# Arbore de calitate bazat pe strategii de detecţie

Ciprian-Bogdan Chirilă
chirila@cs.utt.ro

18 iunie 2002

**Rezumat**

Împreună cu creşterea dimensiunilor aplicaţiilor software a crescut şi importanţa calităţii software-ului. Pentru a urmări calitatea software-ului, tehnologia pentru a specifica şi evalua calitatea produsului software este de importanţă vitală. Este nevoie de un model care să evalueze calitatea sistemelor software.

În aceast articol vom prezenta un model de arbore de calitate bazat pe strategii de detecţie şi un instrument software care implementează modelul şi măsoară calitatea sistemelor software industriale realizate în tehnologia programării orientate-obiect. Arborele de calitate este un model alcătuit din strategii de detecţie şi funcţii matematice care are ca scop automatizarea procesului de detecţie a calităţii software-ului.

## 1 Introducere

Un obiectiv principal al ingineriei software este acela de a îmbunătăţi calitatea produselor software. Vom realiza acest lucru printr-un mecanism de măsurare a calităţii unui sistem software pornind de la codul sursă al acestuia.

## 2 Arbore de calitate bazat pe strategii de detecţie

Ne propunem să automatizăm măsurarea calităţii software-ului prin intermediul arborelui de calitate bazat pe strategii de detecţie.

Abordarea se bazează pe articolul domnului Marinescu [2, Marinescu2001a] care se referă la detecţia carenţelor de proiectare bazate pe metrici şi teza sa de doctorat care are ca temă interpretarea mă sură torilor în sisteme orientate obiect [1, Marinescu2002]. Abordarea are la bază **analiza statică a codului sursă** a sistemului software. Mecanismul de măsurare a calităţii nu lucrează direct pe codul sursă ci exploatează un metamodel care este obţinut pe baza acestuia.

## 2.1 Calitate, factor de calitate

Abordarea calităţii este una decompoziţională, abordare pe care o găsim şi în arborele lui McCall şi a lui Boehm. Ea este mai apropiată de ideea lui McCall pentru că păstrăm noţiunea de factor în schimb noţiunea de criteriu se metamorfozează în strategie de detecţie. Avantajul acestei metamorfoze aceea că prin eliminarea criteriilor care vor conduce la folosirea unor anume metrici, fără a putea interpreta concludent rezultatele şi introducerea strategiilor de detecţie vom putea verifica dacă este respectată paradigma programării orientate-obiect.

Formal, un arbore de calitate poate fi descris de relaţiile următoare:

$$Calitate := f_F(Factor_1, Factor_2, Factor_3, \cdots, Factor_n)$$

$$Factor_i := f_{SD}(StrategieDeDetectie_1, StrategieDeDetectie_2,$$

$$StrategieDeDetectie_3, \cdots, StrategieDeDetectie_n)$$

## 2.2 Strategie de detecţie

Strategia de detecţie a unei carenţe poate fi definită în felul următor: [2], [3] expresie cuantificabilă a unei reguli prin care entităţile afectate de carenţa respectivă să poată fi automat detectate. Pornim de la următoarea definiţie formală bazată pe metrici:

$$S := {M_1}^{O_1} * {M_2}^{O_2} * \cdots * {M_n}^{O_n}$$

$$* := \cup \quad | \quad \cap \quad | \quad \setminus$$

Implementarea conceptului de strategie de detecţie este ilustrată prin regulile următoare:

```
DetectionStrategy     := StrategyDefinition SymbolsDefinition

# reguli pentru definirea unei strategii de detectie
```

