VZORCI IN PONOVNA UPORABA IZKUŠENJ

Tomaž Domajnko, Ivan Rozman, Marjan Heričko, József Györkös
Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko

Povzetek

V prispevku so podani osnovni koncepti vzorcev. Vzorci opisujejo uspešne rešitve težav programskega inženirstva. Predstavljena je vloga uporabe vzorcev v razvojnem procesu. Vzorci ne pomenijo le novega pristopa k razvoju, temveč tudi izboljšujejo komunikacijo v razvojni skupini. Končni cilj uporabe vzorcev je razvoj priročnikov za programsko inženirstvo, ki bodo vsebovali preizkušene rešitve problemov v določenem kontekstu.

Abstract

In the article, basic concepts of software patterns are introduced. A software pattern describes a successful solution to a set of similar software development problems. Patterns do not only describe useful techniques, they also improve communication between members of development teams. The final goal of patterns usage is to provide handbooks for software engineering, containing tested solutions of problems in certain context.

PONOVNA UPORABA

Objektna tehnologija je vodilna tehnologija razvoja informacijskih sistemov. Uveljavitev objektne tehnologije je v razvoj programskih sistemov prinesla nove pristope, tehnike in zahtevo po novih znanjih. Ena od ključnih prednosti objektne tehnologije je, da gradi kompleksne sisteme iz razmeroma preprostih gradnikov – objektov. Pri tem se poraja ideja, da je preproste gradnike možno uporabiti na več načinov in s tem doseči boljšo produktivnost. Govorimo o ponovni uporabi. V okviru objektne tehnologije obstaja več pristopov k ponovni uporabi, na katero gledamo kot na sposobnost razvoja novih sistemov na podlagi že obstoječih komponent. Pri tem se moramo zavedati, da se ponovna uporaba ne bo zgodila sama od sebe. Potrebno je skrbno načrtovati aktivnosti, ki bodo privedle do ponovne uporabe in s tem krajših razvojnih časov, nižjih stroškov razvoja in vzdrževanja ter povečane kakovosti programski proizvodov [9].

Uvajanje aktivnosti v razvojni proces pomeni spremembe. Spremembe lahko grobo razdelimo na dve skupini. Prvo skupino tvorijo spremembe, ki prilagajajo razvojni proces za ponovno uporabo, medtem ko drugo skupino tvorijo spremembe, namenjene dvigu stopnje ponovne uporabe programskih proizvodov. Ponovno lahko uporabimo vse programske proizvode, pod pogojem da so prilagojeni ponovni uporabi. To pomeni, da so primerno dokumentirani, dostopni in uporabni v različnih kontekstih.

V času razvoja nastajajo programski proizvodi, ki vplivajo na način ponovne uporabe. Tako lahko ponovno uporabo programskih proizvodov razdelimo na:

- *ponovno uporabo delov programske kode*, kar pomeni ponovno uporabo na najnižjem nivoju abstrakcije, saj ponovno uporabljamo konkretno implementacijo dela problema,
- *ponovno uporabo razredov*, pri kateri ponovno uporabljamo implementacijo abstrakcije določenega dela realnega sveta,
- *ponovno uporabo idej, ki vodijo do rešitve*, to je ponovno uporabo na najvišjem nivoju abstrakcije, saj ponovno uporabimo idejo, ki je v preteklosti vodila do uspešne rešitve v določenem kontekstu.

Ponovna uporaba prvih dveh nivojev abstrakcije je danes splošno poznana in uporabljana. Tretja možnost ponovne uporabe pa pomeni novost, saj do sedaj nismo govorili o ponovni uporabi idej rešitev problemov. V situacijah, ko rešitev zadanega problema ne obstaja in ne najdemo niti delne rešitve je potrebno rešitev razviti samostojno. Vemo, da izkušnje razvojne skupine pomembno vplivajo na čas razvoja, kakovost rešitve in višino stroškov razvoja. Prispevek opisuje pristop k ponovni uporabi idej preizkušenih rešitev, ki lahko v veliki meri pripomore k zmanjšanju stroškov razvoja.

VZORCI

V vsakdanjem življenju opažamo, da vedno znova uporabljamo zelo podobne pristope k reševanju problemov. Vsakomur je jasno, da izkušnje pomagajo tudi pri reševanju novih problemov, saj človek poseduje sposobnost uporabe znanja v drugem kontekstu, kot si ga je pridobil (*prednost človeškega razmišljanja pred računsko močjo računalnika*).

Nezadostno število strokovnjakov z izkušnjami v informacijskih tehnologiji zahteva nov pristop k reševanju problema izkušenj. V prejšnjem poglavju smo videli, da je ena od možnosti ponovna uporaba. Pa ne samo ponovna uporaba implementacij konkretnih razredov in funkcij. Ponovno lahko uporabimo način, kako objekti med seboj komunicirajo, kako objekte poimenujemo, povezujemo v hierarhije ali združujemo v komponente. Ta način, kako ponovno uporabljati ideje rešitev, zagotavlja izkoriščanje nakopičenih izkušenj, zapisanih v obliki vzorcev. Vzorec je torej najsplošnejša možnost ponovne uporabe pridobljenega znanja.

