Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Hausarbeit im Proseminar:

Softwareentwicklung in der Wissenschaft

Modern programming languages: Clojure

Christian Wolff

christian.wolff-1@studium.uni-hamburg.de

Studiengang: Informatik

Matr.-Nr.: 7315329

Betreuerin: Georgiana Mania

Table of contents

1 Introduction

Since its release, Clojure has found its way into well-known companies such as Netflix, Apple, and Walmart that use this programming language within their IT-system. These companies and more than 173 others find Clojure convenient to adopt due to its focus on being dynamic, offering interactive development support, and a stable infrastructure for multithreaded programming through an as-standard immutable data structure.[1][2] While it was only being hosted on the Java Virtual Machine in the beginning, Clojure gained further popularity through its development towards implementing interoperability to other platforms, e.g. Javascript and C#. Hence, it is possible to use this particular language on multiple platforms. The combination of a simple but concise syntax and a compact code, which is still very expressive, led to a high demand on Clojure programmers in those companies that has not been saturated until now.[3]

A simple syntax and compact code or simplicity can often be misunderstood as being an easy to use and easy to learn programming language, which cannot provide the level of complexity that is needed for today's requirements in the IT-industry. Clojure, however, offers mighty tools to satisfy the level of complexity that is needed nowadays through its source-code-is-data principle and the flexible macro system. In addition, it also provides the non-text-based syntax, which it has taken over from its predecessor LISP. This leads people to refer to Clojure as a programmable programming language. Namely, it is the functional programming aspect with the requirement of treating everything as data offering nearly infinite possibilities on how to encounter problems when programming. Combining all this with an REPL that focuses on an interactive programming experience ensures the growth of Clojure's community and popularity. [4]

Furthermore, the REPL, as a tool for direct feedback of your coding, leads to an overall better understanding of coherences within the code to support debugging and maintenance. Beside the other reasons mentioned above, the REPL may have been one of the top reasons why programmers enjoy using Clojure. This programmers' attitude has in turn facilitated the establishment of Clojure as an all-purpose programming language.

To elaborate the features of Clojure, this paper will first talk about the development of Clojure and its predecessor LIPS. Besides, it will also highlight its benefits of being a hosted language on the Java Virtual Machine and show the differences in size and usability of the code base. Not only does being hosted ensure Clojure's interoperability and portability but also it grants easy access to the libraries of the hosting platform helping to continue using best practices from other platforms in Clojure. Also, this paper will introduce the basics of functional programming, Clojure's persistent data structure and its approach to concurrency. Finally, a discussion of the findings and the conclusion of the entire paper will be presented in the last chapter.

1.1 The development of Clojure

According to its developer, Richard Hickey, the name Clojure is meant as a pun towards “closure”, which is a technique in functional programming. Besides, it also contains the initial letters of C#, Lisp, and Java that had major influences in the design of Clojure as a programming language.

Before Clojure was released, Hickey tried to provide interoperability between Lisp and Java in projects such as “jfli”, “FOIL” and “Lisplets”, which have been the base for the development of Clojure, in his earlier attempts. After 2,5 years of its development, Hickey released his first version of Clojure in 2007. Most of this development process had been self-funded until then. After the release of the first version, Hickey shifted the further development process of this programming language to be more community-driven and everyone was invited to submit improvements or problems on clojure.org. Even though the community-driven development was widely promoted by Richard Hickey, as the benevolent dictator for life (BDFL) of Clojure, he has the last say in any major decisions made regarding Clojure.

To further improve its commercial support and to stay in touch with the community, Hickey holds conferences all over the world every year. The majority of these conferences are organized by Cognitect, a company co-funded by Richard Hickey, which offers IT-solutions focusing on Clojure. His hard work and dedications seem to pay off. Not only is he able to grow his community through these conferences, he has also been able to attract major companies across different industries. Nasa, Netflix, Apple, and Walmart are among those companies that have been implementing Clojure in their IT-solution to support their businesses.

