Pregled in analiza programskih ogrodij in sorodnih tehnologij

Gregor Polančič, Boštjan Šumak
Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Inštitut za informatiko
{gregor.polancic, bostjan.sumak}@uni-mb.si

Izvleček

V prispevku so predstavljena programska ogrodja, ki danes predstavljajo eno izmed ključnih tehnologij razvoja programske opreme in storitev. Predstavljeni so zgodovina in osnovne značilnosti programskih ogrodij ter položaj, ki ga ta zasedajo med tehnikami ponovne uporabe. Sledi pregled poglavitnih prednosti in slabosti programskih ogrodij in njihovo mesto v procesu razvoja programske opreme, ki je ponazorjeno v obliki procesa. Širokemu spektru uporabe programskih ogrodij je namenjen razdelek, ki navaja vrste in klasifikacije ogrodij. Ker se ogrodja pogosto zamenjujejo s sorodnimi tehnologijami, so v predzadnjem razdelku predstavljene podobnosti in razlike med koncepti in tehnologijami, ki so sorodne oz. komplementarne ogrodjem.

Abstract

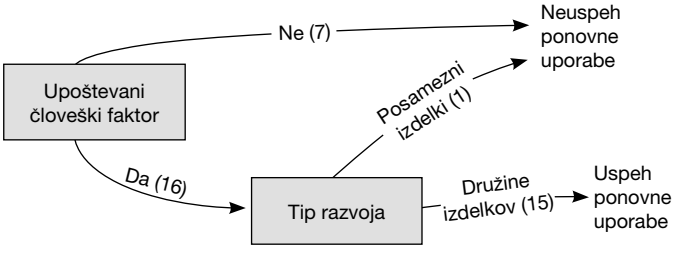
AN INTRODUCTION TO SOFTWARE FRAMEWORKS

This article presents a high-level introduction to software frameworks, which nowadays represent a focal technology for software and service development. The history, main characteristics and position of software frameworks in software reuse techniques are presented. Afterwards, the main benefits and drawbacks of using frameworks in the process of software development are explained. The wide scope of software frameworks usage is illustrated by means of defining basic frameworks types and their classifications. In order to avoid the frameworks being mistaken with other software reuse techniques, a complete chapter is assigned to the explanation of similarities and differences with similar or complementary software reuse techniques.

1 UVOD

Člani projektov razvoja programske opreme se ubadajo z vedno večjimi konkurenčnimi pritiski, ki vladajo na trgu programske opreme. Ustrezen odziv na vse večjo in kakovostno ponudbo zahteva hiter razvoj novih programskih proizvodov oz. nadgradenj obstoječih proizvodov, širitev ponudbe, zagotavljanje skladnosti s standardi in visoko stopnjo povezljivosti z drugimi proizvodi. Ponovna uporaba je učinkovito sredstvo za doseganje omenjenih ciljev.

Ponovna uporaba v programskem inženirstvu je definirana kot primer dejanj, v katerih se enak programski izdelek uporabi v različnih kontekstih. Izmed številnih vrst ponovne uporabe (Leach 1996; Sindre, Conradi & Karlsson 1995) spadajo produktne linije (angl. product line software) med najučinkovitejše in predstavljajo enega izmed kritičnih faktorjev uspeha ponovne uporabe (slika 1).



Slika 1: **Poglavitni faktorji, ki vplivajo na uspešnost ponovne uporabe (Morisio, Ezran & Tully 2002)**

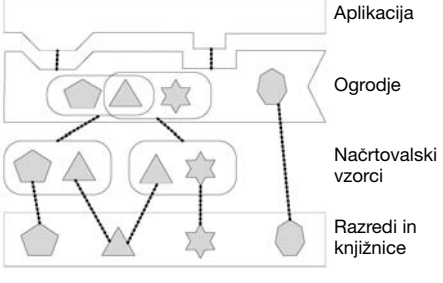
Ideja produktnih linij temelji na razvoju družine izdelkov, ki so zgrajeni na enotni programski osnovi (angl. core asset base). Pokazalo se je, da takšen pristop zagotavlja dolgoročne ekonomske prednosti glede na razvoj posameznih izdelkov, kar je razvidno iz ekonomskega modela produktnih linij (Bockle et al. 2004).

Produktne linije se najpogosteje implementirajo z razvojem na podlagi programskih ogrodij. Ta omogočijo produktnim linijam enotno podlago s tem, da zagotavljajo generične rešitve za množico podobnih problemov v domeni produktne linije.

V nadaljevanju prispevka so podrobneje predstavljena programska ogrodja, njihova zgodovina, položaj v programskem inženirstvu in v procesu razvoja programske opreme. Predstavljene so vrste, prednosti in slabosti ogrodij. V tretjem razdelku so predstavljene tehnologije in koncepti, ki so podobni programskim ogrodjem oz. se dopolnjujejo z njimi. V zadnjem razdelku so podane sklepne misli.

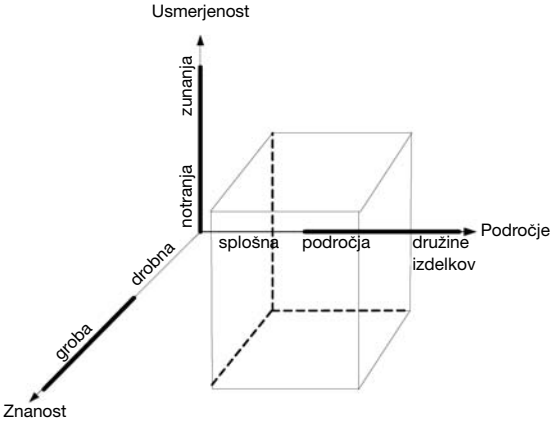
2 PROGRAMSKA OGRODJA

Programska ogrodja (v nadaljevanju ogrodja) so nepopolni sistemi, ki vsebujejo gradnike, enotne vsem aplikacijam v produktni liniji, in gradnike, ki jih je mogoče prilagajati in predstavljajo edinstvene dele posameznih aplikacij v produktni liniji (Srinivasan 1999). Ogrodja se razlikujejo od preostalih vrst ponovne uporabe v programskem inženirstvu (programske komponente, knjižnice, načrtovalski vzorci), saj težijo k ponovni uporabi večjih delov programske kode in zasnove na višji ravni (slika 2) (Morisio, Romano & Stamelos 2002). V nasprotju z drugimi tehnikami ponovne uporabe programske kode definirajo ogrodja tok izvajanja in zato delujejo kot podlaga na njih temelječih aplikacij. Ogrodja spadajo med učinkovitejše tehnike ponovne uporabe, saj poleg ponovne uporabe programske kode enkapsulirajo še znanje načrtovanja (Oliveira et al. 2004).



