UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

MITJA GOLOUH

UPORABA AGILNIH PRISTOPOV PRI RAZVOJU PODATKOVNIH BAZ

MAGISTRSKO DELO

Ljubljana, 2009

Univerza
v Ljubljani

Fakulteta *za računalništvo*
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št.: 74-MAG-ISO/2006
Datum: 18. 9. 2006

Mitja G O L O U H, univ. dipl. inž. rač. in inf.

L j u b l j a n a

Fakulteta za računalništvo in informatiko Univerze v Ljubljani izdaja naslednjo magistrsko nalogo

Naslov naloge: **UPORABA AGILNIH PRISTOPOV PRI RAZVOJU PODATKOVNIH BAZ**

Tematika naloge:

Naloga naj obsega opis uporabe agilnih pristopov pri razvoju podatkovnih baz, s poudarkom na opisu področij, kjer obstoječe informacijske rešitve ne nudijo ustrezne podpore agilnim razvojnim pristopom. Naloga naj razdela uporabo sistemov za vodenje konfiguracij programske opreme, implementacijo zasebnih razvojnih okolij in usklajevanje sprememb med njimi. Obravnava naj posebnosti pri postavitvi in nadgradnji podatkovne sheme ter problematiko usklajevanja postavitve odjemalske aplikacije in podatkovne baze.

Mentor:

doc. dr. Marko Bajec

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

za Dekan:

prof.dr. Boštjan Vilfan

Zahvala

Zahvaljujem se svojemu mentorju doc. dr. Marku Bajcu za strokovne nasvete in pomoč pri izdelavi naloge. Mag. Janu Dujcu in kolegom s podjetja Genis, kjer je nastala ideja za izdelavo orodja SQLMake. Sodelavcem iz podjetja Hermes Softlab, kjer sem pridobil mnogo praktičnih izkušenj pri vodenju konfiguracije.
Tomažu Bregarju za lektoriranje.
Staršem, ki so me spodbujali ves čas študija.

Prav posebna zahvala gre moji ženi ter hčerki Pini in sinu Štefanu, ki so mi ves čas stali ob strani in z razumevanjem sprejemali mojo odsotnost.

Kazalo vsebine

Povzetek 1
Abstract 2
1 Uvod 3
1.1 Motivacija 3
1.2 Opredelitev problema 4
1.3 Opredelitev cilja in rezultata 5
1.4 Metode dela 5
1.5 Temeljni viri 6
1.6 Implementacijske opombe 6
1.7 Zgradba nadaljnjega besedila 7
2 Agilni pristopi pri razvoju podatkovnih baz 8
2.1 Agilne metodologije 8
2.2 Agilnost in podatkovne baze 9
2.2.1 Uporaba sistemov za vodenje konfiguracije 9
2.2.2 Zasebno razvojno okolje 12
2.2.3 Preoblikovanje podatkovne sheme 16
2.2.4 Regresijsko testiranje 17
2.2.5 Evolucijsko podatkovno modeliranje 18
2.3 Sorodne raziskave 20
2.3.1 Evolucija podatkovne sheme 20
2.3.2 Verzioniranje podatkovne sheme in časovne podatkovne baze 22
2.3.3 Lahkotna evolucija podatkovne sheme 23
3 Agilni pristopi v praksi 24
3.1 Primer 1: Centralni register prebivalstva 24
3.1.1 Razvojno okolje 24
3.1.2 Namestitev podatkovne sheme 24
3.1.3 Nadgradnja podatkovne sheme 25
3.1.4 Faza vzdrževanja 25
3.2 Primer 2: Matični register in novi register stalnega prebivalstva 27
3.2.1 Razvojno okolje 27
3.2.2 Sql skripte 28
3.2.3 Namestitev podatkovne sheme 31

3.2.4 Nadgradnja podatkovne sheme 33
3.2.5 Statistika razvoja podatkovne sheme 35
3.3 Primer 3: Unified Colection Gateway 36
3.3.1 Razvojno okolje 36
3.3.2 Sql skripte 36
3.3.3 Namestitev podatkovne sheme 39
3.3.4 Nadgradnja podatkovne sheme 39
3.3.5 Statistika razvoja podatkovne sheme 40
3.4 Povzetek 41
4 Namestitev podatkovne sheme 43
4.1 Definicije 43
4.2 Življenjski krog podatkovne sheme 44
4.2.1 Enostaven življenjski krog 44
4.2.2 Kompleksen življenjski krog 46
4.3 Začetna namestitev podatkovne sheme 48
4.3.1 Namestitev SUPB 48
4.3.2 Priprava okolja v SUPB 48
4.3.3 Kreiranje lastnika podatkovne sheme 49
4.3.4 Namestitev struktur podatkovne sheme 49
4.3.5 Ponovitev začetne namestitve 51
4.4 Nadgradnja podatkovnega modela 51
4.4.1 Oznaka gradnje 51
4.4.2 Časovno označevanje 52
4.4.3 Enolična globalna oznaka 52
4.4.4 Zaporedna številka 53
4.5 Nadgradnja shranjenih procedur 53
4.6 Podporne aktivnosti pri nameščanju 55
4.6.1 Verzija podatkovne sheme 55
4.6.2 Napake pri izvajanju skript 56
4.7 Povzetek 56
5 SQLMake 58
5.1 Uvod 58
5.2 Sorodna orodja 58
5.2.1 Dbdeploy 59

5.2.2 UpToDater 60
5.2.3 MIGRATEdb 60
5.2.4 Liquibase 61
5.3 Razlike glede na sorodna orodja 62
5.4 Glavne funkcionalnosti 63
5.4.1 Namestitev struktur podatkovne sheme 63
5.4.2 Delo s shranjenimi procedurami 64
5.4.3 Nadgradnja podatkovne sheme 66
5.4.4 Nastavitve delovanja 67
5.4.5 Vodenje verzije sheme 68
5.4.6 Status sheme in delo z napakami 69
5.5 Vpliv na način dela 71
5.5.1 Organizacija datotečnega sistema 71
5.5.2 Izdelava razvojnega okolja 71
5.5.3 Spremembe podatkovnega modela 71
5.5.4 Razvoj shranjenih procedur 72
5.5.5 Shranjevanje sprememb v sistem za vodenje konfiguracije 72
5.5.6 Osveževanje zasebnega razvojnega okolja 73
5.6 Hitrost delovanja 73
5.7 Povzetek 75
6 Sklepne ugotovitve 77
6.1 Povzetek ugotovitev naloge 77
6.2 Možnosti nadaljnjega dela 78
7 Viri 79
8 Ostali viri 81

Kazalo slik

Slika 1: Primer orodja za primerjanje in združevanje razlik med modeli 11
Slika 2: Okolja in prenos sprememb med njimi 12
Slika 3: Shema idealnega razvojnega okolja 13
Slika 4: Souporaba iste podatkovne sheme 14
Slika 5: Prenos sprememb iz sistema za vodenje konfiguracije v zasebno razvojno okolje 15
Slika 6: SUPB kot centralno vozlišče 16
Slika 7: Življenjski cikel modeliranja z AMDD 19
Slika 8: Postopek izdelave sql skript pri MRnRSP 29
Slika 9: Vmesnik orodja SQL Build 32
Slika 10: Postopek izdelave sql skript pri UCG 38
Slika 11: Enostaven življenjski krog podatkovne sheme 45
Slika 12: Kompleksen življenjski krog podatkovne sheme 46

Kazalo primerov

Primer 1: Evolucija relacije zaposleni 21
Primer 2: Izbira verzije sheme v SQL/Temporal 23
Primer 3: Struktura direktorijev pri MRnRSP projektu 29
Primer 4: Zapis SQL ukazov za nadgradnjo podatkovne sheme pri MRnRSP 34
Primer 5: Struktura direktorijev pri UCG projektu 37
Primer 6: Poimenovanje skript za nadgradnjo pri UCG projektu 39
Primer 7: Primer sql skripte 44
Primer 8: Rezultat izvajanja sql skripte 44
Primer 9: Vpliv vrstnega reda izvajanja evolucije podatkovne sheme na končno strukturo tabele 45
Primer 10: SQL stavek za izdelavo tabele z opredeljenimi fizičnimi lastnosti 49
Primer 11: Sql skripta za namestitev struktur podatkovne sheme 50
Primer 12: Primer shranjene procedure v sql skripti 53
Primer 13: Skripta za nadgradnjo shranjenih procedur 54
Primer 14: Poimenovanje skript za nadgradnjo pri orodju Dbdeploy 59
Primer 15: Skripta za nadgradnjo z UNDO sekcijo, ki jo podpira Dbdeploy 59
Primer 16: XML zapis spremembe podatkovne sheme v MIGRATEdb 60
Primer 17: Odsek iz skripte za nadgradnjo podatkovne sheme Bugzille 61

Primer 18: Dodajanje novega stolpca z orodjem Liquibase 62
Primer 19: Statistika vsebine sistemskih namestitvenih sql skript za Oracle XE..... 64
Primer 20: Izpis razlik v shranjenih procedurah med skriptami in podatkovno shemo z SQLMake..... 65
Primer 21: Poimenovanje skript za nadgradnjo..... 66
Primer 22: Vsebina skripte za nadgradnjo..... 66
Primer 23: Nadgradnja podatkovne sheme z SQLMake 67
Primer 24: Konfiguracijska tabela orodja SQLMake 68
Primer 25: Izpis statusa podatkovne sheme z SQLMake 69
Primer 26: Izpis napak pri namestitvi/nadgradnji podatkovne sheme z SQLMake 70

Kazalo tabel

Tabela 1: Evidenca namestitev zahtevkov pri projektu CRP 26
Tabela 2: Končnice datotek sql skript in njihova vsebina 30
Tabela 3: Statistika sprememb razvoja podatkovne sheme MRnRSP v fazi vzdrževanja 35
Tabela 4: Statistika sprememb razvoja podatkovne sheme UCG..... 41
Tabela 5: Statistika SQL ukazov za posamezne primere sistemov 74
Tabela 6: Rezultati nameščanja testnega primer z SQLMake 75

Seznam uporabljenih kratic in simbolov

Kratica	Pomen	Razlaga
MOF	Meta Object Factory	standard za formalno definicijo modelov in metamodelov
MDA	Model Driven Architecture	modelno voden razvoj
SQL	Structured Query Language	strukturiran povpraševalni jezik za delo s podatkovnimi bazami
XML	Extensible Markup Language	format podatkov za izmenjavo strukturiranih dokumentov v spletu
XMI	XML Metadata Interchange	standard za izmenjavo MOF modelov in meta podatkov
SUPB	Sistem za upravljanje podatkovnih baz	sistem za upravljanje podatkovnih baz kot so na primer Oracle Database, MySQL, Microsoft SQL Server, IBM DB2
AMDD	Agile Model Driven Development	agilni razvoja na podlagi modeliranja, izpeljanka iz MDD (Model Driven Development)
MRnRSP	Matični register in novi register stalnega prebivalstva	primer informacijskega sistema
UCG	Unified Collection Gateway	primer informacijskega sistema
CRP	Centralni register prebivalstva	primer informacijskega sistema
EMRIS	Enotna metodologija razvoja informacijskih sistemov	metodologija razvoja informacijskih sistemov, ki obsega opis procesa razvoja ter opis metod in tehnik
DDL	Data Definition Language	SQL ukazi za definiranje in spreminjanje podatkovne sheme
DBA	Database Administrator	administrator podatkovne baze
DML	Data Manipulation Language	SQL ukazi za delo s podatki: select, insert, update, delete
GUID	Global unique identifier	globalno enolični identifikator
PL/SQL	Procedural Language/Structured Query Language	razširitev jezika SQL s programskimi konstrukti, ki se uporablja v Oracle SUPB

Povzetek

Magistrska naloga obravnava uporabo agilnih pristopov pri razvoju podatkovne baze. Podrobno sta predstavljena naslednja dva pristopa: uporaba sistemov za vodenje konfiguracije in zasebna razvojna okolja. Gre za dva temeljna pristopa, ki pogojujeta uspešno uporabo preostalih agilnih pristopov. Njuna uporaba je tesno povezana s podpornima aktivnostima namestitve in nadgradnje podatkovne sheme, ki sta v magistrskem delu podrobno razdelani.

Pri izdelavi naloge je bilo ključnega pomena sodelovanje na številnih praktičnih projektih razvoja informacijskih sistemov, kjer so zabeleženi konkretni problemi in rešitve pri razvoju podatkovnh baz. Najbolj izstopajoče dejstvo je velik delež sprememb shranjenih procedur. V magistrskem delu je predlagan nov pristop nameščanja shranjenih procedur, ki se zgleduje po ideji selektivnega prevajanja v orodju *make*. Druge ugotovitve so: preveč kompleksna vzpostavitev novega okolja podatkovne sheme, velik obseg dela pri vzdrževanju sql skript in pomanjkljiva evidenca napak pri nameščanju oziroma nadgradnji.

Ugotovitve iz prakse podkrepljene s teorijo na področju agilnih metodologij so privedle do izdelave rešitve SQLMake, s katero se postopki namestitve in nadgradnje podatkovne sheme poenostavijo in avtomatizirajo. Na ta način je omogočena preprosta uporaba zasebnih razvojnih okolij v razvoju podatkovne sheme in s tem odprta pot za implementacijo preostalih agilnih pristopov.

Naloga je razdeljena na šest delov. V prvem poglavju so opredeljeni problemi, ki jih obravnava naloga in cilji magistrske naloge. Drugo poglavje vsebuje pregled agilnih pristopov pri razvoju podatkovnih baz in sorodne raziskave s področja sprememb podatkovne sheme. V tretjem poglavju so predstavljeni trije primeri razvoja podatkovne sheme iz prakse. V četrtem poglavju je podrobno razdelan postopek nameščanja in nadgradnje podatkovne sheme. Peto poglavje je namenjeno predstavitvi rešitve SQLMake in njene uporabe. Zadnje, šesto poglavje vsebuje sklepne ugotovitve.

Ključne besede:
sql, sql skripta, podatkovna shema, podatkovna baza, agilen, namestitev, nadgradnja, sistem za vodenje konfiguracije, zasebna razvojna okolja, make

Abstract

The master's thesis presents techniques used in an agile driven database development. Two techniques are discussed in details: configuration management of database artifacts and developer sandboxes. Both of them are necessary for successful implementation of other agile driven database development techniques. Good knowledge of database schema install and upgrade procedures is required when implementing configuration management of database artifacts and developer sandboxes.

Key area of this thesis is a field study on different development projects where database development specifics were recorded. Interesting finding is related to high number of stored procedures changes. In thesis we propose a new approach for stored procedure development using an idea of selective compilation from *make* tool. Other findings are: too complex set up of a new schema for developer's sandbox, keeping scripts up to date is very time consuming and support for managing deployment errors is insufficient.

Findings combined with existing knowledge of agile techniques inspired the creation of a new tool SQLMake which simplifies and automates database schema install and upgrade procedures, sql script maintenance, error management and supports a new way of stored procedure development. All together SQLMake enables easier configuration management of database artifacts and use of developer sandboxes.

Thesis is divided into six chapters. First chapter describes the problem and goals. Second chapter contains an overview of agile database development techniques and related work in the field of database change management. Third chapter presents three case studies of database development. Fourth chapter deals with detailed description of database schema install and upgrade procedure. SQLMake tool and its usage are presented in fifth chapter. Last, sixth chapter contains conclusion and suggestions for future work on SQLMake tool.

Keywords:

sql, sql script, database schema, database, agile, install, upgrade, configuration management, developer sandbox, make

1 Uvod

Razvoj informacijskih sistemov je povezan s številnimi spremembami, zaradi katerih je potrebno dopolnjevati in spreminjati modele, programsko kodo, podatkovno shemo in dokumentacijo. Vsaka sprememba lahko vpliva na že zaključene izdelke, zato se jim poskuša čim bolj izogniti.

Agilne metodologije na spremembe gledajo v drugačni luči in jih sprejemajo kot sestavni del razvojnega procesa. Kakovost končnega izdelka kljub številnim spremembam zagotavljajo z naslednjimi pristopi: vodenje konfiguracije, zasebna razvojna okolja, preoblikovanje (angl. *refactoring*), neprestana gradnja (angl. *continuous build*) in regresijsko testiranje.

V sklopu teh pristopov magistrsko delo predstavlja podroben vpogled v probleme pri vodenju konfiguracije in uporabi zasebnih razvojnih okolij pri razvoju podatkovne sheme ter kot odgovor na težave nudi predlog zasnove razvojnega procesa, ki je podprt s programsko rešitvijo SQLMake.

1.1 Motivacija

V letu 2004 in 2005 sem sodeloval pri razvoju Matičnega Registra (MR) in novega Registra Stalnega Prebivalstva (nRSP). Številna razvojna, testna in produkcijsko okolje so zahtevali neprestane namestitve in nadgradnje podatkovne sheme, vračanje stare verzije podatkovne sheme, osvežitev testnih podatkov in tako dalje. Nameščanje novih verzij podatkovne sheme je bilo hitro identificirano kot kritičen korak, ker je zaradi ročnih postopkov in občasnih napak v skriptah pogosto presegalo predviden čas trajanja.

Podobne probleme pri razvoju podatkovne sheme je bilo mogoče opaziti tudi na drugih projektih, med katerimi so nekateri namesto Oracle SUPB (sistem za upravljanje podatkovnih baz) uporabljali Microsoft SQL Server. To je privedlo do razmišljanja, da problemi niso vezani samo na konkreten SUPB in konkretna razvojna orodja, ampak da so vzroki drugje. Pri iskanju skupnega imenovalca so na vseh projektih identificirani kratki razvojni cikli. V njih smo začeli iskati vzroke za nastale probleme pri nameščanju in nadgradnji podatkovne sheme, ki smo jih zaradi operativnih potreb bolj ali manj uspešno odpravljali sproti.

1.2 Opredelitev problema

Osrednji problem, ki je obravnavan, je pomanjkljiva podpora vodenju konfiguracije in zasebnim razvojnim okoljem pri razvoju podatkovne sheme. V nadaljevanju je problem razčlenjen na tri dele:

- *izdelava sql skript,*
- *nameščanje skript za nadgradnjo* in
- *razvoj shranjenih procedur*.

Izdelava sql skript

Razvoj podatkovne sheme se izvaja s pomočjo različnih razvojnih orodij, kot so orodja za podatkovno modeliranje[1], orodja za razvoj shranjenih procedur[2], za izdelavo testnih podatkov idr. Končni rezultat razvoja so ne glede na uporabljeno razvojno orodje sql skripte, ki se uporabljajo za namestitev podatkovne sheme.

Razvojna orodja izhajajo iz koncepta, da se sql skripte generira na podlagi stanja njihovega lokalnega razvojnega okolja, ki je lahko repozitorij ali pa na primer razvojna podatkovna shema. Koncept popolnoma ustreza razvoju z dolgimi razvojnimi cikli, kjer je ob koncu cikla razvojno okolje stabilno in se v celoti preslika v sql skripte.

V primeru uporabe kratkih razvojnih ciklov je utopično pričakovati stabilno razvojno okolje z ustrezno konfiguracijo, ki je potrebno za izdelavo sql skript. Problem je rešljiv z uporabo zasebnih razvojnih okolij, kjer se v sistem za vodenje konfiguracije iz zasebnega razvojnega okolja prenaša le izbrane spremembe in s tem nadzoruje konfiguracijo podatkovne sheme. Težava nastopi, ker večina orodij za razvoj podatkovne sheme ne podpira oziroma slabo podpira koncept zasebnih razvojnih okolij. Vzdrževanja skript se je zato potrebno lotiti ročno, kar prinese dodatno delo in poveča možnost napake.

Nameščanje skript za nadgradnjo

Po začetni namestitvi podatkovne sheme se vse nove verzije oziroma spremembe podatkovne sheme namešča z uporabo sql skript za nadgradnjo. Število sprememb in posledično skript za nadgradnjo se pri uporabi kratkih razvojnih ciklov močno poveča. Zaradi tega se pojavi potreba po administraciji nameščanja skript za nadgradnjo: katero skripto za

---

[1] Na primer Oracle Designer, Erwin DbDesigner, Power Builder, Enterprise Architect, Rational Rose idr.

[2] Na primer SQL Developer, TOAD, PL/SQL Developer idr.

nadgradnjo se je že namestilo in katero je še potrebno namestiti? Zelo napačno bi bilo na primer, da se dvakrat požene isto skripto za nadgradnjo, ki zaposlenim pretvori plačo iz tolarske valute v evre. Pomembno vprašanje, ki ga je potrebno pri tem rešiti, je tudi poimenovanje skript za nadgradnjo in zaporedje njihovega izvajanja.

Dodaten problem nastopi z večanjem števila okolij. Za vsako razvojno, testno in produkcijsko okolje je potrebno voditi lastno evidenco o nameščenih skriptah za nadgradnjo. Ročna evidenca hitro nastopi kot omejitev in vir napak, zato je potrebno razmišljati v smeri avtomatizacije nameščanja skript za nadgradnjo, ki je podprto z ustrezno evidenco namestitev.

Razvoj shranjenih procedur

Najbolj dinamičen del razvojnega okolja je programska koda, kamor sodijo tudi shranjene procedure. Njihova uporaba se je zelo razširila, zato ni nič nenavadnega, če se v podatkovni shemi nahaja več sto shranjenih procedur. Takšno količino programske kode razvija več programerjev, ki svoje delo opravljajo vsak v svojem zasebnem razvojnem okolju, v svoji razvojni podatkovni shemi.

Ob nameščanju sprememb v zasebno razvojno okolje je delež dejansko spremenjenih shranjenih procedur v večini primerov zelo majhen. Kljub temu ne obstaja avtomatski postopek namestitve, ki bi namestil samo spremenjene shranjene procedure, temveč je potrebno vsakič namestiti vse shranjene procedure od začetka.

1.3 Opredelitev cilja in rezultata

Cilj magistrske naloge je izdelava pregleda metod dobre prakse za uporabo zasebnih razvojnih okolij in sistema za vodenje konfiguracije pri razvoju podatkovne sheme. Pri tem je kot ključna izpostavljena definicija in dopolnitev podporne aktivnosti namestitve (angl. *install*) in nadgradnje (angl. *upgrade*) podatkovne sheme na podlagi sql skript.

Rezultat magistrske naloge je rešitev SQLMake, s katero so omenjeni postopki namestitve in nadgradnje podatkovne sheme z uporabo sql skript poenostavljeni in avtomatizirani.

1.4 Metode dela

Pri raziskavi uporabe agilnih pristopov v razvoju podatkovnih baz so uporabljene naslednje metode dela:

- študij teorije na področju agilnih metodologij in agilnih pristopov,

- sodelovanje na praktičnih projektih, kjer so zabeleženi konkretni problemi in prakse pri namestitvi in nadgradnji podatkovne sheme ter
- izdelava rešitve SQLMake, s pomočjo katere je izvedena poenostavitev in avtomatizacija postopka namestitve in nadgradnje podatkovne sheme.

1.5 Temeljni viri

Kot osrednji vir s področja agilnih metodologij in agilnih pristopov pri razvoju podatkovnih baz je uporabljeno delo Amblerja in Sadalaga, kjer so predstavljeni agilni pristopi na področju razvoja podatkovnih baz [Ambler06]. V magistrski nalogi je razdelan njun prispevek s področja vodenja konfiguracije in zasebnih razvojnih okolij. Njun prispevek je dopolnjen s poenostavljenim in avtomatiziranim postopkom namestitve in nadgradnje podatkovne sheme, ki predstavlja pomemben element pri zagotavljanju kakovosti končnega izdelka.

Osrednji vir s področja gradnje in namestitve predstavlja članek, ki ga je objavil S. I. Feldman v letu 1979 [Feldman]. Članek opisuje način delovanja orodja make, ki še vedno velja kot pomembno orodje za gradnjo programske opreme. Ideja selektivnega prevajanja, ki jo opiše Feldman, je prenesena v proces razvoja shranjenih procedur.

Ker je glavni rezultat magistrske naloge praktično usmerjen, so kot literatura služili tudi tehnični priročniki za SQL*Plus in katalog Oraclove SUPB.

1.6 Implementacijske opombe

Cilji in rezultat magistrske naloge so praktično usmerjeni, zato so v tem poglavju navedene implementacijske omejitve:

- primeri SQL ukazov in skript so zapisani v Oracle SQL oziroma v Oracle SQL*Plus skriptnem jeziku;
- rezultat magistrske naloge, rešitev SQLMake, je namenjen uporabi s SUPB proizvajalca Oracle in je izdelan v okolju Visual Studio .NET s programskim jezikom C#.

Rešitve SQLMake ni možno neposredno uporabiti z drugimi SUPB, ker so za učinkovito implementacijo uporabljene lastnosti Oracle SUPB, ki niso del ANSI SQL standarda [SQL03].

1.7 Zgradba nadaljnjega besedila

Na začetku je opredeljen problem, ki izhaja iz uvajanja agilnih pristopov v razvoj podatkovne sheme.

