

29 KONFIGURÁCIÓKEZELÉS

TÉMAKÖRÖK

A fejezet célja, hogy bemutassa az evolúciós szoftverrendszerek kódjának és dokumentációjának kezelési folyamatát. A fejezet elolvasása után:

- megértjük, miért fontos a szoftver-konfigurációkezelés;
- megismerjük, hogy mi a négy legfőbb konfigurációkezelési tevékenység – a konfigurációkezelés megtervezése, változtatáskezelés, verzió és kiadás kezelése, valamint a rendszerépítés;
- megértjük, hogyan használhatjuk a CASE-eszközöket a konfigurációkezelési folyamatok támogatására a konfigurációkezelés során.

TARTALOM

- 29.1. A konfigurációkezelés megtervezése
- 29.2. Változtatások kezelése
- 29.3. Verzió- és kiadáskezelés
- 29.4. Rendszerépítés
- 29.5. CASE-eszközök a konfigurációkezeléshez

A konfigurációkezelés eljárások és szabványok fejlesztése és alkalmazása a változó rendszertermékek kezelésénél. Ahogyan azt a 7. fejezetben kifejtettem, a rendszerkövetelmények mindig változnak a fejlesztés és a használat során, és ezeket a követelményeket be kell építeni a rendszer új verzióiba. A rendszer fejlődését kezelni kell, mert nagyon könnyű eltéveszteni, hogy melyik változás a rendszer melyik verziójába került be. A verziók tartalmazzák a változtatások terveit, a hibák javítását és a különböző hardverekhez és operációs rendszerekhez való illesztéseket. Több verzió lehet egyazon időben fejlesztés alatt és használatban. Ha nincs hatékony konfigurációkezelési eljárásunk, akkor helytelen rendszerverziót fogunk módosítani és átadni az ügyfélnek, és nem fogjuk tudni nyomon követni, hogy hol tárolódik a szoftver forráskódja.

A konfigurációkezelési eljárások definiálják, hogyan kell jegyzőkönyvezni és véghezvinni a javasolt rendszerváltoztatásokat; hogyan kell ezeket a rendszerkomponensekhez kapcsolni; illetve azokat a módszereket, amelyekkel a rendszer

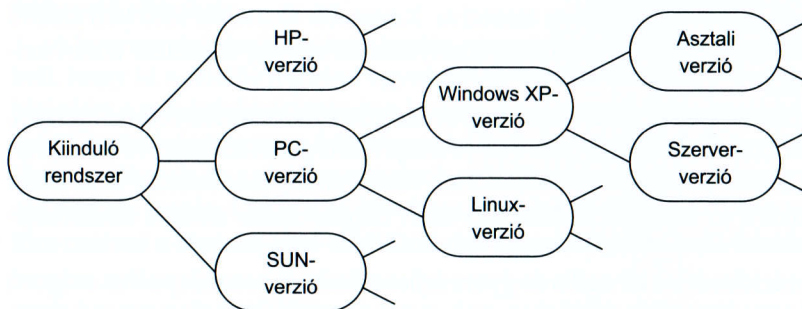
különböző verzióit azonosíthatjuk. A konfigurációkezelő eszközök arra valók, hogy tárolják a rendszerkomponensek verzióit, ezekből a komponensekből felépítsék a rendszert és nyomon kövessék a vásárlónak kiadott rendszerverziókat.

A konfigurációkezelést néha egy sokkal általánosabb folyamat, a rendszerminőség kezelése (27. fejezet) részének tekintik. A minőségkezelés és a konfigurációkezelés tartozhat egy vezető felelősségi körébe. A fejlesztők a szoftvert a minőség-ellenőrző csapatnak adják ki, amely ellenőrzi, hogy a rendszer elfogadható minőségű-e. A szoftver azután a konfigurációkezelő csapathoz kerül, amely felel a szoftver változtatásának irányításáért. Az irányított rendszereket néha *alaplavonálnak* nevezik, mivel ezek az irányított fejlődés kiindulópontjai.

Sok oka lehet annak, hogy miért létezik egy rendszernek több különböző konfigurációja. Különböző konfigurációk készülhetnek a különböző számítógépekre, különböző operációs rendszerekre, vagy kliensspecifikus funkciókat tartalmaznak és így tovább (29.1. ábra). A konfigurációkezelők felelőssége a különböző szoftververziók közti különbségek nyomon követése, annak biztosítása, hogy az új verziók irányított módon készüljenek el, és hogy az új verziókat a megfelelő időben, a megfelelő vásárlóknak adják ki.

A konfigurációkezelési szabványok definiálása és használata alapvető a minőségi tanúsítványok számára az ISO 9000, CMM és CMMI esetén egyaránt (Paulk és mások, 1995; Ahern és mások, 2001; Peach, 1996). Ilyen szabványra példa az IEEE 828-1998, amely a konfigurációkezelés tervezésének szabványát definiálja. Ezeket a szabványokat a szervezetben konfigurációkezelési kézikönyvként vagy minőségi kézikönyv részeként kell közzétenni. Külső szabványokat használhatunk a részletesebb szervezeti szabványok alapjául, amelyek már egy meghatározott környezetre vannak kialakítva.

A hagyományos szoftverfejlesztési folyamat során, amely a vízesésmodellt követi (4. fejezet), a szoftvert azután továbbítják a konfigurációkezelő csapathoz, miután a fejlesztés befejeződött, és az egyes szoftverkomponenseket tesztelték. Így erre a csapatra hárul a teljes rendszer építésének és tesztelésének felelőssége. A rendszer tesztelése során felfedezett hibák visszakerülnek a fejlesztőcsapathoz javításra. Ők kijavítják a hibát, majd a javított komponensek új verzióját kiadják a konfigurációkezelő csapatnak. Ha a minőség megfelelő, akkor ez lehet a további rendszerfejlesztés alapvonala.



29.1. ábra. Rendszercsaládok

Ez a modell, ahol a konfigurációkezelő csapat irányítja a rendszer-integrációs és tesztelési folyamatokat, alapvetően befolyásolja a konfigurációkezelési szabványok fejlesztését. A legtöbb konfigurációkezelési szabvány azon a feltételezésen alapszik, hogy a rendszerfejlesztésnél a vízesésmodellt használják (Bersoff és Davis, 1991). Ez azt jelenti, hogy a szabványokat adaptálni kell a modern, inkrementális specifikáción és fejlesztésen alapuló szoftverfejlesztési megoldásokra. Hass (Hass, 2003) tárgyalja ezeket az adaptációkat a szoftverfejlesztési folyamatokra, köztük az agilis fejlesztésre is.

Az inkrementális fejlesztés támogatására egyes szervezetek a konfigurációkezelés egy módosított megközelítését fejlesztették ki, amely támogatja a konkurens fejlesztést és a rendszertesztelést. Ez a megközelítés azon alapszik, hogy a teljes rendszert nagyon gyakran (akár naponta) felépítsék a komponenseiből:

1. A fejlesztőszervezet kitűz egy időpontot (például délután 2 óra), amikor a rendszer komponenseit ki kell adni. Ha a fejlesztőknek az általuk írt komponensekből új verziójuk van, akkor azt ki kell adniuk a megadott időpontra. A komponensek lehetnek befejezetlenek, de biztosítaniuk kell egyes alapfunkciókat, amelyeket tesztelni lehet.
2. Ezeknek a komponenseknek a fordításából és szerkesztéséből felépül a rendszer új verziója.
3. Ezt a rendszert továbbadják a tesztelőcsapatnak, akik előre meghatározott teszteket hajtanak végre. Ez idő alatt a fejlesztők dolgoznak a saját komponenseiken, új funkciókkal egészítik ki azokat, és javítják az előző tesztelés során felfedett hibákat.
4. A rendszer tesztelése során felfedezett hibákat dokumentálják és eljuttatják a rendszerfejlesztőknek. Ők a komponens egy későbbi verziójában kijavítják ezeket a hibákat.

A naponkénti szoftverépítés fő előnye, hogy megnő annak az esélye, hogy a fejlesztés korai szakaszában megtalálják a komponensek kapcsolatából származó problémákat. Továbbá a naponkénti építés a komponensek alapos egységtesztelésére buzdít. A fejlesztőkre lélektani nyomás nehezedik, hogy „ne szakítsák meg az építést”, például a komponens olyan verziójának kiadásával, amely az egész rendszer összeomlását eredményezi. Ezért nem adják ki a komponensük olyan verzióját, amely nincs megfelelően tesztelve. Kevesebb tesztelési időt kell szánni az olyan hibák keresésére és javítására, amelyeket az egység tesztelése során kelne megtalálni.

A napi építés sikeres alkalmazása nagyon szigorú változtatáskezelést igényel, hogy nyomon tudjuk követni a felfedett és a kijavított problémákat. Továbbá nagyon sok rendszer- és komponensverzióhoz vezet, amelyeket kezelni kell. Ezért elengedhetetlen a jó konfigurációkezelés ezen megközelítés sikeres alkalmazásához.

