Universidade Federal do Pampa

Training Basic Material

Marcelo Schmitt Laser

May 1, 2017

Introduction

Welcome to the ProjetoMBT training programme. In this document, you should be able to find all the necessary material to guide you in studying the topics used in our project, be it in regarding theory, analysis, architecture or development. The goal of this document is to help you navigate through the sources for each topic and direct your studies to each topic's most important parts. This is *not* meant to serve as a sole material for studying these topics, as it is vastly incomplete and is written under the expectation that the student will use complementary material as needed.

This document is an ongoing project, and as such any errors or incomplete sections should be notified to the author (marcelo.laser@gmail.com). It is written in English on purpose, so as to help in the study of the language and to evaluate the ability of the trainees in assessing English literature, a skill that is fundamental to the participation in this project. Each of the following paragraphs summarizes one of the document's chapters, explaining and listing its purpose, topics covered, primary sources used and primarily interested parties.

Chapter 1 discusses the topic of programming paradigms, and is meant to give the trainee a foundation of programming concepts. The purpose of this chapter is to show the differences between some programming paradigms in order to expose the notion of basic programming concepts. This way, the trainee should be able to identify more precisely what programming paradigms mean and understand how paradigms can be implemented in various languages. The paradigms explored in this chapter are Imperative (with a particular distinction with Procedural), Object-Oriented, Aspect-Oriented, Declarative, Functional, Logical, Event-Driven and Reflective. This chapter is primarily directed towards programmers, but also towards software architects and low-level software engineers, and is based on the works of Van Roy et al. [12] and Ghezzi et al. [6].

Chapter 2 discusses the basic concepts of logic for programming, with an emphasis on its uses for defining architectural constraints in software projects. The purpose of this chapter is to open the trainee to the notion of formal logic so that he may identify its overall applications. This chapter is meant only as an introduction to formal logic, and is intended to help the trainees develop the skillset related to it, not explain its practical applications. The specific logics addressed are Propositional Logic and First-Order Logic. This chapter is primarily directed at software engineers and architects, and is based on the work of Huth et al. [8].

Chapter 3 is meant as a review of software modeling concepts, with an emphasis on the UML. The purpose of this chapter is to reinforce the importance and practical application of software modeling, as well as the formal notion of software model, in order to improve the communication between members of the project. First the fundamental concepts of software modeling are presented, followed by a more in-depth exposition of certain UML diagrams (Use-Case, Activity, Class, Component, Sequence, Communications and State-Machine) and an overview of some other modeling languages and techniques. This chapter is directed at all team members, as software models are the primary means of communication within a software project. Several sources are used in this chapter, of which the main ones are the works of Taylor et al. [10] and Rumbaugh et al. [9] [2].

Chapter 4 presents an extensive treatment of Data Structures, ranging from theory and characteristics, through implementation, and ending with advantages and recommendations of each one covered. The purpose of this chapter is to enhance the understanding of the trainees regarding data structures and their importance in software architecture and development, especially in regards to how appropriate use of data structures can increase modularization and facilitate separation of concerns. The data structures covered are Arrays, Matrices, Lists (Array Lists and Linked Lists), Stacks and Queues, Trees (with an emphasis on Binary Trees and Heaps), Graphs, Classes and Tables (particularly Hash Tables and Dictionaries). This chapter is primarily directed towards programmers, though understanding it is important for low-level software engineers, particularly those that intend to work with data-intensive applications. The primary source used in this chapter is the work of Cormen et al. [3].

Chapter 5 expands on the knowledge of Chapter 4 by analysing sorting and search algorithms that are basic for the understanding of algorithm construction and evaluation. This chapter serves the same purpose as Chapter 4, presenting the importance of well-thought algorithms and standardized solutions in software engineering and development. The algorithms covered are Insertion, Bubble, Merge and Quicksort, and Breadth-First Search, Depth-First Search, A*, Dijkstra's Algorithm and Floyd-Warshall. This chapter is primarily directed at programmers, but low-level software engineers might benefit from these concepts as they are essential for architecting appropriate data-intensive applications. The primary source used in this chapter is the work of Cormen et al. [3].

Chapter 6 delves into the basic concepts of Software Patterns, the different types of patterns, the reasons for using patterns and the specifics of certain important patterns. This chapter is meant to aid in solving a large number of common problems in Software Engineering, as Software Patterns are an indispensable tool. The number of patterns covered in this chapter is very high, and therefore a complete list is not presented here. The study of patterns is the culmination of the topics seen up to this point, combining Programming Paradigm Concepts, Data Structures and Algorithms to solve practical Software Engineering problems, and making use of Formal Logic and Software Modeling to present these solutions in a concise manner. Knowledge of software patterns is indispensable to any serious Software Engineer, and therefore this chapter is directed to all trainees. The sources used in this chapter are mainly the works of Taylor et al. [10], the Gang of Four [5] and Fowler et al. [4].