Definicija vzorca

Področje vzorcev je danes v velikem razcvetu, kar posledično pomeni, da standardizirane definicije vzorca še ni. Vseeno sta v svetu in pri nas najbolj uveljavljeni naslednji definiciji:

- Vzorec je ideja rešitve, ki se je izkazala uporabna v določenem kontekstu in bi bila najverjetneje koristna tudi v drugih kontekstih. Vzorec predstavlja večkratne odločitve strokovnjaka, ki sicer vodijo do različnih rešitev, a vsebujejo določen nivo kakovosti [4],
- Vzorec opisuje problem, ki se večkrat pojavlja v našem okolju, podaja jedro njegove rešitve na tak način, da lahko idejo rešitve uporabimo v milijon različnih primerih, ne da bi pot od ideje do rešitve prehodili na enak način (Cristopher Alexander, [3]).

Dokumentiranje vzorcev

Vzorci pokrivajo tako veliko definicijsko območje, da je njihova struktura zelo individualna. To je tudi razlog, da v svetu obstaja veliko število predlogov predstavitev vzorcev. Pri tem se pojavljajo tako popolnoma grafične predstavitve, kot tudi predstavitve v obliki spletnih strani.

V splošnem mora definicija vzorca vključevati naslednje elemente (bolj kot na obliko se osredotočimo na vsebino predstavitve vzorca):

- *Ime* - omogoča sklicevanje na vzorec z eno samo besedo ali kratko besedno zvezo.
- *Namen* - razloži problem, ki ga lahko s pomočjo vzorca rešimo - kdaj uporabimo vzorec, kakšne so njegove naloge in cilji, kaj z njim dosežemo oz. kakšen je njegov vpliv v določenem kontekstu.
- *Uporabnost* - podaja stanje oziroma predpogoje, kjer nastopi takšen problem, da je rešitev, ki jo vzorec nudi, ustrezna.
- *Strukturo, sodelovanje, udeležence* - podajajo način realizacije rešitve s pomočjo statičnih povezav in dinamičnih pravil. Opis lahko zajema slike, diagrame in besedilo, ki prikažejo strukturo vzorca, njegovo vlogo in delovanje, da opišemo postopek reševanja problema. Poleg statične strukture rešitev podaja tudi dinamično obnašanje.
- *Učinek* – podaja stanje ali konfiguracijo sistema po uporabi vzorca, vključno s posledicami, ki jih prinese uporaba vzorca, ter drugimi problemi in vzorci, ki se pojavijo v novem kontekstu. Dokumentacija stanja po uporabi vzorca pomaga pri primerjavi z začetnim stanjem drugega vzorca.
- *Implementacijo* - podaja probleme, s katerimi se lahko srečamo pri sami implementaciji ideje rešitve.
- *Razlago korakov, pravil* - utemelji korake oziroma pravila uporabe vzorcev.
- *Sorodne vzorce* - opiše statične in dinamične povezave med vzorci. Medsebojno povezani vzorci si delijo različne učinke.
- *Primere uporabe* - beleži opis že znane uporabe vzorcev v obstoječih sistemih. Pomaga pri ugotavljanju primernosti vzorca za reševanje problema.

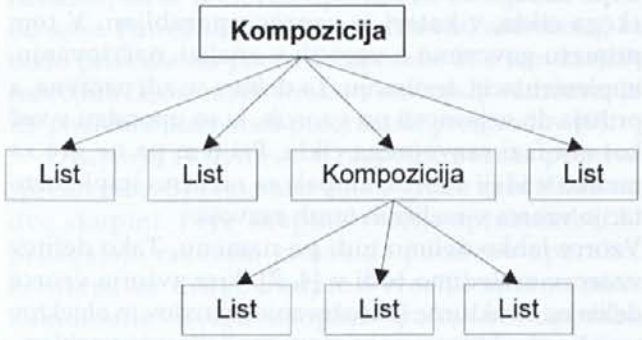
Poglejmo si primer vzorca za faze načrtovanja, ki pomaga pri ustvarjanju delne hierarhije razredov v našem sistemu. Uporaben je takrat, ko ima uporabnik (objekt) opravka z velikim številom različnih objektov. Vzorec s svojim opisom podaja idejo, da objekte združimo v hierarhijo in odpravimo razločevanje med skupkom objektov (kompozicijo) in atomarnim objektom (poimenovan tudi list). Tipično uporabo vzorca prikazuje slika 1.

Ena od pomembnih aktivnosti uporabe vzorcev je njihova klasifikacija. Klasifikacija pomaga tako uporabnikom kot tudi razvijalcem vzorcev pri iskanju vzorcev. Obstaja več različnih klasifikacij vzorcev. Kot osnovno klasifikacijo vzemimo razvojno fazo življenjskega cikla, v kateri je vzorec uporabljen. V tem primeru govorimo o vzorcih v analizi, načrtovanju, implementaciji, testiranju. Ta delitev se zdi naravna, a prihaja do nejasnosti pri vzorcih, ki so uporabni v več kot eni fazi razvojnega cikla. Pri tem pa ne gre za razliko v ideji vzorca, ampak za različno implementacijo vzorca v različnih fazah razvoja.

