VLOGA PONOVNE UPORABE PRI RAZVOJU PROGRAMSKE OPREME

Evelin Vatovec Krmac

Povzetek :

Ponovna uporaba je koncept razvoja aplikacij na osnovi obstoječe programske opreme. Pogosto se navaja kot osnovna prednost objektne tehnologije in kot strategija izboljšanja procesa razvoja programske opreme. Ponovna uporaba je sredstvo za izboljšanje kvalitete programske opreme, povečanje produktivnosti in zmanjšanje napora pri gradnji sistemov. Učinkovita in sistematična ponovna uporaba pa zahteva organizacijske spremembe, spremembe razvojnega procesa in vlaganja.

Abstract:

Reuse is the ability to develop applications based on existing software. It's often described as a principal benefit of object technology and a strategy to improve software development process. Reuse is also means of improving software quality, increasing productivity and reducing the effort to build systems. Successful and systematic reuse requires organizational changes, changes in development process and investment.

1. UVOD

Ponovna uporaba programske opreme je koncept razvoja aplikacij na osnovi obstoječe programske opreme in se pogosto navaja kot osnovna prednost objektne tehnologije. Ponovna uporaba programske opreme pa ni nova - knjižnice funkcij in modulov se pri konvencionalnem razvoju uporabljajo že dolga leta. Uspeh ponovne uporabe je bil predvsem odvisen od navdušenosti in sposobnosti programerjev za iskanje in uporabo programov. Težko je bilo vedeti, kaj je razpoložljivega in kako neko funkcijo ali modul uporabiti. Poleg tega je bilo število argumentov v splošnih programih precej veliko, tako da je bil tudi napor, vložen v razumevanje in prilagajanje teh programov specifičnim potrebam, dokaj velik. Zaradi omenjenih problemov so se razvijalci zelo pogosto raje lotili razvoja novih programov.

Objektna tehnologija je nekatere izmed omenjenih problemov rešila. Objekti in razredi so že sami po sebi boljše enote za ponovno uporabo – objekti so koncepti, stvari, ki opisujejo posamezne entitete realnega sveta v programskem okolju, sistema, ki ga gradimo. Objekt v sebi združuje tako podatkovno strukturo kakor obnašanje. Razredi pa so skupki objektov s podobnimi lastnostmi (atributi), obnašanjem (operacijami), pomenom in povezavami z drugimi objekti. Objekti so lažje razumljivi kot funkcije pri konvencionalnem programiranju, saj obstaja med njimi in vsakdanjimi izkušnjami večja analogija - ljudje razumemo svet preko objektov, ne preko funkcij [11]. S pomočjo ograjevanja, skrivanja podatkov ločimo zunanje aspekte sistema, ki so dostopni ostalim objektom (vmesnik objekta - imena operacij, preko katerih lahko drugi objekti dostopajo do podatkovne strukture tega objekta), od notranje implementacije objekta (implementacija podatkovne strukture in algoritmov, ki jih uporabljajo operacije), ki je drugim objektom skrita. To zagotavlja šibko povezanost med moduli aplikacije, ki uporabljajo isti objekt, implementacijo objekta pa lahko spremenimo, ne da bi pri tem morali spreminjati ves sistem. Glede na to, da je vmesnik edini možni način dostopanja do objektove podatkovne strukture, je visoka stopnja ponovne uporabnosti objekta zagotovljena, saj ga lahko uporabimo v različnih sistemih, ne da bi nam bilo potrebno spreminjati njegov vmesnik.

Sama objektna tehnologija pa za dosego sistematične in razširjene ponovne uporabe še ni dovolj. Sistematična ponovna uporaba je institucionaliziran organizacijski pristop k načrtnemu razvoju programskih komponent za ponovno uporabo [4]. Razvite komponente je nato potrebno vzdrževati in skladno uporabljati tako, da obdržijo visoko stopnjo ponovne uporabnosti. Na tak način organizacija optimizira sposob-

nost hitre in učinkovite izdelave programskih produktov.

V praksi se je izkazalo, da se je za sistematično ponovno uporabo potrebno soočiti tako z vrsto tehničnih, kakor z vrsto netehničnih dejavnikov. Zato so potrebni ustrezni smernice, pristopi, procesi, modeli, metode in orodja.

