Gränssnittsändringar som framtvingar ändringar i applikationsprogram

Kandidatuppsats
i datavetenskap 2010.

Jens Boström

Abstrakt

Programmeringsgränssnitt förväntas ofta av programmerare att de förblir stabila genom biblioteks och program pakets evolution. Trots det implicita förtroende mot programmeringsgränssnitten genomgår de också sin egen evolution och alla gånger så kan inte kompatibiliteten garanteras efter en uppdatering till nyare versioner. Denna uppsats ämnar beskriva användning av programmeringsgränssnitt inom Java och hur inkompatibilitet i programmeringsgränssnitt kan uppstå vid uppdatering av bibliotek och program med avseende på gränssnitt.

Innehållsförteckning

1. Introduktion

Under existensen för ett mjukvaruprojekt eller en komponents livstid så sker det oftast en hel del utveckling, även inom programmeringsgränssnitt som projektet visar till andra system. Förändringen av det externa programmeringsgränssnittet som en komponent eller ett bibliotek visar är i många avseenden oönskad. Detta på grund av att denna förändring direkt påverkar kompatibiliteten med andra komponenter som använder sig av gränssnittet. I denna uppsats ämnar jag fokusera på de faktorer som kräver en förändring i programmeringsgränssnitt av bibliotek, särskilt gällande migration av applikationsprogram till nya biblioteksversioner eller komponeter. Syftet med uppsatsen är att presentera en litteratursammanfattning av samtida forskning inom evolution av gränssnitt inom java applikationsprogram med en speciell fokus på ändringar som inför inkompatibilitet mellan bibliotek eller komponenter och applikationsprogram.

Dig och Johnson(Dig 2005) påpekar att omstrukturering av kod är en viktig del av programmeringsgränssnittets utveckling och det kommer att spela en central roll gällande förändring i objektorienterade gränssnitt. Begreppet omstrukturering av kod (eng. refactoring) är ett omfattande begrepp. Trots det så har Fowler (1999) försökt att definiera det som en process som inte förändrar det yttre beteendet utan som istället förbättrar den inre strukturen av systemet. En anmärkningsvärd observation är att en förändring i programmeringsgränssnittet inte nödvändigtvis strider emot tanken att bevara systemets beteende. Det här innebär att den omstrukturering som medför en förändring i gränssnittet är av speciellt intresse för uppsatsen.

Objektorienterade programmeringsspråk i allmänhet erbjuder en bra möjlighet att illustrera förändring av programmeringsgränssnittet. Detta eftersom det finns en klar skillnad mellan allmänna metoder som är del av programmeringsgränssnittet samt privata metoder vilka endast är användbara för objektens egna behov.

Mjukvara evolution och underhåll är termer som har en del gemensamt, denna uppsats försöker ge en kort förklaring för skillnaden mellan de båda termerna men främst hur de sammanfaller.

2. Programmeringsgränssnitt inom Java

Användningen av programmeringsgränssnitt som ett enda omfattande koncept kan eventuellt ses som förvirrande. För att förenkla uppfattningen om vad programmeringsgränssnitt egentligen är så skall jag närmare presentera termen och dess användning genom att definiera begreppet samt ge en överblick av olika sätt som det tar form i praktiken. Efter denna presentation följer en kort genomgång av hur det är förknippat med Java och dess programmeringsgränssnitt.

2.1 Programmeringsgränssnitt enligt ISO/IEC JTC 1

ISO/IEC JTC 1 är Internationella standardiseringsorganisationens (ISO) och Internationella elektrotekniska kommissionens (IEC) gemensamma tekniska kommitté. Kommitténs uppgift är att utveckla grundläggande standarder inom informationsteknologi utgående ifrån vilka andra instanser kan utveckla mera specifika standarder (JTC 1 standing document).

Enligt ISO/IEC JTC 1 är ett programmeringsgränssnitt "a boundary across which application software uses facilities of programming languages to invoke services." Detta innebär att det är ett gränssnitt mellan programmeringsspråkets resurser och ett program. Denna gräns möjliggör att programmet kan använda någon av resursernas funktionaliteter. Av kommunikation som överskrider denna gräns krävs det att den går att binda till semantiken och syntaxen för programmeringsspråket som använder gränssnittet (ISO/IEC JTC 1 Directives). En viktig aspekt av definitionen ovan är att det inte finns krav på att kommunikationen skall kunna bindas andra vägen, det vill säga från programmet till gränssnittet, vilket i sin tur möjliggör mera abstrakta programmeringsgränssnitt.

