Kotlin

best student

November 21, 2017

Abstract

10-15 lines with the software technology and the highlights from the project that has been undertaken.

Contents

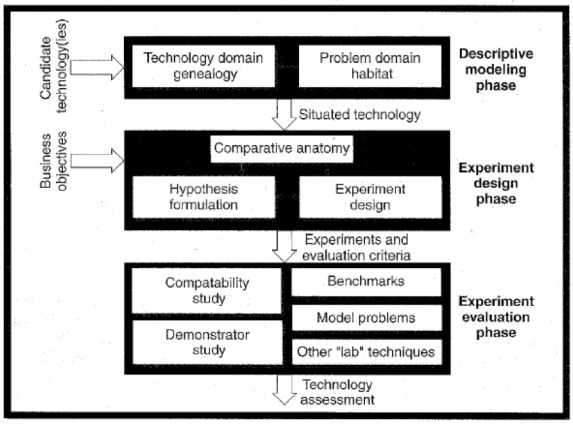

Figure 1: Software technology evaluation framework.

1 Introduction

About 4 pages that introduces in (sufficient) depth the key concepts and architecture of the technology. May use a running example to introduce the technology.

This part and other parts of the report probably needs to refer to figures. Figure 1 from [**?**] just illustrates how figure can be included in the report.

1.1 Motivation

we are a motivated lot ,';j

1.2 Kotlin

1.2.1 What is Kotlin?

Kotlin is a new and open source programming language. It is statically typed programming language that combines object oriented programming with functional programming. That means that one can use both styles of programming interchangeably. Kotlin comes from from JetBrains, the folk behind world's best IDE's. That means it comes from the industry and not academia so it focuses on solving some of the day-to-day tasks developers have to deal with. Kotlin is concise - which means one can expect to reduce amount of code lines written. Safe - it avoids errors like null pointer exceptions. Interoperable - use existing libraries for JVM, Android and the browser. Kotlin compiles down to JVM and Javascript which means that the language can be used for many purposes. Kotlin is also tool-friendly, one can pick any Java IDE or build directly from command line.

1.2.2 History

1.2.3 Functionality

Expressiveness - Kotlin's support for type safe builders and delegated properties help to build easy to user abstractions.

Scalability - Because of Kotlin's coroutines server side applications can be scaled with minimal hardware requirements. Coroutines suspend computations in a thread without blocking it. Thread blockage is often expensive, with coroutines a library can decide whether to suspend a computation or not.

Interoperability - Kotlin is compatible with all frameworks that work with Java. This is great because it lets developers use familiar technology while having the benefits of a modern language.

Migration- One can start using Kotlin and still keep older Java codebases.

Compatibility - Kotlin is fully compatible with JDK 6. This ensures that it can run on older android devices for example.

Performance - Kotlin applications run just as fast as Java due to compiling to the same bytecode. Due to inline functions and code using lambdas it runs faster than code written in Java.

1.2.4 Where to use Kotlin

Kotlin is great for server-side applications because of its many functionalities as mentioned in 1.2.3.

Kotlin can be used for Android development, it introduces new features while avoiding new restrictions that usually come when introducing a new technology.

JavaScript can be targeted by Kotlin and one can also use third party libraries such as JQuery and ReactJS.

Kotlin/Native is in undergoing development where compiling Kotlin to Native libraries will not require a VM.

1.3 Kotlin Syntax

-On the Syntax of Kotlin (some examples)

2 Android application (Blackjack)

About 5 pages that gives:

1. High-level view of the demonstrator and its purpose.
2. Details of how the demonstrator has been implemented.
3. May involve presentation of code snippets.

The example below shows how you may include code. There are similar styles for many other langages - in case you do not use Java in your project. You can wrap the listing into a figure in case you need to refer to it. How to create a figure was shown in Section 1.

```
1  public class BoksVolum {
2
3     public static void main(String[] args) {
4
5        int b, h, d;
6        String btext, htext, dtext;
7
8        [ ... ]
9
10       int volum = b * h * d;
11
12       String respons =
13             "Volum [" + htext + "," + btext + "," + dtext + "] = " +
                   volum;
14
15    }
16 }
```

2.1 Functionality

functionality of our prototype

Config	Property	States	Edges	Peak	E-Time	C-Time	T-Time
22-2	A	7,944	22,419	6.6 %	7 ms	42.9%	485.7%
22-2	A	7,944	22,419	6.6 %	7 ms	42.9%	471.4%
30-2	B	14,672	41,611	4.9 %	14 ms	42.9%	464.3%
30-2	C	14,672	41,611	4.9 %	15 ms	40.0%	420.0%
10-3	D	24,052	98,671	19.8 %	35 ms	31.4%	285.7%
10-3	E	24,052	98,671	19.8 %	35 ms	34.3%	308.6%

Table 1: Selected experimental results on the communication protocol example.

