Educational Administration: Theory and Practice
2023, 29(02), 949-961
ISSN: 2148-2403
https://kuey.net/

Research Article

Analyzing Student's Perception of Attributes in Programming Assignment

Amitkumar Patel[1*] and Dr. Hardik J. Joshi[2]

[1*]Department of Computer Science, Gujarat University, Ahmedabad, Gujarat, India
[2] Department of Computer Science, Gujarat University, Ahmedabad, Gujarat, India hardikjoshi@gujaratuniversity.ac.in

***Corresponding author(s).** E-mail(s): patelamit@gujaratuniversity.ac.in ; ORCiD: 0000-0001-6607-1870

Citation: Amitkumar Patel, et al. (2023). Analyzing Student's Perception of Attributes in Programming Assignment, *Educational Administration: Theory and Practice*, 29(02) 949-961
Doi: 10.53555/kuey.v29i2.9999

ARTICLE INFO

Submitted: 27/March/2023
Reviewed: 10/May/2023
Accepted: 26/May/2023
Published: 19/June/2023

ABSTRACT

The increasing complexity of programming education necessitates a deeper understanding of student challenges and preferences to enhance learning outcomes. This study addresses this critical gap by exploring students' perceptions of programming assignments using advanced computational methods. This research investigated three key questions: the major challenges students face in programming, their preferred methods for skill assessment, and the skills they associate with effective programmers. The methodology involved a survey of 508 participants, including Bachelors of Computer Application (BCA), Master in Computer Application (MCA), and alumni cohorts, utilizing mixed-format questions. Natural language processing techniques, including Term Frequency-Inverse Document Frequency (TF-IDF) with K-means clustering and VADER sentiment analysis, were employed to analyze the responses. Key findings revealed five distinct clusters of challenges with prevailing logic and syntax-related difficulties, a strong preference for error-explanation-based assessments over traditional methods, and prioritization of problem-solving and analytical thinking skills, accompanied by a neutral sentiment toward programmer attributes. These insights highlight the diverse obstacles that students encounter and the skills they value. These implications are significant for educational practice, suggesting tailored teaching strategies to address identified challenges and redesigning grading systems to incorporate interactive feedback mechanisms. This study contributes to the field by demonstrating the efficacy of natural language processing (NLP) and machine learning (ML) in educational data mining and offering a scalable approach to curriculum development. In conclusion, these findings lay the foundation for data-driven improvements in programming education with the potential to shape future pedagogical innovations and technology-enhanced learning environments.

Keywords: Programming education, Natural Language Processing, Student perceptions, Programming challenges, Assessment methods, Educational data mining

1. Introduction

Programming education is vital for equipping students with the technical competencies required in the software development and computational fields, where coding skills are increasingly foundational. As class sizes grow, the traditional manual grading of programming assignments struggles to provide timely and consistent feedback, prompting a shift toward automated solutions [1]. While these systems efficiently evaluate code correctness, they often neglect behavioral attributes, such as engagement, collaboration, and debugging proficiency, which shape students' learning trajectories [2]. Such attributes, observable through interactions in Moodle and various learning management systems (LMS), offer critical insights into the coding process; however, their integration into assessment frameworks remains underexplored. This study leveraged students' perceptions to address this shortfall, enhancing programming pedagogy by aligning assessment with learner needs, and potentially improving skill acquisition and retention.

Conducted with 508 undergraduate and postgraduate students, the survey used a mixed-method approach with keyword extraction and sentiment analysis to examine perceptions of programming challenges, assessment preferences, and essential coder abilities. As an initial step toward modeling student behavior in programming assignments, this study informs the design of an autograding system that evaluates both code correctness and behavioral metrics, such as participation and debugging skill. By capturing students' perspectives, it bridges the conventional grading limitations with a more comprehensive assessment paradigm. Programming education has evolved from theoretical lectures to practice-driven curricula, reflecting the discipline's hands-on nature. Early assessments relied on manual code reviews, which, although detailed, became impractical with rising enrollments [3]. The emergence of automated grading tools in the 2010s, such as CodeGrade, introduced scalability, focusing on functional output evaluation [2]. In recent years, machine learning has enhanced the precision of these systems. However, their scope remains largely syntactic. Concurrently, research has highlighted the role of behavioral factors in coding mastery. Engagement, as evidenced by consistent coding efforts, supports conceptual understanding [4] , while collaboration via platforms such as GitHub fosters peer learning [5]. Debugging proficiency, a hallmark of skilled coders, is often underassessed [6]. These trends underscore the need to explore students' perceptions of grade innovations that capture both the technical and process-oriented dimensions.

This study investigated students' perceptions of programming assignments, focusing on their reported challenges, preferred assessment methods, and valued coders' abilities. Using a survey of 508 undergraduate and postgraduate students, keyword extraction and sentiment analysis were conducted to identify response patterns. While exploratory and survey-based, it establishes a foundation for modeling behaviors, such as engagement, participation, debugging, and collaboration, providing qualitative insights to guide autograding system design aligned with educational priorities.

The primary objective is to analyze students' perceptions through survey data to inform a behavior-aware autograding system. Specific objectives include:

- To identify the significant challenges students encounter in programming and to illuminate the factors influencing engagement and debugging.
- To determine the students' preferred methods for assessing coding knowledge, user-centric automated grading development was supported.
- To examine the abilities students associate with effective coders, guiding the integration of collaboration and debugging metrics into grading systems.

Recent advancements in automated grading have improved syntactic evaluation, with studies such as [2] demonstrating the efficacy of machine learning in assessing code correctness. A Python grading tool that achieved high accuracy, but focused solely on the output [7]. The authors in [1] Analyzed engagement via submission patterns, but their findings stopped short of grading integration. The role of collaboration in learning has been explored through GitHub interactions [8], although quantitative assessments have remained elusive. The authors in [9] advanced a debugging assessment with semantic analysis; however, behavioral integration into autograding is rare. Perception-based studies have examined students' attitudes toward programming [10]; however, they lack a focus on linking these perceptions to behavioral modeling for assessment. From 2015 to 2022, no study has systematically connected students' perceptions of challenges, assessment preferences, and coder abilities to behavioral metrics for autograding. Perceptions shape coding behaviors, such as effort or teamwork, yet their role in informing grading systems remains underexplored a gap this study addresses with qualitative insights. To address this gap, this study poses the following research question:

- (RQ1) What are the biggest problems that students face when learning code?
- (RQ2) What ways do students think they are best at checking their coding knowledge?
- (RQ3) What abilities do students think are important to the coders?

