Validarea metricilor de proiectare a sistemelor de programe orientate pe obiecte

Lect. Marian CRISTESCU
Universitatea "Lucian Blaga" Sibiu
marian.cristescu@ulbsibiu.ro

Object oriented development requires not only a different approach to design and implementation, it requires a different approach to software metrics. Since object oriented technology uses objects and not algorithms as its fundamental building blocks, the approach to software metrics for object oriented programs must be different from the standard metrics set. Some metrics, such as lines of code and cyclomatic complexity, have become accepted as "standard" for traditional functional/ procedural programs, but for object-oriented, there are many proposed object oriented metrics in the literature.
Keywords: *Metrics, Object-Oriented Software Development, software reliability, Cyclomatic Complexity, Error Prediction, C++ Programming Language.*

1 Introducere

Dezvoltarea sistemelor de programe mari este o activitate care consuma timp si resurse si chiar daca gradul de automatizare a activitatilor din domeniul dezvoltarii software a crescut, resursele reprezinta înca o limitare importanta. De aceea, trebuie sa se asigure informatii corecte si elemente de "ghidaj" managerilor, pentru a-i ajuta în procesul de luare a deciziilor, în planificarea si programarea activitatilor, si alocarea resurselor pentru diferitele activitati care au loc în timpul procesului de dezvoltare software.

În ultimul deceniu, multe companii au început sa introduca tehnologia OO în mediile lor de dezvoltare software. Metodele de proiectare sau analiza OO, limbajele de programare OO si mediile de dezvoltare sunt elemente cunoscute în toata lumea, atât în organizatiile software mai mici, cât si în cele mai mari. Introducerea tehnologiei OO în industria de software, a creat noi provocari pentru companiile care folosesc metricile pentru produs ca instrument pentru monitorizare, control si îmbunatatire a modului de dezvoltare sau întretinere a softului. De aceea, metricile care reflecta specificitatea paradigmei programarii OO trebuie definite si validate pentru a fi folosite în industria de software. Astfel, s-a ajuns la concluzia ca metricile "*traditionale*" pentru produs nu sunt suficiente pentru caracterizarea, evaluarea si predictia calitatii sistemelor software orientate pe obiecte.

2. Studiu de caz

2.1. Fundamentul teoretic

Chidamber si Kemerer au efectuat un studiu [CHK1], al carui scop a fost de a analiza, pe cale experimentala, metricile de proiectare a software-ului OO si pentru a evalua daca aceste metrici au sau nu succes în predictia probabilitatii de detectare a claselor cu erori. Din definitia acestor metrici rezulta ca nu sunt independente de limbaj. De exemplu, pentru a reflecta specificitatea limbajului C++, câteva dintre metricile prezentate de Chidamber si Kemerer trebuie sa fie usor "ajustate":

- *Importanta(semnificatia) metodelor dintr-o clasa* (WMC – Weighted Methods per Class) - masoara complexitatea unei clase individuale. Daca se considera ca toate metodele unei clase sunt la fel de complexe, atunci WMC exprima numarul metodelor definite în fiecare clasa. Cu toate acestea, WMC este definit ca fiind "*numarul tuturor functiilor membre si a operatorilor definiti în fiecare clasa* " [CHK1].
- *Adâncimea arborelui de mostenire al unei clase* (DIT – Depth of Inheritence Tree of a class) - este definita ca adâncimea maxima a grafului de mostenire a fiecarei clase. Limbajul C++ permite mosteniri multiple si

drept urmare clasele pot fi organizate în grafuri directionate aciclic în loc de arbori. În anumite cazuri se considera ca DIT masoara "numarul de stramosi ai unei clase".