2. KAJ

2.1. Kaj je ponovna uporaba ?

Definicij ponovne uporabe je veliko. Razlikujejo se predvsem po vrsti ponovne uporabe, ki jo obravnavajo (ponovna uporaba brez spreminjanja, ponovna uporaba s prilagoditvijo, ponovna uporaba z razvojem podrazredov - dedovanjem, direktna in indirektna ponovna uporaba, ipd.) in po vrsti ponovno uporabnih komponent. Vsem definicijam pa je skupno: ponovno uporabiti obstoječo programsko opremo pri razvoju nove. Primeri "klasičnih" definicij ponovne uporabe:

- "Ponovna uporaba je proces ponovne uporabe programske opreme, ki je bila načrtovana za ponovno uporabo." [14]
- "Ponovna uporaba je posebna oblika "reševanja" programske opreme - ponovna uporaba programske opreme, ki ni bila načrtovana za ponovno uporabo." [14]
- "Ponovna uporaba je sestavljanje celega ali dela sistema iz že obstoječih komponent. " [3]
- "Ponovna uporaba programske opreme je proces kreiranja programskih sistemov iz že obstoječih programskih komponent." [Krueger]
- "Ponovna uporaba je sposobnost razvoja aplikacij s pomočjo obstoječe programske opreme." [6]
- "Ponovna uporaba je sposobnost uporabe predhodno, v aplikaciji definirane, programske opreme. Komponenta je pri ponovni uporabi samo razširjena, ne pa spremenjena. Komponento lahko vključimo v novo aplikacijo, vendar mora ostati še vedno povezana s svojo prvotno definicjo." [6]
- "Ponovna uporaba se nanaša na uporabo komponent, ki so bile razvite z nekim izdelkom, v nekem drugem izdelku z drugačno funkcionalnostjo. Na tak način nam olajša razvoj drugih izdelkov. Pri tem pa ni nujno, da je ponovno uporabna komponenta modul ali del programa, lahko je načrt, del priročnika, množica testnih podatkov ali ocena stroškov." [12]

2.2. Kaj lahko ponovno uporabimo?

Ponovno uporabna komponenta je katerakoli komponenta, ki je bila namensko ter načrtno razvita za uporabo in tudi dejansko uporabljena v več kot enem kontekstu.

Ponovno uporabne komponente so lahko programi, specifikacije zahtev in načrtovanja, znanje o domeni, procesi, metode, dokumentacija, testni primeri, celi sistemi ali podsistemi, modeli, vzorci, ogrodja, implementacije razredov, primerki objektov ipd. Skratka, gre za vse možne izdelke, ki nastanejo v katerikoli fazi življenjskega cikla razvoja programske opreme od analize do testiranja.

Seveda pa pri ponovni uporabi velja zlato pravilo, ki pravi "preden karkoli ponovno uporabimo, mora biti ponovno uporabno". To pomeni, da mora biti komponenta :

1. razvita za ponovno uporabo,
2. shranjena v repozitoriju, kjer jo brez težav najdemo in ustrezno dokumentirana (vedeti moramo, kaj komponenta "dela" in kako jo lahko ponovno uporabimo).
3. Zgornje zahteve se nanašajo na še vedno dokaj pereče probleme kot so shranjevanje in iskanje ponovno uporabnih komponent, opis komponente in ustrezna dokumentiranost komponente.

3. ZAKAJ

Za vsako novo tehniko, metodologijo, pristop obstajajo njeni zagovorniki in nasprotniki. Tudi ponovna uporaba ni izjema. Trazc pravi, da je "ponovna uporaba religija - religija, ki je niso sprejeli vsi, vsaj ne v enaki meri" [14].

3.1. Zakaj "da" ?

Ponovna uporaba ni samo cilj, ampak sredstvo za dosego splošnih ciljev neke organizacije. Dandanes so organizacije pod precejšnjim pritiskom tržišča, prisiljene so zmanjašati tako razvojni čas kakor čas posredovanja izdelka na tržišče, povečati morajo raznolikost ponujenih izdelkov, poleg tega pa povečati standardiziranost in interoperativnost izdelkov. Ponovna uporaba je sredstvo za dosego teh ciljev. Povečanje produktivnosti razvoja je danes v večini organizacij realna potreba. "Kriza programske opreme" obstaja - razvoj in vzdrževanje programske opreme sta predraga. Graditev in uporaba ponovno uporabne programske opreme bi morala omogočiti, da programska oprema postane premoženje, pridobitev neke organizacije. Ponovna uporaba lahko pripomore k zadovoljevanju strankinih zahtev po večji kvaliteti in funkcionalnosti programske opreme. Vendar prednosti, ki jih ponovna uporaba prinaša, še niso popolnoma izkoriščene in realizirane. Razširjene ponovne uporabe programske opreme, ki bi zajemala celotno "znanje" neke organizacije, še ni [6]. Objektna tehnologija vsebuje gradnike (objekte - razrede), iz katerih je možno graditi ponovno uporabno programsko opremo, vendar ti gradniki sami po sebi niso dovolj za dosego ponovne uporabe. Potrebna sta še primerna

organizacijska infrastruktura in upravljanje razvojnega procesa programske opreme.

Ponovna uporaba torej zahteva tako programsko opremo, ki jo je možno ponovno uporabiti, kakor tudi proces kreiranja ponovno uporabne programske opreme (razvoj za ponovno uporabo). Brez teh dveh stvari se ponovna uporaba ne bo "kar zgodila".

Prednosti, ki jih lahko z izvajanjem sistematične in učinkovite ponovne uporabe dosežemo, so :

- nižji stroški bodočega vzdrževanja in samega razvoja (manj testiranja in dokumentiranja, sistemi so sestavljeni iz dobro razumljivih delov),
- hitrejši razvoj,
- večja kvaliteta (ponovno uporabna programska oprema je dobro načrtovana, dobro testirana in dokumentirana) in
- višja produktivnost.

