A Comparative Analysis of Prescribed Syllabus and Textbook Content of Higher Secondary Computer Science in COHSEM, Manipur

Ningthoujam Chetan Das
Wahengbam Jyotirmoy Singh

Abstract

The alignment between curricular objectives and textbook content is a foundation of effective education, particularly in academic education, where precision and comprehensiveness are most important in teaching and learning. The paper conducts an in-depth comparative analysis of the syllabus prescribed and the textbook for class 12th computer science by the Council of Higher Secondary Education (COHSEM), which focuses on the first two chapters: Exception and File Handling in Python and Stack. Through a very careful and detailed content analysis, this study identifies areas of orderly and consistent relation, inconsistencies, and offers insight into curriculum design and instructional strategies. The findings emphasize the necessity of continuous alignment reviews to ensure that educational resources have strong matches with the educational syllabus, textbook content, and technological demands of future needs.

Key words: Content Analysis, Curriculum, Python, Stack, Syllabus-Textbook Alignment.

Introduction

The teaching and learning of Computer Science at the higher secondary level demand a curriculum that is well-aligned with its corresponding textbook to ensure effective comprehension and skill development among learners. A thorough examination of alignment between the prescribed syllabus of the Council of Higher Secondary Education, Manipur

(COHSEM), and the content of the officially prescribed textbook is essential to maintain curricular integrity and instructional relevance.

This paper undertakes a comprehensive comparative analysis of the syllabus for Class 12 Computer Science, as prescribed by COHSEM, and the content of its prescribed textbook, focusing on the first two chapters: *Exception and File Handling in Python* and *Stack,* by breaking down the syllabus objectives against textbook content. The alignment between a prescribed syllabus and its corresponding textbook is a foundation of effective pedagogical delivery, shaping the depth, consistency, and applicability of knowledge. It adopts an evaluative framework, examining aspects such as *conceptual coverage*, *theoretical depth*, *alignment with syllabus and textbook*, and *content assessment gaps*. Further, the paper explores whether the textbook adequately bridges theoretical constructs (e.g., LIFO principles in stacks) with real-world applications as planned by COHSEM. The analysis is contextualized within larger educational directives, such as curriculum design effectiveness, resource effective use, and the cultivation of industry-relevant skills in an evolving scenario. The study aims to inform educators, policymakers, and textbook developers to promote improvements that enhance student preparedness for higher education and technological careers. The following investigates methodological thoroughness, sectional comparisons, and actionable recommendations, ultimately contributing to education quality in higher secondary schooling systems.

Finally, in the field of education, the important relationship between curricular guidelines, syllabus, and prescribed textbook content is necessary to have good education, considering the aim and objective of the course of education.

Objective of the Study:

The study is to

1. Compare the coverage, depth, alignment, and gaps between the prescribed syllabus and the content of the textbook of Computer Science prescribed by COHSEM.
2. Propose the feedback so as to get more aligned in the future

Research Question

The research questions of the study were as under:

1. Is teaching computer science at the higher secondary level aligned with the specific objectives that were designed in the curriculum?
2. Are the textbooks of computer science for XII aligned with the syllabus of COHSEM?

Literature Review

Curriculum-textbook alignment is a central theme in educational research, particularly in technical and STEM education domains. Alignment ensures that the educational materials, including textbooks, reflect the competencies and outcomes stated in the curriculum (Smith, 2020). **Williams (2022)** emphasizes the importance of pedagogical strategies that prioritize curriculum alignment with practical content in programming education. The author asserts that programming instruction must integrate real-world problem-solving exercises, such as exception handling and data structures, which are often inadequately covered in traditional textbooks.

Krippendorff (2018) provides a methodological foundation for analysing alignment through content analysis. His framework, applied in various educational evaluations, focuses on four essential dimensions: coverage, depth, alignment, and gaps. These dimensions are instrumental in identifying discrepancies between prescribed learning outcomes and instructional content.

Moreover, **Brown and Lee (2019)** explore the impact of misalignment in STEM subjects, revealing that inadequate representation of essential programming concepts like exception handling and file manipulation hinders students' ability to debug code and understand abstract programming paradigms. Their findings echo concerns highlighted by **Zhu (2020),** who argues that excluding advanced Python concepts such as user-defined exceptions limits students' capacity for writing modular, maintainable programs.