Vzorce lahko delimo tudi po namenu. Tako delitev vzorcev zasledimo tudi v [4, 7], kjer avtorja vzorce delita na strukturne (združevanje razredov in objektov v večje strukture z namenom modeliranja problemskega področja), vedenjske (vedenjski vzorci se ukvarjajo z algoritmi in dodelitvami odgovornosti med

Ime	KOMPOZICIJA
Namen	Združi objekte v drevesno strukturo za predstavitev delne hierarhije. Kompozicija dovoljuje uporabniku enako obravnavo posameznih in sestavljenih objektov.
Uporabnost	Vzorec uporabimo, če želimo predstaviti del hierarhije objektov, če želimo, da uporabnik ne bi razločeval med kompozicijo objektov in posameznimi objekti.
Vpliv	Nastane delna hierarhija in zato izvajanje dodatne kode, sicer je vpliv na izvajanje zanemarljiv.
Udeleženci	*Komponenta* (deklarira vmesnik za objekte v kompoziciji, implementira osnovno obnašanje vmesnika za vse razrede, deklarira vmesnik za dostop in upravljanje komponent naslednikov), *List* (predstavlja zadnje komponente v kompoziciji - listi nimajo naslednikov, definirajo obnašanje primitivnih objektov v kompoziciji), *Kompozicija* (definira obnašanje komponente, ki ima naslednike, implementira operacije, ki se nanašajo na naslednike), objekt *Uporabnik* (upravlja z objekti v kompoziciji preko vmesnika razreda *Komponenta*).
Sodelovanje	Uporabnik uporablja vmesnik razreda *Komponenta* za komunikacijo z objekti v kompozicijski strukturi. Če je sprejemnik *List*, potem se zahteva obravnava takoj, če pa je sprejemnik *Kompozicija*, potem se zahteva posreduje naprej do naslednikov, po možnosti se dodatne operacije izvedejo pred ali za posredovanjem.
Učinek	Vzorec definira razredno strukturo primitivnih in sestavljenih objektov. Primitivni objekti se lahko združujejo v kompleksnejše objekte. S tem poenostavi uporabnika združenih objektov, saj lahko le ta obravnava primitivne in sestavljene objekte enako. Omogoča enostavno dodajanje novih vrst komponent, saj novo definirani podrazredi (tipa *List* ali *Kompozicija)* neposredno podedujejo lastnosti svojih nadrazredov v strukturi. Slabost vzorca je, da lahko naredi načrtovanje preveč splošno. Slabost enostavnega dodajanja novih komponent je ta, da težko omejimo število komponent celotne kompozicije.
Implementacija	*Določiti je potrebno zvezo s starši*, saj povezave med komponento naslednikom in starši poenostavi prečkanje in upravljanje kompozicijske strukture. Pogosto je uporabno deliti komponente. Ena od nalog vzorca je omogočiti, da se uporabnik ne zaveda ali uporablja razrede tipa *List* ali *Kompozicija*. Za dosego te naloge mora razred *Komponenta* definirati čim več osnovnih operacij za razrede *List* in *Kompozicija*. Razred *Komponenta* ponavadi priskrbi osnovno implementacijo operacij, ki jih podrazredi predefinirajo.
Razlaga korakov	Poiščemo skupino objektov, katere največkrat uporabljamo skupaj in uporabimo vzorec. Pri tem nastane delna hierarhija, ki rešuje težave obvladovanja velikega števila objektov različnih tipov.
Sorodni vzorci	Veriga odgovornosti, Dekorator, Zrno, Iterator, Obiskovalec.
Primeri uporabe	Modeliranje prodajnih artiklov, organiziranje proizvodnje, vodenje evidence o zaposlenih.

Tabela 1: Vzorec *Kompozicija*



Slika 1: Tipična struktura sestavljenih objektov

objekti, pri tem pa opisujejo vzorce za komunikacijo med objekti) in arhitekturne vzorce (zagotavljanje trdne in hkrati fleksibilne arhitekture celotnega sistema).

Poglejmo si predlog večnivojske delitve vzorcev. Kot primarno delitev predlagamo delitev vzorcev glede na razvojno fazo (vzorci analize, načrtovanja in implementacije). Drugi nivo delitve vzorcev je delitev glede na namen uporabe (kot osnovo uporabimo delitev [4, 7], kateri dodamo nove skupine vzorcev), v tretjem nivoju pa vzorce razdelimo glede na področje uporabnosti (razred, objekt). S tako delitvijo pridobimo preglednost in povečamo učinkovitost dostopa do vzorcev. Tabela 2 prikazuje uporabo delitve na vzorcih

faze načrtovanja, pri čemer tabela prikazuje le tri vrste vzorcev glede na namen, kar je prikazano kot vodoravna delitev. Navpično vzorce razdelimo po kriteriju področja uporabnosti.

	Ustvarjalni	Strukturni	Vedenjski
Razred	Tovarniška metoda	Prilagoditev	Interpreter
	Abstraktna tovarna Graditelj	Most Kompozicija	Iterator Posredovalec
Objekt	Prototip Edinec	Dekorator Fasada Namestnik	Opazovalec Strategija Obiskovalec

Tabela 2: Delitev vzorcev načrtovanja glede na namen in definicijsko območje

V tabeli 2 je potrebno razumevanje razlike med razredi (po definiciji predstavlja razred deklaracijo objektov, ki imajo enake lastnosti/atribute in vedenje/metode) in objekti (po definiciji predstavlja objekt primerek razreda z določenimi vrednostmi atributov, ima svojo identiteto in stanje, ki določa njegovo vedenje). Kriterij *področje uporabnosti* določa definicijsko območje uporabe vzorca (vzorec povezuje razrede ali objekte).

