Reviewer 1.

Feedback comment:
There were many editorial suggestions

How incorporated:
Most have been incorporated. Thanks to the reviewer for pointing these out.

Why not incorporated:

Date considered:

Feedback comment:

- How programs work: specifications and their implementations, state and state transitions, preconditions and postconditions, loop and data invariants, termination

How incorporated:

- How imperative programs work: state and state transitions on execution of statements, flow of control.

Why not incorporated:

Date considered:

Feedback comment:

1. Read, and interpret code segments provided, and explain why they implement their specifications.

How incorporated:

- Read a given program and explain what it does

Why not incorporated:

Date considered:

Feedback comment:

- Read and interpret explain why given code works, including adequacy of refinements, completeness of case analyses, and tracing the flow of control during execution
How incorporated:

- Read and explain given code including tracing the flow of control during execution

Why not incorporated:

Date considered:

Feedback comment:

- Read, and understand given code, and explain why it implements a given specification.

How incorporated:

- Read and understand given code and explain it.

Why not incorporated:

Date considered:

Feedback comment:

- Specifications and their implementations, state and state transitions, preconditions and postconditions, loop and data invariants, termination

How incorporated:

- How imperative programs work: state and state transitions on execution of statements, flow of control.

Why not incorporated:

Date considered:

Reviewer 2.

Feedback comment:

Editorial suggestions

How incorporated:

Mostly incorporated.

Why not incorporated:

Date considered:
Feedback comment:
Should exception handling be specified as it is not in some more traditional languages (e.g., C) that are still used.

How incorporated:
Topic is changed to: Dealing with runtime errors in programs (e.g., exception handling)

Why not incorporated:

Date considered:

Feedback comment:
These seem rather advanced given that neither trees nor graphs appear in any listing of data structures students are expected to learn (about Tree traversal and graph algorithms)

How incorporated:
The algorithms are now indicated as examples (which algorithms to discuss is left to the instructor)

Why not incorporated:

Date considered:

Feedback comment:
I am not sure what this parenthetical adds…. Suggests deletion.

How incorporated:
The parenthetical comment "(perhaps using some frameworks)" has been deleted.

Why not incorporated:

Date considered:

Reviewer 3.
Feedback comment:
Main suggestion is regarding including multi-file source code.

- Reading, writing and understanding multi-files code
- Write and execute a multiple source files program.
- ...

How incorporated:

Why not incorporated:

While desirable, we feel that developing multi-file code is not part of minimum necessary knowledge from SDF. Of course instructors who want to can easily incorporate it by taking a wider view of reading / writing from/to files.

Date considered:

Feedback comment:

How incorporated:

Why not incorporated:

Date considered:

Reviewer 4.

Feedback comment:

This introduction is all about writing programs, whereas in practice, reading and comprehension are at least as important

How incorporated:

"reading and understanding programs" has been added in the introduction.

Why not incorporated:
**Date considered:**

*Feedback comment:*

It is quite inconsistent what is deemed a concept and what is deemed a construct. Most notably, «assignment» appears as construct above and as concept here.

*How incorporated:*

It has been clarified that in concepts it is about "assignments changing the state of variables" and in constructs "assignment statements" is used along with conditional/iterative statements.

*Why not incorporated:*

**Date considered:**

*Feedback comment:*

Consoles are such an old concept that it is questionable whether this should be that prominent in a forward looking document.

*How incorporated:*

Console I/O has been moved to basic constructs.

*Why not incorporated:*

**Date considered:**

*Feedback comment:*

Do you mean ‘try-except’ blocks or the students’ ability to understand what a runtime error is and how to deal with it when it occurs?

*How incorporated:*

The topic has been changed to: Dealing with runtime errors in programs (e.g., exception handling)

*Why not incorporated:*

**Date considered:**
Feedback comment:
… Reading and understanding is not a «meta»-topic (such as writing)....

How incorporated:
We have deleted "Reading and explaining code" as a topic (as it is generally not a topic of teaching). However, reading and understanding code has been further strengthened in the LOs.

Why not incorporated:

Date considered:

Feedback comment:
Somewhere we use "Design, develop, test..." and somewhere we use "write"

How incorporated:
We have defined: In these the term "Develop" means "design, write, test and debug"
And then used Develop for the LOs

Why not incorporated:

Date considered:

Feedback comment:
Measure the performance how? Taking simple time measurement or a profiler or determine how much data it can process in a specific time...?....

How incorporated:
From some other comments also we feel that measuring is not suitable in SDF. It has been replaced with "explain/assess" the impact of data structures/algorithms on performance...

Why not incorporated:

Date considered:

Feedback comment:
This is extremely vague and open. What exactly is a common algorithm?

**How incorporated:**

The topics has been replaced with: Some common algorithms (e.g., like: Sorting, Searching, Tree traversal, Graph traversal)

It has been kept not prescriptive as we are leaving it to the instructor to select which algorithm are to be discussed (as this is SDF and not algorithm design, the intent is only to expose the students to algorithms and their role/impact.)

**Why not incorporated:**

---

**Feedback comment:**

- Explain the importance of algorithms in the problem-solving process. : Is vague

**How incorporated:**

Wording changed.

**Why not incorporated:**

---

**Feedback comment:**

Some other places where expression was not fully clear

**How incorporated:**

Have changed the wordings to make the topics/LOs clearer

**Why not incorporated:**

---

**Reviewer 5.**
Feedback comment:
Small changes and editorial changes. These have been incorporated.
Could the console input be moved to bulleted point 1?

How incorporated:
Console I/O has been moved to basic constructs (2nd bullet).

