Golden Hammer

- **AntiPattern Name:** Golden Hammer
- **Also Known As:** Old Yeller, Head-in-the sand
- **Most Applicable Scale:** Application
- **Refactored Solution Name:** Expand your horizons
- **Refactored Solution Type:** Process
- **Root Causes:** Ignorance, Pride, Narrow-Mindedness
- **Unbalanced Forces:** Management of Technology Transfer
- **Anecdotal Evidence:** "I have a hammer and everything else is a nail." "Our database is our architecture." "Maybe we shouldn't have used Excel macros for this job after all."

Background

This is one of the most common AntiPatterns in the industry. Frequently, a vendor, specifically a database vendor, will advocate using its growing product suite as a solution to most of the needs of an organization. Given the initial expense of adopting a specific database solution, such a vendor often provides extensions to the technology that are proven to work well with its deployed products at greatly reduced prices.

General Form

A software development team has gained a high level of competence in a particular solution or vendor product, referred to here as the Golden Hammer. As a result, every new product or development effort is viewed as something that is best solved with it. In many cases, the Golden Hammer is a mismatch for the problem, but minimal effort is devoted to exploring alternative solutions.

This AntiPattern results in the misapplication of a favored tool or concept. Developers and managers are comfortable with an existing approach and unwilling to learn and apply one that is better suited. This is typified by the common "our database is our architecture" mind-set, particularly common in the world's banking community.

Frequently, an advocate will propose the Golden Hammer and its associated product suite as a solution to most of the needs of an organization. Given the initial expense of adopting a specific solution, Golden Hammer advocates will argue that future extensions to the technology, which are designed to work with their existing products, will minimize risk and cost.

Symptoms And Consequences

- Identical tools and products are used for wide array of conceptually diverse products.
- Solutions have inferior performance, scalability, and so on when compared to other solutions in the industry.
- System architecture is best described by a particular product, application suite, or vendor tool set.
- Developers debate system requirements with system analysts and end users, frequently advocating requirements that are easily accommodated by a particular tool and steering them away from areas where the adopted tool is insufficient.
- Developers become isolated from the industry. They demonstrate a lack of knowledge and experience with alternative approaches.
- Requirements are not fully met, in an attempt to leverage existing investment.
- Existing products dictate design and system architecture.
- New development relies heavily on a specific vendor product or technology.

Typical Causes

- Several successes have used a particular approach.
- Large investment has been made in training and/or gaining experience in a product or technology.
- Group is isolated from industry, other companies.

- Reliance on proprietary product features that aren't readily available in other industry products.
- "Corncob" proposing the solution (see Corncob AntiPattern).

Known Exceptions

The Golden Hammer AntiPattern sometimes works:

1. If the product that defines the architectural constraints is the intended strategic solution for the long term; for example, using an Oracle database for persistent storage and wrapped stored procedures for secure access to data.
2. If the product is part of a vendor suite that provides for most of the software needs.

Refactored Solution

This solution involves a philosophical aspect as well as a change in the development process. Philosophically, an organization needs to develop a commitment to an exploration of new technologies.

Without such a commitment, the lurking danger of overreliance on a specific technology or vendor tool set exists. This solution requires a two-pronged approach: A greater commitment by management in the professional development of their developers, along with a development strategy that requires explicit software boundaries to enable technology migration.

Software systems need to be designed and developed with well-defined boundaries that facilitate the replaceability of individual software components. A component should insulate the system from proprietary features in its implementation.

If the system is developed using explicit software boundaries, the interfaces that make up the boundaries become points at which the software used in the implementation may be replaced with a new implementation, without affecting the other components

in the system. An industry standard, such as the OMG IDL specification, is an invaluable tool for incorporating rigid software boundaries between components.

In addition, software developers need to be up to date on technology trends, both within the organization's domain and in the software industry at large. This can be accomplished through several activities that encourage the interchange of technical ideas. For example, developers can establish groups to discuss technical developments (design patterns, emerging standards, new products) that may impact the organization in the future.

They can also form book study clubs to track and discuss new publications that describe innovative approaches to software development. In practice, we have found the book study club paradigm to be a very effective way to exchange ideas and new approaches.

