An Empirical Study of Maven Archetype

Xinlei Ma† Yan Liu† *
†School of Software Engineering, Tongji University, Shanghai, China
* Corresponding Author
{1831592, yanliu.sse}@tongji.edu.cn

***Abstract*—Archetype is a Maven project templating toolkit. An archetype is defined as an "original pattern" of the representative Maven project. Using archetypes enables Maven developers to quickly work in a way consistent with the best practices which demonstrate many Maven features and usage patterns of Maven components. Nowadays, more and more developers utilize archetypes to standardize development within their organizations. Despite the ever-growing use, there are still limited experimental evidence and guidance on how to leverage the power of archetype. Meanwhile, because of the enormous scale and spotty quality of projects, it is incredibly challenging to perform analysis on the whole Maven central repository. As the simple "artifacts" of the Maven best practices in many diverse domains, Maven archetypes are ideal for studying the "Maven Way" of configuration and the usage pattern of Maven libraries. Therefore, we perform the first empirical study on 2,326 archetypes retrieved from the Maven central repository to discover the archetype characteristics. Our results identify the configuration schema patterns, structural patterns, the uses of dependencies/plugins in archetypes, and summarize some evolution characteristics of archetypes as well. The primary archetype characteristics capture the potential research value of archetypes. The guidance on how to configure the archetype and utilize Maven libraries can be leveraged to maintenance, automatic completion both for archetypes, and Maven projects.**

***Index Terms*—Maven; Archetype; Configuration; Software Analytics**

I. Introduction

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting, and documentation from a central piece of information [1], [2].

Archetype is a Maven project templating toolkit. It helps archetype authors create Maven project templates for Maven developers and provides developers with the means to generate Maven projects from those project templates. An archetype is defined as an "original pattern" or "model", which helps Maven developers get started as quickly as possible. More encouragingly, it will introduce new users to the best practices employed by Maven [3]. Nowadays, more and more developers adopt the archetypes to standardize development within their organizations. Despite the ever-growing use, there are still limited experimental evidence and guidance on how to configure archetypes.

Maven central repository is one of the most popular and widely used repositories of Maven projects, which contains more than 2.8M Java Maven projects [4]. However, those who want to analyze the whole Maven central repository face challenges on an enormous scale and spotty quality of projects [4]. As the simple "artifacts" of Maven best practices, archetypes also present the "Maven Way" [5] of configuration and some usage patterns of Maven libraries. Therefore, we perform the empirical study on Maven archetypes in the Maven central repository instead of all Maven projects. To our knowledge, we are the first to perform the empirical study on Maven archetypes [6]. Previous archetype-related researches [7] paid more attention to its practical usage but failed to leverage its hidden knowledge.

An archetype consists of the archetype metadata which describes the contents of archetype and Maven project templates which can generate a working project [3]. These project templates include *pom.xml* files, Java files, and other resource files. In concept, the archetype is a code "skeleton" or a very simple "artifact" [3]. Maven archetypes and projects are configured by a POM, which is stored in a *pom.xml* file. Since our research object is studying archetype configuration and usage patterns of Maven libraries, we focus on POMs in *pom.xml* files.

This work start with four research questions on the configuration levels, design appropriate analysis process to explore archetype patterns, uses of dependency/plugin, evolution, and obtain some interesting conclusions finally. The main contributions of our work are as follows:

- Gaining novel insight into configuration-level analysis across representative Maven projects.
- Presenting a first complete process on archetype analysis with appropriate static and quantitative methodologies.
- Releasing a pre-processed datasets in https://zenodo.org/record/3702349#.Xmd21JMzZQI, including all 2,326 archetypes which can be used for further research.

The remainder of this paper is organized as follows: Section 2 presents the terminologies related to this paper. Section 3 describes the methodology applied to conduct the analysis. Section 4 presents the experimental results of the four proposed questions. Section 5 presents the conclusions of the study. Due to space limitations, all graphs and Table II and Table III are available at https://drive.google.com/file/d/1EhqU1GY1KBS5xipo4DkUtCiurMmE2pdS/view.

DOI reference number: 10.18293/SEKE2020-021

II. Terminology

Maven artifacts such archetypes, dependencies, and plugins are identified by the 3-tuple "GroupId:ArtifactId:Version", where GroupId is the organization of this project, ArtifactId is the name of this project, and Version is the version for this project respectively. We provide some terminologies of POM here to make readers understand better.

POM Tree. A POM is stored in a well-defined eXtensible Markup Language (XML) file consisting of element tags, which can be modelled as a rooted tree with element nodes: the POM XML tree [8].