Chapter 7 covers the notions of Formal Languages, with a broad introduction to the subject and its practical application in the field of Model-Based Testing, and indeed, all of Model-Based Software Engineering. As compilers theory forms the essential building blocks to all computational parsing, it is important to understand what compilers are, what they do and how they are formed, and that is what this chapter aims to do. It is not meant as a comprehensive explanation for implementing compilers, as that is beyond the scope of this training. The specific topics viewed are Regular Expressions, Automata Theory (very superficially), Lexical, Syntactic and Semantic Analysis (also superficially) and the basics of the Backus-Naur Form of grammars. This chapter is primarily directed at developers and Software Engineers with an interest in Model-Based Software Engineering. The sources used are the works of Aho et al. [1] and Hopcroft et al. [7].

Chapter 8 is a brief presentation of software testing directed at the basic applications of testing in generalist Software Engineering. This chapter in no way intends to cover the entire body of knowledge in regards to Software Testing tools, techniques, theory, cases, etc. that is studied within the ProjetoMBT group. Rather, it is intended as a basis of applied software testing so that the trainees can become acquainted with basic quality criteria and validation. The test types that will be covered are primarily Functional and Unit testing, with a brief introduction being given on Performance, Integration, Usability and Acceptance testing. This chapter is directed at the software engineers, analysts and developers amongst the trainees. Again, it is not meant as a guide to the research of the Software Testing field done in the ProjetoMBT group; for this, you should seek guidance from the domain specialists in the group.

Contents

Chapter 1

Programming Paradigms

The concept of Programming Paradigms is formally described by Van Roy and Haridi [12] as "an approach to programming a computer based on a mathematical theory or a coherent set of principles". A programming language-oriented definition, given by Ghezzi and Jayazeri [6], is that (paraphrase) "a programming paradigm is the way by which a programming language enforces a certain programming style". Finally, a more practical definition is given by Van Roy in [11] (paraphrased): "A paradigm is defined by a set of programming concepts". While all three may seem too academic an introduction to the subject, a careful analysis of these definitions can yield a good understanding of what programming paradigms mean and how they affect Software Engineering.

First, let us look at the more academical definition given by Van Roy and Haridi. The foremost characteristic of this definition, in comparison to the more orthodox one given by Ghezzi and Jayazeri, is the complete omission of the term "programming language". This is important because it emphasizes that a programming paradigm is *not just a characteristic of a language*, but rather an *approach to computational solutions*. Programming paradigms encode certain concepts and practices that have been found to be suitable for solving particular types of problems. They are a "mathematical theory" or a "coherent set of principles", and understanding the theory and principles behind a programming paradigm is extremely important for understanding how it works, what it is meant for and how best to use it.

Looking at the definition given by Ghezzi and Jayazeri, what we have is a direct correlation between programming paradigms and programming languages through what they call a *programming style*. The premise behind this definition is that programming languages must be studied within the context of software development, as this is their purpose, and therefore must be coherent with the rest of the theory of computer science, particularly Software Engineering. The authors propose that the study of programming paradigms must be seen from the viewpoint of Software Engineering, particularly software design methods. The core idea here is encoded in the following sentence: "Ideally, the design method and the paradigm supported by the language should be the same".

That is, the choice of a programming paradigm, and indeed the choice of a programming language, must be made in conjunction with the choices in design method and, consequently, with the choices in software development process. The understanding of programming paradigms is therefore essential to Software Engineering.

Finally, we have the definition given by Van Roy in his tutorial of programming concepts. This definition, unlike the previous two, is more direct and concrete, and tells of the composition of a programming paradigm, rather than its purpose or inspiration. Essentially, Van Roy is stating that programming paradigms are similar in that they share certain commonalities. Understanding these commonalities, called programming concepts by Van Roy and programming language concepts by Ghezzi and Jayazeri, allows one to better evaluate a particular paradigm in regards to what it is suited for, what are its limitations and how it is best used. By studying these "building blocks" of programming paradigms, a Software Engineer is better able to identify which design techniques to use and what processes suit what technologies.

This chapter is written following these three main precepts of programming paradigms: a) that there is strong and formal theory behind each paradigm; b) that understanding and choosing paradigms needs to be a part of the software development process, and; c) that understanding programming concepts allows one to better evaluate different paradigms. While the formal theory of programming language concepts is not presented in full, both due to lack of time and space and due to it not being the objective of this training programme, we do explore the theory of a few fundamental programming concepts and what they represent. Once a few concepts have been studied, specific paradigms will be presented to evaluate the ability of the trainees in identifying programming concepts. These paradigms are presented in comparison to one another, and were chosen due either to their representativeness of certain concepts or to their practical application within the environment of the ProjetoMBT group.

Upon completion of this stage of the training programme, the trainee should:

- understand the relationship between programming languages and paradigms;
- understand the fundaments of the presented programming concepts, and;
- understand the differences between the Imperative, Object-Oriented and Declarative programming paradigm types.

