

## Fejlesztés és működtetés

Ebbe a témakörbe tartoznak az alkalmazási rendszerek *létrehozásának és folyamatos működtetésének* teljes életciklusánál (igényfelmérés, tervezés, kivitelezés, bevezetés és üzemeltetés) használt módszerek és eszközök, beleértve a stratégiaalkotás, a fejlesztési munkafolyamatok és az állandó megújítás (innovációs) folyamatai technológiájának és szervezésének támogatását is.

Az informatikai rendszerek a technológia történelmében eddig nem tapasztalt mértékű *bonyolultságot* képesek elérni és *rugalmasságot* tudnak biztosítani az emberek számára. Bonyolultságuk és alakíthatóságuk kezdi közelíteni a biológiai rendszerekét.

A rendszerfejlesztés és -működtetés technológiai és munkaszervezési eszközeinek területén elért jelentős eredmények ellenére azonban az egyre összetettebb és kritikus feladatokat ellátó rendszerek a *határidőre* való elkészülés és *nagy megbízhatósággal* való működtetés terén továbbra sem felelnek meg a felhasználók – egyre növekvő – kívánalmainak.

A rendszerek bonyolult szoftverelemeit (például operációs rendszereket, adatbáziskezelőket) a kidolgozójuk gyakran termékként forgalmazza, azok működéséért, illetve továbbfejlesztéséért a felelősséget elvileg vállalja, ez azonban a gyakorlatban csak nehezen és töredékesen valósul meg.

Nagyon lényeges tendencia, hogy a bonyolultság növekedésével ez egyre inkább tarthatatlanná válik, és (többek között) ennek következtében jelennek meg a „nyílt forráskód” elvén alapuló megoldások, amikor a termékek karbantartását és követését a fejlesztők széles és jól együttműködő közössége végzi.

A szoftvertermékek felhasználásával kapcsolatos követelmények gyors és dinamikus változása párosulva a szoftver kézenfekvő és könnyen kivitelezhető módosításának igényével olyan új fejlesztési módszerek, valamint üzemeltetési és üzleti modellek kialakulásához vezet, amelyekben a termékek előállításával szemben a szolgáltatásszerű, azaz *szolgáltatásminőséget garantálni tudó* szempontok kerülnek előtérbe.

Ez egyrészt abban a *paradigmaváltásban* jelenik meg, hogy a hagyományos „programozás” helyébe a rendszerek meglévő – sok esetben webszolgáltatások formájában az interneten található és garantált – elemekből való összeépítése, integrálása lép. Másrészt a különböző alkalmazás- és infrastruktúra-üzemeltető cégek megjelenése, valamint a szoftver- és hardverelemek egységes IT-szolgáltatásokba történő beintegrálása mögött is ez áll.

### 1. Alkalmazásfejlesztés

Az alkalmazásfejlesztés fő célja, hogy egy adott felhasználói kör igényeit a lehetőségek szerinti legmagasabb szinten egy alkalmas szoftverrendszerrel kielégítse.

A szoftver különleges jellegzetessége, hogy technikai értelemben könnyű módosítani, ami nagyfokú rugalmasságot kölcsönöz az informatikai rendszereknek, ugyanakkor az ellenőrizetlen módosítások jelentős mértékben növelhetik a bonyolultságot, ami viszont alapvető problémát jelentettek az ilyen rendszerek létrehozásánál a kezdetektől fogva (ld. az először az 1960-as évek közepén jelentkező – majd utána többször „megújuló” – szoftverkrízis jelenségét).

Ma már jól látni, hogy a probléma az *egyszeri túl nagy változtatásban* és a *változtatások egyedi, nehezen ellenőrizhető jellegében* gyökeredzik. Az ilyen jellegű fejlesztés és működtetés merőben eltér a biológiai rendszerek kis változásokon és azonnali, egyértelmű teszten (ld. túlélés) alapuló fejlődési és működési modelljétől. Nem meglepő ezért, hogy a korszerű fejlesztési és működtetési módszerek igyekeznek ilyen biológiai indíttatású (gyorsfejlesztés, evolúciós fejlesztés, extrém programozás stb.) megközelítéseket alkalmazni.