POM Relationship. POM relationships generally include dependencies, inheritance, and aggregation [2]. POM Inheritance means that a POM extends its parent POM directly and can inherit and override its parent. A Maven project can be represented as a sub-module with its own POM. Through POM, a multi-module or aggregator project can group many sub-module projects [2].

Then, for better analysis, we give detailed definitions of essential concepts in this paper.

POM XML Element Sequence. In a POM Tree, each element node has a unique path from the root ("project" element) to itself. Given a specific element Ele, we can represent it as a tuple $<Ele, S_{ele}>$ where Ele is the tail element of the sequence, and S_{ele} is the prefix of this element in the POM tree.

POM Inheritance Depth. It defined as the depth of the given POM in the POM inheritance chain. In other words, it is the number of POMs preceding a given POM in the POM inheritance chain.

Evolution Type of Dependency and Plugin. We define the following kinds of dependency changes: *dep_add*, *dep_del*, *dep_add_del*, *dep_v*, *dep_scope*, and *dep_exclutions*. The evolution type of plugin is similar. We use *dep*, *plu*, *v*, and *del* to denote dependency, plugin, version, and deletion.

III. Methodology

In this empirical study, firstly, we give some definitions for some concepts related to our study. Then we propose four archetype-based research questions, collect the dataset, and design corresponding experiments. Based on the findings of each experiment, we provide conclusions for each research question or explanations of experimental results. The overall workflow is presented in Fig. 1.

A. Research Questions

We propose four questions from the aspects of patterns, uses of dependency/plugin, and evolution. The questions are:

- RQ0: What are typical configuration schema patterns in archetype POMs?
- RQ1: How are structural patterns used in archetypes?
- RQ2: What is the usage pattern of dependencies and plugins in archetypes?
- RQ3: How archetypes evolve during the lifecycle?

B. Dataset Preparation

1) Collecting raw dataset: We collect all the second latest archetypes in JAR format from the Maven central repository, which include 6,432 POM files and 11,050 Java code file. Moreover, for archetype evolution study, our dataset contains 10,184 archetypes releases for a total of 2,326 archetypes, which gives an average of 4 releases per archetype.

2) Processing raw dataset: In Maven projects or archetypes, the *pom.xml* file specifies the project structure, settings for different build steps, and libraries on which the project depends. We decompress the JAR archetypes, retrieve *pom.xml* files from decompressed archetypes, and parse them to extract useful data (such as adopted dependency and pom relationships). In this work, we mainly focus on the root *pom.xml* files in archetypes except for the POM relationships (which are extracted from the *pom.xml* file for archetype generation process).

C. Technologies

Apriori [9] frequent itemset mining algorithm is used to investigate what type of dependencies or plugins always work along, with Confidence level (C), Support (S), and Lift (L) as metrics.

IV. Observations and Findings

A. RQ0: What are typical configuration schema patterns in archetype POMs?

As a configuration file, a *pom.xml* file should base on the POM schema. *Configuration Schema Pattern*s of POM are useful for POM automatic completion. We define *Configuration Schema Pattern* as a combination of frequent element sequences to implement a configuration concern in POM. To answer this question, we propose three subquestions:

- RQ0.1 Which element tags are frequently used in root *pom.xml* files?
- RQ0.2 Which element sequences are frequently used in root *pom.xml* files?
- RQ0.3 Which configuration schema patterns are frequently used in root *pom.xml* files?

Figure 3 and Figure 4 illustrate the top frequently used element tags and element sequences in the root *pom.xml* files separately. Referring to these element sequences with corresponding frequency, we summarize several typical schema patterns by combining relational element sequences into an element tree. Figure 2 displays some typical schema patterns, and the root element of these patterns are *build* (the top-level build element under project), *dependency*, *plugin*, and *profile*. Regarding each element in the element tree, we manually divide their direct suffix element tags into two sets according to frequency. In Figure 2, the dotted lines with boxes mean the set of highly frequently used element tags, and the original lines with boxes mean the set of less frequently used element tags. The orange box means the non-leaf node, and the green box means the leaf node.

RQ0 We analyze the XML schema patterns used in POM files to detect the recurring element tags and sequences. For POM automatic completion, several configurations patterns are detected and concerns principally the tags *build*, *dependency*, *plugin*, and *profile*.

B. RQ1: How are structural patterns used in archetypes?

Our investigation of structural patterns in archetypes focuses on aggregation and inheritance. POM aggregation is used to manage a complex system which involves hundreds of interrelated sub-modules. POM inheritance can effectively reduce the repetition of configuration code and make it easier to reuse configuration code. Some typical "Maven Ways", such as centralized management of dependencies, predefining of public components, can be used through POM aggregation and POM inheritance. In this part, we propose two subquestions:

- RQ1.1 Which proportion of archetypes adopts POM inheritance or POM aggregation in our dataset? Is the POM Inherited Depth usually not more than one?
- RQ1.2 Are archetypes which adopt POM inheritance and POM aggregation simultaneously in proportion in our dataset?

