e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

TEMPLATIZATION: AN APPROACH TO IMPROVE SCALABILITY AND MAINTAINABILITY OF CODEBASES

Vishnu Vardhan Reddy Chilukoori[*1]

[*1]Lead Data Engineer, Amazon.Com Services LLC, India.

DOI : https://www.doi.org/10.56726/IRJMETS59798

ABSTRACT

Templatization is a software engineering practice that involves using templates, patterns, and abstractions to modularize and standardize code components. This approach can significantly improve the scalability and maintainability of codebases by reducing redundancy, streamlining development processes, and promoting code reuse. In the context of data engineering, templatization can be applied to various aspects, such as data partitioning, indexing strategies, and query optimization, to enhance performance and ensure consistency across different projects and environments. By embracing templatization, software development teams can cultivate scalable, maintainable codebases that are adaptable to future advancements in technology and business requirements.

Keywords: Apache Spark, Containerization, Data Engineering, Data Warehousing, Docker, Performance Optimization, Query Optimization Algorithms, Regulatory Compliance, Reproducible Workflows, Secure Coding Practices, Software Development, Version Control.

I. INTRODUCTION

In the dynamic landscape of software development, creating a scalable and maintainable codebase is crucial for long-term success. As software projects grow in complexity and the number of contributors increases, the need for efficient development practices becomes paramount. Templatization, a software engineering approach that emphasizes the use of templates, patterns, and abstractions to modularize and standardize code components, has emerged as a promising solution to address these challenges.

Chilukoori et al. [10] highlight the use of Docker to create standardized, reproducible development environments for Apache Spark, a popular framework in data engineering. This approach aligns with the principles of templatization, as Docker containers encapsulate the entire Spark environment, including dependencies and configurations, into a reusable template. This ensures consistency across different development stages and deployment environments, promoting reproducibility and reducing environment-related issues.

Furthermore, templatization fosters scalability by enabling rapid prototyping and deployment of new features. Developers can leverage pre-defined templates to build upon existing code efficiently, minimizing the need for repetitive coding tasks. This abstraction accelerates development cycles and empowers teams to respond swiftly to evolving requirements and market demands.

II. LITERATURE REVIEW

Software engineering is a relatively young discipline compared to more traditional fields of engineering, such as civil or mechanical engineering. While the latter benefit from well-established theoretical foundations and principles rooted in mathematics and physics, software engineering is still working to establish a set of widely accepted fundamental principles (Al-Sarayreh et al.) [1]. In this context, Al-Sarayreh et al. [1] conducted a systematic mapping study to identify and analyze research on software engineering principles (SEPs).Their study revealed that while numerous SEPs have been proposed, there is a lack of consensus and a need for further research to refine and consolidate these principles. The authors also highlighted the importance of developing a standardized process for identifying and applying SEPs throughout the software development lifecycle.

The importance of software engineering principles is also highlighted in Wohlin et al. [2], where the authors propose a general theory of software engineering that emphasizes the balancing of three intellectual capitals: human, social, and organizational. The authors argue that software development is a knowledge-intensive

activity that relies heavily on people, their interactions, and the organizational structures that support them. This theory underscores the need for a holistic approach to software engineering, considering not only technical aspects but also the human and social dimensions. In the realm of software measurement, Murphy and Cormican [3] present a case study of a global software company that implemented a radical measurement program but failed to achieve the desired productivity improvements. The study attributes this failure to several factors, including a narrow focus on projects rather than organizational goals, a lack of staff training in measurement activities, and a project centric culture that prioritized on-time delivery over quality and productivity measurement. The findings of this study emphasize the importance of aligning measurement programs with organizational goals, providing adequate training, and fostering a culture that values measurement and continuous improvement.

The challenges associated with software measurement are further explored in the work of Gobert et al. [4], which focuses on testing database manipulation code. The authors conducted a study on open-source projects and found that database manipulation code was often poorly tested. They identified several challenges faced by developers in this context, including database management, mocking, parallelization, and framework/tool usage. The study highlights the need for better tools and practices to support the testing of database manipulation code, which is crucial for ensuring the reliability and quality of software systems. In the context of emerging technologies, van Vulpen et al. [5] investigate the governance of decentralized autonomous organizations (DAOs) that produce open-source software (OSS). The authors propose a governance framework for such DAOs, emphasizing the need to balance the particularities of OSS production with the challenges of decentralized governance. The study highlights the importance of considering various governance mechanisms, including leadership and role structure, decision-making processes, legal foundations, project chartering, incentives, community management, and software development processes. The findings of this study are relevant to the broader discussion of software engineering principles, as they demonstrate the need for adaptable and effective governance structures in the context of decentralized software development.