### **1.1 Tervezési-elemzési nyelvek és támogató eszközeik**

A tervezés-elemzés meghatározó szerepet játszik az alkalmazási környezetben és a technológiában bekövetkező változások megértésében és „átvezetésében” az alkalmazási rendszerekbe.

Ennek megfelelően egyre jobban integrálódnak az objektum-orientált elemzési-tervezési nyelvek, (melyek tényleges szabványa ma már egyértelműen a Unified Modeling Language, UML), másrészt az ún. üzletifolyamat-elemző (business process analysis, BPA) eszközök, valamint az adatbázistervezést támogató eszközök között. A következő öt évben várható, hogy az üzleti és technológiai modellek átláthatóvá és átjárhatóvá válnak az alkalmazásfejlesztés teljes életciklusában, és az üzletifolyamat-elemző (BPA), az UML-alapú modellező és az adatbázistervező eszközök – a versenyben megmaradó gyártóknál – egyetlen integrált eszközkészletté alakulnak át.

Az UML második verziója fogalmak szélesebb körét támogatja nagyobb következetességgel, és szűkíti a rést az üzleti és a műszaki követelmények között, azonban ezen új elemek szoftvertámogatásának elégtelenségei még késleltetik a terjedését.

### **1.2 Modellvezérelt és komponensalapú fejlesztés**

Az alkalmazásfejlesztésben két tendencia érvényesül: az egyik az, hogy a teljes rendszer *modellekben*, azaz a további gépi feldolgozást lehetővé tevő, formális leírásokban fogalmazódik meg; a másik pedig, hogy a rendszer elemei egyre jobban – ha különböző mértékben is, de – szabványosakká válnak, azaz egyre inkább *standard komponensekből* épülnek fel.

A modellvezérelt alkalmazásfejlesztés a rendszer teljes, architekturális modelljéből indul ki, és két vagy több hierarchiaszinten keresztül finomítva és konkretizálva (például model driven architecture, MDA) jut el olyan modellekig, amelyekből már *programgenerálással* lehet az alkalmazási szoftver nagy részét automatikusan előállítani. Manapság az ún. „szolgáltatások” válnak a szoftverek komponensekre való felosztásának elsődleges egységeivé. Az – egymáshoz nyílt és szabványos felületen kapcsolódó – szolgáltatásokból a korábbinál jóval rugalmasabb architektúrával rendelkező szoftvereket lehet létrehozni (service oriented architecture, SOA).

Az architektúratervezésre, gyorsfejlesztésre, modellezésre és szolgáltatásszerű komponensekre alapuló megközelítéseknek a fejlesztés során való *együttes alkalmazásával* jelentős termelékenységjavulás érhető el. Ezt várhatóan olyan nyílt és széles körben használt keretrendszerek, illetve platformok teszik majd lehetővé, amelyeket mind az eszközyártók, mind a nyílt forráskódot támogatók, mind pedig a fejlesztők közösségei elfogadnak.

Azok az IT-szervezetek, amelyek modellvezérelt alkalmazásfejlesztést és szolgáltatásalapú keretrendszereket használnak a rendszerkomponensek létrehozására, kb. másfélszer *termelékenyebbek*, mint a hagyományos (3GL) integrált fejlesztési környezetek használói, és *reagáló képességük* is hasonlóan javul.

Nem véletlen ezért, hogy a nagy szoftverfejlesztő cégek ma már szolgáltatás-orientált architektúrára írják át alkalmazáscsomagjaikat, amely lazán integrált (és jellemzően már kész) ún. szolgáltatásokból építi fel. A SOA-ra való áttérés folyamán a legnagyobb kihívást az üzleti folyamatok és adatok *szemantikai egységesítése* jelenti, amelynek révén a gyakorlatban is ezen az új (szemantikai) szinten fog megvalósulni a folyamatok közötti együttműködésnek.