3.2. Zakaj "ne" ?

Na drugi strani brega reke so tisti, ki jih ponovna uporaba ni prepričala. Razlogi "proti" so tako tehnične (komponente je težko integrirati in spreminjati, kvalitetnih ponovno uporabnih komponent je premalo, splošne komponente so preveč neučinkovite, komponente je težko najti) kakor tudi netehnične narave (psihološki, sociološki in ekonomski).

Tracz [14] je zbral nekatera zelo pogosta netehnična opravičila - razloge, ki izražajo nenavdušenje razvijalcev in programerjev nad ponovno uporabo:

- Samo nesposobni posegajo po programski opremi, ki jo je ustvaril nekdo drug.
- Ponovna uporaba programske opreme izničuje sposobnost ustvarjanja nove programske opreme.
- Z vpeljavo ponovno uporabne programske opreme bom izgubil(a) službo.
- Ponovno uporabna programska oprema ne more biti učinkovita.
- Nočem biti prvi(a).
- Poskušati ponovno uporabiti programsko opremo nekoga drugega je izguba časa.
- Ne verjamem, da je ponovna uporabnost programske opreme koncept, ki bo preživel.

Prva večja ovira je torej ego - vse preveč profesionalnih razvijalcev programske opreme se bo raje lotilo pisanja programske opreme na novo, kakor da bi ponovno uporabilo že razvito (in testirano) programsko opremo, ki jo je napisal nekdo drug. To je problem managementa, ki ga le-ta lahko odpravi, če je z njim seznanjen.

Druga ovira je ekonomske narave. Nekateri razvijalci se poskušajo izogniti pisanju programske opreme, ki je preveč splošno namenska, ker se bojijo, da se bo s ponovno uporabo take programske opreme količina dela, ki jo morajo vložiti, preveč povečala. Tudi ta problem se da rešiti z ustreznim posredovanjem managementa.

Tretja ovira je problem iskanja. Neka organizacija ima lahko zelo veliko potencialno ponovno uporabnih komponent. Postavlja se vprašanje, kako te komponente shraniti, da bo iskanje učinkovito. Rešitev problema shranjevanja in iskanja komponent je tehnične narave. Danes je za ta problem podanih že kar precej različnih rešitev - že pregledovalniki, brkljalniki, CASE orodja so pri pregledovanju knjižnic lahko zelo učinkoviti (nekatere rešitve najdete v Meyer, 1987; Prieto-Diaz, 1991; Karlsson, 1995; in drugi).

Četrto oviro predstavlja cena ponovne uporabe. Ponovna uporaba je draga. Tracz [14] je stroške ponovne uporabe razdelil v tri skupine: strošek, ki ga imamo, ko želimo "nekaj" narediti ponovno uporabno, strošek pri ponovni uporabi le-tega in strošek definiranja ter implementiranja procesa ponovne uporabe. Ocenil je, da že sam postopek, ko želimo komponento narediti ponovno uporabno, poveča ceno te komponente za vsaj 60 odstotkov. Nekatere organizacije so poročale o 200 in celo več kot 480 odstotnem povečanju stroškov, pri projektu ponovne uporabe firme Hewlett-Packard pa je bilo to povečanje samo 11 odstotno.

Peta ovira je najtežje premostljiva in rešljiva. Gre za pravne probleme, ki se porodijo pri pogodbeni programski opremi. V pogodbi, ki jo podpišeta proizvajalec in kupec programske opreme, navadno piše, da je programska oprema last stranke, kupca. Torej, če razvijalec v novem izdelku za novo stranko ponovno uporabi komponento, ki je del izdelka, ki ga je neki drugi stranki prodal, pomeni to kršitev pogodbe. Ko sta razvijalec in stranka v isti organizaciji, navadno to ni poseben problem [12].

Torej, razen pravnih problemov, ne obstajajo resnejše ovire, ki bi preprečevale implementiranje ponovne uporabe v organizaciji, ki se ukvarja z razvojem programske opreme.

4. KDAJ

4.1. Kdaj začeti s ponovno uporabo ?

Zastavljeno vprašanje bi lahko tudi preoblikovali v vprašanje "kje se ponovna uporaba začne". Pri odgovoru na vprašanje se moramo osredotočiti na tri področja - na tri P-je ponovne uporabe programske opreme [14]. Ti so:

1. produkt (izdelek) ali kaj bomo ponovno uporabili,
2. proces ali kdaj bomo ponovno uporabo izvajali in
3. personal (osebje) ali kdo bo ponovno uporabo izvajal.

Lahko bi jih tudi poimenovali trije K-ji ponovne uporabe : Kaj, Kdaj in Kdo.