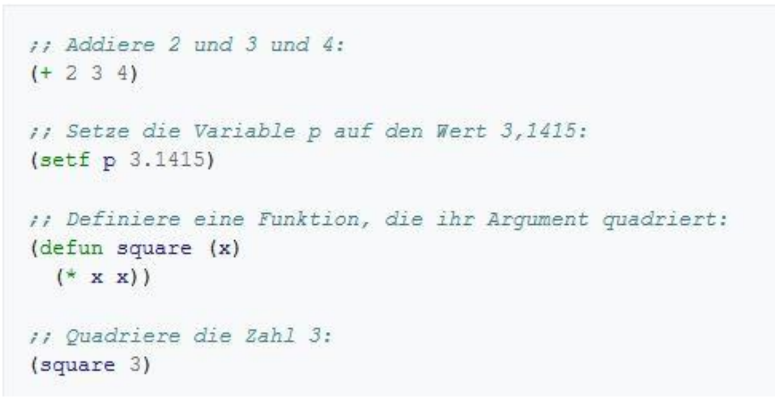
The principle behind Clojure is to support concurrency and offering an interface between a modern Lisp, as a functional programming, and Java, as a well-integrated platform. This is due to the fact that Lisp cannot provide the functionality needed to ensure the interoperability towards Java on its own because it is considered to be outdated in the programming industry. Therefore, Clojure is not often referred to as a programming language on its own but as a dialect of Lisp. This means that Clojure takes over all the basic functions of Lisp such as the syntax and the macro system and improves them. [5]

1.2 LISP

LISP, short for List Processor, was developed in 1958 at the Massachusetts Institute of Technology (MIT). It is the second oldest programming language right after Fortran, another programming language that is still in use until today. Even though pure LISP itself is not widely used anymore, some of its dialects such as “Common Lisp”, “Scheme” and “Clojure” are still quite popular and commonly used by programmers nowadays.

The basic structure of LISP are atoms and lists. Lists can consist of either other lists, which can also have further lists nested within them, or atoms that consist of letters, numbers or strings. LISP is also homoiconic, namely it does not differentiate between data and programs since everything comprises of lists. In addition, it is also referred to as a dynamic language because it will automatically reserve the space for executing a program.

The most visible difference between LISP and Java or Python is the non-text-based syntax which is common for functional programming languages. [6][7]



```
;; Addiere 2 und 3 und 4:
(+ 2 3 4)

;; Setze die Variable p auf den Wert 3,1415:
(setf p 3.1415)

;; Definiere eine Funktion, die ihr Argument quadriert:
(defun square (x)
  (* x x))

;; Quadriere die Zahl 3:
(square 3)
```

Figure 1: Example of LISP code (Common Lisp) [A1]

The syntax in LISP consists of a lot of brackets. It is possible to directly define operations such as simply add up numbers or define variables such as pi within those brackets. Besides, it is also manageable to define and call functions like in the example above. First, there is a function defined that calculates the square of a given number. Afterwards it is called to calculate the square of three. Since LISP is not text-based, this is all that is needed to create or call a function showing the difference in size between LISP and Java or Python.

2 Features of Clojure

To date, there is a huge variety of programming languages that are capable of solving any given problem, with differences in speed, handling comfort, and complexity. Therefore, there is no specific need for a certain programming language. However, to be able to compete with other well-implemented and popular languages, it is important for a programming language to offer features that can help them to stand out. For instance, features that support usability and speed will allow a programming language to be more popular because of its ability to provide solutions for every individual problem. These are some of the reasons why Clojure can stand out among other programming languages. In addition, it has a self-understanding of not only being a niche language.

2.1 Using the Java Virtual Machine

The Java Virtual Machine (JVM) is a well implemented platform which were introduced in 1994. Since then, many updates have improved the security and usability of the JVM. Clojure, which has been designed to be a hosted language greatly benefits from this well implemented platform by compiling all functions into Java bytecode. By using Java bytecode, Clojure is also taking advantage of the portability and platform-independency of Java.

