EPiC Series in Computing

Volume 88, 2022, Pages 1–7

Proceedings of 31st International Conference
on Software Engineering and Data Engineering

Software Development: Past, Present, and Future

Jalal H. Kiswani[1], Sergiu M. Dascalu[2], and Frederick C. Harris, Jr[2]

[1] Smart-API.com,
jalal.kiswani@smart-api.com

[2] University of Nevada, Reno, U.S.A.
dascalus@cse.unr.edu, fred.harris@cse.unr.edu

Abstract

In the field of software development, the processes, technologies, and practices have matured over time to achieve a higher level of delivery and quality. However, the development phase, which is an essential part of the software development life cycle (SDLC), is still consuming a significant cost (time and resources) in both the waterfall and agile approaches. The reason behind that, current technologies and approaches of software development are somehow following the same rules and practices they have for decades, and have not evolved with the proper velocity over the time. In this article, and based on real-life case studies, we will discuss how the utilization of components re-usability (API's and frameworks), metadata-driven development, code generation, and Artificial Intelligence can make the software development more efficient by creating a holistic approach to creating software systems.

1 Introduction

Software development is an essential part of all the modern evolution's in technology, starting from the software installed on main-frames in the beginning era's of computing to solve complex mathematical problems, to the current evolution of autonomous driving, home robots, and the virtual worlds of the meta-verse.

However, while the applications of using software are widely expanded to include almost every area of our lives, the software development process itself, at least most of the time, is being done the same way it has been done for decades.

In the past, the process was based on having a smart-person who was good at math and had excellent analytical skills to translate the needed requirements into computer instructions. However, this didn't scale well, especially with the invention of personal computers in the 1980's, where the need of expanding the programmers base was important; also, this demand has grown more and more with the evolution of internet, in 1990's, and lately, the demand also has increased to be able to deliver the goals of organizations, led by the digital transformation wave.

All the above have demanded a better ways of software development, which started by waterfall, then agile and iterative development, followed by components re-use, and lately,

emphasising on Rapid Application Development(RAD) approaches led by prototyping, low-code no-code tools, platforms, and frameworks [7].

2 Waterfall

Waterfall is following the plan-driven process model. Where it follows an organized processes that is inspired by the engineering discipline of having a clear sequence of phases that includes: a) specifications, b) design, c) development, d) validation, and finally, e) evolution.

In practice, the requirements phase is performed by a system analyst (who is also commonly called business analyst). The design phase is done by different architectural roles, including a data architect, User Interface/User Experience engineer (UI/UX), and an application architect. The development phase it self is performed by front-end developers, back-end developers, database developers, and some times could be handled by a one-man-show – full-stack developer. The validation, which is also commonly called the Testing phase, is curried-out by the Quality Assurance (QA) and/or Quality Control engineers (QC), which also based on the nature of testing, which could be manual or automated testing. This is illustrated in Figure 1

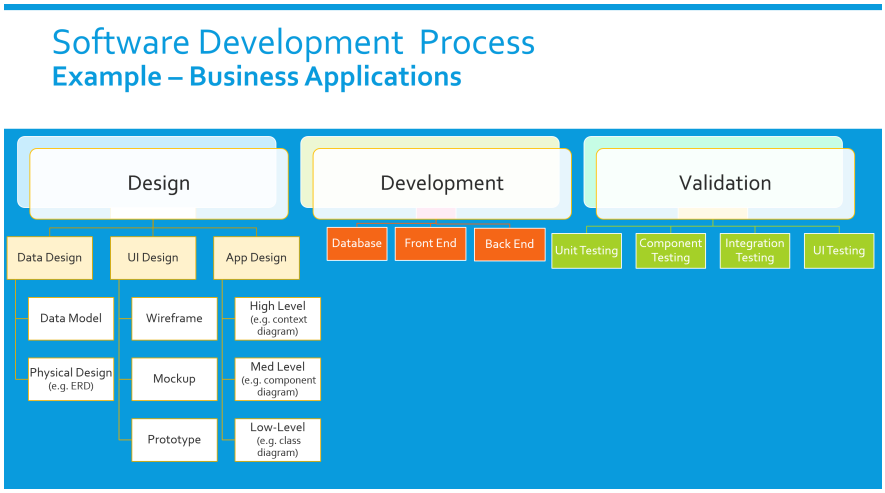

Figure 1: The Software Development Process in the Waterfall Model.

While the waterfall approach had a successes in some fields and domains (e.g., health, military, and financial), in addition of looks elegant, controllable, and well-structured, it didn't work will during the internet-boom in the late 1990's, since in Silicon-Valley, time-to-market was more important than software quality, and the nature of having frequent changes to requirements in some-domains led to some projects failures [5].