A konfigurációkezelés az agilis és gyors fejlesztéseknél nem alapulhat szigorú eljárásokra és papírmunkára. Jóllehet ezek szükségesek lehetnek a nagy, komplex projekteknel, de lelassítják a fejlesztési folyamatot. A gondos nyilvántartás

alapvető a nagy, komplex, több honlapot is felölelő rendszereknél, de felesleges a kis projektekénél. Ezekben a projektekben minden csapattag együtt, ugyanabban a szobában dolgozik, és az állandó feljegyzéskészítés fölöslegesen lelassítja a fejlesztési folyamatot. Természetesen ez nem jelenti azt, hogy a konfigurációkezelés teljesen kihagyható akkor, amikor gyors fejlesztés szükséges. Ehelyett az agilis módszer olyan egyszerű konfigurációkezelési eszközöket használ (például verziókezelés és rendszerépítő eszközök), amelyek kikényszerítenek egyfajta ellenőrzést. Minden csapattag megtanulja ezen eszközök használatát, és igazodik az általa sugallt elvekhez.

29.1. A KONFIGURÁCIÓKEZELÉS MEGTERVEZÉSE

A konfigurációkezelési terv leírja azokat a szabványokat és folyamatokat, amelyeket a konfigurációkezelés során alkalmazunk. A terv fejlesztésének kiindulópontja néhány általános, vállalati szintű konfigurációkezelési szabvány, és szükség szerint ezeket kell alkalmazni minden egyes projektre. A konfigurációkezelési tervet fejezetekre kell taglalni, és tartalmaznia kell a következőket:

1. azon egyedek definícióit, amelyeket kezelünk, és egy formális módszert, amellyel ezeket az egyedeket azonosíthatjuk;
2. nyilatkozatot arról, hogy ki a felelős a konfigurációkezelési folyamatokért és az irányított egyedek előterjesztéséért a konfigurációkezelő csapat felé;
3. a konfigurációkezelési politikákat, amelyek a változtatásvezérlés és a konfigurációkezelés során követendők;
4. a konfigurációkezelés során használt eszközök leírását és azokat a folyamatokat, amelyeket ezen eszközök használata során alkalmazunk;
5. a konfigurációs adatbázis szerkezetének leírását, amely a konfigurációs információkat tárolja, és azon információkat (a konfigurációs rekordokat), amelyeket az adatbázisban kezelni kell.

A konfigurációkezelési terv egyéb információkat is tartalmazhat, mint például a szoftver kezelése külső szállító által, vagy a konfigurációkezelési folyamatot felügyelő eljárások.

A konfigurációkezelési terv fontos része a felelősségek definiálása. Definiálni kell, hogy ki a felelős minden egyes dokumentum és szoftverkomponens továbbításáért a minőségbiztosítás és a konfigurációkezelés felé. Továbbá definiálhatja mindegyik dokumentum felülvizsgálatát. A dokumentumok továbbításáért felelős személynek és a dokumentumok előállításáért felelős személynek nem kell azonosnak lennie. Gyakran kényelmes a projektmenedzserre vagy a csapatvezetőre osztani a csapata által előállított dokumentumokért való felelősséget.

29.1.1. Konfigurációtétel azonosítása

Egy nagy szoftverrendszer fejlesztése során több ezer forráskódmodul, tesztszkript, tervdokumentum stb. készülhet. Ezek különböző emberektől származnak, és azonos vagy hasonló nevűek lehetnek. Ahhoz, hogy ezeket az információkat nyomon lehessen követni, és az állományok megtalálhatók legyenek, konzisztens azonosítási séma bevezetése szükséges a konfigurációkezelési rendszer minden eleme számára.

A konfigurációkezelés megtervezése közben el kell döntenünk, hogy pontosan mely tételeket (vagy tételek osztályát) fogjuk szabályozni. A konfigurációkezelés alatt álló dokumentumok vagy összetartozó dokumentumok csoportjai, a formális dokumentumok vagy a konfigurációs tételek. A projektterveket, specifikációkat, terveket, programokat és tesztadatokat általában konfigurációs tételként kezeljük. Azonban minden olyan dokumentumot kontrollálni kell a konfigurációkezelési rendszerben, amely szükséges lehet a későbbi rendszer-evolúció során.

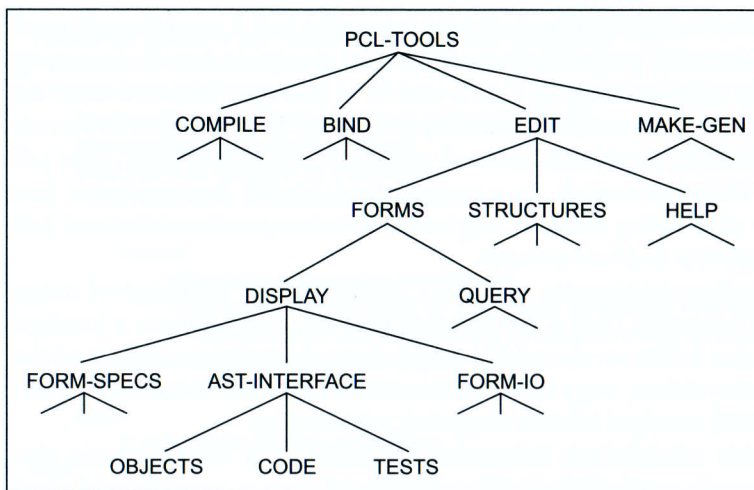
Ez azonban nem jelenti azt, hogy minden előállított dokumentumot és állományt be kell vonni a konfigurációkezelésbe. A technikai munkadokumentumok, amelyek a további fejlesztési elképzelések pillanatfelvételei, a csoporttalálkozó percek, a tervvázlatok és -javaslatok stb. nem rendelkeznek hosszú távú kihatással, és nem szükségesek a rendszer jövőbeli karbantartásához.

A dokumentumok elnevezési módjának egyéni nevet kell rendelnie a konfigurációkezelésnek alávetett dokumentumokhoz. Ennek a névnek tükröznie kell a tétel típusát, a rendszer azon részét, ahol alkalmazzák, a tétel létrehozóját stb. Az elnevezési sémánkban a tételek közötti kapcsolatokat rögzíthetjük úgy, hogy a kapcsolatban álló dokumentumok közös ősőtől kapják a nevüket. Ez hierarchikus elnevezési rendszert eredményez, amelyben a következőkhöz hasonló nevek jelennek meg:

```
PCL-TOOLS/EDIT/FORMS/DISPLAY/AST-INTERFACE/CODE  
PCL-TOOLS/EDIT/HELP/QUERY/HELPPFRAMES/FR-1
```

A név kezdete a projekt neve, PCL-TOOLS. Ebben a projektben több különböző eszközt fejlesztünk, így az eszköz nevét (EDIT) a név következő része jelöli. Minden eszköz különböző névvel rendelkező modulokból épül fel, amelyek a tételazonosító következő komponensét adják (FORM, HELP). Ez a dekompozíciós folyamat addig tart, amíg el nem jutunk az alapszintű formális dokumentumokig (29.2. ábra). A dokumentációs hierarchia levelei a formális projektdokumentumok. A 29.2. ábra mutatja, hogy minden egyes szabályozott tételhez három formális dokumentum szükséges. Ezek az objektumleírás (OBJECTS), a komponens kódja (CODE) és tesztek az adott kódhoz (TESTS). Az olyan tételek, mint a sugórendszer, szintén kezelendők, és eltérő nevet kapnak (FR-1 a második példában).

A hierarchikus elnevezési séma egyszerű és könnyen érthető, és gyakran leképezi a projekt állományait tároló könyvtárszerkezetet, ugyanakkor a szoftvert ki-



29.2. ábra. Egy konfigurációhierarchia a tételazonosítók megadásához

fejlesztő projekt struktúráját tükrözi. Az azonosítók a komponenseket egy adott projekthez kötik. Ez csökkenti az újrafelhasználhatóságot. Nagyon nehéz megtalálni az összetartozó komponenseket (például az egy programozó által fejlesztetteket), ha a kapcsolatot nem tükrözi a tételnevezési séma.

29.1.2. A konfigurációs adatbázis

A konfigurációs adatbázis tárolja a konfigurációkhoz kapcsolódó összes információt. Elsődleges funkciója támogatni a rendszerváltoztatások hatásainak becslését, és információkat szolgáltatni a konfigurációkezelésről. A konfigurációs adatbázis sémájának definiálása mellett a projekt információinak rögzítését és visszanyerését is definiálni kell a konfigurációkezelés tervezése során.

Egy konfigurációs adatbázis nem tartalmaz információt a konfigurációs tételekről. Viszont tárolja az olyan információkat, mint a komponensek felhasználói, ügyfelek, végrehajtási platform, javasolt változtatások stb. Képesnek kell lennie különféle lekérdezésekre választ adni a rendszer-konfigurációval kapcsolatban. Tipikus lekérdezések lehetnek a következők:

1. Mely ügyfelekhez juttatták el a rendszer egy meghatározott verzióját?
2. Milyen hardver és operációs rendszer szükséges egy adott rendszerverzió futtatásához?
3. A rendszer hány verziója készült el, és mikor?
4. A rendszer mely verzióit érinti egy adott komponens megváltozása?
5. Hány nem végrehajtott változtatási kérelem van egy adott verzióra?
6. Hány bejegyzett hiba van egy adott verzióban?

Ideális esetben a konfigurációs adatbázist integrálni kell a verziókezelő rendszerrel, amely a formális projektdokumentumokat tárolja és kezeli. Ez a megközelítés, amelyet néhány integrált CASE-eszköz is támogat, lehetővé teszi azt, hogy a változtatásokat közvetlenül összekapcsoljuk azokkal a dokumentumokkal és komponensekkel, amelyekre hatással vannak. A dokumentumok, mint például a tervezési dokumentumok, és a programkód közötti kapcsolatokat fenn lehet tartani, így viszonylag könnyű megtalálni mindent, amit módosítani kell, amikor egy változtatást indítványozunk.