V drugem poglavju je napravljen pregled agilnih pristopov pri razvoju podatkovnih baz: vodenje konfiguracije, zasebna razvojna okolja, regresijsko testiranje, preoblikovanje in evolucijski razvoj podatkovnih baz. Pregled je dopolnjen s predstavitvijo sorodnih raziskav, ki nudijo dodatne poglede na razvoj podatkovne sheme.

V tretjem poglavju so opisani trije primeri iz prakse: razvoj informacijskega sistema CRP (Centralni register prebivalstva), razvoj informacijskega sistema MRnRSP (Matični register in novi register stalnega prebivalstva) in razvoj informacijskega sistema UCG (Unified Collection Gateway). Razvoj pri prvem je potekal po metodologiji strukturnega razvoja EMRIS. Pri drugih dveh primerih je uporabljena razvojna metodologija s kratkimi iterativno-inkrementalnimi razvojnimi cikli. Pri vseh so opisani postopki razvoja s stališča dveh agilnih pristopov pri razvoju podatkovnih shem: vodenja konfiguracije in zasebnih razvojnih okolij.

V četrtem poglavju je predstavljen pregled postopka namestitve podatkovne sheme, ki se pri predstavljenih primerih iz prakse izkaže kot kritičen. Predstavljene so osnovne definicije, življenjski krog podatkovne sheme, postopek namestitve in postopek nadgradnje podatkovne sheme. Zaradi pogoste uporabe shranjenih procedur v sodobnih informacijskih sistemih je predstavljena podobnost z razvojnim ciklom programskih jezikov tretje generacije in uporaba ideje selektivnega prevajanja [Feldman].

V petem poglavju je predstavljena rešitev SQLMake in spremembe ter prednosti, ki jih uporaba rešitve prinese glede na obstoječe postopke namestitve in nadgradnje podatkovne sheme.

V sklepu so ovrednotene glavne ugotovitve magistrske naloge in navedene možnosti za nadaljnje delo.

2 Agilni pristopi pri razvoju podatkovnih baz

V tem poglavju je napravljen kratek uvod v agilne metodologije in raziskave s tega področja. Sledi pregled agilnih pristopov pri razvoju podatkovnih baz, ki je povzet po [Ambler06]. Podrobno sta opisana dva temeljna pristopa: vodenje konfiguracije in zasebna razvojna okolja ter težave pri njuni uporabi v razvoju podatkovne sheme.

V drugem delu so predstavljene sorodne raziskave, ki se ukvarjajo s spremembami oziroma z obvladovanjem sprememb podatkovne sheme:

- evolucija podatkovne sheme,
- verzioniranje podatkovne sheme in
- lahkotna evolucija podatkovne sheme.

2.1 Agilne metodologije

V devetdesetih letih prejšnjega stoletja so se pojavile številne metodologije, ki so nastale delno kot odraz nezadovoljstva pri uporabi tako imenovanih težkih metodologij ter delno kot reakcija na nove izzive, ki jih je s seboj prinesla doba interneta. Nekaj izmed novih metodologij je: *Adaptive Software Development* (ASD), *Crystal*, *Dynamic System Development Method* (DSDM), *Extreme Modeling*[3], *Extreme Programming* (XP), *Feature Driven Development* (FDD), *Lean Development*, *Scrum* idr.

Sedemnajst predstavnikov teh novih metodologij, imenovanih tudi lahke metodologije, se je sestalo decembra 2001, z namenom, da definirajo skupne točke svojih metodologij. Srečanje so sklenili z objavo izjave o agilnosti (angl. *agile manifesto*), ki predstavlja začetek tako imenovanih agilnih metodologij [Agile Manifesto].

Vse zgoraj navedene lahke metodologije so opredeljene kot agilne, vendar to ne pomeni, da je agilnost povezana s težo metodologije. Tudi težka metodologija je lahko agilna, če omogoča prilagajanje obsega metodologije dejanskim potrebam projekta [Bajec]. Bistvo agilnih metodologij je poleg prilagodljivosti zapisano v sledečih štirih načelih:

- **posamezniki in njihova komunikacija** imajo prednost pred procesi in orodji,
- **delujoča programska oprema** ima prednost pred popolno dokumentacijo,
- **vključevanje (sodelovanje) uporabnika** ima prednost pred pogajanji na osnovi pogodb in

[3] Kasneje je preimenovana v *Agile Modeling* (AM)

- **upoštevanje sprememb** ima prednost pred sledenjem načrtu.

Jasno je izražena usmeritev v poudarjanje posameznikovega znanja, v medsebojno komunikacijo in upoštevanje sprememb kot sestavnega dela razvojnega procesa. Načela s tem seveda ne izključujejo procesov, orodij, dokumentacije, pogodb in načrtov, ki še vedno ostajajo zelo pomemben del razvoja informacijskih sistemov.

Pomemben koncept, ki je povezan z agilnimi metodologijami, je iterativno-inkrementalni razvojni cikel. Iterativnost in inkrementalnost sta opredeljeni med štirimi lastnostmi agilnih metodologij na prvi eDelavnici o agilnih metodologijah junija 2002 [Agile eWorkshop02]. Sam koncept iterativno-inkrementalnih razvojnih ciklov je že zelo star [Larman], venar je ravno s popularizacijo agilnih metodologij stopil v ospredje. Sočasno s tem konceptom so dozoreli tudi podporni pristopi, ki kljub pogostim spremembam služijo pri zagotavljanju potrebne ravni kakovosti programske kode:

- preoblikovanje (angl. *refactoring*),
- avtomatsko testiranje ter
- neprestana gradnja in integracija (angl. *continous build and integration*).

2.2 Agilnost in podatkovne baze

Ideje agilnih metodologij na področje razvoja podatkovnih baz preneseta Ambler in Sadaladge, ki definirata evolucijski razvoj podatkovnih baz kot sintezo agilnih pristopov (*Agile Data* - AD, *Agile Modeling* - AM, *Agile Model Driven Development* - AMDD) in vloge sodobnega razvoja podatkovne sheme [Ambler06]. Evolucijski razvoj podatkovnih baz je sestavljen iz petih pristopov:

1. uporaba sistemov za vodenje konfiguracije,
2. zasebna razvojna okolja,
3. regresijsko testiranje,
4. preoblikovanje podatkovne sheme in
5. evolucijsko podatkovno modeliranje.

V nadaljevanju je opisan vsak izmed naštetih petih pristopov.

2.2.1 Uporaba sistemov za vodenje konfiguracije

Standardi kakovosti kot temelj kakovosti programskega inženirstva postavljajo vodenje konfiguracije [Pivka]. Tudi metodologija EMRIS ugotavlja, da je uspešnost in

učinkovitost iterativno-inkrementalnega razvojnega procesa zelo odvisna ravno od ustreznega nadzora verzij in upravljanja konfiguracije [EMRIS-Objektni].

Številni produkti, kot so Clear Case, Perforce, Subversion in CVS, pričajo, da so sistemi za vodenje konfiguracije zreli produkti, ki nudijo številne prednosti: vodenje zgodovine verzij posameznih datotek, nemoteno vzporedno delo več razvijalcev, povrnitev starejše verzije datoteke v primeru, da se sprememba izkaže za slabo, vejanje (angl. *branching*), ki omogoča vzporedno delo na novejši različici informacijskega sistema idr.

Večina naštetih komercialnih in odprtokodnih sistemov za vodenje konfiguracije temelji na datotečnem sistemu. Ta generičen pristop ima močne omejitve pri vodenju verzij izdelkov s kompleksno interno strukturo, kot so podatkovne sheme in metapodatkovni repozitoriji, zato je uporaba sistemov za vodenje konfiguracije na področju razvoja podatkovnih shem omejena na zelo osnovno raven.

Da se lahko prednosti sistema za vodenje konfiguracije izkoristi v polni meri, je potrebno razvoj podatkovne sheme organizirati z uporabo sql skript. V obliki sql skript se lahko v sistemu za vodenje konfiguracije zapiše naslednje elemente podatkovne sheme:

- skripte za kreiranje podatkovne sheme,
  - tabele, indeksi, omejitve,
  - pogledi,
  - shranjene procedure in prožilci,
- skripte za prevedbo podatkov, vzpostavitev začetnega stanja in izvoz podatkov,
- šifranti,
- testni podatki in skripte za generiranje testnih podatkov ter
- testne skripte.

Shranjevanje elementov podatkovne sheme v datoteke ni definirano znotraj ANSI SQL standardov, zato se naleti na različne pristope pri poimenovanju končnice datotek. Splošno uveljavljena končnica za sql skripte je »*.sql*«, možne so tudi druge končnice kot je na primer »*.tab*« za tabele. S stališča uporabe sistemov za vodenje konfiguracije je smiselno vsak element podatkovne sheme voditi v ločeni datoteki, ker lahko na ta način transparentno vodimo njegovo verzijo in zgodovino.

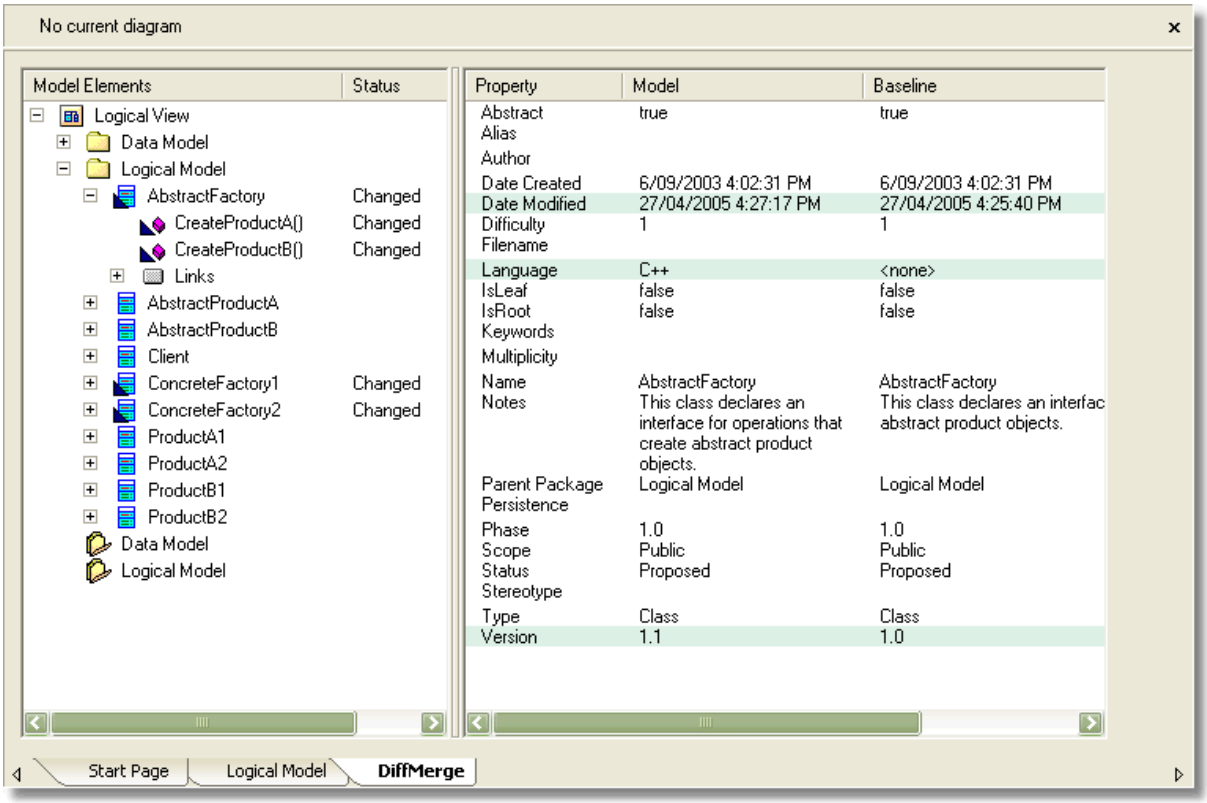
2.2.1.1 Vodenje konfiguracije in podatkovno modeliranje

Pri izdelavi podatkovnega modela se najpogosteje uporablja orodja za podatkovno modeliranje. Njihova povezava s sistemi za vodenje konfiguracije ni preprosta, ker so ti

zasnovani generično in imajo številne omejitve pri verzioniranju izdelkov s kompleksno interno strukturo, kot so podatkovni modeli. Omejitve so med drugim nezmožnost natančnega vodenja verzij posameznih elementov podatkovnega modela, nezmožnost konsistentnega združevanja (angl. *merge*) in odkrivanje konfliktov med posameznimi elementi [Odyssey-VCS].

Nekatera izmed orodij, kot so Enterprise Architect[4], Borland Together[5] in Rational Software Architect[6], nudijo integracijo z obstoječimi datotečnimi sistemi za vodenje konfiguracije. Običajno razdelijo model na manjše dele, ki jih nato v obliki datotek vodijo v sistemu za vodenje konfiguracije. Ta pristop omogoča uporabo pesimističnega zaklepanja, s katerim se nadzoruje sočasnost dela.

Slika 1: Primer orodja za primerjanje in združevanje razlik med modeli



Za uporabo optimističnega zaklepanja je pri združevanju potrebno uporabiti zunanje orodje za združevanje razlik med modeli, kot ga prikazuje slika 1. Z uporabo standardiziranega XMI zapisa je odprta pot za razvoj univerzalnih orodij za primerjanje in združevanje razlik med modeli [Girschick].

---

[4] http://www.sparxsystems.com.au/

[5] http://www.borland.com/us/products/together/index.html

[6] http://www-01.ibm.com/software/awdtools/architect/swarchitect/

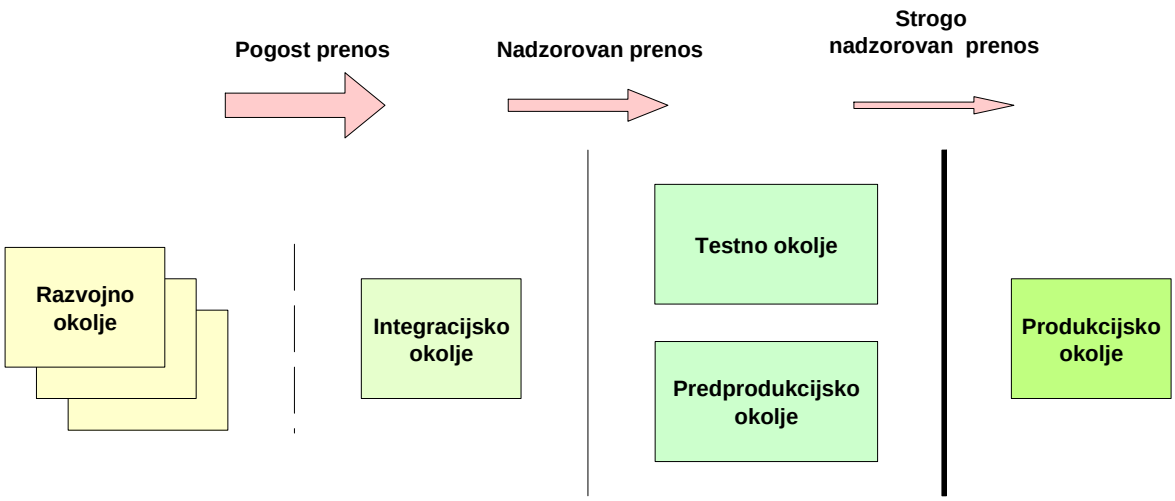
Namesto datotečnih sistemov za vodenje konfiguracije je možno izbrati tudi alternativne pristope. Pri podjetju Oracle so poskušali znotraj repozitorija orodja Oracle Designer razviti lasten sistem za vodenje konfiguracije meta podatkov. Rezultat je daleč od uporabnosti razširjenih sistemov za vodenje konfiguracije, zato je Oracle nadaljnji razvoj opustil.

V prihodnosti je na področju vodenja verzij podatkovnih in drugih modelov največ pričakovanj od standarda Meta Object Facility[7] (MOF), ki med drugim predpisuje tudi repozitorij za shranjevanje meta podatkov in v zadnji verziji specifikacije iz maja 2007 definira tudi način vodenja verzij.

2.2.2 Zasebno razvojno okolje

Zasebno razvojno okolje ali peskovnik (angl. *sandbox*) je razvojno okolje s polno funkcionalnostjo, v katerem se lahko informacijski sistem neodvisno namesti, testira in/ali uporablja. Poleg zasebnih razvojnih okolij v razvoju nastopajo tudi druga okolja, kot so integracijsko okolje, testno okolje, predprodukcijsko okolje in produkcijsko okolje. Takšna organizacija zagotavlja medsebojno neodvisna okolja, kjer lahko programerji nemoteno programirajo, testerji nemoteno testirajo in uporabniki informacijskega sistema nemoteno uporabljajo produkcijsko okolje. Kot prikazuje slika 2, se vse spremembe, ki nastanejo na podatkovni shemi, preko vmesnih okolij propagirajo do produkcijskega okolja.

Slika 2: Okolja in prenos sprememb med njimi



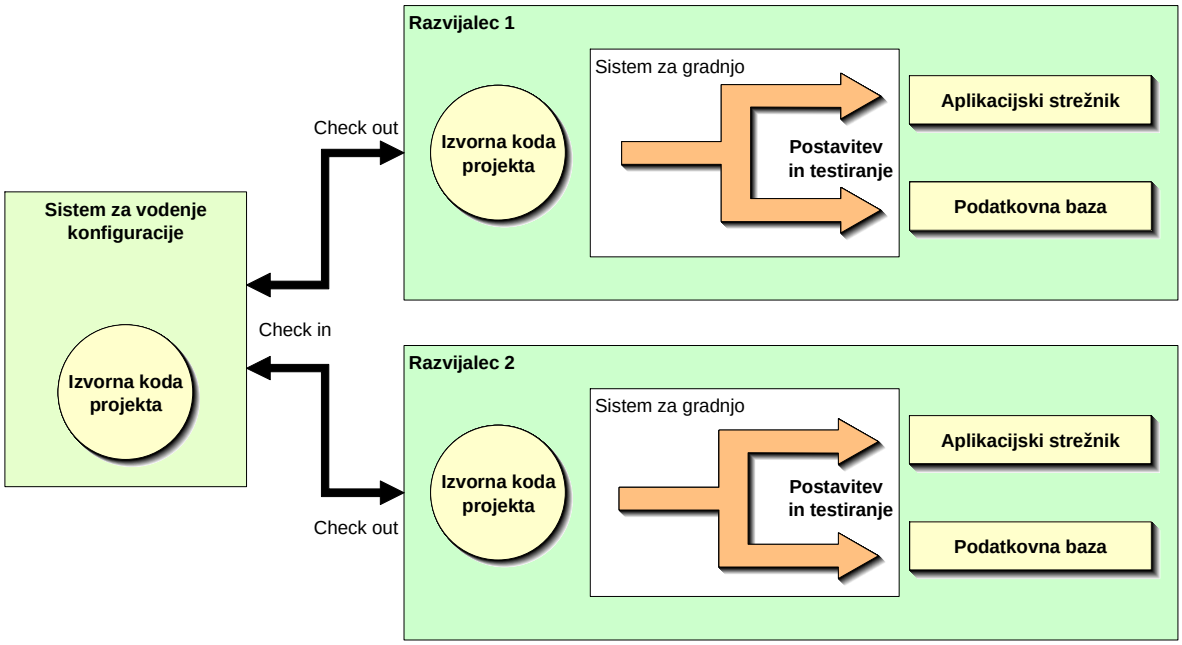
[7] http://www.omg.org/mof/

Posamezna okolja, kot jih prikazuje slika 2, se lahko razgradi podrobneje. Slika 3 prikazuje shematski prikaz idealnega razvojnega okolja tipične trinivojske aplikacije, ki vključuje aplikacijski strežnik in podatkovno bazo. Vsak razvijalec ima pri sebi na voljo celotno razvojno okolje z lastno podatkovno bazo in aplikacijskim strežnikom. Znotraj svojega okolja je razvijalec popolnoma samostojen, tako da njegove spremembe nimajo neposrednega vpliva na ostale razvijalce.

Idealno okolje je opisano z naslednjimi karakteristikami [Schuh]:

- samozadostno,
- čim bolj podobno produkcijskemu okolju in
- vsebuje sistem za gradnjo aplikacije (angl. *build system*), ki prevede izvorno kodo v izvršilno obliko (angl. *executable*), jo avtomatsko namesti na aplikacijski strežnik in podatkovno bazo ter izvede avtomatsko regresijsko testiranje.

Slika 3: Shema idealnega razvojnega okolja

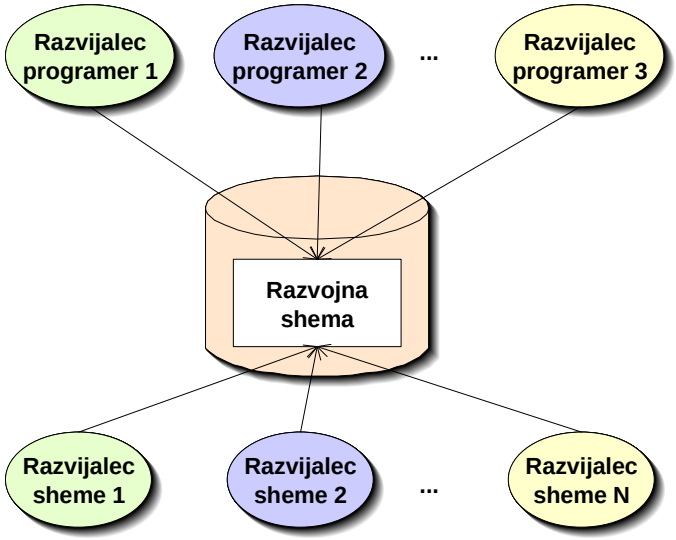


Vzdrževanje kompleksne infrastrukture v zasebnem razvojnem okolju je lahko zahtevno, zato se pogosto uvede sledeče poenostavitve:

- namesto pravega aplikacijskega strežnika se uporabi aplikacijski strežnik, ki je del integriranega razvojnega okolja, kot sta na primer Visual Studio in Eclipse. Integrirane različice porabijo precej manj virov na delovni postaji in so manj zahtevne glede konfiguracije,

- namesto lastne SUPB vsi razvijalci uporabljajo skupno SUPB, ki jo centralno upravlja izkušen administrator. Ločena okolja so zagotovljena tako, da ima vsak razvijalec svojo lastno podatkovno shemo znotraj skupne SUPB.
- pogost pojav je souporaba skupne razvojne podatkovne sheme, ki si jo delijo vsi razvijalci (slika 4). Souporaba skupne razvojne sheme lahko privede do medsebojnih motenj, katerih učinek se zmanjša z dobro organizacijo in komunikacijo med razvijalci. Kljub temu se razvijalci pogosto namesto s trenutno nalogo ubadajo z raziskovanjem problema, ki je največkrat posledica nestabilnega okolja v skupni razvojni shemi.
- možen pristop je tudi uporaba predpripravljenih virtualnih strojev znotraj virtualnih okolij, kot sta Vmware in VirtualPC. Virtualni stroji so enostavno prenosljivi, vnaprej konfigurirani in omogočajo preprosto izdelavo novega okolja.

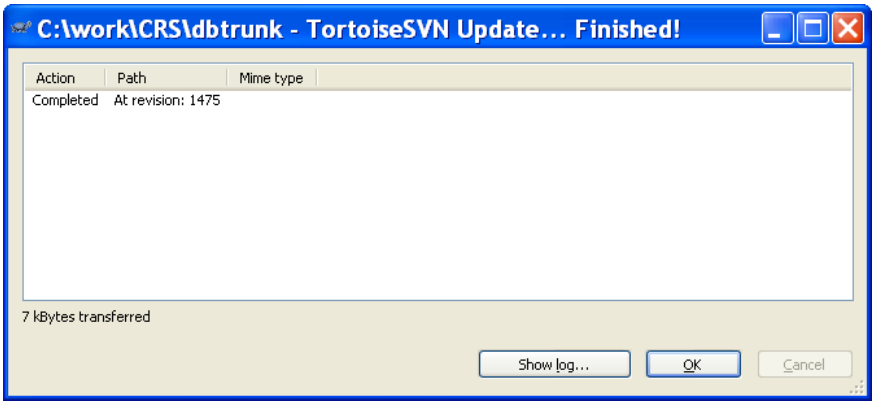
Slika 4: Souporaba iste podatkovne sheme



Spremembe, ki jih razvijalci naredijo v svojem zasebnem razvojnem okolju, ob določenih časovnih trenutkih prenašajo v centralni sistem za vodenje konfiguracije. Istočasno se iz njega prevzame spremembe, ki so jih naredili drugi razvijalci. Slika 5 prikazuje, kako preprosto je izvesti osveževanje lokalne kopije datotek izvorne kode z zadnjim stanjem v sistemu za vodenje konfiguracije Subversion. Postopek osveževanja sodi med osnovne funkcionalnosti sistemov za vodenje konfiguracije.

