Architecture By Implication

- **AntiPattern Name:** Architecture by Implication
- **Also Known As:** Wherefore art thou architecture?
- **Most Frequent Scale:** System
- **Refactored Solution Name:** Goal Question Architecture
- **Refactored Solution Type:** Documentation
- **Root Causes:** Pride, Sloth
- **Unbalanced Forces:** Management of Complexity, Change, and Risk
- **Anecdotal Evidence:** "We've done systems like this before!" "There is no risk; we know what we're doing!“

Background

Dwight Eisenhower said that planning is essential, but plans are inconsequential. Another soldier said that no plans survive first contact with the enemy. The planning culture in modern management owes some credit to Robert McNamara, founder of the RAND Corporation.

In McNamara's approach, plans are generated for speculative purposes, to investigate the potential benefits and consequences of different courses of action. Given the large number of unknowns in systems development, planning for IT systems must be more pragmatic and iterative.

One professional planner said that 20 percent of an engineer's time should be devoted to planning. As we gain experience, our belief in this assertion increases. Productivity and efficiency can be greatly amplified when the work is well organized through planning.

The unfortunate consequence is that many organizations attempt to formalize too much of the planning. Planning is most effective when it is personally motivated and utilized. Time management experts teach that a key element of stress reduction is planning to balance life's overall priorities. The form and use of time-management systems becomes increasingly personalized as the practice matures.

A group of CEOs from DoD Systems integration firms was formed to answer the question, "Wherefore art thou architecture?" The goal was to reflect on the changing nature of systems development, which has evolved into the reuse of existing legacy components and commercial software, and away from greenfield, custom code development (see the Reinvent the Wheel AntiPattern).

General Form

This AntiPattern is characterized by 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.

This overconfidence leads to exacerbated risks in key areas that affect system success. Architecture definitions are often missing from one or more of these areas:

- Software architecture and specifications that include language use, library use, coding standards, memory management, and so forth.
- Hardware architecture that includes client and service configurations.
- Communications architecture that includes networking protocols and devices.
- Persistence architecture that includes databases and file-handling mechanisms.
- Application security architecture that includes thread models and trusted system base.
- Systems management architecture.

Symptoms And Consequences

- Lack of architecture planning and specification; insufficient definition of architecture for software, hardware, communications, persistence, security, and systems management.
- Hidden risks caused by scale, domain knowledge, technology, and complexity, all of which emerge as the project progresses.
- Impending project failure or unsuccessful system due to inadequate performance, excess complexity, misunderstood requirements, usability, and other system characteristics. For example, approximately 1 of 3 systems encounter serious performance problems during development and operations.
- Ignorance of new technologies.
- Absence of technical backup and contingency plans.

Typical Causes

- No risk management.
- Overconfidence of managers, architects, and/or developers.
- Reliance on previous experience, which may differ in critical areas.
- Implicit and unresolved architecture issues caused by gaps in systems engineering.

Known Exceptions

The Architecture by Implication AntiPattern is acceptable for a repeated solution, where there are minor differences in the code, such as installation scripts. This AntiPattern may also be useful in a new project domain as an exploratory effort to determine whether existing techniques are transferable into a new area.

Refactored Solution

The refactored solution to the Architecture by Implication AntiPattern entails an organized approach to systems architecture definition, and relies on multiple views of the system. Each view models the system from the perspective of a system stakeholder, who may be real or imaginary, individual or aggregate. Each stakeholder is responsible for a high-priority set of questions and issues, and each view represents the entire information system and answers these key questions and issues.

The views comprising a set of diagrams, tables, or specifications, are linked for consistency. Generally, a view is a lightweight specification. The purpose of the architecture documentation is to communicate architecture decisions and other issues resolutions. The documentation should be easy to understand and inexpensive to maintain.

That said, the only people who can define and implement a successful architecture are those who already fully understand it. Unfortunately, this is often not the case, as many projects adopt some new technology that is not well understood. Therefore, developing a good architecture from scratch is an iterative process and should be recognized as such.

An initial reference architecture should have strong strategies that can be implemented within the time period of the first product development. Thereafter, it will have to be incrementally refined by future versions of the reference architecture, and driven by new versions of the first product or new products.

The steps to define a system architecture using viewpoints are as follows :

1. *Define the architecture goals*. What must this architecture achieve? Which stakeholders, real and imaginary, must be satisfied with the design and implementation? What is the vision for the system? Where are we now and where are we going?

2. *Define the questions*. What are the specific questions that must be addressed to satisfy the stakeholder issues? Prioritize the questions to support view selection.

3. *Select the views*. Each view will represent a blueprint of the system architecture.

4. *Analyze each view*. Detail the architecture definition from each viewpoint. Create the system blueprints.

5. *Integrate the blueprints*. Verify that the views present a consistent architecture definition.