TABLE I: Uses of POM inheritance and aggregation in primary dataset and extended dataset

Item	Num	Avg	Mid	Max	Scope
inheritance	1,817	/	/	/	primary
aggregation	1	/	/	/	primary
inheritance depth	/	2.18	2.00	7.00	primary
aggregation	491	/	/	/	extended
aggregation submodule	/	10.03	6.00	62.00	extended
inheritance & aggregation	1229	/	/	/	extended

For RQ1.1, we find that 1,817 archetypes have the specified parent, while only one archetype has sub-modules(it has specified parent as well). The latter phenomenon is unreasonable, and we infer it is because these archetypes are sub-modules of other archetypes which are out of our study range. Therefore, besides previous basic 2,326 archetypes, we expand our study scope to parent archetypes upward POM inheritance chains and find that 491 archetypes are sub-module projects of their direct parent. Moreover, Table I shows the average, median, and max POM Inherited Depth for 1,817 child archetypes, and the average, median, and max sub-module number for 1,296 archetypes which are both multi-module and parent projects.

RQ1 *Pom Inheritance* is adopted by more than 2/3 of the studied archetypes, and POM Inherited Depth is always more than 1. About 1/5 of the studied archetypes are sub-modules of their direct parent, which means POM aggregation and inheritance are often used simultaneously. When developers desire to scale up their archetypes, POM aggregation and inheritance are encouraged to be utilized.

C. RQ2: What is the usage pattern of dependencies and plugins in archetypes?

According to the principle about the wisdom of the crowds in software engineering, referring to other developers' decisions on the library can avoid some pitfalls experienced by other developers [10]. Therefore, we try to explore the following subquestions:

- RQ2.1 What is the utilization distribution of dependencies and plugins?
- RQ2.2 Which dependencies and plugins in archetypes are frequently adopted, and in which scenario?
- RQ2.3 Which dependencies and plugins are frequently adopted at the same time and why?

Figure 5a and Figure 5b present the utilization distribution of dependencies and plugins, respectively, and each one of distributions is right-skewed. The x-axis shows the number of archetypes using a type from the dependency or plugin. The y-axis shows the fraction of total dependencies or plugins falling within that utilization number bin. It indicates that the majority of dependencies or plugins show low utilization, and this result is similar to the study about utilization distribution of Maven Components among open-source Java projects [11].

For RQ2.2, Figure 6a and Figure 6b illustrate the top 20 frequently used dependencies (from overall 3,188 distinct dependencies) and the top 20 frequently used plugins (from overall 444 distinct plugins) at the level of GroupId. The most popular dependency is junit:junit appearing in 536 archetypes, and the most frequently used plugin is org.apache.maven.plugins:maven-compiler-plugin appearing in 643 archetypes.

The result of frequent item-set mining shown in Table III presents what types of dependencies or plugins are utilized together in high frequency (RQ2.3). We remove the most frequently used dependencies or plugins in this result like junit since they nearly appear in every archetype. In Table III, the tag means the category of dependency or plugin. And S, C, L, D, P means support, confidence level, lift, dependency, and plugin, respectively.

Moreover, we summarize three relationships based on the mining result to explain why they always work along, which are also indicated in Table III:

- **Functionally Related.** Functionally related dependencies or plugins usually belong to the same Java function module, such as log and test.
- **Tool.** Some dependencies (or plugins) may support others as tools.
- **Up-Down-Stream.** The downstream dependencies (or plugins) usually depend on the upstream dependencies (or plugins) to realize interfaces.

RQ2 Very few dependencies or plugins are frequently used, while most of them are hardly used. We make a two-level classification for popular dependencies according to their function and usage scenarios. The result of frequent item-set mining gives a primary answer to what types of dependencies or plugins always work along, and we summarize the relationships to explain the mining result.

D. RQ3: How archetypes evolve during the lifecycle?

Configuration management includes controlling the changes to the items such as dependencies in the system throughout their life cycle [12]. In order to analyze the evolution of archetypes during the life cycle of POM files, we try to answer several subquestions as follows:

- RQ3.1 How often do archetypes release a new version?
- RQ3.2 Which are the most frequently changed items when archetypes evolve?