Pri gradnji aplikacije obstajajo tudi številne pomožne datoteke, ki jih za svojo delovanje uporabljata prevajalnik in povezovalnik. Te datoteke nastanejo s prevajanjem datotek izvorne kode. Ko se iz sistema za vodenje konfiguracije prevzame novo verzijo izvorne kode, je potrebno vse te pomožne datoteke ponovno izdelati. Toda katere? Prevajanje je časovno zelo zahtevna operacija. Bolj smiselno je prevesti le tiste pomožne datoteke, ki so povezane s spremenjeno izvorno kodo.

Slika 5: Prenos sprememb iz sistema za vodenje konfiguracije v zasebno razvojno okolje



Tu se srečamo s problemom, ki ga je v letu 1979 opisal Feldman, ko je izdelal orodje m*ake* [Feldman]. Make se še vedno množično uporablja pri razvoju, vzdrževanju in nadgrajevanju programske opreme v UNIX okolju. Počasi ga nadomeščajo novejša orodja, kot je ANT. Glavna funkcionalnost orodja make je, da ob vsaki spremembi v izvorni kodi vzdržuje vse datoteke, ki so povezane s to spremembo. Rezultat je usklajeno stanje izvorne kode in vseh datotek, ki so iz nje nastale. Z upoštevanjem odvisnosti v izvorni kodi je celoten postopek izveden v najmanjšem možnem obsegu.

Podoben problem neusklajenosti se nahaja tudi med sql skriptami in razvojno podatkovno shemo. Po prenosu spremenjenih sql skript iz sistema za vodenje konfiguracije v lokalni datotečni sistem je vsebina razvojne podatkovne sheme neusklajena z vsebino sql skript. Potreben je postopek, s katerim se podatkovna shema ažurira na trenutno veljavno stanje, ki se nahaja v sql skriptah. Postopek je predstavljen v nadaljevanju kot del rešitve SQLMake.

2.2.2.1 Podatkovno modeliranje

Orodja za podatkovno modeliranje se zaradi slabe povezanosti s sistemi za vodenje konfiguracije zelo težko vključi v koncept zasebnih razvojnih okolij. Medtem ko se nekaterih orodij, kot je Oracle Designer, sploh ne da neposredno povezati z zunanjimi sistemi za

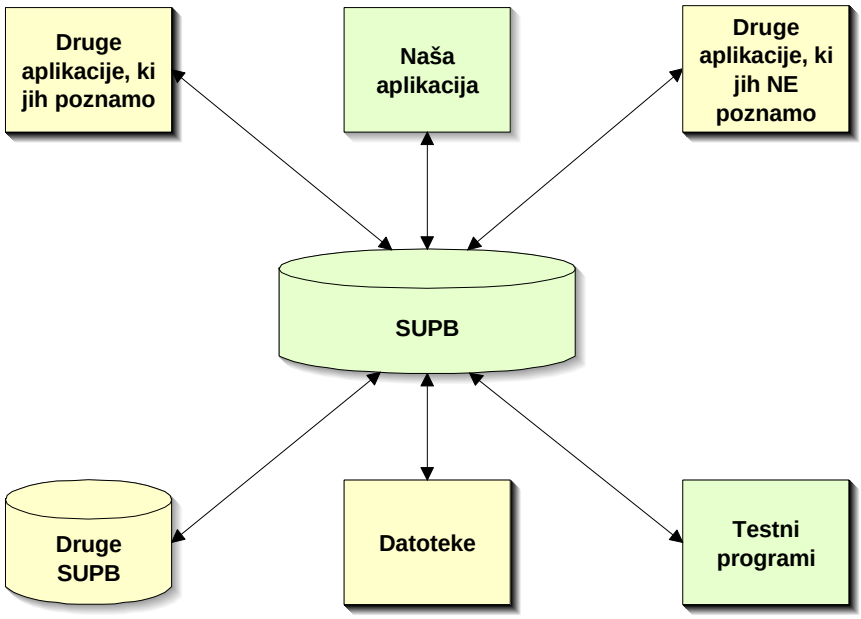
vodenje konfiguracije, je pri drugih orodjih, kot je Sparx Enterprise Architect, ta povezava funkcionalno zelo slabo izdelana, ker omogoča le pesimistično zaklepanje na ravni celotnega modela.

2.2.3 Preoblikovanje podatkovne sheme

Fowler definira preoblikovanje (angl. *refactoring*) kot discipliniran način uvajanja sprememb v izvorno kodo z namenom izboljšanja strukture in lažjega vzdrževanja [Fowler99]. Glavna lastnost preoblikovanja je ohranjanje obnašanja izvorne kode. Preoblikovanje ne dodaja in ne odvzema funkcionalnosti, temveč samo izboljšuje kakovost. Primer preoblikovanja je na primer preimenovanje metode vrniStudent() v vrniOseba().

Podobno definira Ambler preoblikovanje podatkovne sheme kot preprosto spremembo na shemi, ki ne vpliva na delovanje aplikacije ter ne spreminja vsebine in pomena informacij [Ambler06]. Ugotavlja tudi, da je preoblikovanje podatkovne sheme težje izvedljivo kot preoblikovanje programske kode. Razlog leži v centralni umestitvi SUPB v informacijski sistem, ki jo prikazuje slika 6. Preoblikovanje podatkovne sheme s seboj pogosto potegne tudi preoblikovanje povezanih sistemov.

Slika 6: SUPB kot centralno vozlišče



Ambler definira naslednje vrste preoblikovanja podatkovne sheme:

- preoblikovanje strukture podatkovne sheme (angl. *structural refactoring*),

- preoblikovanje z namenom izboljšanja kakovosti podatkov (angl. *data quality refactoring*),
- preoblikovanje referencialne integritete (angl. *referential integrity refactoring*),
- preoblikovanje arhitekture dostopa do podatkovne sheme (angl. *architectural refactoring*) in
- preoblikovanje shranjenih procedur (angl. *method refactoring*).

Preoblikovanje implementira s postopki evolucije podatkovne sheme. Pri tem se loti tudi vpliva preoblikovanja na (ne)delovanje aplikacije, kar rešuje z uvedbo dodatne logike glede na tip preoblikovanja. To doseže na različne načine, na primer s sočasnim vzdrževanjem istega podatka na dveh mestih. Pri tem predpostavi tudi prehodno obdobje, v katerem je potrebno vse odvisne aplikacije popraviti, da dostopajo do podatka na novem mestu[8]. Po prehodnem obdobju se dvojno vzdrževanje podatka ukine.

Probleme odvisnosti med aplikacijo in SUPB je možno reševati tudi s konceptom verzioniranja podatkovne sheme, kjer se dostop do podatkov izvaja preko poljubno izbrane verzije podatkovne sheme. Obstoječe aplikacije lahko, kljub spremembi podatkovne sheme, do podatkov dostopajo preko predhodne verzije, tako da spremembe na njih ne vplivajo. Žal koncept verzioniranja podatkovne sheme, kljub temu, da je sestavni del SQL3 standarda, ni implementiran v nobeni izmed razširjenih relacijskih SUPB[9].

2.2.4 Regresijsko testiranje

Fowler v svoji knjigi o preoblikovanju programske kode kot prvi korak k uspešnemu preoblikovanju navede avtomatizirano testiranje enot [Fowler99]. Avtomatizirano testiranje je postalo pomembna praksa agilnih metodologij, ki s pogostim izvajanjem testov zagotavljajo kakovost končnega izdelka. Najbolj izrazit poudarek na testiranju ima testno voden razvoj (angl. *Test Driven Development*, TDD), kjer testiranje predstavlja tudi način načrtovanja in razvijalcu pomaga pri razumevanju, definiranju in modeliranju problema [Beck03].

Ambler [Ambler06] namesto izraza avtomatizirano testiranje in testiranje enot uporablja izraz regresijsko testiranje, kljub temu, da se pri praktičnih primerih sklicuje na avtomatizirano testiranje enot. Razlika je precejšnja, ker je namen regresijskega testiranja precej bolj splošen kot testiranje enot in je definiran kot odkrivanje sprememb v delovanju

---

[8] Obstoječa funkcionalnost, ki jo nadomešča nova funkcionalnost in bo v prihodnjih različicah verjetno odstranjena, se v računalništvu označuje s pridevnikom *deprecated*.

[9] Preverjeno pri SUPB Oracle 11g, SQL Server 2005, MySQL in DB2

programske opreme po njeni spremembi ali spremembi njenih nastavitev. Regresijsko testiranje je lahko ročno ali avtomatizirano, uporabljamo lahko testiranje enot ali druge vrste testiranja, kot so integracijsko testiranje, sistemsko testiranje in obremenitveni test. Avtomatizirano testiranje enot, ki ga Fowler postavi kot prvi korak k uspešnemu preoblikovanju, je tako le del regresijskega testiranja.

Razloge, da Ambler razširi testiranje enot na regresijsko testiranje, lahko iščemo v tem, da podatkovno shemo le stežka testiramo samostojno. Izkaže se, da pri testiranju podatkovne sheme pomemben del testiranja opravimo že pri testiranju aplikacije, ki dostopa do podatkovne sheme. Testiranje enot v aplikaciji vključuje tudi dostope do podatkovne sheme. Pri tem le težko govorimo o pravem testiranju enot, ker je programska koda, ki se testira, odvisna od stanja podatkovne sheme in se s tem dejansko opravlja integracijsko testiranje.

Poleg testiranja programske kode za dostop do podatkovne sheme je potrebno izdelati tudi teste programske kode, ki se nahaja v podatkovni shemi: shranjene procedure, prožilci, pogledi, privzete vrednosti in omejitve. Testiranja v okolju podatkovne sheme se lotimo na podoben način kot testiranja katerekoli druge programske kode. Pri tem se lahko uporabi odprtokodno knjižnico utPLSQL[10], ki je nastala po vzoru xUnit knjižnic za testiranje, ali komercialna orodja, kot je Quest Unit Tester[11].

2.2.5 Evolucijsko podatkovno modeliranje

Ambler v svojih delih izpostavi modeliranje kot ključni element za doseganje učinkovitosti programskega razvoja. Predlaga, da se podatkovne modele, podobno kot ostale dele informacijskega sistema, izdeluje iterativno in inkrementalno ter s tem zagotavlja, da podatkovni model raste sočasno s programsko kodo. Vsi predhodno opisani agilni pristopi služijo kot podporni elementi za uspešno izvajanje tako definiranega evolucijskega podatkovnega modeliranja.

Evolucijsko podatkovno modeliranje je izpeljano iz agilnega razvoja na podlagi modeliranja (angl. *Agile Model Driven Development*, AMDD). Slika 7 prikazuje splošen proces AMDD, kjer so pri vsaki aktivnosti navedeni tudi približni časovni okviri za njeno izvedbo. Pri evolucijskem podatkovnem modeliranju se namesto obsežnega podatkovnega

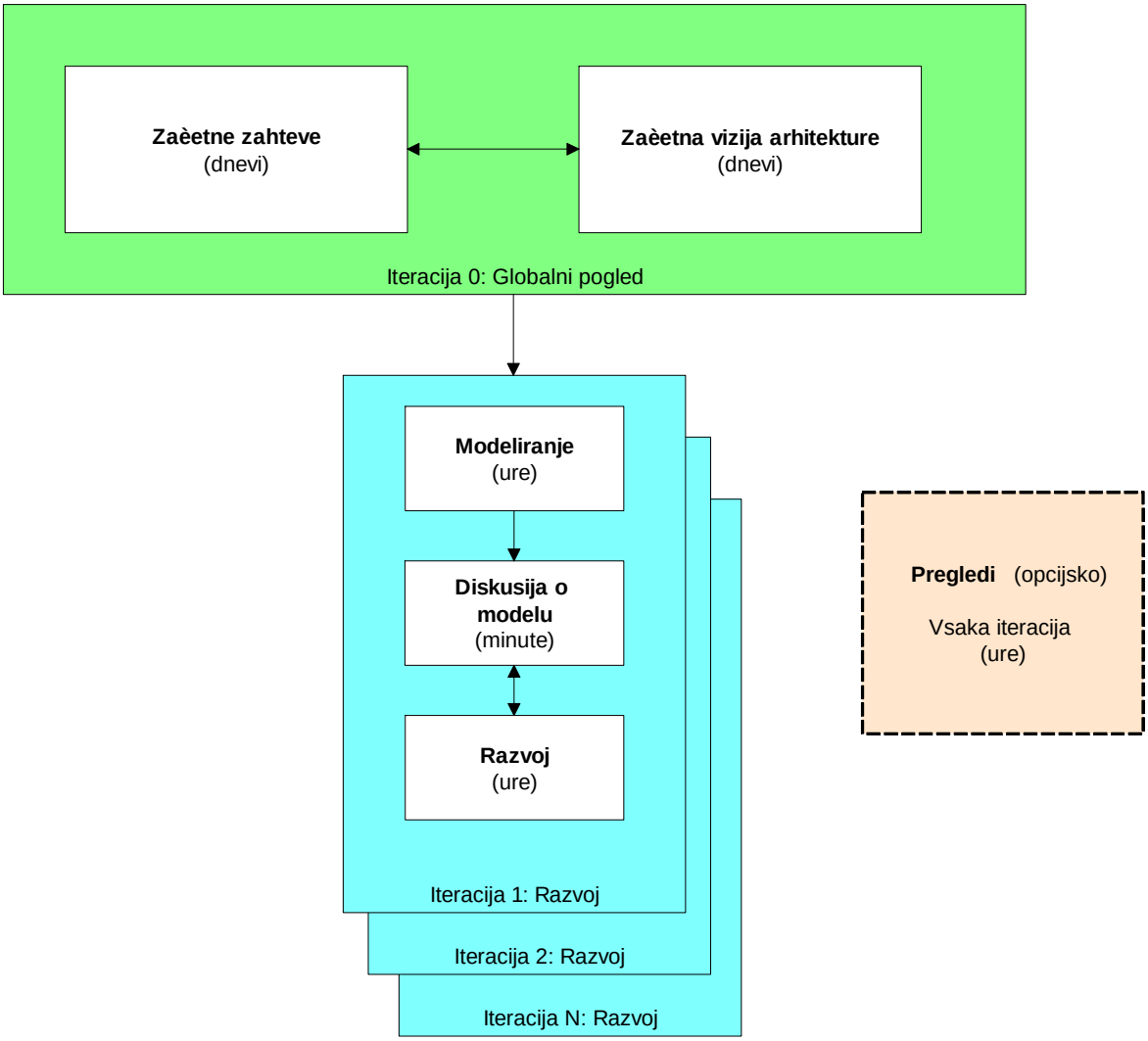
---

[10] http://utplsql.sourceforge.net/

[11] http://unittest.inside.quest.com

modela v ničti iteraciji izdela začetni globalni podatkovni model, ki se nato skozi vsako iteracijo dopolnjuje. Ambler zagovarja izdelavo ravno pravšnjih modelov, s katerimi zadovoljivo končamo posamezno iteracijo razvoja.

Slika 7: Življenjski cikel modeliranja z AMDD



Vlogo AMDD v razvojnem procesu se lahko dodatno opredeli z opisom drugega pristopa, ki predstavlja njegovo diametralno nasprotje. Gre za vnaprejšnjo izdelavo velikega načrta (angl. *Big Design Up Front*, BDUF), ki temelji na naslednjih dveh predpostavkah:

1) Primarni izdelki podatkovnega modeliranja so konceptualni, logični in fizični podatkovni model. Ker so podatki ključna vrednota, jih obravnavamo kot gonilno silo celotnih naporov pri razvoju informacijskega sistema.
2) Čim hitreje je potrebno izdelati vse tri omenjene modele. Reševanja večjih problemov se lotimo na začetku, da se izognemo kasnejšim presenečenjem.

Izdelani modeli omogočijo paralelno izvajanje nadaljnjih aktivnosti, povezanih s prevedbo podatkov in izdelavo aplikacije.

BDUF pristop je značilen za metodologije informacijskega inženiringa, ki so tipični predstavniki težkih metodologij.

2.3 Sorodne raziskave

Med sorodnimi raziskavami je predstavljena evolucija podatkovne sheme, ki predstavlja temelj vsakega spreminjanja podatkovne sheme. Pri njeni uporabi nastopijo težave, ker evolucija podatkovne sheme ni neodvisna aktivnost, saj pogosto povzroči tudi spremembo povezanih aplikacij.

Uporaba kratkih razvojnih ciklov problem spremembe podatkovne sheme dodatno potencira, zato se v okviru agilnih pristopov izdatno posveča sprotni gradnji podatkovne sheme in aplikacije ter njunemu testiranju. Obstajajo pa tudi drugi pristopi k reševanju tega problema, predstavljena sta naslednja dva: verzioniranje podatkovne sheme in lahkotna evolucija podatkovne sheme.

Najbolj znan pristop je verzioniranje podatkovne sheme, ki predstavlja združitev raziskav evolucije podatkovne sheme in časovnih podatkovnih baz. Zanimiv pristop ponuja lahkotna evolucija podatkovne sheme, ki z zapisom sprememb podatkovne sheme v obliki operatorjev omogoča avtomatsko nadgradnjo poizvedb ter s tem omogoča avtomatsko nadgradnjo aplikacij.

2.3.1 Evolucija podatkovne sheme

Evolucija podatkovne sheme predstavlja temelj vsakega razvoja podatkovne sheme. Roddick postavlja sledečo definicijo evolucije podatkovne sheme: »*Evolucija podatkovne sheme je podprta, ko SUPB omogoča spreminjanje strukture polne podatkovne baze, ne da pri tem izgubimo podatke*« [Roddick95]. Evolucija ne zahteva vodenja zgodovine kataloga SUPB, temveč predpostavlja le možnost spreminjanja kataloga brez izgube podatkov. V praksi je potreba po ohranjanju predhodnih definicij sheme zaželena, vendar to področje pokriva verzioniranje podatkovne sheme.

Proces evolucije podatkovne sheme je sestavljen iz drobnih korakov, kot je na primer dodajanje novega stolpca, sprememba podatkovnega tipa in podobno. Primer evolucije je prikazan na spodnjem primeru.

Primer 1: Evolucija relacije zaposleni

1. Januar 2006 Relacija ZAPOSLENI je definirana kot:

```
Id       number(6),
Ime      varchar2(35),
Placa    number(4)
```

```
create table zaposleni (
  Id        number(6),
  Ime       varchar2(35),
  Placa     number(4)
)
```

1. Februar 2006 Dodatni atributi

```
Spol           char(1),
ZakonskiStan   char(1)
```

```
alter table zaposleni add(
  spol char(1),
  zakonskiStan char(1)
)
```

1. Marec 2006 Odstranjen atribut zakonski stan

```
alter table zaposleni
drop (zakonskiStan)
```

ali

```
alter table zaposleni
set unused (zakonskiStan)
```

1. April 2006 Redefinicija atributa plača

```
Placa    number(5,2)
```

```
alter table zaposleni
modify (placa number(5,2))
```

2. April 2006 Redefinicija atributa plača

```
Placa    number(5)
```

```
alter table zaposleni
modify (placa number(5))
```

Končna verzija relacije ZAPOSLENI je:

```
Id       number(6),
Ime      varchar2(35),
Placa    number(5),
Spol     char(1)
```

Pri izvajanju evolucije je potrebno upoštevati različne omejitve, ki so povezane z obstoječimi podatki v tabelah. Kot primer: medtem ko povečanje dolžine stolpca ni problematično, je problem krajšanje dolžine stolpca, v primeru, da že obstajajo zapisi z večjo dolžino. Problematične so lahko tudi pretvorbe med osnovnimi podatkovnimi tipi, ki niso vedno izvedljive, na primer sprememba znakovnega tipa v numerični tip.

K omejitvam se šteje tudi čas, ki je potreben za izvajanje postopkov evolucije. Izvajanje sprememb nad večmilijonsko tabelo vključuje dostop in ažuriranje zapisov v tabeli, kar lahko traja zelo dolgo.

Zadnja izmed omejitev je povezana s transakcijskim izvajanjem evolucije podatkovne sheme. Evolucija je sestavljena iz zaporedja drobnih korakov. V primeru, da pri enem izmed korakov pride do napake, se predpostavlja, da se lahko s pomočjo ukaza *rollback* vedno povrne prvotno stanje pred začetkom izvajanja evolucije. DDL ukazi (angl. *data definition* language), s katerimi se izvaja postopke evolucije, v Oracle SUPB vključujejo tudi implicitno potrditev transakcije, tako da se zaporedje korakov evolucije ne izvaja transakcijsko[12]. V primeru napake je potrebno s pomočjo ročne kompenzacije vzpostaviti prvotno stanje.

Kljub naštetim omejitvam je evolucija podatkovne sheme edini razširjen način spreminjanja podatkovne sheme, ki ga podpirajo vsi razširjeni relacijski SUPB.

2.3.2 Verzioniranje podatkovne sheme in časovne podatkovne baze

Verzioniranje podatkovne sheme si prizadeva, da vse obstoječe aplikacije nemoteno delujejo kljub spremenjeni shemi. Roddick postavlja naslednjo definicijo verzioniranja podatkovne sheme: »*Verzioniranje podatkovne sheme je podprto, ko SUPB omogoča dostop do vseh podatkov, tako preteklih kot bodočih, preko uporabniško izbrane verzije sheme*« [Roddick95]. Aplikacija si sama izbere, preko katere verzije sheme dostopa do podatkov, tako da morebitne spremembe sheme nanjo ne vplivajo.

Težave pri verzioniranju nastopijo predvsem pri dopuščanju popravljanja podatkov skozi zgodovinske verzije sheme. Za omilitev tega se razlikuje:

- *Delno verzioniranje sheme*, ki je podprto, ko SUPB omogoča vpogled do vseh podatkov, tako preteklih kot bodočih, preko uporabniško izbrane verzije sheme. Vnos, popravljanje in brisanje podatkov je omogočeno le preko določene (navadno trenutne) verzije sheme.
- *Polno verzioniranje sheme*, ki je podprto, ko SUPB omogoča vpogled, vnos, popravljanje in brisanje vseh podatkov, tako preteklih kot bodočih, preko uporabniško izbrane verzije sheme.

Podpora verzioniranju podatkovne sheme je zasnovana na podlagi časovnih podatkovnih baz, kjer med najvidnejšimi avtorji nastopa Richard T. Snodgrass. Njegov predlog za razširitev jezika SQL-92 s časovno dimenzijo, TSQL2 [Snodgrass94], je bil sprejet

---

[12] ANSI SQL standard definira ukaza *create schema* in *alter schema*, ki omogočata transakcijsko izvajanje DDL ukazov. Ta del standarda v Oracle SUPB ni v celoti implementiran.

v okviru standarda SQL3 [SQL03] kot del sedmega zvezka z naslovom SQL/Temporal in vključuje tudi verzioniranje podatkovne sheme.

Z SQL/Temporal se odpira možnost dostopa do podatkov preko različnih verzij sheme. SQL/Temporal definira ukaz »SET SCHEMA«, s katerim se eksplicitno izbere verzija sheme. Primer 2 prikazuje SQL stavek, ki izpiše vsebino tabele zaposleni v strukturi, ki je veljala na datum 1. 2. 2006.

Primer 2: Izbira verzije sheme v SQL/Temporal

```
SET SCHEMA DATE '01.02.2006';
SELECT * FROM zaposleni;
```

Kot številni drugi deli SQL standardov tudi SQL/Temporal ni v celoti implementiran v razširjenih relacijskih SUPB.

2.3.3 Lahkotna evolucija podatkovne sheme

Raziskave lahkotne evolucije podatkovne sheme (angl. *graceful schema evolution*), katerih rezultat je orodje PRISM (angl. *Panta Rhei Information & Schema Manager*) [Curino], se problema sprememb podatkovne sheme lotevajo s podporo avtomatiziranemu spreminjanju SQL poizvedb, ki se nahajajo v aplikaciji. Sočasno z nadgradnjo podatkovne sheme se avtomatsko nadgradi tudi podatke in poizvedbe na novo verzijo podatkovne sheme. Na ta način se orodju PRISM prepusti delo, ki sledi spremembam podatkovne sheme.

Postopek spreminjanja podatkovne sheme se začne tako, da DBA spremembe podatkovne sheme opiše preko operatorjev za spremembo sheme (angl. *Schema Modification Operators, SMO*). Orodje na podlagi operaterjev avtomatsko izdela ukaze za nadgradnjo podatkovne sheme, vključno s podatki. Prav tako spremeni obstoječe poizvedbe, da delujejo z novo verzijo podatkovne sheme.

Orodje je še vedno v raziskovalni fazi. V okviru testiranj je podan primer evolucije podatkovne sheme Mediawiki[13], kjer orodje dosega 97,2% uspešnost pri avtomatski spremembi obstoječih poizvedb. Pri preostalih 2,8% poizvedb je za pravilno delovanje aplikacije potreben ročen popravek poizvedb.

