# Why clojure

## 1. Introduction

Purpose of this document is to write comprehensive comparison and reasoning why we propose developing POC application with clojure.

## 2. Functional programming

As most of bleeding edge techniques we use for software development today, functional programming was first introduced long time ago. It was first introduced in 1950 with lisp. Mathematical background of functional programming is in lambda calculus and it offers equal computational power as Touring machine does.

Practical advantages of functional programming are:

1. Pure functions are easier to reason about
2. Testing is easier, and pure functions lend themselves well to techniques like property-based testing
3. Debugging is easier
4. Programs are written at a higher level, and are therefore easier to comprehend
5. Function signatures are more meaningful
6. Parallel/concurrent programming is easier
7. Complexity is reduced since state is kept to a minimum

*“Functional programming is often regarded as the best-kept secret of scientific modelers, mathematicians, artificial intelligence researchers, financial institutions, graphic designers, CPU designers, compiler programmers, and telecommunications engineers.”*

The Wikipedia F# page

## 4. Clojure

Lisp is one of the oldest of all programming languages, invented by John McCarthy in 1958. The original language spawned many variant dialects, the most predominant of which today are Common Lisp and Scheme. Clojure is a new dialect of Lisp created by Rich Hickey. Like Scheme, Clojure is a functional dialect, meaning that it supports and encourages programming in a "functional style".

However, Clojure differs from older lisps in several important ways. Clojure code has much more syntactic sugar. For example, instead of typing (set 1 2 3), you can write #{1 2 3}. Clojure has borrowed from Python and Haskell its heavy reliance on dumb data structures: sets, maps, list, and vectors. Clojure also features complete interoperability with Java libraries. Clojure is very pragmatic language which allows you rapid development without lot of ceremony. It is everything about the data.

### *4.1. Applications*

Very important topic in choosing language is development speed. We often observe startups and rate at which they can deliver features. Programming languages used by startups are mostly dynamic languages which are perfectly suited for rapid prototyping.  Problem with language choice becomes apparent when development teams starts to grow in number. That is point where companies move to more “Serious” programming language (Usually C#, Java).

Clojure is dynamic functional programming language. It offers immutable and persistent data structures. With access to all Java ecosystem libraries, clojure can match requirements of small rapid teams and large scale enterprise projects. It is stable, reliable and well supported ecosystem on one side. But with its dynamic properties allow very fast development pace for any team.

### *4.2. Onboarding*

Functional programming is different. It requires new way of thinking. But recent new features in Java and similar programming language are mimicking this way of thinking. And that is point where we can say that moving toward functional style programming is becoming more convenient.

But let focus on language as well. Clojure is very simple programming language. To learn syntax of language is very easy. IDE support is good and development environment is much versatile.

Another big advantage is the confidence that you get that once your Code compiles it most likely does what you intended. It forces you to also think about error scenarios at an early stage. This reduces in general time of debugging which is very expensive and time consuming.

I can claim with my recent experience that time to onboard someone on language like Typescript or Java  is same as it would be to onboard one on Clojure and tech person how to enter functional paradigm.

## *4.3. Simplicity*

Entire clojure ecosystem and community is focused on simplicity. Clojure is simple by design. Simplicity is a powerful technique for writing better programs, programs that have:

- concision: write programs 5-10x smaller codebase than before
- robustness: easily write programs that work correctly
- generality: use the same simple ideas over and over, instead of masses and masses of redundant classes
- agility: add new capabilities with ease

## *4.4. Interactive Development (REPL)*

Full access to real environment. Clojure offer native ability to connect remote REPL running even on PROD environment. This allows easy debugging. Code developed in repl is 1:1 production code and can be just copied into source.

Development machine requires not additional tools (SQL developer, Postman). Code is developed in REPL.

## *4.6. Clojure in the front end*

ClojureScript allows front end development in Clojure. It is a compiler which transforms Clojure to JavaScript. But unlike Google Web Toolkit which has a dedicated compilation run, ClojureScript does the compilation on the fly and even interacts with the browser. It is possible to develop functionality in ClojureScript without browser reload by just replacing the changed HTML snippet. (This is not even possible in Angular!)
This works also for developing native mobile apps. No compilation run is needed, changes are reflected immediately.
Clojure code can be written to run in the front end and in the back end.