Why not incorporated:

Date considered:

Feedback comment:
On 1st LO: Could this be split and/or some of it moved to a later illustrative LO? It may be a little intimidating for a single LO, especially as it is the first for some novice educators.

How incorporated:
It has been split into two.
There is no ordering in LOs - they are numbered for reference purposes.

Why not incorporated:

Date considered:

Feedback comment:
Change ordering of API LO and Library LO
On API. Again sorry for being picky, what about sending data?

How incorporated:
There is no ordering in LOs - they are numbered for reference purposes.
The LO has been changed to: Develop programs that uses APIs to access or update data (e.g., from the web, where applicable)

Why not incorporated:

Date considered:

Feedback comment:
Some suggestions regarding ordering of LOs as they are numbered

**How incorporated:**
LOs are not be numbered. They are illustrative LOs.

**Why not incorporated:**

**Date considered:**

---

**Feedback comment:**
Should Strings and string processing be included in the fundamentals unit as they are often considered (rightly or wrongly) similar to lists in some languages (I am talking mostly about Python here)

**How incorporated:**

**Why not incorporated:**
We feel that they are best maintained within data structures, as some modern languages provide it as a data type with its own operations which have differences from lists. Also, strings are sometimes treated as immutable while lists are mutable (e.g. in Python)

**Date considered:**

---

**Feedback comment:**
On: measure the performance of the program based on the choice of data structure(s) used

**How incorporated:**
Due to other concerns, we have replaced the "measurement of performance impact" related LOs with "assess the performance impact"

**Why not incorporated:**

**Date considered:**

---

---

**Reviewer 6.**

**Feedback comment:**
• Algorithms to perform simple operations on these abstract data types (such as ‘pop’, ‘push’ in stacks).

How incorporated:
The LO regarding data types has been enhanced to
• Standard abstract data types such as lists, stacks, queues, sets, and maps/dictionaries, and operations on them.

Why not incorporated:

Date considered:

Feedback comment:
• Concept of time complexity of an algorithm and notion of algorithm efficiency, including the big Oh notation
• Demonstrate how a problem may be solved by multiple different algorithms, each with different properties that include simplicity, efficiency, or scalability.

How incorporated:
Notion of efficiency has been included. Simplicity is implied thought not specifically mentioned.

Why not incorporated:
Properties like scalability are not suitable for SDF level.

Date considered:

Feedback comment:
• add "Iteration" to the topic "Recursion"

How incorporated:

Why not incorporated:
Iteration is explicitly mentioned in an LO above, and hence not repeated here.

Date considered:

Feedback comment:
- Concept of time complexity of an algorithm and notion of algorithm efficiency, including the big Oh notation.

**How incorporated:**

**Why not incorporated:**

We feel that the big Oh notation is best left for the AL KU.

**Date considered:**

Feedback comment:

- Demonstrate how a problem may be solved by multiple different algorithms, each with different properties that include simplicity, efficiency, or scalability

**How incorporated:**

**Why not incorporated:**

We feel that in SDF notion of efficiency is adequate (particularly since the goal is to only include the minimum required.)

**Date considered:**

Reviewer 7.

Feedback comment:

Version control?

**How incorporated:**

**Why not incorporated:**

We had earlier a discussion on it, but we felt that it should be left out of SDF (it is not part of necessary minimum knowledge and most instructors do not cover it; also version control necessarily requires use of new tools, which is not desirable.)

**Date considered:**
Feedback comment:
I think it improves understanding to get the students to build their own implementations of data structures, where possible, before using the built in ones

How incorporated:

Why not incorporated:
SDF focus is on leveraging the data type capabilities provided by the language for developing good programs. Later (e.g. in algorithms) they can learn how the operations may be implemented.

Date considered:

Feedback comment:
I know this is just illustrative so might not be relevant here but should an LO be included for functional processing where the language supports it? As an example our main language is Java which added this functionality in Java 8 and which is now widely used in industry. Obviously this functionality was available in other languages (ignoring functional ones like Haskell) much earlier.

How incorporated:

Why not incorporated:
SDF is already quite loaded, and its focus is clearly on imperative programs. Adding a new programming approach, even if using the same programming language, is not feasible in many situations and is not part of the minimum expected knowledge from SDF. (Of course in situations where this is feasible, it is not prohibited as instructors can add additional concepts to the ones proposed.)

Date considered:

Reviewer - CDER

Feedback comment:
The main suggestion is about adding some exposure to distributed and parallel computing, e.g. by showing how some of the operations on some the data structures (e.g. iterative over sets) can be executed in parallel, that API calling is distributed processing/RPC, iterative structures don't always have strict ordering, that there are parallel equivalents of some algorithm, that libraries exist to facilitate parallel/distributed computing ....

How incorporated:
Why not incorporated:

The committee appreciates the sentiments, and agrees that it will be desirable if students can be taught these concepts. However, we felt that it should be left out of SDF. Some of the reasons for this are:

- Whatever is recommended in SDF is for the wide variety of institutions with varying capabilities
- Curriculum recommendations are what is the minimum acceptable for the knowledge area
- SDF is often taught for a wide variety of majors, not just CS - we should be cognizant of it
- Just hinting or introducing these rather difficult concepts (e.g. introduce distributed processing concepts while discussing APIs or parallel computing while discussing set data type) can actually add confusion without providing any understanding of distributed/parallel computing
- Teaching sequential programming is hard enough

Colleges which have the capability to introduce some of these concepts in SDF, of course, can do so.

Date considered:

----------------------------------------------------------------------------------------------------------------------------------