UML Reverse engineering

Other ref:
http://www.ibm.com/developerworks/rational/library/08/0610_xu-wood/

1.Introducere

Termenul de reverse engineering reprezinta un subiect destul de discutat si de cercetat in ultima perioada si in tot acest timp s-au dezvoltat o serie de tehnologii menite sa imbunatatesca si sa aduca un plus in acest domeniu.

Reverse engineering, numită și back engineering înapoi, este procesul prin care un obiect dezvoltat de om (program software) este deconstruit pentru a-și dezvălui proiectarea, arhitectura sau pentru a extrage cunoștințe din obiect; similar cu cercetarea științifică.

IEEE defineste astfel reverse engineering: "procesul de analiză a unui sistem pentru a identifica componentele sistemului și relațiile dintre ele și pentru a crea reprezentări ale sistemului într-o altă formă sau la un nivel superior de abstractizare".

Există două componente în reverse engineering: redocumentarea și recuperarea designului (proiectarii).

Redocumentarea este crearea unei noi reprezentări a codului dezvoltati, astfel încât să fie mai ușor de înțeles.

Recuperarea designului (proiectarii) este utilizarea de deductii sau raționamente de la cunoștințele generale sau experiența personală a dezvoltarii produsului cu scopul de a înțelege pe deplin funcționalitatea produsului.

Exista o serie de CASE-tool-uri disponibile pentru a aplica reverse engineering pe diagramele UML dar in majoritatea cazurilor aceste case-tool-uri ce suporta UML sunt limitate la a extrage diagrama claselor. In cazul programelor software scrise in Java, case-tool-urile pot afisa si ierarhiile pachetelor utilizate. Exista de asemenea si limitari deoarece diagrama claselor overa ajutor limitat in a intelege arhitectura software utilizata in programul software. Exista necesitatea ca aceste tool-uri pentru reverse engineering UML sa poata recunoaste de asemenea si design patternurile utilizate dar si diagrama claselor abstractizate.

In timp ce diagrama unei clase arata informatia statica la nivelul cel mai de jos in UML, cu toate acestea ea reprezinta o abstractizare a codului actual orientat pe obiecte al sistemului software. De asemenea, UML nu poate specifica modul in care modelul este implementat si nici nu exista o mapare unu-la-unu intre elementele din diagrama claselor si codul sursa. Asocierile sunt destul de greu de detectat, in special pentru limbaje scrise dinamic. Utilizarea claselor abstracte si a interfetelor in limbaje pur obiectuale difera mult de utilizarea lor in limbaje hibride ca C++ de exemplu.

Datorita diferentelor de concept la nivelul de design si nivelul de implementare, sunt necesare interpretari pentru extractia diagramei claselor din codul sursa iar in momentul de fata nu exista un mod standard de a face acest lucru. In plus, tool-urile nu permit utilizatorului sa influenteze interpretarile deoarece interpretarile codului sunt scrise in algoritmul tool-ului. Datorita acestei diferente de interpretare a codului intre tool-uri, duc la inconsistente, neintelegeri si limitari. Daca aceste case-tool-uri suporta round-trip engineering, aceeasi interpretare poate si ar trebui sa fie utilizata in mod consistent atat in reverse engineering cat si in generarea codului.

In timp ce aceste tool-uri care suporta generarea de cod permit utilizatorului sa influenteze conventiile de codare, este suprinzator ca personalizari similare sunt rar suportate atunci cand se aplica reverse engineering in diagrama claselor din codul sursa.

In aceasta lucrare, obiectivul este acela de a examina si a raporta un state-of-the-art legat de termenul de reverse engineering aplicat pe limbaje software pur obiectuale si o comparatie intre diferite tool-uri care aplica reverse engineering.