While the JVM itself is not platform-independent, since there are different JVM´s for each operating system, the bytecode of Java is. A class written on a JVM on a MacOS can also be read on a JVM running on windows, which also translates to the code written in Clojure. Also, by using the JVM, Clojure’s users are able to use the java libraries such as “java.swing” to easily implement a graphical user interface (GUI). As for the security aspect the JVM already monitors during runtime the execution of a program to prevent buffer overflows, which are a commonly used security breach. [8][9]

„Clojure shrinks our code base to about one-fifth the size it would be if we had written in Java “

- *Anthony Marcar* [10]

This quote shows another feature of Clojure, especially compared to Java, where WalmartLabs was able to reduce its overall code base by one-fifth when they started to use Clojure instead of Java.

Being able to reduce the amount of code while at the same time not creating complex nested functions is another benefit of using Clojure. Less Code (with the premise mentioned above) means advantages in debugging and maintenance.

2.2 Functional Programming

Clojure is focusing on the programming paradigm of functional programming, in which the functions are treated as "first-class citizen". In functional programming, functions are not only defined and used but treated as any other data. They can be connected, used as parameters or for function results. Programs are constructed by applying and composing functions. In this context, "first-class citizen" or "first-class function" means that they can also be bound to names, passed as arguments, or also returned from other functions since functions are treated as data. Instead of using a sequence of instructions, functional programming uses complex functions which resolve in a better understanding, e.g. for calculations.[11] [12]

	imperative [13]	**functional** [14]
1	`(defn fact [n]`	`(defn fac [n]`
2	`  (loop [i n result 1]`	`      (if (zero? n) 1`
3	`    (if (zero? i)`	`            (* n (fac (dec n)))))`
4	`      result`	
5	`      (recur (dec i)`	
6	`            (* result i)))))`	

At first glance, the difference in size between both programs can be easily identified. While the functional programming only needs 3 lines of code, the imperative way needs twice as much. The imperative way of solving the factorial is by using a loop (line 2). After checking for a zero (line 3), the calculation is done recursively (line 5 and 6). When functional programming is used for the calculation of the factorial, the recursive calculation (line 3) is immediately used after an input-check for zero (line 2). This is a good example of showing that smaller code does not always mean that it must be more complex. In this comparison, the use of the functional method also helps to understand which mathematical way the code is using to calculate the factorial, while the imperative way hinders the mathematical understanding.

2.3 Dynamic/ dynamic type system

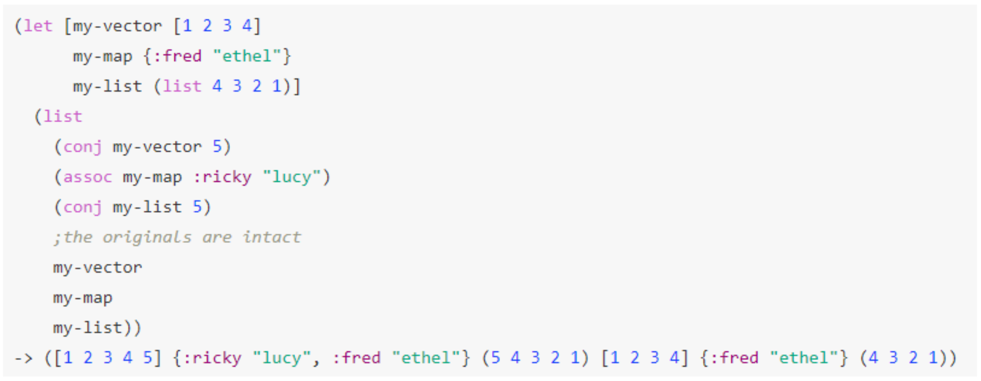
Both Clojure and Java are dynamic languages that do not require to manually reserve any kind of space on the hard drive, RAM, or cache. This means that basic programming can be done without the need to care about where something is stored, and how much space is used. Also, Clojure supports a dynamic type system where the declaration of the variables is not made during the compiling time but automatically done during the runtime.[15][16][17]

	Java	**Clojure**
1	int i=5;	(def i 5)
2	String x = “Hello World”;	(def x “Hello World”)

While it is necessary to tell, when programming in Java that “i” is an integer or “x” is a String, it will be decided and automatically interpreted during runtime when Clojure is used.