Slika 2: **Ogrodja in njihova povezava z drugimi tehnikami ponovne uporabe (Sangdon et al. 1999)**

Ogrodja predstavljajo tehniko ponovne uporabe, zato je njihov glavni cilj dvig produktivnosti v programskem inženirstvu (Mattsson 1996). Z vidika obsega ponovne uporabe se programska ogrodja uvrščajo med tehnike področne ponovne uporabe (angl. domain reuse) oz. ponovne uporabe, ki je namenjena družinam programskih izdelkov (angl. product families). Z vidika zrnatosti se programska ogrodja nahajajo na visokem (grobem) nivoju zrnatosti (glej območje kvadra na sliki 3).



Slika 3: **Umestitev ogrodij v REBOOT-klasifikacijo ponovne uporabe**

Najpogosteje uporabljena definicija ogrodja je (Johnson & Foote 1988): Ogrodje je množica razredov, ki vključujejo abstrakten načrt rešitve za družino povezanih problemov.[1]

Večina sodobnih ogrodij je objektno orientiranih (angl. Object-Oriented Framework – OOF) (Krajnc 2006). Definicija objektno orientiranega ogrodja je povzeta po Gamma et al. (Gamma et al. 1995) in se glasi: Ogrodje je množica sodelujočih razredov, ki sestavljajo ponovno uporaben načrt za specifično vrsto programske opreme. Ogrodje določa arhitekturne smernice z razdelitvijo načrta v abstraktne razrede in z definiranjem njihovih odgovornosti in sodelovanj. Razvijalec prilagaja ogrodje za posamezno aplikacijo s povezovanjem primerkov razredov ogrodja.

Obstaja še več definicij ogrodij, ki jih je analiziral Mattsson (1996) in na njihovi podlagi oblikoval lastno, generično definicijo ogrodja: Ogrodje predstavlja generično arhitekturo, ki je zasnovana z namenom

[1] A framework is a set of classes that embodies an abstract design for solutions to a family of related problems.

zviševanja ponovne uporabnosti. Ogrodja vključujejo množico sodelujočih abstraktnih in konkretnih razredov, ki enkapsulirajo obnašanje podedovanih specializacij.[2]

Ogrodja so torej vzorci, ki vsebujejo zamisel načrta rešitve določene problemske domene in množico gradnikov, ki vsak zase izpolnjujejo posamezno vlogo v ogrodju. Poenostavljeno povedano predstavljajo ogrodja programsko opremo, ki jo lahko programer uporabi, prilagodi in razširi z namenom ustrezati zahtevam končne programske rešitve. Ogrodja (objektno orientirana) temeljijo na uveljavljenih vzorcih in izkoriščajo prednosti treh konceptov objektne paradigme: abstraktnih podatkov (razredov), polimorfizma in dedovanja. Takšnim ogrodjem so skupne naslednje karakteristike (Johnson & Foote 1988):

- **razredi odjemalci** (angl. client classes) – Končne programske rešitve se ogrodju prilegajo na t. i. razširitvenih točkah;
- **sodelovanje objektov** (angl. collaboration of objects) – (Abstraktni) razredi ogrodja definirajo model obnašanja (angl. model of interaction). Končne programske rešitve se zato obnašajo po definiranem modelu;
- **zamenjava nadzora** (angl. inversion of control) – Model obnašanja ogrodja določa način vključevanja razredov odjemalcev, kar pomeni, da igra ogrodje vlogo glavnega programa (v nasprotju z vključevanjem programskih knjižnic). Koncept je poznan kot »hollywoodsko načelo« (angl. Hollywood principle).

2.1 Zgodovina ogrodij

Pojem »programska ogrodja« ni nov, saj se je koncept ogrodij pojavil že v osemdesetih letih prejšnjega stoletja, in sicer v okoljih Smalltalk (Adele 1984) in Apple Inc. (Kurt 1986). Prvo široko uporabljeno ogrodje je bil uporabniški vmesnik Smalltalk-80, znan pod imenom model-pogled-nadzornik (angl. Model-View-Controller) ali krajše MVC.

V devetdesetih letih prejšnjega stoletja so se ogrodja iz domene uporabniških vmesnikov razširila na preostale programske domene. Med pomembnejša ogrodja, razvita v devetdesetih, spadajo CommonPoint,[3] HotDraw,[4] ACE,[5] JAWS,[6] CORBA (angl. Common Object Request Broker Architecture) in MFC (angl. Microsoft Foundation Classes). Ogrodje MVC je bilo ob ogrodju OWL (angl. Object Windows Library) kar nekaj časa dejansko industrijski standard za razvoj grafičnih aplikacij za osebne računalnike.

K razširitvi in uveljavitvi ogrodij v devetdesetih letih je pripomogel tudi programski jezik java. Večina ogrodij za javo nastaja znotraj delovnih skupin v procesu JCP (angl. Java Community Process), ki ga upravlja podjetje Sun. Med ogrodja, ki so nastala v okviru JCP, spadajo EJB (angl. Enterprise JavaBeans), RMI (angl. Remote Method Invocation), AWT (angl. Abstract Window Toolkit), Swing, JFC (Java Foundation Classes), JSP (JavaServer Pages), JSF (angl. JavaServer Faces), Collection Framework, JMF (angl. Java Media Framework) in JAF (angl. JavaBeans Activation Framework). Veliko ogrodij na podlagi jave nastaja tudi v odprtokodnih projektih. Primeri takšnih ogrodij so Struts, Spring, Hybernate, JUnit, Avalon in JCorporate Expresso (Krajnc & Heričko 2004).

Med najpomembnejša in najpogosteje uporabljana ogrodja zadnje generacije spadajo Microsoft.NET, Spring, Jakarta Struts, Django, Hybernate, Ruby on Rails in Eclipse framework.

Poleg omenjenih ogrodij, ki so javno dostopna, številna podjetja razvijajo še lastna ogrodja (angl. in-house framework). Takšna ogrodja se uporabljajo samo interno in niso namenjena za prodajo ali javno uporabo. V zadnjem času je mogoče zaslediti tudi uporabo ogrodij za razvoj programskih orodij (Krajnc et al. 2005). Primer takšnega orodja je Eclipse IDE.