The remainder of this paper is organize as follows. Section 2 reviews related work, synthesizing recent advancements in automated grading, behavioral analytics in programming education, and perception-based studies to contextualize the research gap. Section 3 details the methodology and describes the survey design with 508 undergraduate and postgraduate students, the five descriptive questions targeting challenges, assessment preferences, and coder abilities, and the application of keyword extraction and sentiment analysis to process responses. Section 4 presents the results, analyzing patterns in students' reported challenges (RQ1), preferred assessment methods (RQ2), and valued coder abilities (RQ5), with findings derived from keyword frequencies and sentiment trends. Section 5 discusses the implications and limitations of these findings, including the exploratory scope of the survey. Finally, Section 6 concludes the study, summarizes key insights, and outlines future research directions, including the development of a behavior-aware autograding system and longitudinal validation of perception-driven interventions in programming education.

2. Literature Review

This literature review synthesizes research from 2015 to 2023, drawing on peer-reviewed articles from Scopus, Springer, ACM, IEEE, and Elsevier-indexed journals, to contextualize the analysis of 508 student responses from the Bachelor of Computer Applications (BCA), Master of Computer Applications (MCA), and Alumni

cohorts. The review is organized thematically to address common challenges in programming education, assessment methodologies, attributes of effective programmers, and identified research gaps, providing a foundation for the current investigation into programming difficulties, assessment preferences, and perceived skills using techniques such as TF-IDF, K-means clustering, frequency analysis, and VADER sentiment scoring.

2.1 Challenges in Programming Education

Programming education is fraught with challenges that hinder student success, with syntax comprehension, logical reasoning, and debugging emerging as predominant barriers. Lister (2016) conducted a qualitative analysis of novice programmers and identified syntax comprehension, logical reasoning, and debugging as persistent barriers, noting that such difficulties often arise from limited practical exposure. In a multi-institutional study published in ACM SIGCSE Bulletin, Study [11] found that introductory students struggle with algorithmic thinking across diverse settings. The Authors in [12] further explored cognitive barriers among Danish computer science students, emphasizing that abstraction and problem decomposition remain significant hurdles, particularly for undergraduate students. As per the Study [13], authors highlighted weak abstraction skills as a common issue, reinforcing the need for tailored interventions. These findings suggest that programming difficulties vary by experience level, underscoring the value of capturing student-reported challenges from diverse cohorts, including alumni, to inform tailored educational interventions, the focus of the current study.

2.2 Assessment Methods in Programming

The evolution of assessment methods for programming proficiency relevant to RQ2 has been a key focus in educational technology research between 2015 and 2023. Study [13] reviewed automated assessment tools highlighting their ability to provide immediate feedback through practical quizzes despite noting scalability limitations. The subjectivity of viva voce assessment was critiqued for subjectivity [14], advocating for structured tasks such as error identification, which aligns with the “Collect Response” option in this study. A study on global assessment preferences in Computer Science Education, suggesting a mix of methods to accommodate diverse learning styles yet noted a lack of cohort-specific data [15]. These studies indicate a diversity of approaches, revealing a gap in understanding student-preferred methods across BCA, MCA, and alumni groups, which this research addresses through its mixed-format survey design.

2.3 Attributes and Skills of Effective Programmers

Studies conducted from 2015 to 2023 examined the perceived skills and characteristics of successful software and web developers, resonating with RQ3. A surveyed on 250 students and professionals in ACM Inroads and identified problem solving, programming proficiency, and teamwork as critical competencies, with a growing emphasis on communicati*on* skills in agile environments [16]. Another study explored novice perceptions in Education and Information Technologies, noting that analytical thinking and debugging are highly valued, but often-underdeveloped [6]. The authors in [3] investigated skill gaps in curricula, highlighting the underrepresentation of version control and collaboration skills. This gap is particularly evident when considering alumni insights, which this study incorporates to bridge the academic and industry perspectives.

2.4 Research Gaps

Despite these contributions, significant gaps in the literature remain. Lister [17] and Simon et al. [18] identified programming difficulties, but did not propose interventions based on student feedback, limiting practical application. Assessment research in [19] focuses on instructor or system perspectives, overlooking student preferences that could enhance engagement, which is a focus of RQ2. Similarly, Fincher and Robins [16] and Joan O Vicente et al. [20] explored skill perceptions across diverse academic stages without linking them to curriculum design. Moreover, in their 2017 study, Wang et al. [21] noted challenges with low silhouette scores in the K-means clustering of educational texts, a concern addressed in this study by optimizing the cluster numbers to k=3, as informed by preliminary analyses. The inclusion of the BCA, MCA, and alumni cohorts mitigates the sample diversity limitation noted by Reinke et al. in their study [22], whereas Lweis [23] acknowledge ethical considerations by the anonymous survey design. By applying NLP and ML techniques to a mixed-format dataset, this study offers a novel framework for aligning student perceptions with pedagogical improvements and bridging these deficiencies.

The literature underscores the complex challenges in programming education, evolving landscape of assessment practices, and diverse skill sets defining effective programmers, with NLP and ML techniques offering robust analytical tools. Identified gaps, such as limited sample diversity, ethical considerations in anonymous surveys, and underutilized qualitative insights, shape the current study’s approach. By leveraging a mixed-methods analysis of 508 responses from BCA, MCA, and alumni cohorts using TF-IDF, K-means clustering, frequency analysis, and VADER sentiment scoring, this research addresses these shortcomings, providing a nuanced perspective on student perceptions. The following sections outline the methodology, present the findings, and discuss the implications of building on this foundation to advance programming education research.

3. Methodology

This study adopted a mixed-methods approach to explore student perceptions of programming assignments, integrating qualitative and quantitative analyses to address three research questions (RQs): RQ1, identifying areas of confusion in programming concepts; RQ2, determining preferred assessment methods; and RQ3, examining essential developmental skills. The methodology encompassed participant recruitment, survey design, data collection, pre-processing, and advanced NLP/ML techniques. A schematic overview of the research workflow is presented in Figure 1.

