MITIGATING PROJECT RISK BY ADOPTING ACCELERATED DELIVERY TECHNIQUES

Description

OVERVIEW

In the process of gaining approval for and initiating projects, many companies create a major risk of failure by committing to a scope, schedule and budget prematurely. This often comes about because project sponsors are disinclined or disincentivized to navigate the internal funding process more than once so they take a rough project definition and attach a heavily padded estimate to it.

ONCE FUNDING IS APPROVED

The day the funding is approved a time and resource envelope is created in which the project team will struggle to deliver as much functionality with as high a quality level as is possible before time or money runs out. This suboptimal but extremely common practice is often rationalized by the wishful notion that the project will be conducted using an Agile methodology to accelerate the requirements analysis, design and delivery process.

To the degree that no project starts in a state of perfect knowledge, there are some good reasons for accepting a calculated amount of this risk. It may be that exogenous circumstances, such as impending regulation or imminent business opportunity justify it, but in many cases it is simply the perpetuation of historical practice that leads enterprises to act this way.

Over the course of development both functionality and approaches to delivering it evolve, which is a normal and expected thing. However, a risk that this creates in many projects is of producing code with less-than-optimal organization, consistency and quality due to the reticence or inability to allocate resources to refactor the existing code base as changes are accommodated and new code is added to it.

The potential downsides relating to these issues include:

- Cost and schedule overruns.
- Scope (functionality) shortfalls.
- Quality problems.
- Suboptimal architecture, code organization or consistency.
- Shorter life expectancy for the application.
- Excessive cost of maintaining and operating it over its lifetime, inflating the Total Cost of Ownership (TCO).

WHAT TO LOOK FOR IN A SOLUTION

There are a number of alternatives to consider in deciding how to address these issues. Ultimately, the goal is to find a balance between project management and execution orthodoxy and a sponsoring organizationâ€™s needs and preferences. The universal need for quick and cost-controlled project deliveries dictates a compromise that balances the project risks associated with methodology shortcuts against realizing benefits as quickly as possible.

In selecting an approach to address this, the objectives are:

- Increasing focus on as broad a scope of analysis as possible *before* implementation.
- Improving requirements analysis and problem abstraction disciplines, leading to more coherent architecture and easier implementation.
- Developing a clearer understanding of solution requirements and interdependencies in order to provide a basis for feature cost-benefit analysis and developing risk mitigation strategy.
- Increasing use of prototyping to perfect the solution before building it in large scale.
- Increasing the ability to iterate the implementation and make refactoring easier and faster.

Both traditional waterfall and Agile approaches represent trade-offs against these goals. The need to meet milestone delivery dates or produce working prototypes often conflicts with resource requirements for some of the behind-the-scenes work of discovery, analysis and detailed design. **The ability to accelerate architectural-level solution evolution and project changes into the in-process application code is crucial to maintaining the solutionâ€™s integrity while it is in development.** Minimizing impediments to doing this is the key to tipping the balance between optimizing resource constraints and maintaining solution quality in your favor.

ONE ANSWER: GENERATIVE SOFTWARE ENGINEERING

Here is a solution that can contribute to meeting this challengeâ€”***generative software engineering***. In generative software engineering ***specifications*** (usually metadata and other information) and ***templates*** (tokenized code models) are used to define and then generate a large proportion of the solution code. The application functions that are most immediately amenable to this are those that are data-driven; for instance, marshaling code that sits between an application and data sources with which it interacts. However, with increasing experience with the tools and methodology, a much wider range of design patterns can be implemented just as easily.

If a data source undergoes a structure change, the marshaling code and anything in the application synchronized with the data structure must change. Automating the process of making and propagating the necessary changes (a) reduces the possibility of errors in what is usually the most error-prone coding task of any development, (b) ensures consistency across related application components, (c) eliminates tedious and wasteful use of coding resources and (d) greatly accelerates the process of modifying the code, allowing for more iterations in the same amount of time than traditional coding practices would.

Even more importantly, the process of defining the specifications that describe the solution domain and identifying the various permutations of objects and components required in the solution promotes an architectural view rather than an implementation-oriented one. Therefore, evolving the solution in development becomes a matter of revising the architectural definition (updating specifications and/or templates), prototyping, testing and perfecting the revised components and then regenerating the solution code. Architectural integrity is maintained, errors are eliminated, coding resource utilization is optimized and refactoring is accomplished without forcing the types of trade-offs that often compromise project results.

Finally, the disciplines involved in generative software engineering combined with the right tool set result in reusable specification repository designs, template structures and implementation-specific templates that are easily transferrable from project to project. Once a generative software team develops a browser-based interface for a data warehouse, for instance, implementing one for another

warehouse is accelerated substantially. The knowledge behind the solution is embodied in the work of the senior programmers that implemented the initial instance, enabling others to produce subsequent ones.

GENERATIVE SOFTWARE ENGINEERING TOOLS

There are a number of tools available today that provide generative engineering support. Many of them have significant limitationsâ€”they operate in narrow domains, i.e., they generate code only for specific languages, integrate with only certain data repositories or support specifications models with limited flexibility. In some cases, they require significant coding to generate even the simplest output.

CodiScent provides tools, methodology and professional services that produce Better, Cheaper and Faster software delivery. In addition, projects built using CodiScent include the benefits of coherent architecture, consistent code bases and optimized artifact reusability. Finally, CodiScent provides a broad spectrum of service and product options that allow clients to work with CodiScent in the way that makes the most sense for them, including everything from hands-off, turnkey development to product adoption and staff training in the technology and methodology.

DISCLOSURE

The author and Evolution Path Associates, Inc. have a mutually beneficial business relationship with CodiScent, Ltd.

Date Created

2013/06/26

Author

howardmwiener