2.2 Prednosti in slabosti ogrodij

Programerji in vodje projektov se za razvoj na podlagi ogrodij odločajo predvsem zaradi: (1) minimiziranja obsega implementacije, ki je potrebna za razvoj aplikacij, in (2) lažjega obvladovanja znanja domene, v kateri organizacija razvija aplikacije (Mattsson 1996). Prednosti ogrodij so torej:

- **Hitrejši in učinkovitejši razvoj.** Z uporabo ogrodja aplikacije nikoli ne gradimo od začetka, temveč ponovno uporabimo programsko kodo oz. storitve, ki jih zagotavlja ogrodje. Ker aplikaci-

[2] A (generative) architecture designed for maximum reuse, represented as a collective set of abstract and concrete classes; encapsulated potential behavior for sub-classed specializations.
[3] Množica ogrodij za hitrejši razvoj aplikacij.
[4] Ogrodje za izgradnjo grafičnih urejevalnikov, napisano v jeziku Smalltalk.
[5] ADAPTIVE Communication Environment – objektno orientirano ogrodje, namenjeno za komunikacijsko programsko opremo.
[6] Adaptive Web Server – spletni strežnik in ogrodje za izgradnjo drugih vrst strežnikov.

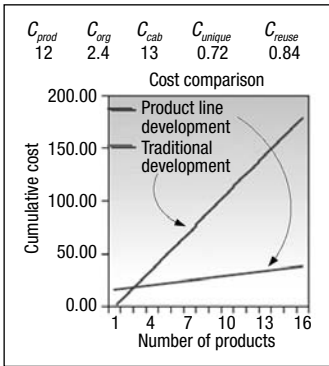
je, ki temeljijo na enakem ogrodju, gradimo po ustaljenem vzorcu, se učinkovitost razvoja še dodatno poveča.

- **Boljša kakovost programske opreme.** Izvorna koda ogrodij običajno temelji na preizkušenih programskih vzorcih in je zaradi večkratne uporabe izpostavljena obsežnim testiranjem.
- **Ogrodja omogočajo ponovno uporabo izvorne kode in načrtovanja.**
- **Omogočajo preusmeritev fokusa s področja sistemskih problemov na področje domene.** Sistemski problemi so rešeni v ogrodjih, zato se razvijalcem aplikacij z njimi ni treba ubadati.
- **Zagotavljajo visoko stopnjo medizvedljivosti** (angl. interoperability)**.** Aplikacije, ki temeljijo na enakem ogrodju, so si glede arhitekture sorodne. Skupaj s podporo uveljavljenim standardom imajo na ogrodju temelječe aplikacije zagotovljeno visoko stopnjo medsebojne izvedljivosti.

Avtorja (Fayad & Schmidt 1997) navajata, da izhajajo prednosti uporabe ogrodij iz naslednjih lastnosti ogrodij:

- **Ponovna uporabnost** (angl. reusability). Stopnja ponovne uporabe ogrodja je običajno višja kot pri preostalih tehnikah ponovne uporabe.[7] Ogrodja omogočajo ponovno uporabo na nivoju programske kode, vzorcev in opisa konceptov, potrebnih za reševanje določenega problema. Z opisovanjem konceptov definirajo ogrodja slovar za problemsko področje. Razvijalec, ki uporablja ogrodje, vidi problemsko področje prav skozi slovar ogrodja. S tem zagotavljajo ogrodja še ponovno uporabo konceptov analize (Roberts & Johnson 1997).
- **Modularnost** (angl. modularity). Ogrodja zvišujejo modularnost programske opreme z ločevanjem vmesnikov od implementacije. Zaradi večje modularnosti je identifikacija napak in sprememb v takšni programski opremi lažja. S tem se: (1) zmanjša napor za razumevanje in vzdrževanje programske opreme in (2) zvišuje kakovost programske opreme.
- **Razširljivost** (angl. extensibility). Ogrodja povečujejo razširljivost programske opreme z zagotavljanjem standardnih razširitvenih točk (angl. hook methods). Razširitvene točke zagotavljajo stabilnost vmesnikov ogrodij z njihovimi najpogostejšimi implementacijami. Zaradi razširitvenih točk so ogrodja lahko hkrati stabilna in razširljiva.

Če zgoraj navedene prednosti programskih ogrodij strnemo in finančno ovrednotimo, ugotovimo, da se prednosti uporabe ogrodij povečujejo s številom ponovnih uporab ogrodij, kar ponazarja slika 4.



Number of products	Product line development	Traditional development
0	15.40	0.00
1	16.96	12.00
2	18.52	24.00
3	20.08	36.00
4	21.64	48.00
5	23.20	60.00
6	24.76	72.00
7	26.32	84.00
8	27.88	96.00
9	29.44	108.00
10	31.00	120.00
11	32.56	132.00
12	34.12	144.00
13	35.68	156.00
14	37.24	168.00
15	38.80	180.00

Slika 4: **Skupni stroški »običajnega razvoja« in razvoja na podlagi produktnih linij (Bockle, Clements, McGregor, Muthig, & Schmid 2004)**

Iz slike 4 je mogoče razbrati, da je za razvoj z uporabo ogrodij (krivulja Product line development) značilen začetni ekonomski deficit, ki nastane zaradi spremembe procesov razvoja (C_{org}) in razvoja lastnega ogrodja oz. spoznavanja obstoječega (C_{cab}). Prednosti ogrodij se nato večajo (relativno glede na »tradicionalni razvoj«) s številom ponovnih uporab ogrodja (»number of products«).

Potencialne slabosti uporabe ogrodij so (Mattsson 1996; Roberts & Johnson 1997):

- **Zapleten razvoj ogrodij.** Razvoj kakovostnega ogrodja je težaven in običajno zahteva bogate izkušnje v arhitekturni zasnovi aplikacij in problemski domeni.
- **Težavno dokumentiranje ogrodij.** Zaradi kompleksnosti je ogrodja težko dokumentirati. Če ogrodja niso ustrezno dokumentirana, jih razvijalci aplikacij ne uporabljajo.
- **Težavno zagotavljanje povezljivosti.** Ogrodja se nenehno razvijajo in spreminjajo, zato je težavno zagotavljati kompatibilnost s predhodniki in njihovimi primerki.
- **Zmanjšana učinkovitost aplikacij.** Splošnost in prožnost ogrodij lahko predstavljata omejitve za razvijalce aplikacij in učinkovitost razvitih aplikacij.
- **Težavno razhroščevanje.** Razhroščevanje ogrodij in primerkov ogrodij je težavno, saj pogosto ni

[7] Ogrodja predstavljajo tudi do 80 odstotkov kode končnega izdelka (Seddon, Staples, Patnayakuni & Bowtell 1999).

