ANALYSE DE : “PLAYING ROLES IN DESIGN PATTERNS: AN EMPIRICAL DESCRIPTIVE AND ANALYTIC STUDY”

PAR BILLY BOUCHARD

Plan

- Infos
- Articles Connexes menant à l'article
- Implémentations
- Métriques intéressantes
- Résultats
- Validités et Biais – ma critique
- Autre Recherche intéressantes
- Application dans le jeu vidéo
- Conclusion personnelles importantes

Informations

- Papier par : Foutse Khomh, Yann-Gaël Guéhéneuc, et Giuliano Antoniol
- Paru dans : Proc. ICSM 2009, Edmonton, Canada
- Repsecte les norms IEEE

ARTICLES CONNEXES – HANNEMANN ET KICZALES

DESIGN PATTERN IMPLEMENTATION IN JAVA AND ASPECTJ

- PROGRAMMATION PAR ASPECT
- DÉFINISSENT OBSERVE DIFFÉRENTES PROPRIÉTÉ DES PATRONS AFIN DE CONNAITRE CERTAINS ASPECT DE LEUR MAINTENANCE

Table 1. Design pattern, roles, and desirable properties of their AspectJ implementations

	Modularity Properties				Kinds of Roles	
Pattern Name	Locality(**)	Reusability	Composition Transparency	(Un)pluggability	Defining(*)	Superimposed
Façade	Same implementation for Java and AspectJ				Façade	-
Abstract Factory	no	no	no	no	Factory, Product	-
Bridge	no	no	no	no	Abstraction, Implementor	-
Builder	no	no	no	no	Builder, (Director)	-
Factory Method	no	no	no	no	Product, Creator	-
Interpreter	no	no	n/a	no	Context, Expression	-
Template Method	(yes)	no	no	(yes)	(AbstractClass), (ConcreteClass)	(AbstractClass), (ConcreteClass)
Adapter	yes	no	yes	yes	Target, Adapter	Adaptee
State	(yes)	no	n/a	(yes)	State	Context
Decorator	yes	no	yes	yes	Component, Decorator	ConcreteComponent
Proxy	(yes)	no	(yes)	(yes)	(Proxy)	(Proxy)
Visitor	(yes)	yes	yes	(yes)	Visitor	Element
Command	(yes)	yes	yes	yes	Command	Commanding, Receiver
Composite	yes	yes	yes	(yes)	(Component)	(Composite, Leaf)
Iterator	yes	yes	yes	yes	(Iterator)	Aggregate
Flyweight	yes	yes	yes	yes	FlyweightFactory	Flyweight
Memento	yes	yes	yes	yes	Memento	Originator
Strategy	yes	yes	yes	yes	Strategy	Context
Mediator	yes	yes	yes	yes	-	(Mediator), Colleague
Chain of Responsibility	yes	yes	yes	yes	-	Handler
Prototype	yes	yes	(yes)	yes	-	Prototype
Singleton	yes	yes	n/a	yes	-	Singleton
Observer	yes	yes	yes	yes	-	Subject, Observer

ARTICLES CONNEXES – BIEMAN ET MCNATT

COUPLING OF DESIGN PATTERNS: COMMON PRACTICES AND THEIR BENEFITS

- LES FORCES ET FAIBLESSE DE L'UTILISATION DE PLUSIEURS PATRON DE CONCEPTION
- QUELS PATRONS SONT LES PLUS SOUVENT COUPLÉ

Table 1. The frequency of occurrences of design patterns in the referenced papers.

Pattern	Pattern Group		
	Creational	Structural	Behavioral
Factory Method	6		
Abstract Factory	7		
Builder	3		
Prototype	1		
Singleton	6		
Adapter		4	
Bridge		2	
Composite		8	
Decorator		2	
Facade		3	
Flyweight		2	
Proxy		5	
Chain of Responsibility			2
Command			4
Interpreter			2
Iterator			3
Mediator			5
Memento			1
Observer			11
State			4
Strategy			8
Template			6
Visitor			4
Totals	23	26	50

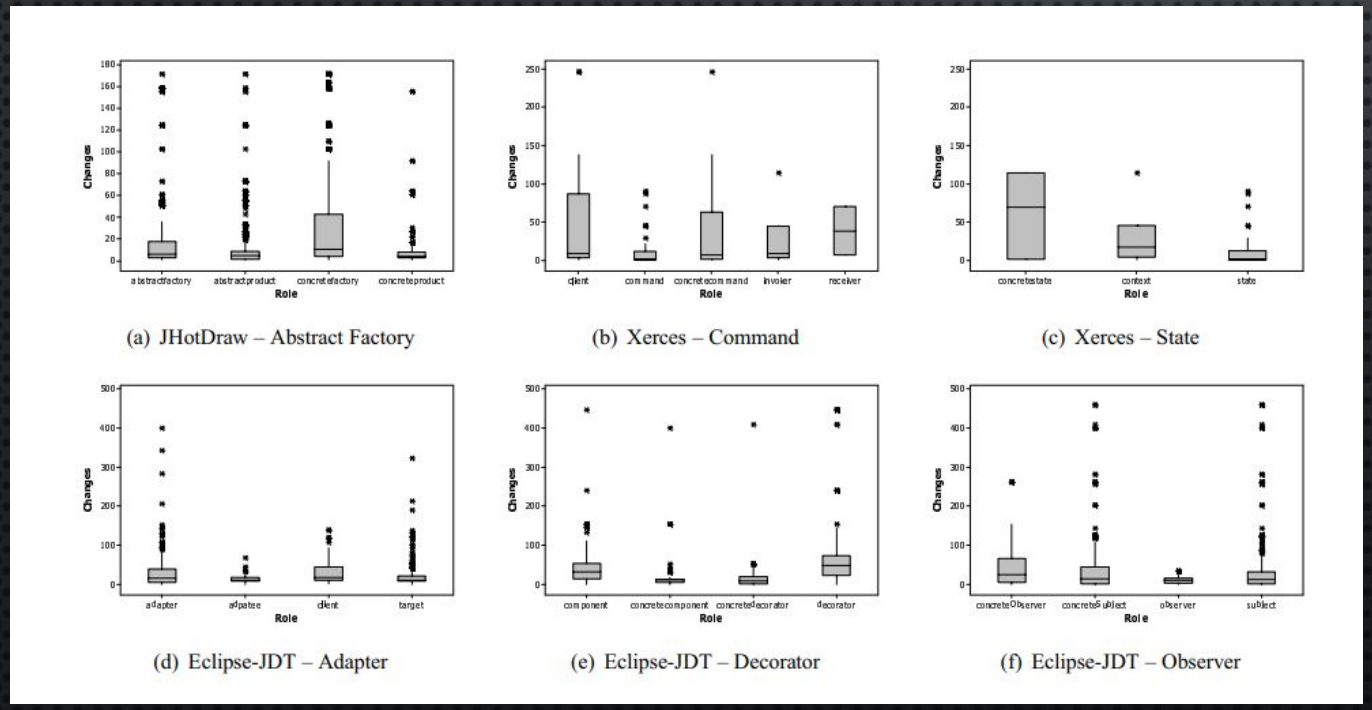
Table 2. Classification of Pattern Couplings

