The Reaction of Open Source Projects to C++ Templates and Lambdas: An Empirical Replication Study

Donghoon Kim
Department of Computer Science
Arkansas State University
Jonesboro, AR, USA
dhkim@astate.edu

Loc Ho
Department of Computer Science
Arkansas State University
Jonesboro, AR, USA
loc.ho@smail.astate.edu

***Abstract*—New language features are added into a programming language to make high quality software. However, new features are not always welcomed in the programming community since they have pros and cons. Templates and lambdas have benefits so both features were added in many programming languages, such as C++, Java, and C#. To find improvements, the programming community wants to know how these features are actually being used in projects. Researchers have conducted these studies in different languages and different experimental environments. In this study, we conduct an empirical replication study with C++ open source projects in the same experimental environment we've conducted with Java and C# to ascertain what the community wants. Our framework of the static analysis tool for Java and C# has been extended to analyze C/C++ open source projects with quantitative data. We investigate how two language features—templates and lambdas—are used in C++ open source projects. We found that C++ templates are used widely by projects and developers, but C++ lambdas are not used widely. These results of this study are similar to those in other programming languages and in other experimental environments.**

***Index Terms*—programming language, language feature, static analysis tool, template, lambda expression, open source project, quantitative data**

I. Introduction

Programming languages have many language features for producing good software. Whenever a new feature is introduced for a language, it is thoroughly tested to see if it is needed for the language [1], [2]. New features can bring efficiency to programmers, improve program performance, and provide benefits in many ways [3], [4]. On the other hand, there are disadvantages to new features. Thus, new features are not always welcomed in the programming community [5]–[7].

Generics feature (templates in C++) offers code reuse without compromising static type checking so it can avoid code duplication [8]. In addition, type checking in C++ templates can be conducted at compile time, rather than runtime. However, C++ templates still have some limits with low readability which can lead to difficulty for debugging the codes when a project uses many templates [9]. Lambda expression (lambda), a core feature of functional programming, makes concise syntax which may lead to reduce LOC (Lines of Code) [10]. However, it may impact performance for executable size and execution time as well as making the code less readable [10]. These features—generics (templates in C++) and lambdas—have clear benefits, but they also have disadvantages that can make them difficult to use [11]. Thus, the programming community, including programming language designers and educators, wonders how these features are actually being used to find improvements for quality software, and teaching programming languages [11]–[15].

This paper presents an empirical study of C++ templates and lambdas. The work is a replication of previous studies with C# and Java [5], [6], [16], [17]. Our previous studies analyzed the usage of generics and lambdas, who used them, and how their benefits appeared in open source projects. In this study, the usage of two programming language features—templates and lambdas—in 20 open source projects are investigated. The results will be discussed by comparing the previous studies, including our studies and other studies, such as C++ templates by Chen *et al.* [14] and C++ lambdas by Uesbeck *et al.* [11]. More specially, the following two research questions (RQ) are created to investigate each feature:

- **RQ1:** Is a language feature widely used in open source projects?
- **RQ2:** Do project members in each project broadly use a language feature after introduction into the project?

The contributions of this paper are as follows:

- **Analysis of the usage of templates and lambdas in C++:** the open source projects written in C++ were analyzed to determine if the features are used widely. The following facts have been discovered: (1) the usage of templates is much higher than that of lambdas, (2) In many projects, templates were not used long after the project started, and (3) For projects that don't use lambdas, the introduction of templates tends to be late.
- **Analysis of the developers in open source projects:** this analysis results show that one or two developers used

DOI reference number: 10.18293/SEKE2020-121.

a lot of templates in the projects. Developers who used lambdas also used templates, but not necessarily those who used templates used lambdas.

- **An empirical replication study in major programming languages:** this study in C++ provides a consistent experimental environment with those in the other two programming languages—Java and C#. Thus, it is possible to consistently compare how language features were used in the three programming languages—C++, Java, and C#. The results in this study were also compared with other studies; similar conclusions were drawn.