---

[13] Wikimedia Foundation, Mediawiki, http://www.mediawiki.org

3 Agilni pristopi v praksi

V tem poglavju so predstavljeni trije primeri razvoja podatkovne sheme: razvoj informacijskega sistema CRP (Centralni register prebivalstva), razvoj informacijskega sistema MRnRSP (Matični register in novi register stalnega prebivalstva) in razvoj informacijskega sistema UCG (Unified Collection Gateway). Razvoj pri prvem je potekal po metodologiji strukturnega razvoja EMRIS. Pri drugih dveh primerih je uporabljena razvojna metodologija s kratkimi iterativno-inkrementalnimi razvojnimi cikli.

Izbrani primeri so predstavljeni s stališča dveh agilnih pristopov pri razvoju podatkovne sheme: uporaba sistemov za vodenje konfiguracije in uporaba zasebnih razvojnih okolij. Opis posameznega primera se začne s kratko predstavitvijo informacijskega sistema in opisom razvojnega okolja. Sledi predstavitev organizacije dela z sql skriptami, postopki namestitve ter nadgradnje podatkovne sheme in statistika[14] količine sprememb sql skript.

3.1 Primer 1: Centralni register prebivalstva

Centralni register prebivalstva (CRP) je informacijski sistem, v katerem so evidentirani prebivalci Slovenije in njihovi pomembni življenjski dogodki, ki se začnejo z rojstvom. CRP ima dolgoletno zgodovino, ki se je začela v osemdesetih letih prejšnjega stoletja [Tršinar]. Svojo zadnjo veliko prenovo je doživel konec devetdesetih let prejšnjega stoletja s prehodom na arhitekturo odjemalec/strežnik s centralno Oracle SUPB.

3.1.1 Razvojno okolje

Razvojna podatkovna shema in repozitorij sta skupna za vse razvijalce. Zaradi razdelitve dela po posameznih modulih so medsebojne motnje med razvijalci minimalne.

Pri razvoju se ni uporabljal sistem za vodenje konfiguracije. Trenutno stanje izvorne kode podatkovne sheme se je nahajalo v SUPB (shranjene procedure) oziroma repozitoriju orodja Oracle Designer (podatkovni model).

3.1.2 Namestitev podatkovne sheme

Razvoj CRP je potekal v skladu z metodologijo EMRIS – strukturni razvoj, kjer se namestitev nove verzije podatkovne sheme v izbrano okolje izvaja ob točno določenih

[14] Statistika ni na voljo pri prvem primeru, kjer se ni uporabljal sistem za vodenje konfiguracije.

aktivnostih, kot so: priprava testnega okolja, priprava okolja za potrditveni test, priprava produkcijskega okolja idr. Namestitev podatkovne sheme je potekala po naslednjem postopku:

- kreiranje sql skript z orodjem Oracle Designer,
- pregled in izvajanje pripravljenih sql skript,
- namestitev podpornih tabel in pogledov za delovanje Oracle Forms in Reports,
- namestitev podpornih tabel za Oracle Headstart[15],
- prenos shranjenih procedur iz razvojne sheme v ciljno podatkovno shemo,
- vnos podatkov, ki so potrebni za delovanje aplikacije, v konfiguracijske tabele in
- vnos podatkov v šifrante.

Postopek namestitve je kompleksen, izvaja se ročno in na številnih mestih ni natančno definiran. Kljub temu nameščanje ni povzročalo večjih preglavic, ker se je izvajalo zelo redko. Dan ali dva, ki sta bila potrebna za vzpostavitev okolja, nista predstavljala velike ovire pri izvajanju nadaljnjih aktivnosti.

3.1.3 Nadgradnja podatkovne sheme

Nadgradenj podatkovne sheme pri razvoju CRP praktično ni bilo. Spremembe po zaključeni fazi testiranja so se odražale predvsem v programski kodi, ki se je nahajala v Oracle Forms in Reports. Vse manjše spremembe v podatkovnih shemi so se v obliki skript za nadgradnjo prenašale ročno. Najpogostejša težava pri nadgrajevanju sheme je bilo vzdrževanje vsebine šifrantov, zaradi česar je pri nadgradnjah pogosto prišlo do težav v delovanju informacijskega sistema.

3.1.4 Faza vzdrževanja

Po prehodu informacijskega sistema CRP v produkcijo se je začela faza vzdrževanja. Dopolnitve, ki so nastajale v prvih dveh letih vzdrževanja, so bile zaradi številnih sprememb zakonodaje precej obsežne. Vsaka dopolnitev predstavlja nov majhen razvojni cikel z vsemi aktivnostmi, ki so potekale v razvoju, vključno z nadgrajevanjem podatkovne sheme.

Število okolij je bilo sprva zaradi tehničnih razlogov zmanjšano na dve: produkcijsko in skupno razvojno/testno okolje. Nameščanje novih verzij je zato potrebno le ob prenosu v produkcijsko podatkovno shemo, pri čemer se uporablja naslednje pristope:

---

[15] Dodatne programske knjižnice in predloge, s katerimi se pospeši razvoj z Oracle Forms.

- primerjava podatkovnega modela v Oracle Designer repozitoriju in produkcijske sheme ter usklajevanje razlik,
- primerjava produkcijske in testne/razvojne sheme glede usklajenosti shranjenih procedur in
- primerjava vsebine konfiguracijskih tabel in šifrantov.

Pogosto se je sočasno izdelovalo več dopolnitev, zato je skupno testno/razvojno okolje vsebovalo precej novosti, ki niso vse primerne za namestitev v produkcijsko okolje. Seznam, kaj točno je potrebno z novo verzijo namestiti v produkcijsko okolje, je pripravila oseba, ki je izdelala spremembe. Pomanjkljivo definiranje tega seznama se je izkazalo kot pogost vir napak.

S selitvijo razvojnega oddelka na samostojno lokacijo je bila razvojna baza ločena od testne baze, prav tako je bil na novo lokacijo preseljen Oracle Designer repozitorij. Pomembna razlika je, da novo razvojno okolje nima neposredne povezave s testno in produkcijsko bazo, ki se nahaja pri naročniku, tako da možnosti primerjave med razvojnim in testnim ter produkcijskim okoljem ni več[16]. Za nameščanje novih verzij je potrebno uporabiti vnaprej pripravljene sql skripte za nadgradnjo.

Sql skripte za nadgradnjo so označene z enolično oznako, ki izhaja iz oznake zahteve za dopolnitev. Oznaka zahteve je definirana kot letnica zahteve in zaporedna številke zahteve v tem letu, na primer: 2004-1, 2004-2, 2004-3 itd. Vse zahteve niso povezane s spremembo podatkovne sheme, zato označevanje skript za nadgradnjo ni zvezno. Tudi zaporedje, ki ga nakazuje zaporedna številka, se ne upošteva vedno, zato je potrebno vzdrževati evidenco namestitev, ki jo prikazuje tabela 1.

Tabela 1: Evidenca namestitev zahtevkov pri projektu CRP

Zahteva	Datum namestitve v testno okolje	Datum namestitve v produkcijsko okolje
2004-1	3. 2. 2004	19. 2. 2004
2004-2	/	/
2004-3	29. 3. 2004	5. 4. 2004
2004-4	17. 3. 2004	
...		

[16] Obstajajo orodja za izdelavo posnetka stanja podatkovne sheme in kasnejšo primerjavo novega stanja z njim, vendar v času trajanja projekta niso bila na voljo.

3.2 Primer 2: Matični register in novi register stalnega prebivalstva

Matični register (MR) je informacijski sistem v katerem so evidentirani vsi matični dogodki, ki so nastali v Republiki Sloveniji. Gre za dogodke kot so rojstvo, poroka, smrt, sprememba osebnega imena idr. Do 1. maja 2005, ko je začel delovati MR, so bili vsi ti dogodki evidentirani v matičnih knjigah, ki so vodene v papirni obliki. Z začetkom delovanja MR so se vsi novi dogodki vnašali v informacijski sistem, sočasno se je vršil tudi prepis dogodkov iz knjig v MR.

Sočasno z izdelavo matičnega registra se je izvajala tudi posodobitev obstoječega registra stalnega prebivalstva (RSP), kjer so evidentirana vsa stalna in začasna prebivališča državljanov Slovenije.

3.2.1 Razvojno okolje

Pri projektu MRnRSP se je že v samem začetku izbrala uporaba sql skript, okrog katerih je organizirano vodenje konfiguracije in sočasno delo celotne razvojne skupine.

MRnRSP je razvit v trinivojski arhitekturi, kjer je aplikacijski strežnik IBM Websphere, ki se v ozadju povezuje na Oracle SUPB. Razvoj se je delil na java skupino, ki je skrbela za aplikacijski strežnik, in oracle skupino, ki je skrbela za podatkovni model in shranjene procedure. Povezava med aplikacijskim strežnikom in bazo je izdelana preko shranjenih procedur.

Java in oracle skupina sta uporabljali skupno razvojno shemo. Vsa izvorna koda podatkovne sheme je shranjena na datotečnem sistemu v obliki sql skript. Posledica skupne razvojne sheme so številni problemi, povezani z nestabilnostjo razvojnega okolja, ki so jo največ čutili v java skupini.

Kot alternativi skupni razvojni podatkovni shemi sta bili preizkušeni tudi naslednji varianti:

a) *ločena oracle razvojna shema in java podatkovna shema*;

Ideja v ozadju te variante je, da se java razvijalcem zagotovi stabilno razvojno okolje. Kot slabost se je izkazalo vzdrževanje java podatkovne sheme, ki je prineslo preveliko obremenitev oracle razvijalcev in močno upočasnilo sodelovanje med obema skupinama.

b) *vsak oracle razvijalec ima lastno podatkovno shemo, končni rezultat je v skupni integracijski podatkovni shemi*;
Ideja je zagotoviti neodvisna zasebna razvojna okolja, kamor se po potrebi povežejo tudi java razvijalci. V primeru, da java razvijalci potrebujejo stabilno okolje, se povežejo na integracijsko shemo. Kot slabost se je izkazalo vzdrževanje lokalnih podatkovnih shem z novimi verzijami. Prav tako stopnja avtomatizacije ni dopuščala avtomatizirane namestitve integracijske sheme. Zaradi preveč ročnega dela in nedorečenosti v postopku je bil koncept opuščen. Občasno se je ločeno razvojno podatkovno shemo uporabljalo ob spreminjanju skupnih shranjenih procedur in ključnih tabel.

Razvoj je potekal v kratkih iterativno-inkrementalnih ciklih, ki so sprva trajali dva do tri tedne, proti koncu so se skrajšali celo na dober teden. Ob koncu vsakega cikla je sledila namestitev nove verzije v testno podatkovno shemo. Lokalnemu testnemu okolju so se kasneje pridružila še številna druga okolja:

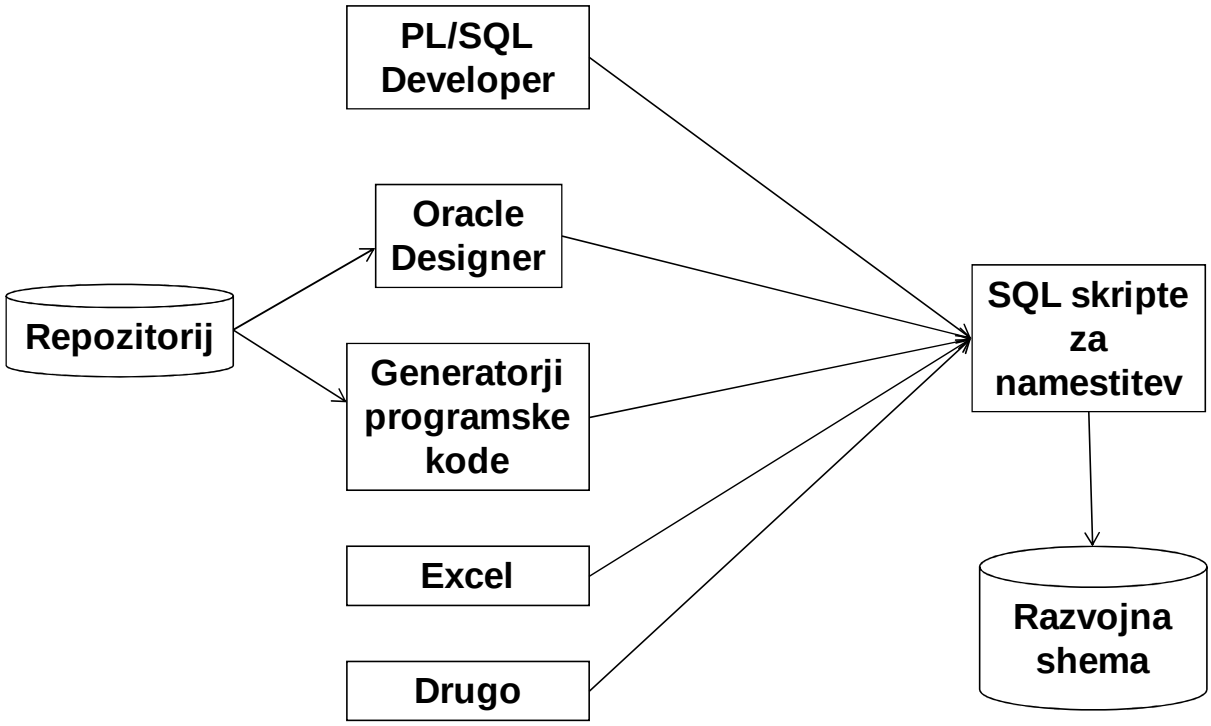
- zunanje testno okolje, do katerega je dostopal naročnik,
- testno okolje pri naročniku, kjer se je testirala strojna konfiguracija in
- uvajalno okolje pri naročniku.

Nameščanje podatkovne sheme je sprva potekalo v obliki namestitev začetne verzije. Po slabih dveh mesecih intenzivnega razvoja se je prešlo na uporabo skript za nadgradnjo.

3.2.2 Sql skripte

Celoten razvoj podatkovne sheme, ki ga prikazuje
slika 8, je usmerjen v sprotno izdelavo sql skript. Način shranjevanja sql skript je zasnovan s ciljem čim bolj preprostega dela z razvojnimi orodji, kjer se za izdelavo in vzdrževanje sql skript porabi čim manj časa.

Slika 8: Postopek izdelave sql skript pri MRnRSP



Število sql skript je med projektom naraslo na nekaj tisoč, zato je potrebna primerna organizacija datotečnega sistema, ki jo prikazuje primer 3.

Primer 3: Struktura direktorijev pri MRnRSP projektu

```
main\
    shema1\
          data
          logic
          syn
          tab
          tapi
          type
          view
    shema2
          mr\
                mr10
                mr11
                mr12
          mn\
          skupno
          syn
    shema3
upgrade
    shema1
    shema2
    shema3
```

MRnRSP je sestavljen iz več podatkovnih shem. Vsaka shema je na datotečnem sistemu zapisana kot istoimenski direktorij. Znotraj vsake sheme je po potrebi nadalje razgrajena drevesna struktura. Uporabljene so naslednje različice poimenovanja:

- tehnično poimenovanje glede na tip strukture, na primer \data, \tab ali \logic;
- vsebinsko poimenovanje po sestavnih delih informacijskega sistema, na primer \sif, \mr ali \mn;
- mešano poimenovanje, kjer sta kombinirana oba omenjena pristopa.

Preprostega pravila glede poimenovanja ni. Izbiro je narekoval predvsem količinski obseg izvorne kode v posameznem direktoriju. Če postane vsebina neobvladljiva, potem se jo smiselno razporedi na poddirektorije.

Poleg drevesne strukture direktorijev je definirana tudi vsebina sql skript. Vsaka skripta vsebuje točno eno strukturo podatkovne sheme oziroma v primeru sekundarnih struktur, kot so indeksi in omejitve, vse indekse oziroma vse omejitve, ki sodijo k isti tabeli. Izjeme so sinonimi in dodeljevanja privilegijev, ki so združena v skupno datoteko. Razlog za tak pristop je v orodju za nameščanje, ki je vsebino sql skripte uganil na podlagi končnice datoteke.

Tabela 2: Končnice datotek sql skript in njihova vsebina

Ime elementa podatkovne sheme	**Datotečna končnica**
Sistemske pravice (angl. *system grant*)	.user
Sinonim (angl. *synonym*)	.syn
Sekvenca (angl. *sequence*)	.sqs, .seq
Tabela (angl. *table*)	.tab
Podatki (angl. *data*)	.dmp, .data
Kontrolne datoteke za SQL*Loader (angl. *SQL*Loader control file*)	.ctl
Pogled (angl. *view*)	.vw
Objektni tip (angl. *type, type body*)	.tps, .tpb
Paket (angl. *package, package body*)	.pks, .pkb .spc, .bdy
Prožilec (angl. *trigger*)	.trg
Procedura (angl. *procedure*)	.prc

Funkcija (angl. *function*)	.fnc
Indeks (angl. *index*)	.ind
Omejitve (angl. *constraint*)	.con
Pravice nad objekti (angl. *grant*)	.grt
Drugo	.runsql

Ime sql skripte je izpeljano iz imena strukture, ki jo vsebuje. SQL ukazi za izdelavo tabele dogodkov so na primer zapisane v sledečih sql skriptah:

- dog.tab (*create table, comment on table, comment on column*),
- dog.con (*alter table, add primary key, unique key, foreign key, check*),
- dog.ind (*create index*),
- dog.sqs (*create sequence*) in
- dog_nongenerated.ind (vsi indeksi, ki so kreirani ročno).

Podobno se iz imena in končnice razbere sledeče shranjene procedure:

- mr10_crud.spc (*package specification* mr10_crud),
- mr10_crud.bdy (*package body* mr10_crud),
- mr10_typ.tps (*type specification* mr10_typ) in
- mr10_typ.tpb (*type body* mr10_typ).

Veliko sql skript je generiranih. Zaradi preglednosti in vzdrževanja sql skript so generirane skripte strogo ločene od ročno izdelanih sql skript.

3.2.3 Namestitev podatkovne sheme

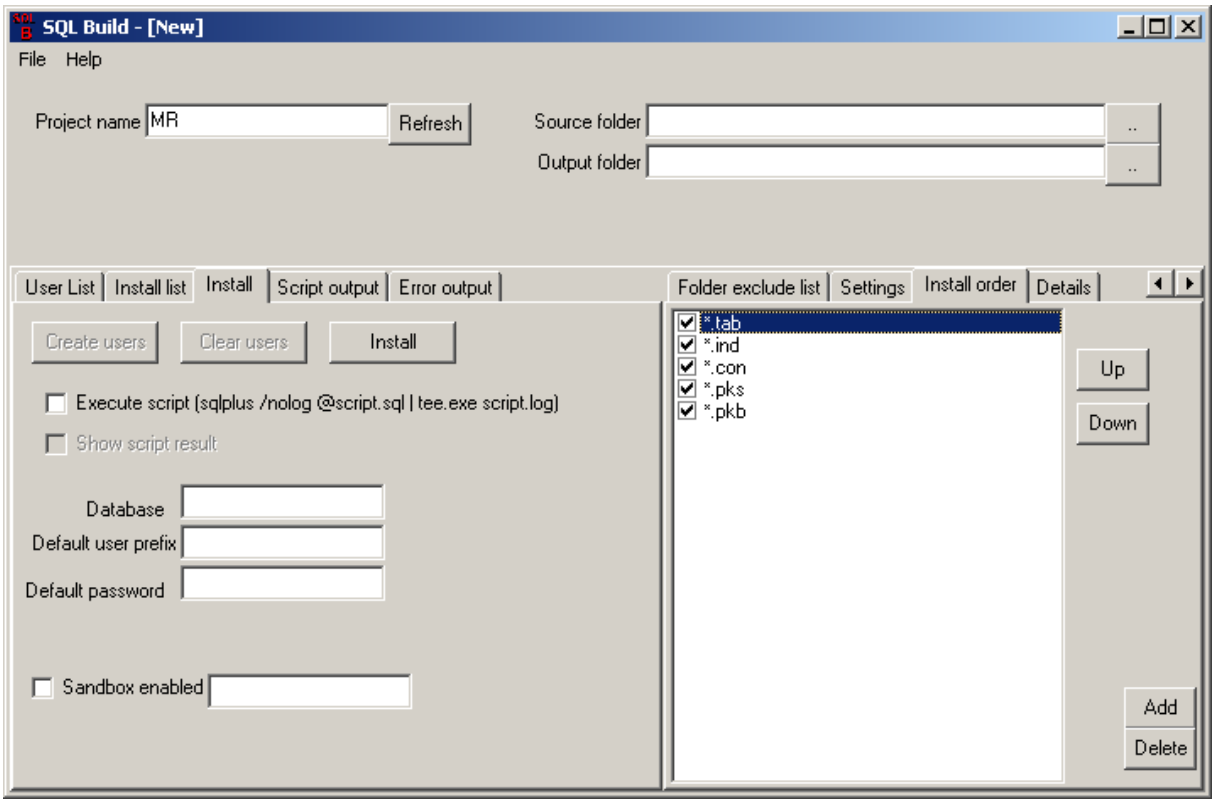
Z izbranim načinom shranjevanja in poimenovanja sql skript je postopek namestitve možno ponazoriti s sledečo psevdo kodo:

```
-- kreiramo sinonime
loop (imeSkripte in »*.syn«) Izvedi imeSkripte
-- kreiramo sekvence
loop (imeSkripte in »*.sqs, *.seq«) Izvedi imeSkripte
-- kreiramo tabele
loop (imeSkripte in »*.tab«) Izvedi imeSkripte
-- vnesemo podatke
loop (imeSkripte in »*.data«) Izvedi imeSkripte
loop (imeCtlDatoteke in »*.ctl«) sqlLoader imeCtlDatoteke
-- kreiramo poglede
loop (imeSkripte in »*.vw«) Izvedi imeSkripte
-- kreiramo definicije shranjenih procedur
loop (imeSkripte in »*.prc, *.fnc, *.pks, *.spc, *.tps«) Izvedi
imeSkripte
```

```
-- kreiramo shranjene procedure
loop (imeSkripte in »*.trg, *.bdy, *.pkb, *.tpb«) Izvedi imeSkripte
-- kreiramo indekse
loop (imeSkripte in »*.ind«) Izvedi imeSkripte
-- kreiramo primarne ključe
loop (imeSkripte in »*.con«) Izvedi kreiranje PK iz imeSkripte
-- kreiramo druge omejitve
loop (imeSkripte in »*.con«) Izvedi vse razen kreiranje PK iz
imeSkripte
-- dodelimo pravice nad objekti drugim shemam
loop (imeSkripte in »*.grt«) Izvedi imeSkripte
-- izvedemo morebitne druge sql skripte
loop (imeSkripte in »*.runsql«) Izvedi imeSkripte
```

Celoten postopek namestitve je implementiran v okviru internega orodja SQL Build, ki ga prikazuje slika 9. Programska podpora postopku nameščanja je močno pripomogla k zmanjšanju nepotrebne administracije pri razvoju podatkovne sheme, saj ni bilo več potrebe po vzdrževanju sql skript, ki so služile namestitvi podatkovne sheme. Dodaten plus je odprava napak pri nameščanju; te so imele vzrok v neažurnih skriptah, ki so služile namestitvi podatkovne sheme.

Slika 9: Vmesnik orodja SQL Build



Pri uporabi orodja SQL Build v praksi se je hitro izkazalo, da imajo nekatere generirane skripte heterogeno vsebino, na primer primarni ključ in tuje ključe v eni skripti, kar ni v skladu s poimenovanjem vsebine skript z oznako končnice datoteke. Ročno spreminjanje vsebine skript je bila nesprejemljiva opcija, zato se je orodje SQL Build dogradilo s kupljeno programsko knjižnico za delo z sql skriptami, s katero se je postopek nameščanja spremenil na naslednjega:

```
Preberi vse sql skripte in jih razbij na posamezne sql ukaze
-- kreiramo sinonime
loop (vsi sql ukazi CREATE SYNONYM) Izvedi sqlUkaz
-- kreiramo sekvence
loop (vsi sql ukazi CREATE SEQUENCE) Izvedi sqlUkaz
-- kreiramo tabele
loop (vsi sql ukazi CREATE TABLE) Izvedi sqlUkaz
-- vnesemo podatke
loop (vsi sql ukazi INSERT) Izvedi sqlUkaz
loop (imeCtlDatoteke in »*.ctl«) sqlLoader imeCtlDatoteke
-- kreiramo poglede
loop (vsi sql ukazi CREATE VIEW) Izvedi sqlUkaz
-- kreiramo definicije shranjenih procedur
loop (vsi sql ukazi CREATE PROCEDURE, FUNCTION, PACKAGE, TYPE) Izvedi
sqlUkaz
-- kreiramo shranjene procedure
loop (vsi sql ukazi CREATE PACKAGE BODYM TYPE BODY, TRIGGER) Izvedi
sqlUkaz
-- kreiramo indekse
loop (vsi sql ukazi CREATE INDEX) Izvedi sqlUkaz
-- kreiramo primarne ključe
loop (vsi sql ukazi ALTER TABLE ADD PRIMARY KEY) Izvedi sqlUkaz
-- kreiramo druge omejitve
loop (vsi sql ukazi ALTER TABLE ADD CONSTRAINT) Izvedi sqlUkaz
-- dodelimo pravice nad objekti drugim shemam
loop (vsi sql ukazi GRANT) Izvedi sqlUkaz
-- izvedemo morebitne druge sql skripte
loop (imeSkripte in »*.runsql«) Izvedi imeSkripte
```

Na ta način dopolnjena funkcionalnost nameščanja se je izkazala za resnično uporabno, ker je minimalno obremenjevala razvojno skupino z dodatno ročno predelavo generiranih sql skript. Edina neprijetna stran je bila počasnost uporabljene knjižnice za delo z sql skriptami, ki pa ni bila kritična.

