezés): megismételhető tevékenységsorozat, mely erőforrásokat, szerepköröket, energiát és információt szervez dokumentálható, megfogható munkafolyamattá .
Process:

- általános process: természetben előforduló, vagy tervezett rendszerek vagy objektumok tulajdonságainak vagy jellemzőinek változásaiból álló sorozat
- mérnöki process: műveletek és kapcsolódó események sorozata, melyek időt, teret és egyéb erőforrásokat igényelnek, valamilyen eredmény elérése érdekében

### *Worklfow Típusok*

- Dokumentum centrikus
    - A folyamat központi eleme egy dokumentum vagy dokumentumkészlet
    - A folyamat életciklusa megegyezik a dokumentum életciklusával
    - Előnyök:
        + gyorsan bevezethető,
        + kész/félkész UI,
        + ideális kis csoportos,
        + dokumentum központú munkaszervezésre
    - Hiányosságok:
        – gyenge control-flow képességek (párhuzamosítás hiánya),
        – tranzakciókezelés hiánya,
        – háttérrendszerek gyenge támogatása,
        – korlátozottan testre szabható UI
    - Példa: kötelező biztosítás kárrendezési folyamata. Központi eleme: kár mappa (image-ek + adat-elemek). Lépései: bejelentés, kárszemle, pótszemle, elismertetés, jóváhagyás, kiutalás
- Alkalmazáscentrikus
    - Adott vállalati alkalmazáscsomaghoz (tipikusan ERP-hez) tartozó beépített workflow motor
    - Az alkalmazáscsomag saját keretrendszerre épül
    - Belső modulok, funkciók gyorsan szervezhetők új folyamatokba
    - Gyakran külső rendszerek felé is átjárhatóságot biztosít
    - Előnyök:
        + gyorsan bevezethető,
        + meglévő ERP alkalmazáson belüli folyamatok rugalmasak,
        + nem igényel külön folyamatmotort
        + olcsó, kész/félkész UI
    - Hiányosságok:
        – gyenge control-flow (párhuzamosítás hiánya),
        – tranzakciókezelés hiánya,
        – nehézkes integráció külső rendszerekkel,
        – korlátozott testre szabhatóság
        – külső folyamatok megvalósítása nehézkes
    - Példa: Siebel (de lehetne SAP is) testre szabható értékesítési folyamat. Főbb lépései: igény észlelés, azonosítás, validálás, ajánlattétel, feltételes megállapodás, szerződés, lezárás. A Workflow-logika: léptetési szabályok, felelősség kiosztása
- Kollaboratív
    - Kevésbé formalizált, emberközpontú workflow,
    - Zömében (ill. kizárólag) humán lépésekből állnak
    - Nagyfokú rugalmasság a folyamat léptetgetésében
    - Gyakran űrlap alapú
    - Előnyök:
        + gyorsan bevezethető,
        + rugalmas (gyorsan kialakíthatók új folyamatok),
        + mező szintű változáskövetés (jó visszakövethetőség),
        + kész/félkész UI
    - Hiányosságok:
        – gyenge control-flow,
        – tranzakciókezelés hiánya,

        - nehézkes integráció külső rendszerekkel,
        - nem kényszerít a szabályok betartására,
        - nehézkes „megkötni” a felhasználó kezét
    - Példa: Eszköz-igénylési folyamat (pl. laptop). Központban az igénylő űrlap. Lépések: igénylő űrlap kitöltése (alkalmazott), manager-i jóváhagyás (közvetlen főnök), ktsg-becslés (beszerzési osztály), pénzügyi jóváhagyás (CFO), beszerzés (beszerzési osztály)
- Tranzakciócentrikus
    - Szigorúan formalizált, tranzakciók (illetve humán lépések) szekvenciája
    - Háttérrendszerek, szolgáltatások integrációjának támogatása
    - Tranzakcionális biztonság
    - Aadattranszformációk
    - Előnyök:
        + külső rendszerek könnyű integrációja,
        + tranzakcióbiztos működés,
        + erős control-flow képességek,
        + robosztus
        + szigorú folyamat betartás
    - Hiányosságok:
        - workflow fejlesztése szakértelmet igényel,
        - merev és formalizált működés,
        - UI gyenge (nem fókusz)
    - Példa: Jelzáloghitel-folyamat. Külső rendszerek, szolgáltatások: banki adóslista, ügyfél portfólió, scoring, ügyfél/számlanyitás, folyósítás, stb. Humán lépések: igénybefogadás, adatrögzítés, jóváhagyás, szerződéskötés, folyósítás jóváhagyása, stb.

***Workflow patternek***

Worklflows Patterns initative: tipikus workflow minták azonosítása, gyakorlati szimulációk, problémák alapján. Összehasonlítási alap, fejlesztési iránytű. Perspektívák:

- Control-flow: tevékenységeket és azok végrehajtási sorrendjét írja le különböző vezérlőszerkezetek révén. A tevékenysége elemi munkaegységek. A Patternek:
    - Basic: sequence, parallel split, sync, exclusive choice, simple merge
    - Branching: multiple choice, sync merge, multi-merge, discriminator (lehet structured, ekkor az első bejövő szálán folytatódik a végrehajtás, ha a többiek is elérnek ide, egyszerűen leállnak, ha mind beért, reset vagy cancelling, ekkor pedig az első beérkező visszavonja a többi beérkező végrehajtását és resetel)
    - Iteration: arbitrary cycles, structured loop
    - Termination: implicit termination (ha nincs több feladat álljon le), explicit termination (bizonyos állapot estén álljon le)
    - Multiple instances: without sync (több példány egy taskból), with a priori design x db előre ismert és szinkronizált, mindenki lefut)with a priori run-time kn (kódban derül ki, mennyi kell, de kiderül), without a priori run-time kn (a taskok futása közben is változhat, hogy hány példány kell).
    - State based: deferred choice (külső beavatkozásra vár), milestone (egyik ág csak akkor mehet tovább, ha a másik valamilyen állapotba ér)
    - Cancellation: cancel task
- Data: Üzleti és feldolgozási adatokkal egészíti ki a control flow-t. Leírja az adatok áramlását a tevékenységek, illetve helyi változók között, valamint a tevékenységek adatszintű elő- és utófeltételeit.
- Resource perspective: szervezeti struktúra, illetve humán és eszköz szerepköreinek hozzárendelése tevékenységekhez
- Operatinal perspective: tevékenységek leképezése elemi lépésekre, a mögöttes alkalmazások megszólítására
- Exception handling perspective: Futás közben bekövetkező hibák, kivételek kezelését írja le, olyan előre látott események, melyek eltérnek a folyamat normál futási ágaitól, de kezelésükre fel kell készülni.

## BPEL

Business Process Execution Language (BPEL) for Web Services: egy nyelv üzleti folyamatok leírására. webszolgáltatásokra épít, webszolgáltatásokat létrehozva (a folyamat maga is webzolgátatásként viselkedik). Fő támogatók: IBM, Microsoft, BEA, SAP, Siebel. Előnyei: iparilag elfogadott folyamatleíró nyelv, platformfüggetlen, hordozható üzleti folyamatok (workflow motorok között). BPEL Tervezési szempontjai:

- Külső rendszer elérése webszolgátatásként,
- XML alapú folyamatleírás (ne legyen grafikus ábrázoláshoz kötve),
- hierarchikus- és gráfalapú vezérlőszerkezet kombinációja,
- Adatmanipulációs funkciók,

- Futó folyamatpéldányok azonosítása, Folyamat inditása/terminálása/felfüggesztése,
- Hosszú lefutású tranzakciós modell,
- Folyamatok megjelenése webszolgátatásként

### *BPEL XML struktúrája*

```
<process ...>
        <partners> ... </partners>
                <!-- Web services the process interacts with -->
        <containers> ... </containers>
                <!- Data used by the process -->
        <correlationSets> ... </correlationSets>
                <!- Used to support asynchronous interactions -->
        <faultHandlers> ... </faultHandlers>
                <!-Alternate execution path to deal with faulty conditions -->
        <compensationHandlers> ... </compensationHandlers>
                <!-Code to execute when "undoing" an action -->
        (activities)*
                <!- What the process actually does -->
</process>
```