The paper is organized as follows. Section II illustrates related works. Section III describes research questions and our methodologies to conduct this study. Section IV answers to the research questions with quantitative data. Finally, Section V concludes this paper.

II. Related Work

Researchers have investigated how language features are used with a variety of methods [18], [19]. Asaduzzaman *et al.* [18] discovered many developers, regardless of experience, misuse exception handling in open source Java projects.

Siek and Taha [20] addressed that templates are a powerful but poorly understood feature of the C++ language. Kim *et al.* [5] conducted an empirical study of C# generics feature in open source projects. They found that C# generics are used widely. They compared the results with Java generics [17] and explained several reasons for the different adoption rate of generics feature between C# and Java. Wu *et al.* [12], [14] analyzed how library templates influenced C++ programming in open source systems. They listed most commonly-used template libraries for C++ novices. Chen *et al.* [14] analyzed how C++ templates are used in 50 open source systems. They found that templates are useful for reducing code and C++ developers who prefer templates have no other programming experience. This work is most related to our work. They focused on the adoption of the different type of templates. Our work focuses on how the features are adopted over time and embraced by developers in open source projects.

Uesbeck *et al.* [11] conducted an empirical study of C++ lambdas and programmer experience. They analyzed participants' behaviors to solve programming tasks using lambda expressions and iterators. They found that the students have difficulty to use lambdas in programs and no benefits of lambdas have been made. Likewise, our quantitative results show that few developers are using C++ lambdas. Mazinanian *et al.* [13] analyzed how lambda expression is adapted by Java programmers. They found the reason why Java developers use lambdas. The reasons are (1) making existing code more succinct and readable and (2) avoiding code duplication. They investigated the introduction rate of lambdas over time, starting from the first commit and the last commit of the project. Lambda expression in Java has an increasing trend in the open source community. We also conduct similar methods with C++ open source projects. Our results show similar trends in C++ projects. However, not many C++ projects used lambdas yet. Nielebock *et al.* investigated the adoption of lambdas in 2,923 open source projects and in three programming languages—C#, C++, and Java [19]. They found that developers are significantly used more lambdas in C# than C++ and Java, but the lambdas are not predominantly applied in concurrent code. Like other studies [5], their study allows us to analyze how a language feature affects usages when implemented differently in different languages.

III. Research Approach

In this section, we explain the approach in this study. Section III-A introduces our research questions. Section III-B introduces the characteristics of 20 projects collected for this study. Section III-C introduces the framework and the procedure to analyze those projects with quantitative data.

A. Research Questions (RQ)

First, we investigate the adoption rate of each language feature. As we explained the pros and cons in the Introduction Section, two language features—templates and lambdas—have similar pros and cons. Developers are responsible for using language features to enhance the project's performance if they are useful. If language features are beneficial, their adoption rate to be chosen by the developers must be high, which thus leads to our first research question:

> **Research Question 1 (RQ1):** Is a language feature widely used in open source projects?

Our speculation was that more software developers use templates rather than lambdas due to several reasons: (1) The template feature was added to C++ much longer than lambda expression; and (2) The syntax of lambda expression is not that easy.

C++ templates were added in 1998 as the standard C++ feature, which is relatively earlier than other major programming languages such as Java (2004) and C# (2005). Lambda expression has been adopted in many programming languages, such as C# (2007), C++ (2011), and Java (2014). Many software developers may know how to use lambda in C++. After a project decides to use a compiler that supports these features, the team can use the features or some individuals may take the initiative on their own [5]. Thus, we want to analyze how many project members actually use templates and lambdas practically in project, which thus leads to our second research question:

> **Research Question 2 (RQ2):** Do project members in each project broadly use a feature after introduction into the project?