Azonban a konfigurációkezelés integrált CASE-eszközök segítségével drága. Sok vállalat nem használja őket a konfigurációkezelés során, hanem a konfigurációs adatbázisukat külön rendszerként tartják fenn. A konfigurációs egységeket tárolhatják állományokban, vagy egy verziókezelő rendszerben, mint amilyen a CVS (Berliner, 1990), amelyet később tárgyalok a fejezetben.

A konfigurációs adatbázisok információkat tárolnak a konfigurációs tételekről, és hivatkozzák a neveiket az állománykezelő vagy a verziókezelő rendszerben. Ugyan ez viszonylag olcsó és rugalmas megközelítés, hátránya, hogy a konfigurációs tételeket meg lehet változtatni a konfigurációs adatbázis érintése nélkül. Ezért nem lehetünk biztosak abban, hogy a konfigurációs adatbázis naprakész információkat tartalmaz a rendszer állapotáról.

29.2. VÁLTOZTATÁSOK KEZELÉSE

A változás elkerülhetetlen a nagy szoftverrendszerek életében. Mint azt a korábbi fejezetekben tárgyaltam, a szervezeti igények és követelmények változnak a szoftver élettartama alatt. Ez a szoftver megfelelő változtatásait vonja maga után. Ahhoz, hogy a rendszerben a változtatások ellenőrzött módon menjenek végbe, egy eszköztámogatással rendelkező változtatáskezelő eljárásrendszer szükséges.

A változtatáskezelő eljárások a javasolt változtatások költségeinek és előnyeinek elemzésével kezdődnek, megadva az érdemleges változtatásokat és nyomon követve azt, hogy a rendszer mely komponensei változnak. A változtatáskezelő folyamat (29.3. ábra) akkor lép hatályba, amikor a szoftver vagy a kapcsolódó dokumentáció a konfigurációkezelő csapat irányítása alá kerül.

A változtatáskezelő folyamat első szakasza egy változtatásigénylő űrlap kitöltése, amelyben az igénylő közzéteszi az igényelt változtatást. Mivel a változtatásokat fel kell jegyezni, az űrlap tartalmazza a változtatásokra vonatkozó ajánlásokat, a változtatás becsült költségét, valamint a változtatás igénylésének, jóváhagyásának, implementálásának és validálásának dátumait. Tartalmazhat továbbá egy bejegyzést, amelyben az elemző vázolja a változtatás implementálásának módját.

A 29.4. ábrán példa látható egy részben kitöltött változtatásigénylő űrlapra. A változtatásigénylő űrlapot általában a konfigurációkezelés tervezési folyamata során definiálják. Ez egy olyan változtatásigénylő űrlap, amely egy nagyon összetett rendszertervezési projektben használható. Azt ajánlom, hogy a változ-

```

Változtatás igénylése egy változtatásigénylő űrlap kitöltésével
A változtatási kérelem elemzése
if a változtatás valós then
    Megbecsülni a változtatás módjának implementálását
    Megbecsülni a változtatás költségét
    Bejegyezni az igénylést az adatbázisba
    Átadni a kérést a változtatást irányító testületnek
    if a változtatást elfogadták then
        repeat
            változtatások elvégzése a szoftveren
            változtatások regisztrálása és a megfelelő változtatási kérelemhez csatolása
            a megváltoztatott szoftver minőségi jóváhagyásra való előterjesztése
        until a szoftver minősége megfelelő
        új verziójú rendszer készítése
    else
        a változtatási kérelem elutasítása
    else
        a változtatási kérelem elutasítása

```

29.3. ábra. A változtatáskezelő folyamat

Változtatásigénylő űrlap

Projekt: Proteus/PCL-Tools **Szám:** 23/02
Változtatást igénylő: I. Sommerville **Dátum:** 1/12/02
Az igényelt változtatás: A struktúra egy komponensének kiválasztásakor jelenjen meg a tárolóállomány neve.

Változtatáselemző: G. Dean **Elemzés dátuma:** 10/12/02
Érintett komponensek: Display-Icon.Select, Display-Icon.Display

Egyéb komponensek: FileTable

Változtatási javaslat: Viszonylag egyszerű implementálni, mivel az állománytábla rendelkezésre áll. Egy megjelenítő mező tervezését és implementálását igényli. A kapcsolódó komponensek változtatását nem igényli.

Változtatás prioritása: Alacsony

Változtatás implementálása:

Becsült ráfordítási idő: 0,5 nap

CCB-hez kerülés dátuma: 15/12/02

CCB döntés dátuma: 1/2/03

CCB döntése: Változtatás elfogadva. A 2.1-es kiadásban kell implementálni.

Változtatásimplementáló:

Változtatás dátuma:

QA elfogadási dátum:

QA döntése:

Konfigurációkezelő elfogadási dátum:

Megjegyzések:

29.4. ábra. Egy részben kitöltött változtatásigénylő űrlap

tatáskérés formálisan legyen leírva, de a változtatásigénylő űrlap kevésbé fókuszáljon a változtatási kérések implementációs vonatkozásaira. A változtatást végző mérnök döntsön a változtatás implementálásáról.

Miután egy változtatásigénylő űrlapot benyújtunk, elemezni kell, hogy a változtatás valós-e. Egyes változtatási kérelmek félreértésekből erednek, nem pedig rendszerhibákból; mások pedig már ismert hibákra utalnak. Ha az elemzés felfedi, hogy a változtatási kérelem nem valós, már ismert, vagy már egyszer át volt gondolva, a változtatást el kell utasítani. Az elutasítás indoklása el kell jusson a változtatást javasoló személyhez.

Az érvényes változtatások esetén a következő szakasz a változtatás mértékének és költségének becslése. Felül kell vizsgálni a változtatásnak a rendszerre gyakorolt hatását. Technikai elemzésre is szükség van annak megállapítására, hogyan implementáljuk a változtatást. Meg kell becsülnünk a változtatás költségeit és az esetlegesen szükséges további változtatásokat a rendszer többi részében. Ezt fel kell tüntetni a változtatásigénylő űrlapon. Ez a becslési folyamat felhasználhatja a konfigurációs adatbázist, amely a komponensek közötti kapcsolatokat is tartalmazza. Ezután mérlegelni lehet a változtatás hatásait a rendszer egyéb komponenseire.

Ha a változtatás nem csak kisebb dokumentációs, weboldal- vagy megjelenítési hibák javításával jár, akkor be kell nyújtani a változtatást irányító testülethez, ahol döntenek a változtatásról. A változtatást irányító testület a változtatás hatásait stratégiai és vállalati, nem pedig technikai szemszögből vizsgálja. Azt nézi meg, hogy a változtatás gazdaságilag indokolt-e, és szólnak-e szervezeti okok a változtatás elfogadása mellett.

A *változtatást irányító testület* elnevezés egy inkább nagyobb méretű csoportot takar, amelyik dönt a változtatásról. Egy formálisan is strukturált változtatást irányító testület (beleértve a tapasztalt klienseket és a vállalkozó személyzetet) a hadi projekteknél követelmény. Kis- és közép méretű projektek esetén a változtatást irányító testület egyszerűen egy projektmenedzserből és két vagy három tervezőből áll, akik nem vesznek részt közvetlenül a szoftverfejlesztésben. Egyes esetekben csak egy változtatás-felülvizsgálóból áll, aki véleményt mond arról, hogy a változtatás indokolt-e vagy sem.

Az általános célú szoftvertermékek, szemben azon rendszerekkel, amelyek egy speciális ügyfél számára készültek, egészen eltérő módon kezelendők. Ezekbe a rendszerekbe az ügyfél nincs közvetlenül bevonva, így az ügyfél üzleti változása nincs közvetlen ráhatással. A változtatási igény ilyenkor a szoftver kiadása utáni, az ügyfelek által végzett rendszertesztek során felfedezett hibákhoz kapcsolódik. Az ügyfelek a hibák bejelentéséhez használhatnak weboldalt vagy e-mailt. A hibakezelő csapat ellenőrzi a hibabejelentést, és ha az valós, átalakítja formális rendszerváltoztatás kéréssé. Ahogy más rendszereknél is, a változások elsődlegesen az implementáció szempontjából, de a hiba nem kerül kijavításra, ha a javítási költség túl magas.

Mikor a napi (vagy még gyakoribb) rendszerépítések során készül el a rendszer egy újabb verziója, akkor egyszerűbb változtatásirányító folyamatot alkalmazunk. A problémákat és változtatásokat továbbra is dokumentálni kell, de azokat a változtatásokat, amelyek csak egy-egy komponensen vagy modulon fejtik ki hatásukat, egyedileg lehet kezelni. Kerülhetnek közvetlenül a rendszerfejlesztőkhöz. Ők vagy elfogadják, vagy megindokolják, hogy miért nem tartják

```
// PROTEUS projekt (IST 6087)
//
// PCL-TOOLS/EDIT/FORMS/DISPLAY/AST-INTERFACE
//
// Tárgy: PCL-Tool-Desc
// Szerző: G. Dean
// Létrehozás dátuma: 2002. november 10.
//
// (c) Lancaster University 2002
//
// Változtatástörténet
// Verzió   Változtató   Dátum   Változás   Eredmény
// 1.0     J. Jones     2002/1/12   Fejléc hozzáadása   Konfigurációkezelőkhöz továbbítva
// 1.1     G. Dean     2003/4/9    Új mező             Változtatási kérelem R07/02
```

29.5. ábra. Komponens fejrész-információk

szükségesnek a változtatásokat. Ha a változtatás olyan rendszermodulokat érint, amelyeket különböző fejlesztőcsapatok tartanak karban, akkor azt értékelnie kell valamiféle változtatásirányító szakértőnek, aki eldönti, hogy kell-e implementálni a változtatást.