2.4 Persistent data structures

Clojure provides immutable data structures such as lists, sets, vectors, and maps. Immutable structures are of a great benefit especially when concurrency is used. Due to the immutable state, changing one of those structures by adding or removing a value does not mean changing the structure but updating it by creating a new list, vector, etc. That way, the original version of the updated structure will still be available after the change. Since the new version is based on the old version, they are both part of the same structure and both are available. [12][18]



```
(let [my-vector [1 2 3 4]
      my-map {:fred "ethel"}
      my-list (list 4 3 2 1)]
  (list
    (conj my-vector 5)
    (assoc my-map :ricky "lucy")
    (conj my-list 5)
    ;the originals are intact
    my-vector
    my-map
    my-list))
-> ([1 2 3 4 5] {:ricky "lucy", :fred "ethel"} (5 4 3 2 1) [1 2 3 4] {:fred "ethel"} (4 3 2 1))
```

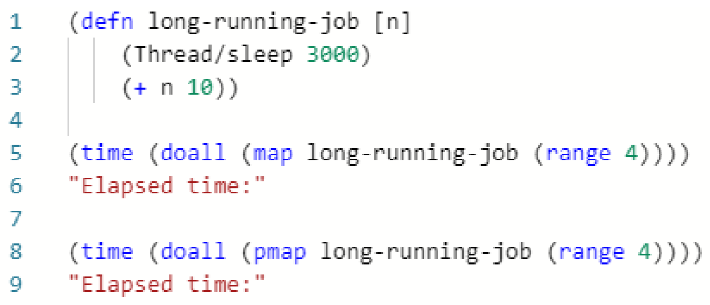
Figure 2: Example of the immutable data structure [A2]

In the example above, a vector, a map, and a list are created and filled (line 1 to 3). After that, elements are added to each one of them (line 5 to 7), which will then be shown on the console (line 12). Both the old and new versions still exist because the old version was not changed but updated to a new version.

2.5 Concurrency

Concurrency is the ability of a program or structure to use multiple threads to solve tasks, problems, or calculations. It is highly supported by Clojure to save time and make use of today multicore processors which also have multiple threads. The problem that often occurs, when concurrency is used, is the inconsistencies or conflicts among shared resources. Instead of using a method for synchronizing, Clojure benefits from its as-standard immutable data structure, as mentioned in the previous subchapter. Therefore, instead of synchronizing each thread, Clojure will copy the object so it can be shared between multiple threads. If the state of an object needs to be changed, the software transactional memory (STM) comes into use. STM is a software-based solution to avoid inconsistencies or deadlocks. [19][20][21]

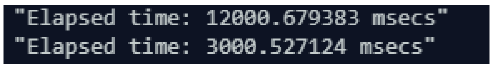
The following is a small example on how concurrency can be used in Clojure with a focus on the time-saving aspect.



```
1    (defn long-running-job [n]
2        (Thread/sleep 3000)
3        (+ n 10))
4
5    (time (doall (map long-running-job (range 4))))
6    "Elapsed time:"
7
8    (time (doall (pmap long-running-job (range 4))))
9    "Elapsed time:"
```

Figure 3: Example on the implementation of concurrency [A3]

This code defines a long-running-job with a variable n (line 1), which at first lets a thread sleep for 3 seconds (line 2) and will add 10 to the given variable/ variables (line 3) afterwards. The 3 seconds waiting time are to simulate a long running job that needs more time than what the adding of numbers would normally take. In line 5, the time measuring function is defined but it does not use concurrency. As values, the long running job will be handed *(range 4)*, which means we are handing over 4 numbers from 0-3 while simultaneously monitoring the time needed to finish the task of long-running-job. In line 8, the same function is defined with the difference where “pmap” is used instead of “map”. This signifies that concurrency is used for this operation.