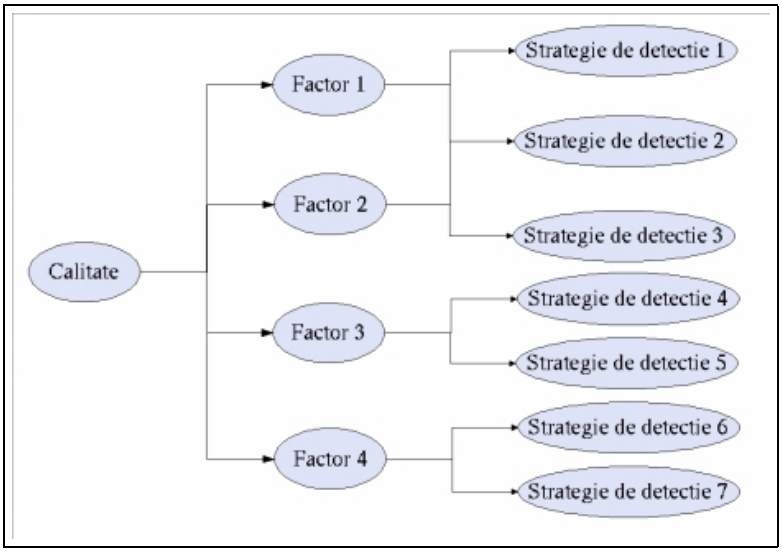


Figura 1: Arbore de calitate bazat pe strategii de detecţie - schemă de principiu

```
StrategyDefinition    := StrategyName ":=" DetectionRule ";"
DetectionRule         := MetricWithOutliers |
                         ComposedDetectionRule
MetricWithOutliers    := "(" MetricName "," OutlierName ")"
ComposedDetectionRule := DetectionRule CompositionOperator
                         DetectionRule
StrategyName          := [A-z][A-z0-9_]
CompositionOperator   := "or" | "and" | "butnotin"

# simbolurile sunt definitii de metrici si operatori
SymbolsDefinition     := MetricDefinition |
                         OutlierDefinition

# reguli pentru definirea metricilor
MetricDefinition   := MetricName  ":=" SqlQuery ";"
MetricName         := [A-z][A-z0-9_]
SqlQuery           := [.]   # SELECT <Entity> <Value>

# reguli pentru definirea operatorilor statistici
OutlierDefinition := OutlierName ":=" OutlierType
                     "(" OutlierParameter ")" ";"
OutlierType       := "TopValues" | "BottomValues" |
```

```
                        "HigherThan"| "LowerThan" | "BoxPlots"
OutlierName       := [A-z][A-z0-9_]
OutlierParameter  := [0-9][0-9,][%]
```

O strategie de detecţie încapsulează principiile paradigmei programării orientate-obiect cuantificând măsura în care ea este respectată într-un anumit sistem software.

## 2.3 Operatori şi funcţii matematice elementare

Funcţiile matematice $f_F$ şi $f_{SD}$ folosite în definiţia arborelui de calitate au rolul de a lega între ele calitatea, factorii şi strategiile de detecţie. Din punct de vedere matematic $f_F$ şi $f_{SD}$ vor fi funcţii compuse din funcţii matematice elementare. Putem abstractiza şi operatorii aritmetici prin intermediul funcţiilor:

$$f = f_F \quad | \quad f_{SD}$$

$$f = f_{e_1} \circ f_{e_2} \circ f_{e_3} \circ \cdots \circ f_{e_n}$$

$$f_{e_i} = + \quad | \quad - \quad | \quad * \quad | \quad / \quad | \quad min \quad | \quad max \quad | \quad avg$$

## 2.4 Tabela de calificative şi punctaje

Tabela de calificative şi punctaje este un mecanism de mă surare prin în cadrare a gradului de calitate la unul din calificativele disponibile. Formal o tabelă de calificative poate fi descrisă astfel:

$$Tabela = \{Calificativ_1, Calificativ_2, \cdots, Calificativ_n\}$$

$$Calificativ_i = (LimitaInferioara, LimitaSuperioara, PunctajAcordat)$$

Fiecărei trăsături a calităţii (calitate, factor, strategie de detecţie) i se va asocia o astfel de tabelă.

**Exemplu de utilizare şi interpretare** Considerăm următoarea tabelă de calificative:

```
ScoreMap
{
  EXCELLENT 0 5 10,
  GOOD 5 10 8,
```

```
  ACCEPTABLE 10 15 6,
  POOR 15 +oo 0
}
```

Tabela de faţa conţine 4 calificative pe care le interpretăm în felul următor: trăsătura de calitate poate fi excelentă dacă au fost detectate între 0 şi 5 entităţi suspecte. Unui astfel de calificativ se acordă conform tabelei 10 puncte. Celelalte calificative se vor interpreta analog.

## 2.5 Implementarea modelului de arbore de calitate

Gramatica care implementează conceptul de arbore de calitate bazat pe strategii de detecţie o vom nota cu SDQT - *Stratetgy driven Quality Tree*. Setul de reguli după care se poate construi un arbore de calitate se poate descompune în trei mari secţiuni după cele 3 trăsături de calitate considerate:

- reguli referitoare la calitate
- reguli referitoare la factori
- reguli referitoare la strategii.

Pe langă aceste reguli mai sunt şi alte reguli mai de detaliu:

- reguli referitoare la tabela de calificative
- reguli referitoare la parametrizarea strategiilor de detecţie
- reguli referitoare la parametrizarea funcţiilor matematice implementate.