Odgovori na ta vprašanja so zelo pomembni, kajti če se odločimo, da bomo zgradili orodje, ki nam bo v pomoč pri ponovni uporabi programske opreme, potem moramo vedeti, kaj bomo poskušali ponovno uporabiti, kdaj bomo to storili in kdo bo to uporabljal.

4.2. Izdelek

Če imamo pri odgovorjanju na zastavljeno vprašanje v mislih izdelke, ki so ponovno uporabni, potem se lahko vprašamo, ali se ponovna uporaba začne s programi (navadno se pri kodi ponovna uporaba neha). Glede na to, da je vrst kode več (izvorna koda, koda objekta, stavek visokonivojskega jezika, funkcija, procedura, paket, modul ali cel program), se moramo zavedati, da je uspeh ponovne uporabe kode odvisen od zrnatosti kode, ki jo ponovno uporabimo. Večja je zrnatost, boljši bo uspeh. Velja namreč, da mora biti napor, vložen v iskanje, razumevanje in integriranje ponovno uporabnih komponent manjši od napora, ki je potreben za načrtovanje in pisanje kode na novo. Zato je bolj priporočljivo ponovno uporabiti objekte z visoko zrnatostjo, kot so paketi, moduli ali razredi.

Naslednji pomemben faktor uspeha je nivo abstrakcije. Ker se nahaja načrtovanje na višjem nivoju abstrakcije kot implementacija, se lahko vprašamo, ali ni umestneje, da s ponovno uporabo začnemo v fazi načrtovanja. Prednost začetka ponovne uporabe pri načrtovanju je ravno višji nivo abstrakcije - načrtovanje vključuje manj implementacijskih podrobnosti. Če sledimo nivoju abstrakcije, potem lahko mirno trdimo, da se ponovna uporaba lahko začne že na nivoju specifikacije ali celo, da se lahko začne z definicijo problema - zahtev.

Kar zagotovo vemo, je, da se ponovna uporaba na splošno konča s ponovno uporabo kode. Kje se bo začela, je odvisno od [14]:

1. količine napora, ki ga želimo vložiti v razvoj ponovno uporabnih izdelkov,
2. kako učinkovito lahko razvoj ponovno uporabnih izdelkov povežemo z implementacijo in
3. kako uspešno lahko implementacijo posplošimo.

Pri prehajanju po fazah slapovnega (kaskadnega) modela navzdol od zahtev do implementacije, lahko ugotovimo, da je vsakemu izdelku dodanih nekaj podrobnosti. Implementacija je torej primerek načrtovanja. Za posamični načrt imamo lahko več implementacij, kakor imamo lahko več načrtov, ki zadoščajo specifikacijam. Ključnega pomena je združevanje skupnih lastnosti (osamitev sprememb), pri čemer ločimo kontekst od koncepta in vsebine, kar pomeni, da implementacijski cilji, kot so specifični operacijski sistem ali odvisnosti od strojne opreme, niso del vsebine. Vsebino sestavljajo algoritem, pretok podatkov ali del specifikacije funkcionalnosti, koncept postane funkcionalna specifikacija. Vsebina postane predloga, vzorec ali splošen objekt. Kontekst postane možna opredelitev parametrov. Po Tracz-ovem mnenju [14] je koda dovolj varno mesto za začetek in, v večini primerov, tudi dovolj varno mesto za zaključek ponovne uporabe.

4.3. Proces

Če opazujemo proces razvoja programske opreme, lahko ugotovimo, da se večina ponovne uporabe začne v fazi implementacije. To je seveda možno, če je bila programska oprema načrtovana za ponovno uporabo - možno jo je prenesti v nov kontekst, in če se je ne da uporabiti brez sprememb, so predvideni in vgrajeni parametri, preko katerih je možno programsko opremo prilagoditi potrebam ali novemu kontekstu. Torej se ponovna uporaba začne že v fazi načrtovanja. Objektno usmerjeno načrtovanje pomaga zmanjšati problem načrtovanja tako, da je le-to neodvisno od implementacije, ne more pa tega problema popolnoma rešiti. Ključnega pomena za nadzor procesa je še vedno parametrizacija [14].

Če začnemo proučevati, kaj je razpoložljivega, bolj zgodaj v procesu razvoja, npr. v fazi specifikacije in analize zahtev, je možno načrtovanje prikrojiti tako, da lahko izkoristimo obstoječo programsko opremo.

V klasični slapovni model življenjskega cikla razvoja programske opreme so strokovnjaki vključili novo fazo, ki podpira ponovno uporabo. To je faza analize domene. Analiza domene je posplošitev analize zahtev - namesto, da bi analizirali zahteve za specifično aplikacijo, ovrednotimo zahteve splošne aplikacije skozi konkretno domeno [8] (analiziramo zahteve za splošne možne aplikacije znotraj domene). Najboljši trenutek za začetek ponovne uporabe je pred začetkom projekta. Takrat lahko definiramo proces razvoja, postavimo knjižnice s ponovno uporabno programsko opremo ter razvijemo standarde in orodja.