Coupled Patterns	Subtype			Strength	
	Intersection	Composite	Embedded (parent listed 1st)	Loose	Tight
Strategy–Abstract Factory	1			1	
Visitor–Template	1			1	
Builder–Template-Strategy	1			1	
Abstract Factory–Factory	1			1	
Mediator–Observer	1			1	
Composite–Visitor	1			1	
Factory–Prototype	1			1	
Singleton–State–Command	2				2
Singleton–State–Observer	1				1
Composite–Decorator–Flyweight	1				1
Observer–Facade–Template	1			1	
Bridge–Proxy–Flyweight	1			1	
Facade–Observer	1			1	
Mediator–Strategy	1				1
Adapter–Strategy	1				1
State–Singleton	1				1
Interpreter–Builder–Abstract Factory		1			1
Visitor–Command		1			1
Composite–Proxy		1			1
Observer–Singleton		1			1
Strategy–Observer–Composite			1	1	
Bridge–Observer–Proxy–Abstract Factory–Factory			1	1	
Mediator–Observer–Chain of Responsibility–Composite			1		1
Composite–Interpreter			1		1
Totals	17	4	4	12	13

ARTICLES CONNEXES – DIPENTA ET AL

AN EMPIRICAL STUDY OF THE RELATIONSHIPS BETWEEN DESIGN PATTERN ROLES AND CLASS CHANGE PRONENESS

- ÉTUDE DE LA PREDISPOSITION AUX CHANGEMENTS DES DIVERS PATRONS



(a) JHotDraw – Abstract Factory (b) Xerces – Command (c) Xerces – State

(d) Eclipse-JDT – Adapter (e) Eclipse-JDT – Decorator (f) Eclipse-JDT – Observer

(a) JHotDraw – Decorator (b) JHotDraw – Prototype (c) Xerces – Composite

(d) Xerces – Factory Method (e) Xerces – Observer (f) Xerces – State

(g) Xerces – Visitor (h) Eclipse-JDT – Adapter (i) Eclipse-JDT - -State

Figure 2. Examples of Kinds of Changes in Roles (A: Attribute change; M: Method; I: Implementation; H: inHeritance).

Questions de recherche

- Questions Descriptives.
  - RQ1: What is the proportion of classes playing zero, one, or two roles in some motif(s)?
- Questions Analytiques.
  - RQ2: What are the internal characteristics of a class that are the most impacted by playing one or two roles wrt. zero role?
  - RQ3: What are the external characteristics of a class that are the most impacted by playing one or two roles wrt. zero role?

Variables

- Indépendantes :
  - 3 échantillons de classe jouant 0, 1 ou 2 rôles
    - Groupe {0,1}, {0,2} et {1,2}
- Dépendantes :
  - 56 différentes métriques internes incluant : LCOM5, CBO, WCM, etc.
  - Le nombre de changement passé et futur

IMPLÉMENTATION

- Processus de selections
  - 6 programmes Java
  - two-sample Wilcoxon test.
    - 58 classes par échantillons
  - Prit a un moment dans le temps pour une période de temps.
    - Permet de voir des changement passé et futur

Programs	NOC	KLOC	Release Dates	Past Changes	Future Changes
ArgoUML v0.18.1	1,267	203	30/04/05	20,290	12,617
Azureus v2.1.0.0	591	84	1/06/04	18,304	483
JDT Core v2.1.2	669	185	3/11/03	23,243	26,923
JHotDraw v5.4b2	413	45	1/02/04	5,793	51
Xalan v2.7.0	734	259	8/08/05	12,298	1,714
Xerces v1.4.4	306	87	13/10/03	5,213	1,209
Total	3,980	862	6 releases	85,141	42,997

Table 1. Statistics for the six programs. (Future refers to the time between the release dates and 31/01/09.)

Programs	Expected	Zero Role	One Role	Two Roles
ArgoUML v0.18.1	17	17	10	10
Azureus v2.1.0.0	9	9	9	9
JDT Core v2.1.2	10	10	17	17
JHotDraw v5.4b2	6	6	6	6
Xalan v2.7.0	11	11	11	11
Xerces v1.4.4	5	5	5	5
Total	58	58	58	58

