Software Development Methodologies

Lecturer: **Raman Ramsin**

Lecture 17: Process Antipatterns

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.
- **Process Antipatterns** deal with common problems in engineering or enacting a development process, and can belong to any of the above viewpoints.

Process Antipatterns: Development

- **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.
- **Golden Hammer:** A familiar technology or concept applied obsessively to many software problems.
- **Walking through a Minefield:** Using today's software technology is analogous to walking through a high-tech mine field: bugs abound.
- **Mushroom Management:** Keeping system developers isolated from the system's end users.

Process 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.

Process 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.

Process 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.

Process 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.

Process 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.

Process Antipatterns: Architectural

- **Cover Your Assets:** Document-driven software processes that produce less-than-useful requirements and specifications because the authors evade making important decisions.
- **Architecture by Implication:** the lack of architecture specifications for a system under development.
- **Design by Committee:** Design by Committee creates overly complex architectures that lack coherence.
- **Reinvent the Wheel:** The pervasive lack of experience transfer between software projects leads to substantial reinvention.
- **The Grand Old Duke of York:** Egalitarian software processes often ignore people's talents to the detriment of the project: We need *abstractionists* as well as *implementationists.*

Process Antipatterns: Architectural – *Cover Your Assets*

- **Cover Your Assets:** Document-driven software processes often produce less-than-useful requirements and specifications because the authors evade making important decisions.
  - In order to avoid making a mistake, the authors take a safer course and elaborate upon alternatives.

- **Solution:**
  - Enforce the production of Architecture blueprints: abstractions of information systems that facilitate communication of requirements and technical plans between the users and developers.
    - An architecture blueprint is a small set of diagrams and tables that communicate the operational, technical, and systems architecture of current and future extensions to information systems.
    - A typical blueprint comprises no more than a dozen diagrams and tables, and can be presented in an hour or less as a viewgraph presentation.

Process Antipatterns: Architectural – *Architecture by Implication*

- **Architecture by Implication:** the lack of architecture specifications for a system under development.
  - Usually, the architects responsible for the project have experience with previous system construction, and therefore assume that documentation is unnecessary.
  - Management of risk in follow-on system development is often overlooked due to overconfidence and recent system successes.
- **Solution:**
  - A general architecture definition approach that is tailored to each application system can help identify unique requirements and risk areas.

Process Antipatterns: Architectural – *Design By Committee*

- **Design by Committee:** The classic Antipattern from standards bodies, Design by Committee creates overly complex architectures that lack coherence:
  - A complex software design that is the product of a committee process.
  - It has so many features and variations that it is infeasible for any group of developers to realize the specifications in a reasonable time frame.
  - Even if the designs were possible, it would not be possible to test the full design due to excessive complexity, ambiguities, overconstraint, and other specification defects.
  - The design would lack conceptual clarity because so many people contributed to it and extended it during its creation.
- **Solution:**
  - Clarification of architectural roles and improved process facilitation can refactor bad meeting processes into highly productive events.

Process Antipatterns: Architectural – *Reinvent the Wheel*

- **Reinvent the Wheel:** The pervasive lack of experience transfer between software projects leads to substantial reinvention.
- "Our problem is unique."
- Virtually all systems development is done in isolation of projects and systems with overlapping functionality.
- **Solution:**
  - Design knowledge buried in legacy assets can be leveraged to reduce time-to-market, cost, and risk.

Process Antipatterns: Architectural – *Grand Old Duke of York*

- **The Grand Old Duke of York:** Egalitarian software processes often ignore people's talents to the detriment of the project.
  - Programming skill does not equate to skill in defining abstractions. There appear to be two distinct groups involved in software development: *abstractionists (Architects)* and their counterparts the *implementationists.*
  - According to experts, implementationists outnumber abstractionists approximately 4 to 1. Thus, unfortunately, abstractionists are often outvoted.
  - Primary consequence: software designs with excessive complexity, which make the system difficult to develop, modify, extend, document, and test.
  - Software usability and system maintenance are impacted by a failure to use effective abstraction principles.
- **Solution:**
  - Identifying and differentiating among distinct development roles, and giving architects control over architectural design.

Process Antipatterns: Management

- **Analysis Paralysis:** Striving for perfection and completeness in the analysis phase leading to project gridlock and excessive work on requirements/models.
- **Death by Planning:** Excessive planning for software projects leading to complex schedules that cause downstream problems.
- **Project Mismanagement:** Inattention to the management of software development processes causing directionlessness and other symptoms. Proper monitoring and control of software projects is necessary.

Process Antipatterns: Management – *Analysis Paralysis*

- **Analysis Paralysis:** Striving for perfection and completeness in the analysis phase often leads to project gridlock and excessive thrashing of requirements/models.
- Developers new to object-oriented methods do too much up-front analysis and design, using analysis modeling as an exercise to feel comfortable in the problem domain.
- A key indicator of Analysis Paralysis is that the analysis documents no longer make sense to the domain experts.
- **Solution:**
  - Iterative-incremental development processes that defer detailed analysis until the knowledge is needed.

Process Antipatterns: Management – *Death by Planning*

- **Death by Planning:** Excessive planning for software projects leading to complex schedules that cause downstream problems.

- **Solution:**
  - Deliverable-based planning, supplemented with validation milestones. Plans should be reviewed and revised on a weekly basis.

Process Antipatterns: Management – *Project Mismanagement*

- **Project Mismanagement:** Inattention to the management of software development processes can cause directionlessness and other symptoms.
- Proper monitoring and control of software projects is necessary for successful development activities.
- Often, key activities are overlooked or minimized. These include technical planning (architecture) and quality-control activities (inspection and test).

- **Solution:**
  - Proper risk management incorporated in the project management process.

References

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

- Neill, C.J., Laplante, P.A., *Antipatterns: Identification, Refactoring, and Management.* CRC Press, 2005.