In the rapidly evolving landscape of software development, the adoption of DevOps practices has gained significant attention. Grande et al. [6] conducted a systematic mapping study to investigate the benefits and challenges of adopting DevOps in distributed and global software engineering settings. The study found that while adopting DevOps in such settings can bring several advantages, such as shorter release cycles, improved reliability, and faster feedback, it also presents challenges related to skill set requirements, communication management, and resistance to change. The findings of this study underscore the importance of carefully considering the specific context and challenges of distributed development when adopting DevOps practices.

The integration of machine learning (ML) components into software systems presents unique challenges for software engineering. Naveed et al. [7] conducted a systematic literature review on model-driven engineering (MDE) for ML components, exploring the motivations, solutions, evaluation techniques, and limitations of existing studies. The study found that while MDE can offer benefits such as reduced complexity and improved quality, there are also challenges related to data preprocessing, scalability, and responsible ML development. The findings of this study highlight the need for further research in this area, particularly in addressing the challenges and ensuring the responsible and ethical use of ML in software systems.

In the context of open science and empirical software engineering, Kessel and Atkinson [8] propose new data structures and a dedicated platform to support the reproducibility and reusability of test-driven software experiments. The authors argue that existing approaches for representing and analyzing experimental data are often ad hoc and lack transparency, hindering the ability of other researchers to repeat, replicate, or reproduce the experiments. The proposed data structures and platform aim to address these challenges by providing a standardized and accessible way to represent, store, and analyze experimental data, thus promoting open science practices in software engineering research.

Finally, Serban et al. [9] investigate the adoption and effects of software engineering practices for machine learning (ML). The authors conducted a mixed-methods study involving a systematic literature review, a survey of practitioners, and validation interviews. The study identified a comprehensive catalog of engineering practices for ML and found that while larger and more experienced teams tend to adopt more practices, trustworthiness practices are often neglected. The study also found that the effects of adopting these practices,

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

such as team agility and accountability, can be predicted from groups of practices. The findings of this study provide valuable insights into the current state of ML engineering practices and their impact on the development of software with ML components.

Chilukoori et al. [10] explore the use of Apache Spark and Docker for data engineering development. They highlight the challenges of setting up local development environments for Spark and how Docker can address these challenges by providing containerized environments. The authors emphasize the benefits of using Docker, such as reproducibility, dependency management, and isolation, which align with data engineering principles. The study provides insights into how Docker can streamline local development, ensure consistent Spark environments, and facilitate smoother transitions to production environments.

In the realm of cloud data warehouses, Chilukoori et al. [11] present a framework for optimizing query performance. The framework includes performance monitoring, bottleneck identification, and optimization implementation. The authors emphasize the importance of data partitioning, indexing strategies, and query optimization algorithms in improving query performance. A case study is presented to demonstrate the effectiveness of the framework in a real-world scenario, highlighting the potential for significant performance improvements and cost savings.

Chilukoori et al. [12] discuss the role of data warehousing in the financial services industry. The authors highlight the challenges faced by the industry, such as data integration, data quality, and scalability, but also emphasize the opportunities presented by data warehousing, such as enhanced risk management, improved customer service, and regulatory compliance. The regulatory landscape, including data privacy, security, and reporting requirements, is also examined. The article provides a comprehensive overview of the current state of data warehousing in the financial services industry and its potential impact on the sector.

Gangarapu et al. [13] delve into the security challenges of embedded firmware engineering in the age of edge computing. The authors explore the unique vulnerabilities of edge computing environments, including resource constraints, physical accessibility, and distributed architecture. They emphasize the crucial role of embedded firmware engineers in mitigating these risks through secure coding practices, hardware-based security features, and secure boot processes. The article also discusses the evolving threat landscape targeting edge devices and proposes future research directions for embedded firmware security.

Chilukoori et al. [14] [15] investigate the use of Scala for accelerating prototype-to-production application development in data engineering. The authors highlight Scala's concise syntax, strong type system, and compatibility with popular frameworks like Apache Spark. Through case studies, they demonstrate how Scala can significantly reduce development time, improve code quality, and enhance scalability in data engineering projects. The findings underscore the potential of Scala as a valuable tool for rapid prototyping and efficient development in the data engineering domain.