B. Projects Studied

To find out the answers for RQs listed above, we downloaded 20 open source projects from `Black Duck Open Hub` (formerly `Ohloh` website). All of the projects have to satisfy these requirements:

https://www.openhub.net/

- Each project should have a high level of activity.
- Each project should have at least 100,000 lines of code in C++ programming language.
- Each project should begin before C++ 11 (2011) was released, but 'xLights' is an exception because we couldn't find a project that meets the first two conditions.

Table I describes the information of 20 selected open-source projects. Twenty projects seems small, but more than several billions LOC has been analyzed in each project since we analyzed the projects over time (e.g., each commit from developers). For example, 'Google' has 1,322,907 LOC and about 56,000 commits. For 'Google', more than 300 billions LOC (= 0.65 millions LOC $\times$ 56,000 commits) has been analyzed. We assume that the average LOC for 'Google' is the half (0.65 millions LOC) of the last number of LOC (1.3 millions LOC). The table includes the name of each project, and the number of total lines of code written in C++ measured by `Black Duck Open Hub` on the date we downloaded for analysis. The name in brackets is a short name for each project that will be used in this paper.

Project	LOC (C++)
Appleseed	410,994
Boost C++ Library (Boost)	3,222,155
deal.II (deal)	1,899,336
digikam	793,638
Dlib C++ Library (Dlib)	309,498
Fawkes Robot Software Framework (Fawkes)	475,294
Google V8 JavaScript Engine (Google)	1,322,907
ICU for C/C++/Java (ICU)	779,201
KDE Frameworks 5 (KDE)	1,037,464
libc++: The LLVM C++ Standard Library (libc++)	551,586
libMesh: A C++ Finite Element Library (libMesh)	685,658
LLVM/Clang C family frontend (LLVM)	1,276,127
mangos-classic (mangos)	350,519
Mantid	1,376,385
MITK	1,162,057
MongoDB	746,533
OpenMS	428,914
Orfeo ToolBox (Orfeo)	364,063
Point Cloud Library (Point)	1,035,913
xLights	343,268

TABLE I: The 20 C/C++ projects under investigation

C. Procedure

We have a framework of the static program analysis tool for Java and C# [5]. The tool is used to analyze how language features are used in open source projects. Recently, we extended our existing framework to analyze C/C++ open source projects with quantitative data. Our framework is written with several programming languages such as python and C++ with `llvm` library and the `ws2_32` library. The new tool can extract the information to answer our research questions, such as the number of templates, lambdas, and developers. The following is the overall steps to analyze open source projects. After choosing projects that meet the requirements based on the information from `Black Duck Open Hub`, we cloned each project from its remote repository using Git and Subversion to a local machine, check out every version of every file from a project's repository and store the different file revisions in an intermediate format, and transfer this information to a database; extract language features information from each file revision and populate the information in the database server; finally analyze the data in the database to answer each research question.

IV. Experimental Results

To get an overview of adoption of the template and lambda expression features, we investigate the usage of both features in the 20 selected projects. We measure the number of both features to observe how those features are adopted. Table II shows the overall data on how templates and lambdas are used by developers. The title in each column indicates as follows:

- **Start Date**: Date the project started
- **Developers**: the number of developers involved in the project
- **First Date in Template and Lambda expression**: the first date when the first template or lambda was used by a developer (N/A means no one used lambda expression in the project.)
- **Developers in Template and Lambda expression**: the number of developers who used templates or lambdas in the project
- **Usage in Template and Lambda expression**: the number of templates or lambdas used by developers in the project

We observe that:

- The total number of templates is much higher than lambdas in all projects. Lambdas are used in only 10 projects (out of 20 projects).
- Templates were used (adopted) in one year after the projects started in 10 projects. On average, templates were used on approximately 6.5 years after the project started. On the other hand, lambdas are used (adopted) in approximately five years after lambdas were added in C++11 in the year 2011. For projects that don't use lambdas, such as *deal, Fawkes, ICU, KDE*, and *libc++*, the introduction of templates tends to be late.
- The total number of developers who used templates is higher than the total number of developers who used lambdas.