## *4.7. Clojure Spec*

It is basically dictionary of predicates which defines rules for your data. It is become possible to share those dictionaries for data validation between backend and frontend. Function can specify requirements on data which it receives as parameter and returns. Spec is nice and can generate synthetic data based on the defined predicates for your tests. For more info please check https://clojure.org/guides/spec

## *4.8. Stability of the Language*

Overall main criteria for any feature in clojure is no Breakage. Acceptance of new versions is incredibly big since there are no breaking changes introduced. Language itself does not change, all features are just libraries.

https://github.com/mcohen01/amazonica

## *4.9. Object Orientation is overrated*

(from https://clojure.org/about/rationale)

- Born of simulation, now used for everything, even when inappropriate
    - Encouraged by Java/C# in all situations, due to their lack of (idiomatic) support for anything else
- Mutable stateful objects are the new spaghetti code
    - Hard to understand, test, reason about
    - Concurrency disaster
- Inheritance is not the only way to do polymorphism.
- "It is better to have 100 functions operate on one data structure than to have 10 functions operate on 10 data structures." - Alan J. Perlis
- Clojure models its data structures as immutable objects represented by interfaces, and otherwise does not offer its own class system.
- Many functions defined on few primary data structures (seq, map, vector, set).
- Write Java in Java, consume and extend Java from Clojure.
- Object Oriented approach suffers from types proliferation. Your application needs to recreate the whole world which it operates on. (As an example you can think of different kind Persons types which are represented in different external systems like SAP/Address Book etc.) Those types are not composable and because of it code  becomes bloated with Adaptors, Mappers, Wrappers, Bridges and so on. This code does not solve any business problems.
- While Smalltalk supported classes and eventually subclassing, Smalltalk was not about classes or subclassing things. It was a functional language inspired as much by Lisp as it was by Simula. Alan Kay considers classes as a code reuse mechanism to be a mistake, and the industry's focus on subclassing to be a distraction from the true benefits of object oriented programming.
    - Alan Kay coined the term "object oriented programming" at grad school in 1966 or 1967. The big idea was to use encapsulated mini-computers in software which communicated via message passing rather than direct data sharing — to stop breaking down programs into separate "data structures" and "procedures".

  - “I’m sorry that I long ago coined the term “objects” for this topic because it gets many people to focus on the lesser idea. The big idea is messaging.” ~ Alan Kay

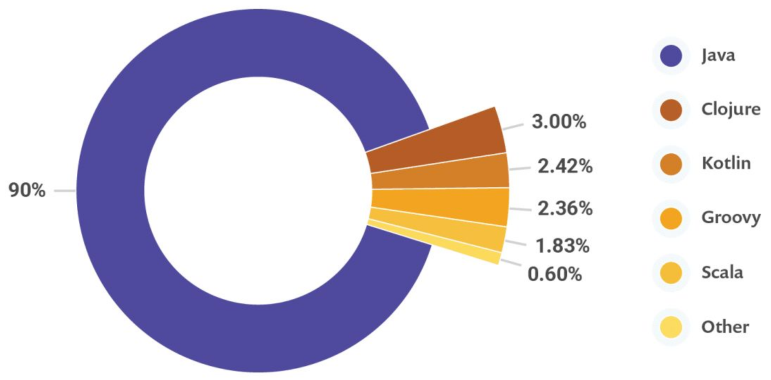
# 5. Comparison with other languages

## 5.1. Scala

This a big point. Clojure is functional and pure, it protects you. If you want to mutate transients for example and one of those is accessed from an outside thread, you get an exception. Not so with Scala.

Scala is functional in the broadest sense of the word: It has immutable constructs. Scala does not in anyway discourage it's users from side-effects and mutable code. Programmers sometimes need the box to be visible in order to not step out of it, when it's unnecessary. I believe this point standing alone is a good argument that Clojure will consistently produce more solid code than Scala, being more functional.

Finally, Clojure does lazy evaluation per default, Scala evaluates strictly. Due to paradigm mix it is require certain discipline to program in functional way.



## 5.2. Kotlin

Kotlin is a language with more sugar, yes, there are new/more aspects of functional programming, but still, it is primarily object-oriented, sharing same paradigm with Java.

When you write programs in Clojure you are forced to shape your thinking, while in Kotlin you can cut corners and write programs exactly in the same way as you'd write in Java, without even trying new features (which mainly come from functional programming).

Immutable data structures are not part of language yet. While in clojure, developers per default have to deal with immutability kotlin does not enforce that kind of behavior. Main clam here is that mutability if most common source of bugs and complexity. I would by all means discourage using Kotlin.

## 6. Forming community

Scope of this proposal is not only limited on developing POC application. Target is to spread knowledge and benefits beyond boundaries of one project.

Current clojure community in Vienna and Europe is very minimal compared to other parts of world. This is the point we want to influence. We would like, by support of company build up strong community that would spread beyond boundaries of company.

As we already started to do minimum contributions to open source world, we would like to make this very important factor in this initiative.