III. IMPLEMENTATION

Templatization is implemented in quite a few ways. The two most common approaches in data engineering are 1. f-strings in Python environments and 2. Jinja templates. We will review these two means of reducing code redundancy.

F-strings

F-strings stands for the formatted strings in Python, released in version 3.6. It simplifies the variable substitution process in a string compared to earlier options, such as string concatenation. Let us look at a straightforward example here of how we can use it with an SQL query to retrieve the inventory of a product.

In the examples seen in Fig. 1, 2, we have produced a dataset with store_id, product_id, and quantity columns from the product_inventory table. Now, if we would like to add more columns to this dataset, all we need to do is add those column names to the columns variable; it takes care of the rest. We can add more complexity to the template, such as a column filter.

We built a reusable template for selecting data from any table. We can further extend this template to calculate a given product's inventory. We have removed the need to write the SQL query for every product available in the store, and it is easy to update the definitions of these metrics, too.

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

```
columns = ["store_id", "product_id", "quantity"]
column_str = ",".join(columns)
table = "product_inventory"

simple_query_template = f"""
select
    {column_str}
from {table}
"""
```

Figure 1: An example of F-Strings in Python.

Jinja Templates

Jinja2 is a powerful templating engine for Python, widely used in web development and beyond. It allows you to create dynamic content by combining template files with data from your Python code. This separation makes your code cleaner, easier to maintain, and prevents accidental mixing of logic and presentation.

Instead of hardcoding values directly into your code, you use placeholders, for e.g., {{ variable }} that Jinja2 replaces with actual data during rendering. We can use conditional statements {% if ... %}, {% else %} and loops {% for ... %} to create more complex templates that adapt to different situations or iterate over lists of items. Jinja2 also provides built-in filters to format dates ({{ date | date:'F j, Y' }}), manipulate strings ({{ name | capitalize }}), and perform other useful transformations. This helps in organizing our templates efficiently by creating base layouts and extending them with specific content blocks in child templates. We can see the example code from Fig 1., converted into Jinja template in Fig. 3. As we see in the Python snippet, we can perform more advanced formatting using if/else, for loops in Jinja, which provides more flexibility than the F-strings.

```
columns = ["store_id", "product_id", "quantity"]
column_str = ",".join(columns)
table = "product_inventory"
limit_clause = ""
filters = {"product_id": "abcd_123", "store_id": "store_abc1"}
where_clause = "WHERE " + " AND ".join([f"{col} = '{val}'" for col, val in
filters.items()])

simple_query_template = f"""
select
    {column_str}
from {table}
{where_clause}
{limit_clause}
"""
print(simple_query_template)

Output:
--------
select
    store_id,product_id,quantity
from product_inventory
WHERE product_id = 'abcd_123' AND store_id = 'store_abc1'
```

Figure 2: Executing our Python example to produce the SQL statement.

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:07/July-2024 Impact Factor- 7.868 www.irjmets.com

```
from jinja2 import Template

# 1. Define your Jinja template (template.html)
template_string = """
select
    {{column_str}}
from {{table}}
{{where_clause}}
{{limit_clause}}
"""

# 2. Create a Jinja template object
template = Template(template_string)

# 3. Prepare your data
data = {
    "column_str": """store_id, product_id, quantity""",
    "table": "product_inventory",
    "where_clause": ["product_id = 'abcd_123'", "store_id='store_abc1'"]
}

# 4. Render the template with data
output = template.render(data)

print(output)

Output:
--------
select
    store_id, product_id, quantity
from product_inventory
WHERE product_id='abcd_123'
    AND store_id='store_abc1'
```

Figure 3: A Jinja template based SQL template.

IV. CONCLUSION

We have looked into two means of utilizing templates effectively to avoid rewriting SQL queries for similar features using Python and Jinja. Templating provides a significant boost in the productivity of engineers while raising the maintainability of the entire codebase.