Obstaja pa tudi možnost začetka ponovne uporabe po zaključenem projektu - v prvem koraku razvijemo programsko opremo, v drugem koraku pa izluščimo podobnosti med aplikacijami. Na tem mestu je vmesno Biggerstaff-ovo "pravilo treh" [13], ki pravi: "Če nisi razvil treh realnih sistemov v določeni domeni, potem si nesposoben izluščiti tiste podrobnosti o domeni, ki so potrebne za uspešno ponovno uporabo v tej domeni." Drugo pravilo treh pa pravi: "Preden lahko izkoristiš prednosti ponovne uporabe, moraš izdelek najprej trikrat ponovno uporabiti."

Kje se v življenjskem ciklu razvoja programske opreme ponovna uporaba začne, je odvisno od tega [14]:

1. Kako nekdo spremeni proces razvoja programske opreme, da bo izkoristil možnost ponovne uporabe, in
2. Kako nekdo bodisi spremeni bodisi razširi življenjski cikel programske opreme, da bo lahko identificiral objekte, ki jih bo "naredil" ponovno uporabne.

4.4 Osebje

Osebje predstavlja ključne akterje v igri ponovne uporabe. Prvi igralec je programer. Od njega je namreč odvisno, kako uspešno bo identificiral obstoječo ponovno uporabno programsko opremo. Lahko bi se torej ponovna uporaba začela pri programerju. Vendar, če želi programer nekaj ponovno uporabiti, mora to že obstajati. Zato mora obstajati kritična masa kvalitetne programske opreme. Imeti prazen repozitorij ali v njem imeti nekvalitetno in slabo dokumentirano programsko opremo, za programerja ni najbolj vzpodbudno. Navedeni problem je povezan z visokimi stroški, zato mu pri tem lahko pomaga le najvišji management, ki se odloči, da bo podprl izvajanje ponovne uporabe in visoke začetne stroške. Denar pa tudi ni dovolj. Potrebno je izobraževanje, osveščanje - tečaji iz analize domene, konstrukcije aplikacij, programiranja s parametriziranjem, na razpolago pa morajo biti že izdelane knjižnice komponent, ki olajšajo gradnjo novih aplikacij.

Ponovna uporaba se lahko začne tudi pri razvijalcu orodij, ali pri stranki, prodajalcih (ti poznajo tržišče in potrebe po ponovno uporabnih komponentah), lahko pa celo pri sistemskem analitiku (zna analizirati problemsko domeno, zna določiti logične podsisteme in funkcije in zna določiti vsebino ali zahteve za module ter predvideti različne kontekste, v katerih lahko te module uporabimo).

V proces vzpostavljanja ponovne uporabe je torej lahko vključenih kar nekaj ljudi. Nekateri odigrajo bolj tehnične, drugi povsem netehnične vloge. Dejstvo pa je, da morajo za dosego učinkovite ponovne uporabe med seboj tesno sodelovati.

5. KAKO

5.1. Kako doseči učinkovito in sistematično ponovno uporabo ?

Za uspešno in učinkovito ponovno uporabo so potrebne organizacijske spremembe, planiranje ponovne uporabe, nove vloge, širjenje "kulture" ponovne uporabe in izobraževanje, razvoj programske opreme za ponovno uporabo, katalogiziranje ponovno uporabnih komponent, predvsem pa se moramo zavedati, da ponovna uporaba ni zastonj, ampak so začetni stroški lahko zelo visoki.

Nekateri ključni pogoji [3], ki morajo biti v organizaciji izpolnjeni, če želimo, da bo ponovna uporaba uspešna, so:

- višji vodilni delavci morajo razumeti in predvsem podpreti program ponovne uporabe,
- potrebne so realne ocene stroškov predvidene investicije,
- potreben je model življenjskega cikla, ki podpira in vzpodbuja ponovno uporabo (kot najprimernejša sta se izkazala spiralni model in model hitrega prototipiranja),
- potrebna je knjižnica ponovno uporabnih komponent,
- potrebna so sredstva za dokumentiranje in dostop do komponent preko njihove specifikacije,
- potrebna so primerna orodja,
- nagrajevalni sistem ekipe za ponovno uporabo mora biti primeren in stimulativen,
- razvijalci morajo biti izkušeni, spretni in predvsem motivirani.

Nujni koraki, ki jih mora izvesti organizacija, ki želi doseči uspešno in učinkovito ponovno uporabo (in izkoristiti prednosti, ki jih ta ponuja), so navedeni v nadaljevanju.

5.2. Ekipa

Sestaviti je potrebno ekipe za ponovno uporabo, določiti vloge posameznih članov ekip in njihove aktivnosti [9].

