**Notes while studying arbitrary software tools.**

**STUDY LOW LEVEL UNDERLYING CONCEPTS**

**(IF YOU ARE GIVEN 1 HOUR TO CUT DOWN A TREE, YOU SHOULD SPEND 45MINUTES SHARPENING YOUR AXE, USE 15 MINUTES TO CUT DOWN THE TREE)**

**YOU ARE A SOFTWARE TEST DEVELOPER**

**REPEAT an exercises 10 TIMES so you can master it**

**Tips: Study at Night (THEORY), Build (PRACTICAL) Programs or Video tutorials during the day**

**Base study = > Programming languages c# and java**

* **BASIC** **Concepts** and **Understanding** of => Computer Science and Programming, Software engineering and development.
* **CHECK Test Automation RoadMap videos on YouTube.**
* Try to understand the **underlying concept** of the software tool, program, code, SDLC (Software Development Lifecycle), Software Testing. Etc
* **Objects** are physical that you can touch and Logical that you cannot touch.
* See a Problem Statement of an **Object** as explaining the problem of the Object. Meaning, when you read a problem statement, see it like you reading the problem of an Object.
* **Problem Solving (understand the problem statements) and Algorithm (identify the inputs, Design the steps to solving the problem) and programming Implementation Skills (how search and to use the object and it members in programming to represent the input/data/value).**
* **After you have understood the problem, the first thing to identify in the problem is the WHAT input parameters (data/set of value) do we need to start solving the problem.**
* **Then, think of where, when and how to source the data from.**
* **INPUT => PROCESS => OUTPUT (Understand and identify all the input parameters in a problem**)
* PLAY VIDEO GAMES, THEY HELP TO SOLVE PUSSLES and for FOCUS AND helps PAY ATTENTION to details.
* Questions (what, when, why, will, would, should, how)?
* What kind of **INPUT** will **PROCESS** and produce or return the expected result or desired **OUTPUT**/outcome? e.g., write a program that return todays date.
* Google or search for tutorials how to use types, e.g DateTime program in C# OR java, look and learn other people’s program code, learn their program building logic, learn their code.
* AN **OBJECT** IN ABSTRACT VIEW CAN BE SOMETHING OR ANYTHING THAT HAS NO NAME, BUT WHEN YOU A LOGICAL NAME TO IT THAT IS USED TO IDENTIFY IT, THEN IT BECOMES A **TYPE**. THEN LEARN HOE THE TYPE BEHAVES. ( data and code)
* Type the code with your hand and focus to understand one specific word or text.
* Learn to develop – create programs – SOFTWARE DEVELOPMENT
* Build or Develop frameworks.
* Objects are created or represented either with values, or reference types and many various types.
* Study and know lots of Types.
* Object of value type, reference types, Json type, xml type etc.
* Theory is the foundation of practical
* Build High level logical learning.
* Use examples (USE CASE, USER STORY, SCENARIO, CIRCUMSTANCES) when you study because it helps to identify and understand the problem using scenarios. Which now helps you to know what Solution you can apply to solve the problem. when you study a solution, think of what kind of problem will it solve. (WHAT, WHERE, WHEN, WHILE, HOW, THEN).
* IMPLEMENT while reading OR studying each statement, word, sentence, of the book. (Theory and Practical: BUILD same time).
* Look at built-in FUNCTIONS and learn how to write functions.

Simple to Complex functions.

* **Functions** must be reusable, if a line of code is repeated more than once, create a function for it, if a step has sub-steps, create a function with all the sub-steps separately and call the function in the appropriate line of code in the base function.
* A function address/ performs a specific problem or task.
* The most two important things in a **Functions** are the return Type and it input parameters.
* Parser means compute or evaluate an expression, then the value returned will be converted (casting) to another Type.
* *7 days learning an arbitrary(random) tool e.g. 90 days of coding C#, Java*
* *File is the fundamental storage abstraction for storing data*
* High level advanced *logical Learning* skill
* *Check the conceptual underlying logical base/root implementations*
* Problem -> Algorithm -> Implementation in a chosen programming language (When trying to understand a Problem, pay care attention on each word and get to understand and define each word with it meaning)
* When learning a particular programming language, study the built-in standard libraries very well because all other child API’s are built on-top of the base programming language. e.g C#
* A value has type, and it must match the assignment instance on the left. The value must be compatible with a specific type. e.g. int x = 0.
* LEARN Cause and Effect principles and concepts.ee
* Types represent values e.g float floatVar = 3.2f.
* Values are created statically/explicitly or dynamically/implicitly
* Streams: value (human readable code) and data/bits (machine readable code 0’s, 1’s)
* Libraries, namespaces, packages => object modules (classes, arrays, delegates, abstract, interfaces).
* Study how COMPILER and INTERPRETER execute code. e.g debugging
* Test your program logic with a set of different test data, write defensive program.

