Lecture 4

Topics on Design Patterns

Last tutorial …

OpenOME, a requirements engineering tool

- We explained the requirements, design and implementation of the tool. We also pointed out how to contribute to the tool.
- After the tutorial
  - I posted the source code to the Sourceforge
  - You can download the branch “ECE450-v1” from the CVS, you can also download the packed SDK
- The design of OpenOME has used some design patterns, such as Observer (MVC), Visitor, Command …
- Hope you can smell right places to apply the design patterns after the lecture …

Last lecture …

Reengineering Engineering

- Goal oriented requirements engineering leads to a better understanding of quality
- Design Patterns can be considered as the *operationalization* of the quality improvements

Two papers in the literature link requirements with patterns:

- Ladan Tahvildari, Kostas Kontogiannis: Improving design quality using meta-pattern transformations: a metric-based approach. Journal of Software Maintenance 16(4-5): 331-361 (2004)
- Ladan Tahvildari, Kostas Kontogiannis, John Mylopoulos: “Quality-driven software re-engineering”. Journal of Systems and Software 66(3): 225-239 (2003)

Today …

Topics on Design Patterns

1. What are design patterns?
2. How are they classified?
3. How to apply design patterns?
4. How to identify design patterns?
5. Which qualities are related to the design patterns?
6. Summary

Reference

The (Gang of Four) book:

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. 1995.

1. What are design patterns?

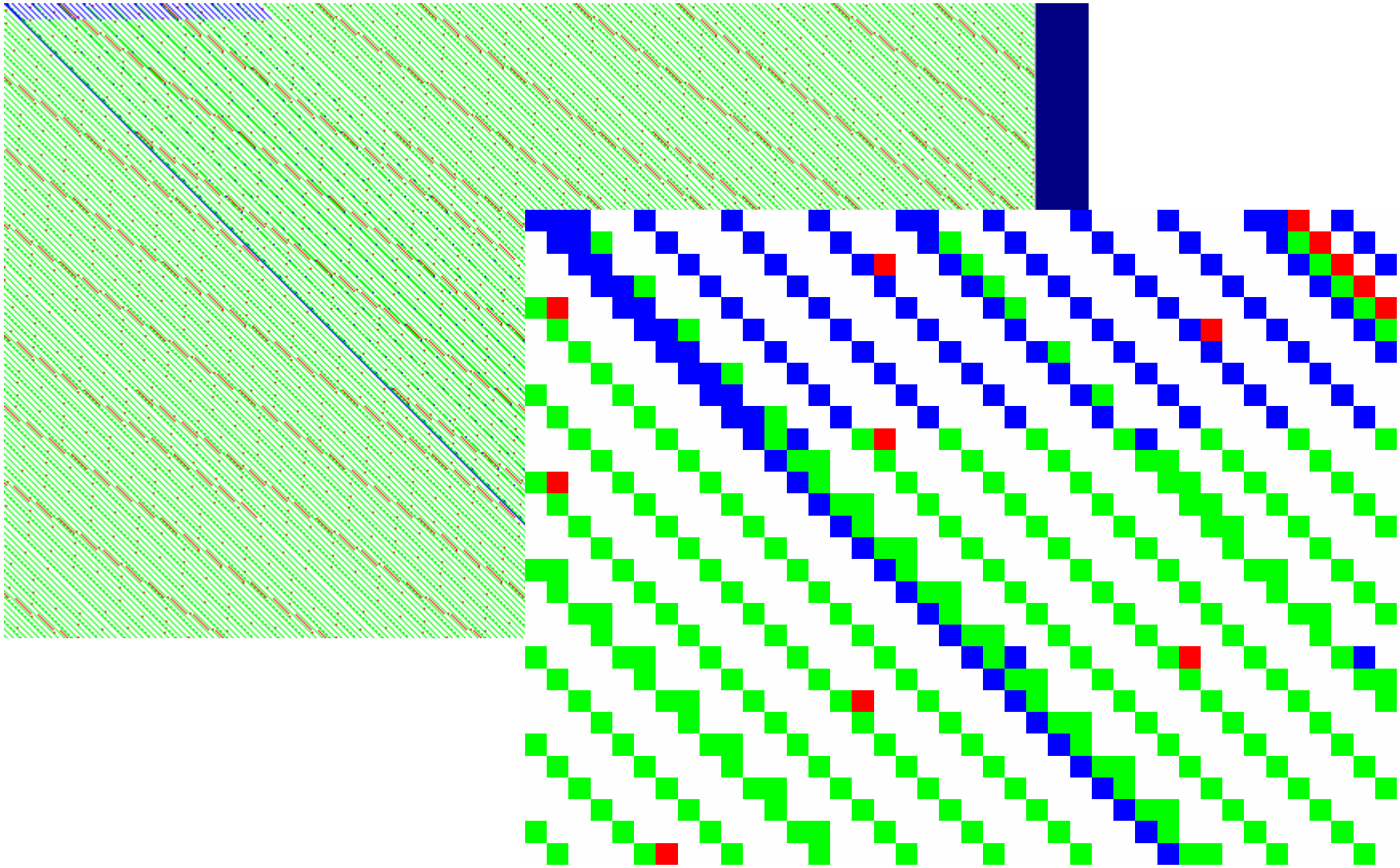
What are the patterns?