Enligt ISO/IEC JTC 1 (ISO/IEC JTC 1 Directives) så delas specifikationer av programmeringsgränssnitt in i tre stycken specifika grupper. Den första gruppen innefattar specifikationer av programmeringsspråk så som till exempel C och ADA vars uppgift är att beskriva själva språket. Den andra gruppen består av en gränssnittsspecifikation som är oberoende av programmeringsspråk och som dessutom går att binda till flera olika programmeringsspråk. Den tredje gruppen är programmeringsspråkets gränssnittsspecifikation som syntax och data typer, det vill säga den specifikation som är till för en programmerare i ett enskilt språk.

2.2 Java plattformens allmänna strukturella gränssnitt

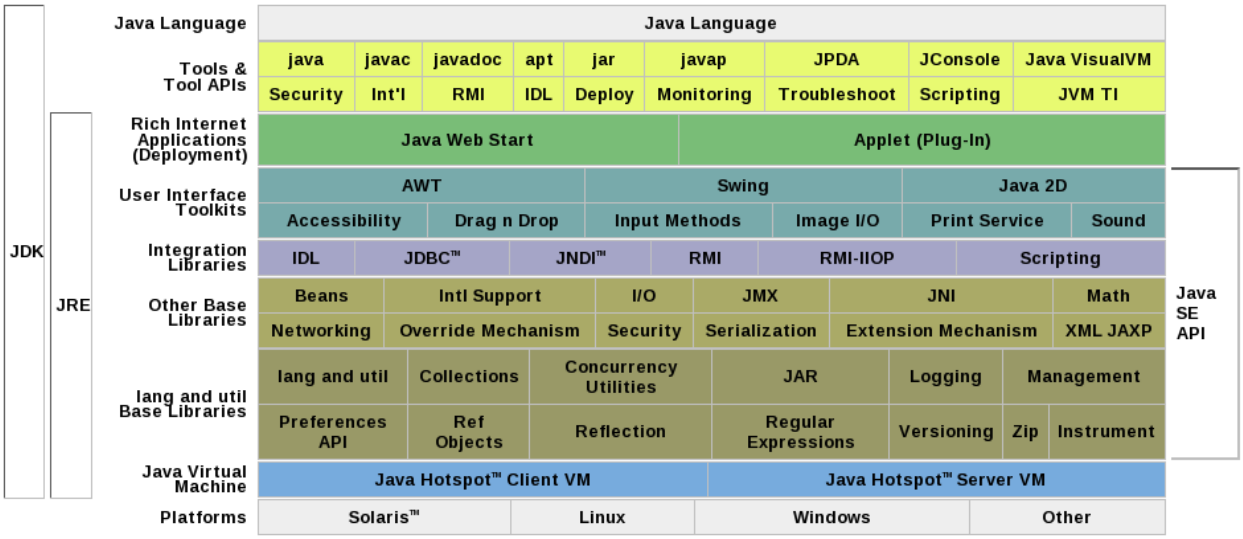
För att få en klarare överblick av hur programmeringsgränssnitten i Java är strukturerade skall jag härnäst presentera de olika delarna av Java-plattformen samt hur gränssnitten utvecklas.

Själva Java-språkspecifikationen beskriver syntaxen och semantiken samt primitiva datatyper (Gosling, Joy, Steele, och Bracha 2005, sid 1). Språkspecifikationen tillhör den första gruppen specifikationer av programmeringsgränssnitt enligt ISO/IEC modellen ovan. Detta innebär att definitionen endast omfattar hur gränssnitten kallas på och används, men inte själva gränssnitten och deras funktion.

En av de grundläggande delarna av Java plattformen är Java Virtual Machine (JVM) som i normala fall är målet för kompilering av Java kod (Gosling m.fl. 2005, sid 1). JVM är en abstrakt maskin som exekverar binär kod given i class filer som är oberoende av både hårdvara och operativsystem (Lindholm och Yellin 1999). Eftersom JVM enbart exekverar binär kod i class fil format och inte nödvändigtvis har någon information om Java-kod överhuvudtaget så uppfattas JVM ofta som en abstrakt del av de flesta användarna av Java-plattformen. Class fil formatet tillåter att andra programmeringsspråk än Java kan kompileras för att exekveras under JVM. Enligt ISO/IEC modellen tillhör JVM som gränssnitt den andra mera abstrakta gränssnittsspecifikationen som är oberoende av programmeringsspråk.

