Design pattern-enabled object-oriented metrics

Khashayar Khosravi

Séminaire GÉLO – 2004/02/18

Département d'informatique et de recherche opérationnelle
Université de Montréal

Outline

- Why do we need design patterns?
- Are they theoretical or practical?
- Our methodology to get an answer
- Our problems
- Conclusion

Why do we need design patterns?

- Why do some programmers write code
  - 10 times faster than others?
  - That executes 10 times faster than others?
  - That has 1/10 as many bugs as others?

  [Keutzer, 2003]

- One thing expert designers know not to do is solve every problem from first principles
  - Rather reuse solutions that have worked for them in the past
  - When they find a good solution they reuse it again and again, such experience makes them experts

  [E. Gamma *et al.*]

Design patterns

- Help software developers
  - Choose design alternatives
  - Document and maintain
  - Standardize terminology
  - Make object-oriented software more
    - Reusable
    - Flexible
    - Modular
    - Understandable

Question

- Are design patterns only theoretical or are they practical too?

Problems with design patterns

- Difficult to learn design patterns just by reading the “design patterns book”

  [B. Wydaeghe, K. Verchaeve, B. Michiels, B. V. Damme, E. Arckens, and V. Jonckers]

- Some programs written with design patterns are too complicated and error prone

  [M. tatsubori and S. Chiba]

- Programs with comments are more understandable when you use design patterns

  [M. tatsubori and S. Chiba]

- Some object-oriented design patterns are more complicated

  [G. Baumgarter,K. Laufer, and F. Russo]

B. Wydaeghe, K. Verchaeve, B. Michiels, B. V. Damme, E. Arckens, and V. Jonckers

- Editor
  - 50,000 lines of code
  - 173 classes
  - Developed, installed on different platforms
- Design patterns
  - Model-View-Controller
  - Observer
  - Visitor
  - Iterator
  - Bridge
  - Façade
  - Chain of Responsibility

B. Wydaeghe, K. Verchaeve, B. Michiels, B. V. Damme, E. Arckens, and V. Jonckers

			Understandability		
	Modularity	Flexibility	Expert	Layman	Reusability
MVC	+	+	+	-	++
Observer	0	+	0	-	+
Visitor	0	+	0	-	+
Iterator	0	+	0	-	+
Bridge	+	+	+	+	++
Facade	+	+	+	+	++
Chain of respon-sability	-	+	0	+	0

Peter Wendorf

- 50 programmers
- Code Size
  - 800 KLOC C++
  - 200 KLOC PL/SQL
- Design patterns
  - Proxy
  - Observer
  - Bridge
  - Command
- 8 years of experience in software engineering

Peter Wendorf

- Dilemma of design patterns
  - Proxy
    - Use it for more flexibility, access control, and performance but in most cases these future needs never materialized
    - Increase the number of classes (and usually files) $\Rightarrow$ Increases size and complexity
    - Overload preprocessing and post-processing for requests to classes
    - Make debugging much harder
  - Bridge
    - Change is so hard
    - Without good documentation, understanding is not possible
  - Command pattern
    - Additional function that are never required
    - Requirement was about 100 elementary operations in sequences on a set of data that should be read from a database, but with command pattern architecture grew into a very complex software

Peter Wendorf

- Cost of removal
  - Proxy
    - Remove 3 out of 7 proxy patterns, reduce size by 200 LOC out of 3000 LOC
  - Observer
    - New design and implementation of the GUI
  - Bridge
    - Merge two classes in one
    - Remove 2 out of 3 bridge patterns, reduce size by 190 out of 1400 LOC

To be, or not to be?

```
public class Welcome {
   public static void main(final String args[]) {
        System.out.println(“Hello World!”);
   }
}
```

Questions

- How can we judge design patterns?
- Are problems from design patterns themselves or from misusing design patterns?