Another important contribution is from **Guo (2018),** who critiques Python textbooks for focusing excessively on syntax at the expense of application and logic-building skills. His findings suggest that many school-level textbooks fail to keep pace with the rapid advancement of Python versions and libraries. This gap contributes to a mismatch between curriculum expectations and textbook content, particularly when newer Python features (e.g., context managers, enhanced error handling) are

absent from learning materials. **Ericson (2021)** also points to the necessity of incorporating multimedia and hands-on learning tools to enhance textbook-based learning. She advocates for blended resources that reinforce curriculum content through visual programming tools and interactive exercises. This approach, if integrated with prescribed syllabi, could address some of the identified instructional gaps.

In the Indian context, **Gupta and Sharma (2021)** conducted a study on curriculum alignment in secondary computer science education and found significant misalignments, particularly in emerging areas such as data serialization (e.g., Pickle module) and exception handling. They emphasize that the curriculum often prescribes topics with minimal depth, leaving textbooks to over- or under-compensate.

Lastly, **Smith (2020)** conducted a cross-national analysis of STEM curriculum-textbook alignment and found that many educational boards, including those in developing countries, tend to have static syllabi that do not adapt to evolving technological landscapes. He recommends regular curriculum reviews and textbook updates to maintain alignment, particularly in programming education.

Methodology

This study employs a qualitative content analysis approach through documentary analysis to evaluate the alignment between COHSEM's syllabus and the prescribed textbook. Data sources include the official syllabus document and the Class XII Python textbook authorized by COHSEM. The analysis focused on the first two chapters: *Exception and File Handling in Python* (Chapter 1) and *Stack Implementation* (Chapter 2).

The methodology was structured around four analytical categories derived from Krippendorff's (2018) framework:

1. Coverage: Whether syllabus-mandated topics are included in the textbook.
2. Depth: The extent of elaboration (e.g., subtopics, code examples).
3. Alignment: Direct correspondence between syllabus subtopics and textbook sections.
4. Gaps: Syllabus topics are absent or more in the textbook.

Comparative Analysis of Syllabus and Textbook Content

The alignment between educational syllabi and prescribed textbooks plays a pivotal role in ensuring interrelated and effective learning outcomes. This paper examines the relationship between the first two syllabus chapters—Exception and File Handling in Python (Table 1) and Stack (Table 2)—and their corresponding textbook chapters, focusing on four critical dimensions: area coverage, depth of content, alignment, and identified gaps. Through this analysis, key insights emerge about the strengths and limitations of both resources, offering actionable recommendations for bridging inconsistencies.

Results and Analysis

Table 1: Comparative analysis of Chapter 1 of the prescribed syllabus and content of the prescribed textbook of COHSEM

Topic/Subtopic in the prescribed Syllabus	Textbook Content in the prescribed textbook
Chapter 1: Exception and file Handling in Python	Chapter 1: Exception Handling in Python
Exception Handling	1.1 Introduction
- Syntax Errors	1.2 Syntax Errors
-Exception	1.3 Exception
- Need for Exception Handling	1.4 Built-in Exceptions
- User-Defined Exceptions - Raising Exceptions	1.5 Raising Exceptions 1.5.1 The raise Statement 1.5.2 The assert Statement
- Handling Exceptions - Catching Exceptions - Try-Except-Else Clause	1.6 Handling Exceptions 1.6.1 Need for Exception Handling 1.6.2 Process of Handling Exception 1.6.3 Catching Exceptions 1.6.4 try...except...else Clause
- Try-Finally Clause	1.7 Finally Clause
- Recovering and Continuing With Finally	1.7.1 Recovering And Continuing With Finally Clause
- Built-in Exceptions	

File Handling	Chapter 2: File Handling in Python
	2.1 Introduction To Files
- Text file and Binary Files	2.2.1 Text File 2.2.2 Binary Files
- File type	2.2 Types of Files
- Open and Close files	2.3 Opening And Closing a Text File 2.3.1 Opening a File 2.3.2 Closing a File 2.3.3 Opening a File Using With Clause
- Reading and Writing Text Files	2.4 Writing to a Text File 2.4.1 The write( ) method 2.4.2 The writelines( ) method 2.5 Reading From A Text File 2.5.1 The read( ) method 2.5.2 The readline ([n]) method 2.5.3 The readlines ( ) method
- Reading and Writing Binary Files using Pickle module	2.8 The Pickle Module 2.8.1` The dump ( ) method 2.8.2 The load ( ) method 2.8.3 File handling using pickle module
- File Access Modes	

The observation of Chapter 1, Exception and File Handling in Python are:

Area Coverage

The syllabus for Exception and File Handling consolidates two broad topics—exception handling and file operations—into a single chapter. It introduces foundational concepts such as syntax errors, exception, user-defined exceptions, and file input/output operations, including the use of the Pickle module. In contrast, the textbook dedicates separate chapters to these topics, allowing for a more expansive exploration. For instance, the textbook's Exception Handling chapter goes into refined subtopics like the raise and assert statements, while the File Handling chapter elaborates on file modes (e.g., with clause) and distinct methods for reading/writing

text and binary files. This bifurcation enables the textbook to cover a wider range of subtopics, such as the mechanics of the dump() and load() methods in Pickle, which are only lightly mentioned in the syllabus.