S-au ales doua case-tool-uri populare si doua prototipuri aflate in stadiul de research ce aplica reverse engineering pornind de la diagramele UML a codului sursa. Cele doua case-tool-uri examinate sunt Togheter si Rational Rose iar prototipurile sunt IDEA si FUJABA. Suportul de reverse engineer al acestor tool-uri a fost comparat prin utilizarea lor in a analiza sistemul software Mathaino scris in Java. In plus, diagrama claselor extrasa din tool-urile TOGHETER, ROSE si IDEA au fost stocate in UML 1.3/XML 1.1 format si importate intr-un tool de modelare numit TED, unde un sistem automat de comparare a modelelor foloseste un prototip aflat in stadiu de cercetare pentru manipularea si compararea modelelor UML.

2. Tooluri pentru examinare

2.1 TOGHETER

Programul TOGHETER suporta reverse engineering pentru sistemele software dezvoltate in C++, Java, C#, VB si multe altele. Atunci cand se doreste aplicarea reverse engineering pentru programe scrise in Java, TOGHETER construieste un tree view similar cu cel produs in ROSE dar in plus produce diagrama claselor UML in acelasi timp (presupunand ca utilizatorul a ales acest tip de diagrame pentru a incepe). TOGHETER de asemenea poate aplica reverse engineer asupra informatiei din codul sursa, fisierele .java, codul scris la nivel de bit (fisierele .class), fisiere .jar sau fisiere .zip impachetate dar modelele pe care le extrage nu sunt la fel pentru fiecare dintre aceste tipuri de fisiere.

TOGHETER poate de asemenea sa ofere un array of metrics asupra codului examinat si un audit asupra coding style-ului folosit.

2.2 Rational ROSE

Rational Rose software suporta reverse engineering pentru sisteme software dezvoltate in C++ si Java. Atunci cand se aplica reverse engineer asupra programelor scrise in Java, Rose construieste un tree view ce contine clase, interfete si asocierile gasite la nivelul cel mai inalt. Metodele si variabilele sunt imbricate in ownerul claselor.

Rose de asemenea construieste la cerere o reprezentare a diagramei claselor pentru informatiile extrase si genereaza un layout default pentru ele. In plus, Rose construieste in mod automat o ierarhie a pachetelor ca un tree view.

Rose poate sa aplice reverse engineering asupra informatiilor din codul sursa, codul byte, fisiere .jar sau pachete zip. Similar cu softul de la punctul 2.1 modulele de reverse engineering in Java pot oferi instructiuni fisierelor, subdirectoarelor , pachetelor si librariilor pentru a fi examinate.

2.3 FUJABA

Fujaba a fost dezvoltat incepand cu anul 1998 la Universitatea Padeborn. Suporta code generatio din diagrama claselor dar si diagrame de activitate, statechart-uri si diagrame de colaborare. Acesta permite utilizarea diagramelor UML ca un fel de limbaj de programare vizual pentru dezvoltarea aplicatiilor fara nici o codare manuala.

Fujaba are ca scop acela de a oferi suport round-trip engineer: daca un developer sau un alt tool modifica codul generat si daca aceste modificari au un anumit standard de codare, atunci Fujaba poate analiza schimbarile din cod si poate re-creea diagramele UML corespondente. Acestea acopera structurile statice adica diagrama claselor dar si structurile dinamice de exemplu metodele bodies.

2.4 IDEA

Tool-ul pentru reverse engineering IDEA a fost dezvoltat la Universitatea Bremen din Germania, in cadrul proiectului UML AID. Principalul obiectiv al IDEA este acela de a redocumenta programele Java utilizand diagramele UML, cu focus pe analiza statica a structurilor orientate pe obiecte utilizand diagrama claselor UML. Aceste sunt considerate in general cele mai folosite si cele mai intelese diagrame incluse in UML.

In ceea ce priveste IDEA, un metamodel pentru limbajul Java a fost dezvoltat. Modelul progamelor actuale examinate sunt stocate ca instante a structurilor de date corespondente metamodelului. A fost dezvoltat un framework de translatie pentru a creea modele UML din modelele Java, oferind o schema de translatie standardizata.