Measurement

- Identify the object
- Each classes of objects have two types of attribute
  - Internal
  - External
- Measure characterized in two ways
  - Direct measure (modularity)
  - Indirect measure (maintainability)

Software Quality Models

- Quality models define software qualities as a hierarchy
  - Quality Factors: Represent a behavioral characteristics (Maintainability)
  - Quality Characteristic: Attribute of a quality factor that relates to software production and design (Analyzability)
  - Quality Metrics: Measure of some aspect of a quality characteristic (N_STMTS)

Our methodology (1/2)

- Evaluate quality characteristic models
  - Find the best definitions for metrics
  - Find the right values for each metrics (numerical result)
  - Find the right metrics for each quality characteristic
  - Find the best characteristics for each quality factors

Our methodology (2/2)

- Evaluate design patterns
  - Measure quality characteristic of different types of design patterns
  - Assess flexibility, reusability, modularity, and understandability of programs implemented with design patterns
  - Measure quality characteristic of programs with and without design patterns
  - Ultimately compare the results of both measurements

Tools for quality evaluation

- Quality factors
  - Standard factors (6 definitions by ISO)
  - Others
- Quality characteristic of software systems
  - Standard characteristic (21 definitions by ISO)
  - Others
- Metrics: Tools to evaluate characteristic of software
  - Standard metrics (30 metrics by ISO)
  - Others

Control Graph Metrics (McCabe)

Label	mnemonic	Format	Min	max
Number of edges	N_EDGES	I	1	50
Number of nodes	N_NODES	I	1	50
Cyclomatic number	VG	I	1	50
Number of entry nodes	N_IN	I	1	1
Number of exit nodes	N_OUT	I	1	1
Essential complexity	ESS_CPI	I	1	1
Design complexity	DES_CPX	I	1	10

Textual metrics (Halstead)

Label	Mnemonic	Format	Max	Min
Total operand occurrences	TOT_OPND	I	1	152
Different operators	DIFF_OPND	I	1	38
Total operators occurrences	TOT_OPTR	I	2	198
Different operators	DIFF_OPTR	I	2	18
Program length	PR_LGTH	I	3	350
Vocabulary size	VOC_SZ	I	3	58
Program size	PR_SZ	F	2:00	274:48
Program Volume	PR_VOL	F	4.75	2032.58
Intelligence content	INTELL	F	4.75	1255.98
Estimated number of errors	N_ERRORS	F	0.00	0.95
Program level	PR_LVL	F	0.027	1.00
Program difficulty	PR_CPXTY	F	1.00	36.00
Mental effort	EFFORT	F	4.75	73172.68
Program time	CODE_T	F	0.26	4065.15
Language level	LANG_LVL	F	1.56	1255.98

Control graph metrics (others)

Label	Mnemonic	Format	Max	Min
Number of unconditional jump (GOTO)	UNCOND_JUMP	I	0	0
Number of exits of condition structures	COND_STRUCT	I	0	1

Basic count

Label	Mnemonic	Format	Max	Min
Number of statements	N_STMTS	I	0	50
Number of comments	N_COM	I	0	10

Call metrics

Label	Mnemonic	Format	Max	Min
Direct called components	DRCT_CALLS	I	0	9

User defined metrics

Label	Mnemonic	Format	Max	Min
Number of nested level	N_NEST = MAX_LVLS – 1	I	0	4
Vocabulary frequency	VOC_F = PR_LGTH / VOC_SZ	F	1.00	4.0
Average size of statements	AVG_S = PR_LGTH / N_STMTS	F	3.00	7.00

Others

- Correctness
  - Traceability
  - Completeness
  - Consistency
- Reliability
  - Consistency
  - Accuracy
  - Error tolerance
- Efficiency
  - Execution efficiency
  - Storage efficiency
- Integrity
  - Access control
  - Access audit

ISO

- Reliability
  - Suitability
  - Accuracy
  - Interoperability
  - Security
- Efficiency
  - Time behavior
  - Resource behavior

