1. Поняття
про патерн(и) проектування

Design Pattern(s)

http://www.vincehuston.org
http://sourcemaking.com
http://www.dofactory.com

https://exceptionnotfound.net/rethinking-the-visitor-pattern-with-the-double-dispatch-approach/?ref=exception-not-found-newsletter

Pattern

1. ім.

1)
   а) зразок, взірець, модель
   б) приклад для наслідування
   to establish / set a pattern — подавати приклад
2)
   а) модель, шаблон
   б) викрій (в пошитті)
   в) метал. форма, модель (для лиття)
3) малюнок, візерунок (на тканині тощо)
4)
   а) система, структура; принцип, модель (організації чогось)

……

Означення

Патерн проектування – це опис взаємодії об’єктів і класів, розроблених для вирішення загальної задачі проектування в конкретному контексті

Адаптоване до SE означення, так будемо трактувати DP далі:

- означення не аксіоматичне
- практично кожне слово вимагає уточнення
- але жодного не можна викинути

Етапи кристалізації ідеї

1. **1977 Ідея патерну (шаблону) проектування в архітектурі**

   prof. Christofer Alexander

   A Pattern Language: Towns, Buildings, Construction

2. **1987 Застосування ідеї патерну в галузі проектування програм**

   Kent Beck, Ward Cunningham

   Using Pattern Languages for Object-Oriented Programs

3. **1995 Формалізація опису 23 найзагальніших патернів проектування програмного забезпечення**

   E.Gamma, R.Helm, R.Johnson, J.Vlissides (GoF)

   Design Patterns: Elements of Reusable Object-Oriented Software"

Патерн як ідіома

- pattern – синоніми: ідіома, артефакт абстракції
- **патерн** не є елементом синтаксису конкретної мови, можна розробляти на основі більш елементарних понять
- **ідіома** традиційно означає шаблон дуже низького рівня (але тут немає однозначності)

  приклад: історична реалізація інтуїтивного поняття ITERATOP

  задача1: копіювати елементи (рядок символів) з одного контейнера в інший

  задача2: перебрати всі елементи контейнера

  . . . . . .

  рішення: за допомогою змінної циклу

  цикл в маш.кодах

  цикл на фортрані DO 22 I=IS,IF

  while(*cp1++ = *cp2++);

  string copy-operator(c-tor)

  string::iterator

- За історичний час у програмуванні встигло вмерти і безслідно щезнути тьма ідей -- поняття ж **патерну** інтенсивно розвивається і експлуатується – свідчення про незаурядність (якщо не **геніальність**) авторів ідеї

Алгоритм створення патерну

1) розглянути приклади (множину) задач , які **ЧАСТО** зустрічаються певному **КОНТЕКСТІ**

2) перейти до відповідної **абстрактної** задачі

3) сформулювати **ІДЕЮ** методу для розв’язування абстрактної задачі, яку можна **ПОВТОРНО ВИКОРИСТОВУВАТИ** для подібних задач

- **ІСТОРИЧНО** такий підхід вперше формально проявився в архітектурі і будівництві

Christopher Wolfgang Alexander

(born October 4, 1936 in Vienna, Austria)

- In 1954, he was awarded the top open scholarship to Trinity College, Cambridge University in chemistry and physics, and went on to read mathematics
- He earned a Bachelor's degree in Architecture and a M.S. degree in Mathematics
- He took doctorate at Harvard (1-st in Architecture) and was elected fellow at Harvard
- During the same period he worked at MIT in transportation theory and in CS, and worked at Harvard in cognition and cognitive studies
- More than 200 building projects in California, Japan, Mexico and around the world

A Pattern Language: Towns, Buildings, Construction

- All **253** patterns together form a language.
- **Patterns describe a problem and then offer a solution**.
- In doing so the authors intend to give ordinary people, not only professionals, a way to work with their neighbors to improve a town or neighborhood, design a house **for themselves** or work with colleagues to design an office, workshop or public building such as a school.