- Patterns in the textile (cloths, tiles)
- Patterns in visualizations

It is a recurring phenomenon …

Y. Yu, K. Beyls, and E.H. D'Hollander. Visualizing the impact of the cache on program execution. In *Proceedings of Fifth International Conference on Information Visualization*, pages 336-341, London, England, July 2001. IEEE Computer Society.

Patterns in cache behaviour

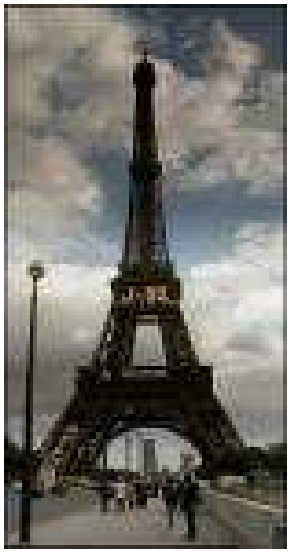


Patterns in Design

- Architect: *Christopher Alexander*
- *A Pattern Language*, and *A Timeless Way of Building*

六合塔，杭州
Hanchow, China

Pisa tower
Pisa, Italy



Eiffel tower
Paris, France

CN Tower
Toronto, Canada

Pearl Tower
Shanghai, China

Design Patterns

- (Design) patterns are devices that allow programs to share knowledge about their design. It is reusable when the same kind of problem reoccur …
- When there are many reusable patterns, they must be classified in a common language to effectively apply the pattern to similar problems again …

2. Catalogue of design patterns

- *Creational*
  Abstract Factory, Builder, Factory, Lazy Initialization, Prototype, Singleton, etc.
- *Structural*
  Adapter, Bridge, Composite, Container, Decorator, Delegation, Extensibility, Façade, Flyweight, Interface, Pipes and filters, Proxy, etc.
- *Behavioural*
  Chain of Responsibility, Command, Event Listener, Immutable, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor, etc.

A pattern language

- Pattern language: “the pattern of patterns” provides a template for patterns
  - The motivation or context that this pattern applies to.
  - Prerequisites that should be satisfied before deciding to use a pattern.
  - A description of the program structure that the pattern will define.
  - A list of the participants needed to complete a pattern.
  - Consequences of using the pattern...both positive and negative.
  - Examples!

The GOF pattern language

- **Pattern Name and Classification:** Every pattern should have a descriptive and unique name that helps in identifying and referring to it. Additionally, the pattern should be classified according to a classification such as the one described earlier. This classification helps in identifying the use of the pattern.
- **Intent:** This section should describe the goal behind the pattern and the reason for using it. It resembles the problem part of the pattern.
- **Also Known As:** A pattern could have more than one name. These names should be documented in this section.
- **Motivation:** This section provides a scenario consisting of a problem and a context in which this pattern can be used. By relating the problem and the context, this section shows when this pattern is used.
- **Applicability:** This section includes situations in which this pattern is usable. It represents the context part of the pattern.
- **Structure:** A graphical representation of the pattern. Class diagrams and Interaction diagrams can be used for this purpose.
- **Participants:** A listing of the classes and objects used in this pattern and their roles in the design.
- **Collaboration:** Describes how classes and objects used in the pattern interact with each other.
- **Consequences:** This section describes the results, side effects, and trade offs caused by using this pattern.
- **Implementation:** This section describes the implementation of the pattern, and represents the solution part of the pattern. It provides the techniques used in implementing this pattern, and suggests ways for this implementation.
- **Sample Code:** An illustration of how this pattern can be used in a programming language
- **Known Uses:** This section includes examples of real usages of this pattern.
- **Related Patterns:** This section includes other patterns that have some relation with this pattern, so that they can be used along with this pattern, or instead of this pattern. It also includes the differences this pattern has with similar patterns.