A SOA elterjedése a szoftvercsomagok licencvásárlása helyett a szolgáltatások *előfizetése* felé fogja eltolni a szoftverbevételeket, valamint a monolitikus szoftvercsomagoktól az *összetett alkalmazások* – azaz több, különböző szolgáltatásból összeépített alkalmazások – irányába való elmozdulást eredményez. Az elkövetkező években már az új alkalmazási szoftverekből származó bevételek nagy része SOA-t használó szoftvertermékekből realizálódik, akár hagyományos licenc-, akár előfizetési díjak formájában. Emellett a szoftverintegrátorok és a szoftvergyártók közötti megkülönböztetés egyre inkább elmosódik, ahogy az alkalmazási csomagokat részekre bontják, és azokat szolgáltatás-orientált alkalmazásként szállítják.

### **1.3 Nyílt forráskódú szoftverek**

A nyílt forráskódú (open source software, OSS), licencköteles termékek piaci részesedése növekszik, és ez több szoftverpiacon is nyomást gyakorol a hagyományos, vásárolt szoftvermegoldásokra. Emellett a nyílt forráskód fejlesztésének bevált gyakorlata alapvető változásokat kényszerít ki az informatikai ipar szoftverfejlesztéshez, -elosztáshoz és -támogatáshoz való hozzáállásában. Ezek a hatások globális méretekben játszódnak le, és hamarosan megváltoztathatják a szoftverpiaci erőviszonyokat.

Az OSS-modell hatással van a szoftvert kifejlesztésének módjára, ugyanis a hangsúlyt a felülről jövő, szigorú ellenőrzéssel szemben az együttes és együttműködő erőfeszítésekre helyezi, valamint a szoftver támogatására is hatással van, mert a hangsúlyt a minőség biztosítása és a szoftver evolúciója terén közvetlenül a felhasználói közösségre helyezi. Mindez befolyásolja a szoftverek használati módját, előtérbe helyezve a szolgáltatásalapú kereskedelmi modelleket a szoftverlicenccel szemben: azaz a felhasználó nem magáért a szoftvertermékért fizet, hanem az annak hosszabb távú felhasználásához szükséges oktatási, testre szabási, karbantartási, üzemeltetési stb. szolgáltatásokért. A nyílt forráskódú szoftverek terjedése jól mutatja, hogy nem önmagában a szoftver kód az érték, hanem az a fejlesztői-üzemeltetői kapacitás, amely mögötte áll.

Mindazonáltal látni kell azt is, hogy a „nyílt forráskód” elsősorban a fejlesztő számára igazi érték: ő ért hozzá, tud tanulni belőle, tud átvenni megoldásokat, és cserében a saját maga által fejlesztett kódot – hasonló feltételekkel – fel tudja ajánlani a tágabb fejlesztői közösség számára. A végfelhasználó (azaz az emberiség túlnyomó többsége) számára azonban továbbra is csak a szoftver által nyújtott és kezelt információ az érték.

### **1.4 Alkalmazásintegráció**

Ellentétben a hagyományos szoftverfejlesztés „először programozni” (és csak azután összeépíteni) megközelítésével, mind a vállalatokon belül, mind a szélesebb (web, illetve

web 2.0) felhasználói körben egyre inkább terjed az *integrációs megközelítés*: azaz, hogy elsődlegesen meglévő szoftverelemek (például webszolgáltatások) összetételével állítani elő alkalmazásokat, és csak ezután foglalkozni azzal, hogy kell-e valamit az így összeállt alkalmazáson esetleg programozással módosítani. Az alkalmazásintegráció a szoftveripar területén az innováció fő hajtóereje volt az elmúlt évtizedben, amely a vállalati informatikán belül alakult ki, de hamar átvette a webközösség is, és az ún. *montázstechnika* (mashup) néven a web 2.0 szerves részévé tette.