V. REFERENCES

[1] K. T. Al-Sarayreh, K. Meridji, and A. Abran, "Software engineering principles: A systematic mapping study and a quantitative literature review," Engineering Science and Technology, an International Journal, vol. 24, no. 3, pp. 768–781, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S221509862034252X

[2] C. Wohlin, D. Šmite, and N. B. Moe, "A general theory of software engineering: Balancing human, social and organizational capitals," Journal of Systems and Software, vol. 109, pp. 229–242, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121215001740

[3] T. Murphy and K. Cormican, "An analysis of non-observance of best practice in a software measurement program," Procedia Technology, vol. 5, pp. 50–58, 2012, 4th Conference of ENTERprise Information Systems – aligning technology, organizations and people (CENTERIS 2012). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212017312004379

[4] M. Gobert, C. Nagy, H. Rocha, S. Demeyer, and A. Cleve, "Best practices of testing database manipulation code," Information Systems, vol. 111, p. 102105, 2023. [Online]. Available: https://www.sciencedirect.com / science /article/pii/S0306437922000886

[5] P. van Vulpen, J. Siu, and S. Jansen, "Governance of decentralized autonomous organizations that produce open source software," Blockchain: Research and Applications, vol. 5, no. 1, p. 100166, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2096720923000416

[6] R. Grande, A. Vizcaíno, and F. O. García, "Is it worth adopting devops practices in global software engineering? possible challenges and benefits," Computer Standards Interfaces, vol. 87, p. 103767, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S092054892300048X

[7] H. Naveed, C. Arora, H. Khalajzadeh, J. Grundy, and O. Haggag, "Model driven engineering for machine learning components: A systematic literature review," Information and Software Technology, vol. 169, p. 107423, 2024. [Online]. Available: https://www.sciencedirect.com /science/ article/pii/ S0950584924000284

[8] M. Kessel and C. Atkinson, "Promoting open science in test-driven software experiments," Journal of Systems and Software, vol. 212, p. 111971, 2024. [Online]. Available: https://www.sciencedirect.com/ science/ article/pii/S0164121224000141

[9] A. Serban, K. van der Blom, H. Hoos, and J. Visser, "Software engineering practices for machine learning – adoption, effects, and team assessment," Journal of Systems and Software, vol. 209, p. 111907, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121223003023

[10] S. S. R. Chilukoori, S. Gangarapu, and C. K. Kadiyala, "Development with apache spark and docker: A data engineering perspective," International Journal of Innovative Research in Science, Engineering and Technology, vol. 13, no. 6, pp. 10 390–10 397, 6 2024. [Online]. Available: https://www.ijirset.com /upload /2024/june/4_Development.pdf

[11] S. S. R. Chilukoori, S. Gangarapu, and C. K. Kadiyala, "Optimizing query performance in cloud data warehouses: A framework for identifying and addressing performance bottlenecks," International Journal of Advanced Research in Engineering and Technology, vol. 15, no. 3, pp. 288–297, May - June 2024. [Online]. Available: https://iaeme.com/Home/article_id/IJARET_15_03_025

[12] S. S. R. Chilukoori, S. Gangarapu, and C. K. Kadiyala, "Data warehousing in the financial services industry: Challenges, opportunities, and regulatory considerations," Journal of Advanced Research Engineering and Technology, vol. 3, no. 1, pp. 34–44, 6 2024. [Online]. Available: https://iaeme.com/ Home/ article_id /JARET_03_01_004

[13] S. Gangarapu, S. S. R. Chilukoori, and C. K. Kadiyala, "Securing the edge: Embedded firmware engineering in the age of edge computing," International Journal of Advanced Research in Engineering and Technology, vol. 15, no. 3, pp. 333–343, May - June 2024. [Online]. Available: https://iaeme.com/ Home/ article_id/IJARET_15_03_029

[14] S. S. R. Chilukoori, S. Gangarapu, and C. K. Kadiyala, "Accelerating prototype to production application development in data engineering with scala," International Research Journal of Engineering and Technology, vol. 11, no. 6, pp. 496–502, 6 2024. [Online]. Available: https://www.irjet.net/archives /V11/i6/IRJET-V11I677.pdf

[15] P. R. Chintala, V. V. R. Chilukoori, and S. S. R. Chilukoori, "A review of pair programming (gen-ai) tools in data engineering," International Research Journal of Modernization in Engineering Technology and Science, vol. 6, no. 4, pp. 7529–7530, 4 2024. [Online]. Available: https://www.irjmets.com/ uploadedfiles/paper//issue_4_april_2024/53795/final/fin_irjmets1713866966.pdf