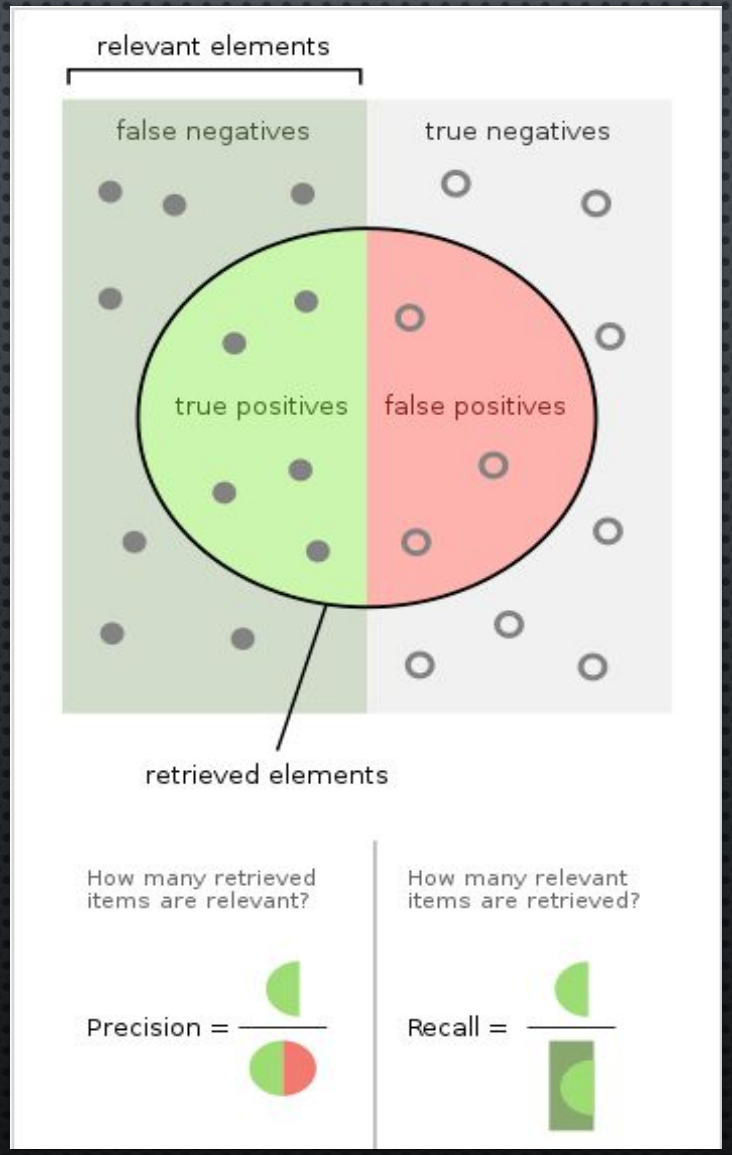
Table 2. Distribution of the sample size among the programs of our strata.

IMPLÉMENTATION

- 6 PATRONS CONSERVÉ :
  - CONSERVE SEULEMENT LES RÔLES PRINCIPAUX
  - CONSTRUCTION DE PAIR DE RÔLE (57 TOTAUX, LES PAIRS IDENTIQUES REFUSÉ)

Patterns	Descriptions	Main Roles
Command	Encapsulates a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations	Command, Invoker
Composite	Composes objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly	Component, Composite
Decorator	Attaches additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality	Component, Decorator
Observer	Defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically	Observer, Subject
Singleton	Defines a mechanism that ensure that the same instance of a class is used throughout a program execution	Singleton
State	Allows an object to alter its behavior when its internal state changes	Context, State

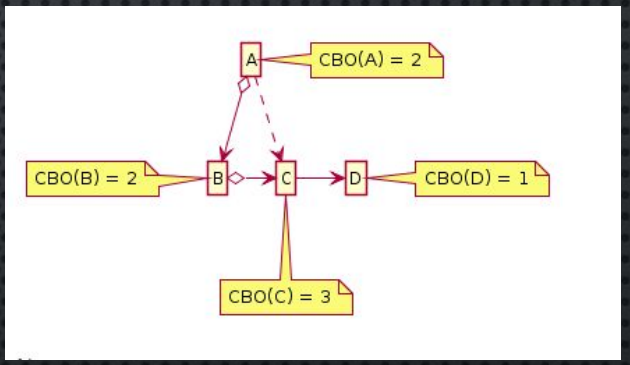
MÉTRIQUES IMPORTANTES



https://en.wikipedia.org/wiki/Precision_and_recall

Métric importante

- Lack of Cohesion in methods – LCOM
  - sum(field_per_method) / (field_qte * methods_qte)
- Coupling between Object (CBO)
  - Nombre de classe qui ont un lien UML avec la classe courante
- Cyclomatic Complexity (CC)
  - nombre minimal de chemin indépendant
- Weighted Method count (WMC)
  - la CC de chaque méthode additionné



https://stackoverflow.com/questions/27515541/cbo-coupling-between-object

DEMIMA

- Outils d'identifications des patrons de conceptions en JAVA
- Précision de 34% pour 12 motif, mais recall de 100%.
  - Recall de 100% permet d'être certains d'avoir tout les classes intéressantes dans nos ensembles
  - Précision faible = doit revérifier les classes

RÉSULTATS

RÉSULTATS – R1

Programs	Candidates	One Role	Two Roles
ArgoUML v0.18.1	21	3	10
		14.28%	47.61%
Azureus v2.1.0.0	64	22	19
		34.37%	29.68%
JDT Core v2.1.2	67	30	22
		44.77%	32.83%
JHotDraw v5.4b2	30	11	13
		36.66%	43.33%
Xalan v2.7.0	55	23	11
		41.81%	20.00%
Xerces v1.4.4	29	10	11
		34.48%	37.93%
Total	266	99	86
		37.21%	32.33%

Table 4. Validated precisions of DeMIMA.

Programs	Total	One Role	Two Roles
ArgoUML v0.18.1	1,267	51	316
	100%	4.02%	24.94%
Azureus v2.1.0.0	591	67	75
	100%	11.33%	12.69%
JDT Core v2.1.2	669	46	178
	100%	6.88%	26.60%
JHotDraw v5.4b2	413	24	101
	100%	5.81%	24.45%
Xalan v2.7.0	734	36	104
	100%	4.90%	14.16%
Xerces v1.4.4	306	94	56
	100%	30.72%	18.30%
Total	3,980	318	830
	100%	7.99%	20.85%

Table 5. Extrapolated numbers and percentages of classes playing no, one, or two roles.

RÉSULTATS – R2

	1 role vs. 0 role	2 role vs. 0 role	2 role vs. 1 role
Unsignificant change	ANA, connectivity, CP, DSC, MFA, NOH, PP, and RPII		
Significant ↗	CIS, CLD, DCAEC, DCMEC, EIC, EIP, ICHClass, LCOM5, MOA, NCM, NCP, NMA, NMD, NMDExtended, NMO, NOC, NOD, NOM, NOParam, NOPM, PIIR, REIP, RFP, SIX, WMC	ACAIC, ACMIC, AID, CAM, CBO, CBOin, CBOout, CIS, CLD, cohesionAttributes, DAM, DCAEC, DCC, DCMEC, DIT, EIC, EIP, ICHClass, LCOM1, LCOM2, LCOM5, McCabe, MOA, NAD, NADExtended, NCM, NCP, NMA, NMD, NMDExtended, NMI, NMO, NOA, NOC, NOD, NOM, NOP, NOParam, NOPM, PIIR, REIP, RFP, RTP, SIX, WMC, WMC1	ACMIC, CAM, CBO, CBOin, CBOout, cohesionAttributes, DAM, DCC, ICHClass, LCOM1, LCOM2, LCOM5, McCabe, MOA, NAD, NADExtended, NMD, NMO, NOA, NOM, NOP, SIX, WMC, WMC1
Significant ↘	LCOM1, LCOM2, RRTP, WMC1	RRFP, RRTP	CLD