Others

- Usability
  - Operability
  - Training
  - Communicativeness
- Maintainability
  - Simplicity
  - Conciseness
  - Self descriptiveness
  - Modularity
- Testability
  - Simplicity
  - Instrumentation
  - Self descriptiveness
  - Modularity

ISO

- Usability
  - Understandability
  - learnability
  - Operability
- Maintainability
  - Analyzability
  - Changeability
  - Stability
  - Testability

Others ISO

- Flexibility
  - Self descriptiveness
  - Expendability
  - Generality
  - Modularity
- Portability
  - Self descriptiveness
  - Software system independence
  - Machine independence
- Reusability
  - Generality
  - Modularity
  - Software system independence
  - Machine dependence
- Interoperability
  - Modularity
  - Communications commonality
  - Data commonality

- Functionality
  - suitability
  - accuracy
  - interoperability
  - security
- Portability
  - Adaptability
  - Installability
  - Conformance
  - Replaceability

Definition of quality

- Maintainability
  - Analyzability
    - 25*VG + 25*N_STMTS + 25*AVG_S
  - Changeability
    - 25*AVG_S + 25*N_NEST + 25*UNCOND_JUMP + 25*VOC_F
  - Stability
    - 25*N_IN + 25*N_OUT + 25*DRCT_CALLS + 25*UNCOND_JUMP
  - Testability
    - 25*COND_STRUCT + 25*N_NEST + 25*UNCOND_JUMP + 25*VG
- Value rate
  - EXCELLENT: 99 - 100
  - GOOD: 66 - 99
  - FAIR: 33 - 66
  - POOR: 0 - 33

Theoretical problems

- Different definitions for
  - Metrics
  - Quality characteristics
  - Quality factors
- Difficult to find the best definitions for quality factors

Theoretical problems

Henderson-Sellers LCOM*

Consider a set of m methods, $M_1, M_2, \ldots, M_m$

The methods access a data attributes, $A_1, A_2, \ldots, A_a$

Let $a(M_k)$ = number of attributes accessed by method M_k

Let $m(A_k)$ = number of methods that access data A_k

Then

$$LCOM^* = \frac{\left(\frac{1}{a}\sum_{j=1}^{a} m(A_j)\right) - m}{1 - m}$$

If each method accesses all attributes then $m(A_k) = m$ so

$$LCOM^* = \frac{\left(\frac{1}{a}\sum_{j=1}^{a} m\right) - m}{1 - m} = \frac{(m) - m}{1 - m} = 0$$

If each method accesses only one attribute and a different attribute then we have:

$$LCOM^* = \frac{\left(\frac{1}{a}\sum_{j=1}^{a} m\right) - m}{1 - m} = \frac{\left(\frac{1}{a}a\right) - m}{1 - m} = 1$$

So at maximum cohesion LCOM* = 0

At Henderson-Sellers' "minimum cohesion" LCOM* = 1

Lack of Cohesion (LCOM)

-Measures the cohesion or lack of a class; evaluate the dissimilarity of methods in a class by instance variables or attributes.

-LCOM is measured by counting the number of pairs of methods that have no attributes in common, minus the number of methods that do. A negative difference corresponds to LCOM value of zero.

-Low cohesion is a sign of high complexity, and shows that the class can be subdivided. High cohesion indicates simplicity and high potential for reuse.

Example:

Device
type:int reading:int mode: boolean
compute(x:int,y:int):int update(a: int):int test(t:int)

```
Class Device {
  int reading, type;
  boolean mode=false;

  public int update (int a) {return a + reading; }
  public int compute(int x, int y) {return x*y*type - reading;}
  public void test (int t) { if t ==1  mode=true;}
}
```

LCOM(Device)= 2 – 1 = 1

Lack of Cohesion in Methods (LCOM):

Definition: The number of different methods within a class that reference a given instance variable.

Facts:

- Encapsulation is promoted by the use of cohesive methods within a class.
- Classes that should be split into smaller classes are ones that have a lack of cohesion.
- Flaws in the design of classes can be identified by any measure of disparateness of methods.
- Errors are more likely to occur in the development process when there is complexity caused by low cohesion.