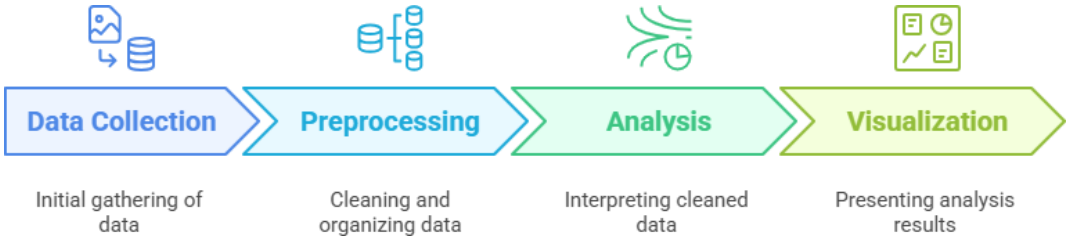

Figure 1: Basic Methodology flow

3.1 Participants

This study involved 508 participants recruited from a single Indian higher education institution that offered computer science programs. The sample consisted of three cohorts: Bachelor of Computer Applications (BCA), Master of Computer Applications (MCA), and alumni of these programs. As presented in Table 1, the BCA cohort comprised 209 students (103 males, 106 females), the MCA cohort included 174 students (80 males, 94 females), and the alumni group consisted of 125 graduates (66 males, 59 females). The overall gender distribution was approximately 49.4% male and 50.6% female, indicating a near-balanced representation across cohorts, which aligns with the increasing gender diversity in computing education (Sharma & Sharma, 2019). Class-level data were not collected because the focus was on cohort- and gender-based perceptions of the programming assignments. This demographic composition facilitated comparative analyses across undergraduate, postgraduate, and post-graduate stages.

3.2 Ethical Consideration

Informed consent was not explicitly obtained owing to the anonymous nature of the survey and institutional approval for noninvasive data collection. This limitation is acknowledged in the Discussion section as it may impact the generalizability of the findings.

Table 1: Participant Demographics

Cohort	Sample Size	Gender	
		Male	Female
BCA	209	103	106
MCA	174	80	94
Alumina	125	66	59
Total	508	249	259

3.3 Survey Design

To ensure a balance between quantitative comparability and qualitative depth, we designed a mixed-format survey to elicit both structured and open-ended responses. The survey comprised three core research questions aligned with the study objectives.

- **RQ1:** "In which specific areas of programming concepts do you find yourself CONFUSED or encounter the GREATEST DIFFICULTIES?" (open-ended). This question aimed to identify precise challenges (e.g., syntax errors and debugging) to inform targeted educational interventions.
- **RQ2:** "Which of the following is the MOST EFFECTIVE way to measure student programming concept understanding?" (multiple-choice with options: 1) Collect Response on Program to Find and Explain Error, 2) Practical Quiz [Written and Lab], and 3) Viva). The purpose was to assess the preferred method for evaluating conceptual grasp and addressing issues, such as copying in practical quizzes or memorization in vivas, with the first option designed to verify the true understanding of program flow and execution.
- **RQ3:** "What are the ESSENTIAL CHARACTERISTICS, SKILLS, and ABILITIES that a SOFTWARE or WEB DEVELOPER requires?" (open-ended). This question sought to elucidate the skills (e.g., problem solving and collaboration) and characteristics deemed necessary for developer success, linking perceptions to professional competencies.

The survey was administered online using Google Forms over a three-week period, in March 2022. Open-ended questions allowed participants to express nuanced perspectives, whereas multiple-choice questions facilitated quantitative comparisons across the cohorts. The survey underwent pilot testing with 15 students to ensure clarity and relevance, with minor revisions to the question phrasing based on feedback.

3.3 Data Collection

Data collection occurred over a two-week period in early 2022. Students were invited via email to voluntarily participate and to participate anonymously to encourage candid responses. A total of 508 valid responses were received, each containing answers to all five questions, yielding approximately. The responses were exported as CSV files for processing to ensure data integrity and traceability.

The data processing and analysis workflow illustrated in Figure 1 guided the transformation of the raw survey responses into actionable insights. Python (version 3.9) was used in the Jupyter Notebook environment with libraries including NLTK, scikit-learn, pandas, and VADER. The workflow comprises of four phases: data collection, preprocessing, analysis, and visualization.

3.4 Data Preprocessing

Open-ended responses for RQ1 and RQ3 underwent preprocessing to ensure consistency in the NLP analysis, following the structured pipeline depicted in Figure 2, which was developed as an original representation of the study's preprocessing workflow. The pipeline included:

1. **Tokenization**: Responses were segmented into words using the NLTK word tokenization function (Bird et al., 2009).
2. **Contraction Handling**: Contractions (e.g., "can't") were expanded (e.g., "cannot") using a custom Python dictionary to standardize text.
3. **Stopwords Removal**: Common English stopwords (e.g., "and," "the") were removed via NLTK's stopwords list to emphasize meaningful terms.
4. **Lemmatization**: Words were normalized to their base forms (e.g., "debugging" to "debug") using NLTK's WordNetLemmatizer to reduce variation.

Multiple-choice responses (RQ2) were encoded as categorical variables using pandas, excluding invalid or incomplete submissions, which yielded 508 valid responses. Data integrity was verified by duplicate and manual inspections. Pseudocode 1 outlines the pre-processing steps.

Pseudocode 1: Pseudocode for Data Preprocessing

```
Input: raw_responses (list of survey texts)
Output: cleaned_tokens (list of processed tokens)
for response in raw_responses:
    # Tokenization
    tokens = word_tokenize(response.lower())
    # Contraction Handling
    tokens = [expand_contractions(token) for token in tokens]
    # Stopword Removal
    tokens = [token for token in tokens if token not in stopwords_list]
    # Lemmatization
    tokens = [lemmatizer.lemmatize(token) for token in tokens]
    cleaned_tokens.append(tokens)
return cleaned_tokens
```

3.5 Data Analysis

Analysis targeted the three RQs with tailored techniques which is depicted in table 2:

Table 2: Analysis Techniques Summary

RQ	Technique	Tool / Library	Output
RQ1	TF-IDF + K-means	scikit-learn	Clusters of confusion
RQ2	Frequency Analysis	Pandas & Counter	Preference Frequencies
RQ3	Frequency + VADER	Counter & VADER	Sentiment Score & Skill Frequencies