In pattern languages used for design, the parts break down in this way:

1. The language description, the **vocabulary**, is a collection of named, described solutions to problems in a field of interest. These are called "design patterns." So, for example, the language for architecture would describe items like: settlements, buildings, rooms, windows, etc.
2. Each solution includes "**syntax**," a description that shows where the solution fits in a larger, more comprehensive or more abstract design. This automatically links the solution into a web of other needed solutions. For example, rooms have ways to get light, and ways to get people in and out.
3. The solution includes "**grammar**" that describes how the solution solves a problem or gets a benefit.
4. . . . . .

Pattern Language as a Grammar

❑ **A Pattern Language** is a form that scientist might call a ***generative grammar*** (Noam Chomsky):

- attempts to give a set of rules that will correctly predict which combinations of words will form grammatical sentences
- the rules will also predict the morphology of a sentence.

❑ ***Morphology*** is the identification, analysis and description of the structure of morphemes and other units of meaning in a language such as words, affixes, parts of speech, intonation/stress, or implied context.

The Nature of Order:
An Essay on the Art of Building and the Nature of the Universe (2003-4)

The Phenomenon of Life, *The Process of Creating Life*, *A Vision of a Living World* and *The Luminous Ground*

A new theory about the nature of space and describes how this theory influences thinking about architecture, building, planning, and the way in which we view the world in general.

The mostly static patterns from *A Pattern Language* have been amended by more dynamic **sequences**, which describe how to work towards patterns (which can roughly be seen as the end result of sequences). Sequences, like patterns, promise to be tools of wider scope than building (just as his theory of space goes beyond architecture).

Kent Beck

(1961)

Kent Beck is an American software engineer:

- the creator of the Extreme Programming and Test Driven Development software development methodologies
- one of the 17 original signatories of the Agile Manifesto in 2001
- M.S. degree in CS from the University of Oregon.
- has pioneered software design patterns, the rediscovery of test-driven development, as well as the commercial application of Smalltalk
- popularized CRC cards with Ward Cunningham
- along with Erich Gamma created the JUnit unit testing framework

Howard G. "Ward" Cunningham

(born May 26, 1949)

Ward Cunningham is an American computer programmer:

- developed the first wiki
- a pioneer in both Design Patterns and Extreme Programming
- started programming the software WikiWikiWeb in 1994 and installed it on the **c2.com**, on March 25, 1995, as an add-on to the Portland Pattern Repository

Using Pattern Languages for Object-Oriented Programs

***Kent Beck**, Apple Computer, Inc., **Ward Cunningham**, Tektronix, Inc.*

September 17, 1987 .Submitted to the OOPSLA-87 workshop

Abstract

We outline our adaptation of Pattern Language to object-oriented programming.

". . .Consider a very small pattern language for designing Smalltalk windows. We suggest the following patterns:

1. Window Per Task
2. Few Panes Per Window
3. Standard Panes
4. Short Menus
5. Nouns and Verbs

We presented these patterns to a team. Without detailed understanding of any of Smalltalk's interface mechanisms (MVC for example) they were able to specify very reasonable interfaces after one day of practice.

Note that we sorted and numbered the patterns. Pattern 1 must be addressed first. It decides what windows will be available and what will be done in them. Next patterns 2 and 3 divide each window into panes. Finally patterns 4 and 5 determine what selections and actions will do within each pane. This order was derived from the topology of influences between each pattern . . . "

GoF

John Vlissides was a consultant during grad school at Stanford before leaving for IBM Research, got a **Ph.D**. in electrical engineering from Stanford University.

Erich Gamma built ET++ (a UI toolkit and programming environment) in the late '80s. He got a **Ph.D**. in CS from University of Zurich and his dissertation described the patterns he discovered.

Richard Helm built programming tools at IBM Research (where he and Vlissides later teamed up), got a **Ph.D**. in CS from the University of Melbourne, Australia.

Ralph Johnson was involved with Smalltalk since graduate school and conducted work on design patterns as a professor at the University of Illinois. He got his **Ph.D.** from Cornell.