Ključna pomanjkljivost orodja SQL Build je nezmožnost avtomatizacije postopka nameščanja in omejena funkcionalnost brez podpore nadgradnje. To področje je dopolnjeno v rezultatu magistrskega dela, v orodju SQL Make.

3.2.4 Nadgradnja podatkovne sheme

SQL ukazi za nadgradnjo podatkovne sheme se vzdržujejo kot komentarji v sql skriptah za namestitev. Z vsako spremembo podatkovnega modela se ažurira SQL ukaz za

izdelavo tabele in kot komentar navede SQL ukaze za nadgradnjo tabele. SQL ukaze, ki so potrebni za nadgradnjo verzije N na verzijo N+1, se dobi na podlagi razlik med skriptami verzije N in verzije N+1. Uporablja se orodje Winmerge[17].

Primer 4: Zapis SQL ukazov za nadgradnjo podatkovne sheme pri MRnRSP

```
/*
-- MG, 1.1.2009
alter table zaposleni add(
  spol char(1),
  zakonskiStan char(1)
)
*/

create table zaposleni (
  Id           number(6),
  Ime          varchar2(35),
  Placa        number(4),
  spol         char(1),
  zakonskiStan char(1)
)
```

Opisani postopek nadgradnje ni avtomatiziran. Za njegovo delovanje je potrebna evidenca verzij nameščene podatkovne sheme, ki je zapisana v konfiguracijsko tabelo v shemi.

S skriptami za nadgradnjo se izvaja nadgradnja podatkovnega modela in podatkov, medtem ko se nadgradnja shranjenih procedur izvaja ločeno na dva možna načina. Prva varianta se izvaja v naslednjih korakih:

- brisanje vseh obstoječih shranjenih procedur v podatkovni shemi,
- kreiranje shranjenih procedur iz sql skript trenutne verzije in
- prevajanje neprevedenih shranjenih procedur.

Druga varianta se izvaja v naslednjih korakih:

- primerjava sql skript shranjenih procedur nove in stare verzije z orodjem Winmerge,
- ročno brisanje shranjenih procedur, ki ne obstajajo v novi verziji,
- izvajanje vseh sql skript s shranjenimi procedurami, ki so se spremenile oziroma so nove in
- prevajanje neprevedenih shranjenih procedur.

---

[17] http://winmerge.org

Druga varianta je navadno časovno krajša, ker se namesti le nove in spremenjene shranjene procedure, vendar ni avtomatizirana, zato se je v praksi redko uporabljala.

3.2.5 Statistika razvoja podatkovne sheme

Uporaba sistema za vodenje konfiguracije nudi med drugim tudi odlično izhodišče za izdelavo različnih statistik. S pomočjo orodja StatSVN so spodaj predstavljene skupne statistike sprememb nad sql skriptami za obe osrednji podatkovni shemi v informacijskem sistemu MR.

Statistika je izdelana na podlagi StatSVN poročila o velikosti in spremembah direktorijev (angl. *Directory sizes*) za obdobje od avgusta 2007 do januarja 2009. Pridobljene statistike po posameznih direktorijih so na podlagi imena direktorija ročno razdeljene na eno izmed štirih kategorij: podatkovni model, shranjene procedure, skripte za nadgradnjo in podatki. Statistika posameznih direktorijev je tako agregirana v statistiko kategorije in predstavljena v tabelarični obliki.

Skupna statistika za dve shemi je izbrana zaradi arhitekturne zasnove informacijskega sistema MRnRSP: vse tabele in interne shranjene procedure so v eni shemi, medtem ko so vse shranjene procedure, s katerimi dela aplikacija, v ločeni shemi.

Tabela 3: Statistika sprememb razvoja podatkovne sheme MRnRSP v fazi vzdrževanja

	Število sprememb	**Odstotek sprememb**
Podatkovni model	597	12,5 %
Shranjene procedure	3922	82,1 %
Skripte za nadgradnjo	ni podatka	ni podatka
Skripte s podatki	257	5,4 %
Skupaj sprememb	4776	

Zelo izstopajoča statistika je število sprememb shranjenih procedur, kar je možno argumentirati na dva načina:

- statistike so izdelane za obdobje vzdrževanja informacijskega sistema, ko je podatkovni model že zelo stabilen in na njem ni veliko sprememb;
- arhitektura informacijskega sistema je zasnovana tako, da je glavnina poslovne logike v shranjenih procedurah, kar dodatno doprinese k količini sprememb shranjenih procedur.

Ne glede na razlog za takšen delež sprememb pri shranjenih procedurah je potrebno postopek razvoja shranjenih procedur dobro premisliti in poiskati skrite rezerve za optimizacijo dela.

3.3 Primer 3: Unified Colection Gateway

Unified Collection Gateway (UCG) je centralni sistem registracije, obračunavanja in plačevanja davkov in prispevkov. Projekt je v fazi implementacije.

3.3.1 Razvojno okolje

UCG sistem ima večnivojsko arhitekturo, ki na eni strani vključuje podporo dokumentnemu delu in na drugi strani transakcijskemu delu z relacijsko bazo. Glavni del aplikacije je izdelan v .NET okolju, ki se povezuje z relacijsko bazo Oracle. Iz te delitve izhaja delitev razvijalcev na NET skupino in oracle skupino.

NET in oracle skupina uporabljata skupno razvojno shemo. Vsa izvorna koda podatkovne sheme je shranjena v sistemu za vodenje konfiguracije v obliki sql skript.

3.3.2 Sql skripte

Način organizacije dela z sql skriptami je zasnovan z naslednjimi omejitvami:

- čim manjše število sql skript in
- namestitev podatkovnega modela namesti tabele, omejitve in indekse v ločenih korakih.

Za izvajanje nameščanja podatkovne sheme se uporablja statične sql skripte, ki so ob preveč dinamični strukturi in vsebini zahtevne za vzdrževanje. Zaradi tega je struktura direktorijev preprosta, prav tako je fiksno določeno poimenovanje in vsebina sql skript podatkovnega modela.

Nekaj več svobode je dopuščene pri shranjenih procedurah, ki so zapisane vsaka v svoji sql skripti. Vzdrževanje sql skript za nameščanje shranjenih procedur je izdelano s pomočjo lupinske skripte (angl. *shell script*).

Uporablja se struktura direktorijev, kot jo prikazuje primer 5. Vsaka podatkovna shema se nahaja v svojem direktoriju, ki vsebuje naslednje poddirektorije:

- podatkovni model (data),
- shranjene procedure (plsql),
- podatke (data) in

- skripte za nadgradnjo (upgrade).

Sql skripte za izdelavo uporabnikov, prostorov za tabele in drugih sistemskih opravil se nahajajo v direktoriju dba.

Primer 5: Struktura direktorijev pri UCG projektu

```
db\
   shema1\
         data
         datamodel
         plsql
         upgrade
   shema2
         data
         datamodel
         plsql
         upgrade
   dba
```

Tipi razvojnih orodij za izdelavo sql skript, ki jih prikazuje, so podobni kot pri projektu MR:

- orodje za podatkovno modeliranje,
- orodje za PLSQL razvoj in
- Excel ter drugi pripomočki za pripravo testnih podatkov.

Zaradi slabe implementacije generatorja sql skript v Enterprise Architect je potrebna dodatna obdelava generiranih skript. Prav tako je dodaten korak vzdrževanje sql skript za nameščanje shranjenih procedur.

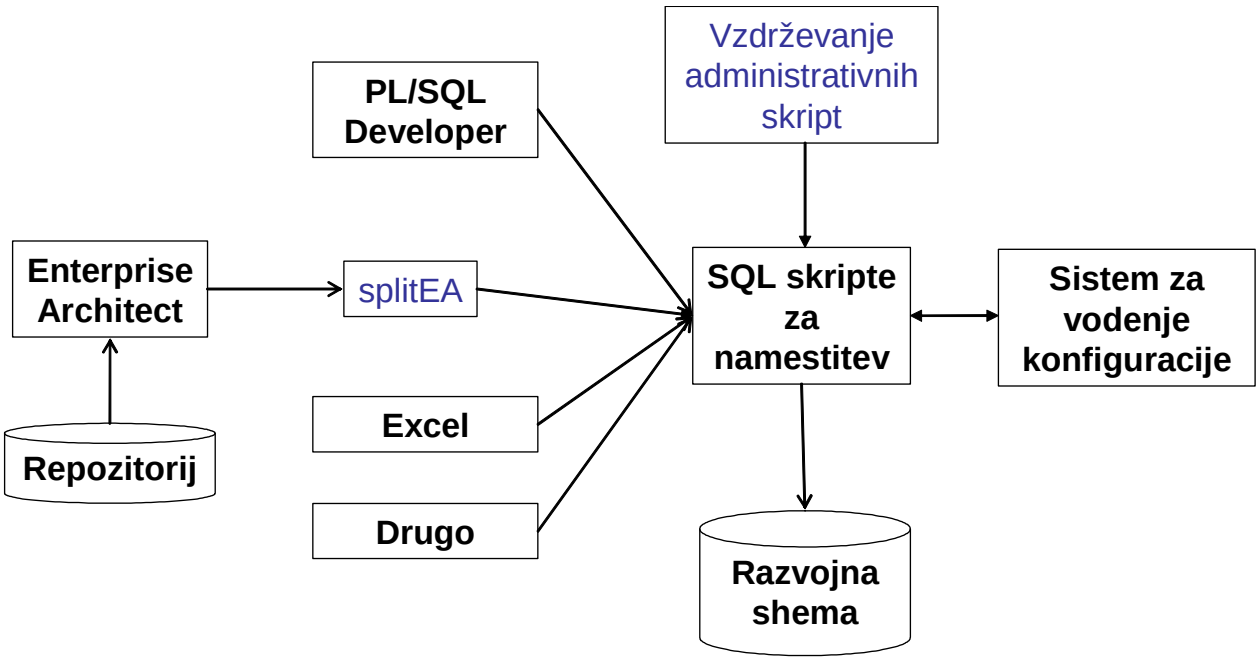
Naknadna obdelava generiranih sql skript

Enterprise Architect generira celoten model v eno samo veliko sql skripto. Ena izmed omejitev, ki je bila podana na začetku razvoja, je ločeno shranjevanje ukazov za kreiranje tabel, indeksov in omejitev.

Dodaten problem, ki se zasledi tudi pri drugih generatorjih, je nedoslednost pri upoštevanju abecednega vrstnega reda pri nizanju sql ukazov v sql skriptah. V praksi je vrstni red zelo pomemben zaradi možnosti primerjave dveh različnih verzij sql skript. Če je v skripti sto in več tabel, ki so urejene naključno, primerjava dveh sql skript ni možna, ker se isti sql stavek enkrat nahaja na začetku in drugič na koncu skripte.

Oba navedena problema sta rešena s programom *splitEA.awk*, ki razbije veliko sql skripto na tri manjše, za ločeno kreiranje tabele, indeksov in omejitev. Pri tem razvrsti sql ukaze v abecednem vrstnem redu glede na ime tabele. Na projektu MRnRSP je ta problem rešen tako, da je vsaka tabela zapisana v svoji skripti.

Slika 10: Postopek izdelave sql skript pri UCG



Vzdrževanje namestitvenih sql skript

Sql skripte s shranjenimi procedurami je zaradi narave dela z razvojnim orodjem potrebno shranjevati vsako v svojo datoteko. Ime datoteke je enako kar imenu shranjene procedure, medtem ko končnica odraža, ali gre za proceduro, funkcijo, paket, tip idr.

Ročno vzdrževanje statične sql skripte, ki služi nameščanju shranjenih procedur, je nadležno opravilo, ki se ga zlahka avtomatizira. Izdelan je preprost program, ki se rekurzivno sprehodi po direktoriju *db\shema1\plsql* in izdela sql skripto *install_sp.sql*. Pri tem sledi naslednjemu vrstnemu redu, ki upošteva medsebojne odvisnosti med posameznimi tipi shranjenih procedur:

- package specification (*.pks),
- type specification (*.tps),
- procedure (*.prc),
- funkcije (*.fnc),
- package body (*.pkb),
- type body (*.tpb) in
- prožilci (*.trg).

Primer generiranega klica:

```
PROMPT
PROMPT Creating package specification testni_paket...
PROMPT=========================================================
@@testni_paket.pks
SHOW ERRORS
```

3.3.3 Namestitev podatkovne sheme

Namestitev podatkovne sheme je zasnovana na podlagi naslednjih omejitev:

- *Enostavnost*; enostavnost postopka nameščanja je vedno dobrodošla, ker nameščanje pogosto izvajajo osebe, ki niso strokovnjaki za izbrano področje. Kot primer vzdrževanje testne podatkovne sheme je prepuščeno testni skupini.
- *Avtomatizacija*; avtomatizacija izhaja iz uporabe sistema neprestane gradnje, ki se v praksi največkrat uporablja pri izdelavi aplikacije. Ker je tudi podatkovna shema predmet številnih sprememb, jo je potrebno vključiti v proces neprestane gradnje.

Izvajanje namestitve je avtomatizirano in se začne z naslednjim ukazom:

```
sqlplus /nolog @setup.sql DEFAULT_PROFILE FULL
```

Od tu dalje poteka celotna namestitev avtomatizirano. *Setup.sql* v ozadju kliče ustrezne sql skripte. Ob koncu se izpiše, ali je namestitev uspešno zaključena. Morebitne probleme se diagnosticira iz datoteke z dnevnikom namestitve.

3.3.4 Nadgradnja podatkovne sheme

Uporabljeno je poimenovanje sql skript za nadgradnjo z zaporedno številko. Skripte so običajno majhne in odražajo posamezno nalogo, kot je na primer nov stolpec. V posamezni gradnji aplikacije se lahko nahaja ena ali več novih skript za nadgradnjo podatkovne sheme.

Primer 6: Poimenovanje skript za nadgradnjo pri UCG projektu

```
10_newTableXYZ.sql
11_PkRenameForFewTables.sql
12_newColumnABC.sql
```

Izdelava skript za nadgradnjo pri UCG projektu ni podprta v razvojnem orodju Enterprise Architect. Slednji izdela le skripte za namestitev. Skripte za nadgradnjo je potrebno izdelati ročno oziroma s primerjavo dveh shem (angl. *schema diff*) po naslednjem postopku:

- namesti podatkovno shemo izhodiščne verzije,
- namesti novo podatkovno shemo ciljne verzije in

- uporabi primerjavo dveh shem, ki izdela skripto za nadgradnjo.

Pri nadgradnji podatkovne sheme na novo verzijo je potrebno namestiti vse skripte za nadgradnjo z večjo zaporedno številko od trenutne verzije podatkovne sheme. Postopek nadgradnje je v celoti avtomatiziran s pomočjo sql skript in se izvede s sledečim ukazom:

```
sqlplus /nolog @setup.sql DEFAULT_PROFILE UPGRADE
```

Ločeno se v naslednjih korakih izvaja postopek nadgradnje shranjenih procedur:

- brisanje vseh obstoječih shranjenih procedur v podatkovni shemi,
- kreiranje shranjenih procedur iz sql skript trenutne verzije in
- prevajanje neprevedenih shranjenih procedur.

Tudi ta postopek je v celoti avtomatiziran na naslednji način:

```
sqlplus /nolog @setup.sql DEFAULT_PROFILE PLSQL
```

Uporaba avtomatiziranih postopkov nameščanja in nadgrajevanja se je izkazala za zelo uspešno, ker so prenekatera opravila, ki jih je običajno izvajal DBA oziroma oracle razvojna skupina, začele izvajati druge osebe:

- testno okolje z ločeno testno podatkovno shemo je z novimi verzijami v celoti vzdrževala testna skupina;
- vodja konfiguracije je samostojno izvajal namestitev ter nadgradnjo pri stranki;
- vodja konfiguracije je samostojno pripravljal nova razvojna okolja za potrebe ločenih vej razvoja, s preprosto spremembo konfiguracije preko tako imenovanih profilov.

Pri tem je potrebno poudariti, da gre za osebe, ki imajo zgolj začetniško znanje jezika SQL in administracije SUPB.

3.3.5 Statistika razvoja podatkovne sheme

Podobno kot pri prejšnjem primeru je izdelana statistika sprememb sql skript s pomočjo StatSVN. Statistika je izdelana za glavno vejo razvoja za celotno obdobje od začetka razvoja informacijskega sistema v decembru 2007 do januarja 2009.

Tabela 4: Statistika sprememb razvoja podatkovne sheme UCG

Shema 1	**Število sprememb**	**Odstotek sprememb**
Podatkovni model	454	18,6 %
Shranjene procedure	1560	64 %
Skripte za nadgradnjo	203	8,3 %
Skripte s podatki	220	9 %
Skupaj sprememb	2437	

Shema 2	**Število sprememb**	**Odstotek sprememb**
Podatkovni model	203	18 %
Shranjene procedure	504	44,8 %
Skripte za nadgradnjo	176	15,6 %
Skripte s podatki	243	21,6 %
Skupaj sprememb	1126	

Izstopajoča statistika je v obeh primerih število sprememb shranjenih procedur, ki je v enem primeru 45% in v drugem primeru 64% vseh sprememb.

3.4 Povzetek

Predstavljeni so trije primeri iz prakse in njihov način organizacije razvoja podatkovne sheme. Prvi primer (CRP) predstavlja klasičen pristop k razvoju podatkovne sheme, ki se odvija v orodju za podatkovno modeliranje in skupni razvojni podatkovni shemi. Hkrati je pri tem primeru že možno zaslediti uporabo nekaterih opisanih agilnih pristopov v fazi vzdrževanja, ki izhajajo predvsem iz potrebe po natančnem vodenju konfiguracije in veliko krajših razvojnih ciklih v fazi vzdrževanja.

Drugi (MRnRSP) in tretji primer (UCG) organizacijo razvoja podatkovne sheme zasnujeta na osnovi sql skript. Vsak na svoj način rešujeta probleme pri uporabi sql skript, ker ni na voljo ustreznega orodja, ki bi ustrezno podprl njihovo nameščanje in nadgrajevanje. Skupna razvojna podatkovna shema je še vedno prisotna, kljub temu, da so pri primeru MRnRSP poskusi uporabe zasebnih razvojnih okolij, ki zaradi težav pri njihovem vzdrževanju niso zaživeli.

V primeru UCG je implementirana avtomatizacija postopka nameščanja in nadgradnje, ki se izkaže za zelo koristno, ker administratorja podatkovne baze razbremeni odvečnega operativnega dela ter hkrati izboljša zanesljivost skript za nadgradnjo.

Drugi in tretji primer zaradi uporabe sistema za vodenje konfiguracijo omogočata enostavno izdelavo statistike sprememb na sql skriptah. Pričakovano največji delež sprememb odpade na shranjene procedure, ki je v primeru MRnRSP celo 82,1 %.

4 Namestitev podatkovne sheme

V tem poglavju je opisan postopek namestitve podatkovne sheme. Opis ne sledi konkretnim orodjem, temveč nalogam in problemom, ki jih mora reševati postopek nameščanja in nadgrajevanja.

V prvem delu sta navedeni definiciji podatkovne sheme in sql skript. Sledi predstavitev življenjskega kroga podatkovne sheme, kjer je opisana razlika med postopkom začetne namestitve in postopkom nadgrajevanja podatkovne sheme. V osrednjem delu sta podrobno predstavljena postopek namestitve in postopek nadgrajevanja podatkovne sheme. Ločeno je opisana namestitev in nadgradnja shranjenih procedur. Ob koncu so opisane podporne aktivnosti, kot so vodenje verzije podatkovne sheme, ugotavljanje statusa podatkovne sheme, pristopi k obravnavanju napak pri izvajanju sql skript in povrnitev začetnega stanja ob neuspeli namestitvi oziroma nadgradnji.

4.1 Definicije

Podatkovna shema je termin, ki je prevzet iz ANSI SQL standarda [SQL03] in predstavlja imenski prostor (angl. *namespace*), znotraj katerega so definirane strukture SUPB. V Oracle SUPB podatkovna shema sovpada z uporabniškim imenom, ki se uporablja pri dostopu do SUPB in služi kot izhodišče za identifikacijo, avtentikacijo in avtorizacijo.

V podatkovni shemi se lahko kreira strukture, kot so tabele, indeksi, omejitve in shranjene procedure, v tabele zapisuje podatke, drugim uporabnikom dodeljuje privilegije in v ozadju z ustrezno fizično organizacijo skrbi za optimalno delovanje.

Sql skripta je tekstovna datoteka, v kateri je zapisano zaporedje SQL ukazov. Za izvajanje skripte je potreben ustrezen program, kot je na primer SQL*Plus, ki se poveže na SUPB in mu zaporedoma pošilja SQL ukaze iz skripte. Postopek izvajanja sql skript je preprost, ker se sintaktično preverjanje SQL ukazov, njihovo izvajanje in preverjanje vseh odvisnosti izvaja znotraj SUPB.

Programi za izvajanje skript omogočajo uporabo dodatnih ukazov SQL*Plus, ki jih ne posredujejo SUPB. Gre za preproste ukaze za kontrolo izgleda, preusmerjanje izpisa v datoteko, izpis sporočil, delo s spremenljivkami, prijava v SUPB in podobno.

Primer 7: Primer sql skripte

```
PROMPT Kreiram tabelo zaposleni ...
create table zaposleni (
  Id        number(6),
  Ime       varchar2(35),
  Placa     number(4)
);

PROMPT Kreiram primarni ključ na tabeli zaposleni ...
alter table zaposleni add constraint zaposleni_pk
primary key (id);

PROMPT Kreiram indeks zaposleni_ime_i na tabeli zaposleni ...
create index zaposleni_ime_i on zaposleni (ime asc);
```

Primer 8: Rezultat izvajanja sql skripte

```
SQL> @"Primer 7.sql"
Kreiram tabelo zaposleni ...

Table created.

Kreiram primarni ključ na tabeli zaposleni ...

Table altered.

Kreiram indeks zaposleni_ime_i na tabeli zaposleni ...

Index created.
```

4.2 Življenjski krog podatkovne sheme

Življenjski krog podatkovne sheme se vedno začne z začetno namestitvijo, znotraj katere se kreira tabele, indekse, shranjene procedure, dodeli privilegije, vnese začetne podatke idr. Vsaka nadaljnja sprememba podatkovne sheme se izvaja v obliki inkrementalne nadgradnje s postopki evolucije podatkovne sheme.

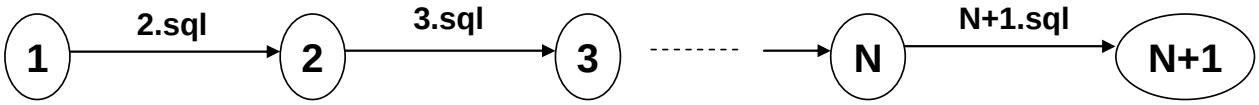
Življenjski krog podatkovne sheme ima lahko različne oblike, dve tipični sta opisani v nadaljevanju:

- enostaven življenjski krog in
- kompleksen življenjski krog.

4.2.1 Enostaven življenjski krog

Najbolj preprost življenjski krog podatkovne sheme je ponazorjen z acikličnim usmerjenim grafom, kjer vozlišča predstavljajo verzijo podatkovne sheme in povezave med vozlišči skripte za nadgradnjo. Vse skripte za nadgradnjo si sledijo v zaporedju.

Slika 11: Enostaven življenjski krog podatkovne sheme



Skripte za nadgradnjo so pripravljene in testirane za nadgrajevanje točno določene verzije podatkovne sheme. V praksi se zaradi enostavnosti in sledljivosti zaporednost izvajanja skript vedno upošteva, čeprav se lahko z upoštevanjem sledečih odvisnosti vrstni red tudi nekoliko priredi:

1) *Strukturna odvisnost.* Nekatere skripte se sklicujejo na že obstoječe strukture podatkovne sheme, ki so kreirane v drugi skripti. Kot primer: če je nov stolpec kreiran v skripti A in indeks nad tem stolpcem v skripti B, potem je s tem ustvarjena odvisnost med obema skriptama. Skripto A je potrebno izvesti pred skripto B.
2) *Programska odvisnost.* Zamenjava vrstnega reda izvajanja skript lahko povzroči različno strukturo tabel, kot prikazuje primer 9. V primeru, da so SQL stavki v programski kodi eksplicitno vezani na zaporedje stolpcev v tabeli, lahko to privede do nedelovanja ali še slabše, do nepravilnega delovanja informacijskega sistema.
3) *Podatkovna odvisnost.* Zadnja odvisnost med skriptami nastane pri uporabi DML ukazov za spreminjanje podatkov. Kot primer: v skripti A se spremeni naziv ulice iz Titova cesta v Dunajska cesta in v skripti B nato naziv iste ulice v Slovenska cesta. Končni rezultat je, da se ulica imenuje Slovenska cesta. V primeru, da se najprej izvede skripta B in šele nato skripta A, se ulica napačno imenuje Dunajska cesta.