RQ1: Identifying Areas of Confusion: The analysis began by converting preprocessed token lists into space-separated text to enable Term Frequency-Inverse Document Frequency (TF-IDF) vectorization, implemented using scikit-learn's TfidfVectorizer with a maximum of 400 features, and unigrams and bigrams (ngram_range=(1, 2)). Dimensionality reduction was performed using Principal Component Analysis (PCA) with 10 components to enhance computational efficiency. The optimal number of clusters (k) was determined by evaluating the elbow method and silhouette score across a range of 2 to 10 clusters, selecting the k that maximized silhouette coherence. K-means clustering was then applied with the optimal k, initialized using the k-means++ algorithm, a random state of 42, and 10 initializations to ensure the stability. Cluster assignments were derived, and the top five unique terms per cluster were extracted based on the TF-IDF scores, ensuring

no overlap between terms. Cross-tabulation by cohort was computed as percentages to assess the distribution across BCA, MCA, and alumni, with the results saved as CSV files for clusters and cross-tabulation.

RQ2: Determining Preferred Assessment Methods: Frequency analysis was conducted on the preprocessed categorical responses, sampling up to 1,000 responses (or the full dataset if smaller) to ensure stability. The Counter object counted the occurrences of each response option (e.g., "Collect Response on Program To Find & Explain Error"), and the top ten preferences were identified. Cross-tabulation by cohort was performed as a percentage distribution, enabling comparisons across BCA, MCA, and alumni. The results were saved as CSV files for frequency and cross-tabulation, facilitating quantitative insights into the assessment method preferences.

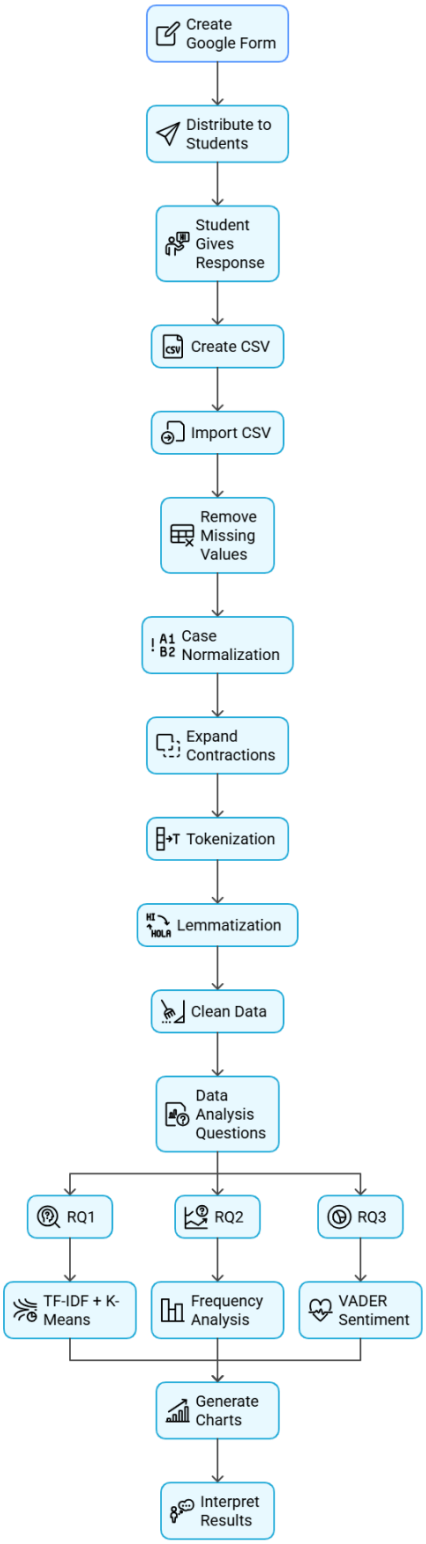

Figure 2: Flow Diagram of Data Processing Pipeline

RQ3: Exploring Essential Skills: The analysis involves generating bigrams from preprocessed token lists using a window of two, with frequencies counted using a counter. A predefined mapping dictionary merges overlapping bigrams (e.g., "solving skill" to "problem solving") to enhance interpretability. The top 10 unique bigrams were selected based on their frequency. Sentiment analysis was conducted using the VADER sentiment analyzer, computing compound scores ranging from -1 (negative) to +1 (positive) for each response. Aggregate metrics—average sentiment and standard deviation—were calculated, along with a sample of the first five sentiment scores. Cross-tabulation by cohort was performed as percentages to check for the presence of top bigrams in the responses. The results were saved as CSV files for bigram frequencies, cross-tabulation, and sentiment metrics, providing a comprehensive view of the skill perceptions and sentiments.
In summary, the methodology combined both quantitative and qualitative techniques to analyze student responses collected via open-ended and multiple-choice survey items. By applying natural language processing tools such as TF-IDF vectorization, K-means clustering, frequency analysis, and sentiment analysis, the study sought to extract meaningful patterns from student perceptions. The following section presents the results of this analysis, highlighting key trends related to programming challenges, assessment preferences, and perceived developer competencies across diverse academic backgrounds.

4. Results

This section presents the findings obtained from the application of natural language processing (NLP) and machine learning (ML) techniques to address three research questions (RQs) pertaining to student responses to programming assignments.

RQ1: Identification of Areas of Confusion in Programming Assignments

The technique employed for RQ1 involved Term Frequency-Inverse Document Frequency (TF-IDF) vectorization combined with K-means clustering, utilizing the scikit-learn library. Following earlier explorations in which the silhouette score was low (e.g., 0.017 with k=5), the number of clusters was reduced to k=3 to improve separation, resulting in a silhouette score of 0.262. The top terms for each cluster, based on TF-IDF weights, are detailed in Table 3, where each cluster represents a distinct theme of confusion: Cluster 0 suggests issues with syntax and error handling, Cluster 1 indicates challenges with logical structuring and implementation, and Cluster 2 reflects difficulties with foundational programming concepts. Cross-tabulation results, showing the percentage of responses from each cohort assigned to each cluster, are provided in Table 4 to enable comparisons across groups. A visual representation of the distribution of these confusion areas across cohorts is shown in Figure 3, which highlights their relative proportions.