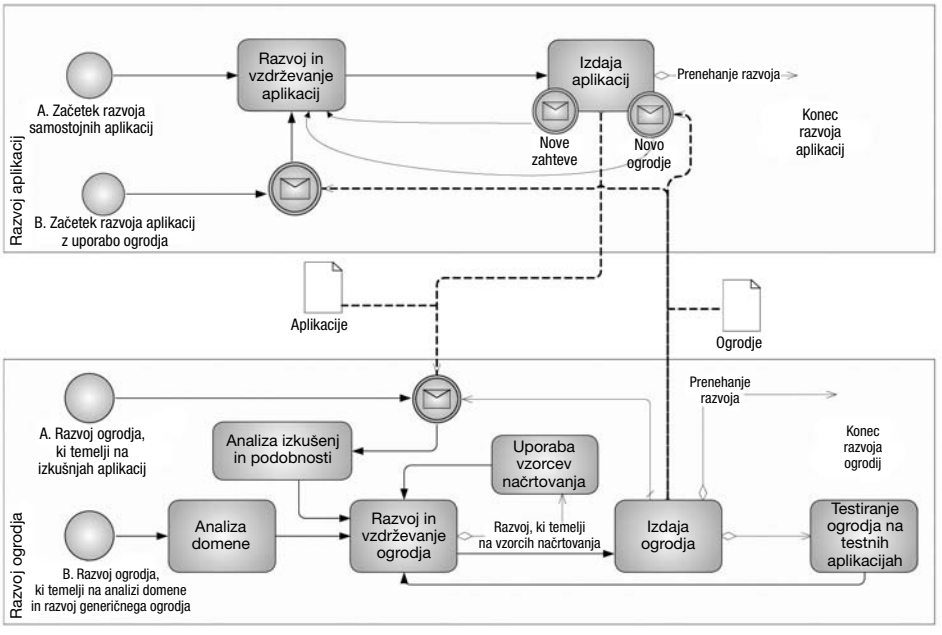
mogoče lokalizirati napak (»Ali se napaka pojavlja v ogrodju ali v aplikaciji?«). Prav tako so lahko napake, ki se pojavijo v ogrodju, neodpravljive za uporabnike ogrodij.[8]

- **Pomanjkanje standardov.** Na področju ogrodij standardi ne obstajajo ali so šele v povojih, kar zmanjšuje njihovo zamenljivost (angl. replaceability). V zadnjem času se podjetja lotevajo omenjene težave z delovnimi skupinami, v katere so vključena različna podjetja in odprtokodne skupnosti.
- **Odvisnost od programskega jezika.** Ker so ogrodja napisana v določenem programskem jeziku, so vezana nanj. Namestitev ogrodja v okolje, ki temelji na drugačnem programskem jeziku, zato ni mogoče.

2.3 Ogrodja v procesu razvoja programske opreme

Proces razvoja programske opreme na osnovi ogrodij predstavlja poseben primer procesa razvoja na podlagi ponovno uporabne programske opreme. Procesov oz. modelov razvoja programske opreme na osnovi ogrodij je več (Mattsson 1996). Na sliki 5 so v obliki modela BPMN[9] predstavljeni štirje procesi razvoja ogrodij in na ogrodju temelječih aplikacij:

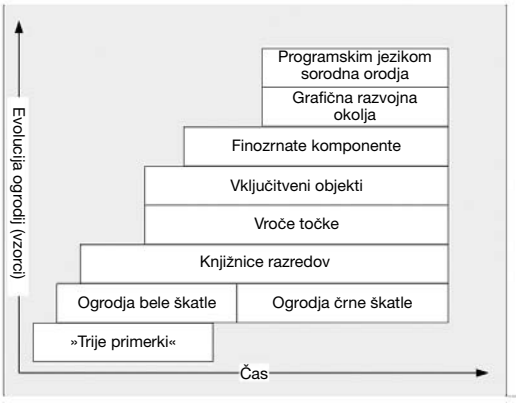
- **Proces razvoja ogrodja, ki temelji na izkušnjah razvoja aplikacij.** Takšen proces se začne z razvojem (družin) aplikacij (glej začetna dogodka A). Na podlagi podobnosti med aplikacijami in izkušenj razvijalcev se lahko razvijalci odločijo skupne funkcionalnosti prenesti v ogrodje. Po izdaji ogrodja se nato vse samostojne aplikacije preoblikujejo v aplikacije, ki temeljijo na ogrodju. Izkušnje pri razvoju takšnih aplikacij se nato ponovno prenesejo v razvoj ogrodja in proces se ponovi.
- **Proces razvoja ogrodja, ki temelji na analizi domene.** Začetek takšnega procesa predstavlja analiziranje abstrakcij v domeni, ki se nato vključijo v razvoj ogrodja (glej začetna dogodka B). Na osnovi ogrodja se nato začnejo razvijati končne aplikacije. Izkušnje z razvojem končnih aplikacij in odzivi uporabnikov se nato upoštevajo pri vzdrževanju ogrodja.
- **Proces razvoja ogrodja, ki temelji na uporabi vzorcev načrtovanja.** Proces se začne z razvojem samostojne aplikacije (glej začetna dogodka A). Na podlagi analiziranja aplikacij se nato z upoštevanjem vzorcev načrtovanja začne razvoj ogrodja. V nadaljevanju se ogrodje posodablja na podlagi izkušenj pri razvoju aplikacij in njihovih uporabnikov.



Slika 5: **Različni procesi razvoja ogrodja, prikazani v modelu BPMN**

[8] V primeru lastniških, zaprtokodnih ogrodij.
[9] BPMN je akronim za Business Process Modelling Notation.

- **Proces razvoja generičnega ogrodja.** Razvoj generičnega ogrodja se začne z analiziranjem domene (glej začetna dogodka B). Na podlagi analize domene se začne razvoj generičnega ogrodja. Testiranje ogrodja se izvede s testnimi aplikacijami, na podlagi katerih se nato izboljšuje ogrodje. Takšen proces je lahko popolnoma neodvisen od razvoja aplikacij.