Erich Gamma

(born 1961 in Zürich)

"... So that is perfectly fine with the community handling, what they do... also the critique on the Gang of Four patterns is really well taken. Not that I agree with all of it, but there is of course good stuff happening in the community and it is also interesting how the community is reinventing themselves right now, there's a new generation of people in there right now."

Gamma has joined Microsoft as a Distinguished Engineer and will work on Visual Studio. He will continue to work out of Zurich, Switzerland, where Microsoft will create a lab to be led by him.

Gamma became famous after Design Patterns. He used to be a Distinguished Engineer for Rational Software from 1998, then for IBM after they acquired Rational.

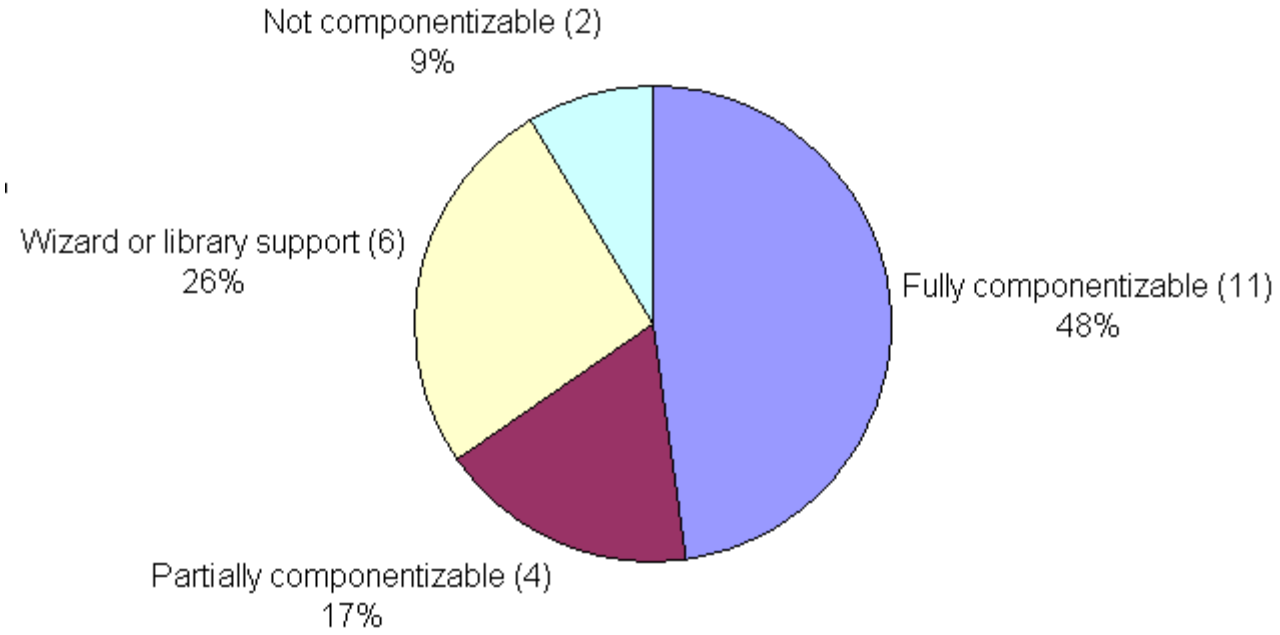
Gamma is also known for JUnit, the unit testing framework he developed with Kent Beck, for his contribution to the Eclipse platform as one of the leads, head of Eclipse Java Development Tools project, and several technical key roles for IBM/Rational Jazz, Team Concert and Collaborative Lifecycle Management.

Design Patterns v.c. Componentization

it's better to reuse than to redo.

Definition: Componentizable pattern

A design pattern is componentizable if it is possible to produce a reusable component which provides all the functions of the pattern.



Componentization results for DP GoF

Bertrand Meyer, Karine Arnout, 2006, Eiffel.

2. “Філософське” ?

Чи нам то треба ?

Наш Всесвіт живе
за законами всесвіту,
а метелик живе,
як хоче.