A jövőbeni alkalmazásintegrációs környezeteknél stabil elemként várható valamilyen központi, *integrált metaadattár* használata, amely az integrációval kapcsolatos összes metaadat egységes kezelését biztosítja. A nem alkalomszerűen végzett integrációnál, például a vállalati IT-infrastruktúrában *szolgáltatásbuszok* (Enterprise Service Bus, ESB) megjelenése várható, mint a köztesszoftverek új típusa, amelyek ötvözik a korábbi köztesszoftverekben található tulajdonságokat. Az ESB-ek szolgáltatásregisztrációt és -felkutatást tesznek lehetővé, a kommunikációban rugalmasságot, és általában a szolgáltatásminőség magasabb szintjét biztosítják. Az ESB-eket ki lehet majd terjeszteni teljes integrációs keretrendszerre is, üzletifolyamat-kezeléssel, B2B-képességekkel és alkalmazási adapterekkel bővítve.

Az alkalmazásintegráció egyéb, jelen pillanatban periférikusnak számító technológiai és platformjai például az ágensalapú integrációs eszközök, a szótár- (pontosabban: ontológia-) alapú transzformációs eszközök és az ún. vállalati információintegráció (Enterprise Information Integration, EII) is hatalmas lehetőségeket rejtenek magukban, és a szoftverinfrastruktúrák területén belül várhatóan itt történik majd a legtöbb innováció.

## 2. Szoftverminőség-menedzsment

A szoftverminőség nem azonos a műszaki kiválósággal (technical excellence), mert magában foglalja a gazdasági szempontokat is. Alapvetően négy dimenzió mentén célszerű vizsgálni:

- 1) *használati* (funkcionális) *tulajdonságok*, azaz milyen mértékben alkalmas a tervezett használatra;
- 2) *működési* (nem-funkcionális) *tulajdonságok*, azaz milyen mértékben áll megbízhatóan rendelkezésre a használathoz;
- 3) *szállítási idő*, azaz mennyi idő szükséges a szoftver létrehozásához vagy megváltoztatásához;
- 4) *ráfordítás*, azaz mennyi munka szükséges a szoftver létrehozásához vagy megváltoztatásához

Ennek értelmében a jó minőségű szoftver olyan optimális állapotnak felel meg, ahol a lehetséges mértékű *ráfordítás* a szükséges információkezelési *tulajdonságok* elegendően rövid *szállítási idő* alatti előállítását teszi lehetővé. A szoftverminőség-menedzsment lényegében e négy dimenzió folyamatos kiegyensúlyozását, és a szoftver (ilyen értelemben vett) optimálishoz közeli állapotban való tartását jelenti, amely a szoftverminőség folyamatos monitorozását is igényli. Ehhez nyújtanak támogatást a szoftvermetrikák, illetve az ezeken alapuló hibahajlandósági, karbantarthatósági, változtathatósági, megérthetőségi modellek, valamint a problémás programrészek azonosítása és ezek (fél)automatikus transzformációja (refactoring).

### 2.1 Teszt- és vizsgálati módszerek

Tipikus, hogy a teljes szoftverfejlesztési ráfordítás közel felét a tesztelési, karbantartási költségek jelentik. Ezért várható olyan szoftvertechnológiai módszerek széles körű elterjedése, amelyekkel ezek a költségek jelentősen csökkenthetők. Azok a szervezetek, amelyek ilyen módszereket nem használnak már a közeljövőben ki fognak szorulni a szoftverfejlesztési piacról.

A gyakran több millió soros rendszerek változtatását és – ehhez kapcsolódóan – folyamatos tesztelését segítik az ún. *regressziós tesztelési eljárások*, amelyek segítségével lehetőség nyílik arra, hogy csak a változások miatt módosult részeket és azok hatásait kelljen újra tesztelni. Ez a technológia a forrásprogramok nagyon pontos függőségi analízisén (szeletelés) alapul, amihez sok technikailag nehéz problémát kell megoldani, például statikus hivatkozások hatékonyság analízise.

