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 select and use appropriate data structures and algorithms, 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. This includes fundamental programming concepts and their effective use in writing programs, use of fundamental data structures which may be provided by the programming language, basics of programming practices for writing good quality programs, and some understanding of the impact of algorithms on the performance of the programs. The 43 hours of material in this knowledge area may be augmented with core material from other knowledge areas 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 associative data types like dictionaries/maps 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. The main change from 2013 is a stronger emphasis on developing fundamental programming skills and effective use of in-built data structures (which many contemporary languages provide) for problem solving.

Changes since CS 2013:

<table>
<thead>
<tr>
<th>Knowledge Units</th>
<th>CS Core</th>
<th>KA Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fundamental Programming Concepts and Practices</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Fundamental Data Structures</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Algorithms</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Software Development Practices</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>43</strong></td>
<td></td>
</tr>
</tbody>
</table>
Knowledge Units

SDF/Fundamental Programming Concepts and Practices

[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 (and methods and classes, if supported in the language) 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 (e.g., arrays, lists), associative containers (e.g., dictionaries, maps), others (e.g., sets, tuples) and when and how to use them.
- Libraries and frameworks provided by the language (when/where applicable)
- Recursion
- Dealing with runtime errors in programs (exception handling)
- Basic concept of programming errors, testing, and debugging
- Reading and understanding code

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, maps, 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. Write a program that can handle a runtime exception.
10. Read and interpret code segments provided
11. Trace the flow of control during the execution of a program (both correct and incorrect).
12. Use appropriate terminology to identify elements of a program (e.g., identifier, operator, operand)

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 key abstract data types / data structures provided in the language (e.g., arrays, tuples/records/structs, lists, stacks, queues, and associative data types like sets, dictionaries/maps.
- 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, alternative data structures, …).

SDF/Algorithms

[6 Core-Tier1 hours]
This unit aims to develop the foundations of efficient algorithms and their analysis. The KU should also empower students in selecting 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 algorithms through measurement

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/Software Development Practices

[5 Core-Tier1 hours]

This unit develops the core concepts relating to modern software development practices. Its aim is to develop student understanding and basic competencies in using modern methods and tools, including some general purpose IDE, use of debuggers, testing, etc.

Topics:

- Basic testing (perhaps using suitable frameworks) including test case design
- Use of a general purpose IDE, including its debugger (which can be also used to strengthen some programming concepts)
- Programming style that improves readability

Illustrative Learning Outcomes

- Apply basic programming style guidelines to aid readability of programs such as comments, indentation, proper naming of variables, etc.
- Build, execute and debug programs using a modern IDE and associated tools such as visual debuggers.
- Develop tests for modules, and apply a variety of strategies to design test cases (perhaps using a testing framework).
- Explain some limitations of testing programs

Professional Dispositions
● Self-Directed. Seeking out solutions to issues on their own (e.g., using technical
forums, FAQs, discussions).
● Experimental. Practical experimentation characterized by experimenting with language
features to understand them, quickly prototyping approaches, using the debugger to
understand why a bug is occurring, etc.
● Technical curiosity. Characterized by, for example, interest in understanding how
programs are executed, what is happening in IDE/editor, how programs and data are
stored, etc.
● Technical adaptability. Characterized by willingness to learn about and use different
tools and technologies that facilitate software development.
● Perseverance. To continue efforts till, for example, a bug is identified, a program is
robust and handles most of the situations, etc.
● Systematic. Characterized by attention to detail and use of orderly processes in
practice.

Math Requirements

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

Shared Concepts and Crosscutting Themes

Shared Concepts:

- Software Engineering (SE). All the topics/LOs mentioned under the KA "Software
  Development Practices"
- Algorithms and Complexity (AL): All topics/LOs listed under the KA "Algorithms"

Course Packaging Suggestions

The SDF KA will generally be covered in introductory courses, often called CS1 and CS2. How
much of the SDF KA can be covered in CS1 and how much is to be left for CS2 is likely to
depend on the choice of programming language for CS1. For languages like Python or Java,
CS1 can cover all of the Programming Concepts and Development Methods KAs, and some of
the Data Structures KA. It is desirable that they be further strengthened in CS2. The topics
under algorithms KA and some topics under data structures KA can be covered in CS2. In case
CS1 uses a language with fewer in-built data structures, then much of the Data Structures KA
and some aspects of the programming KA may also need to be covered in CS2. With the former
approach, the introductory course in programming can include the following:
1. SDF/Fundamental Programming Concepts and Practices, 20 hours
2. SDF/Fundamental Data Structures, 12 hours
3. SDF/Algorithms, 2 hours (the remaining 4 hours should be covered in CS2 via AL)
4. SDF/Software Development Practices, 2 hours (the remaining 3 hrs should be covered via SE)
5. KUs from SEP Knowledge Area: 2 to 4 hours (exact KUs to be discussed with SEP)

Pre-requisites: School Maths (Sets, Relations, and Logic)

Skill statement: A student who completes this course should be able to:

- Design, code, test, and debug a modest sized program that effectively uses the functional abstraction.
- Select and use the appropriate language provided data structure for a given problem (like: arrays, tuples/records/structs, lists, stacks, queues, and associative data types like sets, dictionaries/maps.)
- Design, code, test, and debug a modest-sized object oriented program using classes and objects.
- Design, code, test, and debug a modest-sized program that uses language provided libraries and frameworks (including for getting data from the web through APIs)
- Read and interpret given code including tracing the flow of control during execution
- Build, execute and debug programs using a modern IDE and associated tools such as visual debuggers.
- Explain the key concepts relating to programming like parameter passing, recursion, runtime exceptions and exception handling.

**Competency Specifications**

Given that SDF is for entry level programming capability, the term program in these means a modest size program.

- **Task 1:** Given specifications, develop a program to implement it.
  - **Competency Statement:** Write code for a function/class or a small program.
  - **Competency area:** Software
  - **Competency unit:** Development, Testing
  - **Required knowledge areas and knowledge units:**
    - SDF/ Fundamental Programming Concepts
  - **Required skill level:** Develop
  - **Core level:** CS core

- **Task 2:** Write documentation for a program.
  - **Competency Statement:** Read and understand given code and explain it.
- **Competency area:** Software
- **Competency unit:** Documentation
- **Required knowledge areas and knowledge units:**
  - SDF/ Fundamental Programming Concepts
- **Required skill level:** Explain
- **Core level:** CS core

---

- **Task 3:** Develop test cases to determine if a program is functionally correct.
- **Competency Statement:** Develop test cases and test a given program.
- **Competency area:** Software
- **Competency unit:** Testing
- **Required knowledge areas and knowledge units:**
  - SDF/ Software Development Practices
- **Required skill level:** Develop
- **Core level:** CS core

---

- **Task 4:** Identify and fix a bug in a program.
- **Competency Statement:** Debug a program.
- **Competency area:** Software
- **Competency unit:** Development, Testing, Evaluation, Maintenance
- **Required knowledge areas and knowledge units:**
  - SDF/ Fundamental Programming Concepts, Software Development Practices
- **Required skill level:** Evaluate / Develop
- **Core level:** CS core

---

- **Task 5:** Perform a code review to evaluate the quality of code.
- **Competency Statement:** Read and understand the code and identify errors in it.
- **Competency area:** Software, Application
- **Competency unit:** Documentation, Evaluation, Improvement
- **Required knowledge areas and knowledge units:**
  - SDF/ Fundamental Programming Concepts, Software Development Practices
- **Required skill level:** Evaluate
- **Core level:** CS core

---

- **Task 6:** Explain a program at different levels of abstraction.
- **Competency Statement:** Read and understand code.
- **Competency area:** Software
- **Competency unit:** Documentation / Evaluation
- **Required knowledge areas and knowledge units:**
  - SDF/ Fundamental Programming Concepts
- **Required skill level:** Explain, Evaluate
- **Core level:** CS core

---

- **Task 7:** For a given programming problem, select the most appropriate data structure.
- **Competency Statement:** Use appropriate data structures to write code for solving a problem.
- **Competency area:** Software, Application
- **Competency unit:** Development, Evaluation
- **Required knowledge areas and knowledge units:**
  - SDF/ Fundamental Programming Concepts, Fundamental Data Structures
- **Required skill level:** Evaluate / Develop
- **Core level:** CS core

---

- **Task 8:** Develop a program that effectively leverages the capabilities of libraries and APIs.
- **Competency Statement:** Write a program using APIs and/or Libraries.
- **Competency area:** Software, Application
- **Competency unit:** Development
- **Required knowledge areas and knowledge units:**
  - SDF/ Fundamental Programming Concepts
- **Required skill level:** Develop
- **Core level:** CS core

---

**Committee**

**Chair:** Pankaj Jalote, Chair, IIIT-Delhi, Delhi, India

**Members:**

- Brett A. Becker, University College Dublin, 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, El Paso, TX, USA
- Karen Reid, University of Toronto, Toronto, Canada
- Adrienne Decker, University at Buffalo, Buffalo, USA
# Appendix: Core Topics and Skill Levels

<table>
<thead>
<tr>
<th>KA</th>
<th>KU</th>
<th>Topic</th>
<th>Skill</th>
<th>Core</th>
<th>Hours</th>
</tr>
</thead>
</table>
| SDF | Fundamental Programming Concepts | ● 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  
      |                      | ● Libraries and frameworks provided by the language (when/where applicable)  
      |                      | ● Recursion                                                      | Develop  | CS   | 18    |
| SDF | Fundamental Programming Concepts | ● Basic concept of programming errors, testing, and debuggingDealing with compile time and runtime errors  
      |                      | ● Reading and understanding code                                           | Evaluate, Apply | CS | 2     |
| SDF | Fundamental Data Structures | ● Standard abstract data types such as lists, stacks, queues, sets, and maps/dictionaries [Shared with: AL]  
      |                      | ● Strings and string processing                                               | Develop | CS | 10    |
| SDF | Fundamental Data Structures | ● When and how to use standard data structures  
      |                      | ● Performance implications of choice of data structure(s)               | Evaluate | CS | 2     |
| SDF | Algorithms            | ● Concept of algorithm and notion of algorithm efficiency  
<pre><code>  |                      | ● Common algorithms like: Sorting, Searching, Tree traversal, Graph traversal, etc. [Shared with AL] | Explain | CS | 4     |
</code></pre>
<table>
<thead>
<tr>
<th>SDF</th>
<th>Algorithms</th>
<th>Evaluating</th>
<th>CS</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDF</td>
<td>Software Development practices</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDF</td>
<td>• Assessing the time/space efficiency of algorithms through measurement [Shared with AL]</td>
<td>Evaluate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDF</td>
<td>• Programming style that improves readability [Shared with SE]</td>
<td>Evaluate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDF</td>
<td>• Basic unit testing (using suitable frameworks) including test case design [Shared with SE]</td>
<td>Develop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDF</td>
<td>• Use of a general purpose IDE, including its debugger (which can be also used to strengthen some programming concepts)</td>
<td>Apply</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>