Depth of Content:

A notable distinction lies in the depth of content. The syllabus for **Exception and File Handling** often lists topics at a conceptual level (e.g., "User-Defined Exceptions") without elaborating on implementation details. Conversely, the textbook adopts a procedural approach, breaking down processes like raising exceptions into (e.g., 1.5.1 The raising statement; 1.5.2 The assert statement) also exception handling into discrete steps (e.g., "1.6.1 Need for Exception Handling;1.6.2 Process of Handling Exception; 1.6.3 Catching Exceptions") and distinguishing between methods such as `read()`, `readline()`, and `readlines()`. This quality of composition trains learners with actionable knowledge but highlights a gap in the syllabus, which omits critical details like the `assert` statement. Also, the file access modes which was present in the syllabus are absent in the textbook, creating loopholes in the syllabus and the textbook.

Alignment:

Alignment between the syllabus and textbook is strongest in areas where topics directly overlap. For **Exception Handling**, subtopics like syntax errors, built-in exceptions, and `try-finally` clauses are well-matched. Similarly, the syllabus's coverage of file handling aligns with textbook sections on text/binary files and the Pickle module. However, inconsistencies emerge in areas such as "File Access Modes," a syllabus topic with no clear textbook counterpart, and the textbook's detailed subsection on "Recovering and Continuing with Finally," which is only implicitly referenced in the syllabus.

Identified Gaps:

The analysis reveals gaps in both resources. For **Exception and File Handling**, the syllabus omits the `assert` statement and lacks clarity on file access modes, while the textbook neglects to explicitly categorize "User-Defined Exceptions" as a standalone subtopic.

Table 2: Comparative analysis of Chapter 2 of the prescribed syllabus and content of the prescribed textbook of COHSEM.

Topic/Subtopic of the prescribed Syllabus Chapter 2, Stack	Textbook Content of Chapter 3, Stack
Introduction to Stack (LIFO Operations)	3.1 Introduction
	3.2 Stack 3.2.1 Application of Stack
Operations on Stack - PUSH and POP	3.3 Operations on Stack 3.3.1 PUSH and POP Operation
Implementation in Python	3.4 Implementation of Stack in Python
Expressions in Prefix, Infix, Postfix notations	3.5 Notations for Arithmetic Expressions
Evaluating arithmetic expressions using stack	3.7 Evaluation of Postfix Expression
Conversion of Infix expression to Postfix expression	3.6 Conversion From INFIX to POSTFIX Notation

The observation of Chapter 2, Stack Implementation are as follow

Area Cover:

Similarly, for **Stack**, the syllabus outlines core concepts like LIFO operations, Python-based implementations, and arithmetic expression conversions. However, the textbook extends its coverage to include practical applications of stacks, such as evaluating postfix expressions, which are absent in the syllabus. While both resources address fundamental operations (e.g., PUSH/POP), the textbook's inclusion of real-world use cases enriches the learner's contextual understanding.

Depth of Content:

For **Stack**, the syllabus adequately explains operations and implementations but lacks depth in illustrating how stacks are applied in computational problems. The textbook compensates for this by dedicating sections to stack applications (e.g., expression evaluation) and conversion algorithms (e.g., infix to postfix), thereby fostering a deeper, application-oriented understanding.

Alignment:
In **Stack**, alignment is evident in discussions of LIFO operations and implementation in Python. Yet, the syllabus oversees the textbook's emphasis on stack applications (e.g., "Application of Stack" in 3.2.1), creating a disconnect between theoretical and practical learning. Additionally, a formatting inconsistency in Table 2 and some missing topics in the textbook compared with the syllabus show there is misalignment, suggesting oversight in syllabus design.

Identified Gaps:
For **Stack**, the syllabus's exclusion of stack applications limits learners' ability to contextualize concepts, whereas the textbook's use of alternative terminology (e.g., "Notations for Arithmetic Expressions" instead of "Prefix/Infix/Postfix") may cause confusion to the learner which give gaps in the syllabus and textbook.