### *BPEL szabvány*

- Activities: a folyamatot alkotó tevékenységeket, lépéseket írják le. Basic Activites:
    - Receive: külső üzenet érkezésére vár. Tipikusan új folyamat példányt indít az üzenet érkezésekor
    - Pick: bizonyos típusú üzenet érkezésére vagy timeout-ra vár
    - Reply: válaszüzenet a Receive-re
    - Empty: üres lépés
    - Invoke: partner metódusának meghívása (külső webszolgáltatás hívása)
    - Terminate: folyamatpéldány futásának megszakítása
    - Assign: adatmanipuláció – folyamatváltozók közötti adatmozgást végez
    - Staff: humán lépést hív
    - Wait: adott ideig vagy időpontig vár
    - JavaSnippet: Java kódrészletet hív (ez implementációfüggő kiterjesztés, nem része a szabványnak!)
    - Throw: Fault-ot generál a folyamaton belülről
    - Subprocess: alfolyamat hívása a folyamaton belülről
- Control Links: a folyamat vezérlését írják le
- Structuring Activities/Elements:
    - Flow: párhuzamos blokk, a lépések párhuzamosan hajtódnak végre
    - Link: tevékenységek egymásutániságát jelöli (végrehajtási feltételt tartalmazó kapcsolat)
    - Sequence: szekvenciális blokk (szigorú sorrend szerinti végrehajtás)
    - Switch: többágú elágazás
    - While: elöltesztelő ciklus
    - Scope: speciális structured activity folyamat finomításra
- Variables: a folyamat állapotát reprezentáló példányváltozók
    - lehet bemeneti/kimeneti változója az Invoke, Receive, Reply lépéseknek,
    - lehet a belső folyamat-állapot reprezentálója
    - WSDL üzenet típusok (XSD)
- PartnerLinks: utalás a partnerre (típus def.), nem kell beégetni a végpontokat a modellbe, partnerek megadása assembly/build során
- CorrelationSets: állapottal bíró, hosszú lefutású, aszinkron interakciók támogatása
    - segítségével a bejövő üzenet „odatalál" a megfelelő folyamatpéldányhoz
    - adatmezők halmaza, mely azonosítja a folyamatpéldányt
    - értéke egyszer inicializálódik és az interakció alatt nem változik
- FaultHandlers: exception-kezeléshez hasonló mechanizmus
    - scope-on belüli fault-ok lekezelését a FaultHandler-ek végzik
    - az egyes fault-ok lekezelése eltérő módon történhet, ekkor az eredeti scope blokkjának végrehajtása végleg megszakad és a FaultHandler hívódik meg
    - Az egyes fault típusok eltérő módon kezelendők le, Fault lehet Invoke eredménye és egy folyamaton belül definiált Throw lépés eredménye
    - Ha fault dobódik egy scope-on belül
        - …és van FaultHandler és a Fault-ra van catch definiálva, a scope állapota Failed lesz, a FaultHandler végrehajtódik, külső scope normál módban folytatódik
        - …és van FaultHandler definiálva, de nincs a Fault-ra catch, a scope állapota Failed lesz a, a Fault pedig tovább dobódik a külső scope-hoz
        - …és nincs FaultHandler defniálva, a Fault tovább dobódik a külső scope-hoz

- vagy lekezeli a Fault-ot egy Handler a scope-on belül, vagy tovább dobódik a külső scope-hoz

### *Kompenzáció*

- Probléma: folyamat visszavonása esetén mi a teendő? Hosszú lefutású folyamatnál nincsen rollback, sőt vannak olyan rendszerek, melyeken nem is tudunk kívülről elérni. Erre ad megoldást a kompenzáció, vagyis az inverz műveletek végrehajtása (pl. számlanyitás – számlamegszűntetés). Vannak műveletek, melyeknek összetett a kompenzációja és vannak, amelyeknél nincs értelme kompenzálni. Szabályok:
  - scope-ra adható meg
  - meghívható explicit módon: compensate activity
  - implicit kompenzáció: ha A scope hibát dob, a tartalmazott, már sikeresen lefutott B scope kompenzációja automatikusan meghívódik (a végrehajtási sorrenddel ellentétesen, ha több van)

### *BPEL 2.0*

- Új activity típusok és vezérlő szerkezetek
- Változó inicializálás
- XSLT a változó transzf. leírására
- XPath elérés változókhoz
- Lokálisan deklarált messageExchange
- Abstract folyamatok tisztább specifikációja

### *Humán lépések*

BPEL nincs felkészítve a humán lépések kezelésére (pedig erőteljes az igény rá). Megoldás: újabb szabvány-javaslat:

## BPEL4People

BPEL4People: emberi tevékenységek specifikálása, szerepkörhöz/felelőshöz rendelés, eszkaláció, nomináció, stb.
Humán lépés típusok:

- Inline human task (human task része a BPEL process-nek)
  - People activity-n belül
  - People activity hívja az inline human task-ot
- Standalone human task (ilyenkor nem része a BPEL processnek)
  - People Activity hív ki a Standalone Human task-ba
  - People Activity a Standalone Human task WSDL interfészén keresztül hív

Humán lépések, olyanok, mint a webszolgáltatás hívások. A humán lépések is jellemezhetőek interfésszel (pl. be és kimeneti adatok, aszinkron hívásként kezelhető). Webszolgáltatáskénti kezelés előnyei: humán lépések és szolgáltatáshívások csereszabatosakká válnak valamint más alkalmazásokból is hívhatóak a humán lépések egyszerű szolgáltatásként.

### *Human Task Container*

Feladata a humán lépések támogatása:

- Feladatlisták kezelése
- Életciklus/állapot kezelése
- Feladat-lekérdezések kiszolgálása
- További beavatkozások: pl. eszkaláció, delegáció, feladat-kiosztás

### *Erőforráshoz rendelés*

Alapja tipikusan: valamely user repository. Gyakori kiosztási minták: users, group members, department members, role members, manager of employee, stb.

Eszkaláció: probléma jelzése, kiterjesztése magasabb szintre (pl. főnökhöz továbbítódik), ha a feladatot adott időn belül nem kezdik elvégezni vagy nem végezték el. Módja: új feladat az eszkaláció felelősének, email értesítés, stb. Eszkalációs szintek, hierarchiák alakíthatók ki .

### *Humán lépés határai (mi egy humán lépés?) best prectices::*

- „Amit egy felhasználó egy ültő helyében elvégez”
- „Egy felhasználóhoz rendelhető zárt egységet képező feladat”
- pageflow: felületi alkalmazásban megjelenő oldalak, képernyők sorozata. Gyakori hiba BPEL folyamatba erőltetni a pageflow logikát. pageflow != workflow!!!

Kiterjesztett humán lépés minták:

- alfeladat (a szülő task subtask-okra osztása)
- utómunka (a szülő task után hajtódik végre)
- lekövetés (idő elteltével újra visszatért a végrehajtáshoz a task)

- 4-szem elv (ketten ugyanaz a taskot végzik)
- delegáció (task továbbítása másnak, új task jön létre a végrehajtásra)
- feladatkiosztás (task végrehajtásának továbbadása másnak, de egyetlen taskban)

# *13. Üzleti monitorozás, folyamatok nyomonkövetése*

## Naplózás és monitorozás elosztott (pl. SOA) környezetben

Az ületi események, folyamtok tipikusan számos komponensen, rendszeren haladnak keresztül. A komponensek eltérő tulajdonságokkal rendelkeznek (pl.: EJB, webszolgáltatás, bussines rule, BPEL folyamat, hosszú lefutású vs tranzakcionális, szinkron vs aszinkron, különböző szervereken elosztva, különböző platformokon). Az elosztottság a hibaforrások elosztottságát is jelenti (nem feltétlenül több hibaforrás, de szétszórtan). A folyamatok nyomon követése kritikus követelmény. A loggolás és a monitorozás céljai:

- Auditálhatóság: az események sorrendjének, paramétereinek utólagos áttekintése és visszakövetése. Gyakran törvényi, szabályozási elvárás
- Értesítés,r iasztás: azonnali reagálás kritikus eseményekre
- Korreláció, nyomonkövetés: események összepárosítása, ok-okozat felderítése. Hiba esetén hibaforrásának azonosítása.
- Üzleti teljesítmény kimutatása: üzleti szempontból releváns információk kinyerése (pl. end-to-end lefutási idő, szűk keresztmetszetek, kritikus utak), üzleti eredményesség (profit, kockázat stb.) mérése.

