

18 SZOFTVER- ÚJRAFELHASZNÁLÁS

TÉMAKÖRÖK

Ezen fejezet célja annak bemutatása és kifejtése, hogy az újrafelhasználás hogyan illeszthető bele a rendszertervezés folyamatába. Mire a fejezet végére érünk:

- megértjük a szoftverkomponensek újrafelhasználásának előnyeit és néhány, az újrafelhasználással kapcsolatban felmerülő problémát;
- megismerkedünk az újrafelhasználás különböző lehetőségeivel;
- megértjük az újrafelhasználás fogalmát, illetve hogy miként lehet beágyazni programgenerátorokba, vagy hogyan lehet például minták segítségével reprezentálni;
- megtanuljuk, hogy miként lehet gyorsan rendszereket fejleszteni a nagy, előre elkészített alkalmazások összeillesztésével;
- megismerkedünk az alkalmazáscsaládok fogalmával, amelyek egy általános központi architektúrából, valamint konfigurálható, újrafelhasználható komponensekből épülnek fel.

TARTALOM

- 18.1. Az újrafelhasználás területe
- 18.2. Tervezési minták
- 18.3. Generátoralapú újrafelhasználás
- 18.4. Alkalmazási keretrendszerek
- 18.5. Alkalmazásrendszerek újrafelhasználása

A mérnöki tudományágakban a tervezési folyamat a meglévő rendszerek vagy komponensek újrafelhasználásán alapszik. A gépész- és villamosmérnökök nem készítenek olyan tervet, amelyben minden komponenst külön kell legyártani. Terveiket már kipróbált és más rendszerekben tesztelt komponensekre alapozzák. Ezen komponensek alatt nemcsak az olyan kis komponenseket kell érteni, mint amilyenek a tömítések és a szelepek, hanem az olyan nagyobb alrendszereket is, mint a hajtóművek, a kondenzátorok vagy a turbinák.

Az újrafelhasználáson alapuló szoftvertervezés egy olyan mérvadó szoftvertervezési stratégia, ahol a fejlesztési folyamat szorosan kapcsolódik a meglévő szoftverek újrafelhasználásához. Annak ellenére, hogy az újrafelhasználás elő-

nyeit már sok évvel ezelőtt felismerték (McIlroy, 1968), csupán az elmúlt 10 évben történt meg a fokozatos áttérés a hagyományos szoftvertervezésről az újrafelhasználáson alapuló szoftvertervezésre. A kialakulásához olyan igények megjelenése vezetett, mint például a szoftverek elkészítési és karbantartási költségeinek csökkentése, a rendszerek gyorsabb átadása és a szoftver minőségére vonatkozó igény növekedése. Egyre több cég ismerte fel, hogy a szoftvert vagyongként kell tekintenünk, és az újrafelhasználás segítségével a fejlesztési befektetések megtérülése növelhető.

Az újrafelhasználáson alapuló szoftvertervezés olyan fejlesztési elv, amely megpróbálja maximalizálni a már létező szoftverek újrafelhasználását. Az újrafelhasznált szoftverkomponensek mérete gyökeresen eltérhet. Nézzünk néhány példát!

1. *Alkalmazási rendszer újrafelhasználása.* Teljes alkalmazási rendszerek újrafelhasználására is lehetőség van, akár azok változtatás nélkül más rendszerekbe történő befoglalásával, akár a különböző vásárlói igényeknek megfelelő újra-konfigurálással, illetve olyan közös architektúrán alapuló alkalmazáscsaládok kifejlesztésével, amelyek bizonyos speciális felhasználói igények kielégítésére készülnek. Az alkalmazási rendszerek újrafelhasználását a 18.5. alfejezetben tárgyalom.
2. *Komponens újrafelhasználása.* Egy alkalmazás újrafelhasználható komponenseinek mérete az alrendszerektől egészen az egyszerű objektumokig terjedhet. Például egy szövegszerkesztő rendszer részeként kifejlesztett mintaillesztő rendszert újra fel lehet használni egy adatbázis-kezelő rendszerben. Erről a 19. fejezetben lesz szó.
3. *Objektum és függvény újrafelhasználása.* Az egyszerű tevékenységet – például egy matematikai függvényt – megvalósító szoftverkomponensek az objektumosztályok is újrafelhasználhatók. Az újrafelhasználhatóságnak ez a szabványos könyvtárakon alapuló formája az elmúlt 40 évben volt megszokott. A függvény- és osztálykönyvtárak többsége eltérő típusú alkalmazások és platformok számára is elérhető. Alkalmazásuk viszonylag egyszerű, mert csak össze kell kapcsolni az alkalmazáskóddal. Azokon a területeken hatékony ez a módszer, ahol különleges szaktudás szükséges a függvények és objektumok fejlesztéséhez, mint például a matematika vagy a grafika.

A szoftverrendszerek és a komponensek sajátosságos újrafelhasználási entitások, és éppen ez a természetük eredményezheti azt, hogy egy új környezethez történő hozzáigazítása akár túl költséges is lehet. Az újrafelhasználás egyik kiegészítő formája a *koncepcionális újrafelhasználás*, ami azt jelenti, hogy komponensek újrafelhasználása helyett olyan absztraktabb entitásokkal dolgozunk, amelyek tervezése során azt tartották szem előtt, hogy széles körben felhasználhatók legyenek a konfigurálhatóságuk révén. A koncepcionális újrafelhasználással találkozhatunk például a tervezési minták, a konfigurálható rendszerek vagy a programgenerátorok esetében. Az újrafelhasználás folyamata – azaz a koncepciók újrafelhasználása egy olyan példányosítási folyamatot tartalmaz, amikor

az absztrakt koncepciókat átalakítjuk egy speciális szituációnak megfelelően. A koncepcionális újrafelhasználás lehetséges megközelítései közül kettőt – a tervezési mintákat és a programgenerátorokat a későbbiekben még kifejtem ebben a fejezetben.

A szoftverek újrafelhasználásának nyilvánvaló előnye, hogy csökkenti a fejlesztés költségét. Kevesebb szoftverkomponenst kell specifikálni, megtervezni, implementálni és ellenőrizni. A költségcsökkentés azonban az újrafelhasználhatóságnak csupán az egyik lehetséges előnye. A szoftvervagyon újrafelhasználásának számos további előnye van, amelyeket a 18.1. ábrán foglaltunk össze.

Előny	Magyarázat
Fokozott megbízhatóság	A működő rendszerek újrafelhasznált komponensei sokkal megbízhatóbbak, mint az új komponensek, mert a tervezési és implementációs hibákat már fedezték és kijavították.
Kisebb kockázat	Egy létező komponens esetén a komponens újrafelhasználásának költségében kisebb a bizonytalansági tényező, mint a komponens kifejlesztésének költségében. Ez lényeges tényező a projektvezetés számára, mivel így csökkenthető a projekt költségbecslésében rejlő bizonytalanság. Ez főként viszonylag nagy komponensek (például alrendszerek) újrafelhasználásakor igaz.
Szakemberek hatékony alkalmazása	Ahelyett, hogy az alkalmazásspecialisták különböző projekteken ugyanazt a munkát végeznék, ezek a szakemberek olyan újrafelhasználható komponenseket fejleszthetnek, amelyekbe beépülnek az ismereteik.
Szabványok alkalmazása	Bizonyos szabványokat – mint például a felhasználói felületek szabványait – szabványos komponensek csoportjaként lehet implementálni. Például egy felhasználói felületen a menük megvalósítására újrafelhasználható komponenseket lehet kifejleszteni. A menüket minden alkalmazás ugyanúgy jeleníti meg. A szabványos felhasználói felületek használata növeli a megbízhatóságot, mivel kevésbé valószínű, hogy a felhasználók a számukra ismerős felületen hibáznak.
Gyorsított fejlesztés	A rendszer mihamarabbi piacra vitele sokszor fontosabb a teljes fejlesztés költségénél. A komponensek újrafelhasználása felgyorsítja a rendszer elkészítésének folyamatát, mivel mind a fejlesztés, mind pedig az ellenőrzés ideje lerövidül.

18.1. ábra. A szoftver újrafelhasználásának előnyei

Azonban az újrafelhasználással kapcsolatban felmerülnek bizonyos költségek és problémák is (18.2. ábra). Különösen jelentős annak a költsége az újrafelhasználás során, hogy eldöntsük egy komponens újrafelhasználhatóságát adott körülmények esetén, illetve, hogy teszteljük a komponens megbízhatóságát. Ezek a többletköltségek egyrészt megakadályozhatják az újrafelhasználás bevezetését, másrészt pedig azt jelenthetik, hogy az újrafelhasználás kisebb mértékben csökkenti a fejlesztés teljes költségét, mint ahogyan azt előzetesen várni lehetett.

