# 目標導向之設計樣式應用方法之研究

# A Goal-Driven Approach to Applying Design Patterns

薛念林
逢甲大學資訊工程學系
nlhsueh@fcu.edu.tw

朱鵬化
逢甲大學資訊工程學系
M9206138@webmail.fcu.edu.tw

## 摘要

近年來，設計樣式的研究越來越受到重視，特別是它對軟體品質的提昇的議題上。設計樣式是一種程式設計師的知識樣板，用以輔助設計者設計可重用、易維護及可擴充性等高品質的軟體系統。然而，如何在適當的時機引用適當的設計樣式對系統開發人員而言卻一直是很大的問題，本論文提出一個系統性的方法以協助系統開發者解決問題。本方法以目標為設計基礎，由需求擷取、物件分析到物件設計提供系統化的方式處理需求，為設計樣式與需求之間建立一個溝通分析的橋樑。

關鍵字：軟體工程、物件導向設計方法、設計樣式。

Keyword: Software Engineering, Object-oriented approach, Design Pattern。

## 一、簡介

在軟體工程的發展中，軟體的品質一直是一項重要的議題。在過去三十年裡，有許多的研究人員及開發人員都致力於提昇軟體品質方法的研究。最近，有越來越多的研究者對於如何應用設計樣式以提昇軟體品質漸感興趣[11,12,18,19]，各種不同的設計樣式陸陸續續的被發表[8,9]，而相關的方法論也被提出[7,13,6]。設計樣式是將過去有經驗的程式設計者的設計經驗抽鍊、整理、包裝成特定的框架，用以解決系統設計上的特定問題，例如擴展性問題、維護性問題與重用性問題等[9]。

為了證明設計樣式真的對軟體的品質有所幫助，許多學者用實驗統計的方式驗證採用設計樣式對軟體品質的幫助[18,19]，亦有學者從軟體衡量的角度間接的測量設計樣式的貢獻[12]，其結果都是正面的。然而，如何應用設計樣式卻不是一件容易的事，主要原因如下：(1)設計樣式採用了大量的物件技巧，對於不熟悉物件技巧的人而言，了解設計樣式是一件負擔；(2) 設計樣式僅是一個樣式、樣板，當套用到問題時，必須分析問題與設計式的相容性，因此花費很多成本；(3) 設計樣式一直在增加，如果缺乏系統的分析整理，很難應用新的設計樣式解決問題。

有鑑於此，許多研究學者提出自動化的方法或工具以自動採用設計樣式[5,7,13]，這一類的研究多應用在反向式軟體開發（reverse engineering）上，透過程式碼與設計樣式的正規化分析找出程式與設計樣式的接合點（hot spot），將原始程式修正成符合設計樣式的架構。這一類的研究目前除了辨識率不高外，並且仍有許多問題：(1)以結構為主的樣式採用可能會建構出違背需求的架構；(2)分析設計樣式所付出的成本過高；(3)沒有考量樣式之間的衝突關係。

表一：會議排程系統部分設計目標

| Goal | Description | Rigid/ soft | Specified by | Achieved/ optimized by |
|---|---|---|---|---|
| MRS | MeetingRequestSatisfied：依照會議可能時間、地點及參予人員喜好時間及厭惡時間建立一個會議排程 | Rigid | Actor | Use case |
| MNOP | MaxNumberOfParticipant：建立的時程必須僅可能在所有參予者的厭惡時間之外，讓最多人參加會議。 | Soft | Actor | Use case |
| MCS | MaxConvenientSchedule：建立的排程必須僅可能的滿足所有參予者的喜好時間 | Soft | Actor | Use case |
| EE | EasilyExtensible：必須僅可能的提高系統的可擴充性 | Soft | Actor | PA, DA, HPAD, PR, SVF |
| PA | PartialAttendence：允許參予者僅參加部分會議 | Rigid | Actor | Use Case |
| DA | DelegatedAttendence：允許參予者委託他人參加會議 | Rigid | Actor | Use Case |
| HPAD | HandlePriorityAmongDates：可依喜好日期之間的權重關係安排會議議程 | Rigid | Actor | Use Case |
| PR | PartialReuse：排程方法必須僅可能的可以重用於課程排程系統 | Soft | Actor | Design |
| SVF | SupportVariantFormat：提供不同的日期顯示格式 | Rigid | Actor | Analysis |