A. RQ1: Usage of Language Features

RQ1: Is a language feature widely used in open source projects?

Templates: We extracted the usage of templates over time. Figure 1(a) shows the number of templates over time. *libc++* used the most templates with 19,012 and *Appleseed* used the second most templates with 3,930 in 20 projects. In most of the projects, the numbers of templates increase sequentially.

As in Table II, you can find relatively large numbers in templates' usage rather than lambdas' usage. However, it is not easy to draw a conclusion if templates are used 'widely' or vice versa because there is a lack of clear criteria to discern between 'widely' and 'not widely' with the number of usage. For this reason, we investigate the usage of other language

Project	Start Date	Developer	Template			Lambda expression		
			First Date	Developer	Usage	First Date	Developer	Usage
Appleseed	7/3/2010	39	7/3/2010	23	3,930	3/18/2018	3	20
Boost	7/7/2000	19	3/4/2004	15	154	7/6/2016	1	1
deal	11/24/1997	10	12/11/2009	3	190	N/A	0	0
digikam	5/5/2004	140	1/9/2007	22	155	N/A	0	0
Dlib	5/2/2008	25	5/2/2008	10	440	10/18/2015	7	40
Fawkes	1/3/2004	25	11/8/2011	4	39	N/A	0	0
Google	6/30/2008	7	10/9/2008	2	8	N/A	0	0
ICU	8/16/1999	72	10/3/2009	20	35	N/A	0	0
KDE	9/28/1999	47	10/18/2009	7	8	N/A	0	0
libc++	5/11/2010	102	5/11/2010	78	19,012	7/31/2015	20	109
libMesh	1/9/2003	4	7/17/2012	3	959	N/A	0	0
LLVM	7/11/2007	181	8/7/2012	88	1,855	9/15/2014	31	130
mangos	10/13/2008	45	10/14/2008	27	166	N/A	0	0
Mantid	4/4/2007	137	10/26/2007	83	907	2/8/2016	32	29
MITK	9/6/1997	218	11/16/2002	77	840	6/3/2016	3	4
MongoDB	10/19/2007	283	12/11/2008	171	2,401	4/9/2015	127	307
OpenMS	6/11/2006	60	2/7/2014	30	128	12/13/2018	3	2
Orfeo	1/5/2006	33	5/15/2006	10	168	N/A	0	0
Point	3/2/2011	244	3/3/2011	82	382	7/4/2019	1	1
xLights	1/29/2013	31	9/26/2014	11	32	N/A	0	0

TABLE II: Software developers involved in the projects and the total usage of template and lambda expression

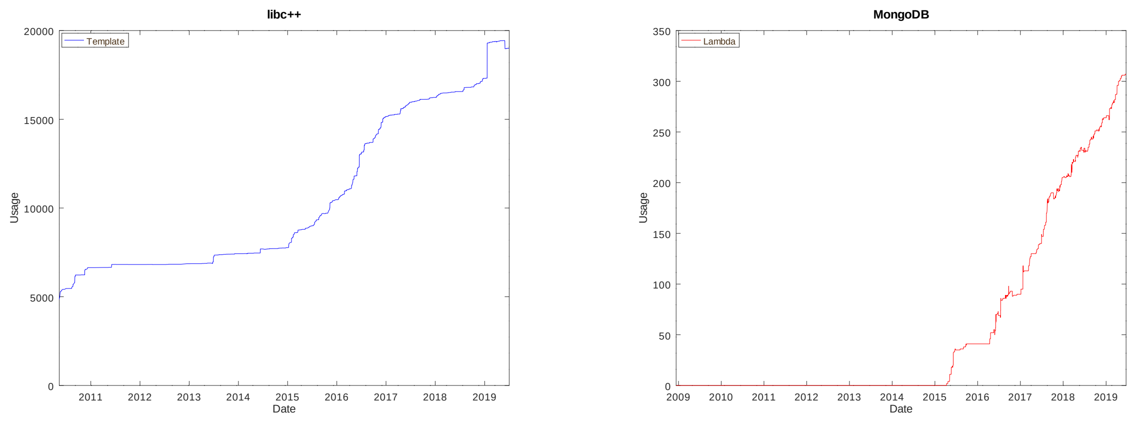


Fig. 1: Usage of features over time: (a) Templates in libc++ (left), (b) Lambdas in MongoDB (right)