6. *Trace views to needs*. The views should address the known questions and issues to discover any gaps not addressed by the architecture specifications. Validate the architecture with respect to formal requirements. Prioritize the outstanding issues.

7. *Iterate the blueprints*. Refine the views until all questions, issues, and gaps are resolved. Utilize review processes to surface any remaining issues. If a significant number of unresolved issues remain, consider creating additional views.

8. *Promote the architecture*. Make an explicit effort to communicate the architecture to key stakeholders, particularly the system developers. Create lasting documents (such as a video tutorial) to provide valuable information throughout the development and maintenance life cycle.

9. *Validate the implementation*. The blueprints should represent an "as-built" design. Determine any deltas between the blueprints and the system implementation. Decide whether these differences should result in system modifications of updates to the blueprints. Upgrade the documentation for consistency.

We refer to this method as the goal-question architecture (GQA), analogous to the goal-question metric approach in software metrics

Variations

A number of approaches consider the system architecture using viewpoints; in some, the viewpoints are predefined. Most of these approaches are open-ended, in that one can select additional viewpoints as described.

The Reference Model for Open Distributed Process (RM-ODP) is a popular, useful standard for distributed architectures. RM-ODP defines five standard viewpoints: enterprise, information, computational, engineering, and technology It also defines a useful set of transparency properties for distributed infrastructure through the engineering viewpoint.

Another approach, the Zachman Framework, analyzes system architectures from the perspectives of data, function, and network Within each perspective are multiple levels of abstraction, corresponding to the planning needs of various groups of stakeholders. Enterprise

Architecture Planning is an approach based upon the Zachman Framework for large-scale systems Neither of these approaches is tailored to object-oriented systems development.

A third approach, the Command, Communication, Control, Computer, Intelligence, Surveillance, and Reconnaissance Architecture Framework (C4ISR-AF), is used to define various command and control system architectures. A version of C4ISR-AF is used for other types of civilian systems. This approach has been very beneficial in enabling communications between architects across disparate domains

A fourth, the 4 + 1 Model View, is a viewpoint-based architecture approach supported by software engineering tools, such as Rational Rose The viewpoints include logical, use-case, process, implementation, and deployment. Finally, GQA is a generalization of the underlying method used in several of these architecture approaches

Example

A common but bad practice is object-oriented modeling without defining the viewpoint. In most modeling approaches, there is a blurring of the viewpoints. Many of the modeling constructs contain implementation detail, and the default practice is to intermingle implementation and specification constructs.

Three fundamental viewpoints are: conceptual, specification, and implementation The*conceptual viewpoint* defines the system from the perspective of the user. This is typically referred to as an *analysis model*. The distinction between what is automated and what is not is usually not represented in the model; rather, the model is drawn so that a user can explain and defend it to his or her peers.

The *specification viewpoint* concerns only interfaces. ISO IDL is one important notation that is strictly limited to defining interface information and excludes implementation specifics. The separation of interfaces from implementations enables the realization of many important object technology benefits, such as reuse, system extension, variation, substitutability, polymorphism, and distributed object computing. The final viewpoint, *implementation,* is best represented by the source code.

Complex implementation structures are beneficially augmented with object-oriented design models to help current and future developers and maintainers understand the code.

Another example of the Architecture by Implication AntiPattern is the following, where the key stakeholders did not have collective experience in what was built. The project was intended to deliver a Microsoft Distributed Common Object Model (DCOM)-based solution to extract legacy mainframe data, filter it based on business rule, and display it on Web pages.

However, the manager was a good software engineer with no distributed object technology (DOT) experience and the architect was a "dyed-in-the-wool" CORBA addict who helped the OMG derive its Object Management Architecture. To compound the problem, the project had few DCOM-aware staff; less than 10 percent.

In addition, the architecture and subsequent design were based on the OMA view of the DOT world, rather than DCOM. This led to an attempt to deliver CORBA services under a DCOM architecture. The resulting product suffered from a set of components that had no DOT consistency and were poor performers. Also, SIs found it very difficult to use, due to lack of a standardized approach. Finally, it failed in the marketplace.

Related Solutions

Architecture by Implication AntiPattern differs from the Stovepipe Systems AntiPattern in scope; the latter focuses on deficiencies in computational architecture. In particular, it identifies how improper abstraction of subsystem APIs leads to brittle architecture solutions. In contrast, the Architecture by Implication AntiPattern involves planning gaps constituted of multiple architecture viewpoints.

Applicability To Other Viewpoints And Scales

This AntiPattern significantly increases risk for managers, who defer important decisions until failures occur; often, it is too late to recover. Developers suffer from a lack of guidance for system implementation.

They are given de facto responsibility for key architectural decisions for which they may not have the necessary architectural perspective. Systemwide consequences of interface design decisions should be considered; in particular: system adaptability, consistent interface abstractions, metadata availability, and management of complexity.

Another important result of this AntiPattern is the deferment of resource allocation. The essential tools and technology components may not be available when needed due to lack of planning.