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 and sizes are leveraging software-based solutions to reduce costs and become more competitive. This vast breadth of use-cases comes with unique constraints that span technical, cultural, and regulatory requirements. For instance, updating an eCommerce website multiple times per day is trivial compared to NASA's challenges with their Mars Rover. Organizations manage these differences by aligning their software engineering models with the business needs.

## Evolution of Release Planning

Traditional software development follows 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 development is agile. Agile comes in many flavors, such as Extreme Programming (XP) and Rapid Application Development (RAD). Each flavor chooses to optimize a specific aspect of the delivery process while sharing many fundamental similarities. The core objective of any agile methodology is enabling teams to deliver small enhancements continuously. These short-release windows incorporate more feedback, promote fierce feature prioritization, and focuses on immediate customer-value.

## Evolution of Architecture

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

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 a 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).

Table 1: Development Strategies

|  |  |  |  |
| --- | --- | --- | --- |
| Process Type | Description | Strengths | Weaknesses |
| Waterfall (CodeAcademy, 2019) | A linear sequence of well-defined stages | Focus on early stages of requirements gathering | Clients might lack understanding of what they want  Assertions disproven later are costly |
| Rapid Development (Idesis, 2017) | Software is clay, not steel.  Rapid prototyping with continuous customer feedback | Deliver sooner, with frequent iterations  Removes unnecessary work and process  Ideal for internal tools | No focus on team structures or communication  Hard to scale across large teams, meeting heavy |
| Extreme Programming (Udacity, 2015) | Lightweight methodology for small to medium teams with vague requirements | Mentality of sufficiency  Frequent releases with numerous checkpoints  Defer building features until needed | An idealistic model that is impractical for many businesses  Challenging to use outside of internal infra teams (Parogi, 2016) |
| Spiral (Easy Learning, 2020) | Loop of planning, risk assessment, development, and evaluation | Focus on what is the next immediate requirement | Large projects require many iterations, and each loop adds costs |
| Reuse (Time2Pretend54, 2017) | Find reusable libraries and open-source project, then build from there | Quickly build solutions by gluing together components  It saves time and money | Less customizable and inflexible  Potential licensing challenges |
| DevOps (Sharma, 2017) | Combined engineer strategy | Reduces gaps and promotes ownership | Requires upskilling across multiple disciplines |
| Service-Oriented Architecture | Component-centric over product-centric design | Quickly add or replace functionality  Well-defined interfaces decouple designs | Abstractions and networking creates performance concerns |
| Dynamic Systems Development Method (ABC, 2021) | Agile methodology focusing on full project lifecycle | Focuses on business needs and continuous delivery | Requires executive sponsorship and org cultural shift |
| Prototype Approach | Software method that defers complexity wherever possible | Quickly validates the technical feasibility of a design | Generally not production quality |
| Lean Software Development | Design method that continuously measures value-add | Aggressively identifies and removes waste | Requires ability to measure a process  Most suitable for repeatable processes |

# References

ABC. (2021, February 28). *What is DSDM*? Retrieved from Agile Business Consortium: https://www.agilebusiness.org/page/whatisdsdm

CodeAcademy. (2019, February 22). *What is Waterfall*? Retrieved from YouTube: https://youtu.be/LxEmGNgqYJA

Easy Learning. (2020, April 4). *Spiral Development Model*. Retrieved from YouTube: https://youtu.be/fTfoB1h2km4

Idesis, S. (2017, October 17). *What is rapid application development*? Retrieved from YouTube: https://youtu.be/JHcxbGwHtsY

Parogi, J. (2016, June 10). *Five reasons why eXtreme programming is not popular*. Retrieved from Medium: https://medium.com/agility-path/5-reasons-why-extreme-programming-isnt-popular-83790418b901

Sharma, S. (2017). *The DevOps adoption playbook.* John Wiley and Sons.

Time2Pretend54. (2017, February 8). *What is reuse-oriented software engineering*? Retrieved from YouTube: https://youtu.be/Dfi4cxCC\_I0

Udacity. (2015, February 23). *Extreme Programming (XP) - Georgia Tech - Software Development Process*. Retrieved from YouTube: https://youtu.be/hbFOwqYIOcU