Egyes agilis módszertanokban (mint például az extrém programozás) az ügyfeleket közvetlenül bevonják a döntésbe, amikor a változtatás implementálásra kerül. Ha ők javasolnak egy változtatást a rendszerkövetelményekben, akkor együttműködnek azzal a csapattal, akik felméri a változtatás hatását, és azután döntenek, hogy a változtatás elsőbbséget élvez-e a tervezett eszközzel szemben a rendszer következő inkrementében. Ugyanakkor a szoftver tökéletesítését célzó változtatásokról szóló döntés a rendszeren dolgozó programozó hatáskörében marad. Az újratervezés, amikor is a rendszert folyamatosan tökéletesítjük, a fejlesztési folyamat szükséges része, és nem egy fölösleges valami.

A szoftverkomponensek változása során minden komponens változtatási dokumentációját kezelni kell. Ezt néha a komponens *származástörténetének* is nevezik. A legjobb módja az ilyen feljegyzések kezelésének, ha a komponensek szabványosított fejrészében helyezjük el (29.5. ábra). Ennek hivatkozniia kell a szoftverváltoztatáshoz tartozó változtatási kérelmet. Meg tudunk írni egy egyszerű szkriptet, amely végignézi minden komponenst, és előállítja a származástörténeti jelentést a komponens változtatásairól. Hasonlóan járhatunk el weblapoknál is. Publikált dokumentumoknál a változtatások megadása a verziókban, a dokumentum elején, külön lapon történik.

29.3. VERZIÓ- ÉS KIADÁSKEZELÉS

A verzió- és kiadáskezelés az a folyamat, amellyel a rendszer különböző verzióit és kiadásait követjük nyomon. A verziómenedzserek olyan eljárásokat találnak ki, melyek segítségével visszanyerhetők a rendszer különböző verziói, amikor

szükség van rájuk, és amelyekkel garantálható, hogy nem változtatjuk meg véletlenül azokat. Kapcsolatban állhatnak a vevőszolgálati személyzettel, hogy eldönthessék, mikor kell piacra dobni a rendszer újabb verzióját. A rendszer új verzióját a konfigurációkezelő személyzetnek kell előállítani, nem pedig a rendszerfejlesztőknek, még akkor is, ha az adott verzió nem kerül kiadásra. Ez könnyebbé teszi a konfigurációs adatbázis konzisztenciájának fenntartását, mivel csak a konfigurációkezelő személyzet változtathatja meg a verzióinformációkat.

A rendszer verziója a rendszer egy példánya, amely valamilyen módon különbözik a rendszer többi példányától. A rendszer új verziói rendelkezhetnek más funkciókkal, lehetnek teljesítménybeli különbségek, vagy egyes rendszerhibák már ki lehetnek javítva. Egyes verziók lehetnek funkcionálisan azonosak, de más hardver- vagy szoftverkonfiguráción futnak. Ha csak kis különbségek vannak a verziók között, akkor az egyiket néha a másik *variánsának* is nevezik.

A *kiadás* egy olyan verzió, amely eljut a vásárlókhoz. Minden kiadásnak vagy új funkciókat kell tartalmaznia, vagy új hardverplatformon kell működnie. Egy rendszernek mindig sokkal több verziója van, mint kiadása, mivel verziók szervezeten belüli tesztelés és fejlesztés során is keletkeznek, ezeket viszont sohasem adják ki.

A verziómenedzselés ma már mindig CASE-eszközök támogatásával folyik, mint ahogy azt a 29.5. alfejezetben tárgyalom. Ezek az eszközök tárolják az összes rendszerverziót, és vezérlik a rendszer komponenseinek elérését. Ezeket ki kell emelni a rendszerből, amikor meg akarjuk változtatni őket. Amikor vissza kerülnek a rendszerbe, egy új verzió jön létre a verziókezelő által adott néven. Mivel az eszközök alapvetően különböznek a biztosított eszközökben és a felhasználói felületben, az itt áttekintett általános verziókezelési alapelvek alkotják az alapját minden támogatóeszköznek.

29.3.1. Verzióazonosítás

Egy rendszer egy önálló verziójának létrehozásához meg kell adni a tartalmazott rendszerkomponens-verziókat. Egy nagy szoftverrendszer több száz szoftverkomponensből állhat, amelyek mindegyikének több verziója is létezhet. A verziókezelés eljárásainak olyan módszert kell használnunk, amellyel egyértelműen azonosítható minden komponens minden verziója. A komponensek speciális verzióit visszaállíthatjuk, ha további változtatásokat akarunk rajtuk végrehajtani.

Három alaptéchnika létezik, amelyeket a komponensek azonosítására használhatunk:

1. *Verziószámozás.* A komponenshez egy explicit, egyedi számot rendelünk. Ez a leggyakrabban használt azonosítási technika.
2. *Attribútumalapú azonosítás.* Minden komponensnek van egy neve (amely nem egyedi a különböző verziókat tekintve), és egy társított attribútumhalmaz, amely a komponens minden verziójában különböző (Estublier és Casallas, 1994). A komponenseket így a nevük és az attribútumhalmaz együttese azonosítja.

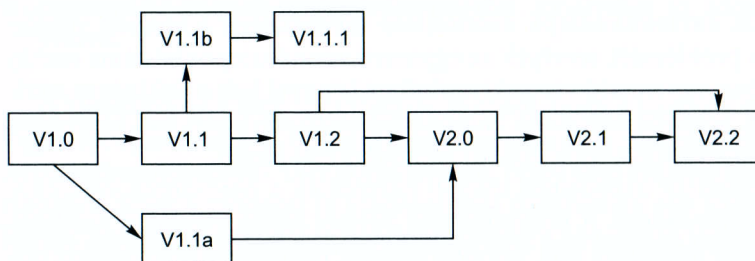
3. *Változtatásorientált azonosítás.* Minden rendszert az attribútumalapú azonosítás szerint nevezünk el, de ehhez még társítunk egy vagy több változtatási kérelmet (Munch és mások, 1993). Ez feltételezi, hogy a komponens minden verziója egy vagy több változtatáskérés alapján jött létre. A rendszerverziót így a név és a komponensben implementált változtatás összekapcsolásával azonosítjuk.

Verziószámozás

Az egyszerű verziószámozási sémában a komponensek és a rendszer nevét egy verziószámmal egészítjük ki. Így beszélhetünk a Solaris 4.3-ról (a Solaris-rendszer 4.3-as verziója) és az 1.4-es verziójú getToken komponensről. Az első verziót nevezhetjük 1.0-nak, az alverziók az 1.1, 1.2 és így tovább. Valamely ponton előáll egy új kiadás (2.0-s kiadás), és a folyamat kezdődik előlről a 2.1-es verzióval. A séma lineáris, mivel abból a feltételezésből indul ki, hogy a rendszer verzióit egymás után állítjuk elő. Sok verziókezelő eszköz (29.5. alfejezet), mint például az RCS (Tichy, 1985) és a CVS (Berliner, 1990), támogatja a verzióazonosításnak ezt a fajta megközelítését.

A 29.6. ábra szemlélteti ezt a megközelítést és azt, hogyan származnak különböző verziójú rendszerek a rendszer előző verzióiból. Ebben a diagramban a nyilak a forrásverziótól az új, forrásból származtatott verzió felé mutatnak. Vegyük észre, hogy a verziók származtatása nem feltétlenül lineáris, és az egymást követő verziószámok származhatnak különböző alapvonalakból. Ez megfigyelhető a 29.6. ábrán, amelyen a 2.2-es verzió az 1.2-es verzióból származik, nem pedig a 2.1-esből. Elvileg bármely létező verzió használható egy új verzió kiindulási pontjaként.

Ez a séma egyszerű, de a kapcsolódó információk megfelelő kezelését igényli, hogy nyomon tudjuk követni a verziók közti különbségeket, valamint a rendszer-változtatási javaslatok és a verziók közti kapcsolatokat. Például az 1.1-es verzió különbözhet az 1.2-estől abban, hogy másik grafikus könyvtárat használunk hozzá. A név nem mond semmit a verzióról vagy arról, hogy miért jött létre. Emiatt a konfigurációs adatbázisban tárolni kell olyan rekordot, amely leír minden verziót és azt is, miért jött az létre. Szükség lehet még továbbá explicit hivatkozásokra is a változtatási kérések és a komponensek különböző verziói között.



29.6. ábra. Verziószármaztatási struktúra

Attribútumalapú azonosítás

Az explicit verzió elnevezési sémák alapvető problémája, hogy nem tükrözik azokat az attribútumokat, amelyeket a verziók azonosítására használhatnánk. Ilyen azonosító attribútumok például:

- vevő,
- fejlesztőnyelv,
- fejlesztési állapot,
- hardverplatform,
- készítés dátuma.

Ha minden verziót egyedi attribútumhalmazzal azonosítunk, akkor könnyű olyan új verziókat létrehozni, amelyek valamelyik létező verzióból származnak. Ezeket az attribútumértékek egyedi halmaza azonosítja. Ezen értékek nagy része azonos a szülőével, így a verziók közötti kapcsolatok követhetők. A verziót előállíthatjuk a szükséges attribútumértékek megadásával. Az attribútumokat kezelő függvények olyan lekérdezéseket támogatnak, mint például „az utoljára készült verzió” vagy „a megadott dátumok között készült verziók”.