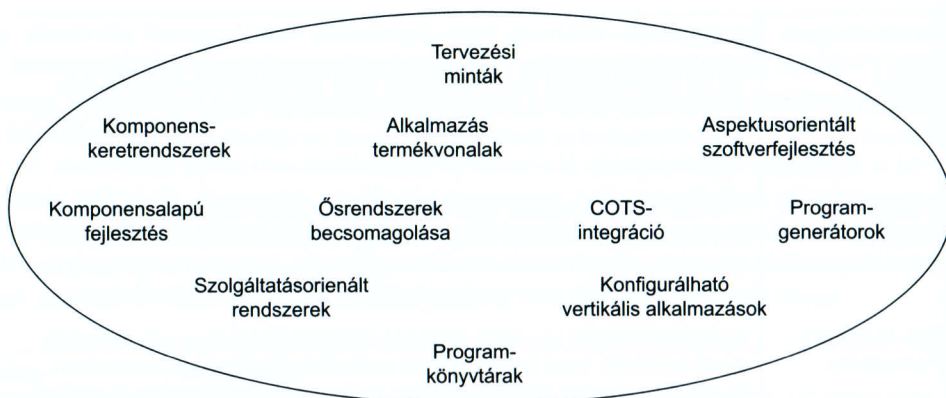
Probléma	Magyarázat
Növekvő karbantartási költségek	Ha az újrafelhasznált szoftverrendszer vagy komponens forráskódja nem hozzáférhető, akkor növekedhetnek a karbantartási költségek, mivel a rendszer változásával az újrafelhasznált elemek mindinkább inkompatibilissé válhatnak.
Az eszköz-támogatás hiánya	A CASE-eszközkészletek nem támogatják az újrafelhasználással történő tervezést. Ezen eszközök integrálása egy komponenskönyvtár-rendszerrel bonyolult, vagy akár lehetetlen is lehet. Az ezen eszközök által támogatott szoftverfolyamatok nem veszik számításba az újrafelhasználhatóságot.
„Nem-itt-találták ki” szindróma	A szoftvertervezők egy része előnyben részesíti a komponensek újírását, mivel azt hiszik, hogy javítani tudnak az újrafelhasználható komponensen. Ez részben bizalom kérdése, részben pedig amiatt van, mert egy új szoftver írása nagyobb kihívást jelent, mint mások szoftverének újrafelhasználása.
Komponens-könyvtárak létrehozása és karbantartása	Egy komponenskönyvtár feltöltése és annak biztosítása, hogy a szoftverfejlesztők ezt használhatják, magas költséggel járhat. A szoftverkomponensek besorolására, katalogizálására és visszakeresésére szolgáló technikák egyelőre még kiforratlanok.
Az újrafelhasználható komponensek megtalálása, megértése és adaptálása	A szoftverkomponenseket egy adott könyvtárban kell felfedezni, meg kell érteni, és adaptálni kell ahhoz, hogy egy új környezetben működjenek. A tervezőknek megfelelő jártasságot kell szerezniük a komponensek könyvtárban történő megtalálásában, és csak ezután tekinthetik normál fejlesztési folyamatuk részének.

18.2. ábra. Az újrafelhasználás problémái

Ezek a nehézségek azt jelentik, hogy a szisztematikus újrafelhasználás nem „csak úgy” történik, hanem egy szervezeti szintű újrafelhasználási program keretében kell megtervezni és bemutatni. Ezt Japánban már évekkel ezelőtt felismerték (Matsomuto, 1984), ahol az újrafelhasználás szerves része a szoftverfejlesztés „gyári” szemléletének (Cusamano, 1989). Az olyan cégek, mint amilyen például a Hewlett–Packard, szintén nagyon sikeresek újrafelhasználási programjuk végrehajtásában (Griss és Wosser, 1995), tapasztalataikat pedig Jacobsen és mások (Jacobsen és mások, 1997) átfogó könyvben foglalták össze.

18.1. AZ ÚJRAFELHASZNÁLÁS TERÜLETE

Az elmúlt 20 évben számos technika jelent meg az újrafelhasználás támogatására. Mind ugyanazon tényre használta ki, hogy az azonos alkalmazási területre készült rendszerek hasonlóak, ami magában rejtí az újrafelhasználás lehetőségét. Ez az újrafelhasználás egyrészt különböző szinteken lehetséges (az egyszerű függvényektől egészen a teljes alkalmazásokig), másrészt az újrafelhasználható komponensek szabványai elősegítik az újrafelhasználást. A 18.3. ábra a szoftver-újrafelhasználás különböző lehetőségeit ábrázolja, amelyek rövid jellemzése a 18.4. ábrán található.



18.3. ábra. Az újrafelhasználás területe

Megközelítés	Magyarázat
Tervezési minták	Olyan, az alkalmazásokban előforduló általános absztrakciókat reprezentáljuk tervezési mintákkal, amelyek az absztrakt és konkrét objektumokat, illetve az azok közötti interakciókat ábrázolják.
Komponensalapú fejlesztés	A rendszerek fejlesztése komponensek (objektumok gyűjteményeinek) integrálásával történik a komponensmodellek szabványainak megfelelően. Ezekkel a 19. fejezet foglalkozik részletesebben.
Alkalmazói keretrendszerek	Olyan absztrakt és konkrét osztályok gyűjteménye, amelyek felhasználhatók, illetve kibővíthetők alkalmazásrendszerek létrehozására.
Ösrendszerek becsomagolása	Olyan ösrendszerek (2. fejezet), amelyek egy interfészhalmoz segítségével „becsomagolhatók”, és biztosítják a hozzáférést ezen ösrendszerekhez a létrehozott interfészek segítségével.
Szolgáltatásorientált rendszerek	Olyan rendszerek, amelyeket megosztott szolgáltatások összekapcsolásával fejlesztenek, esetlegesen külső szolgáltatásokkal is.
Alkalmazás termékvonalak	Olyan alkalmazástípusok, amelyek egy általános architektúra köré épülnek, így a különböző vásárlók által adaptálhatók.
COTS-integráció	Olyan rendszerek, amelyeket már meglévő alkalmazásrendszerek integrációjával fejlesztenek.
Konfigurálható vertikális alkalmazások	Olyan általánosan megtervezett rendszerek, amelyek a specifikus rendszervásárlók igényeinek megfelelően konfigurálhatók.
Programkönyvtárak	Olyan, az újrafelhasználás számára megfelelő függvény- és osztálykönyvtárak, amelyek általánosan használt absztrakciókat implementálnak.
Programgenerátorok	Olyan generátorrendszerek, amelyek egy sajátos alkalmazástípusról tartalmaznak ismereteket, így az adott szakterülethez tartozó rendszereket, illetve rendszerelemeket tudnak előállítani.
Aspektusorientált szoftverfejlesztés	A program fordítása során a megosztott komponensek az alkalmazás különböző helyeire kerülnek beszövése.

18.4. ábra. Az újrafelhasználást támogató megközelítések

Miután az újrafelhasználási technikáknak egy egész sorát láttuk, a kulcskérdés az, hogy melyik a legmegfelelőbb technika? Nyilvánvalóan ez a kifejlesztendő rendszer elvárásaitól, a rendelkezésre álló technikáktól és újrafelhasználható eszközöktől, illetve a fejlesztőcsapat szaktudásától függ. Az újrafelhasználás tervezésénél az alábbi kulcstényezőket vegyük figyelembe:

1. *A szoftver fejlesztési ütemterve.* Amennyiben a szoftvert gyorsan kell kifejleszteni, az újrafelhasználás során célszerűbb megpróbálni a felhasználásra kész rendszereket alkalmazni szemben a különálló komponensekkel. Ezek többnyire nagyméretű újrafelhasználható eszközök. Bár a követelményeknek nem biztos, hogy tökéletesen megfelel ez a megközelítés, de a fejlesztéshez szükséges időt minimalizálja.
2. *A szoftver várható élettartama.* Amennyiben egy hosszú élettartamú rendszert kell kifejleszteni, akkor a rendszer karbantarthatóságára kell összpontosítani. Ilyen körülmények között nemcsak az újrafelhasználhatóság közvetlen lehetőségeire kell gondolnunk, hanem a hosszú távú következményeire is. Előfordulhat, hogy a rendszerbe új követelményeket is be kell építeni, ami valószínűleg magukat a komponenseket, valamint azok felhasználását is érintheti. Amennyiben a külső szállítók rendszereinek és komponenseinek a forráskódjához nem férünk hozzá, jobb, ha kerüljük az alkalmazásukat, mert nem lehetünk biztosak abban, hogy ezek a szállítók a későbbiekben képesek lesznek a szoftver újrafelhasznált változatának a támogatására.
3. *A fejlesztőcsapat tudása, tapasztalata és háttere.* Az újrafelhasználási technikák többsége igen összetett, és sok időbe telhet a megértésük és hatékony használatuk elsajátítása. Éppen ezért, ha a fejlesztőcsapatnak egy adott területen van tapasztalata, akkor valószínűleg ez az a terület, amire összpontosítani kell.
4. *A szoftver kritikussága és nemfunkcionális követelményei.* Az olyan kritikus rendszerek számára, amelyeket külső rendezőknek kell hitelesíteni, célszerű létrehozni egy üzembiztonsági esetet (24. fejezet). Ez a szoftver forráskódjának hiányában nehézségekbe ütközhet. Amennyiben a szoftverrel szemben a teljesítménykövetelmények szigorúak, előfordulhat, hogy bizonyos stratégiák (mint például a programgenerátorokkal támogatott újrafelhasználás) alkalmazhatatlanná válnak. Ezen rendszerek hajlamosak az igencsak használhatatlan kód előállítására.
5. *Az alkalmazási szakterület.* Számos alkalmazási szakterületen (mint például a gyártás vagy az egészségügyi információs rendszerek) léteznek olyan általános termékek, amelyek újrafelhasználhatók a helyi körülményeknek megfelelő átkonfigurálással. Amennyiben ezen a területen dolgozunk, mindig mérlegeljük ezt, mint egy lehetőséget.
6. *A rendszert futtató platform.* Bizonyos komponensmodellek, mint például a COM/Active X, Microsoft-specifikusak. Amennyiben erre a platformra fejlesztünk, talán ez lehet a legmegfelelőbb megközelítés. Hasonlóan az általános alkalmazásrendszerek is lehetnek platformspecifikusak, és csak akkor leszünk képesek ezek újrafelhasználására, ha mi is arra a platformra tervezzük a rendszerünket.

A legtöbb esetben az elérhető újrafelhasználási technikák széles köre tesz lehetővé bizonyos fokú szoftver-újrafelhasználást. Akár elértük az újrafelhasználhatóságot, akár nem, a legtöbb esetben ez sokkal inkább menedzsmenti döntés, mintsem technikai. A menedzserek vonakodhatnak a követelményeik megváltoztatásától, hogy lehetővé tegyék az újrafelhasználható komponensek alkalmazását, illetve dönthetnek amellett, hogy az eredeti komponensfejlesztés elősegítheti a szoftvertőke megalapozását. Esetleg nincsenek tisztában az újrafelhasználással járó kockázatokkal, nem úgy, mint az eredeti fejlesztés esetén. Következésképpen az új szoftver fejlesztésének magasabb kockázata ellenére léteznek olyan menedzserek, akik jobban szeretik megismerni az ismeretlen kockázatokat.