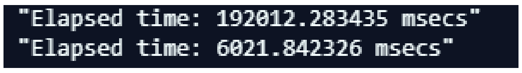
```
"Elapsed time: 12000.679383 msecs"
"Elapsed time: 3000.527124 msecs"
```

Figure 4: Results of calculating with and without concurrency (range 4)

When concurrency is not used in the long-running-job, the task will be completed after about 12 seconds, while the same task using concurrency takes only around 3 seconds to be completed. This is a prove that concurrency may safe some time in a long running job. Without concurrency, where only a single thread can be used, this thread is handed 0, the thread will wait three seconds and then adds 10, then the process starts over by using the 1, waiting three seconds and adding 10, etc. Because the waiting time per turn is always three seconds, the needed time is at least 3*4=12 seconds.

When the task uses concurrency, every thread gets handed one of the numbers of range 4, they all wait once for three seconds simultaneously and then add 10 to their given number. Since the 3 second waiting time is happening at the same time, the total time needed will be just over 3 seconds.

While a difference of nine seconds might not look like necessary, one can alter the task and increase the input range from 4 to 64 with the following result to show the advantages of concurrency:



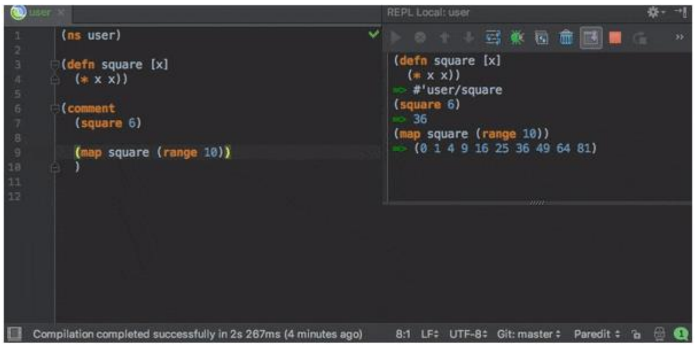
```
"Elapsed time: 192012.283435 msecs"
"Elapsed time: 6021.842326 msecs"
```

Figure 5 Results of calculating with and without concurrency (range 64)

By increasing the workload, the time saving will also be increasing drastically. When concurrency is used, the time needed to complete the task will be just above 6 seconds, whereas the single thread worked for about 3 minutes. Therefore, when time saving is needed concurrency is of great benefit.

2.6 Dynamic compilation/ REPL

In Clojure, the Read-Eval-Print-Loop (REPL) acts as a programming environment, in which it is possible to interact with your own program. Besides, it enables the identification of the effects of changes and visualizes the workflow of functions “on the fly”, with the goal to create an interactive experience. Getting a direct feedback to your input helps to prevent bugs and, for debugging purposes, it helps to reproduce the problem as well as narrow it down. [22]



```
(ns user)

(defn square [x]
  (* x x))

(comment
  (square 6)

  (map square (range 10))
  )
```

```
(defn square [x]
  (* x x))
=> #'user/square
(square 6)
=> 36
(map square (range 10))
=> (0 1 4 9 16 25 36 49 64 81)
```

```
Compilation completed successfully in 2s 267ms (4 minutes ago)    8:1  LF÷  UTF-8÷  Git: master ÷  Paredit ÷
```

Figure 6: Example of programming with the REPL [A6]

While using the REPL, one has to write down the code on the left window. By doing this, the code will automatically be evaluated and printed on the right small window, to better understand the coherences of the program. Also, it is possible to use the REPL to test just parts of any code outside of the program, allowing a better overall understanding.