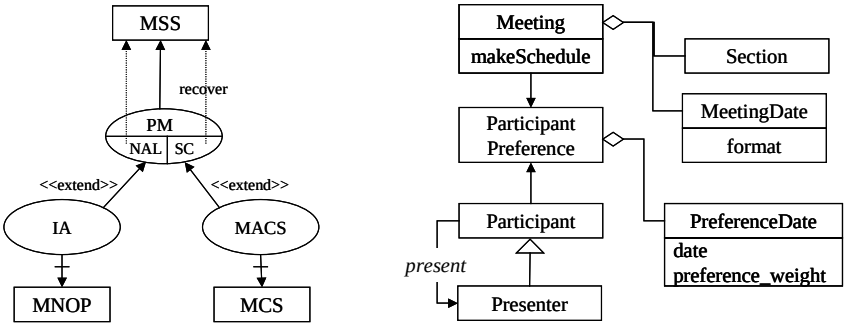
另一方面，目標導向的方法論亦在需求工程中受到重視，目標導向的精義在於它強調「系統為何而建」，據以提供以下優點：(1)藉由思考如何達到目標而協助需求的建立；(2)藉由分析需求與目標的關係推導復得需求間的衝突關係。在之前的研究 GDUC 中，我們提出目標導向的方法論以建構使用案例[15,16]。本研究將應用並延伸目標的方法論以協助設計樣式之應用。本方法的主要特點如下：(1) 需求導向：設計樣式的應用主要是符合使用的需求，而非架構上的相容；(2)品質提昇：考慮品質因素，利用設計樣式提昇軟體的品質；(3)系統方法：利用目標的概念做系統化的分析，從需求分析、物件分析以至於物件設計提供一致化的處理；(4)紀錄設計原理(design rationale)：利用目標的概念作為設計樣式與需求間的橋樑，以紀錄設計樣式採用的原因。

## 二、系統化設計樣式應用

在之前的研究 GDUC[15]中，我們提出目標導向的方法論以建構使用案例。本研究將應用並延伸其目標的方法論以協助設計樣式之應用。目標導向的方法強調系統為何而建，據以提供以下優點：(1)藉由思考如何達到目標而協助需求的建立；(2)藉由分析不同需求(或設計方法)對目標的關係分析需求(或設計方法)間的衝突關係。3.1 中會先描述本方法的目標架構，並在後續的小節中描述如何應用這個架構建立使用案例、物件分析模型與設計模型。

### 2.1 目標架構

本方法的目標架構乃延伸 GDUC 所提出的目標架構，其主要觀念如下：(1)目標為一抽象性之描述，用以引導需求及設計的建立; (2)目標可分為硬性目標（rigid goal）與軟性目標(soft goal)；前者表示系統必須完全滿足，後者表示該目標可程度性的滿足; (3) 功能性需求可完全滿足，屬於硬性目標；非功能需求及品質要求僅可程度性滿足，在本架構中以軟性目標模組之。(4) 品質目標為需求無關（requirement-irrelevant）的目標，亦即其不會在需求階段出現，而是在後續設計階段才會出現的目標，這樣的架構理念是：儘管品質目標並非需求所提及，但達成此目標可提升軟體品質，故在設計階段仍盡可能的滿足




圖一：使用案例與物件分析圖

此目標。關於品質相關的需求若在需求階段定義，則稱之為非功能性需求，並以非功能性目標模組之; (5) 目標可分解成為許多子目標。子目標的達成將有助於父目標滿足程度的提昇。(6) 目標多半由使用者所訂定(actor-specified)，據以引導設計出符合使用者的需求，例如會議排程最佳化、友善介面等；目標也可由設計者所訂定(developer-specified)，據以符合設計的維護需求[1]。(7)目標可以由透過互動的設定(use case)與設計(design decision)達成，統稱為目標實踐(goal-operationalization)。(8)不論是透過設計樣式或設計策略以達成目標，其過程都有可能對其他的目標產生影響，包含正面的(有助於目標滿足程度的提昇)，及負面的(有礙目標滿足程度的提昇)。