Aktivnosti članov ekip za ponovno uporabo so: *definiranje* (kaj naj bo ponovno uporabljeno in kako), *identificiranje* (določitev komponent, ki bodo ponovno uporabne), *pridobivanje* (graditev, nakup ali pogodbena pridobitev ponovno uporabnih komponent), *certificiranje* (sledenje smernicam spremenljivosti), *klasificiranje* (organiziranje in označevanje komponent), *shranjevanje*, *komuniciranje* (iskanje močnih ponovnih uporabnikov), *lociranje* (iskanje shranjenih komponent), *ponovno pridobivanje komponent*, *razumevanje* (določitev namena in značilnosti komponent), *uporaba* (integriranje komponent v nov kontekst), *ažuriranje komponent* (prilagajanje, spreminjanje, razširjanje, popravljanje, testiranje komponent), *ažuriranje ponovnih uporabnikov* (ažuriranje vseh sistemov, ki uporabljajo spremenjene komponente).

Ključne vloge v ekipah za ponovno uporabo so : *nadzornik* (planira in nadzoruje proces ponovne uporabe), *administrator* (identificira in pridobiva komponente), *vzdrževalec* (vzdržuje obstoječe komponente v knjižnici), *ocenjevalec* (zagotavlja ustreznost komponent, proizvaja in izvaja testne plane), *knjižničar* (certificira, kategorizira in shranjuje komponente v knjižnico, zagotavlja ujemanje dokumentacije in razširja kataloge knjižnic), *inženir* (gradi ponovno uporabne komponente).

5.3. Planiranje

Uspešno in sistematično ponovno uporabo je potrebno planirati. Zato je potrebno sestaviti program ponovne uporabe, pri čemer so najpomembnejše aktivnosti:

- določitev nadzornika, nato pa še administratorja, inženirja in knjižničarja (ključne vloge),

- oblikovanje celotne ekipe (sodelujejo lahko tudi stranke, dobavitelji, upravljalci),
- definiranje ponovne uporabe (kaj in kako ponovno uporabiti) in identificiranje ponovno uporabnih komponent,
- izvajanje različnih klasifikacijskih shem,
- certificiranje začetne množice komponent (približno 3 mesece po začetku izvajanja programa),
- formaliziranje sheme certificiranja in določitev načina merjenja ter ocenjevanja komponent, ki so že v repozitoriju (po šestih mesecih) - pilotski projekti,
- ocenitev programa metrik (po enem letu).

5.4. Katalogiziranje

Množice ponovno uporabnih komponent je potrebno katalogizirati [10]. Katalog množic ponovno uporabnih komponent programske opreme (SAC-Software Asset Catalog) je elektronski katalog, podoben kartotečnemu katalogu v knjižnici, v katerem vsebuje vsak kartonček kratek opis enote, podatek o tem, kje se nahaja, kdo je njen avtor in druge informacije.

Za tak katalog morajo biti definirani proces dodajanja množice komponent katalogu, iskalni mehanizem, katalog mora biti dostopen večim uporabnikom, zagotovljena mora biti varnost, vsebovati mora pripomoček za vključevanje drugih izdelkov programske opreme, definiran mora biti klasifikacijski mehanizem, realizirana mora biti registracija oz. prijava uporabnika kataloga in katalog mora biti dostopen širšemu krogu uporabnikov.

Koraki pri kreiranju ali ustanovitvi kataloga množic ponovno uporabnih komponent so :

1. Kreiranje osnovnega klasifikacijskega načrta na podlagi domene (organizacija podatkov, enote v katalogu, tip kataloga, kataloška orodja in njihove iskalne sposobnosti) v sodelovanju z uporabniki kataloga.
2. Določitev repozitorija množice ponovno uporabnih komponent (SAR - Software Asset Repository) - lokacije, na kateri bodo komponente shranjene.
3. Določitev začetnega seznama množic ponovno uporabnih komponent (20 do 30 komponent).
4. Kreiranje ali nakup orodja za katalogiziranje ponovno uporabnih komponent (orodje, ki zagotavlja osnovne iskalne in klasifikacijske sposobnosti).
5. Definiranje procesa dodajanja množic ponovno uporabnih komponent v katalog (definiranje procesa katalogiziranja, ki navadno vključuje odobritveno-potrditveni korak, le-ta pa omogoča pregled množice ponovno uporabnih komponent pred vključitvijo v katalog).
6. Ponovni pregled klasifikacijskega načrta (ob povečanju števila množic ponovno uporabnih komponent) in izboljšanje le-tega, če je potrebno.
7. Postopno dodajanje množic ponovno uporabnih komponent v katalog.

Zgraditev kataloga ponovno uporabnih komponent je lahko prvi korak k doseganju ponovne uporabe, vendar brez vzpostavitve samega procesa ponovne uporabe in procesa kreiranja ponovno uporabnih komponent (razvoja za ponovno uporabo) uspeh ni zagotovljen. Sam proces ponovne uporabe vzpostavimo z dolgoročnim programom ponovne uporabe, ki ga pa spremljajo tudi nujne spremembe.

Prieto-Diaz (93) poudarja, da problem inženirstva programske opreme ni pomanjkanje ponovne uporabe, ampak pomanjkanje razširjene, sistematične ponovne uporabe.