3 Conclusion

Clojure offers a wide range of interesting features and advantages especially when compared to Java. Both Java and Clojure are hosted on the Java-Virtual-Machine (JVM) but Clojure uses significantly less amount of code lines than Java does. This in turn facilitates the debugging and maintenance of systems. Choosing the JVM as a platform is especially interesting for people who have worked with Java before since they can keep using the libraries that they are used to. In addition, the interactive work with the REPL and the easy use of concurrency makes Clojure a well performing, all-purpose programming language, with a good entry into functional programming. [4]

The functional programming aspect might be one of the reasons why Clojure is still quite unpopular among programmers despite a recurring trend towards functional programming in the recent years.[23] Another aspect contributing to the obscurity of Clojure is the use of a non-text-based syntax. Switching from a text-based syntax might feel like a step in the wrong direction, especially with languages such as Python where the programmer is focused on a heavy text-based syntax. Combining both of these aspects, functional programming and the non-text-based syntax, might be the reason that hinders programmers from having a closer look at Clojure. [3][4]

This non-recognition of Clojure, on one hand, makes it harder for people to learn the language autodidactic over the internet, since the number of sources, compared to the popular languages such as Java or Python, are way less. On the other hand, its low popularity relative to the high demand in the job market makes programming with Clojure one of the highest paid professions. In that sense, learning Clojure can be financially motivated for anyone aspiring to be a programmer. [3]

Finally, it should be pointed out that there are experimental implementations for interoperability between Clojure and Perl, C++, and Python, which might help increase Clojure' popularity. These three languages as such are already well implemented. Therefore, increasing the platforms that Clojure can use may also increase the number of users. [4]

4 Bibliography

[1] https://clojure.org/

[2] Companies using Clojure:

https://clojure.org/community/companies

[3] Stackoverflow "Developer Survey Results 2019"

https://insights.stackoverflow.com/survey/2019

[4] "State of Clojure Community 2020"

https://de.surveymonkey.com/results/SM-CDBF7CYT7/

[5] "Closure"

https://en.wikipedia.org/wiki/Clojure#cite_note-37

[6] "Lisp (programming language)"

https://en.wikipedia.org/wiki/Lisp_(programming_language)

[7] "Lisp"

https://de.wikipedia.org/wiki/Lisp

[8] "Hosted on the JVM"

https://clojure.org/about/jvm_hosted

[9] "Java Virtual Machine"

https://de.wikipedia.org/wiki/Java_Virtual_Machine

[10] "Clojure Made Simple" 02.06.2015

https://www.youtube.com/watch?v=VSdnJDO-xdg

[11] "Functional Programming"

https://clojure.org/about/functional_programming

[12] "Functional programming"

https://en.wikipedia.org/wiki/Functional_programming

[13] Code by G. Mania

[14] Code from: https://gist.github.com/akonring/7804273

[15] "No, dynamic type systems are not inherently more open"

https://lexi-lambda.github.io/blog/2020/01/19/no-dynamic-type-systems-are-not-inherently-more-open/

[16] "Programming Concepts: Static vs. Dynamic Type Checking"

https://thecodeboss.dev/2015/11/programming-concepts-static-vs-dynamic-type-checking/

[17] "Dynamic Development"

https://clojure.org/about/dynamic

[18] “Data Structures”

https://clojure.org/reference/data_structures

[19] “Concurrent Programming”

https://clojure.org/about/concurrent_programming

[20] “Concurrency (computer science)”

https://en.wikipedia.org/wiki/Concurrency_(computer_science)

[21] “Software transactional memory”

https://en.wikipedia.org/wiki/Software_transactional_memory

[22] “Programming at the REPL: Introduction”

https://clojure.org/guides/repl/introduction

[23] “Functional Programming is on the rise” by Roman Elizarov

https://medium.com/@elizarov/functional-programing-is-on-the-rise-ebd5c705eaef

[A1] https://de.wikipedia.org/wiki/Lisp

[A2] https://clojure.org/about/functional_programming

[A3] Code from: https://clojuredocs.org/clojure.core/pmap

[A6] https://clojure.org/guides/repl/introduction