### 2.2 開發程序

為了說明整個方法論，本論文引用會議排程系統部分的需求來說明本方法論。本研究將設計階段分為目標導向需求擷取(goal-driven requirement elicitation)、使用者中心需求分析(user-centered requirement analysis)、物件導向分析(object-oriented analysis)與物件導向設計(object-oriented design)等階段。前三個階段的詳細設計方法描述於 GDUC[15,16]中，本節將中重點第四個階段。

### 2.3 目標導向需求擷取

這個階段的重點是定義使用者使用系統的目的。使用者使用系統的目的可能是功能性的或非功能性的，分別以硬性目標與軟性目標表達之，見表一。

請注意表一中的 Achieved_by 可以有三種情況。第一種情況是 Use Case，表示此目的可以透過使用者-系統之間互動的設計來達成，例如 MCS 可以透過使用者反覆的修正會議喜好程度來達成。但這並不代表設計階段不需考量此目標，而是表示此目標的處理已表現在使用案例中，之後的處理只要將使用案例轉為物件設計即可。第二種情況是物件設計，表示無法透過使用者-系統之間互動來解決，必須透過物件設計來解決，例如系統的可維護性、可重用性等。第三種是透過分解，例如 EasilyExtensible，必須透過其他子目標的達成而達成。

### 2.4 使用者中心需求分析

此階段的主要目的是建立使用案例。使用案例描述系統與使用者之間的互動，藉此了解系統必須提供的功能。GDUC 除了描述系統功能外，亦擴充使之描述使用案例與非功能性需求間的關係。圖一(a) 為部分的目標導向

[1] NFR 框架有類似的觀念，其區分為 customer-oriented 與 developer-oriented[11]。

使用案例圖，其中 PM 為使用案例 plan a meeting、IA 為使用案例 increase attendance、MACS 為使用案例 make a convenient schedule，這些使用案例都各自連接相關的目標，用以敘述如何透過使用者-系統的互動來達成目標。請注意 PM 中包含兩個例外處理 NAL(not allowable location) 與 SC(strong conflict)，用以描述目標不能達成時的處理方式。並不是所有使用者所定義的目標都可以在使用階段解決，例如 PartialReuse 就必須透過設計結構的改變，例如將可再使用的部分抽離為獨立的模組。在本論文中，我們將使用設計樣式來解決這個問題。

### 2.5 物件導向分析

此階段主要是參考使用案例的敘述建立物件模組。所建立的物件主要以領域物件（domain object）為主。圖一(b)為精簡的模組分析圖，Meeting 包含所有關於會議的訊息與功能；Participant 包含潛在參予者的基本資料；ParticipantPrefrence 則管理參予者對會議的喜好厭惡資料。

另外，為了滿足 PartialAttendance 的目的，我們利用模組切割(decouple)的方式將會議的 Section 自會議中獨立出來；為了滿足 DelegatedPresentation 的目的，我們利用 subclass 的方式將 Presenter 自 Participant 獨立出來；PreferenceDate 也自 ParticipantPreference 中獨立出來以便更容易的處理會議喜好日期的權重設定。

### 2.6 物件導向設計

此階段主要的工作為架構設計、問題解決與物件的細部設計(例如更明確的方法參數設定)等。本論文將重點放在如何利用設計樣式來解決問題。

#### 2.6.1 設計樣式的結構

為了輔助使用者了解設計樣式，設計樣式通常包含許多項目(section)以便從各方面描述其所內涵的設計知識、目的。這些項目可區分為 why 與 how：(1) 為何使用此設計樣式(why)?這方面包含的項目有設計樣式的意圖(intent)、動機(motivation)、適用時機(applicability)及影響(consequence)。設計樣式的意圖通常與系統設計的品質相關，例如提昇系統的可維護性、可擴充性等。(2) 設計樣式如何解決問題(how)?包含設計樣式的結構(structure)、合作(collaboration)及範例程式(example)等。結構通常描述設計樣式所採用的物件結構，例如繼承、包含、委託、延遲黏和(late binding)等。這方面(how)的核心是結構，其餘的項目都是以結構為基礎的延伸。