Slika 6: **Evolucija ogrodij, temelječa na vzorcih zasnove (Roberts & Johnson 1997)[9]**

Vzporedno s procesom razvoja in vzdrževanja so ogrodja izpostavljena tudi evoluciji oz. »zorenju« ogrodja. Roberts in Johnson (1997) sta na podlagi vzorcev zasnove ogrodij opredelila stopnje zrelosti ogrodij:

- **Ogrodja bele škatle** (angl. white-box framework). Instanciranje takšnega ogrodja temelji na modifikacijah izvorne kode ogrodja in na dedovanju razredov ogrodja.
- **Knjižnice komponent** (angl. component library). Razredi, ki so skupni aplikacijam v domeni ogrodja, so v ogrodje vključeni v obliki knjižnic.
- **Vroče točke** (angl. hot spots). V takšnih orodjih je koda, ki se pogosto spreminja, ločena od kode, ki se ne spreminja (angl. frozen spots). Zaradi lažjega obvladovanja je koda, ki se spreminja, združena v razredih, ki se najpogosteje razširjajo s kompozicijo.
- **Vključitveni objekti** (angl. pluggable objects). Namesto trivialnih podrazredov vsebuje takšno ogrodje podrazrede, ki jih je mogoče parameterizirati.
- **Drobnozrnate komponente** (angl. fine-grained components). S ciljem povečanja stopnje ponovne uporabnosti so razredi in knjižnice v takšnih ogrodjih drobnozrnati.
- **Ogrodja črne škatle** (angl. black-box frameworks). V takšnih ogrodjih so knjižnice strukturirane na osnovi dedovanja, medtem ko se za njihovo povezovanje uporablja kompozicija.
- **Grafična razvojna okolja** (angl. visual building tools). Grafična razvojna okolja so v pomoč razvijalcem aplikacij pri specifikaciji in povezovanju objektov v primerkih ogrodja.
- **Programskim jezikom sorodna orodja** (angl. language tools). Takšnim ogrodjem so dodana orodja za njihov nadzor izvajanja in pomoč pri razhroščevanju.

2.4 Vrste in klasifikacije ogrodij

Ogrodja se po svoji zasnovi, obsežnosti in namenu zelo razlikujejo. Čeprav obstajajo najrazličnejše klasifikacije ogrodij, ogrodja najpogosteje delimo na (Johnson & Foote 1988):

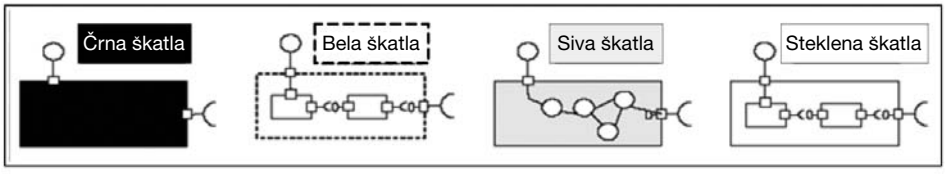
- domenska ogrodja (angl. domain framework) – naslavljajo določene problemske domene (npr. zavarovalništvo, računovodstvo in upravljanje človeških virov),
- orodna ogrodja (angl. utility framework) – naslavljajo določene programske domene (npr. trajnost podatkov, uporabniški vmesnik in testiranje kod),
- aplikacijska ogrodja (angl. application framework) – obsežna ogrodja, ki so uporabna za različne problemske domene in naslavljajo številne programske domene.

V zadnjem času je vse več poskusov uveljavljanja t. i. organizacijskih ogrodij (angl. enterprise frameworks), ki zaokrožujejo posamezno problemsko domeno poslovanja. Če jih primerjamo z drugimi ogrodji, so organizacijska ogrodja po obsegu večja in bolj kompleksna; v njih so lahko vsebovane najrazličnejše komponente in druga ogrodja. Organizacijska ogrodja vključujejo infrastrukturni, domenski in arhitekturni vidik (Fayad & Hamu 2000).

Poleg predstavljene delitve se ogrodja pogosto delijo glede na tip razvoja, in sicer na: (1) odprtokodna (angl. open source frameworks), (2) lastniška (angl. proprietary frameworks) in (3) ogrodja, ki so razvita za lastne potrebe (angl. in-house frame-

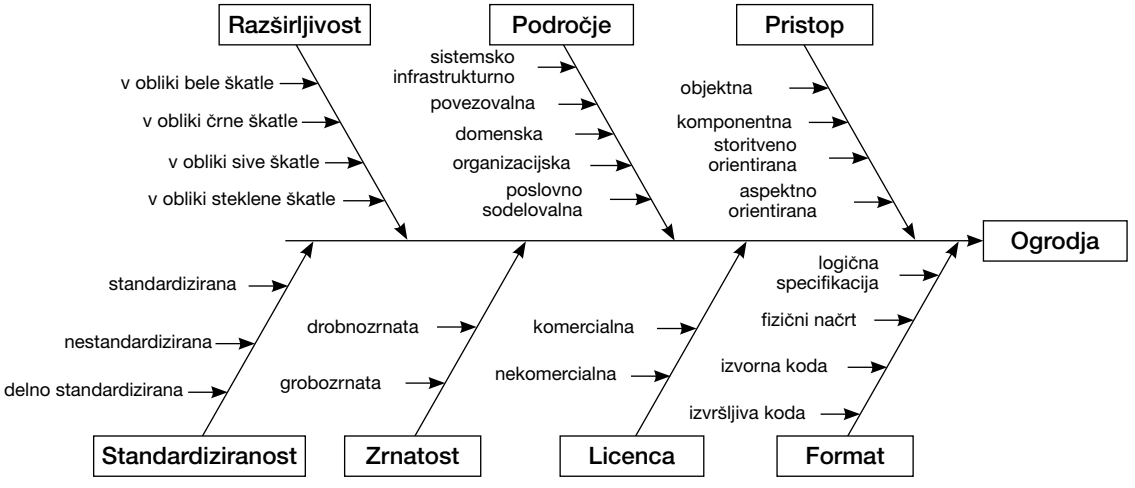
[9] Prvo fazo v vzorcih evolucije ogrodij predstavljajo trije primerki razvoja aplikacij v domeni, v kateri naj bi se razvilo ogrodje. Takšen pristop razvoja ogrodij je skladen s procesom razvoja ogrodja na izkušnjah razvoja aplikacij (slika 4).

works). Glede na tehniko uporabe lahko ogrodja razvrstimo na ogrodja bele škatle in ogrodja črne škatle. Za ogrodja bele škatle je značilno, da se za doseganje razširljivosti močno opirajo na lastnosti objektivno orientiranih jezikov, kot sta dedovanje ali povezovanje v času izvajanja (angl. dynamic binding). Ogrodja črne škatle podpirajo razširljivost skozi definicijo vmesnikov za komponente, ki jih lahko nato vključimo v ogrodje z uporabo kompozicije objektov. Obstajajo še ogrodja sive škatle in steklene škatle, ki predstavljajo vmesne rešitve (slika 6).



Slika 7: **Vrste ogrodij glede na razširljivost**

Na spodnji sliki 7 je prikazan model celovite klasifikacije ogrodij, ki temelji na deskriptivni študiji predhodno izvedenih klasifikacij ogrodij (Krajnc & Heričko 2003).