М.Зарічний. Вербалізація вербалогу, ст.129

3. Types of Design Patterns in SE

Класифікація GoF: фундаментальні патерни

- **Creational patterns** Патерни утворення об'єктів (твірні)

  Singleton, Builder, Abstract Factory, Factory Method, Prototype

- **Structural patterns** Структурні патерни

  Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy

- **Behavioral patterns** Патерни поведінки

  Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template method, Visitor

Domain-specific patterns

- user interface DP
- information visualization DP
- secure DP
- "secure usability" DP
- Web DP
- business model DP
- concurrency DP - deal with multi-threaded programming

Higher Level Abstractions

- **Algorithm strategy patterns** addressing concerns related to high-level strategies describing how to exploit application characteristic on a computing platform.

- **Computational design patterns** addressing concerns related to key computation identification.

- **Execution patterns** that address concerns related to supporting application execution, including strategies in executing streams of tasks and building blocks to support task synchronization.

- **Implementation strategy patterns** addressing concerns related to implementing source code to support program organization, and the common data structures specific to parallel programming.

- **Structural design patterns** addressing concerns related to high-level structures of applications being developed.

GRASP

General Responsibility Assignment Software Patterns (or **Principles**) guidelines for assigning responsibility to classes and objects:

1. **Information Expert**
2. **Creator**
3. **Controller**
4. **Low Coupling**
5. **High Cohesion**
6. **Polymorphism**
7. **Pure Fabrication**
8. **Indirection**
9. **Protected Variations**

These techniques have not been invented to create new ways of working but to better document and standardize old, tried-and-tested programming principles in object oriented design.

It has been said that "the critical design tool for software development is a mind well educated in design principles. “ GRASP is really a **mental** toolset, a learning aid to help in the design of object oriented software.

Системні (архітектурні) патерни + фреймворки

MVC, Session, Callback, Router, Transaction, MVVM, . . .

MVC is decomposable into just two patterns:

- **Observer** : the View is an Observer on the Model
- **Strategy :** the Controller is a Strategy for the View

Model View ViewModel

- **MVVM** is an architectural pattern that originated from Microsoft as a specialization of the **Presentation Model** (introduced by M.Fowler).

- Elements of the MVVM :

  - **Model**: as in MVC, refers to either (a) an object model that represents the real state content, or (b) the data access layer that represents that content (a data-centric approach).
  - **View**: as in MVC, refers to all elements GUI.
  - **ViewModel**: is a "Model of the View" : an abstraction of View that also serves in data binding between View and Model. It could be seen as a **Controller** (in the MVC pattern) that acts as a data binder/converter that changes Model information into View information and passes commands from View into Model. ViewModel exposes public **properties, commands, and abstractions**. ViewModel has been likened to a conceptual state of the data as opposed to the real state of the data in the Model.

- **Controller**: some references for MVVM also include a Controller layer or illustrate that the ViewModel is a specialized functional set in parallel with a Controller, while others do not. This difference is an ongoing area of discussion regarding the standardization of the MVVM pattern.

- Allowing for the layers of an application to be developed in multiple work streams.

4. Методологія DP (GoF)

Стандартизація інформації патернами

Патерн проектування – це опис взаємодії об’єктів і класів, розроблених для вирішення загальної задачі проектування в конкретному контексті:

- Назва та класифікація
- Призначення
- Відомий також за назвою
- Мотивація
- Застосування
- Структура
- Учасники
- Відношення
- Результати
- Реалізація
- Приклад коду
- Відомі застосування
- Споріднені патерни

Проектування за допомогою DP

- Пошук відповідних об’єктів
- Визначення рівня деталізації об’єктів
- Специфікація інтерфейсів об’єктів
- Специфікація реалізацій об’єктів
- Визначення механізмів повторного використання
- Врахувати можливі зміни в майбутньому

Вибір патерну: рекомендації GoF

1. Проаналізувати, як конкретні патерни вирішують проблеми дизайну
2. Переглянути в каталозі призначення патернів
3. Визначити **взаємозв’язки патернів**
4. Проаналізувати патерни з подібними цілями
5. З’ясувати причини, що можуть зумовити перепроектування

Приклад:
Глава 2. Проектування редактора документів

DP Relationships