Neupoštevanje zaporedja izvajanja ima lahko nepredvidljive posledice, zato je potrebno vsako odstopanje pred tem testirati.

Primer 9: Vpliv vrstnega reda izvajanja evolucije podatkovne sheme na končno strukturo tabele

Začetna struktura relacije RACUNI

```
Id          number(6),
ImeKupca    varchar2(70)
```

```
create table racuni (
  Id        number(6),
  ImeKupca  varchar2(35)
)
```

Prva pot nadgradnje

Najprej se doda DatumRacuna:

```
alter table racuni
add (DatumRacuna date)
```

Nato se doda Znesek:

```
alter table racuni
add (Znesek number(10,2))
```

Končna struktura relacije RACUNI:

```
Id            number(6),
ImeKupca      varchar2(70),
DatumRacuna   date,
Znesek        number(10,2)
```

Druga pot nadgradnje

Najprej se doda Znesek:

```
alter table racuni
add (Znesek number(10,2))
```

Nato se doda DatumRacuna:

```
alter table racuni
add (DatumRacuna date)
```

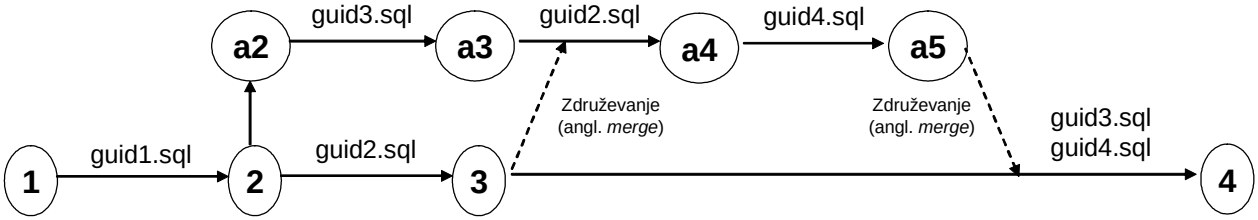
Končna struktura relacije RACUNI:

```
Id            number(6),
ImeKupca      varchar2(70),
Znesek        number(10,2),
DatumRacuna   date
```

4.2.2 Kompleksen življenjski krog

Slika 12 predstavlja kompleksen življenjski krog podatkovne sheme, kjer je kreirana vzporedna veje razvoja. Primeri takega življenjskega kroga nastopijo pri vzdrževanju obstoječih produktov, kjer se vzdrževanje produkta izvaja na stranski veji in razvoj naslednje verzije produkta na glavni veji razvoja.

Slika 12: Kompleksen življenjski krog podatkovne sheme



Primer, ki ga prikazuje slika 12, je sestavljen iz naslednjih korakov:

- začetna verzija podatkovne sheme je 1
- guid1.sql nadgradi podatkovno shemo na verzijo 2

- iz verzije 2 se kreira vzporedna razvojna veja, ki ima oznako a2
- stranska veja
  - guid3.sql nadgradi podatkovno shemo na verzijo a3
  - z združevanjem se iz glavne veje na vzporedno vejo prenese guid2.sql, ki nadgradi podatkovno shemo na verzijo a4
  - guid4.sql nadgradi podatkovno shemo na verzijo a5
- glavna veja
  - guid2.sql nadgradi verzijo podatkovne sheme na 3
  - z združevanjem se iz stranske veje na glavno vejo prenese skripti guid3.sql in guid4.sql, ki nadgradita podatkovno shemo na verzijo 4

Funkcionalnost sistema za vodenje konfiguracije, ki se uporablja pri prikazanem primeru, je združevanje (angl. *merge*). Z združevanjem se prenaša spremembe, ki so izdelane na eni veji razvoja, v drugo vejo razvoja. Postopek združevanja je avtomatski, razen v primeru konfliktne situacije, ki jo je potrebno razrešiti ročno.

Uporaba združevanja pri skriptah za nadgradnjo zahteva dodatno pazljivost pri:

- *poimenovanju skript za nadgradnjo*: zagotoviti je potrebno enolično poimenovanje skript za nadgradnjo, sicer se lahko z združevanjem združi dve skripti s povsem različno vsebino. V primeru, ki ga prikazuje slika 12, so skripte za nadgradnjo poimenovane z enoličnim globalnim identifikatorjem GUID.
- *podvajanju SQL ukazov*: ista funkcionalnost se lahko neodvisno implementira na glavni in stranski veji razvoja, kar lahko privede do napak pri izvajanju skript za nadgradnjo. Kot primer: skripti guid2.sql in guid3.sql obe vsebujeta SQL ukaz za kreiranje iste tabele. Avtomatski postopek za združevanje teh problemov ne zazna;
- *strukturni odvisnosti*: z združevanjem se lahko iz ene veje na drugo vejo prenese le del razlik. Potrebno je paziti, da se skripte za nadgradnjo, med katerimi obstajajo strukturne odvisnosti, prenese sočasno.
- *programski odvisnosti*: z uporabo združevanja se zelo pogosto srečujemo s problemom programske odvisnosti, kot ga prikazuje primer 9. Potrebna je dodatna pazljivost pri združevanju programske kode in defenzivno programiranje, ki omejuje vpliv programskih odvisnosti.

- *podatkovni odvisnosti*: pri vzporednih vejah razvoja je povečana možnost spreminjanja istih podatkov. Avtomatski postopek združevanja ne zazna konfliktov pri ažuriranju istih tabel oziroma podatkov.

4.3 Začetna namestitev podatkovne sheme

Začetna namestitev podatkovne sheme se uporablja ob prvi namestitvi informacijskega sistema v novo okolje. V okviru začetne namestitve se namesti in konfigurira SUPB, kreira sheme s tabelami, indeksi, omejitvami, shranjenimi procedurami, dodeli privilegije, vnese začetne podatke idr. Migracija podatkov ne sodi k začetni namestitvi, ampak se izvaja kot samostojna aktivnost.

Kot ugotavlja Hall, je začetna namestitev pogosto najkompleksnejša aktivnost med aktivnostmi postavitve (angl. *deployment*) in hkrati tudi najbolj podprta s strani različnih specializiranih orodij [Hall]. Kompleksnost izhaja iz zahtev po vzpostavitvi in konfiguraciji vseh potrebnih virov za izvedbo namestitve.

Postopek začetne namestitve podatkovne sheme se razdeli na naslednje korake:

- namestitev SUPB,
- priprava okolja v SUPB,
- kreiranje lastnika podatkovne sheme in
- namestitev struktur podatkovne sheme.

Aktivnosti namestitve in konfiguracije SUPB prinašata glavnino kompleksnosti začetne namestitve podatkovne sheme, ki jo omenja Hall.

4.3.1 Namestitev SUPB

Namestitev sistema za upravljanje podatkovnih baz (SUPB) je v primeru produkcijskega okolja lahko zelo kompleksno opravilo. V produkcijskem okolju se uporablja namensko strežniško opremo, diskovna polja, tračne enote za izdelavo varnostnih kopij, gruče strežnikov (angl. cluster). Namestitev in konfiguracije naštetih elementov sistemskega in SUPB okolja je zelo zahtevno opravilo, ki ga izvajajo izkušeni sistemski administratorji in administratorji SUPB (angl. *DataBase Administrator*, DBA).

4.3.2 Priprava okolja v SUPB

Kljub občutnemu napredku tehnologije pomnilniških medijev so ti še vedno ozko grlo pri delu s podatkovnimi bazami. Za dobro delovanje produkcijskega sistema je potrebno

usklajeno delovanje arhitekta informacijskega sistema in administratorja podatkovne baze, ki se dogovorita o potrebnih pomnilniških in diskovnih virih.

Del tega dogovora je tudi definiranje prostora za tabele[18] (angl. *tablespace*). Prostor za tabele kreira administrator SUPB in predstavlja nivo abstrakcije med diskovnim sistemom in SUPB. Primer prostora za tabele je na primer prostor za arhivske tabele, ki ga administrator kreira na starih in počasnih diskih.

Pri definiranju prostorov za tabele obstajajo različne strategije, ki izvirajo iz prakse. Kot primer strategije je ločen prostor za indekse in ločen prostor za tabele, delitev glede na module informacijskega sistema ali tehnično glede na velikosti tabel in indeksov.

Primer 10: SQL stavek za izdelavo tabele z opredeljenimi fizičnimi lastnosti

```
create table rac_racuni (
  rac_id number,
  rac_dtm date,
  rac_znesek number
)
tablespace velike_tabele
storage (initial 10m next 1m pctincrease 0);
```

Kot prikazuje primer 10, je sklicevanje na prostor za tabele sestavni del SQL ukaza za kreiranje tabele, zato je kreiranje prostora za tabele potreben korak pred izvajanjem namestitve struktur podatkovne sheme.

4.3.3 Kreiranje lastnika podatkovne sheme

V pripravljenem okolju administrator SUPB kreira uporabniško ime, preko katerega se dostopa do podatkovne sheme. Kreiranemu uporabniku se dodeli zahtevane sistemske pravice, ki so potrebne za namestitev in kasnejšo uporabo podatkovne sheme. V primeru uporabe podatkovne integracije se uporabniku dodeli dostop do zahtevanih tabel v drugih shemah.

4.3.4 Namestitev struktur podatkovne sheme

V pripravljeno podatkovno shemo se namesti strukture podatkovne sheme. Namestitev je sestavljena iz izvajanja ene ali več sql skript v zaporedju, ki je predpisano v navodilih za namestitev. Ob uspešnem zaključku so v podatkovno shemo nameščene vse tabele, indeksi, shranjene procedure in začetni podatki, potrebni za delovanje informacijskega sistema.

---

[18] Ni del ANSI SQL standarda, temveč del Oracle SUPB

Število sql skript, ki se uporabljajo za namestitev, je lahko zelo različno. V praksi se zasledi naslednje pristope:

- ena sama velika skripta za namestitev;
- več manjših skript za namestitev, kjer so skupaj zbrane istovrstne strukture, kot so na primer tabele, indeksi in shranjene procedure;
- več manjših skript za namestitev, kjer so skupaj zbrane vsebinsko povezane strukture;
- vsaka struktura je zapisana v svoji skripti, kjer je ime skripte enako imenu strukture in končnica skripte enaka imenu tipa strukture. Prednost te oblike je, da se že na prvi pogled identificira vsebina sql skripte. Kot primer: shranjena procedura z imenom test je zapisana v datoteki test.prc;
- kombinacija zgoraj navedenih pristopov, na primer ena skripta za celoten podatkovni model in posamične skripte za vsako izmed shranjenih procedur.

V primeru, da je namestitev sestavljena iz več kot ene skripte, je njihovo zaporedje izvajanja najpogosteje definirano preko krovne sql skripte, kot jo prikazuje
primer 11.

Primer 11: Sql skripta za namestitev struktur podatkovne sheme

```
PROMPT Kreiram tabele ...
@10_tables.sql
PROMPT Kreiram indekse ...
@20_indexes.sql
PROMPT Kreiram primarne ključe, tuje ključe in druge omejitve ...
@30_constraints.sql
PROMPT Kreiram sekvence ...
@40_sequences.sql
```

Zaradi medsebojne odvisnosti struktur podatkovne sheme je potrebno namestitev izvajati v pravilnem vrstnem redu. Kot primer: indeks nad neko tabelo se lahko kreira šele zatem, ko tabela že obstaja. V nasprotnem primeru indeks ne bo kreiran, ker SUPB javi napako, da tabela ne obstaja.

Primer 11 prikazuje najbolj pogosto zaporedje namestitve, s katerim zadostimo navedenim odvisnostim.

Dobra praksa pri izvajanju namestitve je pisanje dnevnika namestitve (angl. *log*), ki služi dokumentiranju poteka in kasnejšemu diagnosticiranju napak. Za izboljšanje diagnostike je priporočljivo sql skripte opremiti z dodatnimi komentarji.

4.3.5 Ponovitev začetne namestitve

V razvojnih in testnih okoljih so pogoste zahteve po namestitvi nove verzije podatkovne sheme. V začetnih fazah razvoja je zaradi velikih sprememb v podatkovnem modelu lažje vzdrževati skripte za namestitev kot izdelati skripto za nadgradnjo. Namestitev nove verzije se izvede na naslednji način:

- brisanje obstoječe podatkovne sheme in
- ponovitev namestitve sheme z novo verzijo skripte za namestitev.

Podatki, ki so že obstajali v podatkovni shemi, so z brisanjem izgubljeni.

4.4 Nadgradnja podatkovnega modela

Nadgradnja podatkovnega modela je postopek, pri katerem se obstoječ podatkovni model verzije N nadgradi v verzijo N+1. Ključna zahteva pri nadgradnji je ohranitev obstoječih podatkov, kar se doseže z SQL ukazi za evolucijo podatkovne sheme.

V primeru večjega števila skript za nadgradnjo je potrebno uvesti ustrezen način njihovega poimenovanja, s katerim zagotovimo [Ambler06]:

- identifikacijo skripte za nadgradnjo in
- identifikacijo odvisnosti med skriptami za nadgradnjo.

Ambler in Sadalage predstavita tri možne strategije poimenovanja, ki vsaka na drug način ustrezajo zadanim pogojem:

- oznaka gradnje (angl. *build number*),
- časovno označevanje (angl. *timestamp*) in
- enolična globalna oznaka (angl. *global unique identifier*, GUID).

Ashley dodatno predlaga uvedbo sledečega poimenovanja [Ashley]:

- zaporedna številka (angl. sequence).

V nadaljevanju so opisane vse štiri strategije poimenovanja.

4.4.1 Oznaka gradnje

Skripte za nadgradnjo se poimenuje glede na oznako gradnje informacijskega sistema. To je tipično naraščajoče naravno število, ki ga dodeljuje sistem za neprestano gradnjo, kot je na primer CruiseControl[19].

---

[19] http:/cruisecontrol.sourceforge.net/

Prednosti:

- preprosto za izvedbo;
- odvisnosti med skriptami za nadgradnjo si sledijo po naraščajočem vrstnem redu oznake za gradnjo;
- verzija podatkovne sheme je neposredno povezana z verzijo aplikacije.

Slabosti:

- predpostavlja povezavo med sistemom za gradnjo in skriptami za nadgradnjo;
- ni možno vzdrževanje zvezne oznake verzije podatkovne sheme, ker v praksi številne gradnje aplikacije niso povezane s spremembo podatkovne sheme;
- težko vzdrževati v primeru, da več aplikacij uporablja isto podatkovno shemo. Vsaka aplikacija ima ločeno označevanje gradnje, ki se lahko prekriva in/ali poruši predpostavko o naraščajočem vrstnem redu;
- dodatni zapleti pri združevanju vzporednih vej razvoja podatkovne sheme. Vsaka veja ima svoje oznake gradnje, ki so naraščajoče znotraj veje. Po združevanju iz imena ni razvidno zaporedje, v katerem si sledijo skripte.

4.4.2 Časovno označevanje

Skripta se poimenuje glede na datum in čas nastanka.

Prednosti:

- preprosto za izvedbo;
- skripte za nadgradnjo si sledijo po naraščajočem datumu in času nastanka.

Slabosti:

- potrebna dodatna evidenca, v kateri se vodi povezava med verzijo gradnje aplikacije in ustreznimi skriptami za nadgradnjo podatkovne sheme;
- dodatni zapleti pri združevanju vzporednih vej razvoja podatkovne sheme, ker po združevanju časovna oznaka ne odraža pravilnega vrstnega reda izvajanja skript.

4.4.3 Enolična globalna oznaka

Skripta se poimenuje glede na enolično oznako kot je to GUID.

Prednosti:

- uporabi se lahko znane generatorje enoličnih globalnih oznak;
- ni problemov pri združevanju vzporednih vej razvoja.

Slabosti:

- poimenovanje datotek z GUID je videti zelo čudno;
- z GUID še vedno potrebujemo način za določanje vrstnega reda posameznih skript za nadgradnjo;
- potrebna je dodatna evidenca, v kateri se vodi povezava med verzijo gradnje aplikacije in ustreznimi skriptami za nadgradnjo podatkovne sheme.

4.4.4 Zaporedna številka

Skripta se poimenuje z zaporedno naraščajočo številko, tako da se za ena poveča številko uporabljeno pri poimenovanju predhodne skripte. Številki se lahko doda tudi komentar, ki opisuje vsebino skripte.

Prednosti:

- preprosta za izvedbo;
- odvisnosti med skriptami za nadgradnjo si sledijo glede na naraščajočo zaporedno številko;
- zvezno označevanje skript za nadgradnjo.

Slabosti:

- potrebna je dodatna evidenca, v kateri se vodi povezava med verzijo gradnje aplikacije in ustreznimi skriptami za nadgradnjo podatkovne sheme;
- dodatni zapleti pri združevanju vzporednih vej razvoja podatkovne sheme, ker po združevanju zaporedna številka ne odraža pravilnega vrstnega reda izvajanja skript.

4.5 Nadgradnja shranjenih procedur

Shranjene procedure sodijo med programske objekte, zato za njihovo posodabljanje ni potrebna uporaba postopkov evolucije podatkovne sheme. Podobno kot pri izvršilnih datotekah se novejša verzija preprosto prepiše čez starejšo verzijo shranjene procedure. Primer 12 prikazuje, kako se z ukazom »*create or replace*« definira prepisovanje obstoječe shranjene procedure z novo verzijo shranjene procedure iz sql skripte.

Primer 12: Primer shranjene procedure v sql skripti

```
create or replace procedure hello_world is
begin
  dbms_output.put_line('Hello world');
end;
/
```

Najbolj enostaven pristop k nadgradnji shranjenih procedur na novo verzijo je ponovno nameščanje vseh shranjenih procedur, ki se nahajajo v sql skriptah. Da v podatkovni shemi po nepotrebnem ne ostajajo stare shranjene procedure, ki v tej verziji niso prisotne, se pred nameščanjem briše vse obstoječe shranjene procedure. Pristop je zelo preprosto avtomatizirati in uporabljati.

Z večanjem števila shranjenih procedur se čas trajanja opisanega pristopa brisanja in ponovnega nameščanja podaljšuje. Namesto da se vedno znova namešča vse shranjene procedure, je smiselno namestiti samo spremembe. Gre za uporabo ideje, ki jo je pri prevajanju programske kode leta 1976 opisal Feldman in na njeni osnovi zasnoval orodje *make* [Feldman].

Ključni problem selektivnega nameščanja je ugotoviti, katere shranjene procedure je potrebno posodobiti in katere ne. V primeru, da sta začetna in končna verzija podatkovne sheme znani in zapisani v sistemu za vodenje konfiguracije, je zelo preprosto narediti medsebojno primerjavo datotek[20] in iz razlik izdelati skripto za nadgradnjo shranjenih procedur, ki jo prikazuje primer 13.

Primer 13: Skripta za nadgradnjo shranjenih procedur

```
PROMPT Odstranitev brisanih shranjenih procedur
drop procedure ...

PROMPT Ponovna namestitev spremenjenih shranjenih procedur
create or replace procedure ....

PROMPT Namestitev novih shranjenih procedur
create or replace procedure ....

PROMPT Dodelitev privilegijev nad novimi shranjenimi procedurami
grant execute on ... to ...
```

Ugotavljanje spremenjenih shranjenih procedur je bolj zahtevno v razvojnem okolju, ki zaradi svoje dinamike nikoli ne ustreza definiranim stanjem v sistemu za vodenje konfiguracije. Iskanje sprememb je potrebno napraviti s primerjavo med trenutnim stanjem sql skript na datotečnem sistemu in shranjenimi procedurami v razvojni shemi. Takšen način sinhronizacije shranjenih procedur med datotečnim sistemom in razvojno podatkovno shemo omogoča preprost in hiter preklop zasebnih razvojnih okolij pri razvoju shranjenih procedur.

[20] Primer orodja za primerjavo vsebine datotek in direktorijev je WinMerge.

Sinhronizacijo je potrebno izvesti vsakič, ko se iz sistema za vodenje konfiguracije osveži lokalno kopijo sql skript.

Pogost razlog za spremembo sql skript s shranjenimi procedurami je tudi generiranje. Pri generiranju nove verzije skript s shranjenimi procedurami je potrebno spremembe uskladiti z vsebino podatkovne sheme.

4.6 Podporne aktivnosti pri nameščanju

V tem poglavju sta predstavljeni podporni aktivnosti vodenja verzije podatkovne sheme in obravnave napak pri izvajanju sql skript.

4.6.1 Verzija podatkovne sheme

Izvajanje sql skript je zelo preprosto in splošno opravilo, ki samo po sebi ne zagotavlja osnovnih zahtev vodenja konfiguracije, kot je identifikacija verzije nameščene podatkovne sheme. Ambler v ta namen predlaga uvedbo tabele, kjer se vodi verzija podatkovne sheme [Ambler06]. Naslednja tabela odraža strategijo poimenovanja skript za nadgradnjo z oznako gradnje aplikacije, uporabna pa je tudi v primeru časovnega in zaporednega označevanja skript za nadgradnjo:

```
CREATE TABLE DatabaseConfiguration
 (SchemaVersion NUMBER NOT NULL);

INSERT INTO DatabaseConfiguration
 ( SchemaVersion ) VALUES (0);
```

Tabela z verzijo se ažurira ob koncu nameščanja skripte za nadgradnjo:

```
UPDATE DatabaseConfiguration
 SET SchemaVersion = 17;
```

Z načinom identifikacije verzije podatkovne sheme je možno avtomatizirati nameščanje nadgradenj in uvesti kontrole za preprečevanje napak pri zamenjavi vrstnega reda nameščanja skript za nadgradnjo. Postopek avtomatskega nadgrajevanja podatkovne sheme je ponazorjen v naslednjih točkah:

1. Prijava na SUPB
2. Preveri obstoj tabele DatabaseConfiguration
3. Namesti vse skripte za nadgradnjo, ki so novejše od verzije v DatabaseConfiguration.SchemaVersion, v naraščajočem vrstnem redu

4. Ob koncu izvajanja vsake skripte za nadgradnjo ažuriraj verzijo podatkovne sheme v DatabaseConfiguration.SchemaVersion

4.6.2 Napake pri izvajanju skript

Ob koncu izvajanja skripte za nadgradnjo je potrebno preveriti, ali je med njenim izvajanjem prišlo do napak. Status izvajanja se zapiše v dopolnjeno tabelo z verzijo podatkovno sheme:

```
ALTER TABLE DatabaseConfiguration ADD
 (UpgradeSucessfull VARCHAR2(1) NOT NULL);
```

V primeru napake je potrebno imeti pripravljen postopek povrnitve prvotnega stanja podatkovne sheme. Avtomatska povrnitev stanja, razen v netrivialnih primerih, navadno ni možna, zato je testiranje skript za nadgradnjo zelo pomemben dejavnik pri njihovi uporabi.

SUPB ne zagotavlja transakcijskih lastnosti ukazov s katerimi spreminjamo strukturo podatkovne sheme (angl. *Data Definition Language*, DDL), zato je za povrnitev prvotnega stanja potrebno uporabiti druge postopke:

1. *Kompenzacija*: večino DDL ukazov se lahko relativno preprosto kompenzira z »inverznimi« SQL ukazi in na ta način rekonstruira prvotno stanje podatkovne sheme. Kot primer: v skripti za nadgradnjo je dodan nov stolpec, ki se po neuspešnem koncu nadgradnje odstrani z ukazom *drop column*.
2. *Obnova iz varnostne kopije*: administrator podatkovne baze povrne prvotno stanje tik pred začetkom nadgradnje.
3. *Druge vrste obnove stanja*: Oracle SUPB nudi možnost hitre povrnitve stanja podatkovne baze v okviru kratkega časovnega okna (angl. *flashback database*). Flashback postopek izvaja administrator podatkovne baze.
4. *Ročna odprava napak*: najpogostejši način je ročno dokončanje namestitve, v okviru katere se odpravi nastale napake.

4.7 Povzetek

Življenjski krog podatkovne sheme se lahko ponazori v obliki acikličnega grafa, kjer vozlišča predstavljajo verzijo podatkovne sheme in povezave med njimi skripte za nadgradnjo. Možne so različne oblike grafov, ki sledijo preprostemu zaporedju ali pa upoštevajo vejanje razvoja.

Začetna namestitev, s katero vzpostavimo začetno verzijo podatkovne sheme, je najkompleksnejša aktivnost pri postavitvi okolja, ker je potrebno namestiti in konfigurirati tudi SUPB in sistemsko okolje. Variacija na temo začetne namestitve je ponovitev začetne namestitve, ki se uporablja predvsem v razvojnih okoljih in v začetnih fazah predstavlja bližnjico za zmanjšanje obsega dela pri izdelavi sql skript in vzdrževanju razvojne podatkovne sheme.