3. Studiu de caz

Pentru a realiza o examinare completa asupra acestor tool-uri si pentru a putea vedea performantele lor este nevoie de o examinare pe un cod destul de complex. In continuare se vor ilustra parametrii si proprietatile utilizate pentru examinare.

3.1 Numarul claselor (NOC)

Aceasta este o proprietate generala prin care se masoara dimensiunea modulului software, de exemplu a pachetelor Java. Valoarea parametrului NOC (Number Of Classes) poate fi numarat in diferite moduluri, depinzand in mare parte de modul in care interfetele sunt reprezentate si de modul in care clasele „speciale”(containere sau inner classes) sunt numarate. Asadar, un numar ridicat al claselor indica o reprezentare mai detaliata.

3.2 Numarul de asocieri (NOA)

Aceasta metrica masoara interconectivitatea dintre diferite clase in termeni de asocieri. Interpretarea valorii NOA necesita o grija suplimentara decat NOC. O valoare scazuta a NOA indica un algoritm de reverse engineering imprecis dar de asemenea poate fi si rezultatul unei abstractizari, de exemplu recunoasterea asocierilor inverse. Asadar, valori scazute sunt considerate mai bune iar NOA trebuie masurata atat inainte cat si dupa abstractizari.

3.3 Tipuri de asocieri

UML suporta diferite tipuri de asocieri ca : directa, bidirectionala, de compositie sau de agregare. In plus, intelesul de asociere poate fi modificat prin aplicarea unor tag-uri sau calificatori la sfarsitul lor. Se va masura care tipuri de asocieri au fost gasite si sub care conditii au fost incluse in reverse engineering.

3.4 Gestiunea interfetelor

O interfata este un specificator pentru operatii externe vizibile a claselor, componentelor sau altor clasificatori fara specificatii a structurii interne. In diagramele UML, interfetele sunt descrise ca si clasificatori rectangulari cu stereotipul <<Interface>> sau pur si simplu un cercuri. Interfetele sunt atasate de o sageata catre clasificatorii care le suporta. Acest lucru indica faptul

ca acele clase pot implementa toate operatiile din interfata. Cercurile sunt folosite cand operatiile interfetei sunt ascunse. O clasa care foloseste operatiile disponibile intr-o interfata, poate fi atasata de acel cerc printr-o sageata.

Din punct de vedere al reverse engineering, generarea acestor dependente este importanta pentru a intelege utilzarea interfetelor si pentru a evidentia dependentele.

De asemenea, moduri diferite de a gestiona interfetele pot avea un impact asupra metricii NOC sau asupra modului de citire a diagramei claselor.

3.5 Java Collection classe

Clasele Java Collections sunt clase menite sa gestioneze colectiile unor obiecte. In design, aceste feature-uri nu sunt vizibile dar sunt mai apoi modelate prin utilizarea unor asocieri ca multiplicitate sau calificatori.

Se vor examina modul in care reverse engineeringul va recunoaste clasele Java Collections si tipurile pe care le contin sau cum sunt gestionate ele ca si clase normale.

3.6 Multiplicitatile

In diagramele UML, prea multe asocieri intre obiecte sunt descrise in termenii unor multiplicitati. Informatia precisa despre multiplicitati este dificil de descris si necesita tehnici de analiza dinamica, care nu sunt suportate de tool-urile pe care le examinam.

3.7 Clasele inner

Java inner classes reprezinta o implementare specifica prn care se ascund definitiile unei clase in specificatiilor altei clase asadar recunoasterea claselor inner este importanta pentru a reflecta intreaga arhitectura a sistemului.

3.8 Class comportment details

Se va face o examinare a nivelului de detalii atunci cand se rezolva semnatura unei metode iar acest lucru este foarte important atunci cand se face o analiza completa a codului sursa din implementarea metodei.