- *Numarul de succesori ai unei clase* (NOC – Number Of Children of a Class) – exprima numarul descendentilor directi pentru fiecare clasa. Clasele cu un numar mare de urmasi sunt dificil de modificat si solicita, în general, mai multe testari datorita faptului ca este posibil ca respectiva clasa sa-si afecteze toti succesorii. Cu toate acestea, o clasa cu numerosi urmasi trebuie sa asigure servicii într-un numar mai mare de contexte si trebuie sa fie mai flexibila. Este de asteptat ca acest fapt sa introduca un grad mai mare de complexitate în modul de proiectare al clasei.
- *Cuplarea între clasele de obiecte* (CBO – Coupling Between Object classes) - O clasa este cuplata cu o alta daca foloseste functiile sale membre si/sau variabilele de instanta. CBO exprima numarul de clase cu care este cuplata clasa specificata. Ipoteza care sta la baza acestei metrici este ca, în general, *cu cât o clasa este cuplata cu mai multe clase de obiecte, cu atât respectiva clasa este mai predispusa la erori.* Asadar, cuplarea între clase ar trebui determinata pentru a concentra procesul testare si/sau verificarile facute asupra acestor clase.
- *Raspunsul dintr-o clasa* (RFC – Response For a Class) - determina numarul de metode care este posibil sa fie executate ca raspuns la un mesaj primit de catre un obiect al respectivei clase. RFC exprima, de asemenea, si numarul de functii invocate, în mod direct, de catre functiile membre sau operatorii unei clase. Ipoteza care sta la baza acestei metrici specifica urmatorul fapt: *cu cât setul de raspunsuri al unei clase creste, creste si gradul de complexitate al clasei, aceasta fiind mai predispusa la erori si, în mod implicit, este mai dificil de modificat.*
- *Lipsa de coeziune a metodelor* (LCOM – Lack of Cohesion On Methods) – specifica numarul de perechi de functii membre care nu împart variabile de instanta din care se scade numarul de perechi de functii membre care împart variabile de instanta. Oricum, metrica este setata la valoarea 0 în orice situatie în care scaderea, mai sus amintita, produce un rezultat negativ. O clasa cu o coeziune scazuta între metodele sale sugereaza un mod de proiectare necorespunzator, lucru care este, probabil, cauza multor erori (de exemplu: încapsularea obiectelor nerelationate ale programului si a functiilor membre, care nu trebuie sa fie împreuna).

2.2. Descrierea studiului de caz

Pentru a valida pe cale experimentala metricile orientate pe obiecte propuse de catre Chidamber si Kemerer în [CHK1] tinând cont de capacitatea lor de a realiza predictii referitoare la probabilitatea de aparitie a defectelor, a fost efectuat un studiu pe parcursul a patru luni (din septembrie pâna în decembrie, 2001). Populatia studiata a fost reprezentata de mai multe sisteme de programe realizate de studentii sectiei de Matematica-Informatica din Universitatea "Lucian Blaga" Sibiu. Studentii au fost grupati, la întâmplare, în 8 echipe. Fiecare echipa a dezvoltat un sistem, de marime medie, pentru controlul informatiei care sa permita "dirijarea" activitatilor desfasurate într-un centru de închiriere a casetelor video si mentinerea bazelor de date ale clientilor si ale casetelor video.

Procesul de dezvoltare a fost efectuat în conformitate cu modelul secvential al ciclului de viata din domeniul ingineriei software, derivat din modelul cascada. Acest model include urmatoarele faze: analiza, proiectarea, implementarea, testarea si întretinerea. La sfârsitul fiecareia dintre aceste faze au fost sintetizate informatiile specifice si s-a încercat alcatuirea unui "document": documentul de analiza, documentul de proiectare, anexele continând codul, rapoartele referitoare la erori, si în final anexele care contin modificarile efectuate în codul sursa.