## Üzleti monitorozás

Business Activity Monitoring (BAM): a szervezet üzleti tevékenységeiből származó információk „kvázi-real time" aggregációja, analízise és megjelenítése. A BAM vonatkozhat üzleti folyamatokra (a BPM részeként) és egyéb rendszereken (pl. ESB) átívelő üzletileg releváns rövid lefutású folyamatokra is. a BAM főbb funkciói:

- Key Performance indicator (KPI)-ok megjelenítése: aggregált pénzügyi-és nem pénzügyi teljesítménymutató mérőszámok, melyek a szervezet hatékonyságát, eredményességét tükrözik pl. hitelfolyamat átlagos átfutási ideje, beérkező hiteligénylések száma, kibocsátott hitelek száma, folyósított hitelek összértéke, átlagos kockázati mutató. KPI jellemzői: típus (szám, idő), mérendő paraméter, aggregáció módja (szumma, átlag, stb), időintervallum, dimenzionális szűkítés (régióra, termékre), célérték (pl. 500 mFt / 200 új hiteligénylés), megfelelőségi intervallumok (0-100 – kevés, 100-200 megfelelő, 200-300 kiváló)
- Részletes analízis: az aggregált mérőszámokból kiindulva részletes analízisek által: tendenciák azonosítása, lebontás terület, termék szerint egészen a folyamat példányok, üzleti események szintjéig.
- Egyszerűbb riportok, kimutatások készítése
- Notifikáció: riasztás (SMS, E-mail), korrektív intézkedés bizonyos üzleti szituációk detektálásakor

### *Tipikus felhasználók, használati esetek*

- Üzleti felhasználók (üzleti eredményesség mérése, hatékonyság áttekintése, üzleti analízis)
- Üzleti elemzők, folyamatszervezők (analízis (trendek, tendenciák), szűk keresztmetszetek azonosítása, folyamat optimalizációs lehetőségek felmérése, visszacsatolás a modellező eszközbe).
- Riasztások kezelése (aggregált mérőszámokhoz rendelt riasztások: üzleti elemzők és menedzsment, példányszintű riasztások: üzleti felhasználók és operátorok)

### *Business Dashboard*

Vezérlőpult jellegű áttekintést nyújt az üzleti folyamatokról, tevékenységekről. Tulajdonságok:

- Információt jelenít meg: KPI-ok jellemzőinek megjelenítése, statisztikák és grafikonok megjelenítése
- Dinamikus analízist tesz lehetővé: riport, kimutatás generálás (Drill Down, Roll-Up, mutatószám szerint valamilyen dimenzió mentén), analitikus funkciók (trend elemzés, eloszlás vizsgálat)
- Bizonyos beavatkozásokat is lehetővé tehet: folyamatok felfüggesztése és indítása

### *Adatpiac modell*

Csillag séma: tény tábla és dimenzió táblák, dimenziók hierarchiája, kocka

### *Adattárház és BAM közti különbségek*

| BAM | Adattárház |
|---|---|
| kvázi real time | Általában T+n napos (n = 1,2…) |
| esemény alapú töltés | Batch jellegű töltés |
| az üzleti tevékenységeket önmagukban értelmezi | keresztanalíziseket is lehetővé tesz, eltérő üzleti tevékenységek között |
| egyszerűbb kimutatások, tendenciák azonosítása | összetett kimutatások, vásárlói szokások analízise |

| | |
|---|---|
| általában rövidebb távra vonatkozik | historikus, hosszú távú adatokat tárol |
| új tevékenység monitorozása gyorsabban bevezethető | új adatforrás lassú, project jellegű bevezetése |

## Folyamatok nyomonkövetése

Elosztott rendszereknél különös kihívás a tranzakciók / folyamatok nyomon követése. Kritikussá válik, ha felmerülnek az alábbi feladatok:

- Auditálhatóság (Törvényi előírás, belső szabályzat, „Ki, mikor hol és mit?", archiválandó)
- Tranzakció nyomonkövetés (helpdesk, telecenter, „hol akadt el XY user z tranzakciója?")
- Hibakeresés (Helpdesk, fejlesztők, „Mely komponensben keletkezett a hiba?", az alkalmazás logok nem biztosítanak áttekintést, nem teszik lehetővé a korrelációt.

### *Naplózási szintek, megoldások*

Alkalmazások log ja(debug, trace)

- Célja: fejlesztés, hibakeresés támogatása
- Tipikus szintek:
  - DEBUG: legrészletesebb, a program működésének ok-okozatait is tartalmazza
  - INFO: a program működésének menetét teszi nyomon követhetővé
  - WARNING: potenciálisan problémát jelentő esetek
  - ERROR: hibák, melyek nem akadályozzák az alkalmazás futását
  - FATAL: végzetes hibák, melyek az alkalmazás futását ellehetetlenítik
- Tipikus helye: fájl vagy rendszernapló
- Fejlesztés alatt tipikusan: DEBUG és INFO, éles üzemben WARNING és ERROR

Audit log

- Célja: tranzakció históriku visszakövetése, törvényi megfelelés
- Tipikus helye: kitüntetett fájlrendszer, adatbázis
- Írása: történhet több alkalmazásból 1 centrális logba, de alkalmazásonként saját audit log is működhet
- Tranzakcionális kezelést igényel: az audit loggolás tényleges tranzakcióval együtt commitálódik, rollback-elődik

Integrációs log

- Célja: tranzakció visszakövetése, helpdesk támogatása, monitoring támogatása továbbá ez az audit log inputja
- Tipikus helye: adatbázis
- Írása: több alkalmazásból egy logba
- Tranzakcionális kezelést igényelhet: az integrációs loggolás a tényleges tranzakcióval együtt commitálódik, rollback-elődik

### *Esemény továbbító infrastruktúra*

Common Event Infrastructure CEI: Az IBM implementációja üzleti, rendszerszintű és hálózati események továbbítására és tárolására. Publish / Subscribe jellegű működés: események közzététele és feliratkozás bizonyos eseményekre. A Common Base Event szabványon alapszik.
Common Base Event (CBE): szabványos üzenetformátum üzleti és IT események reprezentációjára. OASIS szabvány. Közös tartalom a CBE-ben: kibocsátó komponens azonosítója, szituáció / esemény, kontextus információk, kiterjesztett adat. A Common Base Event a Common Event infrastructure üzeneteinek formátuma

### *Események korrelációja*

Event Correlation Sphere (ECS): a hívótól kap egy azonosítót, szülő-gyermek kapcsolat, automatikusan továbbterjed az eventekben, ha EJB folyamatot hív, Folyamat aktivitást indít, Aktivitás EJB-t hív EJB java metódust hív.