18.2. TERVEZÉSI MINTÁK

Ha végrehajtható komponenseket próbálunk újrafelhasználni, akkor ezen komponensek implementálását végző személyek részletes tervezési döntései elkerülhetetlenül megkötik a kezünket. Ezek a döntések a komponensek implementálására használt egyéni algoritmusoktól egészen a komponens interfészének objektumaiig és típusaiig terjednek. Ha ezek a tervezési döntések összeütközésben vannak az egyedi követelményeinkkel, akkor a komponens újrafelhasználása lehetetlen vagy alacsony hatékonyságú.

Ez elkerülhető az absztraktabb, implementációs részletet nem tartalmazó tervek újrafelhasználásával. Ezeket aztán speciális módon úgy implementáljuk, hogy azok megfeleljenek az alkalmazás követelményeinek. Az újrafelhasználás ilyen megközelítése először az alapvető algoritmusok dokumentációjában és kiadásában bukkant fel (Knuth, 1971), később pedig az absztrakt adattípusok (például veremk, fák és listák) dokumentációjában jelent meg (Booch, 1987). A legutóbbi időkben az újrafelhasználásnak ez a módja a tervezési minták fogalmába ágyazódott be.

A tervezési minták a Christopher Alexander (Alexander és mások, 1977) által felvetett elképzelésekből származnak, aki azt javasolta, hogy egy terv felépítésének legyenek bizonyos ismétlődő mintái, amelyek természetüknél fogva könnyen használhatók és hatékonyak. A minta nem más, mint a probléma leírása és megoldásának lényege, így a megoldás különböző környezetekben újrafelhasználható. A minta nem részletes specifikáció, inkább felhalmozott bölcsességként és tapasztalatként képzelhető el, egy gyakori probléma jól bevált megoldása. Az újrafelhasználásban betöltött szerepüket egy, a mintákról elérhető információkra szakosodott weboldal, a *hillside.net* idézete írja le:

A minták és mintanyelvek a bevált megoldások, jó tervek és felhalmozott tapasztalatok leírásának olyan útja, amely lehetővé teszi a többi felhasználó számára ennek a tapasztalatnak az újrafelhasználását.

A legtöbb tervező úgy gondol a tervezési mintákra, mint az objektumorientált tervezést támogató útra. Az általánosság biztosításának céljából a tervezési minták gyakran olyan objektumjellemzőkre támaszkodnak, mint például az öröklő-

dés vagy a polimorfizmus. Az általános elv azonban egyformán alkalmazható a szoftvertervezés minden megközelítésére.

Gamma és mások (Gamma és mások, 1995) megadják a tervezési minták négy alapvető elemét:

1. Egy beszélő név, amellyel hivatkozni tudunk a mintára.
2. A probléma leírása, amely kifejti, hogy a minta mikor alkalmazható.
3. A megoldás leírása, amely a megoldási terv különböző részeit, azok kapcsolatait és felelősségeit írja le. Ez nem egy konkrét terv leírása. Ez a megoldási terv sablonja, amely különféle módon példányosítható. Gyakran grafikus módon van leírva, és megmutatja a megoldás objektumai és objektumosztályai közötti kapcsolatokat.
4. Egy kimutatás a minta alkalmazásának következményeiről – az eredményekről és a kompromisszumokról. Ez segít a tervezőknek annak megértésében, hogy egy bizonyos esetben egy minta hatékonyan alkalmazható-e vagy sem.

A minták leírásának ezen alapvető elemei még tovább bonthatók, mint például ahogyan ez a 18.5. ábrán is látható. Például Gamma és társszerzői a probléma leírását motivációra (annak leírása, hogy a minta miért hasznos) és alkalmazhatóságra (olyan szituációk leírása, amikor a minta használható) bontották szét. A megoldás leírásánál ők a minta szerkezetét, a résztvevőket, az együttműködéseket és az implementációt adják meg.

A minta neve: Observer

Leírás: Különválasztja az objektum állapotának megjelenítését magától az objektumtól, és lehetővé teszi alternatív megjelenítők megadását. Ha egy objektum állapota megváltozik, minden megjelenítő automatikusan értesítésre kerül és frissül a változásnak megfelelően.

A probléma leírása: Sok esetben ugyanazt az állapotinformációt többféleképpen kell megjeleníteni, például grafikusan és táblázatos formában is. Az információ megadásakor nem biztos, hogy ezek mindegyike ismert. Minden alternatív megjelenítés támogatja az interakciót, és ha az állapot megváltozik, frissíteni kell a megjelenítőket.

Ez a minta minden olyan helyzetben alkalmazható, amelyben az állapotinformáció megjelenítésére egynél több megjelenítési formátumra van szükség, és ahol az állapotinformációt karbantartó objektumnak nem szükséges ismernie a használt speciális megjelenítési formátumokat.

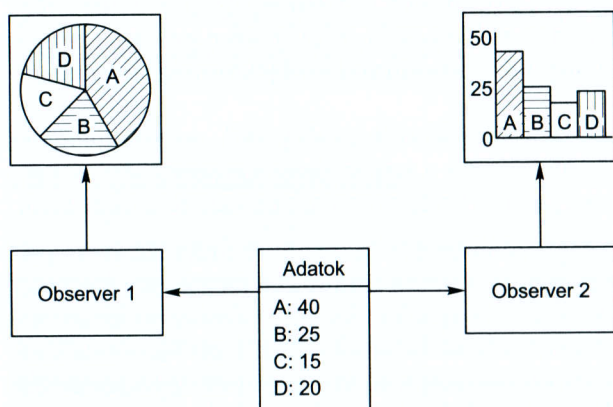
A megoldás leírása: A minta szerkezete a 18.7. ábrán látható, amely két absztrakt (Subject és Observer), valamint két konkrét (ConcreteSubject és ConcreteObserver) objektumot definiál, mely utóbbiak öröklik a megfelelő absztrakt objektumok attribútumait. A kijelzendő állapot karbantartását a ConcreteSubject végzi, amely a Subjecttől öröklí az Observerek hozzáadását és törlését lehetővé tevő, valamint az állapot megváltozása esetén értesítést kiadó műveleteket.

A ConcreteObserver a ConcreteSubject állapota egy másolatának karbantartását végzi és implementálja az Observer másolatok frissítését lehetővé tevő Update() interfészét. A ConcreteObserver automatikusan kijelzi állapotát – ez rendes körülmények között nem interfészművelet.

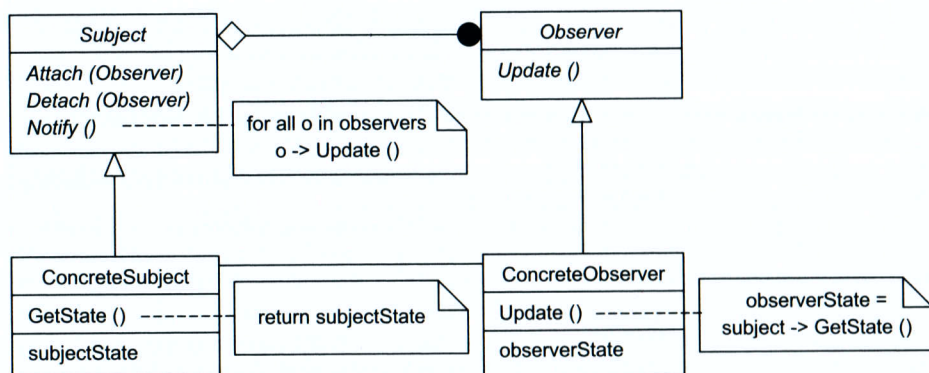
Következmények: A megfigyelt csak az absztrakt Observerről tud, és nem ismeri a konkrét osztály részleteit. Éppen ezért ezen objektumok között nem túl szoros a kapcsolat. Ennek az ismeretnek a hiányában a megjelenítési teljesítményt fokozó optimalizáció nem célravezető. A megfigyelt megváltozása a megfigyelők felé olyan összekapcsolt frissítések sorozatát válthatja ki, amelyek némelyike nem feltétlenül szükséges.

18.5. ábra. Az Observer minta leírása

A minták leírásának bemutatására az Observer mintát választottam a Gamma és mások által írt könyvből. Ezt a mintát akkor használjuk, ha egy objektum állapotát többféleképpen kell megjeleníteni. Különválasztja a megjelenítendő objektumot a megjelenítés különböző módjaitól. Ezt a 18.6. ábra szemlélteti, amelyen ugyanannak az adathalmaznak két különféle megjelenítési módja látható. A leírásban a négy alapvető leíróelemet használjuk, egy rövid megállapítással kiegészítve, amely arra vonatkozik, hogy mire képes a minta.



18.6. ábra. Különféle megjelenítések



18.7. ábra. Az Observer minta

A mintában használt objektumosztályok és azok közötti kapcsolatok ábrázolása általában grafikus módon történik. Kiegészíti a mintaleírást, és további részleteket ad a megoldás leírásához. A 18.7. ábrán az Observer minta UML-beli reprezentációja látható.

Jelenleg nagy számban érhetők el publikált minták, amelyek számos alkalmazási szakterületet és nyelvet ölelnek fel (lásd a linkeket a könyv internetes oldalán). A minta fogalma, mint az újrafelhasználhatóság koncepciója számos, a szoftvertervezéstől távoli területen is kifejlődött, magában foglalva a konfigu-

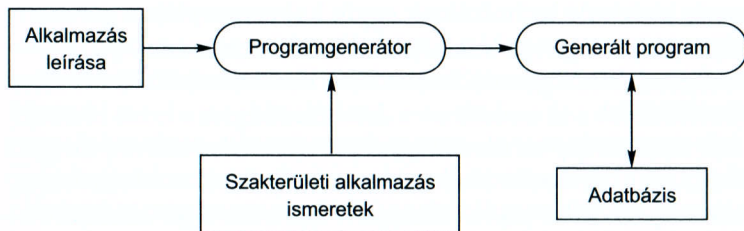
rációmenedzsmentet, felhasználói felületeket és interakciós forgatókönyveket (Berczuk és Appleton, 2002; Borchers, 2001; Martin és mások, 2001; 2002).

A minták alkalmazása az újrafelhasználás nagyon hatékony formája. De meggyőződésünk szerint, a tervezési folyamatokba történő bevezetését csak tapasztalt, a minták terén mély tudással rendelkező programozók tudják hatékonyan alkalmazni. A mintákat csak tapasztalt programozók tudják megfelelően használni, mivel csak ők képesek azoknak az általános helyzeteknek a felismerésére, amelyekben egy mintát alkalmazni lehet. A tapasztalatlan programozók – még ha el is olvasták a mintákról szóló könyveket – mindig nehezen tudják eldönteni, hogy egy mintát kell-e újrafelhasználni, vagy inkább egy speciális célú megoldásra van-e szükségük.