Ko razvoj doseže stabilno stanje, se za nameščanje novih verzij uporablja skripte za nadgradnjo. Največji problem njihove uporabe predstavlja identifikacija skripte za nadgradnjo in zaporedje nameščanja. Obstajajo različni pristopi poimenovanja skript za nadgradnjo, ki poleg identifikacije vključujejo tudi zaporedje izvajanja skript za nadgradnjo. Večji izzivi uporabe skript za nadgradnjo nastopijo z uporabo vejanja, kjer preprosta poimenovanja, kot je oznaka gradnje, ne zadostujejo več za definicijo pravilnega zaporedja.

Pri shranjenih procedurah, kjer razvoj poteka najbolj dinamično, se namestitev poenostavi z uporabo selektivnega nameščanja. Ta pristop je namenjen predvsem vzdrževanju zasebne razvojne podatkovne sheme, ki zaradi svoje dinamike nikoli ne ustreza definiranim stanjem v sistemu za vodenje konfiguracije.

Postopek nameščanja vključuje tudi podporne aktivnosti za vodenje verzije podatkovne sheme in obravnavo napak pri izvajanju sql skript.

5 SQLMake

Postopki nameščanja in nadgrajevanja, ki so predstavljeni v predhodnem poglavju, nimajo zadostne podpore v razvojnih orodjih. Ker ročno izvajanje teh postopkov vzame veliko časa in dopušča možnost napak, je v tem poglavju predstavljena rešitev SQLMake, ki te postopke avtomatizira.

V uvodu so predstavljeni glavni cilji pri izdelavi orodja SQLMake, sorodna orodja ter kaj so razlike med njimi in SQLMake. Sledi predstavitev glavnih funkcionalnosti SQLMake in kakšne spremembe razvojnega procesa zahteva njegova uporaba. Na koncu sledi prikaz hitrosti delovanja SQLMake v primerjavi s klasičnimi postopki nameščanja podatkovne sheme.

5.1 Uvod

Problem, ki se rešuje s shranjevanjem sql skript v sistemu za vodenje konfiguracije, je pomanjkljivo vodenje konfiguracije pri razvoju podatkovne sheme, kar se dodatno potencira z uporabo kratkih razvojnih ciklov. SQLMake pri uporabi sistema za vodenje konfiguracije v razvoju podatkovne sheme nastopa kot vezni člen med sql skriptami in podatkovno shemo.

Glavni cilj orodja SQLMake je omogočiti preprosto in hitro nameščanje ter nadgrajevanje podatkovne sheme na osnovi sql skript, ki so shranjene v sistemu za vodenje konfiguracije. S tem se ustvari platforma za učinkovito implementacijo agilnih pristopov v prvi vrsti zasebnih razvojnih okolij in preoblikovanja.

Osrednja zahteva pri izdelavi SQLMake je, da ne posega v funkcionalnosti obstoječih orodij za podatkovno modeliranje in orodij za razvoj shranjenih procedur, temveč le zaokrožuje nekoliko spremenjeno zaporedje razvojnega cikla in s tem omogoča njihovo preprosto vključitev v agilni razvoj.

5.2 Sorodna orodja

S pojavom agilnih metodologij in prenosom ideje preoblikovanja na področje razvoja podatkovnih shem se je pojavila tudi potreba po orodju za nadgrajevanje podatkovne sheme [Ambler06]. Spodaj našteta orodja te ideje vsaka na svoj način umeščajo v prakso.

Med navedenimi orodji ni orodij, ki temeljijo na medsebojni primerjavi in usklajevanju vsebine dveh podatkovnih shem. V prvi vrsti je problem njihove uporabe vodenje konfiguracije, ker se implicitno pričakuje razvoj podatkovne sheme brez uporabe sql

skript. Drug problem je funkcionalna omejitev primerjave dveh stanj, ki onemogoča avtomatizacijo nadgradnje. Kot primer navajamo preimenovanje tabele, ki ga orodja za primerjavo zaznajo kot brisanje obstoječe in kreiranje nove tabele namesto preprostega preimenovanja obstoječe tabele.

5.2.1 Dbdeploy

Dbdeploy je produkt zaposlenih v podjetju ThoughWorks[21] in je opisan kot orodje za preoblikovanje podatkovne sheme. Namenjen je avtomatiziranemu izvajanju že pripravljenih sql skript za nadgradnjo.

Primer 14: Poimenovanje skript za nadgradnjo pri orodju Dbdeploy

- 10 New Table XYZ.sql
- 11 Pk Rename For Table XYZ.sql
- 12 new Column ABC.sql

Način delovanja je zelo preprost. Vse skripte za nadgradnjo so označene z zaporedno številko. Dbdeploy v konfiguracijski tabeli v podatkovni shemi vodi oznako zadnje nameščene skripte za nadgradnjom, na primer 9. Z novo nadgradnjo orodje dbdeploy poišče vse nove skripte za nadgradnjo, ki imajo večjo zaporedno številko, na primer 10, 11 in 12, ter jih zapiše v izhodno skripto. Pri tem se naraščajoča zaporedna številka upošteva kot vrstni red pri pripravi izhodne skripte. Dbdeploy ne zna izvajati sql skript, tako da se za nameščanje pripravljene skripte uporablja druga orodja.

Primer 15: Skripta za nadgradnjo z UNDO sekcijo, ki jo podpira Dbdeploy

```
CREATE TABLE OSEBE (
OSEBA_ID INTEGER NOT NULL
,IME VARCHAR(300)
);
ALTER TABLE OSEBE ADD CONSTRAINT PK_OSEBA PRIMARY KEY (OSEBA_ID);
--//@UNDO
DROP TABLE OSEBE;
```

V sintaksi skript za nadgradnjo je podprta tudi povrnitev stanja podatkovne sheme na stanje pred namestitvijo skripte za nadgradnjo. V sami skripti za nadgradnjo je možno dodati sekcijo UNDO, kot prikazuje primer 15.

---

[21] Soavtor knjige Refactoring Databases: Evolutionary Database Design [Ambler06] Pramod Sadalage je prav tako zaposlen pri podjetju ThoughtWorks

Povezava na preoblikovanje podatkovne sheme je implementirana preko načina poimenovanja skript za nadgradnjo.

5.2.2 UpToDater

UpToDater je orodje za nadzor sprememb podatkovne sheme, ki je namenjeno agilnim razvojnim skupinam. Spremembe podatkovne sheme so avtomatsko nameščene ob namestitvi nove programske kode na aplikacijski strežnik.

Način delovanja je zelo podoben orodju dbdeploy. Glavna razlika je predvsem v prožilcu, ki sproži nameščanje. Ob gradnji aplikacije se vse sql skripte vključi v aplikacijo kot del namestitvenega paketa (datoteka EAR ali WAR). Ob namestitvi aplikacije na aplikacijski strežnik se avtomatsko aktivira UpToDater, ki primerja sql skripte za nadgradnjo v namestitvenem paketu s predhodno nameščenimi skriptami in izvede vse nove sql skripte.

Spremembe podatkovne sheme so shranjene v obliki sql skript. Zaporedje njihovega nameščanja je definirano z abecednim vrstnim redom, zato je priporočeno poimenovanje z datumom, na primer *2008-16-12-update-customer-types.sql*. Status nameščenih sql skript je zapisan v konfiguracijski tabeli z imenom *uptodater*.

Aktivnosti na tem projektu so potekale v letu 2005, sedaj pa so popolnoma zamrle.

5.2.3 MIGRATEdb

Orodje ima nekoliko drugačen pristop glede na obe že opisani orodji. Kot izhodišče za spreminjanje podatkovne sheme služi XML datoteka s spremembo, ki je sestavljena iz dveh delov:

- test, s pomočjo katerega se preveri, ali je sprememba že nameščena v podatkovno shemo in
- seznam akcij, s katerimi se podatkovna shema spreminja.

Glavna razlika je, da se z uporabo testa izogne potrebi po identifikacije posamezne spremembe in posledično konfiguracijski tabeli z informacijo o nameščenih spremembah.

Primer 16: XML zapis spremembe podatkovne sheme v MIGRATEdb

```
<change>
  <sqltest exists="false">
    SELECT NULL FROM USER_OBJECTS
    WHERE OBJECT_NAME = 'OSEBE'
  </sqltest>
  <sqlaction>
    CREATE TABLE osebe (
```

```
        oseba_id NUMBER
       ,ime VARCHAR2(300)
      )
    </sqlaction>
    <sqlaction>
      ALTER TABLE osebe ADD CONSTRAINT oseba_pk (
      PRIMARY KEY (oseba_id)
      USING INDEX
    </sqlaction>
  </change>
```

Podoben pristop uporablja tudi produkt Bugzilla, kjer se kreiranje in nadgrajevanje podatkovne sheme izvaja z eno samo veliko perl skripto. Nova sprememba podatkovne sheme se vedno doda na koncu in se jo opremi s pripadajočim testom, ki zagotavlja, da se sprememba izvede samo enkrat. Nekateri testi so vključeni kar implicitno v pripravljene ukaze za spremembo podatkovne sheme. Ob kreiranju novega stolpca v tabeli s proceduro *bz_add_column* se vedno preverja, če ta stolpec že obstaja v tabeli, ob spremembi stolpca s proceduro *bz_alter_column* se preverja, če je njegova obstoječa definicija različna od nove definicije in tako dalje.

Primer 17: Odsek iz skripte za nadgradnjo podatkovne sheme Bugzille

```
$dbh->bz_add_column('attachments', 'isobsolete',
                    {TYPE => 'BOOLEAN',
                     NOTNULL => 1,
                     DEFAULT => 'FALSE'
                    });
$dbh->bz_drop_column("profiles", "emailnotification");
$dbh->bz_drop_column("profiles", "newemailtech");
$dbh->bz_alter_column('profiles', 'cryptpassword',
                      {TYPE => 'varchar(128)'});
```

5.2.4 Liquibase

Liquibase je med vsemi naštetimi orodji najbolj ambiciozno zastavljen, izpopolnjen in se tudi aktivno razvija. Podobno kot MIGRATEdb uporablja XML datoteke s spremembami, vendar je funkcionalnost mnogo bolj dovršena. V XML strukturi je definiran zapis SQL ukazov, ki se ob nameščanju spremembe generirajo iz XML zapisa.

Za izdelavo in urejanje datotek s spremembami je na voljo grafični vmesnik, ki omogoča enostavno preoblikovanje podatkovne sheme in hkratno izdelavo datoteke s spremembami.

Za identifikacijo posamezne spremembe se uporablja enolični identifikator:

- avtor spremembe in
- poljubna identifikacija (privzeto je GUID).

Medsebojno zaporedje sprememb je implicitno definirano z zaporedjem v datoteki sprememb. Z vsako spremembo se lahko dodatno definirajo tudi predpogoji. Na primer ali je tabela prazna preden se izvede *drop table*, ali se je že izvedla neka druga sprememba in podobno.

Primer 18: Dodajanje novega stolpca z orodjem Liquibase

```
<changeSet id="4" author="joze">
  <addColumn tableName="osebe">
    <column name="telefonska_stevilka" type="varchar(255)"/>
  </addColumn>
</changeSet>
```

Med zanimivimi lastnostmi je tudi definiranje tako imenovanih kontekstov, s katerimi se ločuje različna okolja kot so na primer razvojno, testno, uvajalno ali produkcijsko okolje. Izvajanje sprememb se lahko veže na izbrani kontekst in s tem loči testne podatke od uvajalnih, vsebino testnih šifrantov od produkcijskih in podobno.

5.3 Razlike glede na sorodna orodja

Orodja, ki so navedena zgoraj, se osredotočajo predvsem na nadgradnjo podatkovnega modela. Primeri iz prakse so pokazali, da so najpogostejši vzrok spremembam podatkovne sheme shranjene procedure. SQLMake zato dopolnjuje razvojni cikel shranjenih procedur z avtomatskim nameščanjem, ki selektivno namešča samo nove in spremenjene shranjene procedure.

Druga pomembna lastnost je podpora delu z sql skriptami, saj je v SQLMake implementiran leksični analizator (angl. *lexer*) in delni razčlenjevalnik (angl. *parser*) za sql skripte in SQL ukaze, ki se nahajajo v njih. Na ta način skripta ni samo datoteka, temveč je razdeljena na posamezne ukaze, ki jih zna SQLMake izvajati. Takšen pristop omogoča pregledno podporo obravnavi napak ob nameščanju.

Dobra podpora delu z sql skriptami zagotavlja nemoteno vključitev SQLMake v obstoječ razvojni proces, kjer se že uporablja obstoječa razvojna orodja, kot sta na primer Oracle Designer in TOAD. Priprava sql skript z obstoječimi orodji poteka v nespremenjeni obliki, ker je SQLMake neobčutljiv na število skript, vrstni red sql ukazov in vsebino skript. Dodatna prednost je, da z uporabo SQLMake ni več potrebno vzdrževati sql skript, ki so namenjene izvajanju namestitve, ker to delo avtomatsko opravlja SQLMake.

Pomembna razlika je podpora ločenim razvojnim okoljem znotraj iste SUPB. Namestitev in vzdrževanje nove podatkovne baze Oracle je kompleksna aktivnost, za katero

je običajno usposobljen le administrator SUPB. Enostavnejša možnost je, da se ta korak izdelave nove SUPB enostavno preskoči ter se nova razvojna shema izdela kar v obstoječi SUPB. Ideja je v tem, da se podatkovna shema namesti pod drugim imenom, na primer namesto z imenom SHEMA se novo okolje namesti v shemo z imenom SHEMA_BETA1. V primeru, da informacijski sistem sestavlja več kot ena podatkovna shema, je potrebno zagotoviti, da se dodeljevanje pravic med shemami izvede pravilno.

5.4 Glavne funkcionalnosti

SQLMake deluje v ukazni vrstici in nudi naslednje funkcionalnosti: začetna namestitev, nadgradnja, status podatkovne sheme, evidentiranje napak in usklajevanje programske kode shranjenih procedur med skriptami in podatkovno shemo.

5.4.1 Namestitev struktur podatkovne sheme

SQLMake za razliko od sorodnih orodij skripte začetne verzije podatkovne sheme obravnava ločeno od skript za nadgradnjo. Razlog je predvsem v tem, da je začetna verzija največkrat sestavljena iz več kot ene sql skripte, tako da je njeno umeščanje v okvir ene same skripte za nadgradnjo nepregledno (na primer 1_zacetnaNamestitev.sql).

Skripte za namestitev struktur začetne verzije se lahko uporablja tudi za namestitev kakšne od kasnejših verzij. Na primer, da se vsakih devet skript za nadgradnjo izdela tudi skripte za namestitev trenutne verzije kot nekakšen posnetek trenutnega stanja (angl. *snapshot*). Na ta način se olajša vzpostavljanje novih okolij, ker ni potrebno vedno namestiti prve verzije podatkovne sheme in celotne verige skript za nadgradnjo, temveč se za izhodišče vzame najnovejšo verzijo podatkovne sheme, ki se nahaja v skriptah za začetno namestitev ter nanjo aplicira manjkajoče skripte za nadgradnjo.

Namestitev struktur podatkovne sheme je implementirana v skladu s funkcionalnostjo, ki je opisana v poglavju 4.3.4. Seznam sql skript, ki so del začetne namestitve, je definiran s pomočjo vhodnih parametrov:

- *Scripts* – pot do sql skript

stikalom:

- *r* – rekurzivno preišči vse poddirektorije

in nastavitvami:

- *SqlFileList* – seznam datotek, ki sodijo med sql skripte na primer *.sql

- *IgnoreFileList* – seznam datotek, ki se ignorirajo pri namestitvi in
- *IgnoreDirList* – seznam direktorijev, ki se ignorirajo pri namestitvi na primer .svn.

Namestitev podatkovne sheme se izvaja v vrstnem redu tipov sql ukazov, ki je definiran z nastavitvijo *InstallOrder*. Tipično so na začetku tabele, nato podatki, primarni ključi, indeksi, tuji ključi in tako dalje.

SQLMake pri namestitvi po izdelavi seznama sql skript iz njih izlušči posamezne sql ukaze in jih razporedi v definiranem vrstnem redu. Sql skripte dopuščajo zelo fleksibilen način zapisa mešanice sql ukazov in sql*plus ukazov, zaradi česar je potrebno implementirati leksični analizator in razčlenjevalnik. Z njuno pomočjo je zelo enostavno razdeliti sql skripte na posamezne sql, plsql in sqlplus ukaze. Kot prikazuje primer 19, se lahko na ta način iz vsebine sql skript pridobi tudi uporabna statistika.

Primer 19: Statistika vsebine sistemskih namestitvenih sql skript za Oracle XE

```
C:\> sqlmake -stats config=sqlmake.oraclexe.config
SQLMake, Copyright Mitja Golouh 2008, mitja.golouh@gmail.com
Revision 7
Loading settings from user defined config SQLMAKE.ORACLEXE.CONFIG
Action: Prints stats

Number of scripts: 930
Number of SQL commands: 31444
Number of PLSQL commands: 2980
Number of SQLPlus commands: 46248
Number of Unknown commands: 0
Elapsed time 00:00:14.3281250 sec
```

Sestavni del namestitve podatkovne sheme so tudi podporne aktivnosti nameščanja: vodenje dnevnika namestitve, vodenje verzije in obravnava napak pri izvajanju sql ukazov.

5.4.2 Delo s shranjenimi procedurami

Delo s shranjenimi procedurami implementira že omenjeni postopek selektivnega nameščanja, ki se zgleduje po orodju *make*. Pomembnost te funkcionalnosti se odraža v izbranem imenu orodja, ki je poimenovano z SQLMake.

Delu s shranjenimi procedurami so namenjeni trije ukazi: *diffPLSQL*, *sync2fs* in *sync2db*. Osrednja funkcionalnost je implementirana v *diffPLSQL*, ki primerja vsebino podatkovne sheme in vsebino skript za naslednje strukture podatkovne sheme:

- prožilce,
- procedure,
- funkcije,
- pakete in
- tipe.

Kot prikazuje primer 20, je rezultat primerjave razdeljen na naslednje disjunktne množice:

- seznam shranjenih procedur, ki se nahajajo samo v skriptah,
- seznam shranjenih procedur, ki se nahajajo samo v podatkovni shemi,
- seznam shranjenih procedur, ki se nahajajo tako v skriptah kot v bazi, vendar imajo različno vsebino in
- število shranjenih procedur, ki se nahajajo tako v skriptah kot v bazi in imajo enako vsebino.

Primer 20: Izpis razlik v shranjenih procedurah med skriptami in podatkovno shemo z SQLMake

```
C:\> sqlmake -diffPLSQL userid=test/geslo@xe scripts=.
SQLMake, Copyright Mitja Golouh 2008, mitja.golouh@gmail.com
Revision 7
No config found. Using default settings

Action: List PLSQL differencies between database schema and scripts

  Objects only on filesystem:
    None
  Objects only in DB:
    None
  Different objects:
    test PACKAGE BODY (.\plsql\test.pkb)
  Equal objects count:
    25 object(s)
Elapsed time 00:00:01.6718750 sec
```

Primerjava, ali je shranjena procedura v skripti enaka tisti v podatkovni shemi, je implementirana preko rezultata zgoščevalne funkcije nad tekstom shranjene procedure[22]. Ker so v praksi pristopi k zapisovanju vsebine sql skript zelo raznoliki, neposredna primerjava med celotno vsebino skripte in tekstom shranjene procedure v katalogu SUPB ni možna:

---

[22] Uporablja se zgoščevalna funkcija MD5.

- v skripti se pogosto nahajajo dodatni SQL*Plus ukazi, kot sta na primer *PROMPT* in *REM*,
- v isti skripti je lahko več shranjenih procedur,
- nekatera orodja, kot je SQL*Plus, ob nameščanju shranjene procedure avtomatsko odrežejo odvečne presledke na koncu vrstic in podobno.

Podobno kot pri nameščanju podatkovne sheme je potrebno vsebino skript razdeliti na sestavne dele in med njimi poiskati shranjene procedure.

Primerjava z *diffPLSQL* ničesar ne spreminja v podatkovni shemi oziroma v skriptah na datotečnem sistemu, temveč le prikaže razliko med skriptami in podatkovno shemo. Funkcionalnosti *sync2db* in *sync2fs* sta namenjeni usklajevanju razlik med obema stanjema, pri čemer prva uskladi vsebino shranjenih procedur v podatkovni shemi s stanjem v sql skriptah in druga ravno nasprotno.

5.4.3 Nadgradnja podatkovne sheme

Osrednji del nadgradnje podatkovne sheme je nadgradnja podatkovnega modela z uporabo skript za nadgradnjo. Implementirana je v skladu z opisom v poglavju 4.4.4. Podprt je način označevanja skript za nadgradnjo z zaporedno številko, ki ga prikazuje primer 21. Skripte za nadgradnjo se nahajajo v direktoriju, ki je definiran z nastavitvijo *UpgradeFolderName*.

SQLMake ob izvajanju nadgradnje v naraščajočem zaporedju namesti vse skripte za nadgradnjo, ki imajo večjo zaporedno številko kot trenutna verzija podatkovne sheme, ki je zapisana v konfiguracijski tabeli.

Primer 21: Poimenovanje skript za nadgradnjo

```
10_newTableXYZ.sql
11_PkRenameForFewTables.sql
12_newColumnABC.sql
```

Vsebina skripte za nadgradnjo ne vsebuje preprosto zaporedje SQL ukazov, s katerimi se izvaja evolucija podatkovnega modela.

Primer 22: Vsebina skripte za nadgradnjo

```
create table test
(
  a varchar2(2)
);
```

Dodano vrednost nameščanja z SQLMake prinaša obravnava napak, ki je integrirana v orodje SQLMake in nudi osnovne mehanizme evidentiranja in označevanja napak. V nadgradnjo podatkovne sheme je vključena tudi sinhronizacija PL/SQL (*sync2db*), ki poskrbi za pravilno stanje shranjenih procedur v podatkovni shemi.

Primer 23: Nadgradnja podatkovne sheme z SQLMake

```
C:\>sqlmake -upgrade userid=test/geslo@xe scripts=.
...
Action: Upgrades schema based on upgrade scripts

Upgrading data model ...
Execute upgrade script 16_test.sql...
create table test...
Setting datamodel version to 16...
PLSQL code synchronization ...
  Objects only on filesystem:
    None
  Objects only in DB:
    None
  Different objects:
    Modified PACKAGE BODY UTIL
  Equal objects count:
    25 object(s)
Recompiling invalid stored procedures ...
Elapsed time 00:00:08.7031250 sec
```

5.4.4 Nastavitve delovanja

Nastavitve so zapisane v nastavitveni datoteki, ki se imenuje *sqlmake.config*. Lokacija nastavitvene datoteke se avtomatsko poišče v naslednjem vrstnem redu:

- lokacija nastavitvene datoteke je podana kot parameter *config*,
- nastavitvena datoteka se nahaja v direktorju s skriptami,
- nastavitvena datoteka se nahaja v trenutnem direktoriju ali
- nastavitvena datoteka ne obstaja na nobeni izmed navedenih lokacij – uporabi privzete vrednosti nastavitev.

V nastavitveni datoteki se lahko definira naslednje nastavitve:

1. *SqlFileList* – seznam datotečnih filtrov, s katerim se definira seznam sql skript, ki so sestavni del začetne namestitve, na primer: **.tab, *.con, *.ind*
2. *PlsqlFileList* – seznam datotečnih filtrov, s katerim se definira seznam sql skript, ki vsebujejo shranjene procedure, na primer: **.pks, *.pkb, *.trg, *.prc, *.fnc*
3. *IgnoreDirList* – seznam direktorijev, ki se ignorirajo pri iskanju sql skript, na primer: *^[.]svn$, ^[.]cvs$*

4. *IgnoreFileList* – seznam datotek, ki se ignorirajo pri iskanju sql skript, na primer vse sql skripte s testi, ki se začnejo na ut_*: *(?i)^ut_.**
5. *InstallOrder* – vrstni red tipov sql ukazov, s katerim je definiran vrstni red namestitve (primer 11), na primer: *SYNONYM, TABLE, VIEW, COMMENT, SEQUENCE, SINGLE TABLE INSERT, PRIMARY KEY, UNIQUE KEY, UNIQUE INDEX, INDEX, FOREIGN KEY, CHECK, FUNCTION, PROCEDURE, PACKAGE, TYPE, PACKAGE BODY, TYPE BODY, TRIGGER, GRANT*
6. *FileWithDatamodelVersion* – ime datoteke, kjer je zapisana verzija podatkovnega modela, na primer: *version.sql*
7. *DatamodelVersionPattern* – vzorec, ki mu sledi verzija podatkovnega modela, na primer: *define datamodel_version =*
8. *UpgradeFolderName* – ime direktorija, kjer se nahajajo skripte za nadgradnjo, na primer: *upgrade*. Ta direktorij se avtomatsko doda na seznam direktorijev, ki se ignorirajo pri iskanju sql skript.