# *14.SOA Governance, szolgáltatástárak*

### *A SOA ígéretei és sok helyen a valóság…*

| | |
|---|---|
| Újrafelhasználás | ~0 újrafelhasználás |
| Folyamatok gyors, rugalmas implementációja | Szaporodó szolgáltatások – redundáns logikák |
| Nagyobb rugalmasság, könnyebb integráció | Átláthatatlan függőségek, kockázatos változtatások és csökkenő rugalmasság |
| Üzlet és IT jobb együtt működése | |

## Governance

Governance = vezetés, irányítás, kontroll. Definiálja a felelősségeket, a jogosultságokat, az együttműködési és kommunikációs normákat, a megmérettetés és számonkérés mikéntjét, a szabályozás módját, a kontrol mechanizmusokat. Célja a működés összehangolása a stratégiával, üzleti értékteremtés és a kockázatok csökkentése (átláthatóbb, auditálható működés).
Az IT Governance biztosítja, hogy az IT segítse a szervezeti célok elérését, az IT erőforrások rugalmas kihasználását, gyors reagálást az új üzleti lehetőségekre, az IT-vel kapcsolatos kockázatok kezelését. A governance működési szabályokból, előírásokból (process-ekből) és kontroll, illetve audit elemekből (pl. COSO, CoBIT) áll. Fókusza: az IT működése az üzleti célok elérése érdekében: tervezés és szervezés, bevezetés és megvalósítás, működtetés és support továbbá monitoring és értékelés.
A SOA Governance az IT Governance egyfajta kiterjesztése. A feladata a SOA eredményes és hatékony működésének biztosítása. Fókuszában a szolgáltatások életciklusa áll: tervezés, keresés, menedzsment, verziózás, security, stb. Céljai a SOA által írt előnyök teljesítése (nagy fokú rugalmasság és gyors reakció), a költségek csökkentése, jobb kontroll és a munka hatékonyságának növelése.
Jól szervezett SOA kellékei

- Architektúra, keretrendszer (pl. mit hogyan implementál, mi a preferált felhasználói felület, stb.)
- Újrahasznosítható asset-ek (pl. felületi alkalmazás template-jei, házi fejlesztésű tag library-k, stb.)
- Best practice-k (pl. hogy érdemes egy szinkron szolgáltatást aszinkron módon hívni..)
- Implementációs szabályok (pl. keretrendszeri ajánlások betartása)
- Szervezési szabályok (pl. kik a szolgáltatások felelősei)
- SOA-gondozó szervezet (ami kivételes helyzetekben dönt, az architektúrát építgeti, aktualizálja, segíti a projectek helyes útra terelését)

Center of Excellence (COE): Szervezeti határokon átívelő csoport, mely fő feladatai:

- SOA-val kapcsolatos tervezési és beruházási döntések
- SOA Governance szabályok kidolgozása és betartása
- szervezeten belüli szolgáltatásorientált szemlélet és információk terjesztése
- szolgáltatás portfólió készítése
- pilot projektek lebonyolítása, kiértékelése
- projektek ellenőrzése, architektúrális és szervezési előírások betartása
- kollégák mentorálása

Tagjai különböző szervezeti egységek képviselői: architektek, területi szakértők (pl. BPM szakértők, integrációs szakértők, rendszertervezők, üzemeltetés képviselői), IT menedzsment képviselői és az üzleti oldal képviselői. Ritkán van szükség az összes résztvevőre, de közös irányelve és stratégia szükséges. A tagok időnként ellenőrzik a projekteket és segítenek helyes irányba terelni őket (konzultálás, dokumentációkkal segítenek, szükség szerint oktatást szerveznek). Begyűjtik a jó gyakorlatokat, segítenek újrahasznosítható asseteket azonosítani. Rálátnak a szolgáltatás portfólióra, segítenek az új szolgáltatások kidolgozásában.

## Szolgáltatástárak

Szolgátatástárak feladatai:

- szolgáltatás metaadatok tárolása, kezelése
- szolgáltatás metaadatok elérése tervezési időben és runtime környezetben
- szolgáltatáshoz kapcsolódó termékek (pl. WSDL és policy állományok) tárolása
- szolgáltatás taxonómia kezelése (területi besorolás)
- szolgáltatás életciklusának kezelése
- értesítés a változásokról
- szolgáltatás verziók kezelése
- szolgáltatás elérés szabályainak nyilvántartása –policy előírások

***UDDI kontra Service Registry***

| UDDI | Service Registry |
|---|---|
| webszolgáltatáok katalógus jellegű nyilvántartására találták ki | szolgáltatás metaadatok nyilvántartása |
| technikai felhasználók katalógusa | üzleti terminológiával kiegészíthető, taxonómiával kiegészíthető |
| Rugalmatlan, szigorú adatmodell | Rugalmas, testreszabható adatmodell |
| Nincsen életciklus és verziókezelés | Függőségek tárolása, életciklus és verziókezelés |
| A nem webszolgáltatás végpontok kezelése nem támogatott | régi és új szabványok támogatás, UDDI kompatibilitás |

Szolgáltatástár termékek pl.: IBM Websphere Service Registry & Repository, Oracle Service Regitry

### *Szolgáltatás tárak tipikus szolgáltatásai*

- szolgáltatás metaadatok kezelése és tárolása
    - Célja az újrafelhasználás elősegítése (költség és időmegtakarítás és a redundánsan implementált üzleti logika eltűntetése)
    - Minél több információ, amire a szolgáltatás használatához szükséges lehet:név, funkció leírása, support, kapcsolattartó, szolgáltatás gazda és felelős, szolgáltatás szint jellemzők (pl. max terhelhetőség, átlagos válaszidők), meghívás költsége
- szolgáltatásokhoz tartozó termékek tárolása:
    - Interfész definíciók (WSDL, XSD) tárolása
    - További formalizált információk tárolása (pl. saját XML leíró állományok, WS-Policy állományok)
    - Kevésbé formális leírások tárolása (pl. fizika rendszernév, felhasználói kézikönyvek, release notes
    - Külső rendszerekben tárolt információkra linkek tárolása
- Szolgáltatás taxonómia kezelése (szolgáltatások területi besorolása, csoportosítása):
    - Funkcionális, üzleti területi (pl. könyvelés, kontrolling, ügyfél menedzsment)
    - Állapot (pl: tervezés vagy implementáció alatt, tesztelhető, éles)
    - Technológia területek (pl .hiba-és kivételkezelés, loggolás)
    - Futtató környezetek (pl. fejlesztői 1. és fejlesztői 2., teszt, éles)
- Szolgáltatások, kapcsolódó termékek és kapcsolatok nyilvántartása
    - Függőségi térkép, hatásanalízis alapja
- Szolgáltatás metaadatok elérése runtime környezetben:szolgáltatás végpont lekérése (hogy ne kelljen beégetni), dinamikus végpont kiválasztás (költség és prioritás tényezők alapján)
- Szolgáltatás életciklusának kezelése: vállalatonként eltérő lehet, hogy a szolgáltatás milyen főbb lépésekben valósul meg, a szolgáltatás életciklus a registryben történhet manuális módon vagy külső (release magamagement) eszköz által. Aktív: állapotváltási feltételek ellenőrzése (pl. be van-e állítva a végpont élesítés előtt, állapotváltás kapcsán végzett műveletek (pl. élesítés tényének bejegyzése más rendszerekbe), állapotváltás utáni műveletek (pl. értesítések kiküldése)
- Értesítés küldés változásokról: a registry célja a szolgáltatásokkal kapcsolatos kommunikáció, információátadás támogatása és nem csupán publikálás és keresés a cél. Hasznos, ha változásokra az érintettek feliratkozhatnak, így autmatikus értesítést kapnak pl. a változásokról, új verziók létrejöttéről, új szolgáltatások üzembe állításáról, szolgáltatás életciklusbeli tovább lépésről. Segíti a belső kommunikkációt.
- Szolgáltatásverziók kezelése
    - Verzióváltások típusai:
        - Kompatibilis verzióváltás (változatlan interfész, a hívó fél számára változatlan működés)
        - Inkompatibilis verzióváltás (változó interfész, a hívó fél módosítását is igényli)
    - A verzióváltás lebonyolítása:
        - pillanatszerűen
        - párhuzamos együttélés
        - ráépítés
        - Párhuzamos együttélés és ráépítés esetén a szolgáltatások kapcsolatait verzióhelyesen kell nyilvántartani.
- Szolgáltatástáskezelés szabályainak nyilvántartása: pl. WS-Policy állományok révén, a WS-Policy kiterjeszthető nyelvtan, mellyel megadhatjuk XML WS alapú rendszerek elemeinek: képességeit, elvárásait, működési karakterisztikáit

### *Mi nem a Service Registry?*

- Nem verziókezelő rendszer! Mivel nem tárol forráskódot, csupán a szolgáltatások interfészeit és a hozzájuk kapcsolódó egyéb dokumentumokat.
- Nem Asset Management Tool! Nem tárol kód-mintákat, sablonokat, best practice-ket. („Nem minden szolgáltás újrahasznosítható „asset” és nem minden újrahasznosítható „asset” szolgáltatás.”).
- Nem Configuration Management Database (CMDB)! A regsitry nem tárol konfiguráció szintű információkat a szolgáltatásokról.