Det kanske mest synliga gränssnittet för programmerare är själva grundbiblioteken som till exempel java.lang och deras dokumentation men även andra bibliotek som är del av någon specifik java version. Utöver de bibliotek som är en del av själva java distributionen så existerar det otaliga utomstående paket och bibliotek vilka tillsammans utgör de gränssnitt som används för program skrivna i Java.

De tekniska gränssnittsspecifikationerna för Java utvecklas och modifieras genom Java Community Process (JCP) vilket är en formell process där medlemmar och expertgrupper samarbetar för att utveckla och revidera specifikationerna (Java Community Process 2009).



Figur 1: Sun Microsystems Java SE 6 plattform

Java plattformen är uppbyggd av diverse mera eller mindre abstrakta gränssnittsspecifikationer vilka implementeras som olika komponenter och tillsammans utgör helheten för en specifik Java plattform. Figur 1 illustrerar Sun Microsystems översikt av deras implementation av Java Standard Edition 6 (Java SE 6). Illustrationen visar sambandet mellan de olika gränssnitten samt hur de förhåller sig till varandra.

2.3 Java gränssnittens konkreta struktur

I detta avsnitt ges en kort översikt av de konstruktionerna inom java språket som är grunden för skapandet av olika gränssnitt för applikationer. Översikten tar enbart de aspekter i beaktande som är av betydelse för gränssnitt.

En av de avgörande faktorer som möjliggör att att java som ett språk kan på applikationsnivå dela gränssnitts specifikationer mellan olika implementationer är gränssnittskonstruktionen som nås genom nyckelordet interface. En interface deklaration konstruerar en referensimplementations typ vilket saknar en specifik implementation. Delar till interface typen kan tillhöra andra interface deklarationer, klasser , konstanter och abstrakta metoder. Ett interface kan utvidga ett eller flera andra interface (Gosling m.f.l. 2005, sid 259).

Klasser som konstrueras genom nyckelordet class definierar samtidigt en referens typ, likadant som interface men samtidigt även implementationen av referenstypen. En klass måste förklaras med nyckelordet public för att kunna hänvisas från andra paket. Alla klasser utöver Object klassen utvidgar en annan klass och kan även implementera ett interface ((Gosling m.f.l. 2005, sid 173)).

Program och bibliotek i java är organiserade som en serie av paket vilka är förklarade med nyckelordet package. Paketets innehåll till exempel metoder, objekt eller till och med andra paket är tillgängliga på från andra instanser enbart ifall de är förklarade med nyckelordet public . Namnen på enskilda fält eller objekt är lokala enbart till paketet de hör till, vilket innebär att andra paket med samma metoder, fält och objekt namn kan konstrueras. Paketens namn följer ett hierarkisk system där namnet för ett subpaket innehåller namnet för paketet det ingår i till exempel java.util.zip är ett paket för att behandla zip filer, vilket ingår i paketet java.util (Gosling m.f.l. 2005, sid 153-154).

Tillsammans utgör paketen,klasser och gränssnitt förklarade med nyckelorden package,class respektive interface de metoder som är tillgängliga för en java applikationsprogrammerare i syfte för att konstruera ett offentligt gränssnitt. I de flesta fallen så måste klassen och gränsnitten förklaras offentliga med nyckelordet public så att utomstående komponenter kan dra nytta av dem.

3. Orsaker som motiverar förändring av gränssnitt

I detta kapitel redogörs för allmänna orsaker till att ett offentligt programmeringsgränssnitt ändras eller modifieras med anknytning till mjukvara evolution. Mjukvara evolution och underhåll är breda begrepp med varierande definitioner (Chapin 2001. sid 7). I ljuset av detta presenteras kort mjukvara evolution och underhåll med gränssnitts evolution som intresse vinkling.