Vzorci, kot so Tovarniška metoda, Prilagoditev in Interpreter delujejo izključno na nivoju razreda. Tako vzorec Tovarniška metoda zagotavlja oblikovanje (kreiranje) primerkov razreda (objektov!) na podlagi definicije razreda in podanih podatkov (tako metodo praviloma imenujemo konstruktor). Vzorec Prilagoditev je razredni vzorec, ki z gradnjo vmesniških razredov (komponent) gradi strukturo sistema, medtem ko je vzorec Interpreter namenjen prikazovanju slovnice kot razredne hierarhije (prevajalnik je v tem primeru implementiran kot operacija nad primerki razredov. Vzorec uporabimo, kadar obstaja jezik za interpretacijo in lahko predstavimo njegove stavke kot abstraktna sintaksna drevesa).

Druga skupina vzorcev pa je skupina objektnih vzorcev, to je vzorcev, katerih področje uporabnosti so izključno objekti. Med temi vzorci najdemo vzorec *Namestnik*, ki podaja idejo o vpeljavi namestniškega objekta namesto izvornega (ker se nahaja v drugem delu računalniškega omrežja ali pa zahteva njegov prenos preveč računalniških sredstev). Uporabnost vzorca je danes poznana in se uporablja v porazdeljenih sistemih (CORBA, COM+), kjer aplikacija ne zahteva izvornega objekta, ampak le njegov namestniški objekt.

Poglejmo še nekaj osnovnih razlik med razrednimi in objektnimi vzorci. Razredni vzorci se ukvarjajo s povezavami med razredi in pripadajočimi podrazredi. Te povezave so pogosto povezane z dedovanjem, so statične in določene v času prevajanja. Objektni vzorci pa se ukvarjajo z povezavami objektov, ki so bolj dinamične in se lahko spreminjajo v času izvajanja. Strukturni vzorci razredov uporabljajo dedovanje za združevanje razredov, medtem ko strukturni vzorci objektov opisujejo pristope za združevanje objektov. Razredni vzorci za obnašanje uporabljajo dedovanje za razlago algoritma in toka kontrole, medtem ko objektni opisujejo kako sodeluje skupina objektov, da dosežemo cilj, ki ga posamezen objekt ne more doseči.

RAZVOJ IN VZORCI

Slika 2: **Pristop k razvoju informacijskih sistemov**

Osnovno skupino aktivnost predlaganega pristopa k razvoju informacijskih sistemov tvorijo aktivnosti razvoja sistemov z uporabo vzorcev (slika 2). Vsebuje aktivnosti, ki zagotavljajo učinkovito in kvalitetno identifikacijo vzorcev in njihovo uporabo v procesu razvoja. Kot najpomembnejši aktivnosti se pojavljata identifikacija primernih vzorcev iz množice vseh dosegljivih vzorcev in aktivnost vrednotenja primernosti vzorca za uporabo v danem kontekstu. Pri tem se je potrebno zavedati dejstva, da vzorci pokrivajo celotno področje razvoja programskih proizvodov, od načina kodiranja programske kode do usmeritev in korektivnih ukrepov pri vodenju projektov programskega inženirstva. Pri tem je seveda potrebno identificirati vrste potencialnih uporabnikov vzorcev, saj s tem po eni strani ožimo izbor vzorcev za posamezne skupine, hkrati pa imamo možnost prilagajanja predstavitve vzorca ciljni skupini in stopnji izkušenj posameznikov.

Seveda prihaja v procesu razvoja do novih idejnih rešitev in s tem možnosti razvoja novih vzorcev (to so lahko nove ideje ali pa način uporabe obstoječih idej za reševanje problema v določenem kontekstu). Ti vzorci so še posebej pomembni, saj neposredno vsebujejo znanje in izkušnje poslovne organizacije, so *naravnani* na našo razvojno organizacijo in ne zahtevajo prilagajanja obliki projektne organizacije.

Drugo skupino aktivnosti tvorijo aktivnosti klasifikacije najdenih vzorcev, uvrstitev v določene razrede ter verifikacije in validacije obstoječih vzorcev. Pri validaciji preverjamo predvsem semantičen pomen vzorcev, medtem ko pri verifikaciji vzorcev preverjamo tako skladnost predstavitve kot tudi skladnost z drugimi vzorci.

Dinamika uporabe in razvoja vzorcev uvaja potrebo po aktivnostih upravljanja z vzorci. Aktivnosti te skupine skrbijo tako za pravilno uporabo vzorcev, kakor tudi za vzdrževanje konsistence predstavitve vzorcev. Vsebujejo aktivnosti nadzorovanja in sledenja uporabe vzorcev (vsaka uporaba vzorca pomeni nov vir informacij o koristnosti vzorca v novem kontekstu) in aktivnosti verifikacije vzorcev. Pri tem izvajamo tako sintaktično kot tudi semantično preverjanje predstavitve novega ali dopolnjenega vzorca.