#### 2.6.2 設計樣式的採用

設計樣式的選擇通常依據其意圖而定，例如觀察者設計樣式的意圖定義為：

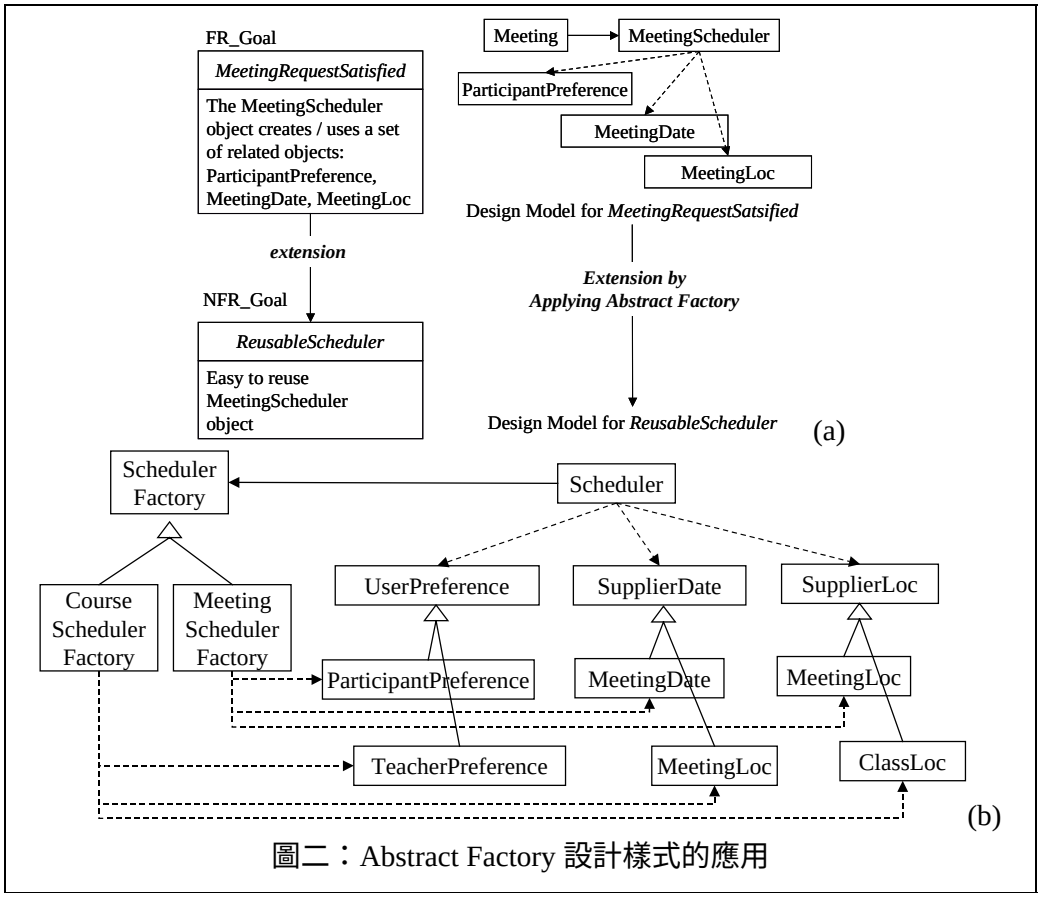
*Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.*

依據此意圖，當物件設計遇到一致性問題及可參考此設計樣式。然而，這樣的方式卻有兩個問題 (1) 多半的設計樣式的設計是為了「改善」軟體的品質，這一點卻無法從設計樣式的意圖中觀察出來；(2) 設計樣式的採用僅取決於問題，而非使用者需求，可能會造成問題解決了卻違反需求的問題。有鑑於此，我們從三面分析設計樣式：需求角度、問題角度與結構角度。