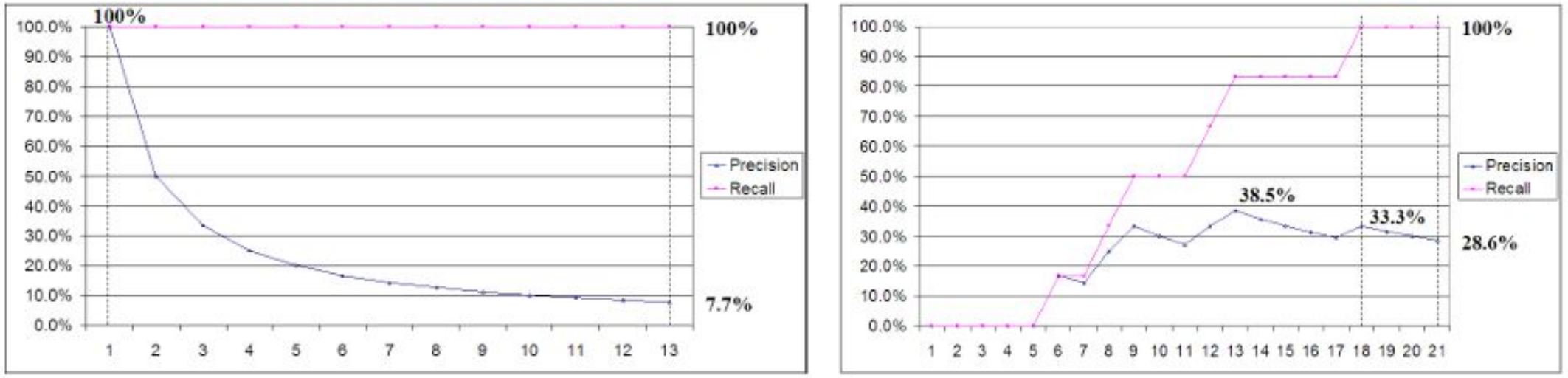
Table 6. Metrics Trends. (↗ or ↘ represent a significant increase (respectively, decrease) of, for example, the metrics values of 1-role classes compared to these of 0-role classes. Metrics that do not appear in a cell are those which values changed with no statistical significance.)

RÉSULTATS - R3

- 2 RÔLES - 56 % DES CHANGEMENTS
- 1 RÔLES – 33 % DES CHANGEMENTS
- 0 RÔLES – 11 % DES CHANGEMENTS

VALIDITÉ

VALIDITÉ – PRECISION & RECALL



(a) Precision and recall of the identification of Decorator.

(b) Precision and recall of the identification of State.

Figure 1. Precision and Recall.

Validité – langage et programmes

- Java.
- Purement Objet.
  - Néglige d'autres solutions dans des domaines plus procédurale ou fonctionnel
- Seulement OPEN Source.
  - Avantage? – mieux maintenu?

Validité - échantillonnage

- Étude sur des cas très spécifiques
- Doit être fait à la main donc petits échantillons
- Seulement 6 patrons vérifié et seulement leurs rôles principaux:
  - Command, Composite, Decorator, Observer, Singleton, State
  - Singleton - anti-patron?
  - Outils d'automatisation

Utilité et application de la recherche

- Comprendre l'impact et la complexité ajouté par les patrons et surtout sur l'utilisation de classe qui jouent plusieurs roles.
  - Les nombreuses métriques prouvent toutes une augmentation de la complexité
- Calculer la quantité de classe qui jouent des rôles dans les patrons et voir leur présence dans le code.
  - On apprend que plus de 30% des classes joue un rôles dans les motifs
- Les classes qui ont tendance a attiré le plus de changement sont celle qui utilisent des patrons de conceptions

Direction de recherche

- Est-ce que les patrons de conceptions ont la même utilité/présence/rôles dans les langage non-typé vs typé?
- Quels patrons de conception ont tendance à créer les classes les plus changeantes?
- Une classe qui joue un rôle dans un motif semble plus disposé au changement
  - ? Est-ce que ça nuit à la maintenabilité?
  - ? Pourquoi? Est-ce que les patrons sont prédisposés au changement? Est-ce le contexte d'utilisation qui les prédispose?
    - Serait-ce pareil si aucun patron n'était utilisé
- Est-ce que les anti-patrons sont aussi prédisposés au changement?

Quels caractéristique les patrons de conceptions amènent-ils

- DE: THE EFFECT OF GANG-OF-FOUR DESIGN PATTERNS USAGE ON DESIGN QUALITY ATTRIBUTES

$$\text{Reusability} = -0.25 * \text{Coupling} + 0.25 * \text{Cohession} + 0.5 * \text{Messaging} + 0.5 * \text{Design Size} \quad (1)$$

$$\text{Flexibility} = 0.25 * \text{Encapsulation} - 0.25 * \text{Coupling} + 0.5 * \text{Composition} + 0.5 * \text{Polymorphism} \quad (2)$$

$$\text{Understandability} = -0.33 * \text{Abstraction} + 0.33 * \text{Cohession} - 0.33 * \text{Coupling} + 0.33 * \text{Encapsulation} - 0.33 * \text{Design Size} - 0.33 * \text{Complexity} - 0.33 * \text{Polymorphism} \quad (3)$$

$$\text{Functionality} = 0.12 * \text{Cohession} + 0.22 * \text{Polymorphism} + 0.22 \text{ Messaging} + 0.22 * \text{Design Size} + 0.22 * \text{Hierarchies} \quad (4)$$

$$\text{Extendebility} = 0.5 * \text{Abstraction} - 0.5 * \text{Coupling} + 0.5 * \text{Inheritance} + 0.5 * \text{Polymorphism} \quad (5)$$