2.2 Tools and frameworks

frameworks we've used

2.3 Implementation

2.3.1 Possibly many subpoints of our different modules

3 Experiment results and comaprison to java

About 3 pages that:

Describes the software used to establish the test-bed and for implementing the demonstrator prototype.

Explains what experiments have been done and the results.

For some reports you may have to include a table with experimental results are other kinds of tables that for instance compares technologies. Table 1 gives an example of how to create a table.

3.1 The switch from Java to Kotlin

As developers with a background mainly based on java-development, we had to alter most of the habits associated with developing generic java code. T he most notable change when developing Kotlin copared to java was probably type declarations. whenever you were going to declare a variable, you could quickly find yourself trying to figure out what type it would be, so that you can declare the type of the variable. This is not necessary, as Kotlin only uses var/val when declaring a variable, where the type will be implicit from the type of what the variable is set to.

Another interesting pattern to look at is the appearances of semicolons. Coming from java, you are used to enter a semicolon whenever you have finished a statement. Since the semicolons are optional in Kotlin we've decided to avoid using them. Still, looking back at our code, you can find the occasional semicolon on lines where our java-habits likely kicked in.
Something that ma have had us hold on to java-practices may be the fact that the language is fully compatible with java, and you can use java-Classes wherever you want to. A result of this is you often end up writing lines exactly as you would in java. As an example, when you want to sort a list in java you may use:

```
1 Collections.sort(myList);
```

This line would be correct in Kotlin as you can import java.util.Collections and use it with your code. You can use your favorite java package wherever you feel like when using Kotlin, which can lead you to miss out on some of the neat functionalities in kotlin.
Fortunately the Android Studio IDE is very helpful when it comes to hints on how to improve your code. Whenever we started writing a nested if-statement, the IDE would offer to convert it to Kotlins *When()* syntax. We can take a look at the initial code we wrote for deciding who the winner of a game is, compared to what the IDE altered it to.
The initial Code:

```
1  fun getWinner(player : List<Card>, dealer : List<Card>):
       EndGameState{
2          val playerScore = getScore(player)
3          val dealerScore = getScore(dealer)
4
5          if(playerScore > 21){
6              return EndGameState.DEALER
7          }
8          else if(dealerScore > 21){
9              return EndGameState.PLAYER
10         }
11         else{
12             if(playerScore > dealerScore) {
13                 return EndGameState.PLAYER
14             }
15             else if(playerScore < dealerScore){
16                 return EndGameState.DEALER
17             }
18             else{
19                 return EndGameState.PUSH
```

```
            }
        }
}
```

Interesting to note is that this looks a lot like the code in our java-version of the project. However, the IDE advised us to change our Kotlin code to the following format:

```
    fun getWinner(player : List<Card>, dealer : List<Card>):
        EndGameState{
        val playerScore = getScore(player)
        val dealerScore = getScore(dealer)

        return when {
            playerScore > 21 -> EndGameState.DEALER
            dealerScore > 21 -> EndGameState.PLAYER
            else -> when {
                playerScore > dealerScore -> EndGameState.PLAYER
                playerScore < dealerScore -> EndGameState.DEALER
                else -> EndGameState.PUSH
            }
        }
}
```

This code looks a lot cleaner, and came as a pleasant surprise from the Android Studio IDE. Active advise from the IDE was definitely helpful when converting from java-development to kotlin-development, as it showed us some neat functionality in kotlin.

3.2 Ease of development

3.3 Boilerplate code

3.4 Documentation

3.5 Comparison to Java

3.5.1 Possibly many subsections here, guys.

4 Conclusions

Concludes on the project, including the technology, its maturity, learning curve, and quality of the documentation.

The references used throughput the report should constitute a well chosen set of references, suitable for someone interesting in learning about the technology.