Project	Template	If	Switch	Lambda
Appleseed	3,930	7,027	56	20
Boost	154	38	0	1
deal	190	1,774	26	0
digikam	155	6,985	63	0
Dlib	440	581	17	40
Fawkes	39	1,968	31	0
Google	8	406	18	0
ICU	35	7,566	62	0
KDE	8	28	0	0
libc++	19,012	1,722	20	109
libMesh	959	1,185	0	0
LLVM	1,855	1,308	58	130
mangos	166	7,690	48	0
Mantid	907	4,993	47	29
MITK	840	5,241	84	4
MongoDB	2,401	32,857	287	307
OpenMS	128	1,660	2	2
Orfeo	168	7,073	64	0
Point	382	12,472	126	1
xLights	32	1,898	32	0

TABLE III: Comparing the usage of language features with *if* and *switch* statements

features to compare the relative numbers. `if` and `switch` statements have been selected [21]. The `If` statement is the most popular feature in projects from GitHub repositories while the `switch` statement is less popular feature [21]. Table III shows the numbers of language features (i.e., Template, If, Switch, and Lambda) used in the projects. The number of templates used by the *Appleseed* project is 3,930. The *libc++* project is the highest number with 19,012 which is quite higher than the number of `if` statement with 1,722. The template feature can be one of popular features since the numbers of templates have a similar relationship with `If` statement. **Overall, our results suggest that templates are widely used in open source projects.**

Lambdas: The numbers of projects using lambdas are much smaller than the numbers of project using templates. 10 projects (out of 20 projects) used lambdas. *MongoDB* used the most lambdas with 307 and *LLVM* used the second most lambdas with 130 in 10 projects. Figure 1(b) shows the number of lambdas over time in *MongoDB*. As mentioned earlier, `switch` statements are a less popular feature [21]. When lambdas are compared with `switch` statements, the number of lambdas is less than the number of switch statements for most of the projects except *Dlib*, *libc++*,

Project	All	None	Template	Template only	Intersection	Lambda only	Lambda
Appleseed	39	16 (41.0 %)	23 (59.0 %)	20	3	0	3 (7.7 %)
Boost	19	4 (21.1 %)	15 (78.9 %)	14	1	0	1 (5.3 %)
deal	10	7 (70.0 %)	3 (30.0 %)	3	0	0	0 (0.0 %)
digikam	140	118 (84.3 %)	22 (15.7 %)	22	0	0	0 (0.0 %)
Dlib	25	15 (60.0 %)	10 (40.0 %)	3	7	0	7 (28.0 %)
Fawkes	25	21 (84.0 %)	4 (16.0 %)	4	0	0	0 (0.0 %)
Google	7	5 (71.4 %)	2 (28.6 %)	2	0	0	0 (0.0 %)
ICU	72	52 (72.2 %)	20 (27.8 %)	20	0	0	0 (0.0 %)
KDE	47	40 (85.1 %)	7 (14.9 %)	7	0	0	0 (0.0 %)
libc++	102	22 (21.6 %)	78 (76.5 %)	60	18	2	20 (19.6 %)
libMesh	4	1 (25.0 %)	3 (75.0 %)	3	0	0	0 (0.0 %)
LLVM	181	92 (50.8 %)	88 (48.6 %)	58	30	1	31 (17.1 %)
mangos	45	18 (40.0 %)	27 (60.0 %)	27	0	0	0 (0.0 %)
Mantid	137	53 (38.7 %)	83 (60.6 %)	52	31	1	32 (23.4 %)
MITK	218	140 (64.2 %)	77 (35.3 %)	75	2	1	3 (1.4 %)
MongoDB	283	89 (31.4 %)	171 (60.4 %)	67	104	23	127 (44.9 %)
OpenMS	60	30 (50.0 %)	30 (50.0 %)	27	3	0	3 (5.0 %)
Orfeo	33	23 (69.7 %)	10 (30.3 %)	10	0	0	0 (0.0 %)
Point	244	162 (66.4 %)	82 (33.6 %)	81	1	0	1 (0.4 %)
xLights	31	20 (64.5 %)	11 (35.5 %)	11	0	0	0 (0.0 %)
SUM	1,722	928 (53.9 %)	766 (44.5 %)				228 (13.2 %)