Például az AC3D-rendszer 2003 januárjában, Java nyelven, Windows XP alá fejlesztett verzióját a következőképpen azonosítanánk:

AC3D (nyelv = Java, platform = XP, dátum = 2003jan)

Az AC3D-ben a komponensek általános specifikációját használva, a verziókezelő eszköz a komponensek azon verzióját tudja leválogatni, amelyekben az attribútumok „Java”, „XP” és „2003jan”.

Lehetséges, hogy az attribútumalapú azonosítás közvetlenül a verziókezelő rendszerbe van implementálva, és a komponensattribútumokat a rendszer adatbázisa kezeli. Lehetséges, hogy az attribútumazonosító rendszer egy rejtett verzióelnevezési séma egy felső rétegébe van beépítve. Ekkor a konfigurációs adatbázis kezeli az azonosító attribútumok és a háttérben lévő rendszer- és komponensverziók közötti kapcsolatokat.

Változtatásorientált azonosítás

A rendszerverziók attribútumalapú azonosítása megszünteti a verziók visszanyerésének egyes problémáit, amelyek az egyszerű verziószámozási séma esetén lépnek fel. Azonban egy verzió visszakereséséhez ismerni kell a szükséges attribútumok értékeit. Továbbá egy külön változtatáskezelő rendszert kell használni, amely a verziók és a változtatások közötti kapcsolatokat kezeli.

A változtatásorientált azonosítást inkább használjuk a rendszerek verzióira, mintsem a komponensekére. A komponensek verziói rejtve vannak a konfigurációkezelő rendszer használói előtt. Minden javasolt rendszerváltoztatáshoz tartozik egy változtatáshalmaz, amely a rendszer különböző komponenseinek az

adott változtatást implementáló változtatásait írja le. A változtatáshalmazokat megadhatjuk egymás után, így, elvileg legalábbis, előállíthatjuk a változtatások tetszőleges halmazát tartalmazó rendszert. Például ha egy olyan rendszer változtatáshalmazát alkalmazzuk, amelyet inkább Linux, mint Solaris alá fejlesztettek, a változtatások követéséhez egy új rendszeradatbázis szükséges. Ekvivalens módon egy Linux/Solaris váltásnál a változtatásoknál követni kell a felhasználói felületek nyelvénél angolról olaszra való átváltását.

A gyakorlatban természetesen nem lehetséges a változtatások tetszőleges halmazát tartalmazó rendszert előállítani. Bizonyos változtatáshalmazok lehetnek inkompatibilisek, így az A változtatáshalmaz megadása a D változtatáshalmaz után érvénytelen rendszert eredményezhet. Továbbá az is problémát okozhat, hogy a különböző változtatások ugyanolyan rendszerkódot eredményezhetnek. Ha a kód az A változtatáshalmaz mellett került megváltoztatásra, akkor a D változtatáshalmaz a továbbiakban nem használható. Hogy elkerüljék ezeket a nehézségeket, azok a verziókezelő eszközök, amelyek támogatják a változtatás-orientált azonosítást, megengedik, hogy a rendszer konzisztenciáját fenntartó szabályokat specifikáljunk, amelyek korlátozzák a változtatáshalmazok kombinálását.

29.3.2. Kiadások kezelése

A kiadás a rendszer olyan verziója, amely eljut a vásárlókhoz. A rendszerkiadás kezelői döntenek el, hogy mikor lehet egy rendszert kiadni a vásárlóknak, felügyelik a kiadások készítésének folyamatát és az elosztó médiát, valamint dokumentálják a kiadásokat, hogy biztosítsák, hogy szükség esetén pontosan elő lehessen állítani a kiadott változatot.

A rendszerkiadás nem csak a rendszer végrehajtható kódja. A kiadás tartalmazhat még:

1. *konfigurációs állományokat*, amelyek azt definiálják, hogy hogyan kell a kiadást konfigurálni egyéni telepítés esetén;
2. *adatállományokat*, amelyek a rendszer sikeres működéséhez szükségesek;
3. *telepítőprogramot*, amely segíti a rendszer telepítését a célhardverre;
4. *elektronikus és papíralapú dokumentációt*, amely leírja a rendszert;
5. *csomagolást és egyéb propagandaanyagokat*, amelyeket az adott kiadáshoz terveztek.

A kiadáskezelők nem indulhatnak ki abból, hogy a vásárlók mindig telepíteni fogják az új kiadásokat. Egyes rendszerfelhasználók megelégedhetnek a már létező rendszerkiadással. Úgy gondolhatják, hogy nem éri meg átváltani egy újabb kiadásra. Ezért a rendszer új kiadásai nem függhetnek egy korábbi kiadás megletétől. Ennek illusztrálására gondoljuk végig a következő forgatókönyvet:

1. A rendszer első változatát kiadják, és használatba kerül.
2. A második kiadás követi, amely új adatállományok telepítését igényli, de egyes vásárlók nem igénylik a második kiadás szolgáltatásait, így az első kiadás mellett maradnak.
3. A harmadik kiadásnak szüksége van a második kiadás által telepített adatállományokra, de maga nem tartalmaz új adatállományokat.

A szoftver elosztója nem indulhat ki abból, hogy a harmadik kiadás által igényelt állományok mindenhol telepítve vannak. Egyes helyeken az első kiadásról közvetlenül váltanának a harmadik kiadásra, a második kiadás kihagyásával. Máshol pedig módosíthatták a második kiadás által telepített állományokat, hogy azok a helyi körülményeket tükrözzék. Ezért az adatállományokat terjesztetni és telepíteni kell a rendszer harmadik kiadásával is.

Döntés a rendszerkiadásról

Egy rendszerkiadás előkészítése és terjesztése költséges folyamat, főleg tömegpiaci szoftvertermékek esetén. Ha a kiadások túl gyakoriak, a vásárlók nem térnek át az új kiadásra, ha túl ritkák, a piaci részesedés csökkenhet, mivel a vásárlók alternatív rendszerekre térnek át. Ez természetesen nem vonatkozik egyedi szoftverekre, amelyeket speciálisan fejlesztettek egy szervezet részére. Azonban az ilyen szoftverek esetében a ritka kiadások jelenthetik a szoftver és a támogatott üzleti folyamatok közti különbségek növekedését.

A döntést, hogy mikor kell a rendszer egy új kiadását megjelentetni, technikai és szervezeti megfontolások alapján kell meghozni, mint azt a 29.7. ábra mutatja.

Tényező	Leírás
A rendszer technikai minősége	Ha olyan komoly problémák merülnek fel, amelyek olyan használat mellett fordulnak elő, ahogy sokan használják a rendszert, akkor szükség lehet egy hibajavító kiadás kibocsátására. Azonban kisebb rendszerhibákat javítások kibocsátásával is kiküszöbölhetünk (gyakran az interneten keresztül terjesztik ezeket), amelyeket a rendszer aktuális kiadására lehet alkalmazni.
Platformváltás	A szoftveralkalmazásnak egy új kiadása létrehozható, ha az operációsrendszer-platform új verziója megjelent.
Lehman ötödik törvénye (21. fejezet)	Ez azt állítja, hogy a kiadásonkénti funkcionalitásnövekedés hozzávetőleg konstans. Emiatt, ha van egy olyan kiadás, amely sok új funkciót tartalmaz, azt általában egy javító kiadás követi.
Versenyhelyzet	Szükség lehet az új kiadásra, ha egy versenytárs terméke megjelenik.
Piaci szükségletek	Egy szervezet piaci részlege megkövetelheti a kiadások adott időpontra való kibocsátását.
Vásárlók változtatási javaslatai	Egyéni rendszerek esetében, a megrendelő javasolhat speciális változtatásokat, amelyekért fizet, és várhatja az új kiadást, amint ezeket a változtatásokat implementálják.

29.7. ábra. A rendszerkiadási stratégiát befolyásoló tényezők

Rendszerkiadás készítése

A kiadáskészítés az a folyamat, amikor előállítjuk azokat az állományokat és dokumentációt, amely egy rendszerkiadás összes komponensét tartalmazza. A programok végrehajtható kódját és minden szükséges adatállományt egybe kell gyűjteni, és azonosítani kell. Konfigurációs leírásokat készíthetünk a különböző hardverekhez és operációs rendszerekhez, és utasításokat azoknak a vásárlóknak, akik maguk konfigurálják a rendszerüket. Ha elektronikus kézikönyveket adunk, akkor azokat a szoftverrel együtt kell tárolni. Telepítőszkripteket is írhatunk. Végül, amikor minden információ a rendelkezésünkre áll, elkészíthetjük a kiadás mesterlemezét, és átadhatjuk kiadásra.

Ma a rendszerkiadások általános hordozó médiumai az optikai lemezek (CD-ROM vagy DVD), amelyek 600 MB-tól 4 GB-ig tárolhatnak adatokat. Továbbá sok szoftverterméket az interneten keresztül is elérhetővé tesznek, így a vásárlók letölthetik azokat. Azonban sok ember hosszúnak tartja a nagy állományok letöltési idejét, és előnyben részesíti a CD-ROM-os kiadást.

Rendszerkiadás dokumentációja

Amikor egy kiadás elkészül, dokumentálni kell, így biztosítjuk, hogy szükség esetén újra elő lehessen állítani a jövőben. Ez különösen fontos a hosszú élettartamú, egyedi, beágyazott rendszerek esetében. A megrendelők évekig használhatják a rendszer ugyanazon kiadását, majd specifikus változtatásokat kérhetnek valamely kiadásba, évekkel az eredeti kiadás kibocsátása után.