#### 2.6.3 需求角度(requirement-view)

表二：需求角度的設計樣式分析

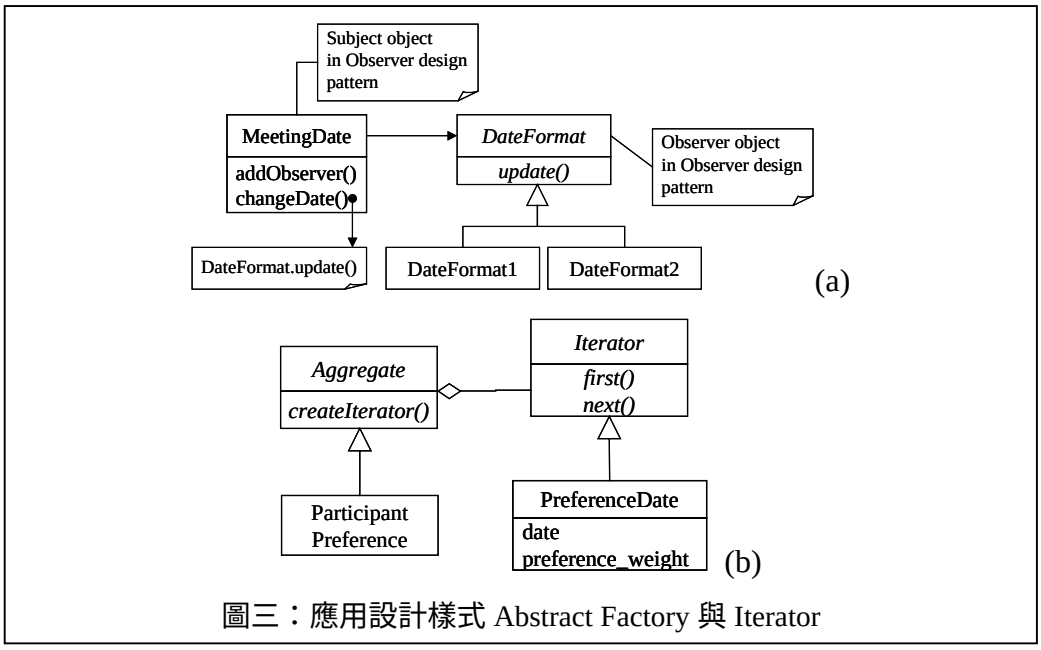
| | 功能性 | 非功能性 |
|---|---|---|
| Abstract Factory | 一物件 client 建立或使用一群相關的物件(product) | (1)容易再利用 client 物件；(2)容易擴充 product 物件 |
| Command | 一系統可儲存(queue)其所產生的要求(request/command)，用以支援 undo 的功能 | 容易擴充要求 |
| State | 物件有明顯的狀態行為(explicit state-based behavior) | 易擴充物件的狀態行為 |
| Strategy | 一物件可使用一演算法以解決特定問題 | 演算法可彈性的抽換或擴充 |




圖二：Abstract Factory 設計樣式的應用

此角度是由 FR-NFR 的角度來分析設計樣式，設計樣式被認為是一種功能性需求的延伸，主要用以解決非功能性的需求。依照此種擴充的關係將設計樣式分成功能性與非功能性，用以突顯設計樣式對非功能性需求的貢獻。表二為此角度下的設計樣式分析。例如在 Abstract Factory 設計樣式中，其功能性是一物件(client)可建立/使用一群物件(product)。以此為基礎，其非功能性則要求可以容易的再使用 client 物件與擴充 product 物件。非功能性要通常與軟體品質有直接的關係，例如再使用、擴充等，表二將這些關於非功能性的字眼用斜體表示。

