Software Architecture Anti-Patterns
What are Anti-patterns?

- “An AntiPattern describes a commonly occurring solution to a problem that generates decidedly negative consequences.”
- Happens because an architect...
  - Does not have sufficient knowledge or experience solving a particular problem
  - Applied a perfectly good design pattern in the wrong context
Examples - 1

- **Jumble**
  *Horizontal* and *vertical* design elements are intermixed (ball of mud). The result is unstable, and limits reusability. The layer pattern is violated.

- **Stovepipe**
  *External systems* and/or *internal subsystems* are *integrated* in an *ad hoc point to point* manner using multiple integration strategies and mechanisms. It is characterized by a *lack of coordination and planning*, extensibility and support are difficult.

- **Cover Your Assets**
  *Less-than-useful requirements* are produced because important decisions are avoided and alternatives are *elaborated*. Obfuscates architecture design
Examples - 2

- **Vendor Lock-In** - systems are **highly dependent upon proprietary architectures**. Architectural isolation layers can provide independence from vendor-specific solutions.

- **Wolf Ticket**  
  A product **claims openness and conformance to unenforceable standards**. Interfaces may vary significantly from the published standard. Marketing motivated (term comes from rock concert ticket scalping)

- **Architecture by Implication**  
  Lack of architecture planning and documentation due to architect over confidence or incompetence leads to implementation risks
Examples - 3

- **Design by Committee**
  Design by Committee creates overly complex architectures that lack coherence. Clarification of architectural roles and improved process facilitation can refactor bad meeting processes into highly productive events.

- **Swiss Army Knife**
  An excessively complex component interface. The designer attempts to provide for all possible uses of the component.

- **Reinvent the Wheel**
  Pervasive lack of technology transfer between software projects leads to substantial reinvention. Design knowledge buried in legacy assets can be leveraged to reduce time-to-market, cost, and risk.

- **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. Distinguish between programmers and design modelers.
References

- AntiPatterns, Muller, University of Victoria, http://www.csc.uvic.ca/~hausi/480/lectures/antipatterns.pdf
- https://sourcemaking.com/antipatterns/software-architecture-antipatterns