Finding

The findings reveal an inconsistent outlook. While the textbook excels in clarifying syntax and basic operations, it hesitates in addressing advanced topics and contextual applications. The finding also contrasts with broader trends in programming education, where textbooks often prioritize rote syntax mastery over problem-solving and critical thinking (Guo, 2018). For instance, the absence of *User-Defined Exceptions* continues a fragmented understanding of error handling, limiting students' ability to design software, and it gives a gap between the syllabus and the textbook.

To reduce these gaps, educators must adopt supplementary strategies. Integrating open-source resources like Python's official documentation or interactive platforms such as Codecademy could bridge knowledge voids (Resnick et al., 2009). Furthermore, the study found that curriculum designers must advocate for textbook revisions that incorporate emerging topics (e.g., context managers for file handling) and pedagogical innovations like case studies also it limits with its narrow focus on the first two chapters and exclusion of pedagogical methods like assessments—invite future research where longitudinal studies and tracking student performance against curriculum-textbook alignment could yield actionable insights.

Feedback on Improving Syllabus-Textbook Alignment

Ensuring close alignment between the prescribed syllabus and textbook content is essential for effective curriculum delivery. To achieve better alignment in future textbook editions, several focused strategies can be adopted. First, it is crucial to begin with a detailed topic-by-topic mapping that directly matches each syllabus subtopic with planned textbook content. This step can prevent common oversights, such as the omission of *User-Defined Exceptions* and *File Access Modes*, which were found missing in the current edition.

Textbook headings should closely reflect the terminology used in the syllabus. This allows students and teachers to easily trace and connect the content to curriculum expectations. Additionally, including a "Syllabus Link" table at the beginning of each chapter—outlining the prescribed topics, corresponding textbook sections, and learning outcomes—can significantly enhance clarity and usability. Textbooks should also ensure that they address all levels of cognitive demand outlined in the syllabus. Beyond basic definitions, students need examples, applications, and opportunities for analysis and problem-solving to build a comprehensive understanding. Teacher involvement is another key area. Engaging classroom educators in the review process brings practical insights that can improve content relevance and pedagogical alignment. Moreover, adding brief "Syllabus Expectation" summaries at the start of each section helps maintain instructional focus. Lastly, establishing a regular review and update process based on classroom feedback and syllabus changes will ensure the textbook remains current and aligned over time.

In summary, aligning textbooks more closely with the syllabus requires intentional planning, clear structure, practical feedback loops, and ongoing revision. These steps will make textbooks more effective tools for both teaching and learning.

Conclusion

The comparative analysis of the two chapters—*Exception and File Handling* and *Stack*— reveals a disorder and confused ordering of the syllabus, showing a gap in alignment between the prescribed syllabus and the textbook content. The chapter on *Stack* shows partial misalignment, with all syllabus topics covered and content structured in the textbook. In addition, the chapter

on *Exception and File Handling* also exhibits partial alignment. While most topics are addressed in sufficient depth, key subtopics such as *User-Defined Exceptions* and *File Access Modes* are missing, leading to content gaps. Overall, the study highlights the need for more consistent and comprehensive syllabus coverage, particularly in the Exception and File Handling chapter, to ensure students receive complete and accurate instruction.

References

Brown, A., & Lee, J. (2019). *Programming Education: Challenges and Innovations*. Springer.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

Ericson, B. (2021). *Introduction to Computing and Programming in Python: A Multimedia Approach*. Pearson.

Guo, P. J. (2018). *Python for Data Analysis: A Critical Toolkit*. O'Reilly Media.

Gupta, R., & Sharma, P. (2021). Curriculum alignment in Indian computer science textbooks: A case study. *International Journal of Educational Technology*, 8(1), 44–58.

Krippendorff, K. (2018). *Content Analysis: An Introduction to Its Methodology* (4th ed.). Sage.

Laaksonen, A. (2017). *Guide to Competitive Programming: Learning and Improving Algorithms Through Contests*. Springer.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., ... & Kafai, Y. (2009). *Scratch: Programming for All*. Communications of the ACM, 52(11), 60–67.

Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley.

Smith, T. (2020). *Curriculum-Textbook Alignment in STEM Education*. Journal of Educational Research, 45(3), 112–129.

Williams, J. (2022). *Pedagogical Strategies in Python Education: A Global Perspective*. International Journal of Computer Science Education, 17(2), 89–104.

Zhu, H. (2020). *Advanced Python Programming: Building Robust Applications*. Packt Publishing.