4. Rezultate optinute cu tool-urile folosite

4.1.Rezultate CASE-tools (TOGHETER si Relation Rose)

4.1.1 Clasele

Pentru un program aplicat asupra celor doua tool-uri Rose a putut gasi un numar de 39 de clase, 42 de clase inner fara nume si anonime si de asemenea inca trei clase inner non-anonime si cu nume.Relation Rose modeleaza clasele inner la fel ca si orice alta clasa asadar numarul total de clasa rezultate a fost de 85.In cazul claselor inner fara nume , Rose genereaza numere in ordine numerica incepand de la 1 pentru a eticheta fiecare clasa inner nedenumita.

In cazul tool-ului Togheter, atunci cand se aplica reverse engineer asupra fisierelor .java , au fost recunoscute 39 de clase din pachetul core si inca alte trei clase inner cu nume. Ceea ce este interesant este ca atunci cand se aplica pe fisiere .class, Togheter a reusit sa recunoasca un numar de 45 de clase inner , atat anonime cat si cu nume. In ambele cazuri, clasele inner au fost afisat in diagrama ca parte din clasele de caer apartin, si nu in dreptunghiuri separate.

Compartimentul numelui pentru clasele asupra carora s-au aplicat reverse engineer de catre Relation Rose contine numele clasei actuale si numele pachetului. Atat atributul cat si compartimentul operatiilor contin numele, tipul si vizibilitatea (public,private protected) variabilelor si metodelor. Pentru fiecare metoda, tipul parametrului sunt date de asemenea.

Atat Relation Rose cat si Togheter atunci cand sunt aplicate pe fisiere .class pot identifica clasele pachetelor externe la cerere sau daca exista o relatie sau o referinta catre/de la pachetele analizate. Nu s-au calculat numarul claselor din pachetele externe

4.1.2 Interfetele

Relation Rose utilizeaza un cerc pentru a ilustra interfetele in diagrama claselor. Metodele abstracte din interfete sunt scrise in interiorul cercului, separate de doua linii orizontale, ceea ce nu reprezinta stilul recomandat de UML.Togheter ilustreaza acestea utilizand clase rectangulare in interiorul unui stereotip <<Interface>>.

Atat Relation Rose cat si Togheter au gasit patru interfete .Atunci cand se utilizeaza notatiile de tip cerc , Rose realizeaza acest lucru printr-o linie mai groasa.Atunci cand o clasa cu stereotipul <<Interface>> este folosita, ambele tool-uri utilizeaza o linie punctata cu un triunghi la sfarsit ce pointeaza catre interfata.

Nici Rose si nici Togheter nu au putut genera dependente intre interfete si clasele care le utilizeaza.Acest aspect reprezinta o limitare evidenta in a intelege rolul interfetelor.

4.1.3 Asocierile

Numarul total de asocieri gasite de Rose a fost de 83.In modelele Rose, relatie dintre o clasa parinte si clasa inner aferenta este specificata printr-o asociere fixa numita This$().45 din asocierile modelate au fost generate pe baza unei relatii de tip inner class – owner

class.Celelalte 38 de asocieri modeleaza relatiile intre o clasa si tipurile de variabile pe care le defineste.In aceste asocieri, doar sfarsitul asocierii care este conectat la o clasa ce reprezinta tipul variabilei ii se atribuie un role name. Numele de rol este denumit dupa variabila in sine. In toate situatiile asocierile sunt directionate.

Togheter nu a reusit sa extraga nici o asociere atunci cand a fost aplicat pe fisierele .class. Atunci cand a fost aplicat pe fisierele .java, au fost descoperite 16 asocieri. Aceste asocieri sunt directionale dar nu specifica nici un rol.

4.1.4 Clasele Java Collection

Atat Relation Rose cat si Togheter pot gestiona similar clase container.

4.2 FUJABA

In general, Fujaba ofera un foarte flexibil mecanism de reverse engineer, care de exemplu permite utilizatorilor sa defineasca pattern-uri specifice pentru a detecta anumite elemente de design pattern de nivel superior. Din pacate, acest lucru nu este inca aplicat pentru mecanismele de reverse engineer basic cum ar fi detectia asocierilor.