A szoftverminőség fontos dimenzióját alkotó, nem-funkcionális tulajdonságokat (teljesítmény, rendelkezésre állás, folytonosság és biztonság) különböző – összefoglaló néven – *teljesítménytesztelési eszközökkel* lehet ellenőrizni és mérni. A távolabbi jövőben pedig a nagy szoftverrendszerek új (például webes) környezetbe való *bevezetésénél* (migration) segíthetnek sokat az olyan módszerek, amelyek segítségével automatikusan megállapíthatók e rendszerek bizonyos funkcionális tulajdonságai (aspect mining), illetve tervezési struktúrája (design pattern recognition).

Növekvő szerepet kapnak a jövőben a különböző tesztelési és egyéb vizsgálati módszerek menedzsmentjét (például ütemezését, automatikus végrehajtását) biztosító ún. *tesztmenedzsment-eszközök*.

## 2.2 Érettségi modellek

A szoftverminőség *szervezeti szintű* garantálásának több mint két évtizede megalkotott ún. képességérettségi modellje (Capability Maturity Model, CMM) egyre szélesebb körben terjed, mivel megújított, integrált változata (Capability Maturity Model Integrated, CMMI) még jobban megfelel a mai korszerű szoftverfejlesztési szemléletnek. A CMMI több szintű folyamat- és szervezetcélmérési megközelítése lényegesen alaposabb és finomabb értékelésre és javaslatételre ad lehetőséget, mint például az ISO 9001 szerinti felülvizsgálat.

A CMMI kiegészítve a fejlesztő csapatokra (Team Software Process, TSP) és a fejlesztőkre, mint egyénekre (Personel Software Process, PSP) szabott fejlesztési folyamattal igen hatásos eszköz lesz a következő évtizedben is a szoftvergyártással foglalkozó szervezetek számára a szoftverminőség garantált és gazdaságos biztosítására. Nem véletlen, hogy a nagy európai IT-tanácsadó és rendszerintegrátor cégek stratégiai célul tűzték ki a CM-modell 3-5. szintjének elérését.

## 2.3 Szoftverprojekt-menedzsment

A szoftverminőség-menedzsment talán legfontosabb területe a fejlesztési projektek menedzsmentje: ennek keretében lehet csak a szoftverminőség mind a négy dimenzióját, azaz a használati és a gazdasági szempontokat egyaránt kiegyensúlyozni.

A projektmenedzsment területén ma két módszert lehet elterjedtnek tekinteni: a PMBOK-ot (Project Management Body Of Knowledge) a PMI-től (Project Management Institute), illetve a PRINCE-t (PRojects IN Controlled Environment) az APM Group-tól (Association of Project Managers). Mind a két módszer igen kiértékelt megközelítés, amelyeket számos projekten sikeresen alkalmaztak. A különbségek nem lényegesek,

inkább szemléletbeliek. Várható, hogy e két módszer (illetőleg újabb változatai) a következő évtizedben is meghatározó szerepet töltenek be azoknál a szervezeteknél, amelyek a projektjeik irányítására megbízható és intézményesíthető megoldásokat keresnek.

A projektmenedzsment általános módszereinek és eszközeinek fejlődése mellett tovább terjednek a fejlesztést, mint *csoporthatást támogató eszközök* (groupware), amelyek figyelembe veszik a fejlesztő csapatok gyors átalakításának (rekonfigurálásának) igényeit is.

A szoftverfejlesztés menedzsmentjében ma már jelentős szerepet játszanak a különböző gyorsfejlesztési módszerek (Rapid application Development) is, amelyek gyakran műszakiterv-vezéreltek (design-driven) és felhasználó-orientáltak a fejlesztés gyors és iteratív megvalósítása érdekében. A gyorsfejlesztési módszereken belül az alábbiak segítségével a fejlesztés kockázatának radikális csökkentése érhető el:

- a követelmények fokozatos (iteratív) feltárása a felhasználók intenzív bevonásával,
- az architektúra folyamatos hozzáigazítása a követelményekhez,
- a programozás jelentős részének kódgenerálással való kiváltása és
- a projektadminisztráció minimálisra csökkentése.