Ob vseh prednostih uporabe vzorcev se moramo zavedati, da vsaka ideja v razvojnem procesu pomeni napredek, a postane splošno uporabna šele, ko tekom uporabe v različnih kontekstih preverimo njeno učinkovitost. Kljub delitvi posameznih aktivnosti obstaja močno prepletanje med uporabo in oblikovanjem vzorcev, saj se aktivnosti izvajajo vzporedno in so največkrat odvisne ena od druge.

Oblikovanje vzorcev

Oblikovanje vzorcev vsebuje skupek aktivnosti za detekcijo, zapis, verifikacijo in hranjenje idej, najdenih v času razvoja. Povedali smo že, da vzorce lahko najdemo kjerkoli v razvojnem procesu in pojavlja se vprašanja, kako vzorce najti. Sama ideja vzorca, vsebovanje strokovnjakovih izkušenj, daje delni odgovor na zastavljeno vprašanje. Vzorce praviloma najde strokovnjak problemskega področja, ki problemsko področje pozna in ima izkušnje z razvojem informacijskih sistemov. Vsak strokovnjak iz svojih izkušenj kuje rezultate svojega dela. In mi bi radi te izkušnje predstavili v obliki, da bi jih lahko uporabili tudi manj izkušeni razvijalci. Izkušnje kažejo, da je praviloma posameznik tisti, ki vzorce najde v razvojnem procesu, je pa zelo učinkovito periodično izvajati delavnice, kjer celotna razvojna skupina pregleda in po potrebi dopolni najdene vzorce.

Glede na definicijo vzorca se pojavlja vprašanje časa iskanja in zapisovanja novih vzorcev s knjižnico dosegljivih vzorcev. Zavedati se moramo, da mora vzorec predstavljati idejo rešitve, ki je vodila (vodi) k učinkoviti rešitvi problema. Torej je mogoče vzorce praviloma najti šele po rešitvi problema (kar pomeni po fazi implementacije in testiranja sistema za veliko večino vzorcev). Prav kreativne delavnice pa predstavljajo način, da se vzorec uvrsti v knjižnico že pred to verifikacijo.

Slika 3 shematično prikazuje kaskadni razvojni proces in faze razvoja, v katerih lahko pričakujemo nove vzorce. Pomembno je, da poskušamo vzorce vpeljati zgodaj v razvojnem projektu in s tem olajšati razvojni proces. Na sliki so prikazane samo projektne aktivnosti razvojnega projekta, seveda pa lahko vzorce uporabimo tudi v predprojektnih (konceptualizacija, študija izvedljivosti) in poprojektnih aktivnostih.

Slika 3: **Oblikovanje vzorcev**

V fazi analize spoznavamo problemsko področje. V tej fazi najdemo vzorce, ki ponujajo ideje o modeliranju problemskega področja. Tipični vzorci so strukturni vzorci (primer organizacija – stranka, uslužbenec – transakcija, načrt – korak načrta...). Prednosti uporabe vzorcev v fazi analize je pokazal tudi pilotski projekt [2], kjer smo primerjali klasičen objekten pristop s pristopom z uporabo vzorcev. Rezultati projekta so bili popolnoma v skladu s pričakovanji in so prikazali prednosti uporabe vzorcev v fazi analize problemskega področja. Primerjava modelov problemskega področja je namreč pokazala boljšo strukturo in večjo stopnjo prilagodljivosti, pri tem pa je bilo porabljenih za oblikovanje modela s pomočjo uporabe vzorcev manj časa in sredstev.

Faza sistemskega načrtovanja poskrbi za gradnjo močne, a fleksibilne arhitekture celotnega sistema. V kontekstu komponentne tehnologije se osredotočamo na večnivojske arhitekture poslovnih objektov, na povezovanja posameznih objektov v celote (podsisteme) in izbiramo način upravljanja s trajnimi podatki. Na tem področju najdemo celo vrsto vzorcev, ki zagotavljajo dobro arhitekturo sistema (strukturni in

arhitekturni vzorci), vzorce z idejami združevanja objektov v podsisteme in oblikovanja več nivojev poslovnih objektov ter vzorce z idejnimi rešitvami upravljanja s trajnimi podatki. Verjetno najbolj znan vzorec je vzorec Podatek-Pogled-Nadzor (*Model-View-Controler*), ki predstavlja osnovno upravljanje z večimi pogledi na enake podatke, pri tem pa zagotavlja ažurnost podatkov (vzorec najdemo v programski opremi, ki omogoča hkratno delo z večimi dokumenti in različnimi pogledi nad dokumenti, v sistemih za upravljanje s podatki...).

V fazi načrtovanja objektov na podlagi modela problemskega področja in skladno z definiranimi arhitekturnimi omejitvami oblikujemo končni model informacijskega sistema. Skupine objektov oblikujemo v enovite celote, ki omogočajo določeno funkcionalnost ter razrešimo nekatere težave (neskladnost vmesnikov objektov, način implementacije posameznih operacij, zagotavljanje konsistentnosti objektov).

Faza implementacije je faza, kjer je potrebno rezultate prejšnjih faz pretvoriti v programsko kodo. Če smo vzorce uporabili v prejšnjih fazah razvoja, lahko sedaj na podlagi primerov uporabe in opisov vzorcev določimo specifičnost posameznih programskih jezikov. Pri tem gre za enako implementacijo, ki pa je odvisna od specifičnosti programskega jezika.