# *15. Általános projektmenedzsment*

### *Bevezető*

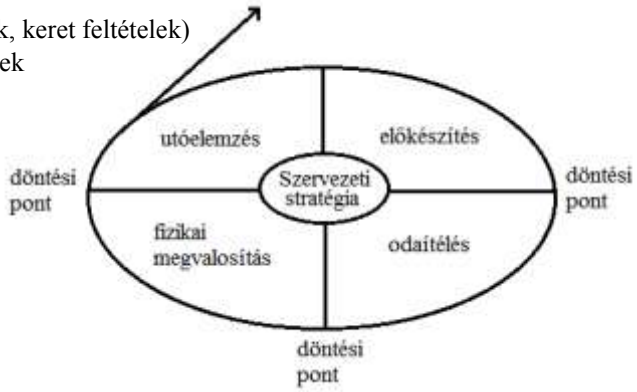
- Globalizáció: alkalmazkodás, változásmenedzsment
- Stratégia, Stratégiai célok: Projektmendzsment

- Vezetési dimenziók:
  - operatív: aktuális közvetlen célok, napi feladatok
  - stratégiai: külső körülmények várható változásaiból és a belső adottságok sajátosságaiból kell meghatározni a szervezet jövőbeni változási pályáját
  - projekt: a stratégia egy-egy jól körülírható, komplex, egyszeri feladata

Stratégia:
- kívánatos jövőbeli állapot
- döntések kiválasztása
- eszközök erőforrások

Stratégiai elemzés: lehetőségek, irányok megfogalmazása (külső környezet, belső adottságok, külső és belső érdekcsoportok elvárásai), feltétel rendszer kialakítása.
Projekt: minden olyan tevékenység, amely egy szervezet számára olyan egyszeri és komplex feladatot jelent, amelynek teljesítési időtartama (kezdés és befejezés) valamint teljesítésének költségei (erőforrások) meghatározottak és egy definiált cél (eredmény) elérésére irányulnak. Projekt karakterisztika:
- cél (eredmény, specifikáció, funkciók)
- időtartam (kezdet és vég határidők)
- költségkeret (erőforrások)

Projekttipológia:
- beruházási projektek (létesítmény)
- kutatási és fejlesztési projektek (termék, technológia)
- szellemi szolgáltatás projektek (működési körülmények, keret feltételek)
- másik lehetőség csoportosításra: külső és belső projektek




### *Projektciklus*

1. Előkészítés
   - igények
   - variációk kialakítása
   - megvalósíthatósági tanulmányok
   - variációk választása
   - döntési pont: projektcélok rögzítése
2. Odaítélés
   - projektstratégia
   - ajánlati felhívások
   - ajánlat értékelés
   - szerződés
   - döntési pont: az eredményekért és a megvalósítás idő-és költség kereteiért való felelősség, kockázat rögzítése
3. Fizikai megvalósítás
   - tervezés
   - megvalósítás
   - tesztelés
   - üzembe helyezés
   - próbaüzem
   - döntési pont: az eredmény műszaki elfogadás (átadás-átvétel)
4. Utóelemzés
   - az eredmény illeszkedése a szervezeti stratégiába
   - a megvalósítási folyamat elemzése

### *Projektstratégia*

Szerződés típusok:
- Tradicionális típusú szerződés: résztevékenységek más-más alvállalkozó
- Kulcsrakész típusú szerződés: egyetlen közreműködő viseli a felelősséget
- Menedzsment típusú szerződés: átmenet (spec: menedzsmentvállalkozó)

Elszámolás módja
- Ár bázisú: a projekttulajdonos által fizetett rész előzetesen rögzítésre kerül. Merev az inflációval szemben (...)
- Költség bázisú: közvetlen költségek + nyereség
- Cél bázisú: elsődleges célokhoz kötött

Projektkockázatok
- Bizonytalanság: a teljesítés kockázatainak forrása
- operatív működési folyamatok újszerűsége

- a spec. terjedelmi és tartalmi rögzítése, mértéke
- az eredményt megvalósító tevékenységfolyamat újszerűsége
- gazdasági környezet stabilitása
- infláció

Versenyeztetési eljárás
- Nyílt: mindenki
- Szelektív: minősítés szükséges az ajánlattételhez
- Kétszintű: az ajánlatadás egybeesik az előminősítéssel
- Meghívásos: csak a meghívottak adnak be ajánlatot

Projektalakítás
- Befolyásoló tényezők
  - Funkciók tartalma és száma
  - Egyes funkciók kapacitásjellemzői és minőségkövetelményei
  - Az operatív működést biztosító eszközök, alkotóelemek száma, összetettsége
- Funkcióstruktúra (hierarchikus kövspec. – követelmény jegyzék)

Tevékenységstruktúrák
- Hierarchikus struktúra
- Fentről lefelé (egyszerűbb)
- Lentről felfelé (ellenőrzés
- Ez alapján kialakíthatók a projekt költség- és időkorlátai

### *Teljesítési tervek*

Időterv: a tevékenységfolyamatok elemeinek időbeli összefüggései. Grafikai megjelenés:
- oszlopdiagramok (sávdiagram) (pl. Gantt-diagram, LOB (Line of Balance))
- hálódiagramok
  - tevékenység = nyíl (egy csomópont a befutó nyilak által reprezentált tevékenységek befejezése, a kiinduló nyilak a tevékenységek elkezdése, sorrendiség és párhuzamosság, CPM, PERT)
  - tevékenység = geometriai alakzat (a nyilak csak kapcsolatokat fejeznek ki, minden tevékenység egy csak egy szám, átfedések, MPM)
  - Összehasonlítás:
    - GANTT: tevékenységek, események kevésbé, folyamatos teljesítés
    - CPM: események, eseményszerű teljesítés, időskálázott CPM
    - PERT: tevékenységszemléletű, valószínűség

### *Errőforrástervezés*

- Beruházási projekt: elsősorban materiális erőforrások
- Kutatási és fejlesztési: szellemi erőforrások is
- Emberi erőforrások:
  - szakmai leltárának mátrixa (ki mihez ért táblázat)
  - feladat/felelősség mátrix (mely tevékenység kinek a feladata, illetve felelősége táblázat)

### *Erőforrástervezési módszerek:*

- Időkorlátos: az idő kötött, de korlátlan erőforrások
- Erőforráskorlátos: az erőforrások kötöttek, de az idő mindegy
- Gyakorlatban ezek keveréke, kompromisszumok: időterv kritikus útjának rövidítése, erőforráskiegyenlítés

### *Időterv kritikus útjának rövidítése*

- Egyfajta időterv optimalizáció
- Több erőforrás: egyszerű, de költséges
- Több párhuzamos rész: több erőforrás, rövidebb időre
- Átfedések növelése: terv részletesség
- Hatékonyság növelése pl: komponensek vásárlása
- Logikai függőségek újrarendezése

### *Erőforráskiegyenlítés*

- Egyenletesebb erőforrásfelhasználás
- Nem kritikus tevékenységek csúsztatása
- Csúcsidőszakokra: bérlet, alvállalkozásba adás

### *Költségtervezés*

Általában kétszeres költségek. Okai:

- korai szakasz becsléseinek hibái
- előre nem látható technikai problémák
- projektalakítás hiányosságai
- teljesítés közbeni változtatások a projekt terjedelmében
- gazdasági és egyéb tényezők

Paraméteres költségbecslés: projektindítás korai fázisában, egy korábbi projekttel összevetve, bizonyos paraméterek eltérésével. Egyéb tényezők: infláció, valutaárfolyamok
Tevékenység alapú költségbecslés: kész a projekt tevékenységi struktúrája: Ismert az időterv és erőforrás-szükséglet is
Tényleges tevékenység költségek, általános terhelés, Egységár: jól kvantifikálható projekt esetén

### *Tartalékkeret (puffer)*

- A tartalékkeret nagysága arányos legyen a meghatározó kockázatokkal
- Érintett tevékenységhez rendeljük a puffert
- A becsült költségeket és a hozzájuk tartozó tartalék keretet külön kell kezelni

### *Tervezési lépések*

1. Funkcióstruktúra
2. Tevékenységek meghatározása
3. Logikai kapcsolat, függőségek
4. Erőforrás allokáció
5. Időtartamok beállítása
6. Időterv ábrázolása
7. Költségtervezés
8. Időterv elemzés (kritikus út)
9. Időterv optimalizálás

### *Projekt kontroll, monitoring*

A megvalósítás (teljesítési szakasz) rendszeres nyomonkövetése, elemzések, korrekciós lépések.