Even without full management buyin, developers can establish informal networks of technology-minded people to investigate and track new technologies and solutions. Industry conferences are also a great way to contact people and vendors and stay informed as to where the industries are headed and what new solutions are available to developers.

On the management side, another useful step is to adopt a commitment to open systems and architectures. Without it, developers often acquire the attitude that achieving short-term results by any means necessary is acceptable. Though this may be desirable in the short term, future results become problematic because rather than building upon a solid foundation of past successes, effort is expended reworking past software to conform to new challenges.

Flexible, reusable code requires an investment in its initial development, otherwise long-term benefits will not be achieved Also, the danger of overreliance on a specific technology or vendor tool set is a potential risk in the product or project development. In-house research programs that develop proof-of-concept prototypes are effective for testing the feasibility of incorporating less risky open technologies into a development effort.

Another management-level way of eliminating or avoiding the Golden Hammer AntiPattern is to encourage the hiring of people from different areas and from

different backgrounds. Teams benefit from having a broader experience base to draw upon in developing solutions. Hiring a database team whose members all have experience in using the same database product greatly limits the likely solution space, in comparison to a similar team whose experience encompasses a wide range of database technology solutions.

Finally, management must actively invest in the professional development of software developers, as well as reward developers who take initiative in improving their own work.

Variations

A common variation of Golden Hammer occurs when a developer uses a favorite software concept obsessively. For example, some developers learn one or two of the GoF patterns and apply them to all phases of software analysis, design, and implementation.

Discussions about intent or purpose are insufficient to sway them from recognizing the applicability of the design pattern's structure and force-fitting its use throughout the entire development process. Education and mentoring is required to help people become aware of other available approaches to software system construction.

Example

A common example of the Golden Hammer AntiPattern is a database-centric environment with no additional architecture except that which is provided by the database vendor. In such an environment, the use of a particular database is assumed even before object-oriented analysis has begun. As such, the software life cycle frequently begins with the creation of an entity-relationship (E-R) diagram that is produced as a requirements document with the customer.

This is frequently destructive, because the E-R diagram ultimately is used to specify the database requirements; and detailing the structure of a subsystem before understanding and modeling the system presumes that the impact of the actual customer requirements on the system design is minimal.

Requirements gathering should enable system developers to understand the user needs to the extent that the external behavior of the solution system is understood by the user as a black box Conceivably, many systems are built to satisfy user requirements without utilizing a database at all. However, with the Golden Hammer AntiPattern in force, such possibilities are discounted up front, leading to every problem incorporating a database element.

Over time, the organization may develop several database-centric products that could have been implemented as independent systems. The database evolves into the basis for interconnectivity between applications, and it manages distribution and shared access to data. In addition, many implementation problems are addressed through using database proprietary features that commit future migrations to parallel the development of a technology of the implementation database.

At some point, it may be necessary to interoperate with systems that either do not share the same database-centric architecture or are implemented using a different database that may not permit unrestricted access to their information. Suddenly, development becomes extremely expensive as unique, special-purpose connections are built to "bridge" between unique systems. If, however, some thought is given to the problem before the situation gets too far out of hand, a common framework can be developed, where products chosen for particular areas are selected based on standard interface specifications, such as CORBA, DCOM, or TCP/IP.

Another example is an insurance company with several stovepipe legacy systems that decided in its move to client/server that Microsoft Access should be the key part of the solution for persistence. The entire front end of the call-center system was architected around an early version of this product. Thereafter, the system's future was fully constrained by the development path of the database product because of a bad architecture decision. Needless to say, the system lasted less than six months.

Related Solutions

- *Lava Flow.* This AntiPattern results when the Golden Hammer AntiPattern is applied over the course of several years and many projects. Typically, older sections based on earlier versions of the Golden Hammer are delegated to remote, seldom-used parts of the overall application. Developers become

reluctant to modify these sections, which build up over time and add to the overall size of the application while implementing functions that are seldom, if ever, used by the customer.

- *Vendor Lock-In*. Vendor lock-in is when developers actively receive vendor support and encouragement in applying the Golden Hammer AntiPattern. A software project is actively committed to relying upon a single vendor's approach in designing and implementing an object-oriented system.