1.1 Programming Concepts

1.1.1 Record

1.1.2 Procedure

1.1.3 State

1.1.4 Object

1.1.5 Class

1.1.6 Port

1.1.7 Cell

1.1.8 Inheritance

1.1.9 Exception

1.2 Paradigms

1.2.1 Imperative

Procedural

1.2.2 Object-Oriented

Event-Driven

Reflective

Aspect-Oriented

1.2.3 Declarative

Functional

Logic

Chapter 2

Programming Logic

2.1 Propositional Logic

2.2 First-Order Logic

2.3 Other Types

Chapter 3

Modeling

3.1 Fundaments

3.1.1 Model vs Diagram

3.1.2 View vs Visualization

3.1.3 Structure vs Behaviour

3.1.4 System vs Software

3.1.5 Analysis vs Architecture

3.2 Unified Modelling Language

3.2.1 Fundamentals

3.2.2 Use-Case Diagram

3.2.3 Activity Diagram

3.2.4 Class Diagram

3.2.5 Component Diagram

3.2.6 Sequence Diagram

3.2.7 Communications Diagram

3.2.8 State Machine Diagram

3.2.9 Other UML Diagrams

3.3 Other Models and Languages

3.3.1 Entity-Relationship

3.3.2 Flow Chart

3.3.3 User Story

3.3.4 Clafer

3.3.5 XML

3.3.6 xADL

Chapter 4

Data Structures

4.1 Fundamentals

4.2 Array

4.2.1 Matrix

4.3 List

4.3.1 Array List

4.3.2 Linked List

4.4 Stack

4.5 Queue

4.6 Tree

4.6.1 Binary Tree

4.6.2 Heap

4.7 Graph

4.7.1 Directed Graph

4.7.2 Weighted Graph

4.8 Class

4.8.1 Object

4.8.2 Struct

4.9 Table

4.9.1 Hash Table

4.9.2 Dictionary

Tuple

Chapter 5

Algorithms

5.1 Sorting Algorithms

5.1.1 Insertion Sort

5.1.2 Bubble Sort

5.1.3 Quicksort

5.1.4 Merge Sort

5.2 Search Algorithms

5.2.1 Breadth-First Search

5.2.2 Depth-First Search

5.2.3 A*

5.2.4 Dijkstra's Algorithm

5.2.5 Floyd-Warshall Algorithm

Chapter 6

Patterns

6.1 Fundamentals

6.1.1 Design Pattern vs Architectural Pattern

6.1.2 Architectural Pattern vs Architectural Style

6.1.3 Architectural Style vs Programming Paradigm

6.2 Architectural Styles

6.2.1 Pipe-and-Filter

6.2.2 Batch-Sequential

6.2.3 Implicit Invocation

6.2.4 Object-Oriented

6.2.5 Distributed Objects (CORBA)

6.2.6 Client-Server

6.2.7 Mobile Code

6.2.8 Peer-to-Peer

6.2.9 Event-Based

6.2.10 Publish-Subscribe

6.2.11 Blackboard

6.2.12 C2

6.3 Architectural Patterns

6.3.1 Model-View-Controller

6.3.2 Broker

6.3.3 Layers

6.3.4 Sensor-Controller-Actuator

6.3.5 Interpreter

Chapter 7

Formal Languages

7.1 Regular Expressions

7.2 Automata

7.2.1 Finite vs Infinite

7.2.2 Deterministic vs Non-deterministic

7.2.3 Probabilistic

7.3 Lexical Analysis

7.4 Backus-Naur Form

7.5 Syntactic Analysis

7.6 Semantic Analysis

Chapter 8

Software Testing

8.1 Fundamentals

8.1.1 Verification vs Validation

8.1.2 Formal Verification vs Testing

8.1.3 System Testing vs Software Testing

8.2 Functional Testing

8.3 Unit Testing

8.4 Performance Testing

8.5 Integration Testing

8.6 Usability Testing

8.7 Acceptance Testing

Bibliography

[1] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition).* Addison Wesley, 2006.

[2] G. Booch, J. Rumbaugh, and I. Jacobson. *Unified Modeling Language User Guide, The (2Nd Edition) (Addison-Wesley Object Technology Series).* Addison-Wesley Professional, 2005.

[3] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms.* McGraw-Hill Higher Education, 2nd edition, 2001.

[4] M. Fowler. *Patterns of Enterprise Application Architecture.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[6] C. Ghezzi and M. Jazayeri. *Programming Language Concepts.* John Wiley & Sons, Inc., New York, NY, USA, 3rd edition, 1997.

[7] J. E. Hopcroft, R. Motwani, Rotwani, and J. D. Ullman. *Introduction to Automata Theory, Languages and Computability.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2000.

[8] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning About Systems.* Cambridge University Press, New York, NY, USA, 2004.

[9] J. Rumbaugh, I. Jacobson, and G. Booch. *Unified Modeling Language Reference Manual, The (2Nd Edition).* Pearson Higher Education, 2004.

[10] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. *Software Architecture: Foundations, Theory, and Practice.* Wiley Publishing, 2009.

[11] P. Van Roy. Programming Paradigms for Dummies: What Every Programmer Should Know. In G. Assayag and A. Gerzso, editors, *New Computational Paradigms for Computer Music.* IRCAM/Delatour, France, 2009.

[12] P. Van Roy and S. Haridi. *Concepts, Techniques, and Models of Computer Programming*. The MIT Press, 1st edition, 2004.