在會議系統中，ReusableSchedule 即是一個非功能性目標，相依於功能性目標 MeetingRequestSatisfied。為了讓會議排程可以讓課程排程再使用(reuse)，我們將 MeetingScheduler 自 Meeting 中分離出來。圖三(a)中可以看出 MeetingScheduler 會用到其他的類別物件如 ParticipantPreference、MeetingDate 與 MeetingLoc 等這樣的需求正




圖三：應用設計樣式 Abstract Factory 與 Iterator

好符合 Abstract Factory 的功能性需求，而 Abstract Factory 的非功能性需求又可以提高模組再使用的程度，因此可採用此設計樣式來達成 ReusableScheduler。新的架構圖如圖二(b)。

### 2.6.4 問題角度(problem-view)

從問題角度來看設計樣式的功能，其目的在解決系統設計時的問題，如一致性問題、物件生成問題等。例如在會議系統中，為了達成 SupportVariantDateFormat，我們將 MeetingDate 自 Meeting 類別中抽離出來，可是如何保持各格式間的一致化便成為一個設計問題，為了解決此問題，我們採用 Observer 設計樣式，其結果模組於圖三(a)。

### 2.6.5 品質角度(quality-view)

在理想的狀況下，經過需求導向與問題導向的設計樣式應用後，系統可以沒有問題的滿足需求，然而，我們還可以再利用設計樣式來讓系統具備更好的品質。相對於一般基本的物件結構，設計樣式所採用的物件結構或技巧通常較能提高軟體品質[12]。因此，當物件模組採用某一基本的物件結構時，它便可透過採用設計樣式所提供的解決方案「提昇」到高階的物件結構。從此角度而言，設計樣式的目的是提昇系統品質。

表三：品質角度的設計樣式分析

| | 基本結構 | 結構轉換 | 品質提昇 |
|---|---|---|---|
| Iterator | 類別 A 瀏覽(navigate)類別 B | 將瀏覽部分抽象為一公開介面、類別 B 實作此一介面並私有化其他功能 | 穩固性 |
| Mediator | 一群物件間有複雜的合作關係 | 將物件間的合作關係抽象出來建立一個類別 | 降低模組間偶合力 |
| Factory method | 在某個方法中生成一群相關的物件 | 將每個生成都獨立成為一個方法，以便讓子類別擴充 | 可擴充性 |
| Decorator | 類別 C 繼承類別 P，以繼承 P 所提供的功能 | 建立 P'為 P 之父類別，讓 C 包含 P'並建立呼叫欲繼承的方法於 C，透過委託的方式將要求送給 P'。 | 動態連結、可擴充性 |

品質角度又稱為結構角度，因為品質的提

昇是因為結構的轉換，表四列舉部分實例。

在會議系統中，我們用到許多物件瀏覽的結構，例如 ParticipantPreference 會瀏覽 PreferenceDate 物件。由於此結構符合 Iterator 的基本結構，Iterator 便成為可能的品質提昇方案，在經過其他的分析，如相容度分析(物件模組的各介面是否與此設計樣式的見面相容)、權衡性分析(是否會與其他需求衝突)後，我們將 Iteractor 套用在此結構中，以提昇系統的穩固性(見圖三(b))。

## 三、結論

應用設計樣式來提高軟體品質是近幾年來軟體工程相關領域十分重視的方法。本研究的目的在提供一個系統化的方式協助設計者有規則的應用設計樣式。本方法將設計樣式的應用區分為三種模式：(1) 需求導向應用模式：設計樣式用以滿足非功能性需求；(2) 問題導向應用模式：設計樣式用以解決設計問題；(3) 品質導向應用模式：設計樣式用以提昇軟體品質。

各種模式應用的成功與否取決於事先對設計樣式的分析是否完備，在未來的研究中，我們將更廣泛的將這些模式套用在各種設計樣式以檢測其實用性。除此以外，本研究仍有許多未解決的問題 (1)目前的設計衝突解決策略為需求優先、問題次之、品質再次之，這對於需求之間的衝突分析仍未考慮。儘管如此，我們可以參考 GDUC 的方式，以目標作為評斷比較的基準來分析。(2)在利用結構轉換以提昇品質方面，本研究的分析並未完整。

## 四、參考文獻


[1] A.I. Anton. Goal-based requirements analysis. In *Proceedings of the International Conference on Requirements Engineering*, pages 136–144, 1996.

