Analysis Paralysis

- **AntiPattern Name:** Analysis Paralysis
- **Also Known As:** Waterfall, Process Mismatch
- **Most Frequent Scale:** System
- **Refactored Solution Name:** Iterative-Incremental Development
- **Refactored Solution Type:** Software
- **Root Causes:** Pride, Narrow-Mindedness
- **Unbalanced Forces:** Management of Complexity
- **Anecdotal Evidence:** "We need to redo this analysis to make it more object-oriented, and use much more inheritance to get lots of reuse." "We need to complete object-oriented analysis, and design before we can begin any coding." "Well, what if the user wants to create the employee list based on the fourth and fifth letters of their first name combined with the project they charged the most hours to between Thanksgiving and Memorial Day of the preceding four years?" "If you treat each object attribute as an object, you can reuse field formatting between unrelated classes."

Background

Analysis Paralysis is one of the classic AntiPatterns in object-oriented software development. Object-oriented analysis is focused on decomposing a problem into its constituent parts, but there is no obvious method for identifying the exact level of detail necessary for system design Frequently, the focus is shifted from decomposing to a level where the problem can be easily understood by the designers to applying the techniques to achieve the mythical "completeness."

Also, system developers often willingly fall prey to Analysis Paralysis, as "designs never fail, only implementations." By prolonging the analysis and design phases, they avoid risking accountability for results. Of course, this is a losing strategy, because usually there is some point after which a working implementation is expected.

General Form

Analysis Paralysis occurs when the goal is to achieve perfection and completeness of the analysis phase. This AntiPattern is characterized by turnover, revision of the models, and the generation of detailed models that are less than useful to downstream processes.

Many developers new to object-oriented methods do too much up-front analysis and design. Sometimes, they use analysis modeling as an exercise to feel comfortable in the problem domain. One of the benefits of object-oriented methods is developing analysis models with the participation of domain experts. Otherwise, it's easy to get bogged down in the analysis process when the goal is to create a comprehensive model.

Analysis Paralysis usually involves waterfall assumptions:

- Detailed analysis can be successfully completed prior to coding.
- Everything about the problem is known a priori.
- The analysis models will not be extended nor revisited during development. Object-oriented development is poorly matched to waterfall analysis, design, and implementation processes. Effective object-oriented development is an incremental process during which incremental and iterative results of analysis are validated through design and implementation and used as feedback into later system analysis.

A key indicator of Analysis Paralysis is that the analysis documents no longer make sense to the domain experts. As Analysis Paralysis progresses, the analysis models more frequently cover details that are of no interest to the domain experts. For example, the domain model of a health care system for a hospital should be understandable by the administrators and staff of the hospital.

If the domain model defines unexpected software concepts, categories, and specializations, the analysis modeling has probably gone too far. If the meaning of these new classes has to be explained in detail to the people intimately familiar with the current system, it is likely that the problem has been overanalyzed.

Symptoms And Consequences

- There are multiple project restarts and much model rework, due to personnel changes or changes in project direction.
- Design and implementation issues are continually reintroduced in the analysis phase.
- Cost of analysis exceeds expectation without a predictable end point.
- The analysis phase no longer involves user interaction. Much of the analysis performed is speculative.
- The complexity of the analysis models results in intricate implementations, making the system difficult to develop, document, and test.
- Design and implementation decisions such as those used in the Gang of Four design patterns are made in the analysis phase.

Typical Causes

- The management process assumes a waterfall progression of phases. In reality, virtually all systems are built incrementally even if not acknowledged in the formal process.
- Management has more confidence in their ability to analyze and decompose the problem than to design and implement.
- Management insists on completing all analysis before the design phase begins.
- Goals in the analysis phase are not well defined.
- Planning or leadership lapses when moving past the analysis phase.
- Management is unwilling to make firm decisions about when parts of the domain are sufficiently described.

- The project vision and focus on the goal/deliverable to customer is diffused. Analysis goes beyond providing meaningful value.

Known Exceptions

There should never be any exceptions to the Analysis Paralysis AntiPattern.

Refactored Solution

Key to the success of object-oriented development is incremental development. Whereas a waterfall process assumes a priori knowledge of the problem, incremental development processes assume that details of the problem and its solution will be learned in the course of the development process.

In incremental development, all phases of the object-oriented process occur with each iteration—analysis, design, coding, test, and validation. Initial analysis comprises a high-level review of the system so that the goals and general functionality of the system can be validated with the users. Each increment fully details a part of the system.

There are two kinds of increments: internal and external. An *internal increment* builds software that is essential to the infrastructure of the implementation. For example, a third-tier database and data-access layer would comprise an internal increment. Internal increments build a common infrastructure that is utilized by multiple use cases. In general, internal increments minimize rework. An *external increment* comprises user-visible functionality.

External increments are essential to winning political consensus for the project by showing progress. External increments are also essential for user validation. They usually involve some throwaway coding to simulate missing parts of the infrastructure and back-end tiers. It's the prerogative of the project manager to select the increments that balance the forces of project survival, user validation, and minimal cost. See the Project Mismanagement AntiPattern regarding scheduling of increments with regard to risk.

Frequently, when performing object-oriented analysis, it's easier to continue the analysis than to end it and move on to the design of the software. Since many of the analysis techniques are also applied at some level to the design, it's easy to use the analysis phase to direct the specifics of the overall design.

Sometimes, this also results from thinking that minimal analysis is needed concurrently with design, and that design can be started then gone back to. Typically, this results in a product that neither resembles a domain model a user can understand nor a desirable design for an implemented system.

Analysis Paralysis is also applicable at the architectural level and takes on a similar form to the AntiPattern at the developmental level. An architecture can certainly be specified far beyond what is necessary for the developers to adhere to the architectural principles and constructs.

For example, specifying the known and allowable subclasses used by an architectural construct designed to operate solely on the object's base class interface is overkill. Rather, it is preferable to specify the necessary constraints in the base class on which the architectural component operates and trust (and perhaps also verify) the developers to maintain the constraints in their subclasses.

At the managerial level, Analysis Paralysis is known as micromanagement and occurs when managers overspecify assignments and oversupervise delegated assignments.