18.3. GENERÁTORALAPÚ ÚJRAFELHASZNÁLÁS

A koncepciók mintákon keresztüli újrafelhasználása a koncepciónak absztrakt módon történő leírására támaszkodik, és a szoftvertervezőre hagyja annak implementációját. Ennek egyik alternatívája a generátoralapú szemlélet (Biggerstaff, 1998). Ebben a megközelítésben az újrafelhasználható tudást egy programgenerátor-rendszerbe gyűjtjük, amelyet a szakértők programozhatnak, egy szakterület-orientált nyelven, vagy egy interaktív CASE-eszközzel, amely támogatja a rendszergenerálást. Az alkalmazás leírása absztrakt módon megadja, hogy mely újrafelhasználható komponenseket kell használni, és hogyan lehet azokat kombinálni és parametrizálni. Ezen információk felhasználásával egy működő szoftverrendszer generálható (18.8. ábra).

A generátoralapú újrafelhasználás annak a ténynek az előnyeit használja ki, hogy az azonos szakterületi alkalmazások, mint például az üzleti rendszerek, közös architektúrákkal rendelkeznek, továbbá összehasonlítható függvényeket alkalmaznak. Ilyenek például a 13. fejezetben már említett adatfeldolgozó rendszerek, amelyek normális esetben egy bemenet-feldolgozás-kimenet modellt követnek, és többnyire olyan műveleteket tartalmaznak, mint például adatellenőrzés és jelentés előállítás. Éppen ezért egy alkalmazásgenerátor magában foglalhat és létrehozhat olyan általános komponenseket, mint például elemek leválogatása és ellenőrzése az adatbázisból, illetve jelentések készítése. Az ilyen komponensek



18.8. ábra. Generátoralapú újrafelhasználás

újrafelhasználásához a programozó egyszerűen kijelöli a feldolgozandó adatele-
meket, a végrehajtandó ellenőrzéseket és a jelentés formátumát.

A generátoralapú újrafelhasználás kimondottan sikeres az üzleti alkalmazási
rendszerknél, és számos különböző üzletialkalmazás-generáló termék létezik.
Ezek előállíthatják a teljes alkalmazást, vagy részben automatizálhatják a létre-
hozást, a sajátos részletek felvitelét bízva a programozóra. Az újrafelhasználás
generátoralapú megközelítése más területeken is használatos, idetartoznak az
alábbiak:

1. *Elemző generátorok a nyelvek feldolgozására.* A generátor bemenete az elemzendő
nyelvet leíró nyelvtan, a kimenete pedig a szintaktikus elemző. Ezt a megkö-
zelítést alkalmazzák többek között olyan rendszerekben, mint például a lex és
a yacc a C nyelvhez és a JavaCC, ami egy fordító a Javához.
2. *CASE-eszközök kódgenerátorai.* A generátorok bemenete egy szoftverterv, kime-
nete pedig a tervezett rendszert implementáló program. Alapjául szolgálhat-
nak például UML-modellek, és a részletezettségüktől függően a kimenet lehet
egy teljes program vagy komponens, illetve egy kódvázlat. A szoftvertervező
ezután további részletekkel egészíti ki, hogy teljes legyen a kód.

A generátoralapú újrafelhasználás ezen irányzatai kihasználják, hogy az em-
lített területeken az alkalmazások megszokott struktúrával rendelkeznek. Ez a
megközelítés olyan területeken is alkalmazható, mint például a parancs- és ve-
zérlőrendszerek (O'Connor és mások, 1994) vagy a tudományos műszerezés (But-
ler, 1994), ahol komponenskönyvtárakat fejlesztettek ki. A szakterületi elemzők
ezt követően egy szakterület-specifikus nyelvet használhatnak a komponensek
összeállítására és alkalmazások létrehozására. Azonban a szakterületi koncepci-
ók meghatározásának és a kompozíciós nyelv implementálásának magas a kez-
deti költsége. Ennek következtében számos vállalat vonakodik vállalni az alkal-
mazásával járó kockázatot.

A generátoralapú újrafelhasználás például az üzleti adatfeldolgozási alkal-
mazások számára költséghatékony. A végfelhasználók számára összehasonlít-
hatatlanul egyszerűbb egy generátor segítségével programot fejleszteni, mint a
komponensalapú újrafelhasználási módszerekkel. Azonban a generált progra-
mokban elkerülhetetlen a hatásfokvesztés. Ez azt jelenti, hogy ennek a megkö-
zelítésnek az alkalmazása valószínűleg nem lehetséges az olyan rendszereknél,
ahol magas az elvárás a teljesítőképességgel és teljesítménnyel szemben.

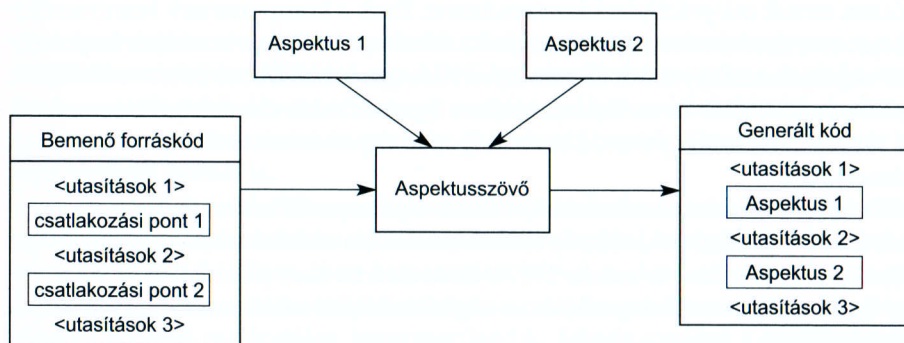
A szoftverfejlesztés kialakuló technikáinak egyik kulcsszereplője a generatív
programozás, amely a programgenerálást kombinálja a komponensalapú fejleszt-
éssel. Czarnecki és Eisenecher könyve (Czarnecki és Eisenecher, 2000) ismerteti
ezeket az új megközelítéseket.

A leginkább fejlesztett irányvonal az aspektusorientált szoftverfejlesztés
(AOSD). Az aspektusorientált szoftverfejlesztés a szoftvertervezés egyik leg-
nagyobb problémájára igyekszik megoldást nyújtani – nevezetesen a vonatko-
zások szétválasztásának problémájára. A vonatkozások szétválasztása alapvető
tervezési irányelv, amely megkívánja, hogy a szoftvertervezésnél minden egyes

komponens vagy egység egy, és csak egy dolgot végezhet. Vegyük például a LIBSYS-rendszert. Léteznie kell olyan komponensnek, amely a dokumentumok megkereséséért felelős, léteznie kell olyan komponensnek, amely a dokumentumok nyomtatásáért felelős, léteznie kell olyan komponensnek, amely a letöltések kezeléséért felelős és így tovább.

Azonban számos olyan helyzet létezik, amikor a vonatkozások nem definiálhatók tisztán alkalmazásszintű függvényekkel, mert azokat keresztülmetszik – azaz a rendszer minden komponensére hatással vannak. Gondoljunk például arra, hogy szeretnénk nyomon követni a rendszermodulok használatát minden egyes rendszerfelhasználó esetén. Ezt nem lehet egyszerűen egy olyan objektumként implementálni, amelyre a komponensek hivatkozhatnak. A végrehajtandó specifikus monitorozásnak szüksége lesz az éppen megfigyelt rendszerfüggvénytől a környezeti információkra is.

Az aspektusorientált programozás során ezek a keresztező vonatkozások jelennek meg aspektusokként, és a programban mi határozhatjuk meg, hogy mely részéhez szükséges hozzárendelni egy adott aspektust. Ezeket hívjuk *csatlakozási pontoknak*. Az aspektusok külön fejleszthetők, és egy *aspektusszövegsnek* nevezett előfordítási lépés során kapcsolódnak hozzá a csatlakozási pontokhoz (18.9. ábra). Az aspektus szövege a programgenerálás egyik formája – a szövvő kimenete egy olyan programkód, amelybe az aspektuskódot integrálták. Az aspektusorientált fejlesztés legismertebb nyelve egy Java fejlesztés, amelyet AspectJ-nek neveznek (Kiczales és mások, 2001).



18.9. ábra. Aspektusok szövése

Az aspektusorientált szoftverfejlesztés egy igen érdekes kutatási irány, de az ipari szoftverfejlesztésben nem terjedt el még széles körben. Több probléma is felmerül ezzel a megközelítéssel, a verifikáció és a validáció nehezzé válik, illetve nagyon pontos ismeretek szükségeltetnek az aspektusok és a nemfunkcionális rendszerjellemzők közötti kapcsolatokról. Mindazonáltal az AOSD nagyon gyorsan fejlődik, úgy, mint egy meghatározó, új szoftverfejlesztési technika. Ennek kapcsán a hetedik, „Kialakuló technológiák” című részben egy teljes fejezet keretében tárgyalom ezt a témakört.

18.4. ALKALMAZÁSI KERETRENDSZEREK

Az objektumorientált fejlesztés korai támogatói azt állították, hogy az újrafelhasználás céljából a legmegfelelőbb absztrakciók az objektumok. Azonban a tapasztalat azt mutatja, hogy az objektumok gyakran túlságosan finom szemcsezettségűek és egy bizonyos alkalmazásra specializáltak. Ehelyett nyilvánvalóvá vált, hogy az objektumorientált fejlesztési folyamat az újrafelhasználást legjobban a *keretrendszereknek* nevezett nagyobb szemcsezettségű absztrakciókon keresztül támogatja.