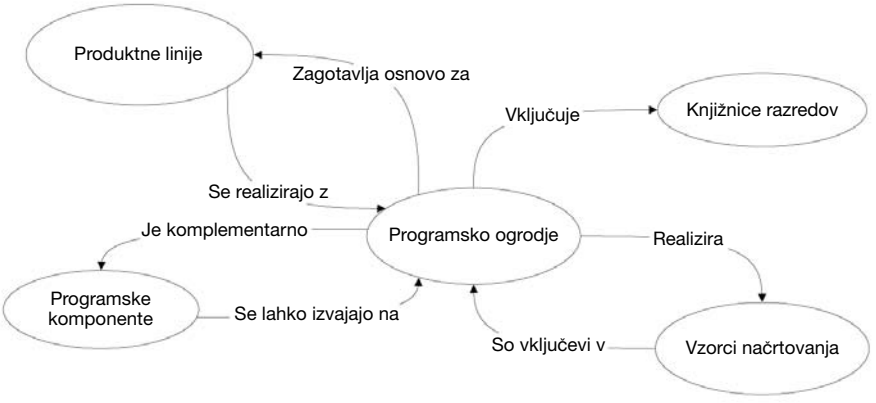


Slika 8: **Celovita klasifikacija ogrodij (Krajnc & Heričko 2004)**

Zgornji model klasifikacije ogrodij deli ogrodja glede na način instanciranja, področje uporabe, pristop razvoja ogrodij, standardiziranost ogrodij, zrnatost ogrodij, licenčni vidik ogrodij in format zapisa ogrodij. Značilnost modela celovite klasifikacije ogrodij je, da so posamezne kategorije odvisne med seboj.

3 OGRODJEM SORODNI KONCEPTI IN TEHNOLOGIJE

Ker so se ogrodja razvila iz drugih, objektno orientiranih konceptov in tehnik ponovne uporabe, so ostala tesno povezana z njimi (slika 8).



Slika 9: **Povezava ogrodij s sorodnimi koncepti in tehnologijami**

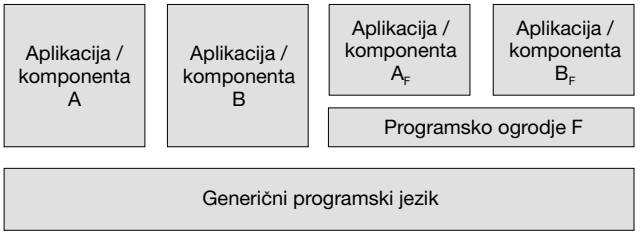
V nadaljevanju so navedene podobnosti in razlike med ogrodji in sorodnimi tehnologijami oz. koncepti.

3.1 Programske komponente

Programska komponenta (v nadaljevanju komponenta) je del sistema, ki zagotavlja določeno storitev ali dogodek in je sposobna komunicirati z drugimi komponentami. Komponenta mora zadostiti temle kriterijem (Messerschmitt & Szyperski 2003):

- večkratna uporaba,
- neodvisnost od konteksta,
- zmožnost sodelovanja z drugimi komponentami,
- skrite podrobnosti in dostopanje prek vmesnikov,
- neodvisnost od namestitve in verzioniranja.

Komponente zagotavljajo ponovno uporabo na ravni implementacije (angl. code reuse), medtem ko zagotavljajo ogrodja še ponovno uporabo načrtovanja (angl. design reuse) in konceptov analize.

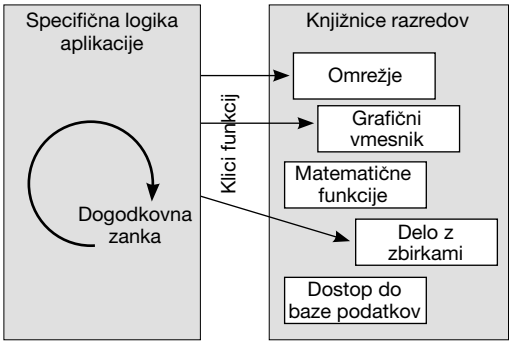


Slika 10: **Povezava med komponentami in ogrodjem**

Ogrodja in komponente sta v več pogledih komplementarni tehnologiji. Ogrodja pogosto zagotavljajo ponovno uporaben kontekst za komponente z zagotavljanjem storitev, kot so izmenjava podatkov, obvladovanje izjem, beleženje dnevnikov ipd. Druga možnost sodelovanja je, da lahko ogrodja zagotovijo podlago za izdelovanje družine komponent (slika 9) (Roberts & Johnson 1997).

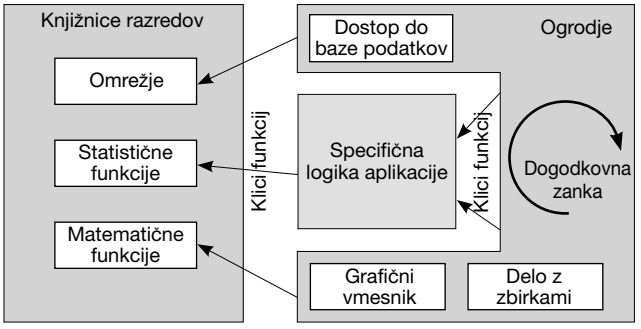
3.2 Knjižnice razredov

Z vidika ogrodij ponujajo knjižnice razredov majhno zrnatost (angl. granularity) in nižjo abstrakcijo ponovne uporabe. Kot prikazuje slika 10, so knjižnice razredov nizkonivojske, relativno neodvisne in splošne komponente, kot npr. pripomočki za matematično in statistično obdelavo, delo z zbirkami, razredi za delo z omrežji, dostop do podatkovne baze. Podobno kot komponente zagotavljajo knjižnice razredov le ponovno uporabo na nivoju implementacije.



Slika 11: **Uporaba knjižnice razredov**

V nasprotju s knjižnicami razredov ogrodja definirajo delne rešitve, ki vključujejo domensko specifične objektne strukture in funkcionalnosti. Obseg ponovne uporabe ogrodij je zato večji kot pri uporabi knjižnic razredov (Morisio, Romano, & Stamelos 2002; Moser & Nierstrasz 1996).



Slika 12: **Razlika v uporabi ogrodja in knjižnic razredov**

Razlika med ogrodji in knjižnicami razredov je tudi v kontroli izvajanja aplikacij. Knjižnice razredov so »pasivne«, kar pomeni, da nimajo glavne kontrole nad tokovi izvajanja aplikacije, temveč izvajanje programa kontrolira kodo aplikacije (slika 10). Ogrodja so »aktivna«, kar pomeni, da se pri izvajanju aplikacije kontrola izvajanja prenese na ogrodje, ki potem po potrebi kliče programsko kodo aplikacij. Takšna arhitektura temelji na povratnih klicih (slika 11) in se imenuje obrat kontrole (angl. inversion of control) ali »hollywoodsko načelo«.[10]

Ogrodja in knjižice razredov sta lahko komplementarni tehnologiji. Ogrodja pri izvajanju programske kode uporabljajo knjižnice razredov. Uporaba je lahko interna (kot del ogrodja) ali eksterna (prek povratnih, aplikacijsko specifičnih klicev) (Krajnc 2006).