Evolutionen av gränssnitt i projekt där gränssnittet inte är ändamålet, utan enbart del av en större helhet är en del av det breda område som är underhåll av mjukvara som i sin tur kan vara till en stor grad påverkad av affärs intressen (Chapin 2001. sid 5). Enligt Ned Chapins (Chapin 2001, sid 20-21) så är mjukvara underhåll en mängd aktiviteter eller processer som antingen ändrar hur mjukvaran använder hårdvaran eller påverkar processer som är förknippade med mjukvaran eller ändrar på växelverkan med mjukvarans intressenter. Processer i detta sammanhang kan innefatta allt från testning till dokumentation, det vill säga det behöver inte beröra gränssnitt eller kod

överhuvudtaget. Till skillnad från mjukvara underhåll definierar Chapin (Chapin 2001, sid 21) mjukvara evolution som en tillämpning av mjukvara underhålls aktiviteter vilka leder till en ny version av mjukvaran med en del ändringar vilka påverkar användningen av mjukvaran .Chapin påpekar också att det kan förekomma att mjukvara underhåll och evolution används som synonymer.

Enligt Chapin (Chapin 2001, sid 12-13) hör typerna av mjukvara evolution och underhåll till två stycken huvudgrupper, affärs regler eller krav och mjukvara attribut. De tre typerna som tillhör affärsreglerna är utökande,korrigerande samt reducerande förändring, vilka främst beskriver den typen av förändring som resulterar i att funktionaliteten av mjukvaran ändras. Den andra gruppen, mjukvara attribut innehåller fyra typer: anpassande, prestandarelaterade, förebyggande eller förbättrande av den inre strukturen. Från gränssnitts evolutionsperspektiv kan båda grupperna vara grund för en förändring dock med den reservationen att en ändring av gränssnittet för en extern komponent innebär att ändringen hör till affärsreglerna, till skillnad från ifall komponenten skulle vara del av det interna systemet vilket skulle kunna innebära att ändringen skulle kunna vara av mjukvara attribut typ.

Gränssnitts evolution behöver inte enbart ske på grund av interna underhålls krav utan kan uppstå på grund av av externa faktorer som till exempel beroende på en utomstående komponent. Xing och Stroulia (Xing 2007, sid 1) påpekar att problemet gällande evolution av gränssnitt med applikationsprogram konstruerade med återanvända komponenter är att programmet och komponenterna följer en av varandra oberoende evolutionsbana. Följden från komponenternas utvecklingssynvinkel är att det måste eventuellt välja mellan att orsaka kompatibilitets problem med applikationsprogrammen eller låta bli att införa ändringar som skulle kunna förbättra övriga kvaliteten av komponenten. Xing och Stroulia påpekar att komponentutvecklarna vid en förändring möjligtvis riskerar att de måste underhålla flera olika versioner åtminstone tills applikationerna är förflyttade till att använda de uppdaterade komponenten. Vid en sådan förändring så är det sannolikt att det uppstår någon brytning i samverkan mellan komponenterna och applikationsprogrammen. En motivering för brytning är att det är orealistiskt att förvänta sig att komponentutvecklarna har möjlighet att uppdatera alla program som använder sig av föråldrade versioner av komponenter. Programutvecklare kan inte heller förväntas underhålla en egen version av en komponent även ifall källkod skulle vara tillgänglig, eftersom det inte längre skulle vara fråga om en återanvändbar komponent utan det skulle bli del av programmet (Xing 2007,sid 1).

Skillnaden mellan applikationsprogram och komponenter behöver inte alltid vara så skarp och i situationer där komponenten inte erbjuder någon önskad funktionalitet kunde det vara gynnsamt ifall applikationsprogrammerarna kunde implementera själv funktionaliteten i stället för att arbeta runt defekterna. Fowler (Fowler 1999. sid 86) kallar problemet för en ofullständig biblioteksklass och föreslår att problemet löses genom att använda en omstrukturerings teknik som införning av extern metod (Fowler 1999. sid 162).

4. Ändringar i gränssnitt och binär kompatibilitet

I detta kapitel genomgås vad begreppet binär kompatibilitet innebär för java applikationsprogram. Motiveringen för fokuset på applikationsprogram är på grund av att java språkspecifikationen är del av java plattformen, vilket kan leda till att en ny version av java plattformen för med sig ändringar i själva språket.