Table 3: Top Terms by Cluster (TF-IDF Weights)

Cluster	Top Terms (TF-IDF Weight)
0	{'error': 21.09, 'syntax': 9.46, 'solving': 6.95}
1	{'logic': 25.99, 'implementation': 8.54, 'pattern': 6.05}
2	{'concept': 15.22, 'programming': 13.48, 'difficult': 5.47}

Table 4: Cross-Tabulation of Clusters by Class (Percentage Distribution)

Class	Cluster_0	Cluster_1	Cluster_2	Cluster_3	Cluster_4
Alumni	62.4	0.8	9.6	6.4	20.8
BCA	57.41627	3.349282	14.35407	8.133971	16.74641
MCA	57.47126	3.448276	14.94253	4.597701	19.54023

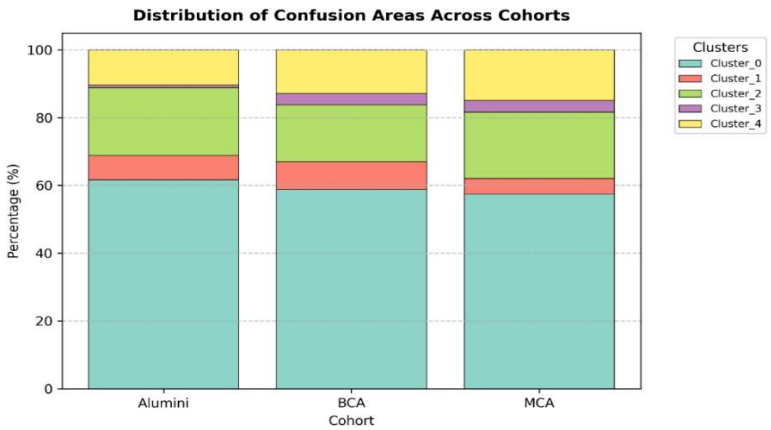

Figure 3: Distribution of Confusion Areas Across Cohorts

Key confusion areas are further illustrated in Figure 4, highlighting the prevalent terms based on the TF-IDF weights.

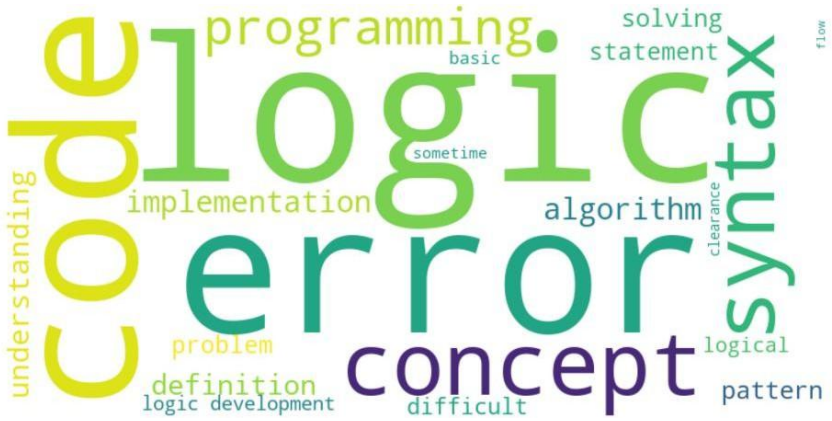

Figure 4: Key Confusion Areas in Programming Assignments (TF-IDF Weights)

RQ2: Determination of Preferred Assessment Methods

The technique employed for RQ2 consisted of frequency analysis, implemented using pandas and the counter class from the collection library, supplemented by cross-tabulation to assess class-wise preferences. Frequency analysis identified the top three preferred methods for measuring students' programming concept understanding based on a sampled dataset. The most frequently selected method was "Collect Response on Program To Find & Explain Error" with 260 occurrences, followed by "Practical Quiz [Written & Lab]" with 191 occurrences, and "Viva" with 57 occurrences. The results are presented in Table 5.

Table 5: Preferred Methods by Frequency

Response	Frequency
Find & Explain Error	260
Practical Quiz	191
Viva	57

Cross-tabulation analysis examined the percentage distribution of the preferred assessment methods across the classes of Alumni, BCA, and MCA. The results indicate that 47.200000% of Alumni, 51.674641% of BCA students, and 53.448276% of MCA students preferred "Collect Response on Program To Find & Explain Error." For "Practical Quiz [Written & Lab]," the preferences were 42.400000% among Alumni, 39.234450% among BCA students, and 32.183908% among MCA students. The method "Viva" was preferred by 10.400000% of Alumni, 9.090909% of BCA students, and 14.367816% of MCA students. The percentage distributions are listed in Table 6.

Table 6: Cross-Tabulation of Preferred Assessment Methods by Class (Percentage)

Class	Find & Explain Error	Practical Quiz	Viva
Alumni	47.2	42.4	10.4
BCA	51.674641	39.23445	9.090909
MCA	53.448276	32.183908	14.36782

RQ3: Exploration of Essential Developer Skills

The technique employed for RQ3 combines frequency analysis using the counter class with VADER sentiment analysis utilizing the vaderSentiment library. The top 6 unique bigrams, representing grouped skills perceived as essential (e.g., 'problem solving,' 'programming skill' and 'debugging skill' as core technical abilities, 'analytical thinking' and 'collaboration skill' as cognitive and teamwork competencies, 'communication skill' as softer or specialized skills), are listed in Table 7. The average sentiment score across responses was 0.134, with a standard deviation of 0.293. Cross-tabulation results, showing the percentage of responses from each cohort mentioning these skills, are presented in Table 8 to compare the skill emphases. The distribution of the sentiment scores by cohort is shown in Fig. 5.