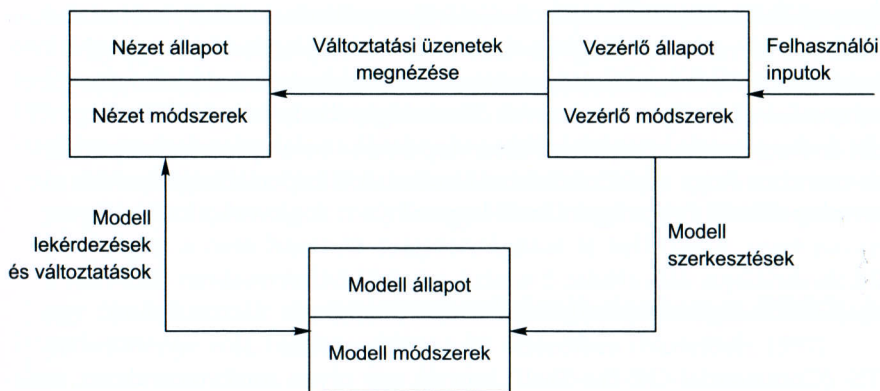
A keretrendszer (vagy alkalmazási keretrendszer) absztrakt és konkrét osztályok gyűjteményéből, valamint a közöttük lévő interfészekből felépülő alrendszerterv (Wirfs-Brock és Johnson, 1990). Az alkalmazás alrendszerének bizonyos részletei komponensek hozzáadásával és a keretrendszerben található absztrakt osztályok számára adott konkrét implementációk megadásával valósíthatók meg. A keretrendszerek a legtöbbször nem önálló alkalmazások. Az alkalmazások normál körülmények között számos keretrendszer integrálásával hozhatók létre.

Fayad és Schmidt (Fayad és Schmidt, 1997) a keretrendszereket három csoportba osztotta:

1. *A rendszer infrastruktúrájának keretrendszerei.* Ezek a keretrendszerek a rendszer infrastruktúrájának (például kommunikáció, felhasználói felületek, fordítóprogramok) fejlesztését támogatják (Schmidt, 1997).
2. *Köztes termék integrációjának keretrendszerei.* Ezek a komponensek kommunikációját és információcseréjét támogató szabványokból és a hozzájuk kapcsolódó objektumosztályokból állnak. Ilyen keretrendszer többek között a CORBA, a Microsoft COM+ és az Enterprise Java Beanek. A keretrendszerek ezen fajtái a 19. fejezetben tárgyalandó szabványos komponensmodellekhez nyújtanak támogatást.
3. *Vállalati alkalmazások keretrendszerei.* Ezek olyan speciális szakterületi alkalmazásokkal foglalkoznak, mint például a telekommunikáció vagy a pénzügyi rendszerek (Baumer és mások, 1997). Ezek szakterületi alkalmazásra vonatkozó ismereteket tartalmaznak, és a végfelhasználói alkalmazások fejlesztését támogatják.

Ahogy az a neve is sugallja, a keretrendszer általános struktúra, amely a specifikusabb alrendszerek és alkalmazások létrehozása érdekében kibővíthető. Maga a keretrendszer konkrét és absztrakt objektumosztályoknak egy gyűjteményéből áll. A keretrendszer kibővítése konkrét osztályok hozzáadásával történhet, amelyek a keretrendszer absztrakt osztályainak műveleteit öröklik. Ezenfelül lehetőség van *visszahívás* megadására is. A visszahívások olyan módszerek, amelyek a keretrendszer által felismert eseményekre adott válaszként hívódnak meg.

A legismertebb és a GUI-tervezéshez széles körben használt keretrendszerek egyike a Model-View-Controller (MVC) keretrendszer (18.10. ábra). Ez eredetileg az 1980-as években a GUI-tervezés olyan megközelítéseként jött létre, amely lehetővé teszi egy objektum többféle megjelenítését és a megjelenítések mindegyikéhez különbözőféle interakciók megadását. Az MVC-keretrendszer az adatok megjelenítését különböző módokon támogatja (18.6. ábra), és az interakciót elválasztja ezektől a megjelenítésektől. Ha az adatok valamelyik megjelenítésen keresztül megváltoznak, akkor a többi megjelenítés is frissül.



18.10. ábra. A Model-View-Controller keretrendszer

A keretrendszerek sokszor minták példányosításai, mint ahogyan azt a 18.2. alfejezetben kifejttem. Például egy MVC-keretrendszer magában foglalja az Observer mintát (18.5. ábra), a Strategy mintát, amely a modell frissítésével foglalkozik; a Composite mintát és sok más, Gamma és mások (Gamma és mások, 1995) által tárgyalt mintát is.

A keretrendszerek felhasználásával létrejött alkalmazások az alkalmazáscsaládok koncepcióján további újrafelhasználás alapjául szolgálhatnak, mint ahogyan azt a 18.5.2. alfejezetben bemutatom. Mivel ezek az alkalmazások egy keretrendszer felhasználásával jöttek létre, a családtagok új családtag létrehozása céljából történő módosítása leegyszerűsödik. Mindazonáltal a keretrendszerek többnyire absztraktabbak, mint az általános termékek, így az alkalmazásoknak sokkal szélesebb körű létrehozását teszik lehetővé.

A keretrendszerek alapvető problémái között meg kell említeni a velük járó bonyolultságot és a beletanulás idejét. Egy keretrendszer teljes megértéséhez több hónapra is szükség lehet, így valószínű, hogy nagy szervezetekben néhány szoftvertervező a keretrendszer szakemberévé válik. A keretrendszer minden kétséget kizáróan hatékony módja az újrafelhasználásnak, de a szoftverfejlesztési folyamatba történő bevezetése magas költségekkel jár.

18.5. ALKALMAZÁSRENDSZEREK ÚJRAFELHASZNÁLÁSA

Az alkalmazásrendszerek újrafelhasználása magában foglalja egész alkalmazásrendszerek újrafelhasználását, akár egy rendszer adott környezethez történő konfigurálásánál, akár két vagy több rendszer integrációjaként egy új alkalmazás létrehozására. Ahogyan az már a 18.1. alfejezetben felmerült, az alkalmazásrendszerek újrafelhasználása többnyire a leghatékonyabb újrafelhasználási technika. Tartalmazza a durvább szemcsézettségű eszközök újrafelhasználását, amelyek gyorsan konfigurálhatók egy új rendszer létrehozására.

Ebben a részben az alkalmazások újrafelhasználásának két típusát mutatom be, elsőként az új rendszerek létrehozásának azon módját, ahol két vagy több kész alkalmazást integrálunk, másodikként pedig az alkalmazáscsaládok fejlesztését. Az alkalmazáscsalád olyan rendszerek összessége, amelyek egy közös alaparchitektúrán és megosztott komponenseken alapulnak. Az alaprendszer speciálisan úgy lett tervezve, hogy a különböző rendszervásárlók speciális igényeinek megfelelően adaptálható és konfigurálható legyen.

18.5.1. COTS-termékek újrafelhasználása

A COTS- (Commercial-Off-the-Shelf) termék egy olyan szoftverrendszer, amelyet a vásárló változtatás nélkül felhasználhat. Gyakorlatilag az összes asztali szoftver és a szerveroldali termékek igen széles köre COTS-szoftver. Mivel az ilyen rendszerek általános felhasználásra lettek tervezve, általában számos szolgáltatással és tulajdonsággal felvértezve, megteremtik az újrafelhasználás lehetőségét különböző alkalmazásokban és környezetekben. Annak ellenére, hogy ennek a megközelítésnek számos problémája lehet a rendszerek létrehozása során (Tracz, 2001), egyre nagyobb számban találkozhatunk sikeres alkalmazásaival, amelyek az életképességét demonstrálják (Baker, 2002; Balk és Kedia, 2000; Pfarr és Reis, 2002).

A COTS-rendszerek néhány fajtáját természetesen már évek óta újrafelhasználják. Erre talán legjobb példa az adatbázis-kezelő rendszereké: csak nagyon kevés fejlesztő gondolkodik saját adatbázis-kezelő rendszer írásán. Egészen az 1990-es évek közepéig általában csak néhány nagy rendszert – mint például az adatbázis-kezelő rendszereket és a távfeldolgozó monitorokat – használtak fel újra. A nagy rendszerek többsége egyedülálló alkalmazásként lett tervezve, és többnyire számos probléma adódott ezen rendszerek együttműködésének megteremtésénél.

Mára azonban bevett szokás lett a nagy rendszereknél, hogy rendelkeznek alkalmazás programozói interfésszel (API), ami lehetővé teszi a rendszerszolgáltatások elérését. Ez azt jelenti, hogy az olyan nagy rendszerek, mint például az e-kereskedelmi rendszerek, COTS-rendszerek egy sorozatának integrálásával történő létrehozását mindig komoly tervezési lehetőségként kell figyelembe venni. Az ezen rendszerek által biztosított funkcionalitás miatt nagyságrendekkel csökkenthetők a költségek és a szállítási idők egy új szoftver fejlesztéséhez ké-

pest. Ezen túlmenően a kockázatok is csökkenthetők, mivel a termék már elérhető, és a menedzserek el tudják dönteni, hogy az megfelel-e igényeiknek.

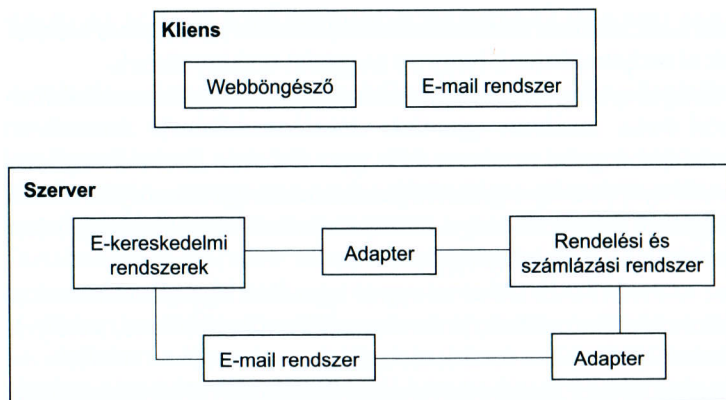
A COTS-termékeken alapuló rendszerfejlesztés során számos tervezési döntést kell meghozni:

1. *Melyik COTS-termék nyújtja a legmegfelelőbb funkcionalitást?* Amennyiben nem rendelkezünk elegendő tapasztalattal a COTS-termékek között, nehéz lehet eldönteni, hogy melyik termék a legmegfelelőbb.
2. *Hogyan történik az adatcsere?* Általában az egyes termékek egyedi adatszerkezeteket és formátumokat használnak, ezért olyan illesztőket kell írni, amelyek az egyes formátumok közötti konverziót elvégzik.
3. *Egy termék mely tulajdonságait fogjuk ténylegesen kihasználni?* A legtöbb COTS-termék több funkcionalitást biztosít, mint amire szükségünk van, továbbá egyes funkcionalitások a különböző termékek között számos esetben többször is előfordulhatnak. Nekünk kell eldöntenünk, hogy a követelményeinknek megfelelő tulajdonságok melyik termékben a legmegfelelőbbek. Amennyiben lehetséges, a nem használt tulajdonságokat le kell tiltani, mert zavarhatják a normális rendszerműködést. Az Ariane 5 rakéta első repülésének hibája is egy újrafelhasznált alrendszer nem használt tulajdonságának hibájának következménye volt, részletesebben a 19. fejezetben (Nuseibeh, 1997).