4.2.1 Clasele

Fujaba a reusit sa identifice clasele top-level si toate clasele inner denumite.Totusi, Fujaba ignora clasele inner anonime deoarece acestea nu sunt folosite pentru a construi mai departe programul software prin forward enginner.

Fujaba reuseste sa sa gaseaca toate clasele non-primitive care sunt folosite ca tip de atribut.Clasele utilizate ca parametru, ca tipuri returnate sau pentru variabile locale sunt afisate optional.

4.2.2 Interfetele

Clasele interfete sunt afisate la fel ca si clasele uzuale dar cu stereotipul <<Interface>>.

4.2.3 Role names

Numele de rol sunt derivate din identificatorii atributelor non-primitive si sunt afisate la ultima asociere din asocierile directionale. Atunci cand se aplica un merge pe asocierile directionale, ambele nume de rol sunt luate in considerare, rezultand astfel asocieri nedirectionate cu numele de rol la ambele capete.

4.2.4. Clasele Java Collections.

Hujaba utilizeaza o lista adaptabila cu clase container pre-definite.Atributele acestui tip sunt automat examinate pentru tipul lor de intrare prin cautarea utilizarii metodelor add.

4.2.5 Multiplicitati.

Avand in vedere considerentele conceptuale, Fujaba nu suporta bounds-uri de multplicitate atat pentru asocieri de tipul to-one, asocieri pentru asocieri to-many cat si pentru reverse si forward engineer.Aceste atribute non-primitive sunt afisate utilizand multiplicitati 0...1 si atribute de container cu tipuri de intrare identificate sunt afisate folosind multiplicitati 0...n.

4.2.6 Clasele inner

Clasele inner ce au nume sunt afisate la fel ca si clasele uzuale in diagrama claselor. Daca se aplica, optiunea de a afisa numele pachetelor contine clasa si/sau numele metodei. Pana acum clasele inner anonime au fost ignorate.

4.2.7. Detaliile compartimentelor claselor

Toate detaliile din compartimentele claselor UML cum ar fi : numele clasei, atributele si metodele sunt suportate la nivelul de detaliu de implementare. Pentru atribute, vizibilitatea, identificatorul, tipul dar si o multiplicitate optionala sunt afisate. Semnatura metodelor include identificatori si tipuri de parametri. Odata cu cresterea diagramei claselor, compartimentul atributelor si a metodelor pot fi resetate la default daca se depaseste oa numita dimensiune.

4.3 Idea

Cand mecanismul de reverse engineering a inceput sa foloseasca pachetul de baza Mathaino cu IDEA, mai multe etape de transformare au fost aplicate nivelului de baza a modelului UML, pentru a recunoaste proprietatile modelului analizat. Descrierile acestor pasi sunt incluse in urmatoarele sectiuni.

4.3.1. Calcularea metricii

Pentru calcularea metricii a NOC, au fost numarate atat numarul de clase din pachetul de baza mathiano cat si numarul total de clase legate de el (ex. Prin asocieri UML).

In total au fost descoperite 42 de clase: 39 de clase plain si 3 named (unice) clase inner. Au fost descoperite 45 de clase inner, dar clasele inner anonime nu au fost considerate. In plus, s-au

descoperit 35 de clase externe, rezultand intr-un numar total de 77 de clase. Numarul initial de asocieri a fost de 56, insa numarul a fost redus dupa fiecare pas al transformarii, rezultand intr-un NOA de 47.

4.3.2. Manipularea interfetelor

Pentru interfete este folosita reprezentarea default UML metamodel ca subclase a Classifier. In pachetul de baza au fost gasite 4 interfete si inca 6 interfete externe au fost folosite.

4.3.3. Role names

Role names sunt derivate din identificatorii atributelor non-primitive si sunt afisate la sfarsitul asocierii obiectivului a asocierilor directate.