Hogy dokumentáljunk egy kiadást, fel kell jegyezni azon komponensek forráskódjának verzióját, amelyeket a végrehajtható kód előállításához használtunk. Továbbá meg kell tartanunk a forráskód, a végrehajtható kód és az összes adat- és konfigurációállomány másolatát. Fel kell jegyeznünk az operációs rendszer, a könyvtárak, a fordítók és a szoftverépítés során használt többi eszköz verzióját is. Ezek azért szükségesek, hogy egy későbbi időpontban felépíthessük pontosan ugyanazt a rendszert. Ezekben az esetekben a platformszoftver és az eszközök másolatait is tárolhatjuk a verziókezelő rendszerben, együtt a célrendszer forráskódjával.

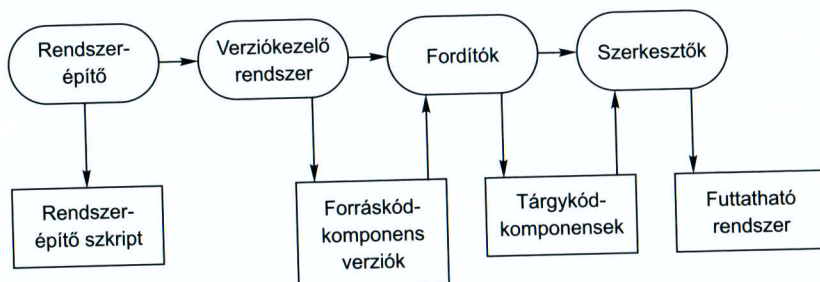
29.4. RENDSZERÉPÍTÉS

A rendszerépítés az a folyamat, amikor a szoftverkomponenseket lefordítjuk és szerkesztjük, így előáll egy program, ami meghatározott célkonfiguráción futtatható. Mikor egy rendszert építünk a komponenseiből, a következő kérdéseket kell átgondolnunk:

1. A rendszerépítéshez szükséges minden komponens benne van az építési utasításokban?

2. A szükséges komponensek megfelelő verzióját tartalmazzák az építési utasítások?
3. Minden szükséges adatállomány elérhető?
4. Ha a komponensek adatállományokra hivatkoznak, akkor a név egyezik-e a célgépen használt névvel?
5. A fordító és a többi szükséges eszköz megfelelő verziója áll rendelkezésünkre? Lehet, hogy a szoftvereszközök jelenlegi verziója inkompatibilis a régebbi verzióval, amelyet a rendszer fejlesztéséhez használtunk.

Manapság konfigurációkezelő eszközök segítségével automatizáljuk az építési folyamatot. A konfigurációkezelő csapat ír egy szkriptet, amely definiálja a rendszer különböző komponensei közti függőségeket. Ugyanez a szkript meghatározza, hogy mely eszközökkel kell a rendszer komponenseit fordítani és szerkeszteni. A rendszerépítő eszköz értelmezi a szkriptet, és meghívja a végrehajtható rendszer építéséhez szükséges programokat. Ezt ábrázolja a 29.8. ábra. Egyes programozási környezetekben (például a Java fejlesztői környezetek) az építőszkript automatikusan létrejön a forráskód elemzésénél, és kiderül, hogy melyik komponens hívódik meg. Természetesen ilyenkor a tárolt komponens és a programbeli komponens neve azonos lesz.



29.8. ábra. Rendszerépítés

A komponensek közti függőségek specifikálva vannak az építőszkriptben, így a rendszerépítő rendszer eldöntheti, mikor kell egy komponenst úrafordítani, és mikor használható újra a már létező tárgykód. Sok eszközben az építőszkript függőségei általában azon fizikai állományok közötti függőségek, amelyek a komponensek forrás- és tárgykódját tartalmazzák. Azonban, amikor több forráskódállomány reprezentálja egy komponens valamely verzióját, néha nem egyértelmű, mely forrásállományokat használták fel a komponens tárgykódjának előállításához. Ez a probléma főleg akkor fordul elő, amikor a forrás- és a tárgykód állományai közti függőség arra támaszkodik, hogy azonos a nevük, csak a kiterjesztésük különbözik (például .c és .o). Ez a probléma csak úgy oldható meg, ha a verziókezelő és a rendszerépítő eszközök integráltak.

29.5. CASE-ESZKÖZÖK A KONFIGURÁCIÓKEZELÉSHEZ

A konfigurációkezelő folyamatok általában szabványosítottak, és előre definiált eljárásokat alkalmaznak. A hatalmas mennyiségű adat óvatos kezelése és a részletekre való figyelés nagyon fontos. Amikor egy rendszert a komponenseinek verzióiból építünk fel, egyetlen konfigurációkezelő hiba ahhoz vezethet, hogy a szoftver nem fog helyesen működni. Következésképpen a CASE-eszközök konfigurációkezelés-támogatása elengedhetetlen, és az 1970-es évek óta rengeteg különböző eszköz jelent meg, amelyek a konfigurációkezelés különböző területeit célozzák meg.

Ezek az eszközök kombinálhatók úgy, hogy létrejöjjön egy olyan konfigurációkezelő munkafelület, amely támogat minden konfigurációkezelési tevékenységet. Kétféle konfigurációkezelő munkafelület ismert:

1. *Nyílt munkafelületek.* A konfigurációkezelési folyamat minden állapotát az eszközök használóinak szabványos vállalati eljárásain keresztül integráló eszközök. Sok kereskedelmi és nyílt forrású speciális célú konfigurációkezelési eszköz elérhető. A konfigurációkezelés támogatható olyan hibakövető eszközökkel, mint a Bugzilla, a verziókezelésnél pedig használható az RCS (Tichy, 1985) vagy a CVS (Berliner, 1990), a rendszerépítéshez rendelkezésre áll a make (Feldman, 1979; Oram és Talbott, 1991) vagy az imake (DuBois, 1996). Ezek mindegyike nyílt forrású és szabadon elérhető.
2. *Integrált munkafelületek.* Ezek a munkafelületek integrált eszközrendszert nyújtanak a verziókezelés, rendszerépítés és változtatáskövetés számára. Például a Rational Unified Change Management folyamata integrált konfigurációkezelési munkafelületet ad, együttműködve a ClearCase-zel (White, 2000) a rendszerépítés és a verziókezelés, illetve a ClearQuest-tel a változtatáskövetés területén. Az integrált konfigurációkezelési munkafelületek előnye, hogy az adatcsere egyszerű, és a munkafelület tartalmaz egy konfigurációkezelési adatbázist. Az integrált konfigurációkezelési munkafelületek korábbi rendszerekből származnak, mint a Lifespan (Whitgift, 1991) a konfigurációkezelésnél és a DSEE (Leblang és Chase, 1987) a verziókezelésnél és a rendszerépítésnél. Ugyanakkor az integrált konfigurációkezelési munkafelületek összetettek és drágák, így sok szervezet az olcsóbb és egyszerűbb egyedi megoldásokat részesíti előnyben.

Több nagy rendszert különböző honlapokon fejlesztenek, ez olyan konfigurációkezelési eszközöket igényel, amelyek támogatják a többlapos munkát és a konfigurációs tételek adatainak többszörös tárolását. A legtöbb konfigurációkezelési eszköz egylapos munkára van tervezve, de egyes eszközök, például a CVS, támogatják a többlapos megoldásokat is (Vesperman, 2003).

29.5.1. Változtatáskezelés támogatása

A változtatáskezelés folyamatában részt vevő emberek felelősek valamely tevékenységért. Miután befejezik ezt a tevékenységet, továbbadják az űrlapokat és a hozzájuk tartozó konfigurációs tételeket valaki másnak. A folyamat eljárás jellege azt jelenti, hogy a változtatáskezelés modelljét tervezhetjük és integrálhatjuk a verziókezelő rendszerbe. Ezután ezt a modellt úgy értelmezhetjük, hogy a megfelelő dokumentumokat kell a megfelelő időben a megfelelő embereknek eljuttatni.

Többféle változtatáskezelő eszköz elérhető a relatíve egyszerű, nyílt forrású eszközöktől (mint a Bugzilla) az olyan átfogó integrált rendszerekig, mint a Rational ClearQuest. Ezek az eszközök a következő képességek közül többel vagy mindegyikkel rendelkezhetnek, hogy támogassák a folyamatot:

1. *Űrlapszerkesztő*, amely lehetővé teszi a változtatási javaslat űrlapok készítését és kitöltését.
2. *Munkafolyamat-rendszer*, amely lehetővé teszi a konfigurációkezelő csapatnak, hogy definiálja a változtatás menetét, és hogy kik azok az emberek, akik feldolgozzák a változtatásigénylő űrlapokat. Ez a rendszer automatikusan továbbíthatja az űrlapokat a megfelelő embernek a megfelelő időben, és értesítheti a csapat többi tagját a változtatás menetéről. A folyamatban részt vevők értesítése elektronikus levélben történik.
3. *Változtatási adatbázis*, amely az összes változtatási javaslatot kezeli, és amelyet össze lehet kapcsolni egy verziókezelő rendszerrel. Általában rendelkezésre állnak azok a lekérdezési lehetőségek, amelyek lehetővé teszik a csapat tagjainak, hogy megtaláljanak egy adott változtatási javaslatot.
4. *Változtatásjelentő rendszer*, amely jelentéseket generál a változtatási kérések állapotáról, és az továbbítható.