- Változtatások: szándékolt eltérés az elsődleges (stratégiai) céloktól
- Eltérések: nem szándékolt, minimalizálás
- Normarendszer: információgyűjtés, elemzés

Elvárások:

- Jelezze a normáktól való várható eltéréseket
- Összhang a szervezettel, rugalmasság
- Elemzési eredmények jól értelmezhetőek
- Nem cél a személyi felelősségrevonás
- Jövőre orientált irányítási rendszer
- Értekezletek, stb.....

### *Projektirányítás szervezeti formái*

- A szervezet alapfunkciója: a tevékenységfolyamatok koordinációja
- Térbeli, időbeli, szakmai elkülönülés
- Közvetlen felügyelet: standardizált folyamat, kevés ismeretlen tényező
- Közös egyeztetés: nem standard folyamat, sok bizonytalansági tényező

## Szervezeti megoldások

### *Lineáris-funkcionális*

- A tevékenységeket a funkcionális szervezeti egységek végzik
- Nincs projektmenedzseri hatáskör: koordinátori szerep, információs központ
- Felelősség és hatáskör nincs összhangban
- Kicsi koordinatív kapacitás: standard tevékenységfolyamatok

### *Projekt-orientált*

- Elkülönült szervezeti egység végzi a projekt teljesítését
- Felelősség és hatáskör azonos szinten
- Jó koordináció
- Rossz skálázhatóság

### *Mátrix-szervezet*

- Megosztott hatáskörök:

    - Funkcionális szervezeti egységek teljesítenek: hogyan (és ki) ?
    - De a projektmenedzsmentnek is van hatásköre_ mit és mikor(ra) ?
- Sérülékeny, de van kohézió
- Jó koordinációs kapacitás

## Fejlesztési módszer(tan)ok és rendszer architektúrák

## eXtreme programming

Egy fegyelmezett és szabályozott szoftverfejlesztési megközelítés. Elsősorban nagy bizonytalanságú, dinamikusan változó követelményekkel rendelkező projektekhez. Jellemzői: ügyfélbevonás, csapatmunka
Extreme Programming módszertan, amely az egyszerűséget, kommunikációt, visszajelzést és bátorságot tekinti a legfontosabb értéknek a fejlesztés során. A megrendelő, menedzser és programozó szerepére, valamint az ezekben a szerepeket végző emberek jogaira és kötelezettségeire helyezi a hangsúlyt. Projekt életciklus:

1. A megrendelő határozza meg a számára üzleti értéket jelentő funkcionalitást
2. A programozó megbecsüli a költségeket
3. A megrendelő meghatározza a prioritást
4. A programozó implementálja a kialkudott funkcionalitást

### *Projektfelbontás*

Releasek (release plan, release meeting):mérföldkövek, dátumokkal user sztorikból
Iterációk: elsősorban prioritás alapján
Taskok: technológiai funkciók a fejlesztők számára

### *User stories*

Mint a Use Case-ek, de mégis mások. Az ügyfelek írják, nem technikai, nem limitált, nemcsak felhasználói felület. Nem túl részletes, becsléshez jó, később pontosítják, ideális időbecslés (hetek). Forrásul szolgálnak Relesase Planning Meeting-hez és az elfogadási tesztekhez.

### *Spike solutions*

Kis technológiai prototípusok a user sztorikban felmerülő problémákra. Pontosítja a user sztorikat.

### *Nevezési konvenciók*

Nemcsak változók, osztályok, kisbetű/nagybetű. Üzleti objektumok: közös elnevezések, ne hozzuk létre kétszer, ne kelljen megkérdezni mi az

### *Release plan*

- A teljes projekt terv (szerű)
- Időbecslés (ideális, nincsenek problémák, várakozások, a tesztelés is benne van és az ügyfél priorizálta)
- A sztorikból kiválasztani a release halmazokat (dátumok, utána az iterációs halmazokat)
- Amíg meg nem egyezik mindenki
- költségek, idő, tartalom, minőség (teszteltség, ...)

### *Iterációs tervek*

Minden iteráció elején. Egy iteráció 1-3 hét hosszú. Feladatok: user sztorik az release planbol és az előző iterációk hibajavításai és taszkok osztása a fejlesztőknek (1-3 nap).

### *Elfogadási tesztek*

User sztorikból az iterációk alatt. Ügyfél adja meg: forgatókönyvek tesztadatok. Automatikus tesztek: unit, bug.

### *Projektsebesség*

Hány user sztori készül el ... Ügyfél: Egy iterációra csak annyi user sztorit szabad választani, amennyit az előző iterációban sikerült megcsinálni. Fejlesztő: Egy iterációra csak annyi taskot szabad választani, amennyit az előző iterációban sikerült megcsinálni.

### *Páros programozás*

Két ember, egy gép. Jobb kódminőség, ami később megtérül. Egyikük gyártja a kódot, a másik gondolkodik a következő lépésen cserélnek.

### *Bugok, tesztek*

Elfogadási tesztek mielőtt debuggolnánk. Minden talált hibára újabb teszt (hogy vissza ne jöjjön). Unit tesztek: automatikus, black boksz, regresszió legyen meg előbb, mint a kód

### *Stand-up meeting*
Minden nap. Egy spontán, rövid közös gyűlés ahol mindenki áll (mint a fasz a lakodalomban).

### *Közös kódtulajdonlás*
Bárki bármi módosíthat: új funkciók, bugfixek, stb. Nincs egy főnök még vezető programozó sem, mindenki tévedhet – és téved is.

### *Könyörtelen refaktorálás*
- Refaktorálás, redundancia megszüntetése
- újraírás
- minták
- frissítés
- Egyszerű, jó minőségű kód

### *Gyakori integráció*
Max 1 nap, de inkább néhány óránként. Mindig a legutóbbi kóddal dolgozzunk. Kompatibilitási problémák hamar előjönnek. Egyszerre egy pár integrál (check-in-el) pl. egy dedikált számítógép.

### *Ember mozgatás*
- Nem támaszkodhatunk egy emberre. Tudás megosztás (duplikációk, minták, best practices)
- Probléma elosztás
- Segítség (mindenki ért mindenhez, load balancing)
- Párok

### *eXtreme programming összefoglalva*
- Foglalkoztatott megrendelő
- Felhasználói történetek (user sztori)
- Elfogadási teszt
- Költségbecslés
- Rövid iterációk, folyamatos program kibocsátás
- A felhasználó határozza meg a program kibocsátást
- Gyors tervezés
- Páros programozás
- Kollektív kód birtoklás
- Unit teszt

## Agile Modeling

Hatékony modellezés és dokumentálás, nem programozás. Kiterjeszti az XP elveit. Gyakorlati útmutatás.

### *AM Értékek*
- Kommunikáció
  - Csoportokon belül (fejlesztő-fejlesztő)
  - Csoportok között (fejlesztő-tervező)
  - Csoporton kívül (megbízóval)
- Egyszerűség (jól értett egyszerű modellek evolúciója)
- Visszacsatolás („Az optimizmus egy szakmai ártalom – a visszacsatolás gyógymód.”)
- Bátorság (Döntések meghozatala, ragaszkodás és változtatás)
- Szerénység (Mindenki tud olyat, amit Te nem. Kezeljük egymást e szerint. Nincs helye a hiúságnak – megbukhat a projekt.)