Az ilyen fejlesztési megközelítéseket gyakran ún. *agilis módszereknek* is nevezik, amelyek közé tartozik például a DSDM (Dynamic System Development Method), a SCRUM és az XP (eXtreme Programming).

A szoftverfejlesztési projektek javításának új irányát képviselik az *empirikus, kísérleti megközelítések* (experimental software engineering), amelyek alapelve, hogy csak olyan változtatásokat szabad végrehajtania folyamatokban, amelyek kimutathatóan és bizonyíthatóan eredményeket hoznak a szoftverminőségben: ne elfogultan az eszközökön vagy a módszereken legyen a hangsúly, hanem mindig az eredményt, illetve a legjobb eredményt hozó megoldásokat valósítsák meg.

### **3. IT-szolgáltatásmenedzsment**

Az IT-szolgáltatásmenedzsment szűkebb és eredeti értelmezésben az informatikai rendszerek üzemeltetésével és ezen belül is elsősorban az IT-infrastruktúra fenntartásával foglalkozó tevékenységkör volt. Ma már azonban a szolgáltatásmenedzsment fogalmát kiterjesztik az *IT-rendszerek teljes életciklusára*, azaz az előkészítés (stratégia), a fejlesztés, a bevezetés és az üzemeltetés szakaszaira, valamint ezek folyamatos továbbfejlesztésének tevékenységkörére.

A vállalati informatika korszerűsítése során egyre nagyobb mértékben tölt be *integráló szerepet* ez a 2000 óta egyébként is erősödő IT-szolgáltatásmenedzsment tevékenységi kör, amellyel *egyensúlyt lehet teremteni* a tárgyi jellegű IT-erőforrások (például hardver, szoftver) és a nem-tárgyi, szellemi jellegű IT-képességek (például folyamat, kultúra, tudás) között. És bár kétségtelen, hogy az IT-szolgáltatásoknál egyre meghatározóbb az automatizálás szerepe, az automatizálásban fennálló esetleges hiányosságokat és elégtelenségeket ilyen módon az IT-szolgáltatásmenedzsmenttel mindig át lehet hidalni. Ez a hangsúlyeltolódás a következő évtized első felében már megmutatkozik abban is, hogy a hardverköltségek és a karbantartási munkaköltségek helyett a szolgáltatásmenedzsment működtetésének és menedzsmentszoftverek költségei lesznek a meghatározóak. Az IT-szolgáltatásmenedzsment folyamatos fejlesztésének képessége

egyre fontosabbá válik mind a belső, mind a külső IT-szolgáltatóknál olyannyira, hogy a piacszerzés eszköze és a versenyben maradás feltétele lesz.

### 3.1 ITIL

Az ITIL (IT Infrastructure Library) a legismertebb és legjobban elterjedt megközelítés az IT-szolgáltatásmenedzsment területén. Az 1980-as évek végén eredetileg az IT-infrastruktúra menedzselésére létrehozott ajánlásgyűjtemény, 2000-től kezdve nötte ki magát a *szolgáltatásmenedzsment nemzetközi szinten is meghatározó keretrendszerévé*, amely alapján hozták létre az IT-szolgáltatásmenedzsment ISO/IEC 20000-es nemzetközi szabványát is. A felgyülemlett tapasztalatok alapján az ITIL-t 2007-ben ismét megújították, és az ekkor kiadott harmadik verziója szerinti szolgáltatásmenedzsment ma már *szolgáltatási életciklusra* épül, amely öt tevékenységből áll.

A *szolgáltatásstratégia* tevékenységei határozzák meg, hogy milyen célokat követve, milyen irányelveket betartva, milyen szolgáltatásokat kell nyújtani rövid, illetve hosszabb távon. A *szolgáltatástervezés, -bevezetés és -üzemeltetés*, az életciklus egymásra épülő szakaszai ezt a stratégiát valósítják meg, és ezek során jönnek létre, változnak és alakulnak át a szolgáltatások. Végül az *állandó szolgáltatásfejlesztés* keretében valósul meg a szervezeti szintű tanulás, és hajtódik végre a szolgáltatások továbbfejlesztése megfelelő intézkedések, projektek és programok segítségével.

