Passing from design to implementation
Issues

- Technology choose
- OO programming language vs script language
- Automatic code generation
- Method implementation
- Property access methods
- Mapping abstract components to concrete classes
- Framework usage
- Code documentation
- Reverse engineering, back-synchronization
Technology choose

- Target execution environment
- Programming language
- Programming environment
- User interface technology
- Database technology
Choose criteria

- Environment availability (price, licenses)
- Knowledge of language and tools
- Need to fit to just existing solutions
- Need to fit to specific user requirements
## Language categories

**Object-oriented programming languages**

- data abstraction
- encapsulation
- modularity
- polymorphism
- inheritance

**Script languages**

- weak typing (unknown class properties and functions)
- flexible fitting to external (third-party made) components
- portability (wide usage)
- slower execution (interpretation or just-in-time compilation)
Automatic code generation

Skeleton code

```java
public class MyClass {
    private String fName;

    public String Name {
        get {
            throw new System.NotImplementedException();
        }
        set {
        }
    }

    public void Rename(String newName) {
        throw new System.NotImplementedException();
    }
}
```
Method implementation

• Parameter list completion
• Overloaded methods vs. default parameter values
• Virtual vs. abstract methods
• Optimization (final, sealed, const)
public int getValue() {
    return _value;
}

public void setValue(int x) {
    _value = x;
}

private int _value;
Mapping abstract design components to concrete implementation classes

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>ASP class</th>
<th>JSP class</th>
</tr>
</thead>
<tbody>
<tr>
<td>form</td>
<td>Page</td>
<td>Page</td>
</tr>
<tr>
<td>button</td>
<td>Button, ImageButton</td>
<td>Button</td>
</tr>
<tr>
<td>label</td>
<td>Label</td>
<td>Label</td>
</tr>
<tr>
<td>text box</td>
<td>TextBox</td>
<td>TextField</td>
</tr>
<tr>
<td>check box</td>
<td>CheckBox</td>
<td>CheckBox</td>
</tr>
</tbody>
</table>

needs adaptor to a specific framework
Framework usage

• Assumption: implementation acceleration
• Constraint: a framework should be well known
• Threats:
  – A framework can not be fitted to the project needs
  – Developers don’t know exactly how to use a framework:
    • Weak documentation
    • Unknown internal framework mechanisms
    • Incomprehensible source code
    • Lack of source code & obscured binary code
  – Sealed classes – no chance for modification
  – Project is late!
Code documentation

• Self-documentation – meaningful names
• Special documentation comments (documentation generation)
• Code structure overview description:
  – files and folders hierarchy
  – component list
  – class reference
Reverse engineering & back synchronization

- Code-based approach
- Model-based approach
Bibliography

• Roger S. Pressman: Software Engineering. A Practitioner's Approach (book, PDF)
• Coad, Yourdon: Object-Oriented Programming (book)