Some retries of following an iterative approach that delivers values faster such as Rational Unified Process (RUP) was introduced. RUP was developed by Rational Software by 1995, and acquired later by IBM in 2003. While RUP was created with the ability to adopt changes with minimum impact of software quality and delivery, the tools, documentation needed, modeling required, and the learning curve, didn't allow to be adopted on large-scale [6].

3 Agile

Following a disciplined approach of software development such as waterfall, and a formal-iterative approaches such as RUP, worked well for a while and in some-domains, however, and as mentioned in the previous section, it didn't scale well for domains that requires faster delivery and high frequency of changes to the requirements [1].

Silicon Valley, as it has been always been, decided to go with a different approach, eXtreme Programming (XP), where they thought that spending a lot of time on requirements specifications and designing software with the maximum quality before writing a single line of code is risky and could take companies out of market, so they decided to go directly to development phase as fast as they could. While it looks risky, in startup companies its not, where the requirements needed to be implemented is short, and could (and in the agile culture it should) be delivered relatively in a short time frame, in practice, from one to two weeks. XP worked well in small teams that had a the need of delivering values faster as Minimum Viable Product(MVP), that is mainly implemented by internal team for internal products [2].

Agile in general and XP in particular, and based on the need of improving quality and reducing risks, introduced some new concepts to software development, such as pair-programming, test-driven development, and continuous integration.

Enterprise and domains that has formal management styles tried to implement agile in different contexts that doesn't match the culture of software development in Internet business, which led to significant issues, such as lack of visibility, and inability to measure project performance, in addition to some management issues such as budgeting and resources allocation. This has led to the a new approach, Scrum, where a Scrum Master, a replacement of Project Manager in waterfall approach, is managing each iteration (in agile approaches could be called story or sprint) and ensuring the agile practices are well-implemented, in addition of having proper visibility on the development process, performance monitoring, mitigating the concerns presented by XP approach.

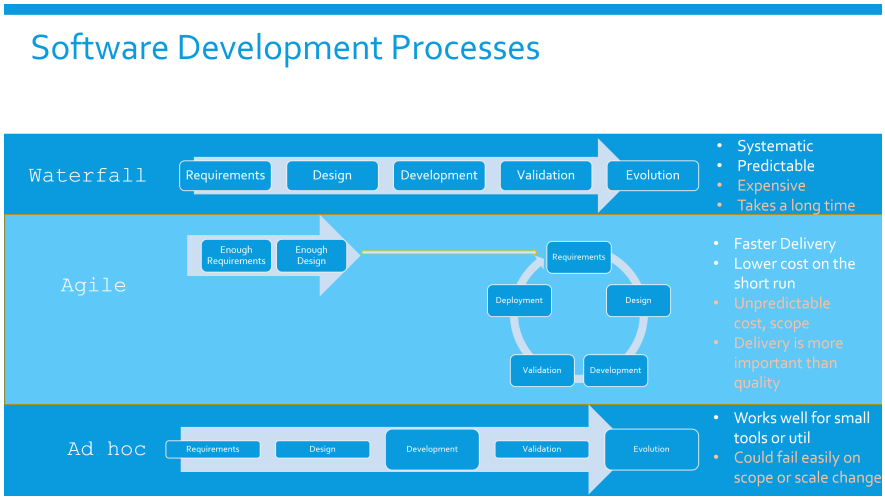

Figure 2: Software Development Process

4 Software and Components Re-usability

Software re-use has dramatically improved the software delivery and time to market, where software development could re-use components from Application Programming Interfaces (API's), using application frameworks such as Microsoft .NET, Java Enterprise Edition (Java EE). or following a Software Product Line Approach (SPL) such as building an Operating System (OS) from the open source Linux kernel created by Linus Torvalds.

The wide traction to this approach was increased due to the availability of open source repositories such as GitHub, SourceForge, and GitLab which have enabled organizations and individuals to share their internal components such as NetFlex frameworks which led the current development practices of Cloud Native Applications.

While software re-use have helped lot of organizations implementing software applications faster than building it from scratch, developers (programmers) most of the time still needs to code basic stuff that consumes a lot of their development time such as User Interface(UI), themes, validations, data access, and binding between the data-access and the UI. In addition to implementing quality attributes, such as security, reliability, integrity, and maintainability, which all led to make the developers focus on stuff that is not directly relevant to the tasks in-hand expected by the management or customers.

5 Rapid Application Development

The dream of having and elegant, easy, faster way, and tools of developing software has been there since along time, however, it still not convincing enough for the management and technical people. Where managers prefer tools that doesn't require special expertise, well-supported (mainly by a wide-community) for along time, and is accepted by the technical team. Same as developers, who looks for tools that keep them productive to satisfy their employers, however, with carefully maintaining their position and their market-value of having an edge over non-technical people [3, 4].