3.3 Vzorci načrtovanja

V programskem inženirstvu so vzorci načrtovanja definirani kot splošne ponovno uporabne rešitve za pogoste probleme, ki se pojavljajo v fazi načrtovanja programske opreme.[11] Objektno orientirani vzorci načrtovanja najpogosteje prikazujejo povezave in sodelovanje med splošnimi razredi in objekti. Vzorci načrtovanja se od algoritmov razlikujejo po tem, da rešujejo načrtovalske probleme in ne računskih problemov.

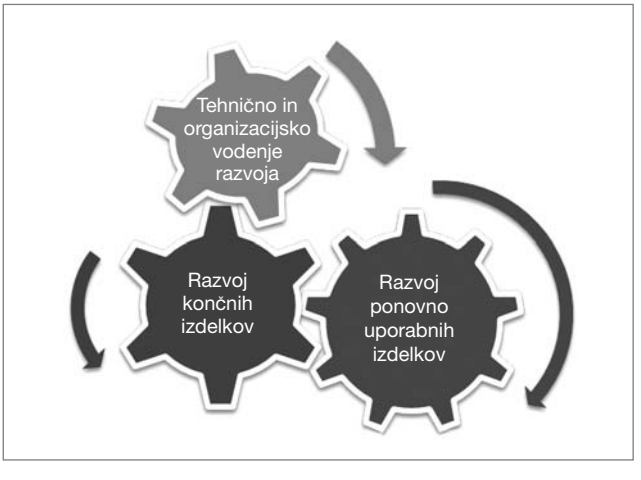
Ogrodja se od vzorcev načrtovanja razlikujejo predvsem po tem, da zagotavljajo še ponovno uporabo na ravni implementacije (angl. code reuse). Vzorcev načrtovanja ne moremo izraziti kot razrede v objektnih programskih jezikih in jih ponovno uporabiti z uporabo dedovanja oz. kompozicije. Vzorci so torej bolj abstraktni kot ogrodja (Krajnc 2006).

Ogrodja in vzorci sta komplementarni tehniki. V ogrodju je pogosto uporabljenih več vzorcev (glej tudi sliko 2), npr. v ogrodju MVC zasledimo uporabo treh načrtovalskih vzorcev: vzorec Opazovalec (angl. Obsever), vzorec Kompozicija (angl. Composite) in vzorec Strategija (angl. Strategy). Na vzorce načrtovanja lahko gledamo kot na mikroarhitekturne elemente ogrodij, saj predstavljajo rešitev, ogrodja pa konkretno implementacijo (Johnson 1997).

3.4 Produktne linije

Produktno linijo programske opreme Bosch (2000) opredeljuje kot »množico sistemov programske opreme, ki si delijo skupno upravljano množico funkcionalnosti, ki zadovoljujejo specifične potrebe določenega tržnega segmenta ali poslanstva in so razvite iz skupne množice osnovnih pridobitev na predpisan način«, Northrop (1999) pa kot »množico produktov, ki si delijo skupno arhitekturo programske opreme in množico ponovno uporabnih komponent«.

Razvoj na podlagi produktnih linij vključuje voden in nadzorovan razvoj množice ponovno uporabnih izdelkov (angl. core asset development) in razvoj množice končnih izdelkov (angl. product development) (slika 12). Pri tem ni določen vrstni red razvoja izdelkov.[12] Produktne linije naj bi zagotavljale dolgoročne ekonomske prednosti glede na razvoj posameznih izdelkov oz. glede na nesistematično ponovno uporabo (Amar & Coffey 2005; Bockle, Clements, McGregor, Muthig & Schmid 2004).



Slika 13: **Osnovne aktivnosti produktnih linij (Northrop 1999)**

[10] »Ne kličite nas, mi bomo poklicali vas.« *(Hollywood principle)*

[11] Povzeto po: http://en.wikipedia.org/wiki/Design_pattern_(computer_science).

[12] Ponovno uporabni izdelki lahko nastajajo na podlagi končnih izdelkov in obratno.

Produktne linije in ogrodja so povezani med sabo, saj se skupna osnova produktnih linij (ponovno uporabni izdelki) najpogosteje realizira z uporabo ogrodja (Batory, Cardone & Smaragdakis 2000; Morisio, Romano, & Stamelos 2002; Philippow & Riebisch 2001). Pojem produktnih linij je torej širši od pojma ogrodij.

3.5 Drugi sorodni koncepti in tehnologije

Ogrodja se pogosto primerjajo še z domensko specifičnimi jeziki in generatorji aplikacij. Domensko specifični jeziki (angl. domain-speciffic language) ali krajše DSL so programski jeziki, ki so namenjeni za uporabo v določeni domeni. DSL, kot je npr. skriptni jezik lupine Unix, predstavljajo nasprotje generičnim programskim jezikom, kot sta npr. C++ in Java. Ogrodja in DSL imajo podoben cilj – dosegati boljše rezultate pri razvoju programske opreme v določeni domeni. Kljub temu se ogrodja in DSL z vidika zasnove in tudi njunih lastnosti močno razlikujejo (Deursen 1997).

Ogrodjem so sorodni tudi generatorji aplikacij (angl. application generator). Generatorji aplikacij so programska oprema, ki generira aplikacije (delno ali v celoti) na podlagi opisa problema. Generatorji aplikacij običajno temeljijo na visokonivojskem domensko specifičnem jeziku (Roberts & Johnson 1997) medtem ko temeljijo ogrodja na generičnih programskih jezikih.

4 SKLEP

Ogrodja imajo danes osrednje mesto v programskem inženirstvu (Manolescu, Noble & Voelter 2006) predvsem na področju razvoja produktnih linij in družin programske opreme (Batory, Cardone & Smaragdakis 2000; Cunningham, Liu & Zhang 2006). Ogrodja združujejo znanje določene programske domene, zato predstavljajo specializacijo generičnih programskih jezikov za določeno področje. Razvijalcem programske opreme zagotavljajo koristi na številnih področjih razvoja programske opreme, in sicer: 1) na področju skupnih lastnosti v domeni njihovega delovanja (domenska ogrodja), 2) na področju tehnik, ki jih vključujejo v razvoj programske opreme (orodna ogrodja), in 3) na področju vrste aplikacij, ki jih izdelujejo (aplikacijska ogrodja). Posledično je njihova uporaba močno zaželena (Sparks, Benner & Faris 1996). Število ogrodij se je v zadnjih letih zelo povečalo, kar nakazuje tudi podatek, da je na spletnem repozitoriju odprtokodnih projektov »Sourceforge.net« več kot 4000 projektov umeščenih med ogrodja.[13] Povzeto po Fontouri (1999) »obstajajo projekcije, da bodo ogrodja postala jedro tehnologij programskega inženirstva prihodnosti«.