Table 7: Top 6 Bigrams by Frequency

Bigram	Frequency
Problem Solving	192
Programming Skill	134
Analytical Thinking	94
Collaboration Skill	70
Communication Skill	27
Debugging Skill	22

Table 8: Proportion of Responses Mentioning Key Skills by Class (Percentage)

Skill	Class	False(%)	True(%)
Problem Solving	Alumni	84.42623	15.57377
	BCA	84.13462	15.86539
	MCA	89.01734	10.98266
Programming Skill	Alumni	91.80328	8.196721
	BCA	89.42308	10.57692
	MCA	94.21965	5.780347
Analytical Thinking	Alumni	88.52459	11.47541
	BCA	86.53846	13.46154
	MCA	93.06358	6.936416
Collaboration Skill	Alumni	86.06557	13.93443
	BCA	96.15385	3.846154
	MCA	86.12717	13.87283
Communication Skill	Alumni	95.90164	4.098361
	BCA	95.19231	4.807692
	MCA	93.06358	6.936416
Debugging Skill	Alumni	94.2623	5.737705
	BCA	96.15385	3.846154
	MCA	95.95376	4.046243

The boxplot in Figure 5 illustrates the distribution of sentiment scores for responses from the MCA, Alumni, and BCA cohorts, derived from the VADER sentiment analysis. Each box represents the interquartile range (IQR) spanning the 25th to 75th percentiles, with the median score marked as a line within the box. Whiskers extend to the minimum and maximum values within 1.5 times the IQR, and any points beyond these whiskers are plotted as outliers. For MCA, the median sentiment score was approximately 0.10, with an IQR from -0.05 to 0.25, and several outliers below -0.5, indicating a range of negative sentiments. The Alumni cohort showed a median of 0.15, with an IQR of 0.00 to 0.30, and fewer outliers, suggesting a slightly more positive distribution. The BCA cohort had a median of 0.12, with an IQR of -0.02 to 0.28, and moderate outlier presence, reflecting a balanced sentiment range. This visualization complements the average sentiment score of 0.134 and the standard deviation of 0.293 reported in the text, offering a detailed view of variability across cohorts.

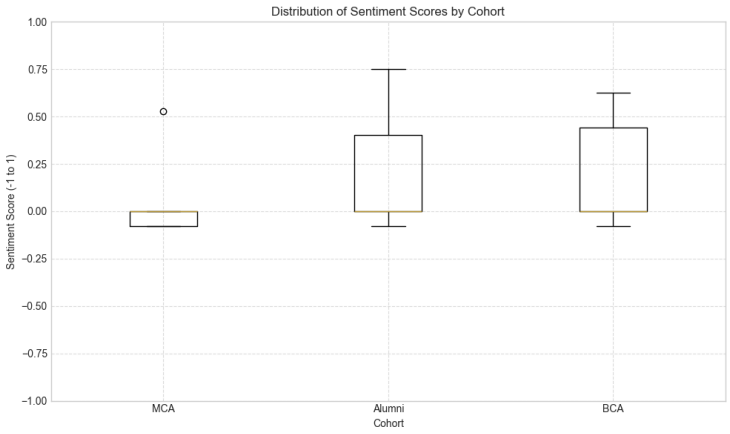

Figure 5: Distribution of Sentiment Scores by Cohort

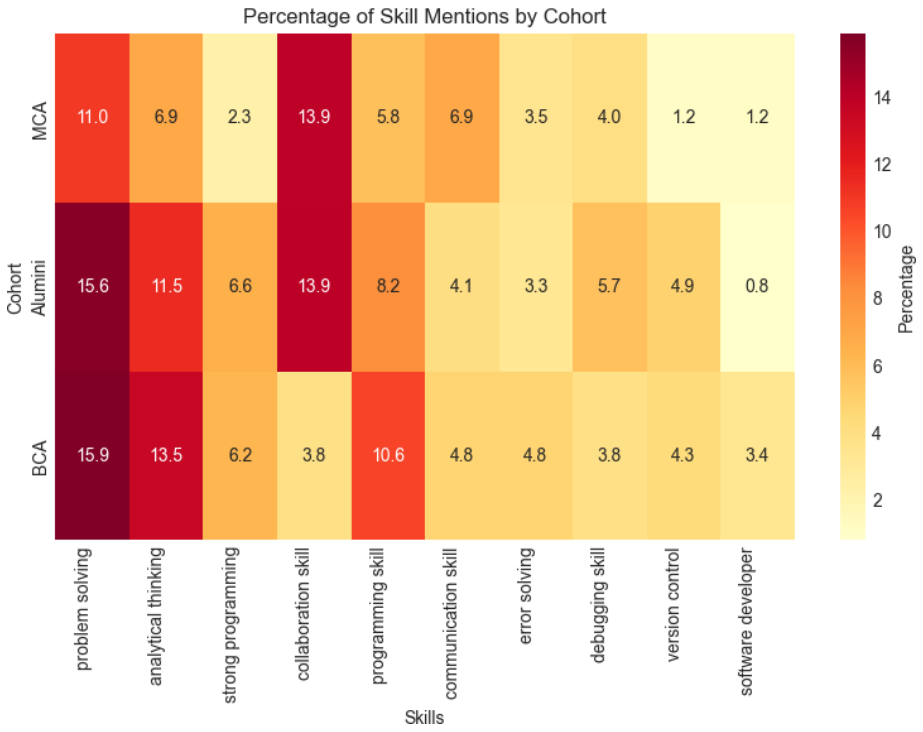

Figure 6: Percentage of Skill Mentions by Cohort

The Heatmap in Figure 6 visualizes the percentage of responses mentioning each of the top 10 skills across the Alumni, BCA, and MCA cohorts, as detailed in Table 8. The color intensity, ranging from light yellow (lower percentages, e.g., 3.8%) to dark red (higher percentages, e.g., 15.9%), represents the "True (%)" values, with a color bar providing the percentage scale. Rows correspond to skills (e.g., 'problem solving and ' programming skill'), and the columns represent cohorts.
For instance, 'problem solving' shows higher mentions in BCA (15.9%) and alumni (15.6%) compared to MCA (11.0%), indicated by darker shades for BCA and Alumni. Conversely, 'collaboration skill' has a notable mention in Alumni (13.9%) and MCA (13.9%), with a lighter shade for BCA (3.8%), highlighting cohort-specific emphases. This visual aid complements tabular data by emphasizing patterns in skill perception across groups.

5. Discussion

The findings from the analysis of 508 student responses in the study "Analyzing Students' Perception of Attributes on Programming Assignments" provide valuable insights into the challenges, preferences, and skill perceptions within programming education, offering a foundation to address the research questions (RQ1: What are the biggest problems students face when learning to code? RQ2: What are the ways in which students think are best at checking their coding knowledge? RQ3: What abilities do students think are important for coders?). This section interprets these results in light of the literature, evaluates their implications, and identifies their limitations and future research directions.