### *AM alapelvek*
- Egyszerűség – KISS (Keep It Simple and Stupid): Nem szabad túlmodellezni, a mindenkori igényeknek kell megfelelni.
- A változások természetesek: változnak a követelmények, azok értelmezése
- Projekttagok jönnek-mennek itt is ott is
- Következő verzió: Most megfelelni a követelményeknek nem elég – a jövőre is gondolni kell
- Evolváló modellek: nem kell rögtön a legjobb, legrészletesebb
- Takarékosság: mintha a saját pénzed volna
- Céltudatosság: Mitől lesz egy modell, dokumentum jó? Hogy a céljának megfelel – ezt kell szem előtt tartani.
- Több modell: a szoftver több aspektusát le kell fedni, de nem kell mindig az összest
- Minőség

- Gyors visszacsatolás: Minden területen: követelmények, tervezés, fejlesztés
- Csak annyi, amennyi kell: nem a dokumentumok karbantartása a cél
- A CÉL minden előtt: a jó szoftver

#### *AM kiegészítő elvek*

- A tartalom többet ér mint a külalak: nem kell mindent dokumentálni, ha megfelel a célnak
- Mindenki tanulhat mindenkitől: a változás az informatika „istene"
- Ismerd a modelleket és eszközöket
- Alkalmazkodás
- Nyitott és őszinte kommunikáció
- „I Can feel them ..." Bízz a megérzésekben – a programozás művészet

#### *AM gyakorlati útmutatások*

- Ügyfél aktív bevonása: nem csak projektmenedzsment, felhasználók, helpdesk, admin, stb.
- Bárki dolgozhat bármin
- Tesztelhetőség: írd meg a tesztet a program előtt
- Nyilvánosság: a modellek legyenek a falon, stb.
- A modellezés a megértés útja: együtt kell dolgozni másokkal
- Szabványos modell reprezentáció: például UML, de nemcsak az!
- Az átmeneti modelleket dobjuk ki

## RUP – Rational Unified Process

Módszertanok és a fejlesztések pozitív tapasztalatait integráló

- komponens alapú, modellszemléletű,
- Use case vezérelt
- Architectúra centrikus
- Iteratív és inkrementális,
- Folyamatorientált

fejlesztési paradigmákat követő filozófia, módszertani keretrendszer. Két dimenzióba szerveződik: idő (iterációk) és fejlesztési feladatok (feladatok, aktivitások). Fejlesztési feladatok: követelmények gyűjtése, analízis, tervezés, implementáció, tesztelés. Fázisok: kiindulás, részletes kidolgozás, építkezés, átmenet

## MSF – Microsoft Soultions Framwork

#### *Projektmenedzsment keret*

- Útmutatás a sikeresebb IT megoldásokhoz: gyorsabban, kevesebb emberrel, kisebb kockázattal, jobb minőségben
- Elvek, folyamatok, best practice-ek gyűjteménye
- NEM módszertan !

#### *Minimális/ideális méret*

- Nem minden projekthez
- De minden méretre vannak használható részek
- Elsősorban nagyobb projektekre: 3-12 hónap (leginkább 4-6), minimum 3 (ideálisan 7-11) létszámú csapatban

#### *Hibák forrásai*

- A cél és funkció eltolódása
- Az üzlet és a technológia távolsága
- A közös nyelv és folyamatok hiánya
- Csapatmunka hiánya
- Változással foglalkozó folyamat hiánya
- Megoldás?

#### *Általános jellemzők*

- A csapat minden tagja tudja, hogy mi történik, miért ki a felelős
- Elosztott kötelezettségek
- Fő cél: üzleti érték szállítása (befektet a minőségbe)
- Az időket a csapattagok becslik, így az idők jobban tarthatók és jobban betartathatók. Az időbecslés bottom-up típusú.

- Agilitás: számít a változásokra
- Tanul a tapasztalatokból
- Kulcspontok: csapatmodell, folyamatmodell, kockázat kezelés

## MSF csapat modell

MSF 6 csapata: Program management, Development, Test, Release Management, User Experience, Product Management

### *Szerep alapú megoldás*

- Egyenrangú szerepek
- Mindenki feladata, hogy terméket készítsen (a szerepkör másodlagos)
- Motiváltság maximalizálása (Sör, póló, stb.)
- Mindenki részt vesz a tervezésben is

### *Skálázás*

- Szerepek összevonása, korlátozásokkal (például Product és Program Manager-t, vagy a Developer-t bárkivel tilos)
- Skálázás felfelé (több 1000 ember) :
    - Functional Teams (több ember egy csoportban)
    - Feature Teams (alcsapatok egy feature-höz)

### *Product manager*

- Ügyfél elégedettség (Marketing)
- Üzleti értékek képviselete

### *Program manager*

- Megoldás szállítása a kereteken belül (Pénz, idő, …)
- Projektmenedzser
    - Folyamatos ellenőrzés
    - Adminisztratív teendők
    - Mérföldkövek, ütemezés
    - Kozkázatkezelések

### *Development*

- Specifikáció szerinti megvalósítás
- Fizikai réteg tervezése
- Becslések
- Technológia

### *Tesztelés*

- Minőségi jellemzők meghatározása
- Ellenőrzése
- Tesztelési tervek, …

### *User exprience*

- Felhasználói hatékonyság növelése
- Továbbképzés, tréning
- Használhatóság
- Többnyelvűség
- Elérhetőség

### *Release manager*

- Gördülékeny szállítás
- Csomagolás
- Telepítés, konfigurálás, testreszabás

## MSF folyamatmodell

Fő elemek a folyamatban:

- Ügyfél: a megrendelő, aki fizet. Neki üzleti értéket kell szállítani. Be kell vonni!
- Felhasználó: aki használja a programot.

- Megoldás: teljes: doc,.. training, support

#### *MSF Mérföldkövek*

A mérföldkövek felülvizsgáló és szinkronizációs pontok. A mérföldkövek lehetővé teszik az addigi végrehajtás értékelését és a szükséges korrekciók megtételét Mérföldkövek típusai:

- Elsődleges mérföldkövek(elsődleges mérföldkő elérése mindig a csapat és ügyfél közötti egyetértés kérdése)
- Belső mérföldkövek (a projekt folyamat és az elért eredmények értékelése)

A leszállítandó közbenső termékek a fizikai bizonyítékai annak, hogy a csapat elérte a mérföldkövet. Jellemző mérföldkövek: Termékelképzel és elfogadva, Projektterv elfogadva, Terjedelem teljes, Kiadás

# *16. SOA projektek és módszertanok*

## SOA belépési pontok

A korai SOA bevezetések során tipikus problémák voltak:

- Big Boom megközelítés: túl sok új technológia bevezetése egy időben, üzletileg kritikus projektek függővé tétele
- Csak technológiai megközelítés: „Az IT drága játékszere.”, üzleti érték nélkül nincs rá pénz
- Technológiák, módszertanok értetlensége: rugalmatlan házilagos megoldások, instabil, rosszul menedzselt környezetek (üzemeltetési, változáskezelési problémák, felelősségi kérdések).