A COTS-integráció illusztrálására képzeljük el, hogy egy nagy szervezet beszerzési rendszert kíván fejleszteni, ami az alkalmazottak számára lehetővé teszi, hogy a megrendeléseiket az asztalukról feladhassák. Ennek a rendszernek a bevezetése után a szervezetnél a megtakarítás elérheti az 5 millió dollárt évenként. A vásárlás központosításával a beszerzési rendszer biztosítani tudja, hogy a megrendelések mindig azon szállítóhoz továbbítódnak, aki a legjobb árakat ajánlja, és emellett még a megrendelésekkel járó papírmunka költségei is csökkenthetők. Hasonlóan a manuális rendszerekhez, a beszerzés magában foglalja a szállítónál elérhető árucikkek kiválasztását, a megrendelés feladását, jóváhagyását, szállítóhoz történő továbbítását, az árucikkek átvételét, valamint a fizetés-meghagyás megerősítését.

A vállalat már rendelkezik egy megrendeléskezelő rendszerrel, amelyet a beszerzési osztály használ, amely már integrálásra került a számlázó- és továbbító rendszerükkel. Az új megrendeléskezelő rendszer létrehozásához a régit integrálják egy webalapú e-kereskedelmi platformmal és egy elektronikus levélküldő rendszerrel, amely a felhasználók közötti kommunikációt kezeli. A 18.11. ábrán láthatjuk a végleges beszerzési rendszer struktúráját, amely COTS segítségével lett felépítve.

A beszerzési rendszer kliens-szerver-alapú, és a kliensen csak egy webböngésző és egy e-mail szoftver szükséges, amelyeket a szoftverszállítók már integráltak. A szerveren az e-kereskedelmi platformot egy adapter segítségével integrálni kell a már meglévő megrendeléskezelő rendszerhez. Az e-kereskedelmi rendszer saját formátumot használ a megrendelésekhez, a szállítás megerősítésére stb., melyeket a megrendeléskezelő rendszer által használt formátumra



18.11. ábra. Egy COTS-alapú beszerzési rendszer

kell konvertálni. Az e-kereskedelmi rendszer már rendelkezik beépített integrációval az e-mail rendszerhez, hogy a felhasználóknak értesítéseket küldhessen, de a megrendeléskezelő rendszer nem így lett tervezve. Ezért egy újabb adapter szükséges, ami az értesítéseket e-mail üzenettké konvertálja.

Elviekben egy nagy COTS-rendszer használata ugyanaz, mint egy bármilyen más specifikus komponens használata. Meg kell érteni a rendszer interfészeit, és a komponenssel történő kommunikációhoz kizárólag ezt szabad használni; kompromisszumokra kell jutni a specifikus követelmények és a gyors fejlesztés és újrafelhasználás között; a COTS-rendszerek együttműködését lehetővé tevő rendszerarchitektúrát kell tervezni.

Azonban az a tény, hogy ezek a termékek általában önálló létjogosultsággal rendelkező, nagy rendszerek, és hogy ezeket gyakran önálló rendszerekként adják el, további problémákat vet fel. Boehm és Abts (Boehm és Abts, 1999) a COTS-rendszerek integrációjával kapcsolatban négy problémát tárgyal:

1. *A funkcionalitás és a teljesítmény felügyeletének hiánya.* Habár úgy tűnhet, hogy egy termék közzétett interfésze biztosítja a szükséges szolgáltatásokat, de lehetséges, hogy ezek nem megfelelően vannak implementálva, vagy rossz teljesítményi mutatókkal rendelkeznek. A terméknek lehetnek olyan rejtett műveletei, amelyek egy speciális esetben zavarhatják a működését. Ezeknek a problémáknak a kijavítása elsődleges lehet a COTS-terméket integrálónak, de nem a termék szállítójának, akinek ezek nem jelentenek valódi gondot. A felhasználóknak egyszerűen meg kell kerülniük a problémákat, ha a COTS-terméket szeretnék újrafelhasználni.
2. *A COTS-rendszerek együttműködési képességeinek problémái.* A COTS-termékeket néha nehéz együttműködésre bírni, mivel minden termék különféle feltevésekkel él arról, hogy őt hogyan kell használni. Garlan és mások (Garlan és mások, 1995) négy COTS-terméket próbáltak integrálni, és azt tapasztalták, hogy ezek közül három eseményalapú, de mindegyik különböző eseménymodellt

használ, és mind feltételezi, hogy kizárólagos hozzáféréssel rendelkezik az események várakozási sorához. Ennek következményeként a projekt az előre jelzethez képest ötszörös erőfeszítést igényelt, és lefutási ideje az előre jósolt hat hónap helyett két év volt.

3. *Ellenőrizhetetlen rendszerfejlődés.* A COTS-termékek szállítói a piaci nyomásnak eleget téve, saját maguk döntenek a rendszer változtatásáról. Különösen a PC-s termékek esetén sokszor nagyon sűrűn jönnek ki az újabb változatok, amelyek esetleg nem is kompatibilisek az előző verziókkal. Lehet, hogy az újabb változatok nem kívánt funkciókat tartalmaznak, a korábbi verziók pedig elérhetetlenné és nem támogatottá válhatnak.
4. *A COTS-szállítók által nyújtott támogatás.* A COTS-szállítók által biztosított támogatás erősen változó. Mivel ezek kulcsrakész rendszerek, a szállítók által nyújtott támogatás különösen fontos, ha problémák merülnek fel, hiszen a fejlesztők sem a rendszer forráskódjához, sem pedig részletes dokumentációjához nem férhetnek hozzá. Míg a szállítók kötelezettséget vállalhatnak a támogatás biztosítására, a piaci és gazdasági körülmények megbonyolíthatják ennek a kötelezettségvállalásnak a teljesítését. Például egy COTS-rendszert szállító cég úgy dönthet, hogy a korlátozott igény miatt felhagy egy termékkel, vagy ezt a céget átveheti egy másik cég, amely nem akar minden jelenlegi termékhez támogatást nyújtani.

Természetesen valószínűtlen, hogy minden esetben valamennyi probléma felerősül, de becsléseink szerint a legtöbb COTS-integrációs projektben legalább az egyikre számítani kell. Ebből az következik, hogy a COTS-újrakelhasználásból származó – a költséget és az ütemezést érintő – haszon kisebb, mint ahogyan az a kezdeti optimista elemzésekből kitűnhet.

Boehm és Abts ráadásul úgy véli, hogy COTS-termékek használata esetén sok esetben nagyobb lehet a rendszer karbantartásának és evolúciójának a költsége. A fent említett nehézségek mindegyike életciklus-probléma, és nem csak a rendszer fejlesztésének kezdeti szakaszát befolyásolják. A rendszert karbantartó emberek minél jobban eltávolodnak az eredeti rendszer fejlesztőitől, annál valószínűbb, hogy az integrált COTS-termékekkel való problémák merülnek fel.

A problémák ellenére a COTS-termékek újrafelhasználásának előnyei nagyon lényegesegek, mert ezek a rendszerek nagyon sok funkciót biztosítanak az újrafelhasználóknak. Egy létező rendszer újrafelhasználásával implementációval töltött hónapokat, sőt néha éveket takaríthatunk meg, és így drasztikusan csökkenthető a rendszerek fejlesztési ideje. A 18.11. ábrán leírt beszerzési rendszer például egy nagyon nagy vállalat esetében kilenc hónap alatt implementálható és kihelyezhető, szemben az eredetileg új rendszer tervezésére kalkulált három évvel. Mivel a rendszerek gyors leszállítása sokak számára kulcsfontosságú követelmény, az újrafelhasználásnak ezt a válfaját valószínűleg egyre szélesebb körben fogják alkalmazni.

18.5.2. Alkalmazáscsaládok

Az újrafelhasználás egyik leghatékonyabb szemléletmódja az alkalmazáscsaládok fogalma köré épül. Egy alkalmazáscsalád, vagy más néven termékvonallal, egymással kapcsolatban lévő, a 13. fejezetben leírt módon közös szakterület-specifikus architektúrával rendelkező alkalmazások csoportja. Az egyes alkalmazások azonban valamilyen módon specializáltak. Az új alkalmazásokat az alkalmazáscsalád közös magjának újrafelhasználásával készítik. Az új fejlesztés magával vonhatja komponensek speciális konfigurálását, néhány további komponens megírását, valamint a komponensek némelyikének az alkalmazás új igényeinek megfelelően történő átdolgozását.

Egy alkalmazáscsalád specializálódásának különféle módjai alakíthatók ki:

1. *Platformspecializáció*, ahol az alkalmazás különböző verzióit különféle platformokra készítik. Például egy alkalmazásnak létezhet Windows, Solaris és Linux platformokon futó változata is. Ebben az esetben normál körülmények között az alkalmazás funkcionalitása változatlan; csak a hardverrel és az operációs rendszerrel érintkező komponensek változnak.
2. *Környezetspecializáció*, ahol az alkalmazás különböző verziói különféle működési környezeteket és perifériákat kezelnek. Például egy sürgősségi szolgáltatásokat ellátó rendszernek különböző változatai lehetnek, a használt rádiós rendszer típusától függően. Ebben az esetben a rendszerkomponensek a használt kommunikációs felszerelés funkcionalitásának megfelelően változnak.
3. *Funkcionális specializáció*, ahol az alkalmazás különböző verzióit eltérő követelményekkel hozzák létre. Például egy könyvtári rendszer attól függően módosítható, hogy azt közkönyvtárban, kézikönyvtárban vagy egyetemi könyvtárban használják-e. Ebben az esetben a funkciókat megvalósító komponensek is megváltozhatnak, és új komponensek is bekerülhetnek a rendszerbe.
4. *Folyamatspecializáció*, ahol a rendszert speciális üzleti folyamatok érdekében adaptálják. Például egy megrendeléskezelő rendszer az egyik vállalatnál egy központosított megrendelési folyamat, míg egy másik vállalat esetén egy elosztott folyamat megoldására adaptálható.