Specificatiile cerute si documentele de proiectare au fost validate în sensul de a verifica faptul ca se potrivesc cerintelor sistemului. Erorile descoperite, în aceste prime doua faze, au fost comunicate studentilor. Acest lucru s-a dorit a fi facut pentru a mari sansele ca implementarea sa înceapa cu o analiza, respectiv, un concept de proiectare OO corecte. Faza de testare a fost realizata de catre

un grup independent compus din programatori cu mai multa "experienta" în dezvoltarea de software. Acest grup a testat toate sistemele prin planuri de teste similare si tehnici functionale de testare. În timpul fazei de depanare, studentilor li s-a cerut sa corecteze sistemul lor bazându-se pe erorile descoperite de grupul de testare independent.

Metoda OMT (Object Modelling Technique), a fost folosita în timpul fazei de proiectare, iar limbajul de programare C++, mediul de dezvoltare software GNU, si biblioteca OSF/MOTIF au fost folosite în timpul implementarii. Ca platforma de implementare au fost folosite statii IBM-PC pe care a rulat Windows95/98. Mediul de dezvoltare si tehnologia folosite sunt reprezentative pentru ceea ce este utilizat, în prezent, în industrie si în mediul academic. Studentilor li s-a facut recomandarea sa foloseasca urmatoarele biblioteci:

a) *MotifApp* - Aceasta biblioteca furnizeaza un set de clase C++ pentru manipularea ferestrelor, dialogurilor, meniurilor, etc., modul ei de utilizare fiind descris, în [YOU1]. Biblioteca *MotifApp* ofera posibilitatea utilizarii unor mici "instrumente" de tip OSF/MOTIF printr-un stil de proiectare si programare orientat pe obiecte.

b) *Biblioteca GNU* - Aceasta biblioteca provine din mediul de programare C++ GNU. Contine functii pentru manipularea sirurilor, fisierelor, listelor etc.

c) *Biblioteca bazelor de date C++* - Aceasta biblioteca pune la dispozitie o implementare C++ a arborilor de tip B multi-indexati.

2.3. Colectiile de date

Au fost colectate date despre: codul sursa al programelor C++ obtinute la sfârsitul fazei de implementare; date despre aceste programe; date despre erorile descoperite în timpul fazei de testare si eliminate în timpul fazei de depanare; codul sursa al programelor C++ depanate, rezultate la sfârsitul întregului ciclu.

Pentru a extrage metricile de design orientat pe obiecte, propuse de Chidamber si Kemerer, în mod direct din codul sursa al programului obtinut la sfârsitul fazei de implementare, a fost folosita tehnica descrisa în [DEV1]. Pentru colectarea datelor de la punctele (2) si (3), au fost folosite doua formulare, care au fost alese dintre cele descrise în [HEL1]: formularul pentru raportarea defectelor; formularul pentru descrierea originii componentelor.

- **Formularul pentru raportarea defectelor** a fost utilizat pentru a colecta datele despre: defectiunile depistate în cadrul fazei de testare; clasele schimbate pentru a corecta anumite defectiuni; efortul efectuat de echipa de corectie. Acesta din urma include:
  - durata necesara pentru determinarea precisa a schimbarilor ce se impun. Aceasta include efortul care este necesar pentru întelegerea schimbarilor sau gasirea cauzelor erorilor, determinarea locurilor unde sunt necesare astfel de schimbari si determinarea faptului ca toate efectele schimbarii au fost luate în considerare;
  - durata necesara pentru implementarea corectiilor. Aceasta include schimbarile de design, modificarile codului, testarea regresiei si actualizarea documentelor.
- **Formularul pentru descrierea originii componentelor** a fost utilizat pentru a tine evidenta informatiilor care caracterizeaza fiecare clasa, în parte, în timpul dezvoltarii proiectului când acesta intra în faza de configurare. Mai întâi, acest formular este folosit pentru identificarea cazului în care o clasa a fost dezvoltata de la început sau din prin reutilizarea altei clase. În cazul din urma, s-au colectat informatii referitoare la modificarile efectuate: *fara modificari*, *putine* (când se modifica mai putin de 25 % din cod), sau *extensiv* (când se modifica mai mult de 25% din cod).