TABLE IV: Analysis of Developers

LLVM, and *MongoDB*. **Overall, our results suggest that lambdas are not widely used in open source projects**. Similar results were found in Java lambda expression [22]. Uesbeck *et al.* observed that lambdas don't benefit developers in terms of time to completion, or compiler errors [11]. However, as Mazinanian *et al.* found an increasing trend in the adoption rate of lambdas [13], the adoption rate of C++ lambdas also shows an increasing trend in some projects which used lambdas 1(b). This indicates that if we analyze C++ lambda expression again in a decade, the answer may be changed.

B. RQ2: Who used Language Features

RQ2: Do project members in each project broadly use a language feature after introduction into the project?

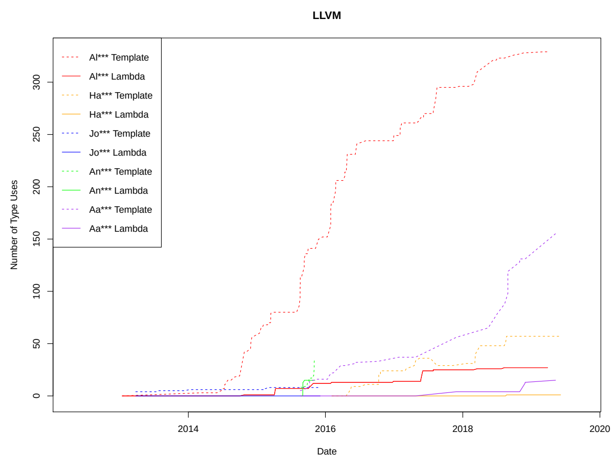
Templates: As the answer to RQ1 , C++ templates are used by most projects. However, few developers may take major usage of templates or vice versa. To evaluate RQ2, we examined how many developers used templates. Table IV shows the number of developers who used templates and lambdas. Each title in Table IV represents:

- **All**: the total number of developers involved in the project
- **None**: the number of developers who didn't use both template and lambda
- **Template (Lambda)**: the number of developers who used templates (lambdas)
- **Template (Lambda) only**: the number of developers who only used templates (lambdas)
- **Intersection**: the number of developers who used both templates and lambdas