Zadnja faza razvoja je testiranje. V tej fazi pride do izraza uporaba vzorcev v prejšnjih fazah, natančen opis pristopov olajša testiranje in podaja pristope k testiranju. Vzorce najdemo tako pri funkcionalnem testiranju (pri zapisu vzorca, ki smo ga uporabili je tudi opis načina testiranja implementacije tega vzorca, mogoče je pristop k testiranju opisati kot posamezen vzorec) kot tudi pri integracijskem testiranju (strukturni vzorci omogočajo hitro gradnjo modelov, vsebujejo pa tudi pristope za verifikacijo).

Zavedati se moramo, da prikazane faze predstavljajo le institucionalizacijo razvojnega procesa, brez aktivnosti upravljanja projekta programskega inženirstva. Ne prikazujejo faz konceptualizacije, študije izvedljivosti projekta in vseh poprojektnih aktivnosti. A tudi v teh fazah se pojavlja množica vzorcev, množica pristopov k reševanju problemov, ki pa imajo namesto razvoja informacijskega sistema namen učinkovito implementirati razvojni proces in njegove aktivnosti.

Upravljanje vzorcev

Upravljanje vzorcev je skupek aktivnosti, ki zagotavljajo stalno kakovost vzorcev in omogočajo njihovo učinkovito izkoriščanje. K tem aktivnostim štejemo aktivnosti hranjenja vzorcev v repozitoriju, ki je skupen celotni razvojni organizaciji (projektni organizaciji). Prva faza uporabljanja vzorcev je njihova identifikacija v repozitoriju. To pomeni, da mora upravljanje vzorcev zagotavljati učinkovito iskanje vzorcev v repozitoriju. Identificirane vzorce moramo ovrednotiti, da izberemo res najprimernejši vzorec, ki ga nato uporabimo. To zahteva, da so merila vrednotenja shranjena skupaj z vzorci, da so vzdrževana in po potrebi obnovljena. Tekom razvoja uporabljamo vzorce in vsako uporabo vzorcev moramo zabeležiti tudi v repozitoriju, saj pomeni verifikacijo primernosti vzorca in daje dodatne informacije o vzorcu (odkriti problemi, novi pristopi...). Vzorec, katerega najdemo v času razvoja, je potrebo opisati, klasificirati in shraniti v repozitorij. Pri tem mora upravljanje vzorcev vsebovati verifikacijske postopke, ki po potrebi zagotavljajo pravilnost opisa vzorca. Komercialnih orodij za upravljanje vzorcev na trgu še ni, zato bodo v nadaljevanju opisane osnovne funkcionalnosti in ideje takega orodja. Funkcionalnost orodja za podporo uporabe vzorcev lahko razdelimo na vsaj tri velika področja. Prvo področje je dokumentiranje vzorcev. Dokumentiranje mora uporabniku omogočati opisovanje vzorcev, pri tem pa mora vršiti preverjanje opisa. Takega preverjanja ne bi bilo težko implementirati, če ne bi bila struktura vzorcev precej individualna in zahtevala različne predstavitve.

Različne predstavitve postavljajo tudi visoke zahteve za omogočanje uporabe vzorcev. Identifikacija in vrednotenje vzorcev v različnih predstavitvah je praviloma zapletena in časovno potratna. A objektna tehnologija ponuja s svojimi pristopi k modeliranju problemskih področij možnosti tudi na tem področju.

Nadzor uporabe vzorcev predstavlja funkcionalnost, ki je po svoji naravi močno povezana z upravljanjem konfiguracije. Modul je namenjen predvsem sledenju uporabe vzorcev, dopolnjevanju predstavitve vzorcev s primeri uporabe in s tem dodatnim pojasnilom in dopušča uporabljanje funkcionalnosti obstoječih orodij za upravljanje konfiguracije.

Seveda pa je razvoj kompleksnih informacijskih sistemov v veliki meri pogojen z uporabo orodja, ki spremlja celoten življenjski cikel. Gre za orodja CASE (*Computer Aided Software Engineering*), ki zagotavljajo sledljivost razvoja in višjo produktivnost razvojne skupine. Torej bi bilo ekonomsko upravičeno nadgraditi taka orodja tudi s podporo vzorcem. Pomemben prispevek uporabi vzorcev je tudi vključitev vzorca kot osnovnega koncepta razrednega diagrama metodologije UML (*Unified Modeling Language*).

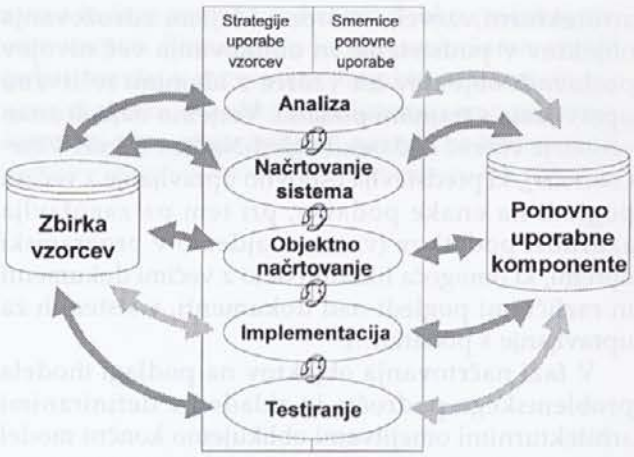
Raziskave kažejo, da zaposleni s srednjo in dolgo delovno dobo v podjetju pogosto migrirajo in s seboj odnašajo mnogo izkušenj. Tako podjetje izgublja informacije o idejah rešitev, ki so v preteklosti vodile