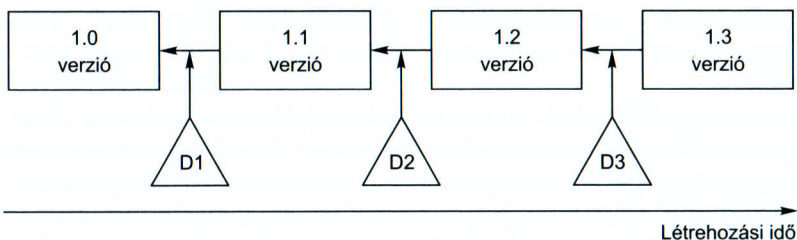
29.5.2. Verziókezelés támogatása

A verziókezelés nagy mennyiségű információ kezelését jelenti, valamint annak biztosítását, hogy rögzítjük és felügyeljük a rendszerváltoztatásokat. A verziókezelő eszközök kezelik a konfigurációs tételek tárolóját, amely megváltoztathatatlan. Hogy dolgozhassunk egy konfigurációs tételen, azt át kell helyezni egy munkakönyvtárba. Miután a munkát befejeztük, vissza lehet helyezni a tárolóba, és ekkor automatikusan létrejön egy új verzió.

Minden verziókezelő rendszer hasonló alapszolgáltatásokat nyújt annak ellenére, hogy némelyik kifinomultabb szolgáltatásokkal is rendelkezik. Példa ezekre a szolgáltatásokra:

1. *Verziók és rendszerkiadások azonosítása*. A kezelt verziók azonosítót kapnak, amikor bekerülnek a rendszerbe. A különböző rendszerek különböző verzióazonosító módszereket támogatnak, melyeket a 29.3.1. alfejezetben taglaltam.

2. *Tároláskezelés.* A különböző verziók – melyek nagyrészt azonosak – által foglalt tárolási terület csökkentésére a verziókezelő rendszerek olyan tároláskezelő szolgáltatásokat nyújtanak, amelyek a verziókat valamely mesterverziótól való különbözőségükkel jellemezznek. A verziók közti különbségeket *deltával* reprezentáljuk, amely magában foglalja a hozzá rendelt rendszerverzió előállításához szükséges utasításokat. Ezt mutatja a 29.9. ábra, amely azt ábrázolja, hogyan használhatók a visszafelé irányuló delták a rendszer korábbi verzióinak előállításához. A legutolsó verzió az 1.3-as. Az 1.2-es verzió előállításához alkalmazzuk a változtatás deltát, amely visszaállítja ezt a verziót.
3. *Változtatástörténet rögzítése.* A rendszeren vagy a komponenseken végzett minden változtatás nyilvántartásba kerül, és bekerül egy listába. Egyes rendszerekben ezek a változtatások használhatók egy adott rendszerverzió kiválasztásához.
4. *Független fejlesztés.* A rendszer különböző verzióit lehet párhuzamosan fejleszteni, és bármely verziót lehet a többitől függetlenül változtatni. Például az 1-es kiadás módosítható a 2-es folyamatban lévő fejlesztése mellett egy új 1-es szintű delta hozzáadásával. A verziókezelő rendszer nyomon követi, hogy mely komponenseket jelöltük meg szerkesztésre, és biztosítja, hogy a különböző fejlesztők által véghezvitt változtatások ne zavarják egymást. Egyes rendszerek a komponensnek csak egy példányát engedik megjelölni szerkesztésre; mások feloldják az átfedéseket, amikor a komponens visszakerül a rendszerbe.
5. *Projekttámogatás.* A rendszer képes támogatni több projektet és több állományt is egyszerre. Az olyan projekttámogató rendszerekben, mint a CVS is, lehetőség van a projekthez kapcsolódó bármely állomány be- vagy kiléptetésére ahelyett, hogy egy időben csak egyetlen állománnyal dolgoznánk.



29.9. ábra. Delta-alapú verziókezelés

29.5.3. Rendszerépítés támogatása

A rendszerépítés számításigényes folyamat. Ha egy nagy rendszer összes komponensét le kell fordítani és szerkeszteni, az több óráig tarthat. Több száz állományról lehet szó, amelyek mindegyike emberi hiba lehetőségét hordozza magában, amennyiben manuálisan fordítják és szerkesztik. A rendszerépítő eszközök automatizálják az építés folyamatát, hogy csökkentsék az emberi hibázás lehetőségét, és ha lehet, csökkentsék a rendszerépítés idejét.

A rendszerépítő eszközök lehetnek különálló eszközök, mint például a Unix make segédprogramjának (Oram és Talbott, 1991) leszármazottai, vagy lehetnek egy verziókezelő eszköz részei. Egy rendszerépítő CASE-eszköz által nyújtott szolgáltatások lehetnek:

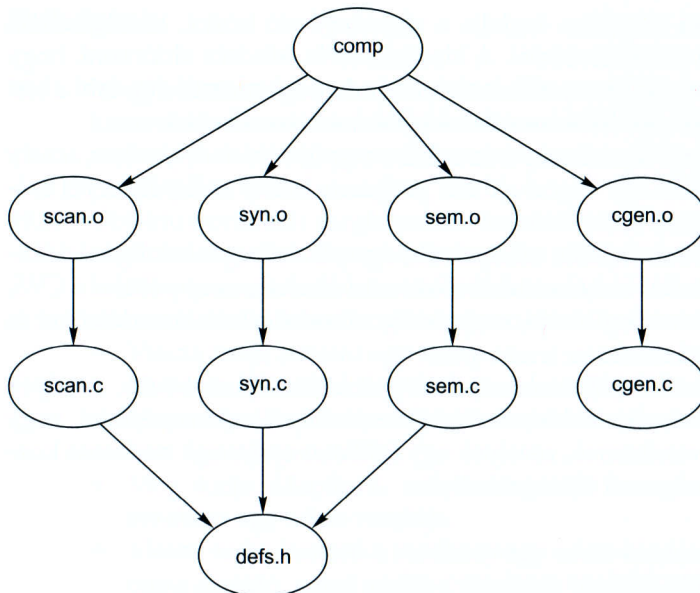
1. *Függőségspecifikáló nyelv és a hozzá tartozó interpreter.* Meghatározhatjuk a komponensfüggőségeket, és csökkenhet az újrafordításra fordított idő. Ezt a későbbiekben részletesebben tárgyalom.
2. *Eszközkiválasztás és példányosítás támogatása.* Ki lehet jelölni a fordítókat és egyéb eszközöket, amelyek részt vesznek a forráskód feldolgozásában.
3. *Osztott fordítás.* Egyes rendszerépítők, főleg azok, amelyek egy integrált konfigurációkezelő rendszer részei, támogatják az osztott, hálózati fordítást. Ahelyett, hogy a fordítás egy gépen történne, a rendszerépítő tétlen processzorokat keres a hálózatban, és több fordítást is elindít párhuzamosan. Ez radikálisan csökkenti a rendszer építéséhez szükséges időt.
4. *Származtatott objektumok kezelése.* A származtatott objektumok olyan objektumok, amelyek más forrásobjektumokból származnak. A származtatott objektumkezelés összekapcsolja a forráskódot a származtatott objektummal, és csak akkor származtat újra egy objektumot, amikor erre a forráskód megváltozása miatt szükség van.

A származtatott objektumok kezelése és az újrafordítás minimalizálása egy példán keresztül jobban megmagyarázható. Tegyük fel, hogy a comp nevű programot a scan.o, syn.o, sem.o és a cgen.o tárgymodulokból állítjuk elő. Minden tárgymodulhoz létezik egy forráskódmodul, melyek neve scan.c, syn.c és cgen.c. A defs.h nevű deklarációs állományt a scan.c, syn.c és a sem.c közösen használják (29.10. ábra). A 29.10. ábrán a nyilak azt jelentik, hogy „függ tőle”. Tehát a comp függ scan.o-tól, syn.o-tól, sem.o-tól és cgen.o-tól, és a scan.o függ a scan.c-től és így tovább.

Ha a scan.c-t megváltoztatjuk, akkor a rendszerépítő eszköz észleli, hogy a scan.o származtatott objektumot újra el kell készíteni. Ezt úgy veszi észre, hogy összehasonlítja a scan.o és a scan.c módosítási idejét, és azt észleli, hogy a scan.c a scan.o után módosult. Ezután meghívja a C-fordítót, hogy fordítsa le a scan.c-t, hogy előálljon az új származtatott objektum, a scan.o.

Ezután az építőeszköz a comp és a scan.o közötti kapcsolat alapján észleli, hogy a comp-ot is újra elő kell állítani a scan.o, syn.o, sem.o és a cgen.o szerkesztésével. A rendszer azt is észleli, hogy a többi tárgykódkomponens változatlan, így a hozzájuk tartozó forráskódok fordítására nincs szükség.

Egyes rendszerépítők az állomány módosításának dátumát használják annak eldöntésére, hogy szükséges-e az újrafordítás. Ha a forráskód állományát a hozzá tartozó tárgykódállomány után módosítottuk, akkor a tárgykódállományt újra elő kell állítani. Azonban ez általában azt jelenti, hogy csak a legújabb forráskódból származó tárgykód létezik. Ha a forráskód egy korábbi verzióját kell újrafordítani, akkor a módosításának dátumát mesterségesen kell beállítani, hogy a rendszerépítő észlelje, hogy újra kell fordítani.