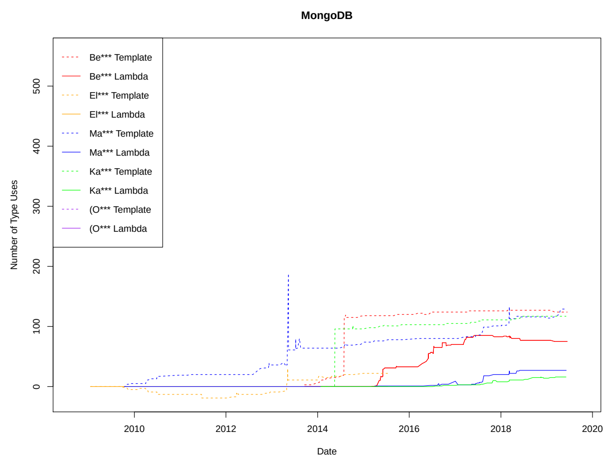
In SUM in Table IV, 44.5% (766 out of 1,722) developers used templates. More than 50% developers used templates in 8 projects. *Boost* is the highest project with 78.9% (15 out of 19) of developers using templates. *libc++* is the second highest project with 76.5% (78 out of 102) of developers using templates. These facts indicate that most developers understand the benefits of the template feature. *digikam* is the lowest project with 15.7% (22 out of 140) of developers using templates. *Fawkes* is the second lowest project with 16.0% (4 out of 25) of developers using templates. In 13 projects, more than 50% developers did not use both templates and lambdas.

Figure 2 shows the introduction and removal of both templates and lambda expressions by the top 5 developers per project. Top 5 developers means 5 developers using the most templates in a project. A dashed line represents the number of templates while a solid line represents the number of lambdas. We observed that one or two developers show higher usage of templates in the projects. For example, Figure 2(a) shows that two developers used significantly higher templates than other developers; *Al**** used more than 300 templates and *Jo**** used more than 150 templates. In the *appleseed* project which is not in Figure 2, one developer (*Fr****) used around 3,700 templates (out of 3,930) while other developers used less than 100 templates. This pattern was observed in many of the other projects such as *appkeseed*, *Boost*, *digikam*, *dlib*, *Fawkes*, *ICU*, *libMesh*, and *Orfeo*. **Overall, our results indicates that templates are used by a small pool of developers.** These results are similar to other previous studies [5], [14], [17].

Lambdas: As can be seen by the analysis results in RQ1, we recognize that not many project members used lambdas. Table IV shows that 13.2% (228 out of 1,722) used lambdas. In *LLVM* (Figure 2(a)), *Al**** (who used the most templates) used the most lambda. *Jo**** (who used the second most templates) used the second most lambdas. In *MongoDB*, 44.9% (127 out of 283) used lambdas. Several developers (*Be****, *Ma****, *Ka****) used almost 100 templates; *Be**** used 75 lambdas. In most projects, developers who used lambdas also used templates, except *MongoDB*; In *MongoDB*, 23 developers only used lambdas, not templates. **Overall, C++ lambdas are used by a very small number of project members.** Nine years have passed since lambda expression was added in C++ in 2011. We need to look at whether each project uses C++11 compiler (or higher version) to enable lambdas and when each project begins to use C++11 compiler, but for now only a few developers use C++ lambdas.



(a) LLVM



(b) MongoDB

Fig. 2: Individual developers' usage of templates and lambdas over time

V. Conclusion

Template and lambda expression features have benefits. Throughout the empirical study with C++ open source projects, we investigated the usage of templates and lambdas and how these features are used by developers. We found that templates are used widely by many developers in open source projects. However, lambdas are not used widely in open source projects, and not many developers use lambdas. These results from quantitative data will help you create a new programming language, add new language features into the programming language, or teach programming with what priorities. Since language features were added at different times, we recognize that quantitative comparisons are limited. Further research with qualitative data may be needed to analyze the reasons for these conclusions.

References

[1] G. Bracha, N. Cohen, C. Kemper, S. Marx, M. Odersky, S.-E. Panitz, D. Stoutamire, K. Thorup, and P. Wadler, "Adding generics to the java programming language: Participant draft specification," 2001.

[2] J. Järvi, J. Freeman, and L. Crowl. (2008, Feb) Lambda functions and closures for C++ (Revision 4), Technical Report N2550=08-0060, ISO/IEC JTC 1, Information technology, Subcommittee SC 22, Programming Language C++. [Online]. Available: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2550.pdf

[3] B. Stroustrup, "Evolving a language in and for the real world: C++ 1991-2006," in *Proceedings of the third ACM SIGPLAN conference on History of programming languages*. ACM, 2007, pp. 4–1.