do uspehov, pa tudi neuspehov. Te izkušnje se pridobivajo skozi daljše časovno obdobje, hkrati pa to pomeni, da mladi člani z razmeroma malo izkušnjami le stežka v krajšem roku nadomestijo ljudi, ki so odšli. V takih situacijah se vzorci izkažejo kot odlično sredstvo hranjenja informacij, prenosa znanja in posredovanja izkušenj med zaposlenimi [7, 8], pod pogojem, da so dokumentirani na dovolj formalen način in dostopni vsej razvojni skupini. Izkušnje kažejo, da uporaba vzorcev v razvojnem procesu močno olajša komunikacijo v razvojni skupini, saj poenoti besednjak in s tem zmanjšuje dvoumnost naravnega jezika.

Uporaba vzorcev

Kot prikazuje slika 4, so temelj za uporabo vzorcev tako smernice ponovne uporabe kot tudi strategije, ki zagotavljajo potrebno znanje. Strategije pri uporabi vzorcev praviloma delimo na funkcionalne strategije (določevanje zahtevanih funkcionalnosti sistema), strategije organizacije sistema (delitev kompleksnega sistema na podsisteme), strategije identifikacije objektov in modeliranja dinamike (zagotavljajo poznavanje problemskega področja in omogočajo uporabo strukturnih arhitekturnih in ustvarjalnih vzorcev) [1]. Osnovni namen strategij je kakovostno zajemanje znanja o problemskem področju, z namenom lažjega identificiranja vzorcev. Prav identifikacija vzorcev je najpomembnejši del razvoja z uporabo vzorcev. Poglejmo si nekaj korakov za iskanje primernega vzorca, ki zagotavljajo učinkovito identifikacijo in izbor vzorcev določenega problemskega področja:

- *Premislimo, kako vzorec rešuje problem* na podlagi spoznanja, da vzorci pomagajo identificirati in specificirati vmesnik ter vlogo objektov.
- *Pregledamo povzetke vzorcev*, da ugotovimo kateri vzorci vsaj malo ustrezajo našemu problemu. Nato se posvetimo tem vzorcem še natančneje.



Slika 4: **Razvoj z uporabo vzorcev**

- *Preučimo relacije med vzorci*, saj opisi vzorcev ali slike povezovanja vzorcev grafično prikazujejo relacije med vzorci. Preučevanje odvisnosti pomaga pri izbiri ustreznega vzorca ali skupine ustreznih vzorcev.
- *Preučimo, kaj naj bo spremenljivo v našem razvoju*, s tem da razmislimo, kaj naj bi bilo na voljo za spremembo, ne da bi bilo potrebno spreminjati celoten razvoj. Osredotočimo se na ograjevanje spreminjajočih konceptov.

Tabela 3 prikazuje koncepte nekaterih vzorcev načrtovanja, ki jih lahko spreminjamo, ne da bi bilo potrebno ponovno izvesti načrtovanje. To omogoča večjo fleksibilnost razvojnega procesa in dodatno zmanjšuje rizik razvoja.

Ko določimo vzorec, ki je primeren za reševanje zadanega problema, ga moramo pravilno uporabiti. Sledijo navodila kako učinkovito uporabiti vzorec izbrani vzorec:

- *Natančno spoznamo vzorec*, še posebej se osredotočimo na uporabnost in vplive vzorca, da se prepričamo, ali vzorec ustreza našemu problemu.

Namen	*Vzorec*	*Vidik, ki se lahko spremeni*
Ustvarjalni	*Graditelj*	Kreiranje sestavljenih objektov
	Tovarniška metoda	Podrazredi objektov, ki so kreirani (oblikovani)
	Prototip	Razredi objektov, ki so kreirani (oblikovani)
Strukturni	*Prilagoditev*	Vmesnik objekta
	Most	Implementacija objekta
	Dekorator	Kompozicija; struktura in kompozicija objektov
		Odgovornost objekta brez tvorbe podrazredov
Vedenjski	Namestnik	Način dostopa do objekta; njegova lokacija
	Interpreter	Slovnica in interpretacija jezika
	Iterator	Dostop ali iskanje po elementih
	Posredovalec	Kako in kateri objekti sodelujejo med seboj
	Spomin	Katere informacije so shranjene izven objekta
	Opazovalec	Število objektov, ki je odvisno od drugega objekta; način zagotavljanja ažurnosti objektov

Tabela 3: Spremenljivi koncepti v definicijah vzorcev

- *Še enkrat preučimo strukturo, udeležence in sodelovanje vzorcev.* Prepričamo se, da razumemo razrede in objekte v vzorcu, in odvisnosti med njimi.
- *Izberemo imena za udeležence v vzorcu, ki so smiselna glede na kontekst,* saj so imena udeležencev v vzorcih ponavadi preveč abstraktna, da bi jih neposredno uporabili v aplikaciji.
- *Definiramo razrede.* Deklariramo vmesnike, relacije dedovanja in definiramo primerke spremenljivk. Identificiramo obstoječe razrede v naši aplikaciji na katere vpliva vzorec, in jih po potrebi modificiramo.
- *Definiramo ustrezna imena operacij v vzorcu.* Prvi del imena operacije naj vsebuje ime vzorca, drugi del imena operacije pa naj opisuje funkcijo, ki jo operacija izvaja.
- *Implementiramo aktivnosti, ki skrbijo za zanesljivost in sodelovanje znotraj vzorca.* Sekcija implementacije v vzorcu ponuja različne namige pri implementaciji. Tudi sekcija s primeri lahko pripomore pri implementaciji.