**Reguli referitoare la trăsăturile de calitate** Aceste reguli ne spun cum se compune calitatea pe baza factorilor, sau factorii pe baza strategiilor. Regulile modelează funcţiile compuse $f_F$ şi $f_{SD}$ aplicată pe factori respectiv pe strategii de detecţie. Sunt modelate reguli care permit crearea de expresii matematice bazate pe operatori aditivi şi multiplicativi cu respectarea ordinii execuţiei operatiilor. Vom analiza o bucată din gramatica SDQT la nivelul definirii calităţii:

```
# Reguli pentru definirea expresiei calitatii
...
```

```
 03 QualityMathematicalExpressionExpression :=
QualityMathematicalTermExpression
QualityMathematicalExpression2Expression

 04 QualityMathematicalExpression2Expression :=
AdditiveOperator QualityMathematicalExpressionExpression | eps

 05 QualityMathematicalTermExpression :=
QualityMathematicalFactorExpression
QualityMathematicalTerm2Expression

 06 QualityMathematicalTerm2Expression :=
MultiplicativeOperator QualityMathematicalTermExpression | eps

 07 QualityMathematicalFactorExpression :=
FactorExpression | Number |
QualityStatisticalFunctionCallExpression |
( QualityMathematicalExpressionExpression )

 08 FactorExpression :=
Identifier

 09 QualityStatisticalFunctionCallExpression :=
StatisticalFunction ( QualityParameterListExpression )
 ...
```

În regulile listate mai sus putem remarca expresia calităţii bazată pe termeni, termenii reprezentând factori, factorii care pot fi apeluri de funcţii matematice. Aceleasi reguli apar şi la celelalte două nivele dar cu mici diferenţe sensibile.

## 2.6 Exemplu de arbore de calitate

Arborele de calitate propus spre studiu este creionat în figura 2:

Descrierea în gramatica SDQT unui astfel de arbore este prezentă pe liniile următoare:

```
# QualityTree-2.0-sample
# 09.06.2002
```

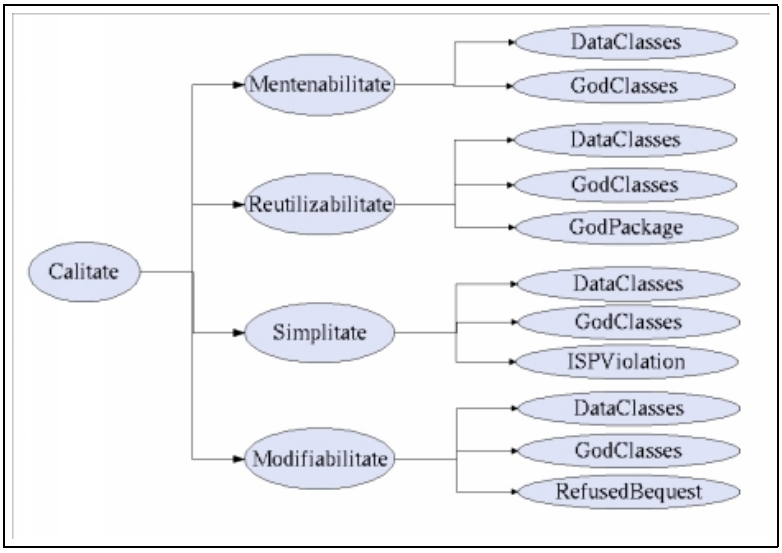


Figura 2: Arbore de calitate bazat pe strategii de detecţie - exemplu

```
# Quality Section
Quality :=
avg(Maintainability, Reusability,
    Understandability, Modifiability)
{
   EXCELLENT   8 10 10,
   GOOD 6 6 8,
   FAIR 4 6 6,
   POOR 0 4 4
};

# Factor Section
Maintainability :=
avg(DataClasses(ScoreMap1,(0,0,0)),
    GodClasses(ScoreMap1,(0,0,0)))
{
  EXCELLENT 8 10 9,
  GOOD 6 8 7,
  ACCEPTABLE 4 6 5,
  POOR 0 4 2
},

Reusability :=
```

```
avg(DataClasses(ScoreMap2,(0,0,0)),
    GodClasses(ScoreMap2,(0,0,0)),
    GodPackage(ScoreMap2,(0,0,0)))
{
  EXCELLENT 8 10 9,
  GOOD 6 8 7,
  ACCEPTABLE 4 6 5,
  POOR 0 4 2
},

Understandability :=
avg(DataClasses(ScoreMap1,(0,0,0)),
    GodClasses(ScoreMap1,(0,0,0)),
    ISPViolation(ScoreMap2,(0,0,0)))
{
  EXCELLENT 8 10 9,
  GOOD 6 8 7,
  ACCEPTABLE 4 6 5,
  POOR 0 4 2
},

Modifiability :=
avg(DataClasses(ScoreMap1,(0,0,0)),
    GodClasses(ScoreMap2,(0,0,0)),
    RefusedBequest(ScoreMap2,(0,0,0)))
{
  EXCELLENT 8 10 9,
  GOOD 6 8 7,
  ACCEPTABLE 4 6 5,
  POOR 0 4 2
};

# Strategy ScoreMap Section

ScoreMap1
{
  EXCELLENT 0 5 10,
  GOOD 5 10 8,
  ACCEPTABLE 10 15 6,
  POOR 15 +oo 0
},
```

```
ScoreMap2
{
  EXCELLENT 0 5 9,
  GOOD 5 10 7,
  ACCEPTABLE 10 15 5,
  POOR 15 +oo 1
};
```