A meglévő infrastruktúra, környezetek szolgáltatásorientáltá tétele nem megy egyik napról a másikra. Célok:

- az induló SOA projektek méretét, scope-ját reálisan meghatározni:
  - Megoldást adni létező problémára (kimutatható üzleti érték)
  - Megvalósítható technológiai célkitűzések
- Építeni a best practicek, iparilag bevált, sikeres módszerekre, eszközökre

SOA belépési pontok:

- People entry Point: interakció és kollaboráció
- Information Entry Point: információ és szolgáltatások
- Connectivity Entry Point: biztonságos és rugalmas összekötettés
- Process Entry Point: folyamat automatizálás
- Reuse Entry Point: magas értékű, igazolt asset-ek készítése és újrafelhasználása

#### *People Entry Point*

Egységes felületet biztosít:

- meglévő alkalmazások elérésére
- az információs rendszerek, szolgáltatások elérésére
- vállalaton belüli és kívüli kollaborációra (együttműködésre)

Érték:

- nagyobb hatékonyság a hétköznapi munkában
- alkalmazások és információs rendszerek költség hatékonyabb elérése
- új szolgáltatások gyors megvalósítása
- folyamatok nagyobb rugalmassága
- jobb külső és belső kommunikáció

Technikai jellemzők:

- portál és kollaborációs rendszerek
- egységesített UI
- széleskörű eszköz támogatás

People Entry Point példa: Front-end / Unified desktop (telco): egységesített, portál alapú nézete az ügyfeleknek, szolgáltatásoknak, szerződéseknek, mely hatékonyabbá teszi: a marketinget, a több csatornás értékesítést, keresztértékesítést, a front- és a back office munkáját, együttműködését

Elvárások:

- háttérrendszerek információinak és részben funkcióinak kivezetése a felületre
- tartalom, dokumentum kezelés
- egységes bejelentkezés, jogosultság kezelés
- folyamat támogatás

#### *Connectivity Entry Point*

Alkalmazás integráció:

- meglévő alkalmazások között együttműködés szolgáltatás orientált módon,
- információs rendszerek elérése

- integráció a vállalaton belüli és kívüli rendszerekkel

Érték:

- alkalmazások közötti gyorsabb, hatékonyabb integráció
- jól kezelhető információ áramlás az alkalmazások között
- könnyebb, hatékonyabb integráció, együttműködés a partnerekkel
- konzisztens felhasználói élmény csatornától függetlenül

Technikai jellemzők:

- messaging rendszerek használata, webszolgáltatások használata
- ESB
- partner gateway megoldások

Példa: vállalati integrációs busz megalapozása: rövidtávon integrálandó rendszerek: SAP, szervíztámogató eszközök, Workforce management

Elvárások:

- üzenet routolás
- transzformáció
- széleskörű protokoll, szabvány támogatás
- nagy rendelkezés állás

### *Process Entry Point*

Folyamatmenedzsment támogatás:

- üzlet-centrikus entry point
- üzleti folyamatmenedzsment (BPM) bevezetése, informatikai támogatása

Érték:

- gyorsabb, hatékonyabban működő, jobban szabályozott folyamatok
- gyors reagálás a változásokra, jobb speed-to-market
- új folyamatok gyors megvalósítása, bevezetése

Technikai jellemzők: BPMS megoldások és azok elemei (folyamatmotor, modellező eszközök és az üzleti monitorozás (dashboard))

### *Information Entry Point*

Folyamatmenedzselés támogatása:

- meglévő rendszerekben rejlő információk elérése újrafelhasználható szolgáltatásként
- heterogén adatforrások integrációja (tisztított..., egységes információk, konzisztens…)

Érték:

- jobb, pontosabb információk a döntéshozatal támogatására
- költségek, kockázatok csökkentése
- gyorsabb, költséghatékonyabb fejlesztés (strukturált és strukturálatlan adatok elérése újrafelhasználható szolgáltatások formájában)

Technikai jellemzők:

- tartalom kezelő rendszerek (content management)
- Master Data Management
- Business Intelligence, adattárház megoldások
- federált adatbázisok, információ integráció

### *Reuse Entry Point*

Meglévő rendszerek újrafelhasználása

- meglévő rendszerekből újrafelhasználható szolgáltatások kiajánlása
- meglévő rendszerek értékének megőrzése
- duplikált fejlesztések, duplán implementált logikák elkerülése

Érték:

- új projektek gyorsabb megvalósulása a meglévő szolgáltatások felhasználásával
- kevesebb költség a duplikált logikák és fejlesztések elkerülésével
- összetett alkalmazások gyors implementációja a meglévőkre építve, gyorsabb alkalmazásintegráció

Technikai jellemzők

- alkalmazásszerverek
- korszerű fejlesztőeszközök
- adapter-rendszerek, ESB-k

Service Integration Maturiy Model (SIMM): 7 szinten definiál a szolgáltatás integráció jellemzésére cégek számára, melyek SOA-t akarnak bevezetni.

***Tipikus szerepkörök IT projektekben***

| Hagyományos szerepkörök | új, SOA-specifikus szerepkörök |
|---|---|
| • It projektvezető<br>• Üzleti elemző, szervező<br>• Fejlesztő<br>• Architekt<br>• Üzemeltető<br>• Teszteő<br>• Security szakértő<br>• „Toolsmith” | • „SOA projekt vezető”<br>• Szolgáltatás tervező<br>• Folyamat szervező<br>• Szolgáltatás fejlesztő<br>• UI fejlesztő<br>• Integrációs fejlesztő<br>• SOA Architekt<br>• Integrációs tesztelő<br>• UDDI / Registry adminisztrátor<br>• „Service governors” |

## Esettanulmány: SOA keretrendszer megalapozás

***Adottságok***

- Monolitikus háttérrendszerek (pl.: számlavezető, RPG-ben megírt, MQ-n keresztül megszólítható, máig fejlesztik) ami azt eredményezi, hogy adott a jól bevált fejlesztő gárda és technológiai korlátok
- Java és CORBA alapú middleware: tehát vendor lock-in és technológiai korlátok
- Továbbá: Active Directory, Lotus notes alapú levelezés, SAS adattárház, Scoring stb.

***Célok***

Fő üzleti célok:

- Megalapozott IT infrastruktúra,
- mely rugalmasságot, üzleti agilitást biztosít
- párhuzamosan több (5-10) fejlesztő csapat is fejleszthet rá
- CRM alkalmazás (fiók plusz telebank)
- Hitelfolyamat automatizálás

IT célok, vetületek

- Átlátható, kézbentartható SOA
- Könnyű üzemeltetés
- Korszerű technológiák bevezetése (pl. portál, workflow motor, stb)
- Célok által érintet SOSA belépési pontok: Connectivity, Reuse, Process, Information

***Terv***

- 1 éves projekt
- Szállítók kiválasztása (2-3 hónap)
- Átlapolódó megvalósulás :
    - keretrendszer infrastruktúra
    - CRM fejlesztés 2 fázisban
    - Hitelfolyamat megvalósítás 1 fázisban
    - Ennek farvizén SOA keretrendszer megszületése
- Választott hardver platfoem: pSeries
- Választott új alap szoftverek: (Websphere Portal (megjelenés), Webpshere Application Server (szolgáltatások), MQ Workflow (workflow motor), DB2 UDB (adatbázis), DB2 Content Manager (dokumentum management), Tivoli Access Manager (jogosultságkezelés, hozzáférés).
- Választott beszállítók: 1 fővállalkozó (szakmai felügyelet, menedzsment) 2 alvállalkozó (szakértelem, fejlesztés, telepítés, konfiguráció)

***Megvalósulás***

- Alkalmazások elnyomták a keretrendszer megalapozását (emiatt már projektek csúsznak)
- lassan nehezen formálódó keretrendszer
- Telepített környezetek (fejlesztői, teszt, éles), hiányzik a pre-elés
- Dokumentáció (versenyfutás az idővel, részben utólag, sajnos a tervek is) más projektek, későbbi javítások csúsztak, költségvonzata volt
- Betanuló kollégák (meglévők ideje kevés, csak utólag vesznek fel újakat), még mindig kevés a házon belüli szakértők, az üzemeltetésen folyamatosan emberhiány van
- CRM
    - első keretrendszer alkalmazás, 2 fázisú iteratív megvalósítás
    - logikai terv után egyből fejlesztés

    - külön csapat fejlesztette a portált és az üzleti logikát
- Hitelfolyamat
    - dokumentumkelezés itt volt bevezetve először
    - iteratív megvalósítás (folyamatlépésekre bontva)
    - Külön csapat fejlesztette a portált, a workflow-t, és az üzleti logikát
- Végül 1év alatt 2 év

***Problémák***

- nincs SOA Governance, nincs CoE (szolgáltatások újrafelhasználása csak esetleges)
- Hibakeresés nehézkes
- Üzemeltetési problémák (emberihiány, minden-borul effektus)

***Előnyök***

- képesek arra, amire korábban nem
- párhuzamosan sok csapat fejleszt
- formalizált metodológia, átlátható infrastruktúra