5 LITERATURA

[1] Adele, G. 1984, SMALLTALK-80: the interactive programming environment Addison-Wesley Longman Publishing Co. Inc.

[2] Amar, L. & Coffey, J. 2005, »Measuring the benefits of software reuse - Examining three different approaches to software reuse«, DR DOBBS JOURNAL, vol. 30, no. 6, pp. 73–76.

[3] Bockle, G., Clements, P., McGregor, J. D., Muthig, D. & Schmid, K. 2004, »Calculating ROI for software product lines«, Ieee Software, vol. 21, no. 3, p. 23–+.

[4] Bosch, J. 2000, Design and use of software architectures: adopting and evolving a product-line approach ACM Press/ Addison-Wesley Publishing Co.

[5] Deursen, A. V. 1997, »Domain - specific languages versus object-oriented frameworks: A financial engineering case study«, in STJA' 97, Ilmenau Technical University, pp. 35–39.

[6] Fayad, M. & Hamu, D. 2000, »Enterprise frameworks: guidelines for selection«, ACM Computing Survey, vol. 32, no. 1, p. 4.

[7] Fayad, M. E. & Schmidt, D. C. 1997, »Object-oriented application frameworks«, Communications of the Acm, vol. 40, no. 10, pp. 32–38.

[8] Fontoura, M. F. 1999, »Object-Oriented Application Frameworks: the Untold Story«, in Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'99).

[9] Gamma, E., Helm, R., Johnson, R. & Vlissides, J. 1995, Design Patterns: Elements of Reusable Object-Oriented Software, 1 edn, Addison-Wesley Professional.

[10] Grimm, L. G. & Yarnold, P. R. 2004, Reading and understanding multivariate statistics, 9th printing edn, Washington (D.C.) : American Psychological Association.

[11] IEEE. IEEE standard glossary of software engineering terminology. 10-12-1990. 1990.

[12] Johnson, R. E. & Foote, B. 1988, »Designing Reusable Classes«, Journal of Object-Oriented Programming, vol. 1, no. 2, p. 22–&.

[13] Krajnc, A. & Heričko, M. »Classification of Object-Oriented Frameworks«, in EUROCON 2003.

[14] Krajnc, A. & Heričko, M. 2004, »Vloga ogrodij pri razvoju sodobnih informacijskih rešitev«, Uporabna informatika, vol. 12, no. 2, pp. 68–79.

[15] Kurt, J. S. 1986, »Object-oriented programming for the Macintosh«, Hayden Books.

[16] Lajoie, R. & Keller, R. K. 1994, »Design and Reuse in Object--Oriented Frameworks: Patterns, Contracts, and Motifs in Concert«, in ACFAS Montreal Canada.

[17] Leach, R. J. 1996, Software Reuse: Methods, Models and Costs McGraw-Hill, Inc.

[18] Lewis, J. A. 1990, »An experiment to determine software reusability factors (abstract)«, ACM, p. 405.

[19] Manolescu, D., Noble, J. & Voelter, M. 2006, »Patterns for Successful Object-oriented Framework Development,« in Pattern Languages of Program Design 5, 1st edn, Addison Wesley Professional, pp. 401–431.

[13] Podatek je po Sourceforge.net povzet dne 13. 3. 2008.

[20] Mattsson, M. 1996, Object-oriented Frameworks, A survey of methodological issues, Lund University - Department of Computer Science.
[21] Messerschmitt, D. G. & Szyperski, C. 2003, Software Ecosystem: Understanding an Indispensable Technology and Industry MIT Press.
[22] Mili, H. 1995, »Reusing Software - Issues and Research Directions«, Ieee Transactions on Software Engineering, vol. 21, no. 6, pp. 528–562.
[23] Morisio, M., Romano, D. & Stamelos, I. 2002, »Quality, productivity, and learning in framework-based development: An exploratory case study«, Ieee Transactions on Software Engineering, vol. 28, no. 9, pp. 876–888.
[24] Moser, S. & Nierstrasz, O. 1996, »The effect of object-oriented frameworks on developer productivity«, Computer, vol. 29, no. 9, p. 45–&.
[25] Northrop, L. M. 1999, »A Framework for Software Product Line Practice«, Springer-Verlag, pp. 365–366.
[26] Oliveira, T. C., Alencar, P. S. C., Filho, I. M., de Lucena, C. J. P., & Cowan, D. D. 2004, »Software process representation and analysis for framework instantiation«, Ieee Transactions on Software Engineering, vol. 30, no. 3, pp. 145–159.
[27] Philippow, I. & Riebisch, M. 2001, »Systematic Definition of Reusable Architectures«, in Eight Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems ECBS '01.
[28] Roberts, D. & Johnson, R. 1997, »Patterns for evolving frameworks,« in Pattern languages of program design 3, Addison-Wesley Longman Publishing Co. Inc., pp. 471–486.
[29] Sindre, G., Conradi, R. & Karlsson, E. A. 1995, »The Reboot Approach to Software Reuse«, Journal of Systems and Software, vol. 30, no. 3, pp. 201–212.
[30] Sparks, S., Benner, K. & Faris, C. 1996, »Managing object-oriented framework reuse«, Computer, vol. 29, no. 9, p. 52–&.
[31] Srinivasan, S. 1999, »Design patterns in object-oriented frameworks«, Computer, vol. 32, no. 2, p. 24–+.
[32] van Gurp, J. & Bosch, J. 2001, »Design, implementation and evolution of object oriented frameworks: concepts and guidelines«, Software-Practice & Experience, vol. 31, no. 3, pp. 277–300.

▪

Gregor Polančič je docent na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Doktoriral je iz področja sprejetosti programskih ogrodij. Med njegova interesna področja spadajo še tehnologije komuniciranja in sodelovanja, modeliranje informacijskih procesov in sociološko-tehnični vidiki informacijskih tehnologij in storitev.

▪

Boštjan Šumak je asistent na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Med njegova interesna področja spadajo analiza, načrtovanje in razvoj sodobnih informacijskih rešitev in e-storitev ter zagotavljanje kakovosti pri razvoju e-storitev v različnih domenah (e-poslovanje, e-učenje, e-zdravje, e-uprava itn.). Kot član Inštituta za informatiko je aktivno sodeloval v več raziskovalnih in razvojnih projektih.