How to figure this out:

Find the number of methods that are in a class that reference a given instance variable by searching from the root node of the class to the last child.

Chidamber and Kemerer (1993) LCOM

Let C be a class, with methods $M_1, M_2, \ldots, M_m$

Let I_k = the set of instance variables used by method M_k

Let $P = \{ (I_k, I_j) \mid I_k \text{ and } I_j \text{ do not intersect} \}$

Let $Q = \{ (I_k, I_j) \mid I_k \text{ and } I_j \text{ do intersect} \}$

If $|P| > |Q|$ then

$LCOM = |P| - |Q|$

else

$LCOM = 0$

Practical problems

- Combination of metrics to implement one characteristics
- Some characteristics can be evaluated by experts only
- Implementation of quality characteristics with standard metrics
- Implementation of metrics

Example: Practical problems

Definition of characteristics

- **Usability (ISO)**
  - **Operability**
  - **Training**
  - **Communicativeness**
- **Usability (McCall)**
  - **Understandability**
  - **Learnability**
  - **Operability**

- **Portability (ISO)**
  - **Adaptability**
  - **Installability**
  - **Conformance**
  - **Replaceability**
- **Portability (TRW)**
  - **Device independence**
  - **Completeness**
- **Portability (McCall)**
  - **Self descriptiveness**
  - **Software system independence**
  - **Machine independence**

Example: Practical problems

ISO/IEC TR 9126-2:2003(E) External Suitability metrics

Functional adequacy

X = 1 – A / B

A = Number of functions in which problems are detected in evaluation

B = Number of functions evaluated

$0 \leq X \leq 1$

(The closer to 1.0, the more adequate)

Functional implementation completeness

X = 1 – A / B

A = Number of missing functions detected in evaluation

B = Number of functions described in requirement specifications

$0 \leq X \leq 1$

(The closer to 1.0, the more adequate)

From that point on…

- Select the best metrics
- Evaluate quality characteristics with the metrics (and experts)
- Find best model for design patterns
- Implement enough programs with and without design patterns
- Evaluate design pattern quality characteristics for all design patterns

Conclusion

- Why do we need design patterns?
  - It is depend on what we are doing…
- Are they theoretical or practical?
  - It is depend on the type of software and models of design patterns
- Our methodology to get the answer
  - Apply quality characteristics models on design pattern models
  - Develop a suite of programs with and without design patterns and measure their quality characteristics
- Our problems
  - What are the best definitions for
    - Software quality factors
    - Quality characteristic
    - Software metrics

References

- Design patterns elements of reusable Object Oriented software,
  E. Gamma, R. Helm, R. Johnson, J. Vlissides
- Characteristics of software quality,
  B. W. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. J. Macleod & M. J. Merrit
- McCabe & Associates,
  www.mccabe.com
- K. Keutzer, UC Berkeley,
  http://www-inst.eecs.berkeley.edu/~cs169/sp03/downloads/lectures/
- M. Tatsubori & S. Chiba, University of Tsukuba, Japan
  http://citeseer.nj.nec.com/tatsubori98programming.html
- F. A. Rabhi, University of New South Wales,
  http://citeseer.nj.nec.com/598519.html
- G. Baumgarter & K. Laufer & V. F. Russo, Purdue University & Loyola University,
  http://citeseer.nj.nec.com/baumgartner96interaction.html
- B. Wydaeghe, K. Verchaeve, B. Michiels, B. V. Damme, E. Arckens & V. Jonckers, Universiteit Brussel,
  http://citeseer.nj.nec.com/288953.html
- L. Tahvildari & A. Singh, University of Waterloo,
  http://www.swen.uwaterloo.ca/~ltahvild
- Midwest Quality Co.,
  http://www.midwestquality.com

Thank you for your attention

- Any Comments?
- Any Question?