Using visual tools such as model-driven development tools is available in multiple flavours, where a lot of Unified Modeling Language tools (UML) tools like Visual Paradigms, could help generating the code from UML models. Other tools such as Microsoft PowerApps, and Oracle Apex are used to enable people to build software directly from the a design console, which somehow follows the prototyping approach, which also currently is being called No-Code approach.

However, the black-box nature of applications generated using No-Code approach such as the previous examples and platforms such Mendix and Appian, is considered risky for some organizations due to the hidden details of code that executed at run-time, in addition to the risks of vendor-lock, data-migration, and integration with other systems.

Generating the code for that is time-consuming, repetitive, and doesn't require special expertise, could be the way-to-go, where having tools or platforms that enable developers to git rid of the all the time-consuming tasks, could enable developers to focus on whats important, satisfy their managements, and deliver values to their customers faster, without having any vendor-lock or black-box apps constrains. Such concept, which is known as Low-Code, could be the future of software development, and makes the software development takes another complete efficient direction that could be implemented on various domains.

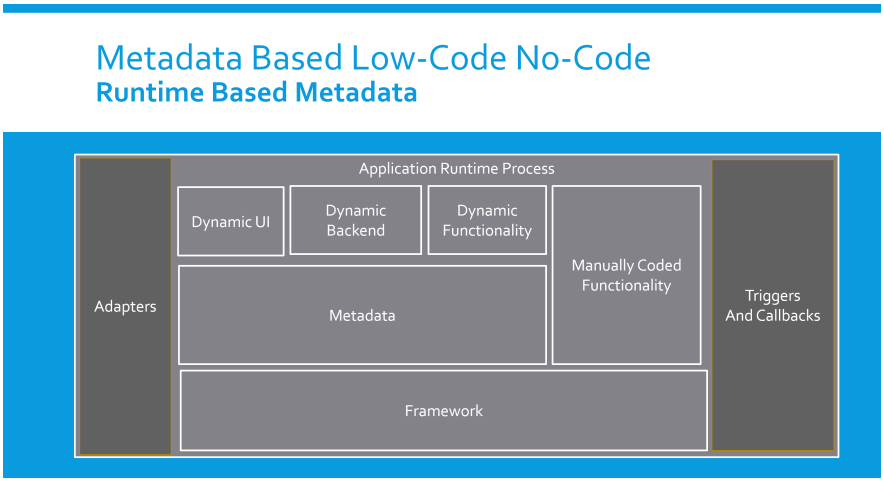

Figure 3: Runtime Based Metadata

6 Metadata Driven Low-Code

Some platforms such as Smart-API.com provide tools and applications that includes low-code no-code tools such as CodeGen, AppGen, and App-Studio.

Smart-API tools and applications utilize the concepts of metadata data, where applications of specific domains, will have (most likely) common project structure, common files, common files sections, and a common development approach. By designing the metadata model, creating a code transformers, building user friendly UI, that enable users to input the metadata for for specific applications, could enable software vendors building low-code platforms that makes software development more efficient (Figure 4, 5).

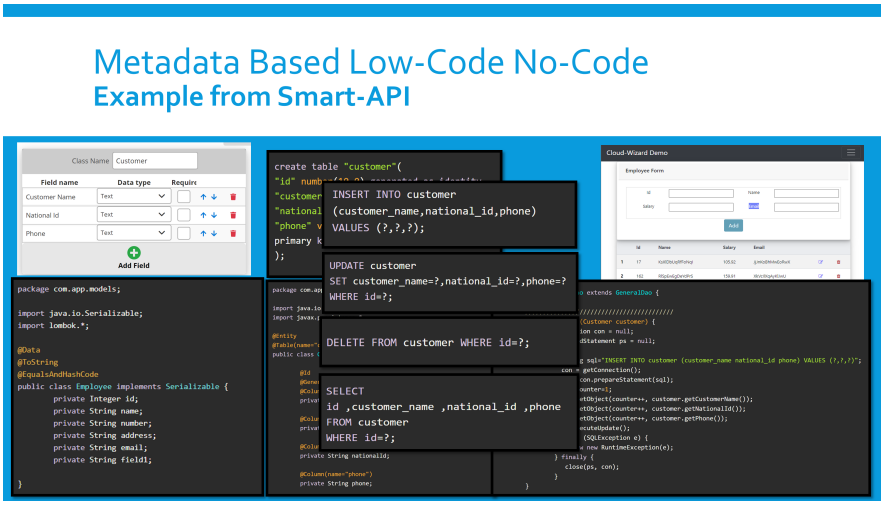

Figure 4: Smart API Approach

However, implementing the above process for each domain, could be a time consuming and