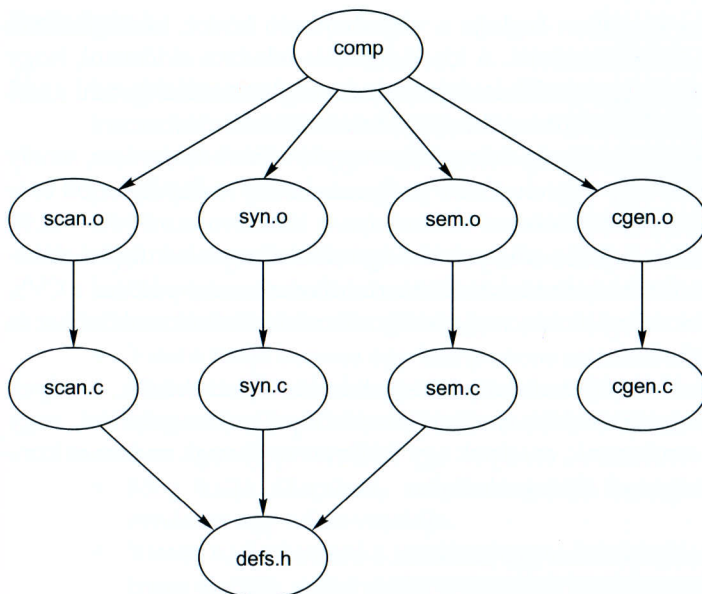


29.10. ábra. Komponensfüggőségek

Más programok kifinomultabb módon közelítik meg a származtatott objektumok kezelését. Kiegészítik a tárgykódot a forráskód verzióazonosítójával, és a tárolási kapacitás határain belül minden származtatott objektumot tárolnak. Így általában lehetséges a forráskódkomponensek minden verziójának tárgykódját visszaállítani újrafordítás nélkül.

Kulcsfogalmak

- A konfigurációkezelés a rendszerváltozások kezelése. Egy rendszer karbantartása során a konfigurációkezelő csapat feladata biztosítani, hogy a változtatásokat felügyelt módon hajtsuk végre.
- Egy nagy projekt esetében formális dokumentum elnevezési sémát kell kialakítani, és ezt kell használni az összes projektbeli dokumentum különböző verzióinak nyomon követésére.
- A konfigurációkezelő csapatot egy konfigurációs adatbázisnak kell támogatnia, amely tartalmaz minden információt a rendszerváltoztatásokról és a hátralévő változtatási kérelmekről. A projekteknek formális eszközöket kell kialakítani a változtatási kérelmekhez.
- Amikor kialakítunk egy konfigurációkezelő sémát, egy konzisztens verzióazonosító sémát is ki kell alakítani. A verziókat azonosíthatjuk verziószámmal, attribútumhalmazzal vagy az általuk implementált rendszer-változtatási javaslatokkal.



29.10. ábra. Komponensfüggőségek

Más programok kifinomultabb módon közelítik meg a származtatott objektumok kezelését. Kiegészítik a tárgykódot a forráskód verzióazonosítójával, és a tárolási kapacitás határain belül minden származtatott objektumot tárolnak. Így általában lehetséges a forráskódkomponensek minden verziójának tárgykódját visszaállítani újrafordítás nélkül.

Kulcsfogalmak

- A konfigurációkezelés a rendszerváltozások kezelése. Egy rendszer karbantartása során a konfigurációkezelő csapat feladata biztosítani, hogy a változtatásokat felügyelt módon hajtsuk végre.
- Egy nagy projekt esetében formális dokumentum elnevezési sémát kell kialakítani, és ezt kell használni az összes projektbeli dokumentum különböző verzióinak nyomon követésére.
- A konfigurációkezelő csapatot egy konfigurációs adatbázisnak kell támogatnia, amely tartalmaz minden információt a rendszerváltoztatásokról és a hátralévő változtatási kérelmekről. A projekteknek formális eszközöket kell kialakítani a változtatási kérelmekhez.
- Amikor kialakítunk egy konfigurációkezelő sémát, egy konzisztens verzióazonosító sémát is kell alakítani. A verziókat azonosíthatjuk verziószámmal, attribútumhalmazzal vagy az általuk implementált rendszer-változtatási javaslatokkal.

- A rendszer kiadása magában foglalja a végrehajtható kódot, konfigurációs állományokat és a dokumentációt. A kiadáskezelés feladata eldönteni, hogy mikor kell kiadni a rendszert, előkészíteni minden dokumentációt, ami a terjesztéshez szükséges, és dokumentálni minden rendszerkiadást.
- A rendszerépítés a rendszerkomponensek összegyűjtésének folyamata, amely eredményeként előáll egy végrehajtható program, amely valamilyen cél számítógépes rendszeren futtatható.
- Vannak olyan CASE-eszközök, amelyek támogatják az összes konfigurációkezelő tevékenységet. Ezek olyan eszközöket tartalmaznak, mint például a CVS, amely a rendszerverziókat kezeli, vagy pedig változtatáskezelő eszközöket és rendszerépítő eszközöket.
- A konfigurációkezelő CASE-eszközök lehetnek különálló eszközök, amelyek támogatják a változtatáskezelést, a verziókezelést és a rendszerépítést, vagy lehetnek integrált rendszerek, amelyek egy felületet nyújtanak az összes konfigurációkezelési folyamat támogatásához.

További irodalom

Configuration Management Principles and Practice. Átfogó könyv, mely áttekinti a szabványos és tradicionális konfigurációkezelési közelítéseket, amelyek továbbra is alkalmazhatók a modern (például agilis) szoftverfejlesztésben. (Hass, A. M. J., 2002, Addison-Wesley.)

„A layered architecture for uniform version management”. Ez a cikk a szoftververzió-kezelés különböző közelítéseit tárgyalja és javasol egy olyan alapmodellt, amely mindegyikhez illeszthető. A terület alapozó munkáinak jó áttekintése. (Westfechel, B. és mások, *IEEE Transactions on Software Engineering*, 27 (12), December 2001.)

„Software configuration management: A roadmap”. Áttekintő cikk, a konfigurációkezelés evolúcióját tárgyalja és a terület nyitott kutatási problémáit nevezi meg. (Estublier, J., *Proc. Int. Conf. on Software Engineering*, 2000, IEEE Press.)

Trends in Software: Configuration Management. A konfigurációkezelés különböző aspektusait bemutató írások gyűjteménye, melyek szerzői aktív kutatói és gyakorlói a területüknek. Jó bevezetés azoknak a hallgatónak és gyakorló szakembereknek, akik szeretnének haladó konfigurációkezelési témákról olvasni. A problémák többsége még jelenleg sem megoldott. (Tichy, W. (ed.), 1995, John Wiley & Sons.)

Feladatok

- 29.1. Magyarázza meg, miért nem használhatja egy dokumentum címét a dokumentum azonosítására egy konfigurációkezelő rendszerben. Javasoljon szabványos dokumentumazonosító sémát, amelyet a szervezet minden projektjében lehet használni.
- 29.2. Objektumorientált megközelítés használatával (8. fejezet) tervezzen egy olyan konfigurációs adatbázismodellt, amely információkat tárol a rendszerkomponensekről, a verziókról, a kiadásokról és a változtatásokról. Az adatmodellről a következőkhöz hasonló dolgokat követelhetünk meg:
 - Vissza tudja keresni egy komponens minden vagy egy adott verzióját.
 - Vissza tudja keresni egy komponens legújabb verzióját.
 - Meg tudja állapítani, mely változtatási kérelmeket implementál a rendszer egy adott verziója.
 - Meg tudja állapítani, milyen verziójú komponenseket tartalmaz a rendszer egy adott verziója.
 - Vissza tudja keresni a rendszer egy adott kiadását, mind a kiadás dátuma alapján, mind pedig a rendszer vásárlói alapján.
- 29.3. Adatfolyam-diagram használatával írjon le egy változtatáskezelési eljárást, amelyet egy olyan szervezet használhat, amely külső klienseknek fejleszt szoftvereket. A változtatásokat javasolhatják mind külső, mind belső forrásból.
- 29.4. Hogyan egyszerűsítheti a projekतालapú konfigurációkezelési rendszer (mint például a CVS) használata a verziókezelő folyamatot?
- 29.5. Magyarázza meg, miért egyszerűsíti le egy rendszer specifikus verzióihoz tartozó komponensek megtalálását egy attribútumalapú verzióazonosító rendszer.
- 29.6. Váolja azokat a nehézségeket, amelyek felléphetnek, amikor a rendszert felépítjük a komponenseiből. Milyen egyedi problémák léphetnek fel, amikor a rendszert egy gazdaszámítógépen építjük föl több célgép számára?
- 29.7. Hivatkozva a rendszerépítésre, magyarázza el, miért tartják karban azokat a régi számítógépeket, amelyeken nagy szoftverrendszereket fejlesztettek.
- 29.8. A rendszerépítés általános problémája az, amikor a fizikai állományneveket beépítik a rendszer kódjába, és a nevekből következő állományszerkezet eltér a célgépétől. Adjon programozói irányelveket, amelyek segítenek elkerülni ezt, és más olyan rendszerépítési problémákat, amik még eszébe jutottak.
- 29.9. Adjon meg öt olyan tényezőt, amelyet egy mérnöknek figyelembe kell vennie nagy szoftverrendszer egy kiadásának építési folyamatában.
- 29.10. Adjon meg két utat, amelyben a rendszerépítő eszköz optimalizálni tudja a rendszer egy verziójának összeállítását komponenseiből.