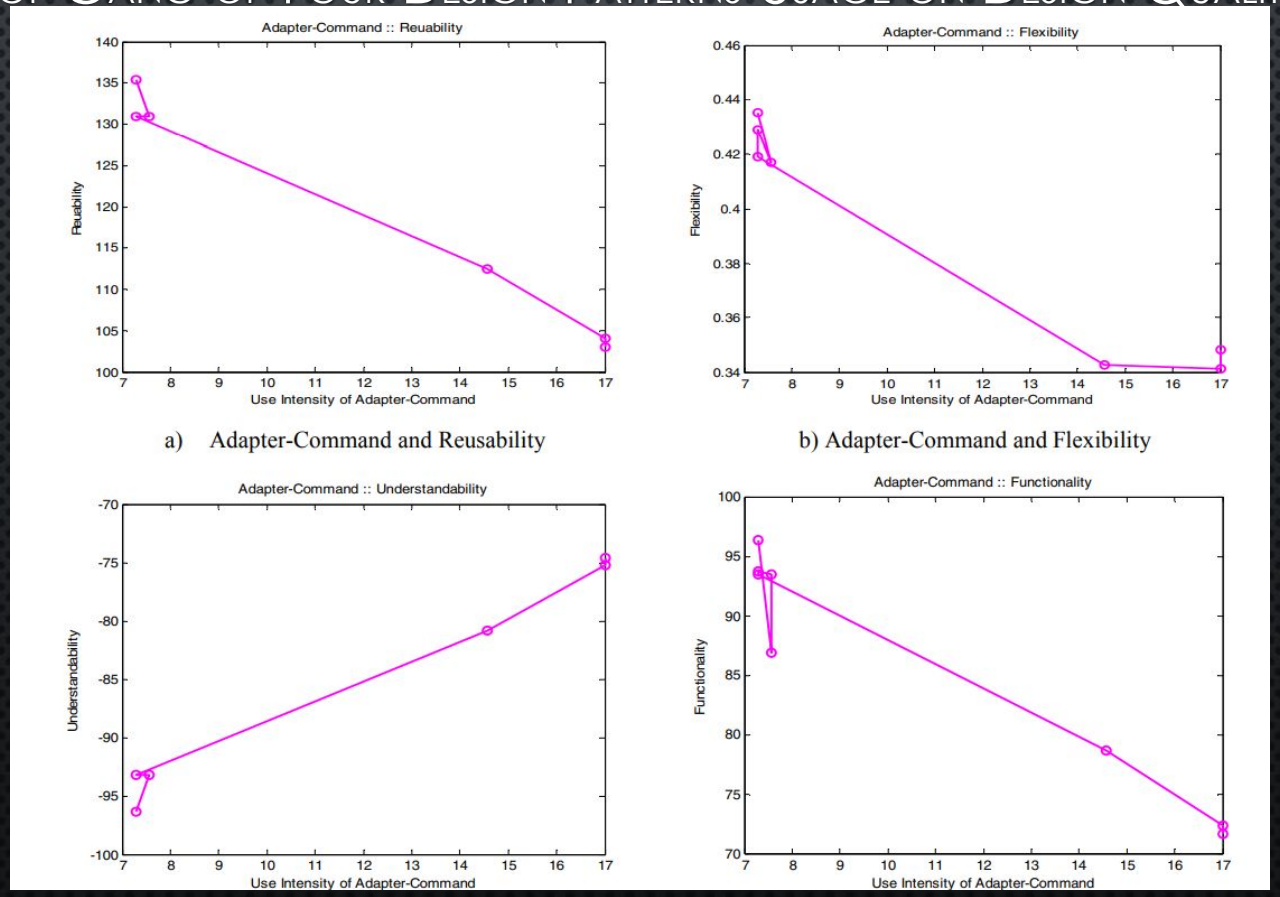
$$\text{Effectiveness} = 0.2 * \text{Abstraction} + 0.2 * \text{Encapsulation} + 0.2 * \text{Composition} + 0.2 * \text{Inheritance} + 0.2 * \text{Polymorphism} \quad (6)$$

Table 2. Spearman Correlation between GoF design pattern and Quality attributes

Design Patterns	Reusability	Flexibility	Understandability	Functionality	Extendibility	Effectiveness
Singleton	0.60(0.00)	0.33(0.03)	-0.60(0.00)	0.58(0.00)	-0.36(0.03)	0.42(0.00)
Factory Method	0.60(0.00)	0.32(0.01)	-0.60(0.00)	0.60(0.00)	-0.34(0.03)	0.37(0.00)
Prototype	0.35(0.01)	-0.01 (0.93)	-0.35(0.01)	0.32(0.02)	-0.23(0.10)	0.22(0.11)
Adapter-Command	0.63(0.00)	0.29(0.03)	-0.64(0.00)	0.65(0.00)	-0.33(0.01)	0.51(0.00)
Observer	0.68(0.00)	0.37(0.02)	-0.68(0.00)	0.67(0.00)	-0.31(0.02)	0.37(0.00)
State-strategy	0.6(0.00)	0.32(0.01)	-0.6(0.00)	0.64(0.00)	-0.34(0.01)	0.47(0.00)
Template Method	0.62(0.00)	0.17(0.24)	-0.62(0.00)	0.62(0.00)	-0.15(0.29)	0.29(0.03)
Visitor	0.07(0.61)	0.09(0.51)	-0.07(0.61)	0.05(0.75)	-0.07(0.63)	0.12(0.39)
Composite	0.23(0.09)	0.09(0.51)	-0.24(0.09)	0.27(0.05)	-0.21(0.14)	0.34(0.01)
Decorator	0.43(0.00)	0.12(0.40)	-0.43(0.00)	0.48(0.00)	-0.23(0.10)	0.37(0.00)
Proxy	0.48(0.00)	0.09(0.00)	-0.48(0.00)	0.48(0.00)	-0.50(0.00)	0.49(0.00)
Proxy2	-0.03(0.84)	0.08(0.58)	0.03(0.85)	-0.01(0.92)	-0.22(0.12)	0.22(0.12)

Quels caractéristique les patrons de conceptions amènent-ils

- DE: The Effect of Gang-of-Four Design Patterns Usage on Design Quality Attributes



Est-ce que les patrons de conceptions aident vraiment à la maintenabilité?

- De : A Controlled Experiment Comparing the Maintainability of Programs Designed with and without Design Patterns—A Replication in a Real Programming Environment
  - Prennent des sujet qui ne connaissent pas les Design Patterns
  - Leur fais faire un programme
  - Donne un cours au sujet
  - leur fais faire le programme

Est-ce que les patrons de conceptions aident vraiment à la maintenabilité?