TABLE IV: Overview of dependency/plugin addition and deletion

Item	Median	Mean
Dependency Add Rate	0.25	0.46
Dependency Del Rate	0.25	0.31
Plugin Add Rate	0.50	0.79
Plugin Del Rate	0.50	0.50

Item	Number
Dependency Addition	932
Dependency Deletion	683
Dependency Addition and Deletion	539
Plugin Addition	461
Plugin Deletion	298
Plugin Addition and Deletion	193

An archetype version number composes of major, minor, incremental version and qualifier. We regard only major and minor version number change as an iteration in this question for incremental numbers and qualifiers are optional. In our dataset, archetypes release a new version every 150 days on average and have 7,018 iterations in total.

From the results in Figure 7, we can find that the majority of change types are associated with dependencies and plugins, especially their version. Fig. 8 shows the occurring ratio for each evolution type. (In one iteration, it will be accumulated when calculating, if the same item changes.)

We also notice that the high frequency of dependency and plugin addition or deletion, so we calculate the added or deleted number of dependencies/plugins and their rate for each iteration. The added dependency rate is calculated as the following equation:

$$rate_{added\_dependency} = \frac{Cnt_{added\_dependency}}{Cnt_{total}} \qquad (1)$$

where $Cnt_{added\_dependency}$ is the number of added dependencies and Cnt_{total} is the number of total dependencies in the old version of archetype. The other three rates are similar to this equation.

As shown in the first part of Table 9, the number of added or deleted plugins is less than that of dependencies in general, while the add rate (or del rate) of plugins is slightly higher than that of dependency as shown in Table IV. The second part of Table IV gives an overview of the total number of addition and deletion behaviors for dependency and plugin. It shows that almost half of addition and deletion appear together. Figure 10a and Figure 10b show the time of dependency/plugin addition and deletion behaviors. Dependency addition and deletion become frequent since 2007, while plugins since 2009.

RQ3 In general, archetypes release a new version every 2 to 5 months. For archetype iterations, the items about dependencies and plugins change most often. The reason is that archetype developers always update the dependency and plugin information and add/delete dependencies and plugins. This reflects the high-level attention on the dependency/plugin management from archetype authors.

V. Threads to Validity

In our study, we use the second latest version of archetype to carry out our research, which may cause subtle deviations in our research results. Another threat to validity concerns the definitions of the archetype iteration, which may slightly influence our findings. However, this will not change the trend of overall results on how archetypes evolve.

VI. Conclusions

In general, Maven archetypes are the best Maven practices which are well worth of research. This paper presents an empirical analysis of 2,326 Maven archetypes hosted by the Maven central repository. We focus on the Maven archetype configuration from the aspects of patterns, uses of dependency/plugin, and evolution, and provide some interesting experimental results. We public our datasets, including 2,326 archetypes, 10,184 releases, and result data in this study. We hope this paper will benefit further study on this topic.

References

[1] Wikipedia, "Apache maven," https://en.wikipedia.org/wiki/Apache_Maven, 2019.

[2] A. S. Project, "Apache maven," https://maven.apache.org/pom.html#What_is_the_POM/, 2019.

[3] ——, "Apache maven archetype," https://maven.apache.org/guides/introduction/introduction-to-archetypes.html, 2019.

[4] A. Benelallam, N. Harrand, C. Soto-Valero, B. Baudry, and O. Barais, "The maven dependency graph: a temporal graph-based representation of maven central," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 344–348.

[5] T. O'Brien and M. V. S. Inc., *Maven: the definitive guide*. O'Reilly, 2008.

[6] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, "A survey of machine learning for big code and naturalness," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, p. 81, 2018.

[7] T. J. Speicher and Y. Cheon, "Composing a cross-platform development environment using maven," in *Proceedings of the RCCS+ SPIDTEC2 Workshop on Regional Consortium for Foundations, Research and Spread of Emerging Technologies in Computing Sciences, Juarez, Mexico*, 2018, pp. 68–80.

[8] Y. Lu, J. Liang, Y. Xiao, S. Huang, D. Yang, W. Wang, and H. Lin, "Xmlvalue: Xml configuration attribute value recommendation," in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1. IEEE, 2017, pp. 202–207.

[9] R. Srikant, "Fast algorithms for mining association rules and sequential patterns," Ph.D. dissertation, Citeseer, 1996.

[10] Y. M. Mileva, V. Dallmeier, M. Burger, and A. Zeller, "Mining trends of library usage," in *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*, 2009, pp. 57–62.

[11] H. Sajnani, V. Saini, J. Ossher, and C. V. Lopes, "Is popularity a measure of quality? an analysis of maven components," in *2014 IEEE international conference on software maintenance and evolution*. IEEE, 2014, pp. 231–240.

[12] I. of Electrical and E. Engineers, *IEEE Standard for Software Configuration Management Plans*. IEEE, 1990.