Object of data type system

An object of reference or value data type

Primitive **Value Data Types:** Struct, int byte, double, float, decimal, enum

**Reference Data Type:** Class, Delegate, Abstracts, Interface, Arrays

* Every variable has a type, which makes the variable an instance of that type, a variable stores a value and can reference the type members (interfaces or operations).
* Processes, Threads, Concurrency
* Read-only, get, input, accessor, immutable
* Write-only, set, output, mutator, mutable
* configuration file, the publishing time, all this can be associated with a problem in staging environment.
* **Logical reasoning** can also use by ASSOCIATION and RELATIONSHIP technique, think about what is associated or related with the problem object or solution object. e.g in staging pipeline, the host server, the virtual machine, the ymal
* **Read a book at least 3 times, Practice the exercises are you study, they help you to understand better.**
* Data (type parameter, type value, type argument, type variable, object - type reference, type expression, type key, type element, type data, input parameter, element type)
* {object instance, object reference}

**DATA TYPE underlying concept – store, accepts or access data and computes or process data.**

**First : Determine what of Type it is.**

**Secondly: LEARN HOW TO CREATE - DECLARE AND DEFINE A TYPE or How to instantiate a datatype (object data type) syntax**

**(Implement) And how to access it type members, for example by static, instance. (Hint: look at the methods, they are either instance or static)**

**Note: all Data are data type**

**Data type parameter and data type argument must match or correspond with each other.**

**For example , string, int e.t.c are all data types.**

|  |  |
| --- | --- |
| **Data** | **Code** |
| **(They are DATA that the TYPE accepts and are used for computation)**  **So, what kind of TYPE is it (Instance or Static) and what kind of data do then TYPE ACCEPT.**  **state, Input, set of values, arguments, read-write, set or get, properties, variable, fields, parameters** | **((Compute the data) They can change the state / data anyhow the programmer wants)**  **Interfaces, methods, operations, behaviour** |

**Object Relationships**

* **Composition**
* **Aggregations**

Every **derived program** has a **base program**, find the base and study it, e.g. JUNIT, TESTNG, NUNIT base is Annotations (Java) and Attributes (C#)

1. **PROBLEM – Read and Analyse the problem, then gather the inputs, verbs, nouns, objects and understand the problem.**

**Write the problem - solution manually on piece of whitepaper.**

1. **ALGORITHM – Use a flowchart or UML or Pseudocode to design the algorithm steps and sub steps.**
2. **IMPLEMENTATION – Choose a programming language, search for program object types that perform the functionalities for the steps defined in the algorithm.**
3. **DESIGN – Apply design pattern techniques, data structuring, 4 OOP principles, object relationships etc.**
4. **CONSTRUCTION - Now start building.**

**Built – in (base object) objects can be used to create custom objects (derived object).**

|  |  |
| --- | --- |
| **Compiler** | **Runtime** |
| **Statically, explicitly, manually** | **Automatically, Implicitly, dynamically** |

**Software Tools & Technologies**

1. Nunit <https://nunit.org/>
2. Junit
3. TestNG

* **WHEN YOU LEARN SOMETHING, TRY TO LOGICALLY THINK OF WHAT COMPUTER COMPUTATIONAL PROBLEM CAN YOU APPLY THIS SOLUTION TO.**
* **Learn to write programs that does a specific and only one task and Build, Build and Build Programs.**
* **Build and debug the program, that how you will learn the program and how to program. Debug and Study line by line of the code. what happens line by line. How does the interpreter or compile, executes each line of code.**
* **Relative and Absolute XPATH (XML Path)**
* **BE INTENTIONAL AND LOGICALL WRITING A PROGRAM CODE, DO NOT WRITE A CODE HOPING IT WILL WORK. (Trial and Error style)**
* **Static object type** members belong or binds to the type and can be invoked directly by the type. E.g string.Compare(), int.Parse(), e.t.c
* **Instance object type** belongs to the object and can be referenced by an instance reference variable. E.g String name = “cool”; name.Contains(), name.Clone();
* when the Object is newly created, the Reference memory address is assigned to the instance reference variable which links or point to the object or copy the objects members to the instance reference variable.E.g Building build = new Building();
* **Learn to organize related objects in specific classes and other types and file systems.**
* **File Structuring and organization**
* **Object node hierarchy, tree and layers of object levels**
* **Find and learn how to approach a problem, how to solve a problem. How to approach a problem. Learn how to approach a problem.**
* Learn the syntax for all Value types and Reference types.

Learn File structuring Strategy Programming for DESIGN PATTERN

* Logically related information (files, data, code, operations, interfaces and so on) should be kept in same location. (Search on google = *file structure for computer programmers*)
* Common actions should be kept in same file location.
* Put all logical unrelated information in there right locations or create a new folder, directory or file and group hem all in same location.
* Cause and Effect theory.

**C# CHECKES and JAVA CHECKS**

* C# AND JAVA built-in types
* Keywords
* Assemblies
* Types Systems
* Namespaces and Package libraries. E.g System.Text, System.Reflection, java.util.List