5.1 RQ1: What are the biggest problems students face when learning to code?

The identification of three confusion clusters—syntax and error handling (Cluster 0), logical structuring and implementation (Cluster 1), and foundational programming concepts (Cluster 2)—through TF-IDF and K-means clustering (k=3, silhouette score 0.262) provides a detailed insight into student difficulties. This aligns with [17] and [18], who identify syntax, logical reasoning, and abstraction as persistent barriers, indicating consistent challenges across educational contexts. The notable presence of alumni in the syntax/error cluster (62.4%) suggests that early stage difficulties may persist without adequate reinforcement, a perspective that is less emphasized in prior literature focused on novices. Cohort variability (e.g., BCA's 14.35% in logical structuring versus MCA's 14.94%) further implies that experience shapes problem perception, offering a novel lens for tailoring interventions. Curriculum designers can develop modular courses addressing alumni syntax and logic for undergraduates, while instructors can integrate hands-on debugging labs to enhance practical skills. Policymakers can advocate competency-based progressions to ensure mastery across stages.

5.2 RQ2: What ways do students think are best for checking their coding knowledge?

For RQ2, the preference for "Collect Response on Program To Find & Explain Error" (47.2%–53.4% across cohorts) over practical quizzes and vivas resonates with [19], who noted the efficacy of automated feedback in improving engagement. This preference, particularly strong among MCA students (53.4%), contrasts with [14],

who reviewed viva subjectivity without considering the students' viewpoints. The results suggest that students value assessments that test conceptual understanding over memorization, a perspective absent from the literature's instructor-centric focus. This shift implies that curriculum designers should prioritize error-identification tasks, potentially integrating automated tools to align with industry demands for diagnostic skills. The lower viva preference (9.1% - 14.4%) across cohorts, despite its traditional use, underscores the need to reconsider its relevance in modern programming education. Curriculum designers should embed error-identification tasks with real-time feedback tools, whereas instructors should adopt hybrid assessments to balance creativity and rigor. Policymakers may update the evaluation standards to reflect student priorities.

5.3 RQ3: What abilities do students think are important for coders?

The prominence of problem-solving (15.9% BCA), programming skills, and collaboration (13.9% alumni) as essential abilities, along with a positive sentiment trend (average 0.134, alumni 0.15), corroborates the findings of [16], who emphasized these competencies in professional settings. The lower MCA sentiment (median 0.10, with negative outliers) diverges from [6], who noted a consistent valuation of analytical thinking, suggesting that advanced coursework may induce frustration, an underexplored aspect in prior studies. Alumni's balanced view of collaboration bridges academic and industry expectations and addresses the curriculum gap identified by Kanika et al. [3]. This evolution highlights the need to develop more dynamic skills. These insights suggest that instructors should integrate collaborative projects and sentiment-aware feedback systems to nurture well-rounded developers.

5.4 Limitations

Several constraints limit the generalizability and depth of these findings. The sample, confined to a single Indian institution, may not reflect global programming education trends, a concern echoed by authors in study [22] regarding diversity in computing studies. The survey's three-question design limits the breadth of insights, possibly omitting nuanced difficulties or preferences, and while the pilot's small sample size (15 students) may not fully validate the question's clarity. Additionally, K-means clustering's reliance on a subjective k=3 choice, despite a silhouette score improvement of 0.262, introduces analytical uncertainty, as Wang et al. in their study [21], cautioned about the low separation in educational text clustering. These factors suggest caution when extrapolating the results beyond the current context.

5.5 Future Directions

To address these limitations and deepen the investigation, future research should expand the sample to include multiple institutions and countries, thus enhancing diversity and generalizability, as recommended by author in their study [22]. Implementing informed consent protocols could strengthen ethical rigor and provide clearer insights into participants' motivations. Longitudinal studies that track the same cohorts over time would elucidate how perceptions and difficulties evolve, thus complementing the current cross-sectional design. Expanding the survey to include questions on advanced topics (e.g., AI programming and version control) could capture a broader skill spectrum, while exploring alternative clustering methods (e.g., hierarchical or DBSCAN) might improve the robustness of the RQ1 analysis, addressing the k=3 subjectivity noted by [21]. Such efforts would further align programming education with students' needs and industrial expectations.

In conclusion, this study illuminates the critical perceptions that shape programming education by offering a foundation for tailored teaching and assessment strategies. By bridging the literature gaps through diverse cohort analyses and advanced NLP techniques, it paves the way for innovative educational technologies and practices, with future research poised to deepen these insights.

6. Conclusion

This study offers a nuanced understanding of students' experience in programming education through the application of advanced computational methods. By addressing the research questions concerning challenges in programming (RQ1), preferred assessment methods (RQ2), and the perceived skills of effective programmers (RQ3), the investigation synthesizes insights from 508 responses across BCA, MCA, and alumni cohorts. The findings reveal a complex landscape where logical and syntactic difficulty predominate, interactive error-focused assessments are favored, and problem-solving emerges as a cornerstone skill, reflecting both persistent hurdles and evolving learner priorities.

The study's primary contribution lies in its innovative use of NLP techniques, including TF-IDF with K-means clustering and VADER sentiment analysis, to extract meaningful patterns from mixed-format feedback. This approach not only enhances the scalability of analyzing open-ended responses but also provides a robust framework for identifying thematic challenges and skill perceptions, a departure from traditional quantitative surveys. Furthermore, the exploration of assessment preferences offers actionable insights for automated grading systems, while the identification of skill hierarchies informs curriculum development, positioning this study as a bridge between educational technology and pedagogical practices. These contributions underscore the potential of ML-driven analytics to transform how educational stakeholders interpret and respond to student needs.

These practical implications are significant for educators, institutions, and designers of grading systems. Educators can leverage the identified challenges to refine teaching strategies, emphasizing logical reasoning and debugging support through targeted exercises or intelligent tutoring systems. Institutions may consider integrating these insights into curriculum redesign to foster a balanced skill set that aligns with industrial demands. For grading system designers, the preference for error-explanation tasks suggests a shift toward diagnostic and feedback-rich tools, thereby encouraging the development of hybrid platforms that combine automation with interactive elements. Such adaptations promise to enhance learning outcomes and align assessment practices with student expectations.