Ändring av gränssnitt med avseende på hur det påverkar tillbaka kompatibilitet , det vill säga hur program kompilerade för en äldre version fungerar på den nya kan enligt Dig med flera (Dig 2006. sid 88) klassificeras som ickebrytande gränssnitts förändring och brytande gränssnitts förändring. Brytande gränssnitts förändring är inte tillbaka kompatibel men förändringen behöver inte enbart ta form som kompilering eller länkningsfel utan kan även bero på ändrad funktionalitet. Ickebrytande förändring är tillbaka kompatibel och kan ta form i olika typer till exempel ökad prestanda eller nya moduler. Java språkspecifikationen (Gosling 2005,sid 339) definierar en typförändring vilket bevarar binär kompatibilitet som en ändring vilket inte ändrar på hur existerande program länkas, det vill säga om ett program kunde länkas mot den tidigare versionen så borde programmet även gå att länkas efter ändringen för att ändringen skall kunna klassificeras som binärt kompatibel.

Java har en del inbyggda mekanismer för att hantera förändringar i gränssnitt så att den binära kompatibilitet bibehålls. Mekanismerna eller metoderna i java språkspecifikationen (Gosling 2005. sid 333) är presenterade genom att definiera en mängd ändringar på paket och klasser vilka är tillåtna med tanke på tillbaka kompatibilitet. De exakta ändringsmetoderna presenteras närmare i kapitel fem, huvudbetoning vid det här fallet är att poängtera att en viss flexibilitet är inbyggd gällande samverkan mellan källkods kompatibilitet och binär kompatibilitet. Enligt java språkspecifikationen (Gosling 2005. sid 334) krävs det uttryckligen att en giltig java

implementation stöder dessa metoder. Metoderna är omfattande till sina funktioner till den grad att det till exempel är möjligt att transformera två olika gränssnitt så att de blir binärt kompatibla.

5. Tillbaka kompatibla gränssnitts ändringar

I detta kapitel presenteras ändringar vilka enligt java språkspecifikationen (Gosling 2005. sid 334) bibehåller binär kompatibilitet med tanke på gränssnits förändringar.

Omimplementering av redan existerande metoder eller konstruktörer i syfte med att öka prestandan ändrar inte typsignaturen. Det formella namnet på eventuella parametrar till metoder kan också ändras men ifall typen på parametrarna ändrar,retur värdet ändrar från void till någonting annat eller från någonting till void samt ifall namnet på själva metoden ändrar så resulterar det i en ändrad typsignatur. En ändrad typsignatur gällande metoder är ekvivalent med att ta bort metoden och sätta en ny metod till (Gosling 2005. sid 352). Metoder som förklarar en Throw sats blir inte påverkade ifall Throw sätts till eller tas bort på grund av att Throw satser enbart kontrolleras under kompileringsskede så det påverkar inte den binära kompatibiliteten (Gosling 2005. sid 354).

Tillsättande av nya fält, metoder eller konstruktörer till en redan existerande klass eller interface ändrar inte typsignatur. Metoder med samma namn som redan existerande metoder kan även tillsättas förutsatt att de är överladdade på så vis att de har en annorlunda metodsignatur. Vid kompileringsskede bestäms signaturen för de metoder som skall anropas vilket medför att de tidigare metoderna anropas till och med i fallet när mera specifika metoder sätts till, i sådana fall så måste programmet kompileras på nytt för att kunna använda de nya metoderna (Gosling 2005. sid 355). Med samma princip fungerar tillsättande av nya klasser eller interface i typhierarkin (Gosling 2005. sid 357).

Borttagning av fält,metoder och konstruktörer som är förklarade med nyckelordet private för en klass bevarar tillbaka kompatibiliteten. Eftersom dessa inte är synliga för andra klasser så går det inte att ha ett direkt beroende på dem (Gosling 2005. sid 343). Enligt samma princip så kan fält, metoder och konstruktörer av klasser och interface tas bort från ett paket när hela paketet blir uppdaterat.

Förflyttning av en metod uppåt i klasshierarkin bevarar kompatibiliteten. Förflyttningen i det här sammanhanget innebär att klassen eller interface som metoden tillhör har åtminstone en superklass eller superinterface som metoden blir förflyttad till. Mekanismen fungerar genom att att under exekveringen evaluera ifall metodsignaturen existerar i den lägsta klassen eller ifall någon av de direkta superklassernas metodsignatur är den som sökes (Gosling 2005. sid 343).

Omplacering av de direkta superinterface av en klass eller interface bevarar tillbaka kompatibiliten så länge som mängden av medlemmar av dem hålls detsamma (Gosling 2005. sid 341).