- Tache 1 - patron Obs vs none:
  - Plus lent quand on ne connait pas
  - Plus vite après le cours
- Tache 2 – Composite et visiteurs vs composite
  - Peu de personne ont compris et bien utilisé le patron visiteurs
  - Les résultats ont été plus vite pour les groupe qui n'utilisait pas le visiteur même apres le cours
  - Aucun groupe n'ont eu un fort niveau de réussite
- Tache 3 – Décorateurs vs None
  - Décorateur semble être plus facile à comprendre et ne nécéssite pas de cours
  - Plus rapide et plus Correct
- Tache 4 – Abstract Factory, Composite vs Abstract Factory
  - Plus facile de rendre les chose Correct avec le patron composite
  - Mais pas plus rapide

EST-CE QUE LES PATRONS DE CONCEPTIONS AIDENT VRAIMENT À LA MAINTENABILITÉ?

Table 1. Descriptions of programs and tasks.

Program	Description and complexity	Tasks	Patterns
Stock Ticker (ST)	Display an incoming stream of data (read from a file) in one or more windows using a supplied, simple GUI library. Simple program with little data and low code complexity. Pat: 441 SLOC, 7 classes Alt: 374 SLOC, 7 classes	1: Add another kind of window (the window itself was supplied). 2: Let the user choose which windows should be visible.	Pat: Observer Alt: None
Boolean Formulas (BO)	A system for storing and manipulating boolean formulas, represented in a hierarchical data structure. Relatively complex, using a recursive data structure. Pat: 471 SLOC, 11 classes Alt: 372 SLOC, 8 classes	1: Evaluation of formulas. 2: Change name of one method, breaking the Composite pattern.	Pat: Composite, visitor Alt: Composite
Comm. Library (CO)	Wrappers for communication primitives such as transmit, receive, compress and decompress. Very little data and not very complex. Simple primitives with similar interfaces. Pat: 404 SLOC, 6 classes Alt: 342 SLOC, 1 class	1: Add a wrapper for a new (supplied) primitive. 2: Determine the conditions leading to a certain status value; determine how to create a channel with certain functionality.	Pat: Decorator Alt: None
Graphics Library (GR)	Represent graphic primitives such as point, line and circle, and a system for drawing them on several kinds of device. Methods for creating devices and corresponding primitives. Data structure is partly recursive, but less complex than in Boolean Formulas. The code is larger than the other programs, but well structured. Pat: 683 SLOC, 13 classes Alt: 667 SLOC, 11 classes	1: Add a new graphics device and corresponding subclasses of primitives. 2: Determine whether a running supplied method will result in a certain output.	Pat: Abstract Factory, Composite Alt: AbstFactory

EST-CE QUE LES PATRONS DE CONCEPTIONS AIDENT VRAIMENT À LA MAINTENABILITÉ?

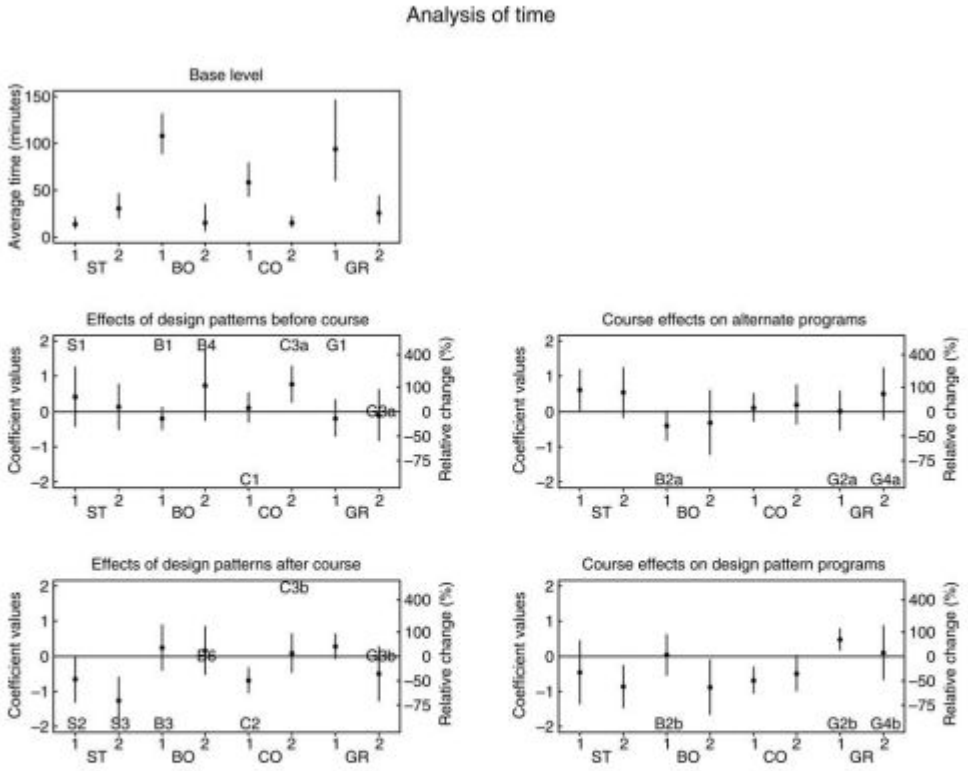


Figure 2. Analysis of elapsed times for all programs and tasks. The upper left panel of the figure shows the base levels b_t for each task, given in minutes. The estimates are given as dots, whereas the vertical lines are 95% confidence intervals. The four lower panels have the same structure as the upper left panel, but show changes relative to the base level. Hypotheses are shown, with the position of the label indicating the direction expected effect. The log scale is given at the left of the panel, and the corresponding relative change in % is given at the right side of the panel.