Dodatno se lahko v nastavitveni datoteki definira vse parametre, ki jih SQLMake podpira preko ukazne vrstice. Na ta način se izognemo tipkanju dolgih ukaznih vrstic. V primeru, da se parameter nahaja v nastavitveni datoteki in v ukazni vrstici, ima slednji prioriteto.

5.4.5 Vodenje verzije sheme

Tako kot sorodna orodja tudi SQLMake uporablja tabelo za evidentiranje verzije podatkovne sheme. Ker se ista tabela uporablja tudi za evidentiranje napak pri nameščanju, je struktura tabele zasnovana zelo splošno.

Primer 24: Konfiguracijska tabela orodja SQLMake

```
create table sqlmake
  path     varchar2(250) not null,
  name     varchar2(30)  not null,
  value    varchar2(30)  not null,
  sdate_i  date          not null,
  suser_i  varchar2(100) not null,
  sdate_u  date          null,
  suser_u  varchar2(100) null,
  value_v1 varchar2(800) null,
  value_v2 varchar2(800) null,
  value_c1 clob          null,
  value_c2 clob          null,
  value_n1 number        null,
  value_n2 number        null)
```

Verzija podatkovne sheme se zapisuje na naslednji način, pri čemer se vedno začne z začetno namestitvijo (INSTALL) in nadaljuje z nadgradnjami podatkovne sheme (UPGRADE):

Path	Name	Value
VERSION	INSTALL	1
VERSION	UPGRADE	2
VERSION	UPGRADE	3

Trenutno verzijo podatkovne sheme vrača naslednji sql ukaz:

```
select max(to_number(value)) value
from sqlmake
where path = 'VERSION'
```

5.4.6 Status sheme in delo z napakami

Ob nameščanju in nadgradnji podatkovne sheme lahko kljub intenzivnemu testiranju nastopijo napake, ki se ob nameščanju z SQLMake avtomatsko evidentirajo v konfiguracijski tabeli. Vsebina te tabele je podlaga za prikazovanje statusa podatkovne sheme.

Primer 25: Izpis statusa podatkovne sheme z SQLMake

```
C:\> sqlmake -status userid=test/geslo@xe
SQLMake, Copyright Mitja Golouh 2008, mitja.golouh@gmail.com
Revisin 7
No config found. Using default settings

Action: Prints target schema status

Current data model version: 15
564 error(s) occured during install
1 error(s) cleared after install
3 invalid object(s)
Installation failed
Elapsed time 00:00:00.8906250 sec
```

Napake pri namestitvi oziroma nadgradnji se v konfiguracijsko tabelo zapisujejo na naslednji način:

Path	value	value_v2	value_v1	value_n1	value_c2	value_c1
'ERRORS.'|| verzija podatkovne sheme	Zaporedna številka napake	Tip SQL ukaza	Ime skripte	Vrstica v skripti, kjer se nahaja SQL ukaz	SQL ukaz	Tekst napake

Napaka, ki je razrešena, se označi na naslednji način:

```
update sqlmake
set    path = 'CLEARED.' || :version,
where  path = 'ERRORS.' || :version
       and value = :seqNo
```

Pregled in reševanje napak sta podprta iz ukazne vrstice z naslednjimi funkcionalnostmi:

- prikaži napake (*list_errors*),
- označi vse napake kot uspešno razrešene (*clear_errors*) in
- označi napako s podano številko kot razrešeno (*clear_error*).

List_errors izpiše vse sql ukaze s kratko identifikacijo lokacije sql skripte in tekstom napake v obliki sql skripte.

Primer 26: Izpis napak pri namestitvi/nadgradnji podatkovne sheme z SQLMake

```
...
PROMPT create foreign key osb_hst_fk on osb_oseba
REM Sequence id: 2137
REM c:\src\db\test\datamodel\30_constraints.sql, Line no: 1728
REM ORA-02275: such a referential constraint already exists in the
table
ALTER TABLE OSB_OSEBA ADD (CONSTRAINT
 OSB_HST_FK FOREIGN KEY
  (HST_ID_NASLOV) REFERENCES HST_HIS_STEV
  (HST_ID))
/
...
```

Clear_error označi posamezno napako kot razrešeno.

```
C:\> sqlmake -clear_error userid=test/geslo@xe 2145
...
Action: Clear error with [sequenceNumber] from last install

Error with sequence number 2145 cleared
Elapsed time 00:00:00.5312500 sec
```

Clear_errors označi vse napake kot razrešene.

```
C:\> sqlmake -clear_errors userid=test/geslo@xe
...
Action: Clear all errors from last install

563 error(s) cleared
Elapsed time 00:00:00.6718750 sec
```

5.5 Vpliv na način dela

SQLMake v sebi združuje avtomatizirano nadgradnjo podatkovnega modela in avtomatizirano nameščanje shranjenih procedur. Na ta način je omogočena uporaba zasebnih razvojnih okolij tudi v razvoju podatkovnih shem.

V tem poglavju so predstavljeni posamezni koraki pri razvoju podatkovne sheme in spremembe, ki jih uvaja SQLMake. Predpostavka pri uporabi SQLMake je, da se celoten razvoj podatkovne sheme sproti odraža v sql skriptah, ki so shranjene v sistemu za vodenje konfiguracije. Tak način dela je z malenkostnimi spremembami obstoječih postopkov izvedljiv z obstoječimi razvojnimi orodji, kar je bila tudi ena izmed zahtev pri izdelavi orodja SQLMake.

5.5.1 Organizacija datotečnega sistema

SQLMake je zaradi načina dela z sql skriptami zelo neobčutljiv na organizacijo direktorijev. Nekaj pravil kljub temu velja:

- znotraj enega direktorija se nahajajo skripte ene podatkovne sheme,
- skripte za nadgradnjo so shranjene v ločenem direktoriju in
- ime skripte za nadgradnjo se začne z zaporedno številko skripte.

Konkretni primeri organizacije datotečnega sistema so podani v poglavju 3.

5.5.2 Izdelava razvojnega okolja

Izdelava zasebnega razvojnega okolja poteka v dveh korakih:

1. izdelava lokalne kopije skript iz sistema za vodenje konfiguracije

   ```
   svn checkout https://svn.hostname/repos/trunk/src/db/
   ```

2. namestitev skript v prazno shemo

   ```
   sqlmake –install userid=username/password@dbalias
   ```

5.5.3 Spremembe podatkovnega modela

Pri podatkovnem modeliranju se največkrat uporablja orodja za podatkovno modeliranje, kot sta Oracle Designer in ERwin. Z uporabo SQLMake se obstoječi postopek razvoja podatkovnega modela ne spreminja, razlika je le v tem, kdaj in s kakšnim imenom se izdela skripta za nadgradnjo.

Priporočljivo je, da se skripte za nadgradnjo izdeluje sproti, s pomočjo generatorja v orodju za podatkovno modeliranje. Generirano skripto za nadgradnjo se dodatno pregleda in shrani v lokalno kopijo projekta. Poimenovanje skript sledi navodilom poimenovanje z zaporedno številko.

Preden se skripto za nadgradnjo zapiše v sistem za vodenje konfiguracije, jo je potrebno testirati. Najbolj enostaven način je izdelava novega okolja s starejšo verzijo podatkovne sheme, ki se ga nadgradi z novo skripto za nadgradnjo in s tem preveri pravilnost njenega delovanja.

5.5.4 Razvoj shranjenih procedur

Tipično se pri razvoju shranjenih procedur uporablja integrirana razvojna orodja, kot sta TOAD in PL/SQL Developer. Način razvoja shranjenih procedur poteka na naslednji način:

- odpri shranjeno proceduro iz razvojne sheme,
- dopolni proceduro in
- prevedi proceduro v razvojni podatkovni shemi.

Vodenje konfiguracije z opisanim pristopom je pomanjkljivo, prav tako nastopijo težave v primeru več razvijalcev. Delo s shranjenimi procedurami, ki je predlagano v okviru orodja SQLMake, te pomanjkljivosti odpravlja, tako da kot osnovno enoto dela obravnava skripto iz sistema za vodenje konfiguracije. Spremenjen razvojni cikel je:

- odpri skripto s shranjeno proceduro,
- dopolni proceduro,
- shrani spremenjeno skripto s proceduro in
- prevedi ter testiraj proceduro v razvojni podatkovni shemi.

Tak način dela s shranjenimi procedurami v razvoj neposredno vključi tudi sistem za vodenje konfiguracije.

5.5.5 Shranjevanje sprememb v sistem za vodenje konfiguracije

Ves čas razvoja je vsebina skript enaka vsebini razvojne podatkovne sheme. Prenos nastalih sprememb v sistem za vodenje konfiguracije ne vključuje nobene izdelave sql skript, temveč preprost zapis sprememb lokalne kopije skript v sistem za vodenje konfiguracije.

```
svn commit
```

5.5.6 Osveževanje zasebnega razvojnega okolja

Ob sočasnem razvoju več razvijalcev je potrebno v zasebno razvojno okolje občasno prevzeti tudi spremembe, ki so jih napravili drugi razvijalci. Prvi korak je osvežitev lokalne kopije skript iz sistema za vodenje konfiguracije:

```
svn update
```

V drugem koraku nastopi SQLMake, ki poskrbi, da se vse spremembe iz skript pravilno namestijo v razvojno podatkovno shemo:

```
sqlmake –upgrade userid=username/password@dbalias
```

Operacija poteka hitro, ker se uporabi le na nove skripte za nadgradnjo in nove oziroma spremenjene shranjene procedure.

5.6 Hitrost delovanja

Glavna prednost SQLMake je avtomatizacija ročnega dela pri vzdrževanju zasebnega razvojnega okolja podatkovne sheme. Ker gre za povsem drug način dela, je primerjava hitrosti delovanja na tem področju nesmiselna. Izpostavlja se lahko le kvalitativne razlike, saj se z avtomatizacijo celoten proces nadgradnje zelo preprosto vključi v sistem neprestane gradnje, prepusti v izvajanje osebam brez poznavanja Oracle SUPB in s tem razbremeni razvojno skupino operativnih nalog, preprosto ponavlja za namene testiranja idr.

Pri kvalitativnih meritvah je možno primerjavo napraviti le pri nameščanju shranjenih procedur, ki je bilo do sedaj avtomatizirano na naslednji način:

- brisanje vseh obstoječih shranjenih procedur v podatkovni shemi,
- kreiranje shranjenih procedur iz sql skript trenutne verzije in
- prevajanje neprevedenih shranjenih procedur.

Opisani standarden postopek je vzet kot izhodišče za primerjavo z delovanjem SQLMake, ki pri nameščanju namesti samo spremenjene shranjene procedure. Predpostavka je, da SQLMake nad majhnim številom sprememb deluje občutno hitreje kot standardni postopek nameščanja, medtem ko je namestitev vseh shranjenih procedur časovno primerljiva s standardnim postopkom nameščanja.

Okolje za testiranje je računalnik s procesorjem Intel Celeron 1,5 GHz in 512 MB pomnilnika. Za testiranje je uporabljen primer, ki vsebuje 825 shranjenih procedur. Kot izhodišče je navedena kratka statistika, ki testni primer umesti v povprečje primerov iz prakse, ki so opisani v poglavju 3.

Tabela 5: Statistika SQL ukazov za posamezne primere sistemov

	Število skript	**Število DDL ukazov**	**Komentarji tabel in stolpcev**	**Število DML ukazov**	**Število PLSQL ukazov**	**Število SQLPlus ukazov**
MR	3254	2514	6186	18442	2084	9228
CRP	1366	2765	2533	92210	420	831
Oracle XE	930	17152	9686	4613	2980	46241
Testni primer	828	178	0	0	825	253
UCG	221	1100	1	66018	200	1963

Za testiranje je na datotečnem sistemu pripravljena naslednja struktura direktorijev in datotek:

shema	Izhodiščna verzija sql skript
shema10	10 spremenjenih skript
shema20	20 spremenjenih skript
shema50	50 spremenjenih skript
shema100	100 spremenjenih skript
shemaVse	Vse skripte so spremenjene

Izvajanje meritve je potekalo na sledeči način:

20 iteracij	c:\primer> sqlmake -sync2db scripts=.\app
10 iteracij	c:\primer> sqlmake -sync2db scripts=.\app10
	c:\primer> sqlmake -sync2db scripts=.\app
10 iteracij	c:\primer> sqlmake -sync2db scripts=.\app20
	c:\primer> sqlmake -sync2db scripts=.\app
10 iteracij	c:\primer> sqlmake -sync2db scripts=.\app50
	c:\primer> sqlmake -sync2db scripts=.\app
10 iteracij	c:\primer> sqlmake -sync2db scripts=.\app100
	c:\primer> sqlmake -sync2db scripts=.\app
10 iteracij	c:\primer> sqlmake -sync2db scripts=.\appVse
	c:\primer> sqlmake -sync2db scripts=.\app
20 iteracij	standardni postopek nameščanja

Pri rezultatih sta izločeni dve najboljši in dve najslabši meritvi. Povprečje preostalih rezultatov je prikazano v spodnji tabeli.

Tabela 6: Rezultati nameščanja testnega primer z SQLMake

Število spremenjenih shranjenih procedur	**0**	**10**	**20**	**50**	**100**	**Vse**	**Standarden postopek nameščanja**
Povprečen čas potreben za nameščanje	10 s	12 s	15 s	18 s	53 s	5 min 56 s	5 min 34 s

Iz rezultatov je razvidno, da je čas osveževanja zasebnega razvojnega okolja s spremembami shranjenih procedur ob uporabi SQLMake veliko krajši. Tudi nameščanje vseh shranjenih procedur je primerljivo s standardnim postopkom nameščanja.

V samem postopku nameščanja shranjenih procedur se skriva še nekaj možnosti za optimizacijo. Manjša izboljšava je, da se ob prvem izvajanju shrani rezultate zgoščevalne funkcije MD5, tako da se ob naslednjem izvajanju kontrolira samo tiste skripte oziroma shranjene procedure, ki so se spremenile. Druga izboljšava je v boljšem razpoznavanju internih odvisnosti med shranjenimi procedurami. SUPB namreč ob nameščanju shranjenih procedur veliko časa porabi za kontrolo in vzdrževanje medsebojnih odvisnosti med shranjenimi procedurami. S pravim vrstnim redom nameščanja shranjenih procedur se lahko ta čas občutno zmanjša in z njim tudi trajanje namestitve.

5.7 Povzetek

SQLMake je orodje za nameščanje in nadgrajevanje podatkovne sheme. Od sorodnih orodij ga loči predvsem podpora delu s shranjenimi procedurami, dobra podpora delu z sql skriptami in možnost izdelave enega ali več zasebnih razvojnih okolij v isti instanci Oracle SUPB.

Funkcionalnosti orodja v celoti pokrijejo korake nameščanja in nadgrajevanja podatkovne sheme ter obravnave napak, pri čemer so potrebne minimalne spremembe v razvojnem procesu.

Hitrost delovanja je preizkušena pri delu s shranjenimi procedurami, kjer selektivno prevajanje občutno zmanjša čas nameščanja novih verzij. Pri vzdrževanju zasebnega razvojnega okolja se zato namesto o minutah govori o sekundah, ki so potrebne, da se obstoječe okolje nadgradi na novo verzijo.

6 Sklepne ugotovitve

V sklepnih ugotovitvah so povzete ugotovitve magistrske naloge ter predlog možnosti nadaljnjega dela na predstavljenem orodju SQLMake.

6.1 Povzetek ugotovitev naloge

Uporaba agilnih metodologij pri razvoju informacijskih sistemov se osredotoča predvsem na razvoj programske kode in manj na razvoj podatkovne baze. Ambler in Sadalage pokažeta, da je številne pristope, ki so podrobno razdelani pri programskih jezikih, možno prenesti tudi na področje razvoja podatkovne baze. Pri tem se v praksi naleti na številne ovire, ki izhajajo iz posebnosti razvoja podatkovnih baz, kot je na primer evolucija podatkovne sheme in iz slabe podpore novim procesom v obstoječih razvojnih orodjih. V nalogi sta podrobno predstavljena prva dva izmed agilnih pristopov pri razvoju podatkovnih baz: uporaba sistemov za vodenje konfiguracije in zasebna razvojna okolja.

Sodobni sistemi za vodenje konfiguracije kot enoto vodenja verzije uporabljajo datoteko, zato nastopijo omejitve pri vodenju verzij izdelkov s kompleksno interno strukturo, kot sta podatkovna shema in metapodatkovni repozitorij. Iz primerov iz prakse je razvidno, da se v sistem za vodenje konfiguracije shranjuje sql skripte, na podlagi katerih se lahko namesti podatkovna shema. Pri tem velik pomen pridobita podporni aktivnosti vzdrževanja in nameščanja sql skript. Njun pomen se dodatno povečuje ob uporabi kratkih razvojnih ciklov.

Pri definiranju aktivnosti nameščanja sql skript so ugotovljene naslednje tri možnosti: začetna namestitev, nadgradnja podatkovne sheme in namestitev shranjenih procedur. Iz analize obstoječih orodij, ki so namenjena nameščanju sql skript, so se izkazale možnosti izboljšave predvsem pri izdelavi novega okolja, nameščanju shranjenih procedur in evidentiranju napak ob izvajanju sql skript. Te pomanjkljivosti so vodile k izdelavi orodja SQLmake, ki skupaj s primerom organizacije sql skript na datotečnem sistemu predstavlja glavni rezultat magistrske naloge.

SQLMake je orodje za avtomatsko nameščanje podatkovne sheme na osnovi sql skript, pri čemer zahteva le minimalne spremembe v obstoječem razvojnem procesu. Z avtomatizacijo procesov namestitve in nadgradnje podatkovne sheme omogoča vključitev podatkovne sheme v proces neprestane gradnje, ki je eden izmed značilnih pristopov agilnih metodologij. Proces neprestane gradnje sproti rešuje probleme pri integraciji aplikacije in

podatkovne sheme ter tako prekine nezaželen vpliv sprememb podatkovne sheme na nedelovanje aplikacije.

Najvidnejša razlika SQLMake glede na sorodna orodja je ločena obravnava nameščanja shranjenih procedur, ki se ne izvaja v okviru skript za nadgradnjo. Razlog za takšno odločitev izhaja iz statistike nad primeri iz prakse, kjer se 55% vseh sprememb podatkovne sheme odraža na shranjene procedure. Pri nameščanju shranjenih procedur je implementiran algoritem selektivnega nameščanja, s katerim se v podatkovno shemo namesti samo spremenjene shranjene procedure. Takšen postopek občutno skrajša čas namestitive nove verzije in dodatno omogoči učinkovito in hitro vzdrževanje zasebnih razvojnih podatkovnih shem.

6.2 Možnosti nadaljnjega dela

Pri nadaljnjem delu je zanimiva predvsem uporaba SQLMake v praksi, kjer se razišče uporabnost predlaganih pristopov pri vzdrževanju zasebnih razvojnih podatkovnih shem. Vzporedno s tem se izvede tudi stabilizacija orodja in priprava dokumentacije s primeri, ki ji sledi objava orodja v odprtokodni skupnosti.

K nadaljnjemu delu sodi tudi razširitev osnovne funkcionalnosti orodja. Selektivno nameščanje je trenutno omejeno na shranjene procedure, kar je smiselno razširiti na vse SQL strukture, ki podpirajo »*create or replace*« oziroma »*grant/revoke*« sintakso (na primer pogledi, sinonimi, dodeljevanje privilegijev). Smiselna je tudi integracija v sistem za gradnjo NANT, ter dopolnitev vključujoče/izključevalne (angl. *inclusive/exclusive*) definicije seznama sql skript s podporo pravemu seznamu v obliki projektne definicije.

Zanimiv problem, ki ni bil obravnavan v magistrski nalogi je priprava in vzdrževanje sql skript s podatki. V praksi je priprava sql skript s podatki zelo zamudna, prav tako popravljanje njihove vsebine ob spremembi strukture šifrantov. Kot možnost nadaljnjega dela navajamo pripravo in predvsem vzdrževanje skript s podatki ob spremembah strukture tabel.

7 Viri

[Ambler06] S. W. Ambler, P. J. Sadalage »*Refactoring Databases: Evolutionary Database Design*«, Addison Wesley Professional, 2006

[Bajec] M. Bajec, M. Krisper, »*Agilne metodologije razvoja informacijskih sistemov*«, Uporabna informatika, Ljubljana, Slovenija, apr., maj, jun. 2003, 11(2), str. 68 – 76

[Beck03] K. Beck, »*Test-Driven Development: By Example*«, Addison-Wesley, 2003

[Curino] C.A. Curino, H. J. Moon, C. Zaniolo, »*Graceful Database Schema Evolution: the PRISM Workbench*«, VLDB, 2008

[Dolstra] E. Dolstra, »*The Purely Functional Software Deployment Model*«, doktorska disertacija

[EMRIS-Objektni] M. Krisper in drugi, »*Enotna metodologija razvoja informacijskih sistemov Objektni razvoj IS*«, Center Vlade RS za informatiko, Ljubljana, januar 2000

[Feldman] S. I. Feldman, »*Make-A Program for Maintaining Computer Programs*«, Software - Practice and Experience, 9(4), 1979, str. 255 – 265

[Fowler99] M. Fowler, »*Refactoring: Improving the Design of Existing Code*«, Addison-Wesley Longman, 1999

[Girschick] M. Girschick, »*Difference Detection and Visualization in UML Class Diagrams*«, Technical Report TUD-CS-2006-5, TU Darmstadt, Department of Computer Science Metamodelling and its Applications, 2006

[Haftmann] F. Haftmann, D. Kossmann, E. Lo, »*A framework for efficient regression tests on database applications*«, VLDB Journal (2007) 16: str. 145–164

[Hall] R. S. Hall in drugi, »*Evaluating Software Deployment Languages and Schema, An Experience Report*«, Universitiy of Colorado, 1998

[Larman] C. Larman, V. Basili, »*Iterative and Incremental Development: A Brief History*«, IEEE Computer, junij 2003

[Odyssey-VCS] L. Murta in drugi, »*Towards Odyssey-VCS 2: Improvements over a UML-based Version Control System*«, Proceedings of the 2008 international workshop on Comparison and versioning of software models, 2008, str 25-

30

[Pivka] M. Pivka, »*Kakovost v programskem inženirstvu*«, Desk, Izola, 1996

[Roddick95] J. F. Roddick, »*A Survey of Schema Versioning Issues for Database Systems*«, Information and Software Technology, 37(7), 1995, str. 383 – 393

[Schuh] P. Schuh, »*Agility and the Database*«, Thoughtworks, 2002

[Snodgrass94] R. T. Snodgrass, I. Ahn in drugi, »*Tsql2 Language Specification*«, SIGMOD Record, 23(1), 1994, str. 65 – 86

[Tršinar] I. Tršinar, »*Centralni register prebivalstva*«, Uradni List RS, 1999

8 Ostali viri

[Ashley] N. Ashley, »*Taking control of your database development*«, dostopno na: http://dbdeploy.com

[Agile Manifesto] Agile alliance, »*Manifesto for Agile Software Development*«, dostopno na: www.agilealliance.org, http://www.agilemanifesto.org/

[Agile eWorkshop02] *Summary of the First eWorkshop on Agile Methods*, 8. april 2002, dostopno na: http://www.agilealliance.org/show/1151

[SQL03] ISO/IEC 9075:2003, »*Database Language-SQL*«, 2003

Izjava o samostojnosti dela

Izjavljam, da sem magistrsko delo izdelal samostojno pod vodstvom mentorja doc. dr. Marka Bajca. Izkazano pomoč drugih sodelavcev sem v celoti navedel v zahvali.

Mitja Golouh

Univerza
v Ljubljani

Fakulteta *za računalništvo*
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št.: 74-MAG-ISO/2006
Datum: 18. 9. 2006

Mitja G O L O U H, univ. dipl. inž. rač. in inf.

L j u b l j a n a

Fakulteta za računalništvo in informatiko Univerze v Ljubljani izdaja naslednjo magistrsko nalogo

Naslov naloge: **UPORABA AGILNIH PRISTOPOV PRI RAZVOJU PODATKOVNIH BAZ**

Tematika naloge:

Naloga naj obsega opis uporabe agilnih pristopov pri razvoju podatkovnih baz, s poudarkom na opisu področij, kjer obstoječe informacijske rešitve ne nudijo ustrezne podpore agilnim razvojnim pristopom. Naloga naj razdela uporabo sistemov za vodenje konfiguracij programske opreme, implementacijo zasebnih razvojnih okolij in usklajevanje sprememb med njimi. Obravnava naj posebnosti pri postavitvi in nadgradnji podatkovne sheme ter problematiko usklajevanja postavitve odjemalske aplikacije in podatkovne baze.

Mentor:

doc. dr. Marko Bajec

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

za Dekan:

prof.dr. Boštjan Vilfan