6. "MITI" PONOVNE UPORABE PROGRAMSKE OPREME

Pred desetimi leti je Will Tracz, že tedaj zagovornik ponovne uporabe, zapisal devet "mitov" ponovne uporabe programske opreme, ki izražajo določene tehnične, organizacijske in psihološke cilje in trende raziskave inženirstva programske opreme v tistem času (kaj naj bi ponovna uporaba programske opreme bila). Ti "miti" delno tudi razložijo, zakaj se ponovna uporaba do danes, ni prav močno razširila, vsaj ne toliko kot so to predpostavljali tedanji "preroki iz sveta programiranja".

1. Ponovna uporaba programske opreme je problem tehnične narave.
2. Za ponovno uporabo programske opreme so potrebna posebna orodja.
3. Rezultat ponovne uporabe kode je precej povečana produktivnost.
4. Umetna inteligenca bo rešila problem ponovne uporabe.
5. Japonci so rešili problem ponovne uporabe.
6. Ada in C++ sta rešila problem ponovne uporabe.
7. Načrtovanje programske opreme na osnovi ponovno uporabnih delov lahko enačimo z načrtovanjem strojne opreme na osnovi integriranih vezij.
8. Ponovno uporabljena programska oprema je isto kot ponovno uporabna programska oprema.
9. Ponovna uporaba se bo "kar zgodila".

Leta 1994 je Tracz ponovno obdelal in preučil teh devet "mitov" in jih glede na takratno stanje (ki se marsikje ne razlikuje veliko od današnjega) ponovne uporabe preoblikoval oziroma dopolnil:

1. Ponovna uporaba je problem tako tehnične kot netehnične narave.
2. Za ponovno uporabo niso potrebna nobena "posebna" orodja. Zadostuje nam razpoložljiva tehnologija podatkovnih baz, ki jo lahko uporabimo pri organiziranju in iskanju programske opreme, shranjene v velikih repozitorijih.

3. Samo ponovna uporaba kode ne bo povzročila velikega povečanja produktivnosti in kvalitete.
4. Umetna inteligenca ima lahko določeno vlogo pri reševanju problema ponovne uporabe.
5. Japonci so naredili prvi korak k rešitvi problema ponovne uporabe.
6. Noben programski jezik ne more sam rešiti problema ponovne uporabe.
7. Načrtovanje programske opreme s pomočjo ponovno uporabnih delov ni ravno enako kot načrtovanje strojne opreme s pomočjo integriranih vezij.
8. Ponovna uporaba programske opreme, ki ni bila načrtovana za ponovno uporabo, je težja, kakor ponovna uporaba programske opreme, načrtovane za ponovno uporabo.
9. Ponovna uporaba se ne bo "kar zgodila".

7. NAMESTO ZAKLJUČKA ali KRITIČNI FAKTORJI USPEHA

Večina študij uspešnih programov ponovne uporabe, ki jih je opravila družba Technology Transfer International (TTI) iz Colorada na primerih precejšnjega števila japonskih organizacij (tudi Fujitsu, Hitachi, Oki in Toshiba), je pokazala, da so za uspešnost potrebni [9]:

- Kritična masa: razpoložljivost dovolj velike množice ponovno uporabnih komponent.
- Razumljivost: sposobnost razumevanja kode, ki so jo pisali drugi.
- Integriranje: sposobnost združevanja ponovno uporabnih komponent z novimi zahtevami.
- Kulturno prelivanje: posredovanje pridobljene kulture ponovne uporabe novincem in pogodbenim strankam.
- Vedenje: premostitev konzervativnih pogledov starejših razvijalcev programske opreme.

Že med študijo so se med člani ekipe pojavile prepričljive ugotovitve o izboljšanju produktivnosti kot posledice sistematične ponovne uporabe:

- razvojni čas se je skrajšal od 10 do 50 odstotkov,
- produktivnost se je zvišala za 15 do 50 odstotkov,
- kvaliteta se je izboljšala za 20 do 35 odstotkov.

Nekaj ponovne uporabe se je pojavilo tudi v fazi analize zahtev. To je možno doseči le, če si že med analizo postavimo za cilj identificirati možnosti za ponovno uporabo. Zanimiva je tudi ugotovitev, da je večina ponovno uporabljenega koda sestavljena iz majhnih komponent (manj kot 1000 vrstic). Po mnenju članov ekipe je omenjene pridobitve možno doseči le, če si jih predhodno postavimo kot dolgoročne cilje.

8. VLOGA PONOVNE UPORABE V SODOBNIH TEHNOLOGIJAH

Trenutno v svetu vse bolj prodira tehnologija porazdeljenih objektov (distributed object technology). Le-ta nam omogoča izgradnjo programske opreme z integracijo komponent – objektov [5]. Ti porazdeljeni objekti niso vezani na noben določen program, programski jezik ali implementacijo.
Eden izmed ključnih problemov, ki jih tehnologija porazdeljenih objektov naslavlja, je torej tudi ponovna uporaba (poleg prenosljivosti in povezljivosti programske opreme). Porazdeljeni objekti so posebne ponovno uporabne komponente, ki se nahajajo nekje v omrežju in so sposobne med seboj komunicirati.