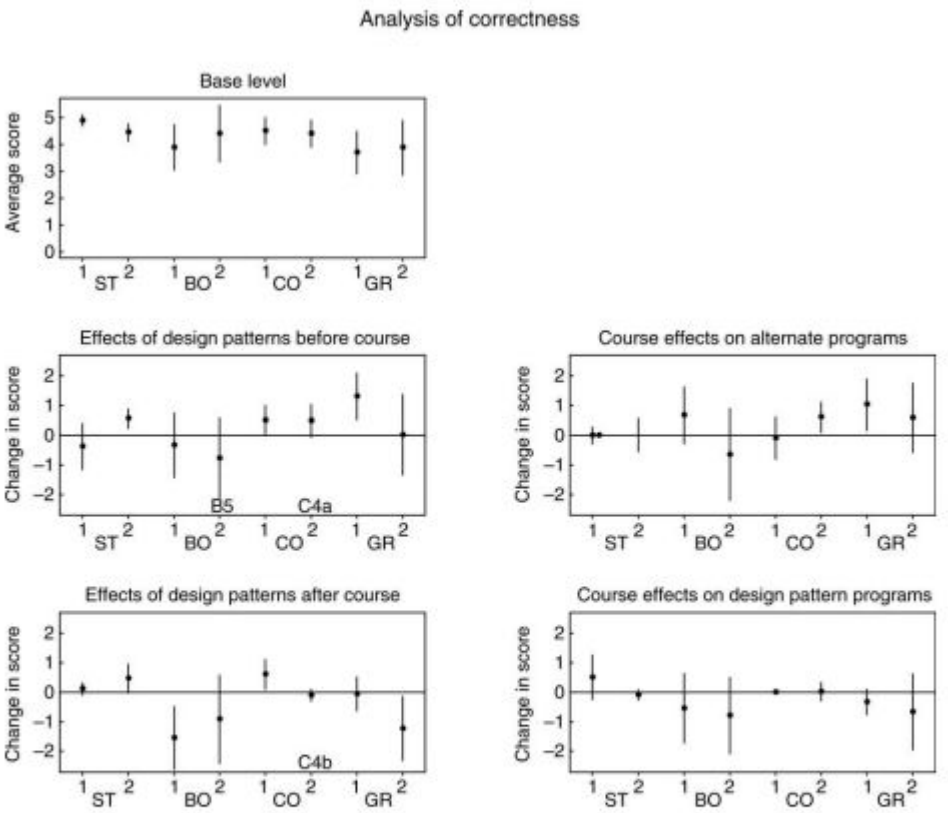


Figure 3. Analysis of correctness effects for all programs and tasks. The upper left panel of the figure shows the base levels a_t for each task, given as average scores. The four lower panels show changes in scores compared with the relevant alternative. Some confidence intervals have zero length. For the corresponding estimates, all relevant data had the same correctness, or the same change in correctness. In such cases, the GEE method is unable to compute confidence intervals.

CONCLUSION PERSONNEL

- Faire attention aux **patrons de conception**, car malgré qu'ils fournissent de belles solutions élégantes, ils ont tendance a **augmenté la complexité du code**
- **Identifier l'utilisation de patron aide les gens à mieux se reconnaître** et améliore la maintenabilité
- Les **classes jouant un role** au sein de patron de conception sont souvent celles qui **créent le plus changements dans notre programme** et doivent donc et celle auxquels **ont fait le plus attention.**
- Une **classe peut jouer un role dans plusieurs patrons**, mais elle **deviant exponentiellement plus complexe** et propice au changement
- **Pas tout le monde connaissent les patrons** et utiliser des patron complexes doit être fait avec parcimonie

CONCLUSIONS PERSONNEL

- “Indeed, the ratio of time spent reading versus writing is well over 10 to 1. We are constantly reading old code as part of the effort to write new code. ...[Therefore,] making it easy to read makes it easier to write”.

- “So if you want to go fast, if you want to get done quickly, if you want your code to be easy to write, make it easy to read.”

- Rendre son code compréhensible est la clé de la réussite

Clean Code - Robert C. Martin

Application dans le jeux Vidéo

- Endroit ou la performance prime sur tout
- Maintenabilité très différente en entreprise