3. How to apply design patterns? Example

Factory method

- Constructing objects based on some input such that functions inside the objects depend upon the input.
- Consider as an example a class to read image files and make thumbnails out of them
  - A bad example
  - An improved one
  - The Use of the pattern

Bad one

```
1.  public class ImageReader {
2.      private int fileType;
3.      private String fileContents;
4.      private byte[] decodedImage;
5.      public ImageReader( InputStream in ) {
6.          // Figure out what type of file this input stream represents
7.          // (eg gif, jpeg, png, tif, etc )
8.          this.fileType = fileType;
9.          decodeFile();
10.     }
11.     private void decodeFile() {
12.         switch( fileType ) {
13.         case ImageReader.GIF:
14.             // do gif decoding (many lines)
15.             break;
16.         case ImageReader.JPEG:
17.             // do jpeg decoding (many lines)
18.             break;
19.         case ImageReader.PNG:
20.             // do png decoding (many lines)
21.             break;
22.         // etc...
23.         }
24.     }
25. }
```

Improved

```
1.  public Interface ImageReader {
2.      public DecodedImage getDecodedImage();
3.  }

4.  public class GifReader implements ImageReader {
5.      public GifReader( InputStream in ) {
6.          // check that it's a gif, throw exception if it's not, then if it is
7.          // decode it.
8.      }
9.      public DecodedImage getDecodedImage() {
10.       return decodedImage;
11.     }
12. }

13. public class JpegReader implements ImageReader {
14.     //...
15. }

16. // Then you would use them as:
17. public class MyProg {

18.     public static void main( String[] args ) {
19.         String filename = args[0];
20.         ImageReader out;
21.         if( endsInDotGif( filename )) {
22.             out = (ImageReader)new GifReader( fileInputStream );
23.         }
24.         if( endsInDotJpeg( filename )) {
25.             out = (ImageReader)new JpegReader( fileInputStream );
26.         }
27.         printOut( out.getDecodedImage );
28.     }
29. }
```

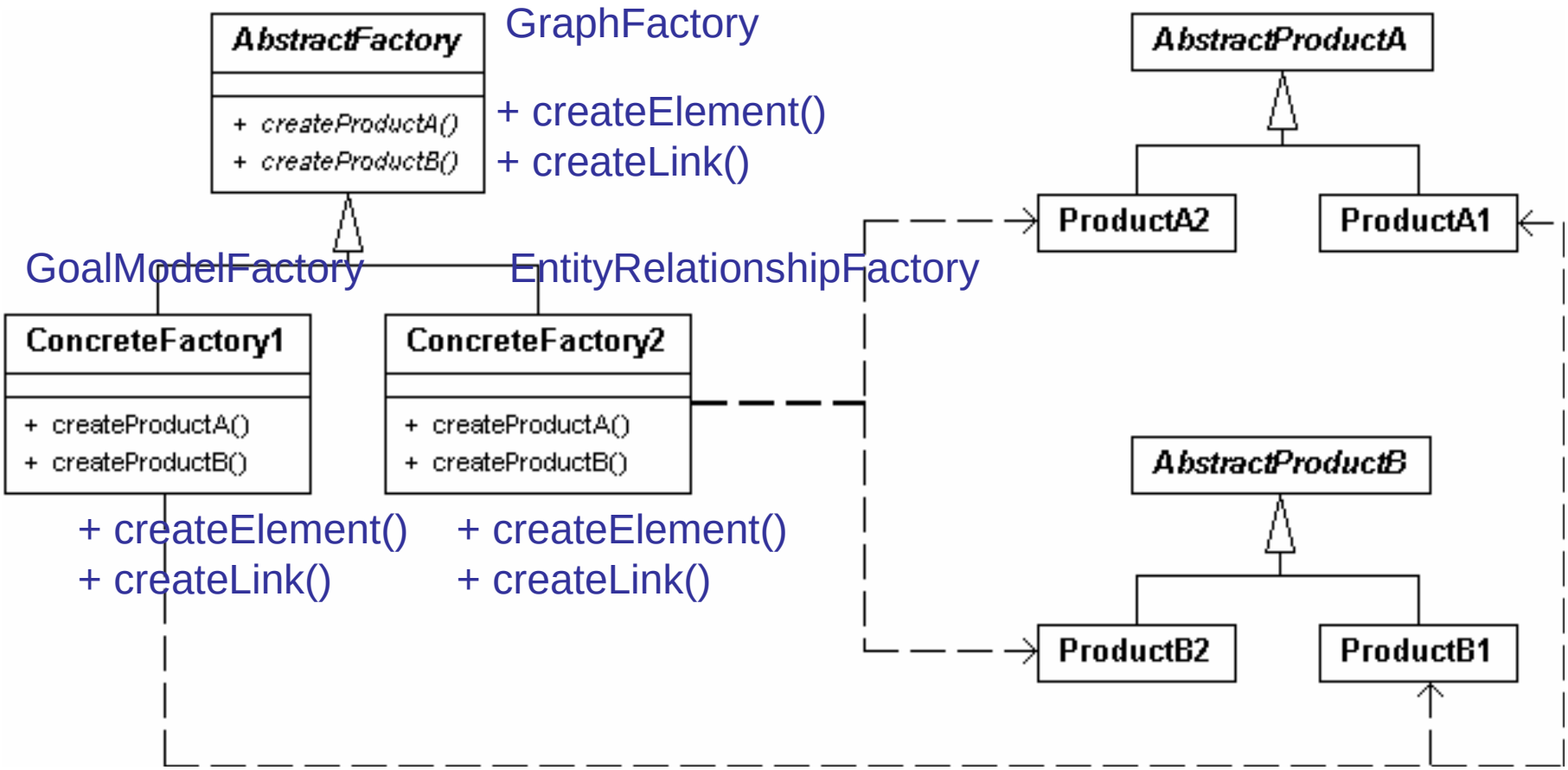
Factory pattern applied

```
1.  public class ImageReaderFactory {
2.      public static ImageReader getImageReader( InputStream is ) {
3.          int ImageType = figureOutImageType( is );
4.          switch( ImageType ) {
5.          case ImageReaderFactory.GIF:
6.              GifReader r = new GifReader( is );
7.              return( (ImageReader)r );
8.              break;
9.          case ImageReaderFactory.JPEG:
10.             JpegReader r = new JpegReader( is );
11.             return( (ImageReader)r );
12.             break;
13.         // etc.
14.         }
15.   }
16.}
```