6. Vanliga orsaker till brytning i gränssnitt och omstrukturerings metoder för dem

I detta kapitel kommer det att presenteras vanliga former av ändringar i gränssnitt vilket resulterar i en icke tillbaka kompatibel ändring. Ändringarna representerar inte alla sett som gränssnits kompatibiliteten kan brytas utan är valda på basis av en fallstudie av Dig (Dig 2006. sid 85-87,92-103) som identifierade vanliga gränssnits brytningar i java projekt. Dig (Dig 2006. sid 85) observerade att över 80% av brytningarna var ett direkt resultat av omstrukturering av kod, med detta som motivering presenteras även exempel på metoder för omstrukturering som är kan uppstå som motivering för en dylik ändring.

6.1 Förändring som inte ändrar funktionaliteten

Förflyttning av en metod

Förflyttning av en metod i sammanhang av omstrukturering av kod (eng. move method) innebär att en metod flyttas från en klass till en annan. Motiveringen för att en metod flyttas till en annan klass kan vara till exempel kunde det vara frågan om att minska kopplingen mellan olika klasser. Lösningen för flyttande av metoder är att återskapa metoden i den nya klassen samt att systematiskt uppdatera alla referenser till den nya metoden (Fowler 1999. sid 142). Alla förflyttningar innebär dock inte att det uppstår en brytning, ifall metoden förflyttas uppåt i hierarkin som nämnt i kapitel fem så bevaras kompatibiliteten.

move field

deleted method

changed argument type

changed return type

replaced method call

renamed stuff

new hook method

extra argument

deleted class

extracted interface

method object

pushed down object

moved class

pulled up method

split package

split class

6.1 Förändring som ändrar själva beteende

new method contract

implemented new interface

changed events order

new enumeration constants

miscellanuous

7. Sammanfattning

Referenser:

Fowler, M. (1999). Refactoring: Improving the design of existing code,
Addison Wesley Longman, Inc. 1999.

Dig, D. och Johnsoon, R. (2005) The Role of Refactorings in API Evolution. IEEE International Conference on Software Maintenance, sid 389 -400.

Dig,D. och Johnsson, R. (2006) How do APIs evolve?: A story of refactoring .Journal of software maintenance and evolution, vol:18 nr:2 sid:83 -107.

Gharaibeh, B. (2007) Coping with API Evolution for Running, Mission-Critical Applications Using Virtual Execution Environment. Quality Software, 2007. QSIC '07. Seventh International Conference, sid 171 -180.

JTC 1 Sekretariatet (2009) Standing Document 2 Historical Background of JTC 1. Länk: http://isotc.iso.org/livelink/livelink/7852467/JTC_1_Standing_Document_2__SD_2__Historical_Background_of_JTC_1.pdf?func=doc.Fetch&nodeid=7852467. Hämtad 28.02.2010.

JTC 1 Sekretariatet (2007) ISO/IEC JTC 1 Directives, 5th Edition, Version 3.0,

Annex j : Guidelines for API standardization. Länk: http://isotc.iso.org/livelink/livelink/6901544/JTC_1_Directives__5th_Edition__Version_3.0?func=doc.Fetch&nodeid=6901544. Hämtad 28.02.2010.

Gosling,J. Joy, B. Steele, G. och Bracha, G. (2005) The Java™ Language Specifikation. Version 3. Addison-Wesley. 2005.

Harrison,W. (2009) Safer typing of complex API usage through java generics;
Proceedings of the 7th International Conference on Principles and Practice of Programming in Java, PPPJ sid:67 -75

http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf

Java Community Process (2009)JCP 2: Process Document,The formal procedures for using the Java Specification development process. Version 2.7. Länk: http://jcp.org/en/procedures/jcp2 . Läst 27.2.210.

Lindholm,T och Yellin, F (1999) The Java™ Virtual Machine Specification Second Edition, Chapter 3: The structure of the Java Virtual Machine. Länk: http://java.sun.com/docs/books/jvms/second_edition/html/Overview.doc.html. Hämtad 1.3.2010.

Types of software evolution and software maintenance. Källa: Journal of software maintenance and evolution [1532-060X] Chapin yr:2001 vol:13 iss:1 pg:3 -30

Zhenchang Xing och Stroulia,E (2007)IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 33, nr 12, December 2007

JSR14 http://www.jcp.org/en/jsr/detail?id=14

Förkortningar:

ISO - Internationella standardiseringsorganisationen

IEC - Internationella elektrotekniska ?

JCP - Java process community

JVM - Java Virtual Machine

Java SE 6 - Java Standard Edition 6

Figurer: