Software Development Fundamentals (SDF)

Fluency in the process of software development is fundamental to the study of computer science. In order to use computers to solve problems most effectively, students must be competent at reading and writing programs in multiple programming languages. Beyond programming skills, however, they must be able to design and analyze algorithms, select appropriate data structures, and utilize modern development and testing tools.

The SDF knowledge area brings together fundamental concepts and skills related to software development, focusing on concepts and skills that should be mastered early in a computer science program, typically in the first year. The 43 hours of material in this knowledge area may be augmented with core material from other knowledge areas to form a complete and coherent experience as students progress to mid- and upper-level courses.

This knowledge area assumes a contemporary programming language with good built-in support for common data types including for associative data types like maps/dictionaries as the vehicle for introducing students to programming (e.g. Python, Java). However, this is not to discourage the use of older or lower-level languages for SDF - the knowledge units below can be suitably adapted for the actual language used.

SDF Knowledge Units

<table>
<thead>
<tr>
<th>Knowledge Units</th>
<th>CS Core</th>
<th>KA Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDF/Fundamental Programming Concepts</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>SDF/Fundamental Data Structures</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>SDF/Algorithms and Design</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>SDF/Development Methods</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

SDF/Fundamental Programming Concepts

[20 Core-Tier1 hours]
This knowledge unit aims to develop core programming concepts through one or more programming languages. It focuses on understanding of basic concepts (e.g., variables, types, expressions), and fluent use of basic constructs (e.g., assignments, conditionals, iteration) as well as modularity constructs (e.g., functions, classes/objects). It also aims to familiarize students with the concept of common libraries and frameworks, including those to facilitate API-based access to resources.

**Topics**

- Basic concepts such as variables, primitive data types, expression evaluation, assignment, etc.
- Basic constructs such as conditional and iterative structures and flow of control
- Key modularity constructs such as functions/methods and classes, and related concepts like parameter passing, scope, abstraction, data encapsulation, etc.
- Input and output using files, console, and APIs
- Structured data types available in the chosen programming language like sequences (arrays, lists, etc), associative containers (dictionaries, maps, etc), others (sets, tuples, etc.) and when and how to use them.
- Libraries and frameworks provided by the language (when/where applicable)
- Recursion
- Dealing with compile time and runtime errors
- Programming style that improves readability

**Illustrative Learning Outcomes**

1. Design, code, test, and debug a program that uses each of the following fundamental programming constructs: assignment and expressions, simple I/O, conditional and iterative structures, functions with parameter passing.
2. Design, code, test, and debug a program that effectively uses the different structured data types provided in the language like strings, arrays/lists, dictionaries, sets
3. Write a program that uses file I/O to provide persistence across multiple executions.
4. Write a program that uses APIs to get data (e.g. from the web, where applicable)
5. Write a program that uses some language-provided libraries and frameworks (where applicable).
6. Write a program that creates simple classes and instantiates objects of those classes (if supported by the language)
7. Explain the concept of recursion, and identify when and how to use it effectively
8. Write recursive functions
9. Trace the flow of control during the execution of a program.
10. Apply basic programming style guidelines to aid readability of programs such as comments, indentation, proper naming of variables, etc.

SDF/Fundamental Data Structures

[12 Core-Tier1 hours]

This unit aims to develop core concepts relating to Data Structures including associated algorithms. Students should understand the important data structures (often available in the programming language or as libraries) for modern applications, and how to use them effectively. This includes choosing appropriate data structures while designing solutions for a given problem.

Topics

- Standard abstract data types such as lists, stacks, queues, sets, and maps/dictionaries
- When and how to use standard data structures
- Strings and string processing
- Performance implications of choice of data structure(s)

Illustrative Learning Outcomes

- Write programs that use each of the abstract data types / data structures provided in the language like arrays, tuples/records/structs etc (depending on the language), lists, stacks, queues, and including associative data types like sets, maps / dictionaries.
- Select the appropriate data structure for a given problem.
- Write programs that work with text by using string processing capabilities provided by the language.
- Measure the performance of a program (e.g. to assess how performance changes with scale, to compare alternative implementations, …).
SDF/Algorithms and Design

[6 Core-Tier1 hours]

This unit aims to develop the foundations of efficient algorithms design and their analysis. The KU should also empower students in selecting and modifying suitable algorithms for building modest-complexity applications.

Topics

- Concept of algorithm and notion of algorithm efficiency
- Common algorithms like: Sorting, Searching, Tree traversal, Graph traversal, etc.
- Assessing the time/space efficiency of an algorithms through measurements

Illustrative Learning Outcomes

- Explain the importance of algorithms in the problem-solving process.
- Demonstrate how a problem may be solved by multiple algorithms, each with different properties.
- Describe common algorithms like: Sorting, Searching, Tree traversal, Graph traversal, etc.
- Experiment with space/time performance of some algorithms.

SDF/Development Methods

[5 Core-Tier1 hours]

This unit develops the core concepts relating to modern software development methods. Its aim is to develop student understanding and basic competencies in using modern methods and tools, including some general purpose IDE, source code control, use of debuggers, unit testing, etc. It should also empower students with the ability to discover solutions to the technical problems they face in their software development using suitable internet based resources.

Topics:

- Reading and understanding code
- Basic unit testing (using suitable frameworks) including test case design
- Basic source code control
● Use of a general purpose IDE, including its debugger (which can be also used to strengthen some programming concepts)

**Illustrative Learning Outcomes**

● Read and interpret code written by someone else

● Build, execute and debug programs using a modern IDE and associated tools such as visual debuggers.

● Use basic source code control tools and methods while writing code (potentially using facilities of the IDE).

● Use unit testing frameworks for developing tests for modules, and apply a variety of strategies to design test cases.

● Explain some limitations of testing programs

**Professional Dispositions for SDF**

Dispositions are: Individual qualities / tendencies of a person, relatively stable over time, that influence behavior and actions performed as part of an individual’s skill set.

Dispositions are not likely to be topics to be "taught", though their development can perhaps be facilitated through suitable exercises, practices, etc. Given that SDF is about "foundations", some desired dispositions can be:

● (Disposition of) Seeking out solutions to the issues they are facing using technical forums, FAQs, discussions, etc. If a student has this disposition, it will help their learning and developing strong SDF.

● (Disposition of) Hands-on/experimental approach. E.g. try to quickly implement an idea; try language features to understand them by writing small programs, go in the debugger to understand why a bug is occurring…. The temperament to try out to supplement thinking is surely beneficial in strengthening SDF.

● (Disposition of) Technical interest to better understand technology behind programming - e.g. how programs are executed and what happens underneath, what is happening in IDE/editor, what is the code being generated for execution for a python/java/… program, ...). If this tendency is there, it will clearly help students strengthen SDF.

● (Disposition of) Technical adaptability - to use newer tools and technologies that facilitate software development.

**Math Needed and Wanted**

As SDF focuses on the first year and is foundational, it assumes only basic math knowledge that students acquire in school.
Shared Concepts

As SDF focuses on the first year and is foundational, it cannot depend on other KAs (as they are likely to be taught in later semesters) to deliver its KUs and LOs.

SDF/Subcommittee

The subcommittee for SDF is:

- Pankaj Jalote, Chair, IIIT-Delhi, Delhi, India
- Brett A. Becker, University College Dublin, Ireland
- Titus Winters, Google, New York City, NY, USA
- Andrew Luxton-Reilly, University of Auckland, New Zealand
- Viraj Kumar, ACM India Education Committee, India
- Christian Servin, El Paso Community College, Texas, USA
- Karen Reid, University of Toronto, Canada
- Adrienne Decker, University at Buffalo, USA