Pitfalls of AspectJ
Implementations of Some of the Gang-of-Four Design Patterns

Miguel Pessoa Monteiro
João Miguel Fernandes
Universidade do Minho, Portugal
Ph.D. Project

- Aims: develop a collection of refactorings (and code smells) for AOP code
- Approach: use of case studies to provide useful insights
- Second Case study: Java & AspectJ implementations of the 23 GoF design patterns:
Problems and Limitations Found

- Exposing the desirable context to the aspect - i.e. a single pointcut requiring context not readily available in a single point.
- Quantifying over objects without violating encapsulations (e.g. non-public fields and methods)
Problems and Limitations Found

- **Inflexible, awkward** or **complex** interfaces
  (case-specific aspects would be simpler and easier to use)

- Proprietary components onto which it is **illegal** to weave extra state and behaviour
  (i.e. the Decorator example by J. Cooper).

- Lack of genericity (generic types) – forces programmers to write case-specific code.
Conclusion

Aspects improve on many things, but...

- Aspects often improve **modularity**, but that does not necessarily result in **reusability**.
- Reusability comes at a **price**, even with AspectJ.
- In some cases, a good Java design may be preferable (e.g. J. Cooper’s Java Memento vs. Hannemann's AspectJ Memento).
Questions?

Miguel Pessoa Monteiro
eMail: mmonteiro@di.uminho.pt
URL: http://gec.di.uminho.pt/mpm/
Ph.D. Project

Method:

- Search for refactorings that transform the Java implementations into the AspectJ
- Pinpoint those refactorings (e.g. mechanics and code examples)
- Test and refine the refactorings with further Java examples (e.g. other Java implementations).