Az ITIL v3 felfogásában az IT-szolgáltatás, azaz az informatika szolgáltatásszerű nyújtása, a szervezetek *információval való ellátásának*, és ezen keresztül a számukra történő *értékteremtésnek* olyan módja, amely a szervezet számára szükséges információkat, és az ezáltal elérni kívánt eredményeket anélkül biztosítja, hogy bizonyos (például tulajdonlásból származó) költségeket és kockázatokat az adott szervezetnek vállalnia kellene.

Az ITIL szolgáltatási életciklusa *ismétlődő jellegű és többdimenziós*, amely biztosítja, hogy a szervezet eszközei és képességei egymással kiegyensúlyozottak legyenek, és minden körülmények között alkalmasak a mindenkorai szolgáltatási célok elérésére. Olyan PDCA-modellre épülő, zártkörű szabályozó rendszert alkalmaz, amely mindenfajta változtatást be tud fogadni és meg tud valósítani – legyen az stratégiai, taktikai vagy akár operatív szinten megfogalmazva.

### 3.2 Közműszerű IT-szolgáltatás

Az üzleti és egyedi felhasználók IT-alkalmazásokra és -infrastruktúrára vonatkozó igényeit egyre inkább „erőforrásraktárak”-ból elégítik ki és nem egyedi, specializált erőforrásokkal. Az ilyen ún. informatikai közmű típusú szolgáltatás magában foglalja mind az infrastruktúra-, mind az alkalmazásszolgáltatásokat.

A közműszerű IT-szolgáltatás úgy határozható meg, mint egy olyan stabil, megbízható, gyakran a szolgáltatás minőségére vonatkozó megállapodásokkal külön is garantált, *tömegigényeket* kielégítő szolgáltatása informatikai kapacitásoknak és funkcióknak, amely mögött korszerű, hatékonyan működtetett IT-infrastruktúrák állnak.

A közműszerű IT-szolgáltatás legkorábbi formája a *grid*, amelyet eredetileg kutató intézetekben, egyetemi kutatóhelyeken használtak és fejlesztettek ki azon célból, hogy a nagy számú, kihasználatlan számítógépes (többnyire PC-s) erőforrást nagy számítási igényű feladatok megoldására egységes infrastruktúrába lehessen szervezni. Jellegetessége ennek a megközelítésnek, hogy nem épít ki külön hardver- és hálózati

infrastruktúrát (különösen nem speciális számítóközpontokat), hanem a meglévő, szigetszerűen működő kapacitásokat valamilyen köztesszoftver segítségével szuperszámítógép-teljesítményű *virtuális infrastruktúrá*vá egyesíti. Legújabb és legperspektivikusabb formája viszont az az ún. *hiperszámítástechnikai* (cloud computing) megközelítés, amikor igen nagy mértékben és mindkét irányban (lefelé és felfelé is, azaz *elasztikusan*) skálázható eszközeiket, valamint IT által támogatott képességeiket a szervezetek „szolgáltatásként” nyújtják nagy számú, külső ügyfelük számára internet-technológiák felhasználásával. Jellegzetessége ennek a megközelítésnek, hogy korszerű, szerverek ezreit alkalmazó, specializált (hiper)számítóközpontokat épít ki, és alkalmaz a közműszerű szolgáltatásnyújtásra. Ez a 2007-ben megjelent hiperszámítástechnikai megközelítés biztosítja leginkább a nyílt, egységes hozzáférést, de – eltérően a gridtől – alapvetően piaci orientáltságú. A mai elnevezési szokás szerint lehet akár *IT Utility 2.0*-nak is nevezni, azaz a web 2.0 korszakában használatos informatikai közműnek.