Na področju inženirstva programske opreme pa postaja vse bolj vroča tema tudi tehnologija komponent oziroma razvoj programske opreme na podlagi komponent (CBSD – Component Based Software Development). Ponovna uporaba in uporaba programskih komponent imata vedno večji vpliv na strukturo programskih sistemov, prav tako pa tudi na način njihove gradnje.

Graditev programske opreme iz komponent obljublja več ponovne uporabe in višjo produktivnost. Sistemi naj bi bili zgrajeni iz med seboj sodelujočih, dobro testiranih komponent, kar zmanjša kompleksnost sistemov (v primerjavi s sodobnimi sistemi, ki so zgrajeni kot monolitni bloki iz velikega števila med seboj nepovezanih delov) in stroške vzdrževanja.

V bodočnosti naj bi torej razvoj programske opreme s ponovno uporabo prevzel vodilno mesto, ki ga danes še vedno zaseda razvoj programske opreme na novo. Razvoj programske opreme naj bi prešel iz kompozicije enostavnih stavkov kode v sintezo velikih komponent iz številnih majhnih (t.i. megaprogramiranje) [8,14] – sistemi se bodo gradili s sestavljanjem komponent, ki so neodvisne od okolja, v katerem so bile razvite in katerih funkcionalnost se da med razvojem direktno uporabiti (lahko jo vključimo tako v specifikacijo kakor v implementacijo aplikacije). Na tak način bo aplikacijam oziroma sistemom omogočeno, kakor pravi Kain [7], da bodo "vsota svojih delov".

Omenjene sodobne tehnologije razvoja programske opreme in rezultati, ki so jih nekatera velika podjetja (IBM, Hewlet-Packard, U.S. DoD, številna japonska podjetja in drugi) pri institucionalizaciji ponovne uporabe dosegla, nam dajejo slutiti, kako pomembna je ponovna uporaba, še posebno pa razvoj ponovno uporabnih komponent.

Literatura:

[1] *BRAMBAUGH, David. E.:*
Object-Oriented Development - Building CASE tools with C++, *John Willey & Sons, 1994*

[2] *CARMICHAEL, Andy:*
Object Development Methods, *SIGS Books, New York, 1994*

[3] *GRAHAM, Ian:*
Migrating to Object Tehnology, *Addison Wesley Pub. Co., 1995*

[4] *GRISS, L. Martin:*
"Software Reuse-Objects and frameworks are not enough", Object Magazine, *Februar 1995, str. 77-79*

[5] *JURIČ, B. Matjaž:*
"Objektni model porazdeljenega procesiranja", Uporabna informatika, *Jan-Feb-Mar 1997, str. 29-36*

[6] *KAIN J.Bradford:*
"Pragmatics of reuse in the enterprise", Object Magazine, *Februar 1994, str. 55-58*

[7] *KAIN J.Bradford:*
"Components: The Basics", Object Magazine, *April 1996, str. 64-69*

[8] *KARLSSON, Even-Andre:*
Software Reuse - a holistic approach, *John Willey & Sons Pub. Co., 1995*

[9] *MCGIBBON, B:*
"Making reuse Happen", Object Expert, *Jul-Avg 1996, str. 43-46*

[10] *NORRIS,:*
"Cataloging Reusable Software Assets", Object Magazine, *April 1996, str. 51-52,93*

[11] *RAMBAUGH, James, BLAHA, Michael, PREMERLANI, William, EDDY Frederick, LORENSEN, William:*
Object-Oriented Modeling and Design, *Prentice-Hall International, 1991*

[12] *SCHACH, Stephen R.:*
Classical and Object-Oriented Software Engineering, *Irwing, Chicago, 1996*

[13] *SULLO, Gary C.:*
Object Engineering - Designing Large-Scale Object-Oriented Systems, *John Wiley & Sons, 1994*

[14] *TRACZ, Will:*
Confessions of a Used Program Salesman: Institutionalizing Software Reuse, *Addison-Wesley Pub. Co., 1995*

◆

Evelin Vatovec Krmac je diplomirala leta 1991 na Fakulteti za računalništvo in informatiko v Ljubljani, kjer pripravlja magistrsko delo na temo Ponovna uporaba v objektno usmerjenem razvoju. Zaposlena je kot asistentka stažistka za informatiko na Fakulteti za pomorstvo in promet v Portorožu.

◆

Vabilo avtorjem

Uredniški odbor revije Uporabna informatika načrtuje razširitev obsega revije. Rezultati ankete med bralci revije so pokazali, da si želijo med drugim prispevke z naslednjih področij: ocena orodij in različnih rešitev, predstavitve primerov iz prakse, predstavitev rešenih informacijskih problemov, pregled stanja na nekaterih področjih v Sloveniji.

S tem vabilom se posebej obračamo na informatike v praksi, da s članki v naši reviji predstavijo svoje ugotovitve in izkušnje.

Navodila za prispevke objavljamo na zadnji strani revije.