Zaključek

Vzorce uporabljamo že vse svoje življenje, le da se tega ne zavedamo. Tudi pri razvoju programske opreme hočemo uporabo vzorcev formalizirati s tem, da vzorce konsistentno opišemo in v razvojnem procesu definiramo aktivnosti, ki so potrebne za izkoriščanje obstoječega znanja. Vzorci so tehnika, ki ob pravilni uporabi pripomore k uspešnejšemu razvoju informacijskih sistemov. V prispevku so opisani osnovni koncepti predstavitve vzorcev in vplivi uporabe vzorcev na razvojni proces.

Ugotovili smo, da uporaba vzorcev uvaja nove aktivnosti v proces razvoja, hkrati pa ponuja nove možnosti ponovne uporabe. Smer razvoja področja vzorcev danes ponuja možnosti integracije vzorcev in ogrodij, združevanje vzorcev v jezike vzorcev za določena področja (paralelno programiranje, nadzorni sistemi, porazdeljeni sistemi) in vključitev vzorcev v metodologije razvoja in podporna orodja.

Pojavlja pa se vprašanje, kje dobiti zbirke vzorcev, če nimamo dovolj velike množice lastnih vzorcev. V svetu obstaja veliko število knjižnic vzorcev, ki pa so praviloma nedostopne širšemu krogu razvijalcev iz ekonomskih razlogov in varovanja poslovnih skrivnosti. Zato bo v prihodnosti potrebno najprej spremeniti mnenje o uporabi rešitev nekoga drugega (kompleks lastnega razvoja) in usmeriti vse moči v poenotenje predstavitve vzorcev. Prav orodja za upravljanje z vzorci ponujajo razmeroma preprosto rešitev, saj s seboj nehote prinašajo poenotenje predstavitve.

Literatura

[1] COAD Peter, Object Models - Strategies, Patterns, &Applications, Yourdon Press, 1995

[2] DOMAJNKO Tomaž, KOREN Silvo, CELCER Borut, DRVARIČ Ivan, Uporaba objektnega pristopa na primeru izgradnje informacijske podpore poslovne funkcije v zavarovalništvu, zbornik OTS'97 Objektna tehnologija v Sloveniji, str. 120-130,

[3] FOWLER Martin, Analisys Patterns - Reusable Object Models, Addison-Wesley, 1996

[4] GAMMA Erich, Design Patterns - Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994

[5] JACOBSON Ivar, Object-Oriented Software Engineering - A Use Case Driven Approach, Addison-Wesley Publishing company, 1992

[6] JACOBSON Ivar, The Object Advantage - Bussines Process Reeingineering with Object Technology, Addison Wesley Publishing Company, 1994

[7] PRINS Robert, Developing Bussines Objects - A framework Driven Approach, McGraw-Hill, 1996

[8] Rumbaugh J. et al., Object-Oriented Modeling and Design, Prentice Hall, 1991

[9] Vatovec – Krmac Evelin, Vloga ponovne uporabe pri razvoju programske opreme, Uporabna informatika, št. 2/1997, str. 28-3s5,

[10] WILKIE George, Object-Oriented Software Engineering - The professional Developer's Guide, Addison-Wesley, 1993

[11] http://www.cs.wustl.edu/~schmidt/

◆

Tomaž Domajnko je diplomiral leta 1996 na FERI Maribor, kjer nadaljuje študij po enovitem doktorskem programu kot mladi raziskovalec. Področje njegovih raziskav je objektna tehnologija, s poudarkom na pristopih k razvoju modernih programskih sistemov.

◆

Mag. Marjan Heričko *je asistent na FERI Maribor. Njegovo raziskovalno delo obsega vse vidike objektne tehnologije, s poudarkom na zagotavljanju kakovosti objektnega razvoja programskih sistemov.*

◆

Dr. Ivan Rozman *je diplomiral na Fakulteti za elektrotehniko v Ljubljani, magistriral in doktoriral pa na Tehniški fakulteti Univerze v Mariboru. Je redni profesor Univerze v Mariboru in ustanovitelj Laboratorija za informacijske sisteme, ki ga vodi še danes. Je avtor številnih publikacij in vodja večih raziskovalnih projektov.*

◆

Dr. József Györkös *je diplomiral, magistriral in doktoriral na Tehniški fakulteti Univerze v Mariboru. Je izredni profesor Univerze v Mariboru in avtor številnih publikacij objavljenih tako doma kot po svetu. Področja njegovih raziskav so informacijska družba, sistemi za podporo odločanju in upravljanje procesov razvoja informacijskih sistemov.*