Az alkalmazáscsaládok tervezésénél szempont az újrakonfigurálhatóság. Ebbe az újrakonfigurálhatóságba beletartozik a komponensek elvétele vagy éppen hozzáadása, a rendszerkomponensek számára paraméterek és megszorítások meghatározása, és az üzleti folyamatok ismeretének bevonása. Az alkalmazáscsaládok a fejlesztési folyamat során két ponton konfigurálhatók:

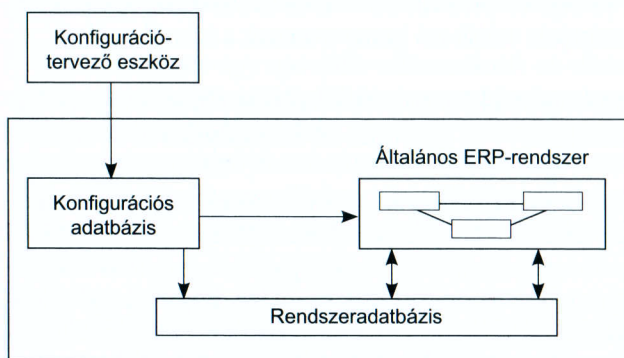
- *Kihelyezési idejű konfigurálás*, amikor az általános rendszer úgy lett tervezve, hogy a konfigurálást a vevő, vagy a vevőnek dolgozó konzultáns végezhesse el. A vevő specifikus követelményei és a rendszer működési környezetének ismeretei konfigurációs állományokban kerülnek tárolásra, melyet majd az általános rendszer felhasznál.

- *Tervezési idejű konfigurálás*, amikor a szoftvert fejlesztő szervezet az alkalmazáscsalád általános magját módosítja további fejlesztéssel, illetve komponensek kiválasztásával vagy adaptálásával, hogy egy új rendszert hozzon létre a vevő számára.

A kihelyezési idejű konfigurálás a vertikális szoftvercsomagok esetében használatos megközelítés, amelyeket speciális alkalmazások, mint például kórházi információkezelő rendszerek számára terveztek. Szintén használatosak az ERP-rendszerekben (O'Leary, 2000) is, mint amilyeneket az SAP vagy a BEA készített. Ezek olyan durva szemcsézettségű integrált rendszerek, amelyeket arra terveztek, hogy az üzleti folyamatokat (például megrendelés és számlázás, leltárkezelés és gyártásütemezés) támogassák. A konfigurálási folyamat magában foglalja a vevő piacának és üzleti folyamatainak részletes információgyűjtését, amelyet ezután a konfigurációs adatbázisban kell eltárolni. Ehhez többnyire a konfigurációs jelöléseknek és eszközöknek a részletes ismerete szükséges, ezért általában ezt a vevővel együttműködő konzultánsok végzik. A 18.12. ábra egy ERP-rendszer szerkezetét mutatja.

Az általános ERP-rendszerek nagy számban tartalmaznak modulokat, amelyeket különböző módon lehet összeilleszteni egy specifikus rendszer létrehozására. A konfigurációs folyamat magában foglalja a szükséges modulok kiválasztását, ezen egyedi modulok konfigurálását, az üzleti folyamatok és üzleti szabályok definiálását, valamint a rendszeradatbázis struktúrájának és szervezésének meghatározását.

Az ERP-rendszerek talán a legszélesebb körben használt esetei a szoftver-újrafelhasználásnak. A nagy vállalatok többsége ezeket a rendszereket alkalmazza a feladatai egy részének vagy egészének támogatására. Mindazonáltal nyilvánvaló az a korlátozás is, hogy a rendszer funkcionalitása az általános mag funkcionalitására van korlátozva. Továbbá a vállalat folyamatait és tevékenységeit a rendszer konfigurációs nyelvén kell kifejezni, és itt előfordulhatnak hibás megfeleltetések az üzleti koncepciók és a konfigurációs nyelv által támogatott koncepciók között. Vegyük például azt az esetet, amikor egy ERP-rendszert eladnak egy egyetemnek, és az ügyfél fogalmát kell definiálni. Ez valós problémákat tud



8.12. ábra. Egy ERP-rendszer konfigurációja

okozni, mert az egyetemek többféle ügyféltípussal találkoznak (hallgatók, kutató-közvetítő ügynökség, oktatási alapok stb.), és ezek egyike sem hasonlítható össze az üzleti élet ügyfeleivel. A rendszer által használt üzleti modell és a vevő által használt modell közötti komoly eltérés nagyon valószínűvé teszi, hogy az ERP-rendszer nem fog megfelelni a vevő elvárásainak (Scott, 1999).

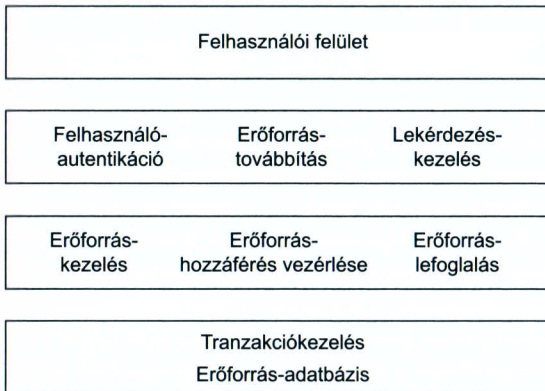
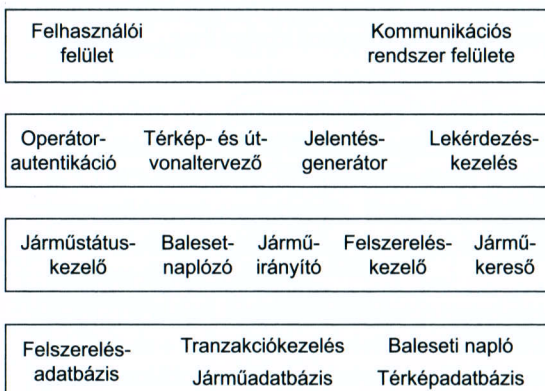
Az alkalmazáscsaládok újrafelhasználásának egyik alternatív megközelítése a rendszerszállító által történő konfigurálás, még mielőtt leszállítaná a vevőnek. A szállító egy általános rendszerből indul ki, ennek moduljait módosítja és bővíti, hogy létrehozza a vevő által elvárt funkcionalitásokkal rendelkező rendszert. Ez a megközelítés többnyire a rendszer magjában a forráskód megváltoztatásával és kibővítésével jár, így nagyobb flexibilitás érhető el, mint a kihelyezési idejű konfigurálással.

Az alkalmazáscsaládok többnyire létező alkalmazásokból fejlődnek ki. Azaz egy szervezet kifejleszt egy alkalmazást, és amikor egy új alkalmazásra van szüksége, akkor ezt használja az új alkalmazás alapjául. Az új alkalmazások iránti további igények pedig a folyamat folytatódását eredményezik. Mindazonáltal a változások az alkalmazás struktúrájának leromlása felé vezetnek, így bizonyos szinten megfogalmazódhat a döntés egy általános alkalmazáscsalád tervezésére. Ez a döntés a kiindulási alkalmazások fejlesztése során összegyűjtött ismeretek újrafelhasználásán alapszik.

Az alkalmazáscsaládokra úgy is gondolhatunk, mint általánosabb alkalmazásarchitektúráknak a példányosítására és specializálására, ahogy azt már a 13. fejezetben kifejtettem. Az alkalmazásarchitektúra eléggé általános, az alkalmazáscsaládok pontosítják az architektúrát egy alkalmazás specifikus típusára. Gondoljunk például egy olyan alkalmazáscsaládra, amelyet sürgősségi szolgálatok járműirányításának kezelésére terveztek. A rendszer kezelői fogadják a hívásokat a balesetekről, megkeresik az erre legalkalmasabb járművet, majd a járművet a baleset helyszínére irányítják. Az ilyen típusú rendszerek fejlesztői elkészíthetik ennek a változatait a rendőrség, a tűzoltóság és a mentőszolgálat számára is.

Ez a járműirányító rendszer alkalmas példa az erőforrás-kezelő rendszerekre, melyeknek az alkalmazásarchitektúrája a 18.13. ábrán látható. Ennek a négyrétegű architektúrának egy példányosítását láthatjuk a 18.14. ábrán, ami azon modulokat ábrázolja, amelyek egy járműirányító alkalmazáscsaládban szerepelhetnek. Az alkalmazáscsalád egyes szintjein található komponensek a következők:

1. A felhasználói felület szintjén olyan komponensek találhatók, amelyek felületet biztosítanak az operátori, illetve az alkalmazott kommunikációs rendszer felé.
2. Az I/O-menedzsment szintjén (2. szint) olyan komponensek találhatók, amelyek kezelik az operátor autentikációját, jelentéseket készítenek a balesetekről és járművezénylésekről, támogatják a térképek kirajzolását és az útvonaltervezést, és lehetőséget biztosítanak az operátorok számára, hogy lekérdezhessék a rendszeradatbázisokat.
3. Az erőforrás-menedzsment szintjén (3. szint) olyan komponensek találhatók, amelyek lehetővé teszik a járművek tartózkodási helyének behatárolását és

**18.13. ábra.** Erőforrás-allokációs rendszer architektúrája**18.14. ábra.** Egy járműirányító rendszer alkalmazáscsaládja

vezénylését, a járművek és a felszerelések státusának módosítását és a balesetek részleteinek naplózását.

