Week 1: Compare Software Engineering Models

Nate Bachmeier

TIM-7140:Software Engineering

February 28, 2021

Northcentral University

# Compare Software Engineering Models

Businesses of all shapes are leveraging software-based solutions to reduce costs and become more competitive. This vast breadth of use-cases comes with unique requirements and constraints. For instance, an eCommerce website is updateable countless times per day and can freely evolve. Meanwhile, the Mars Rover necessitates careful change management to avoid becoming a pile of space rubble. Organizations manage these differences by aligning their software engineering models with the business needs.

## Evolution of Release Planning

Traditional software development some form of Waterfall delivery. This sequential process collects requirements, designs systems, validates components, manages operations, and finally, decommission solutions. For projects like the Rover, using this methodology enables them to align lengthy business-cycles with software releases. However, many businesses desire shorter iterations cycles so that they can be more *agile*.

Modern software is agile. While many flavors exist, such as Extreme Programming (XP) and Rapid Application Development (RAD), their approach is reasonably consistent. Under an agile methodology, engineering teams continuously deliver small product enhancements. These short-release windows incorporate more feedback, promoting fierce feature prioritization that delivers the right capabilities for customers.

## Evolution of Architecture

Monolithic systems are challenging to update due to their tightly coupled components and subtle dependencies. While automated testing can detect behavioral regressions, it does not reduce the engineering complexity to make changes. Further complicating matters, development teams must synchronize release schedules to avoid shipping broken code. Architectural patterns such as feature flags partially mitigate these issues.

Instead, many organizations are transitioning away from product-centric to component-centric methodologies. Each component exposes well-defined interfaces and single-purpose design. The component’s implementation can range from an in-process Object-Oriented Programming (OOP) class to remote network service. Regardless of the execution model, the core tenants are to scope responsibility and simplify replacement. These capabilities enable organizations to deliver more consistent change with faster time-to-market.

## Evolution of Operational Model

Designing, implementing, testing, and supporting software are traditionally different specializations. However, this limits the flow of ideas and discourages ownership of problems. Why should developers spend time looking for defects in their code? If it is a real issue, then the quality assurance team will catch it. Similarly, operations teams do not require insights into applications running in their environment. Their responsibility is to maintain the infrastructure, not improve the end user’s experience.

Businesses are sealing these gaps through combined engineering strategies. DevOps is the most well-recognized implementation. It requires development teams to own their code from inception to production. Besides promoting ownership and quality, it also frees support and reliability engineers to focus on value-differentiating tasks. These tasks include more advanced functions across financial (FinOps), security (SecOps), and governance (GovOps).