ref : https://gameprogrammingpatterns.com/contents.html

Singleton ou comment ne pas l'utilisé

- Règle 2 problème :
  - accès global
  - 1 seule instance

ref : https://gameprogrammingpatterns.com/contents.html

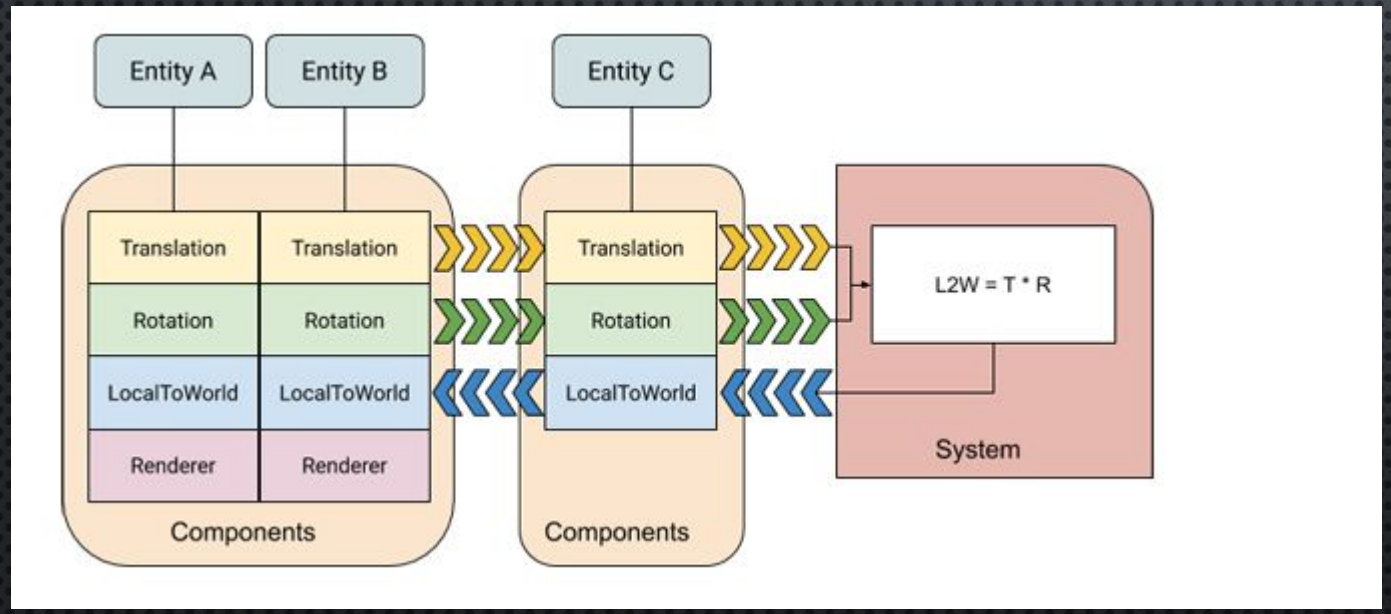
Data Locality

Original or traditionally transformed program	Our approach
`real A(N,N),Y`	`real A(N*N),Y`
`.`	`.`
`.`	`.`
`do i = 1,N`	`do i = 1,N`
`  do j = 1,4`	`  do j = 1,4`
`    Y=Y+A(j,i)`	`    Y=Y+A(4*(i-1)+j)`
`  enddo`	`  enddo`
`enddo`	`enddo`
`.`	`.`
`.`	`.`
for N=8000,	for N=8000,
# L1 data cache misses: 8,196	# L1 data cache misses: 4,200
# L2 data cache misses: 7,828	# L2 data cache misses: 1,017
# TLB misses: 8,337	# TLB misses: 8

Figure 1. First motivating example.

ref : https://gameprogrammingpatterns.com/contents.html

ECS - Entity Component System



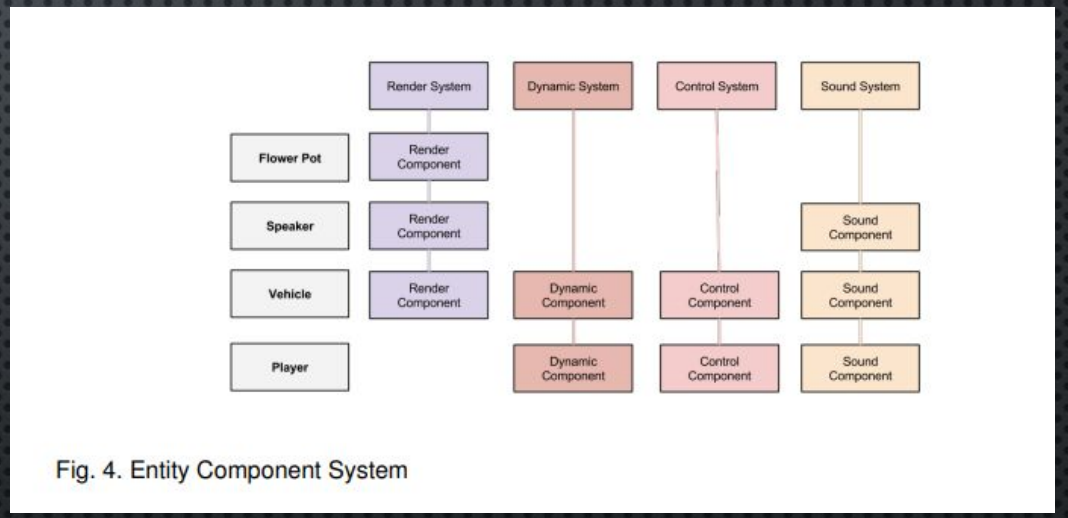


Fig. 4. Entity Component System

ref : https://gameprogrammingpatterns.com/contents.html

Questions?

BIBLIOGRAPHIE

1. F. Khomh, Y. -G. Gueheneuc and G. Antoniol, "Playing roles in design patterns: An empirical descriptive and analytic study," 2009 IEEE International Conference on Software Maintenance, Edmonton, AB, Canada, 2009, pp. 83-92, doi: 10.1109/ICSM.2009.5306327.
2. Posnett, D., Bird, C. & Dévanbu, P. An empirical study on the influence of pattern roles on change-proneness. *Empir Software Eng* **16**, 396–423 (2011). https://doi.org/10.1007/s10664-010-9148-2
3. Y. -G. Guéhéneuc and G. Antoniol, "DeMIMA: A Multilayered Approach for Design Pattern Identification," in *IEEE Transactions on Software Engineering*, vol. 34, no. 5, pp. 667-684, Sept.-Oct. 2008, doi: 10.1109/TSE.2008.48.
4. F. Jaafar, Y. -G. Guéhéneuc, S. Hamel and F. Khomh, "Mining the relationship between anti-patterns dependencies and fault-proneness," *2013 20th Working Conference on Reverse Engineering (WCRE)*, Koblenz, Germany, 2013, pp. 351-360, doi: 10.1109/WCRE.2013.6671310.
5. Hegedűs, P., Bán, D., Ferenc, R., Gyimóthy, T. (2012). Myth or Reality? Analyzing the Effect of Design Patterns on Software Maintainability. In: Kim, Th., Ramos, C., Kim, Hk., Kiumi, A., Mohammed, S., Ślęzak, D. (eds) Computer Applications for Software Engineering, Disaster Recovery, and Business Continuity. Communications in Computer and Information Science, vol 340. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-35267-6_18
6. Vokáč, M., Tichy, W., SjØberg, D.I.K. *et al.* A Controlled Experiment Comparing the Maintainability of Programs Designed with and without Design Patterns—A Replication in a Real Programming Environment. *Empirical Software Engineering* **9**, 149–195 (2004). https://doi.org/10.1023/B:EMSE.0000027778.69251.1f
7. W. B. McNatt and J. M. Bieman. Coupling of design patterns: Common practices and their benefits. In Proceedings of the 25th Computer Software and Applications Conference, pages 574–579. IEEE Computer Society Press, October 2001.
8. M. Di Penta, Luigi Cerulo, Y.-G. Gu´eh´eneuc, and G. Antoniol. An empirical study of the relationships between design pattern roles and class change proneness. In Proceedings of the 24th International Conference on Software Maintenance (ICSM). IEEE Computer Society Press, September–October 2008. 10 pages.
9. J. Hannemann and G. Kiczales. Design pattern implementation in Java and AspectJ. In Proceedings of the 17th Conference on Object-Oriented Programming, Systems, Languages, and Applications, pages 161–173. ACM Press, November 2002.
10. Martin, R. C., Coplien, J. O. (2009). *Clean code: a handbook of agile software craftsmanship*. Upper Saddle River, NJ [etc.]: Prentice Hall. ISBN: 9780132350884 0132350882
11. S. Hussain, J. Keung and A. A. Khan, "The Effect of Gang-of-Four Design Patterns Usage on Design Quality Attributes," *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, Prague, Czech Republic, 2017, pp. 263-273, doi: 10.1109/QRS.2017.37.