[2] C. Souveyet C. Rolland and C.B. Achour. Guiding goal modeling using scenarios. *IEEE Transactions on Software Engineering*, 24(12):1055–1071, December 1998.

[3] L. Chung, K. Cooper, and A. Yi. Developing adaptable software architectures using design patterns: an nfr approach. *Computer Standards and Interfaces*, 23:253–260, 2003.

[4] L. Chung, B.A. Nixon, E. Yu, and J. Mylopoulos. *non-functional requirements in software engineering*. Kluwer publishing, 2000.

[5] M.O. Cinneide and P. Nixon. A methodology for the automated introduction of design patterns. In *Proceedings of the international conference on software maintenance*, pages 463–472, 1999.

[6] A. Dardenne, A.van Lamsweerde, and S. Fickas. Goal-directed requirements acquistion. *Science of Computer Programming*, 20:3–50, 1993.

[7] A.H. Eden, A. Yehudai, and J. Gil. Precise specification and automatic application of design patterns. In *Proceedings of the 12th international conference on automated software engineering*, pages 143 –152, 1997.

[8] M. Fowler. Pattern. *IEEE Software*, March/April 2003.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Software*. Addison-Wesley, 1994.

[10] Alan R. Graves and Chris Czarnecki.

Design patterns for behavior-based robotics. *IEEE Transactions on Systems, Man, and cybernetics-part A: systems and humans*, 30(1), January 2000.

[11] D. Gross and E.Yu. From non-functional requirements to design through patterns. In P.Loucopoulos and J. Mylopoulos, editors, *Requirements Eng*, volume 6, pages 18–36. Springer-Verlag, 2001.

[12] B. Huston. The effects of design pattern application on metric scores. *The Journal of systems and software*, 58:261–269, 2001.

[13] S.U. Jeon, J.S. Lee, and D.H. Bae. An automated refactoring approach to design pattern-based program transformations in java programs. In *Proceedings of Ninth Asia-Pacific Conference of Software Engineering*, pages 337 –345, 2002.

[14] I. Khriss, R.K. Keller, and I.A. Hamid. Pattern-based refinement schemas for design knowledge transfer. *Knowledgebased system*, 13:403–415, 2000.

[15] J. Lee and N.L. Xue. Analyzing user requirements by use cases: A goal-driven approach. *IEEE Software*, 16(4):92–101, 1999.

[16] J. Lee, N.L. Xue, and J.Y. Kuo. Structuring requirement specifications with goals. *Information and Software Technology*, pages 121–135, February 2001.

[17] R.T. Monroe, A. Kompanek, R. Melton, and D. Garlan. Architecurual styles, design patterns, and objects. *IEEE Software*, 14(1):43–52, January 1997.

[18] L. Prechelt, M. Philippsen B.U. Lamprecht, and W.F. Tichy. Two controlled experiments assessing the usefulness of design pattern documentation in program maintenance. *IEEE transaction on software engineering*, 28(6), June 2002.

[19] L. Prechelt, B. Unger, W.F. Tichy, and L.G. Votta. A controlled experiment in maintenance comparing design patterns to simpler solutions. *IEEE transaction on software engineering*, 27(12), 2001.

[20] L. Tahvildari and K. Kontogiannis. A software transformation framework for quality-driven object-oriented reengineering.In *Proceedings of the international conference on software maintenance (ICSM02)*, 2002

[21] A. van Lamsweerde, R. Darimont, and P. Massonet. Goal-directed elaboration of requirements for a meeting scheduler problems and lessons learnt. Technical Report RR-94-10, Universite Catholique de Louvain, Departement d'Informatique, B-1348 Louvain-la-Neuve, Belgium, 1994.

[22] J. Zhu and P. Jossman. Application of design patterns for object-oriented modeling of power systems. *IEEE Transactionson Power Systems*, 14(2):532–537, 1998