Nevertheless, this study has some limitations. A single-institution sample restricts the generalizability of the findings, potentially overlooking institutional or cultural variations. Additionally, the absence of gender-based analysis, despite balanced participant distribution, limits the ability to address sex-specific perceptions, an area that warrants ethical and methodological attention. These constraints highlight the need for cautious interpretation and broader application.

Future research should include multi-institutional and culturally diverse cohorts to validate and enrich these findings. Incorporating gender and other demographic analyses could uncover nuanced perspectives and address critical gaps in educational data mining. Longitudinal studies tracking skill perception evolution over time, particularly among alumni, would further illuminate long-term educational impacts. Moreover, experimental investigations of NLP-enhanced interventions based on these clusters could assess their efficacy, paving the way for scalable educational technology. By pursuing these directions, this field can build on the foundation of this study to advance programming education in an increasingly data-driven era.

References

[1] E. Hegarty-Kelly and D. A. Mooney, “Analysis of an automatic grading system within first year Computer Science programming modules,” Jan. 2021, pp. 17–20. doi: 10.1145/3437914.3437973.

[2] J. C. Paiva, A. Figueira, and J. P. Leal, “Automated Assessment in Computer Science Education: A State-of-the-Art Review,” ACM Transactions on Computing Education, vol. 22, no. 3, pp. 1–40, Jun. 2022, doi: 10.1145/3513140.

[3] K. Kanika, P. Chakraborty, and S. Chakraverty, “Tools and Techniques for Teaching Computer Programming: A Review,” Journal of Educational Technology Systems, vol. 49, no. 2, pp. 170–198, May 2020, doi: 10.1177/0047239520926971.

[4] X. Xia, D. Lo, Z. Wan, and P. S. Kochhar, “How Practitioners Perceive Coding Proficiency,” May 2019, pp. 924–935. doi: 10.1109/icse.2019.00098.

[5] E. Kalliamvakou, K. Blincoe, L. Singer, D. Damian, and D. Germán, “Open source-style collaborative development practices in commercial projects using GitHub,” May 2015, vol. 1, pp. 574–585. doi: 10.5555/2818754.2818825.

[6] T. Michaeli and R. Romeike, “Improving Debugging Skills in the Classroom,” Oct. 2019. doi: 10.1145/3361721.3361724.

[7] A. Verma, N. Kn, P. Udhayanan, R. M. Shankar, and S. K. Chakrabarti, “Source-Code Similarity Measurement: Syntax Tree Fingerprinting for Automated Evaluation,” Oct. 2021, pp. 1–7. doi: 10.1145/3486001.3486228.

[8] M. Tushev, G. Williams, and A. Mahmoud, “Using GitHub in large software engineering classes. An exploratory case study,” Computer Science Education, vol. 30, no. 2, pp. 155–186, Dec. 2019, doi: 10.1080/08993408.2019.1696168.

[9] G. Salvaneschi and M. Mezini, “Debugging for reactive programming,” May 2016. doi: 10.1145/2884781.2884815.

[10] A. Yusuf and N. M. Noor, “Revising the computer programming attitude scale in the context of attitude ambivalence,” Journal of Computer Assisted Learning, vol. 39, no. 6, pp. 1751–1768, May 2023, doi: 10.1111/jcal.12838.

[11] M. Mccracken et al., “A multi-national, multi-institutional study of assessment of programming skills of first-year CS students,” ACM SIGCSE Bulletin, vol. 33, no. 4, pp. 125–180, Dec. 2001, doi: 10.1145/572139.572181.

[12] K. Mangaroska, K. Sharma, D. Gašević, and M. Giannakos, “Exploring students’ cognitive and affective states during problem solving through multimodal data: Lessons learned from a programming activity,” Journal of Computer Assisted Learning, vol. 38, no. 1, pp. 40–59, Sep. 2021, doi: 10.1111/jcal.12590.

[13] N. Enders, V. Kubik, and R. Gaschler, “Online Quizzes with Closed Questions in Formal Assessment: How Elaborate Feedback can Promote Learning,” Psychology Learning & Teaching, vol. 20, no. 1, pp. 91–106, Nov. 2020, doi: 10.1177/1475725720971205.

[14] S. J. Reckinger and S. M. Reckinger, “A Study of the Effects of Oral Proficiency Exams in Introductory Programming Courses on Underrepresented Groups,” Feb. 2022, pp. 633–639. doi: 10.1145/3478431.3499382.

[15] A. Luxton-Reilly et al., “Introductory programming: a systematic literature review,” Jul. 2018, pp. 55–106. doi: 10.1145/3293881.3295779.

[16] S. A. Fincher and A. V. Robins, The Cambridge Handbook of Computing Education Research. cambridge university, 2019. doi: 10.1017/9781108654555.
[17] R. Lister, "Toward a Developmental Epistemology of Computer Programming," Oct. 2016, vol. 136, pp. 5–16. doi: 10.1145/2978249.2978251.
[18] S. Simon et al., "Pass Rates in Introductory Programming and in other STEM Disciplines," Dec. 2019, pp. 53–71. doi: 10.1145/3344429.3372502.
[19] P. Ihantola, T. Ahoniemi, O. Seppälä, and V. Karavirta, "Review of recent systems for automatic assessment of programming assignments," Oct. 2010, vol. 22, pp. 86–93. doi: 10.1145/1930464.1930480.
[20] A. Joan O Vicente, A. Ray O Yu, and T. Adelaine G Tan, "Collaborative Approach in Software Engineering Education: An Interdisciplinary Case," Journal of Information Technology Education: Innovations in Practice, vol. 17, no. 1, pp. 127–152, Jan. 2018, doi: 10.28945/4062.
[21] F. Wang, R. Ross, J. D. Kelleher, H.-H. Franco-Penya, and J. Pugh, "An Analysis of the Application of Simplified Silhouette to the Evaluation of k-means Clustering Validity," springer, 2017, pp. 291–305. doi: 10.1007/978-3-319-62416-7_21.
[22] W. M. Reinke, N. Dong, and K. C. Herman, "The Incredible Years Teacher Classroom Management Program: Outcomes from a Group Randomized Trial.," Prevention Science, vol. 19, no. 8, pp. 1043–1054, Jul. 2018, doi: 10.1007/s11121-018-0932-3.
[23] S. Lewis, "Qualitative Inquiry and Research Design: Choosing Among Five Approaches," Health Promotion Practice