4.3.4. Manipularea claselor Java Collection

La nivelul implementarii, containerele sunt afisate ca si clase individuale. Pentru a dezvalui adevarata relatie dintre un obiect sursa si obiectele depozitate intr-un container, un proces de rezolutie este angajat, care analizeaza codul sursa pentru acces la interfata obiectelor din container.

Analizand pachetul de baza Mathaino, 16 atribute de tip container au fost gasite, toate fiind vectori. 15 dintre acestea au putut fi rezolvate. Doua nu au putut fi reprezentate grafic din cauza faptului ca tipul obiectelor continute a fost String, care nu este reprezentat ca o clasa individuala in diagrama, ci tratat ca un atribut primitiv. Un atribut de tip container nu a putu fi rezolvat deloc

4.3.5. Merging Inverse Associations

Au fost descoperite 7 perechi potentiale de asocieri inverse. Toate au fost unice, asa ca s-a hotarat sa se imbine (merge) toate acestea. Au fost gasite 3 tipuri de inverse: Intre clase independente, Intre clasa si clasa inclusa si Intre clasa si clasa inner.

4.3.6. Agregare si compozitie

Conform documentelor de specificatii UML, agregarea este o trasatura informala, care nu poate fi caracterizata intr-un fel precis, cum ar fi necesar pentru a putea fi recunoscuta din codul sursa. In UML, compozitia defineste constrangerile legate de instantele claselor. Pentru o recunoastere corecta a compozitiilor este necesara o tehnica de analiza dinamica, care nu este inca suportata de IDEA.

4.3.7. Multiplicitatile

Urmatoarele valori de multiplicitate si domenii sunt descoperite de instrumentul IDEA: ’0..1’ (elemenul tinta este intializat undeva in codul sursa), ’1’ (elementul tinta este initializat la crearea obiectului), ’*’ (reprezinta o clasa container: ex. un set sau o colectie).

4.3.8. Clase Inner

IDEA recunoaste clasele inner non-anonime prin parsarea de byte code. Acestea sunt reprezentate ca si clase conventionale, utilizand conventia Java de numire pentru numele claselor inner (UML neincluzand acest concept).

4.3.9. Detaliile clasei Compartment

Toate detaliile clasei Compartment standard UML (ex: numele clasei, atribute si metode) sunt suportate la nivelul detaliului implementarii. Pentru atribute, vizibilitatea, indetificatorul, tipul si o multiplicitate optionala pentru tablouri (arrays) sunt afisate.

5.Analizele rezultatelor

Au fost comparate rezultatele instrumentelor utilizate in 2 categorii diferite: concepte de baza si avansate. Conceptele de baza se refera la elementele de baza UML ca la clase si asociatii. A doua categorie evalueaza capacitatea instrumentelor de a genera o reprezentare mai abstracta.

Toate instrumentele examinate au reusit sa recunoasca trasaturile de baza UML cum sunt: clasele, interfetele si asociatiile. Numai intr-un singur caz, Together a esuat sa recunoasca o parte a asociatiilor plain.

La comparatiile avansate s-a putut vedea urmatorul lucru: capabilitatiile de reverse engineering a instrumentelor industriale nu merg mult mai departe de trasaturile de baza UML. Reprezentarile mai abstracte si recunoasterea trasaturilor mai avansate sunt clar de domeniul instrumentelor de cercetare.

6. Concluzii

Reverse engineering a modelelor UML pentru sistemul de software orientat pe obiecte poate fi efectuat in conformitate cu doua principii: afland cat mai multe informatii despre sistemul

subiect si modelandu-l cumva folosind modele UML sau tintind la nivelul design modele care sunt populate cu informatia codului sursa de cate ori este convenabil.

Rezultatele au aratat ca desi toate instrumentele ofera functionalitati de reverse engineering de incredere, numai prototipurile in curs de cercetare ofera algoritmi pentru analize avansate.

7.Bibliografie

1. http://en.wikipedia.org/wiki/Reverse_engineering

2. people.auc.ca/xu/present/reverse.ppt

3. http://cs.gmu.edu/~duminda/classes/spring04/State_Of_Art.pdf