Why Abstract Factory pattern?

- As the factory only returns an abstract object, the client code (which requested the object from the factory) does not know - and is not burdened by - the actual concrete type of the object which was just created.
- Adding new concrete types is done by modifying the client code to use a different factory…. the Abstract Factory pattern

Abstract Factory



4. How to identify design patterns

1. Convert code into visual forms, such as UML diagrams through reverse engineering tools
2. Recognize patterns in the diagrams either through naked eyes, or automated tools
   - Computer games such as CHESS, GO

- Design recovery (still hot topic)
  - If both are automated, you can rely on internal program representation such as Abstract Syntax Tree to reveal structural patterns
  - Behaviour patterns recovery could, however, rely on semantics rather than syntax

References

- *C. Krämer and L. Prechelt. "Design recovery by automated search for structural design patterns in object-oriented software". WCRE'96*
- Jörg Niere, Wilhelm Schäfer, Jörg P. Wadsack, Lothar Wendehals. "*Towards Pattern-Based Design Recovery*". ICSE, 2002.

5. What qualities are associated with design patterns?

- Understandability
- Maintainability
- Reusability
- Flexibility and Extensibility
  and so on …

- Consider object-oriented metrics, correctly applying design patterns leads to better design: modularity (less coupling, high cohesion)

References

- Ladan Tahvildari, Kostas Kontogiannis: Improving design quality using meta-pattern transformations: a metric-based approach. Journal of Software Maintenance 16(4-5): 331-361 (2004)

6. Summary

- Design patterns collect a set of reusable solutions to common recurring problems in the design of (object-oriented) software
- To effectively reuse them, you need to classify them into catalogues and express them in a pattern language
- The recognition of design patterns can be assisted by reverse engineering tools
- The effects of design patterns can be known by measuring the OO structural metrics of the software product

Further readings

- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. 1995.
- *C. Krämer and L. Prechelt. "Design recovery by automated search for structural design patterns in object-oriented software". WCRE'96*
- Jörg Niere, Wilhelm Schäfer, Jörg P. Wadsack, Lothar Wendehals. "*Towards Pattern-Based Design Recovery*". ICSE, 2002.
- Ladan Tahvildari, Kostas Kontogiannis: "Improving design quality using meta-pattern transformations: a metric-based approach". *Journal of Software Maintenance 16*(4-5): 331-361 (2004)
- Ladan Tahvildari, Kostas Kontogiannis, John Mylopoulos: "Quality-driven software re-engineering". *Journal of Systems and Software 66*(3): 225-239 (2003)

What's next …

- A Tutorial on more Design Patterns
- Next lectures will explain refactoring techniques