Vom descrie puţin semantica acestui arbore de calitate simplificat justificând alegerile făcute pentru strategiile de detecţie.

Pentru început vom analiza *mentenabilitatea*. Mentenabilitatea este factor de calitate care se referă la toate atributele unui sistem software de care depinde efortul de a face modificări specificate [5]. În acest sens am optat pentru două strategii de detecţie: DataClasses şi GodClasses, care descoperă nerespectarea principiului paradigmei orientate-obiect conform căreia atributele şi comportamentul unui obiect trebuiesc declarate în acceaşi clasă.

Reutilizabilitatea este factor definit prin atributele unui sistem software care se referă la efortul de a reutiliza părţi din acesta. Pentru a atinge acest aspect vom apela din nou la cele două strategii DataClasses şi GodClasses, dar vom mai opta şi pentru strategia GodPackage care detectează pachetele mari în care locuiesc foarte multe clase, şi care sunt foarte intens utilizate de clienţii acestora. Strategia în capsulează principiul conform căruia clasele care au legături logice între ele sau sunt în relaţie trebuiesc împachetate împreună. Evident că reutilizabiliatea este afectată de o astfel de în călcare a principiului enunţat anterior.

Facilitatea de înţelegere a codului poate fi considerată a fi influenţată de separarea comportamentului de date la nivelul claselor dar şi de nerespectarea principiului segragării interfeţelor. Conform acestui principiu pentru fiecare client al unui obiect va exista o interfaţă prin care clientul va avea acces la serviciile obiectului. Un obiect cu multiple funcţionalită ţi va putea fi folosit cu restricţie în funcţie de context.

Modifiabilitatea unui sistem software o reprezintă setul de atribute pe care se bazează efortul de a aduce modificări de funcţionalitate sistemlui. Am propus utilizarea strategiilor: DataClasses, GodClasses, RefusedBequest. Ultima strategie detectează clasele care au o relaţie de moştenire forţată în urma necesităţii de a adăuga comportament sau stare unui obiect. Relaţia de moştenire ar putea fi înlocuită cu o relaţie de compoziţie nemaifiind astfel afectată modifiabilitatea sistemului.

# 3 Concluzii

Utilizând instrumentul software câştigăm:

- *abordare sistematică* prin descrierea modelului de calitate în limbajul SDQT
- *repetabilitate* bazată pe faptul că arborele de calitate poate fi reutilizat, parametrii strategiilor pot varia, dar arborele rămâne aceeaşi
- *scalabilitate* asigurată de instrumentul software - PRODEOOS cât şi de rezultatele obţinute pe sisteme industriale.

# Bibliografie


[1] R. Marinescu *Measurements Interpretation in Object-Oriented Systems A Goal-Driven Approach.*
Ph. D Thesis, "Politehnica" University Timişoara, 2002 (va fi publicată).

[2] R. Marinescu. *Detecting Design Flaws via Metrics in Object-Oriented Systems.* Proceedings of the TOOLS-USA 39, ISBN 0-7695-1251-8, IEEE Computer Society, 2001.

[3] R. Marinescu. *Design Flaws and Detection Strategies.*
PhD Presentation, Karlsruhe, April 2001.

[4] C.B. Chirilă. *Automatizarea detecţiei carenţelor de proiectare în sisteme orientate-obiect.* "Politehnica" University Timişoara 2001.

[5] ISO/IEC 9126 *Information technology - Software product evaluation - Quality characteristics and their guidelines for their use*

[6] N.E. Fenton, S.L. Pfleeger. *Software Metrics, A Rigourous & Practical Approach.*
PWS Publishing Company, Boston, 1997.

[7] R. Martin. *Design Principles and Patterns.*
http://www.objectmentor.com, 2000.