Patterns in Software Engineering

Lecturer: Raman Ramsin

Lecture 17

AntiPatterns

<u>Part 2</u>

AntiPatterns: Architectural

- **Stovepipe System/Enterprise:** Subsystems/systems are integrated in an ad hoc manner using multiple integration strategies and mechanisms.
- **Cover Your Assets:** Document-driven software processes that produce less-than-useful requirements and specifications because the authors evade making important decisions.
- **Vendor Lock–In:** Vendor Lock–In occurs in systems that are highly dependent upon proprietary architectures.
- **Architecture by Implication:** the lack of architecture specifications for a system under development.

AntiPatterns: Architectural (Contd.)

- **Design by Committee:** Design by Committee creates overly complex architectures that lack coherence.
- **Swiss Army Knife:** An excessively complex interface.
- **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.*

AntiPatterns: Architectural – *Stovepipe System/Enterprise*

- **Stovepipe System/Enterprise:** Subsystems/systems are integrated in an ad hoc manner using multiple integration strategies and mechanisms.
- *Stovepipe* is a popular term used to describe software systems with ad hoc architectures.
  - The key problem in a Stovepipe System is the lack of common subsystem abstractions.
  - the key problem in a Stovepipe Enterprise is the absence of common multisystem conventions.
- **Solution:**
  - Enhance encapsulation and introduce common abstractions through layered architectures.

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.

AntiPatterns: Architectural – *Vendor Lock−In*

- **Vendor Lock−In:** Occurs in systems that are highly dependent upon proprietary architectures.
- A software project adopts a product technology and becomes completely dependent upon the vendor's implementation.
  - When upgrades are done, software changes and interoperability problems occur, and continuous maintenance is required to keep the system running.
  - Expected new product features are often delayed, causing schedule slips and an inability to complete desired application software features.
- **Solution:**
  - Introduce an *isolation layer* that separates software packages and technology.

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.

AntiPatterns: Architectural – *Design By Committee*

- **Design by Committee:** The classic AntiPattern from standards bodies;
  - Creates overly complex architectures that lack coherence.
  - 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 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.

AntiPatterns: Architectural – *Swiss Army Knife*

- **Swiss Army Knife:** An excessively complex interface.
- The designer attempts to provide for all possible uses of the class. In the attempt, he or she adds a large number of interface signatures in an attempt to meet all possible needs.
- Prevalent in commercial software interfaces, where vendors are attempting to make their products applicable to all possible applications.
- **Solution:**
  - Define a clear purpose for the component and properly abstract the interface to manage complexity.
    - Wrap the Interface in simplifying adapters. Apply the Interface Segregation Principle (ISP).

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.

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.

AntiPatterns: Management

- **Analysis Paralysis:** Striving for perfection and completeness in the analysis phase leading to project gridlock and excessive work on requirements/models.
- **Viewgraph Engineering:** On some projects, developers become stuck preparing viewgraphs and documents instead of developing software.
- **Death by Planning:** Excessive planning for software projects leading to complex schedules that cause downstream problems.
- **Fear of Success:** Often occurs when people and projects are on the brink of success. Some people begin to worry obsessively about the kinds of things that *can* go wrong*.*

AntiPatterns: Management (Contd.)

- **Corncob:** Difficult people frequently obstruct and divert the software development process.
- **Intellectual Violence:** Intellectual violence occurs when someone who understands a theory, technology, or buzzword uses this knowledge to intimidate others in a meeting situation.
- **Smoke and Mirrors:** Demonstration systems are important sales tools, but they are often interpreted by end users as representational of production-quality capabilities.
- **Project Mismanagement:** Inattention to the management of software development processes causing directionlessness and other symptoms.

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.

AntiPatterns: Management – *Viewgraph Engineering*

- **Viewgraph Engineering:** Developers become stuck preparing viewgraphs and documents instead of developing software.
- Organizations with limited technical capabilities for system development are taken at face value because they produce substantive documents and polished briefings.
- **Solution:**
  - Verify the development capabilities of the organization and key project staff.
  - Utilize prototyping and mock–ups as part of any system development 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.

AntiPatterns: Management – *Fear of Success*

- **Fear of Success:** Often occurs when people and projects are on the brink of success.
- Some people begin to worry obsessively about the kinds of things that *can* go wrong.
- **Solution:**
  - When project completion is imminent, make a clear declaration of success.

AntiPatterns: Management – *Corncob*

- **Corncob:** Difficult people frequently obstruct and divert the software development process.
- This attitude can be due to aspects of individual personality, but often, difficulties arise from personal motivations for recognition or monetary incentives.
- **Solution:** Address agendas of the individual through various tactical, operational, and strategic organizational actions.
  - *Transfer the responsibility.*
  - *Isolate the issue.*
  - *Question the question.*
  - *Corrective interview.*
  - *Friendly outplacement.*
  - *Corncob support group.*
  - *Empty department.*
  - *Reduction in force.*

AntiPatterns: Management – *Intellectual Violence*

- **Intellectual Violence:** Intellectual violence occurs when someone who understands a theory, technology, or buzzword uses this knowledge to intimidate others in a meeting situation.

- **Solution:**
  - Encourage education and practice mentoring throughout the organization.

AntiPatterns: Management – *Smoke and Mirrors*

- **Smoke and Mirrors:** Demonstration systems are important sales tools, but they are often interpreted by end users as representational of production-quality capabilities.
- **Solution:**
  - Practice proper ethics to manage expectations, risk, liabilities, and consequences in computing sales and marketing situations.

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.

Reference

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