[4] R. Dyer, H. Rajan, H. A. Nguyen, and T. N. Nguyen, "Mining billions of ast nodes to study actual and potential usage of java language features," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 779–790.

[5] D. Kim, E. R. Murphy-Hill, C. Parnin, C. Bird, and R. Garcia, "The reaction of open-source projects to new language features: An empirical study of c# generics." *Journal of Object Technology*, vol. 12, no. 4, pp. 1–1, 2013.

[6] C. Liddell and D. Kim, "Analyzing the adoption rate of local variable type inference in open-source java 10 projects," *Journal of the Arkansas Academy of Science*, vol. 73, no. 1, pp. 51–54, 2019.

[7] D. Kim and G. Yi, "Measuring syntactic sugar usage in programming languages: an empirical study of c# and java projects," in *Advances in Computer Science and its Applications*. Springer, 2014, pp. 279–284.

[8] M. H. Austern, *Generic programming and the STL: using and extending the C++ Standard Template Library*. Addison-Wesley, 1999.

[9] Á. Sinkovics and Z. Porkoláb, "Implementing monads for c++ template metaprograms," *Science of computer programming*, vol. 78, no. 9, pp. 1600–1621, 2013.

[10] J. Järvi and J. Freeman, "C++ lambda expressions and closures," *Science of Computer Programming*, vol. 75, no. 9, pp. 762–772, 2010.

[11] P. M. Uesbeck, A. Stefik, S. Hanenberg, J. Pedersen, and P. Daleiden, "An empirical study on the impact of c++ lambdas and programmer experience," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 760–771.

[12] D. W. 0014, L. Chen, Y. Zhou, and B. Xu, "An empirical study on the adoption of c++ templates: Library templates versus user defined templates." in *SEKE*, 2014, pp. 144–149.

[13] D. Mazinanian, A. Ketkar, N. Tsantalis, and D. Dig, "Understanding the use of lambda expressions in java," *Proceedings of the ACM on Programming Languages*, vol. 1, no. OOPSLA, p. 85, 2017.

[14] L. Chen, D. Wu, W. Ma, Y. Zhou, B. Xu, and H. Leung, "How c++ templates are used for generic programming: An empirical study on 50 open source systems," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 29, no. 1, pp. 1–49, 2020.

[15] A. P. Black, K. B. Bruce, M. Homer, and J. Noble, "Grace: the absence of (inessential) difficulty," in *Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software*. ACM, 2012, pp. 85–98.

[16] C. Saldivar, R. Clayton, and D. Kim, "The adoption rate of lambda expressions in java open source projects," *31st Annual National Conference on Undergraduate Research*, 2017.

[17] C. Parnin, C. Bird, and E. Murphy-Hill, "Java generics adoption: how new features are introduced, championed, or ignored," in *Proceedings of the 8th Working Conference on Mining Software Repositories*. ACM, 2011, pp. 3–12.

[18] M. Asaduzzaman, M. Ahasanuzzaman, C. K. Roy, and K. A. Schneider, "How developers use exception handling in java?" in *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 2016, pp. 516–519.

[19] S. Nielebock, R. Heumüller, and F. Ortmeier, "Programmers do not favor lambda expressions for concurrent object-oriented code," *Empirical Software Engineering*, vol. 24, no. 1, pp. 103–138, 2019.

[20] J. Siek and W. Taha, "A semantic analysis of c++ templates," in *European Conference on Object-Oriented Programming*. Springer, 2006, pp. 304–327.

[21] M. J. Lemay, "Understanding java usability by mining github repositories," in *9th Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[22] C. Mcdougal, B. Staufer, J. Reach, and D. Kim, "The evolution of java involving lambda," *Fifteenth Annual Consortium for Computing Sciences in Colleges Mid-South Conference In Cooperation With ACM/SIGCSE*, 2017.