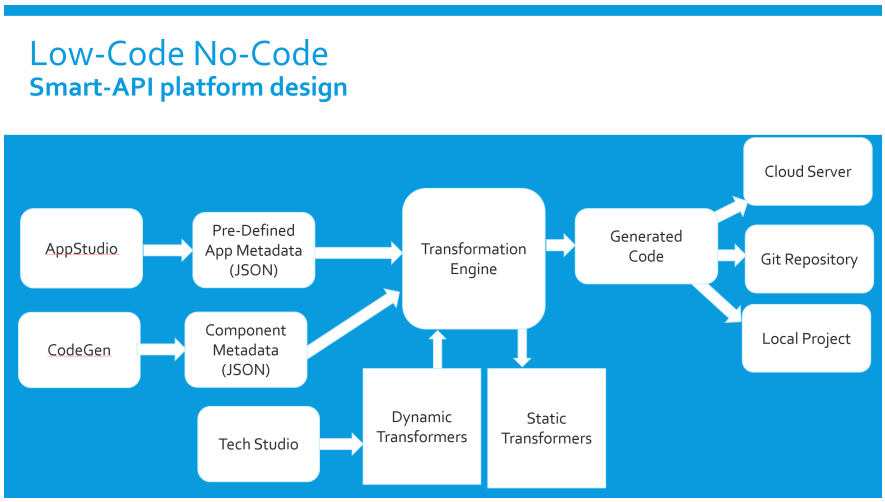

Figure 5: Smart API Platform Design

requires a special expertise, where designing the metadata models and creating the transformers manually for each domain could be not practical. Utilizing an Artificial Intelligence (AI) approach for that could change the future of software development (Figure 6).

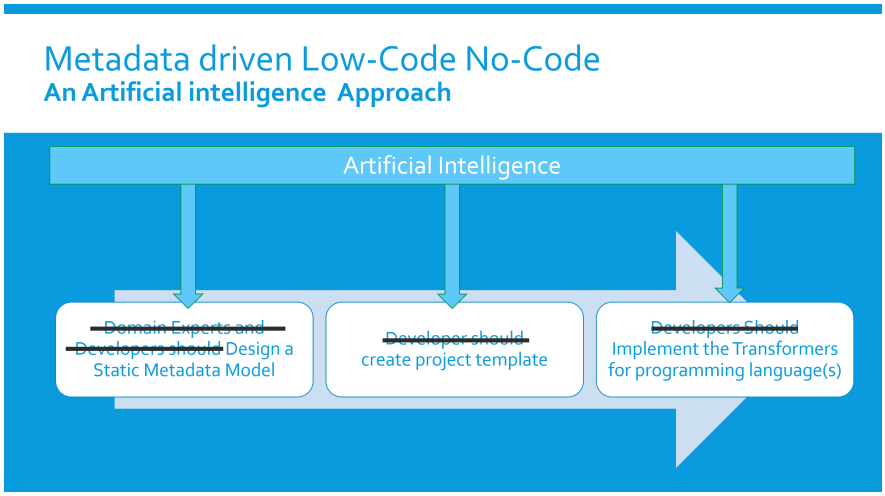

Figure 6: AI Approaches could change the Model.

7 Conclusion

While software is being a key part of every aspect of our lives, software development it self is still time-consuming, expensive, and limited in features to be delivered.

Enabling a Metadata Driven Low-Code No-Code approach could enable a more efficient delivery of software development, where building applications for the same domains doesn't

need to be repetitive.

However, designing the metadata models for each domain manually is not practical, so utilizing an AI approach that analyzes projects from same domain, extract common project structure, common files, common sections, and any other repetitive metadata data could enable a faster adoption of Low-Code No-Code Metadata approach.

References

[1] Pekka Abrahamsson, Juhani Warsta, Mikko T Siponen, and Jussi Ronkainen. New directions on agile methods: a comparative analysis. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 244–254. Ieee, 2003.

[2] Homa Bahrami. The emerging flexible organization: Perspectives from silicon valley. *California management review*, 34(4):33–52, 1992.

[3] Paul Beynon-Davies, Chris Carne, Hugh Mackay, and Douglas Tudhope. Rapid application development (rad): an empirical review. *European Journal of Information Systems*, 8(3):211–223, 1999.

[4] Robin Lichtenthäler, Sebastian Böhm, Johannes Manner, and Stefan Winzinger. A use case-based investigation of low-code development platforms. In *ZEUS*, pages 76–83, 2022.

[5] Kai Petersen, Claes Wohlin, and Dejan Baca. The waterfall model in large-scale development. In *International Conference on Product-Focused Software Process Improvement*, pages 386–400. Springer, 2009.

[6] Sara Shafiee, Yves Wautelet, Lars Hvam, Enrico Sandrin, and Cipriano Forza. Scrum versus rational unified process in facing the main challenges of product configuration systems development. *Journal of Systems and Software*, 170:110732, 2020.

[7] Ian Sommerville. Software engineering. Pearson, 2016.