4. Az adatbázis szintjén, valamint a szokásos tranzakciókezelésnél különálló adatbázisok vannak a járművek, a felszerelés és a térképek kezelésére.

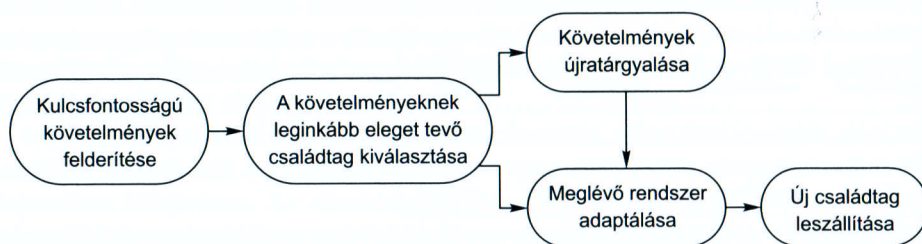
Ezen rendszer egy speciális változatának az elkészítéséhez az egyes komponenseket kell módosítani. Például a rendőrségnek sok autója van, de kevés járműtípussal, míg a tűzoltóság esetében számos speciális típusú jármű létezik, így a rendszerben eltérő járműadatbázis-struktúra bevezetésére lehet szükség.

A 18.15. ábrán egy alkalmazáscsalád új alkalmazás létrehozásának céljából történő átdolgozásának lépései láthatók. Az általános folyamat lépései a következők:

1. *A kulcsfontosságú követelmények felderítése.* Ez egy hagyományos követelménytervezési folyamaton alapszik. Mivel azonban valamilyen rendszer már létezik, ez normál körülmények között magával vonja a rendszerrel történő szem-

léltetést és kísérletezgetést, valamint a követelmények szükséges módosítások formájában történő kifejezését is.

2. *A követelményeknek leginkább eleget tevő családtag kiválasztása.* A követelmények elemzését követően az ezekhez legközelebb álló családtagot kell kiválasztani módosításra. Ez nem szükségképpen az a rendszer, amelyen a rendszert szemléltettük.
3. *A követelmények újratárgyalása.* Ahogyan a létező rendszeren szükséges változtatások egyre több részlete válik ismertté és a projektterv elkészül, a szükséges változtatások minimalizálása érdekében bizonyos követelmények újratárgyalása válhat szükségessé.
4. *A meglévő rendszer adaptálása.* A már létező rendszerhez új modulokat fejlesztünk, a létező rendszermodulokat pedig a követelményeknek megfelelően átdolgozzuk.
5. *Az új családtag leszállítása.* Az alkalmazáscsalád új tagját leszállítjuk az ügyfélnek. Ebben a szakaszban dokumentálni kell a legfontosabb jellemzőket, és ez a jövőben több rendszer fejlesztésének is alapjául szolgálhat.



18.15. ábra. Alkalmazáscsalád elemének fejlesztése

Egy alkalmazáscsalád új tagjának létrehozásakor kompromisszumot kell találnunk az általános alkalmazás lehető legnagyobb mértékű újrafelhasználása és a kulcsfontosságú részletes követelmények kielégítése között. Minél részletesebbek a rendszerkövetelmények, annál kevésbé valószínű, hogy a létező komponensek megfelelnek ezeknek a követelményeknek. Azonban ha a kulcsfigurák eléggé rugalmasak, és hajlandók a rendszer módosításait a szükségesre korlátozni, akkor a rendszert általában gyorsabban és kisebb költségráfordítással lehet leszállítani.

Általánosságban, az alkalmazásfejlesztés azon módja, amikor egy alkalmazás általános verzióját adaptáljuk, azt jelenti, hogy az alkalmazás kódjának nagyon nagy részét újrafelhasználjuk. Továbbá az alkalmazással szerzett tapasztalat gyakran átvihető egyik rendszerről a másikra, így amikor egy szoftverfejlesztő csatlakozik a fejlesztőcsoporthoz, akkor a betanulási folyamata rövidül. A tesztelés is egyszerűsödik, mert az alkalmazás nagy részén a tesztek szintén újrafelhasználhatók, csökkentve az alkalmazás teljes fejlesztési idejét.

Kulcsfogalmak

- A szoftver újrafelhasználásának előnyei: alacsonyabb költségek, gyorsabb szoftverfejlesztés és alacsonyabb kockázatok. A rendszer megbízhatósága nő, és hatékonyabban lehet a szakembereket úgy alkalmazni, hogy szakértelmüket újrafelhasználható komponensek tervezésénél kamatoztathassák.
- A tervezési minták a sikeres tervezési megoldásokat dokumentáló magas szintű absztrakciók, az objektumorientált fejlesztésben a terv újrafelhasználásának alapvető eszközei. Egy minta leírásának tartalmaznia kell a minta nevét, a probléma és a megoldás leírását és a minta használatának eredményeiről és kompromisszumairól készült kimutatást.
- A programgenerátor a koncepció újrafelhasználásának egy alternatív megközelítése, ahol az újrafelhasználható koncepciók beágyazódnak egy generátorrendszerbe. A tervező meghatározza a szükséges absztrakciókat egy szakterület-specifikus nyelv használatával, és ezután egy futtatható program legenerálódik.
- Az alkalmazás keretrendszer konkrét és absztrakt objektumok gyűjteménye, amelyeket úgy terveztek, hogy újrafelhasználhatók legyenek specializációval, valamint további, új objektumok hozzáadásával.
- A COTS-termékek újrafelhasználása a nagy, kulcsrakész rendszerek újrafelhasználásával foglalkozik. Ezek a rendszerek rengeteg funkciót biztosítanak, és újrafelhasználásuk nagymértékben csökkentheti mind a költségeket, mind pedig a fejlesztés idejét.
- A COTS-alapú újrafelhasználás potenciális problémái közé tartozik a funkcionalitás és a teljesítmény feletti és a rendszer-evolúció feletti kontroll hiánya, a külső szállítók támogatásának szükségessége, valamint a rendszerek együttműködésének biztosításából adódó nehézségek.
- A vállalati erőforrás-kezelő rendszerek széles körben használatosak. Specifikus ERP-rendszerek hozhatók létre egy általános rendszer kihelyezési idejű konfigurálásával, amivel a vevő üzletére vonatkozó információkat építjük be.
- Az alkalmazáscsaládok egymással kapcsolatban álló alkalmazások, amelyek egy vagy több alapalkalmazásból jönnek létre. Egy általános rendszert úgy dolgoznak át és szakosítanak, hogy az eleget tegyen a funkcionalitás, a célplatform és a működési konfiguráció speciális követelményeinek.

További irodalom

Reuse-based Software Engineering. Egy átfogó eszmefuttatás a szoftver-újrafelhasználás különböző megközelítéseiről. A szerzők az újrafelhasználás technikai kérdéseit és az újrafelhasználási folyamat menedzselését is tárgyalják. (Mili, H. és mások, 2002, John Wiley & Sons.)

„A Lifecycle Process for the effective reuse of commercial off-the-shelf software”. Egy jó általános bevezető, amely a COTS-termékek szoftverfejlesztésben történő alkalmazásának előnyeit és hátrányait tárgyalja. (Braun, C. L., *Proc.*

Symposium on Software Reusability, Los Angeles, 1999, ACM Press. Elérhető az ACM digitális könyvtárban.)

Design Patterns: Elements of Reusable Object-oriented Software. Ez a szoftverminták-ról szóló eredeti kézikönyv, amely bevezette a széles tömegeket a szoftverminták fogalmába. (Gamma és mások, 1995, Addison-Wesley.)

„Aspect-oriented programming”. A CACM különszáma, amely számos cikket tartalmaz az aspektusorientált szoftverfejlesztésről. Kitűnő kiindulási pont a témában. (*Comm. ACM*, 44(10), October 2001.)

Feladatok

- 18.1. Mik a szoftver-újrafelhasználást akadályozó fő technikai és nem technikai tényezők? Saját tapasztalatából kiindulva, ön sok szoftvert használ újra? Ha nem, miért nem?
- 18.2. Magyarázza el, hogy a létező szoftver újrafelhasználásából adódó költség-megtakarítás miért nem egyszerűen az újrafelhasznált komponensek méretével arányos.
- 18.3. Adjon meg négy olyan körülményt, amelyek esetén nem javasolja a szoftver-újrafelhasználást.
- 18.4. A terv újrafelhasználásának miért hatékony formái a minták? Mik a hátrányai ennek a megközelítésnek?
- 18.5. Az eddig érintett alkalmazások szakterületén kívül javasoljon még két olyan területet, ahol a generátoralapú újrafelhasználás sikeres lehet. Magyarázza el, hogy miért gondolja az újrafelhasználás ezen megközelítését költséghatékonynak ezeken a szakterületeken.
- 18.6. Magyarázza el, hogy általában miért szükségesek adapterek, ha a rendszert COTS-termékek integrációjával hozzuk létre.
- 18.7. Soroljon fel hat lehetséges kockázatot, amelyek akkor jelentkezhetnek, ha a rendszert COTS-termékek segítségével hozzuk létre. Milyen lépéseket tehet egy vállalat ezen kockázatok csökkentésének érdekében?
- 18.8. Egy általános információs rendszer architektúrát kiindulási pontként tekintve (13. fejezet) tervezzen egy alkalmazás családot könyvtári információs rendszerek részére, amelyet fel lehet használni könyv, film, zene és újság könyvtárakban.
- 18.9. A 14. fejezetben leírt meteorológiai állomás rendszere példájának felhasználásával javasoljon architektúrát egy távfelügyelettel és adatgyűjtéssel foglalkozó alkalmazás család számára.
- 18.10. A szoftver újrafelhasználása számos szerzői jogi és szellemi tulajdoni kérdést vet fel. Ha egy ügyfél fizet egy szoftver szállítójának egy rendszer kifejlesztéséért, akkor kinek van joga az ily módon kifejlesztett kód újrafelhasználására? Van-e joga a szoftver szállítójának ezt a kódot egy általános komponens alapjául használnia? Milyen fizetési mechanizmusok használhatók az újrafelhasználható komponenseket szállítók díjazására? Gondolja át ezeket és a szoftver újrafelhasználásával kapcsolatos egyéb erkölcsi kérdéseket is.