Patterns in Software Engineering

Lecturer: Raman Ramsin

Lecture 16

AntiPatterns

Part 1

AntiPatterns

- Compiled and presented by Brown et al. in 1998.
- "An AntiPattern describes a commonly occurring solution to a problem that generates decidedly negative consequences."
- The AntiPattern may be the result of a manager or developer:
  - not knowing any better,
  - not having sufficient knowledge or experience in solving a particular type of problem, or
  - having applied a perfectly good pattern in the wrong context.

AntiPatterns: Viewpoints

- **AntiPatterns** are presented from three perspectives – *developer*, *architect*, and *manager:*
  - ***Development AntiPatterns:*** comprise technical problems and solutions that are encountered by programmers.
  - ***Architectural AntiPatterns:*** identify and resolve common problems in how systems are structured.
  - ***Managerial AntiPatterns:*** address common problems in software processes and development organizations.

AntiPatterns: Development

- **The Blob:** Procedural–style design leads to one object with most of the responsibilities, while most other objects only hold data or simple operations.
- **Lava Flow:** Dead code and forgotten design information is frozen in an ever-changing design.
- **Ambiguous Viewpoint:** Object-oriented analysis and design models presented without clarifying the viewpoint represented by the model.
- **Functional Decomposition:** The output of nonobject–oriented developers who design and implement an application in an object–oriented language.
- **Poltergeists:** Classes with very limited roles and effective life cycles. They often start processes for other objects.

AntiPatterns: Development (Contd.)

- **Golden Hammer:** A familiar technology or concept applied obsessively to many software problems.
- **Spaghetti Code:** Ad hoc software structure makes it difficult to extend and optimize code.
- **Walking through a Minefield:** Using today's software technology is analogous to walking through a high-tech mine field: bugs abound.
- **Cut–and–Paste Programming:** Code reused by copying source statements leads to significant maintenance problems.
- **Mushroom Management:** Keeping system developers isolated from the system's end users.

AntiPatterns: Development – *The Blob*

- **The Blob:** Found in designs where one class monopolizes the processing, and other classes primarily encapsulate data.
- The key problem here is that the majority of the responsibilities are allocated to a single class which acts as a controller.
- **Solution:** Decompose the class and redistribute the responsibilities.

AntiPatterns: Development – *Lava Flow*

- **Lava Flow:** Dead code and forgotten design information is frozen in an ever-changing design.
- **Causes:**
  - R&D code placed into production without configuration management.
  - Uncontrolled distribution of unfinished code.
  - Implementation of several trial approaches for implementing a function.
  - Single-developer (lone wolf) design or written code.
  - Lack of configuration management or process management policies.
  - Lack of architecture, or non-architecture-driven development.
  - Repetitive development process.
  - Architectural scars: Architectural mistakes not removed.
- **To solve:** include a configuration management process that eliminates dead code and evolves or refactors design toward increasing quality.
- **To avoid:** ensure that sound architecture precedes code development.

AntiPatterns: Development – *Ambiguous Viewpoint*

- **Ambiguous Viewpoint:** Object-oriented analysis and design (OOA&D) models that are presented without clarifying the viewpoint represented by the model.
- There are three fundamental **viewpoints** for OOA&D models:
  - **Business** viewpoint (Problem-Domain/Conceptual/Essential)
  - **Specification** viewpoint (System)
  - **Implementation** viewpoint (Software/Design)
- By default, OOA&D models denote an implementation viewpoint that is potentially the least useful. Mixed viewpoints don't allow the fundamental separation of interfaces from implementation details.
- **Solution:** Separate Viewpoints explicitly.

AntiPatterns: Development – *Functional Decomposition*

- **Functional Decomposition:** The result of experienced, nonobject–oriented developers who design and implement an application in an object–oriented language.
- When developers are comfortable with a "main" routine that calls numerous subroutines, they may tend to make every subroutine a class, ignoring class hierarchy altogether.

- **Solution:** Redesign using OO principles:
  - *Solution 1:* Try to identify key problem-domain classes by developing an analysis model, translate it into a design model, and refactor.
  - *Solution 2:* Consider database entities as design classes, and refactor.
  - Although the above techniques may work, there is no straightforward way to resolve this problem.

AntiPatterns: Development – *Poltergeists*

- **Poltergeists:** Classes with limited responsibilities and roles to play in the system; therefore,
  - their effective life cycle is quite brief;
  - they clutter software designs, creating unnecessary abstractions;
  - They can be excessively complex, hard to understand, and hard to maintain.

- **Solution:** Remove them from the class hierarchy altogether. The functionality that was provided by it must be replaced;
  - Move the controlling actions initially encapsulated in the Poltergeist into the related classes that they invoked.

AntiPatterns: Development – *Golden Hammer*

- **Golden Hammer:** A Golden Hammer is a familiar technology or concept applied obsessively to many software problems.
- "When your only tool is a hammer, everything else is a nail."
- **Solution:**
  - expanding the knowledge of developers through education, training, and book study groups to expose developers to alternative technologies and approaches.

AntiPatterns: Development – *Spaghetti Code*

- **Spaghetti Code:** Ad hoc software structure makes it difficult to extend and optimize code.
  - Coding and progressive extensions have compromised the software structure to such an extent that the structure lacks clarity, even to the original developer.
  - If developed using an OO language, the software may include a small number of objects that contain methods with very large implementations.
  - The system is very difficult to maintain and extend, and there is no opportunity to reuse the objects and modules in other similar systems.
- **Solution:**
  - Clean up and restructure the code using reengineering.

AntiPatterns: Development – *Walking through a Minefield*

- **Walking through a Minefield:** Using today's software technology is analogous to walking through a high-tech mine field: Numerous bugs are found in released software products.
- **Solution:**
  - Proper investment in software testing is required to make systems relatively bug-free. In some progressive companies, the size of testing staff exceeds programming staff.
  - The most important change to make to testing procedures is configuration control of test cases.
  - automation of test execution and test design.

AntiPatterns: Development – *Cut−and−Paste Programming*

- **Cut−and−Paste Programming:** Code reused by copying source statements.
- It comes from the notion that it's easier to modify existing software than program from scratch.

- **Solution:**
  - Eliminate duplication through refactoring and reengineering.
  - Replace white-box reuse with black-box reuse.

AntiPatterns: Development – *Mushroom Management*

- **Mushroom Management:** In some architecture and management circles, there is an explicit policy to keep system developers isolated from the system's end users.
- Requirements are passed second-hand through intermediaries, including architects, managers, or requirements analysts.
- Motto: "Keep your developers in the dark and feed them fertilizer."
- Mushroom Management assumes that requirements are well understood by both end users and the software project at project inception. It is assumed that requirements are stable.
- **Solution:**
  - Risk-driven development: spiral development process based upon prototyping and user feedback.

Reference

- Brown, W. J., Malveau, R. C., McCormick, H., Mowbray, T., *Antipatterns: Refactoring Software, Architectures, and Projects in Crisis*. Wiley, 1998.