Aceste informatii au fost necesare pentru a satisface cerintele sistemului si ale designului, aici fiind incluse si numele claselor reutilizate. Clasele reutilizate fara modificari au fost catalogate: *reutilizate textual(literal)*. În plus, numele sub-sistemului caruia îi apartinea clasa a fost, de asemenea, retinut. Au fost întâlnite trei tipuri de sub-sisteme: interfata grafica utilizator (GUI), interfata textuala utilizator (TUI), si procesarea bazei de date (DB).

3. Analiza rezultatelor

În cadrul experimentului s-a încercat estimarea, aproximativa, a faptului ca metricile de design OO, definite în [CHK1], sunt utile estimarilor referitoare la clasele predispuse la erori. Acest lucru se doreste a fi utilizat în evaluarea acestor metrici ca indicatori ai calitatii si pentru determinarea modului cum pot fi comparate metricile comune ale codului. S-a încercat sa se puna la dispozitie o modalitate de validare empirica, fapt considerat ca fiind necesar înaintea oricarei încercari de a folosi astfel de metrici, ca indicatori obiectivi si primari ai calitatii.

Tabelul 1 prezinta statisticile descriptive, comune, ale distributiilor metricilor, iar distributiilor metricilor OO analizate se bazeaza pe 180 de clase prezente în sistemele studiate. Aceste rezultate indica ierarhiile de mostenire care sunt oarecum uniforme (DIT), iar clasele au, în general, putini urmasi (NOC). În plus, majoritatea claselor arata o lipsa de coeziune (LCOM) apropiata de 0. Aceasta ultima metrica nu realizeaza o diferentiere prea buna a claselor ceea ce poate împiedica modul sau de definire, si nu permite nici o masura negativa.

Tabelul 1. Statisticile descriptive ale metricilor OO analizate

	WMC	DIT	RFC	NOC	LCOM	CBO
Maxim	99.00	9.00	105.00	13.00	426.00	30.00
Minim	1.00	0.00	0.00	0.00	0.00	0.00
Median	9.50	0.00	19.50	0.00	0.00	5.00
Mediu	13.40	1.32	33.91	0.23	9.70	6.80
Deviatia standard	14.90	1.99	33.37	1.54	63.77	7.56

Statisticile descriptive sunt utile pentru interpretarea rezultatelor analizei, în plus, ele vor facilita actiunile de comparare a rezultatelor cu viitoare studii similare.

Bibliografie

[AMA1] – Amadeus Software Research, *"Getting Started with Amadeus"*, Amadeus Measurement System, http://www.amadeus/syste~/uk/cs/docs.htm

[BRI1] - Briand L., Morasca S., Basili V., *"Defining and Validating High-Level Design Metrics"*, CS-TR-3301, University of Maryland, College Park, MD, 20742, 1999

[CHK1] - Chidamber S.R., Kemerer C.F., *"A Metrics Suite for Object-Oriented Design"*, IEEE Transactions on Software Engineering, 20(6), 1998, pp.476-493;

[DEV1] - Devanbu P., *"GENOA/GENII - A Customizable Language and Front-End Independent Code Analyzer"*, in Proceedings of the 14th International Conference on Software Engineering, Melbourne, Australia, 1992, pp.234-261;

[HEL1] - Heller G., Valett J., Wild M., "*Data Collection Procedure for the Software Engineering Laboratory (SEL) Database*", SEL Series, 1998, SEL-98-002;

[HOS1]– Hosmer D., Lemeshow S., *"Applied Logistic Regression"*, Wiley-Intersciences, 1999, pp.56-92

[LIH1] - Li W., Henry S., *"Object-oriented Metrics that Predict Maintainability"*, Journal of System and Software, 23(2), 1998, pp. 111-122

[YOU1] - Young D.A. "*Object-Oriented Programming with C++ and OSF/MOTIF*", Reference Manual, Prentice-Hall, 1992.