A Bot Assisted Instructional Framework for Teaching Introductory Programming Course(s)

Deepak Dawar
daward@miamioh.edu
Dept. of Computer and Information Technology
Miami University
Hamilton, Ohio 45011, U.S.A.

Abstract

Learning computer programming is a challenging task for most beginners. Demotivation and learned helplessness are pretty common. A novel instructional technique that leverages the value-expectancy motivational model of student learning was conceptualized by the author to counter the lack of motivation in the introductory class. The result was a frequency adherent scaffolded instructional technique called An Assignment A Day (AAAD). Instead of writing an assignment and a lab for each module/chapter, students were asked to complete one assignment a day, not exceeding four assignments a week. The assignments were incrementally difficult and had to be done almost every day. With the application of AAAD for two consecutive semesters, there was a meaningful improvement in the final grades. This technique, though initially encouraging, created a significant load on the instructor in terms of assignments graded and questions answered every day. A natural language processing (NLP) based conversational agent was designed and integrated with AAAD to counter this overload. The idea was simple – relay commonly asked course questions to an NLP based chatbot and let the instructor handle the complex queries. This integrated system was named Conversational Agent Supported Scaffolded Approach (CASSA). The main contribution of this work is the construction of a conversational agent and its integration with AAAD. The conversational agent is currently being assessed for overall efficacy, though preliminary results are discussed. The vision is to create a generic virtual assistant template that can be re-used across multiple courses to assist instructors.

Keywords: Conversational agents, NLP, introductory programming, pedagogy, value-expectation, student procrastination.

1. INTRODUCTION

Computer programming is an arduous learning process for most beginners, and high failure rates have been reported continuously (Allan & Kolesar, 1997; Newman, Gatward, & Poppleton, 1970; Bennedsen & Caspersen, 2007; Sheard & Hagan, 1998; Watson & Li, 2014; Beaubouef & Mason, 2005; Howles, 2009; Kinnunen & Malmi 2006; Mendes et al., 2012). Given the complex nature of the programming (Kim & Lerch, 1997; Rogalski & Samurçay, 1990; Robins, Rountree & Rountree, 2003), students frequently get demotivated. While teaching multiple introductory programming courses over many years, the author observed that apart from the complex

nature of programming, there were other factors at play that feed the demotivation loop. Some examples are:

- Less than desirable instructor presence
- High temporal disengagement with the programming activities
- Students internal lack of motivation

Keeping these factors in mind, and inspired by value-expectancy (Keller, 1983) & cognitive load theory (Paas, Renkl, & Brünken, 2010; Sweller, 1988, 1994), a novel instructional technique called An Assignment A Day (AAAD) approach was designed. Instead of completing a lab and assignment per chapter, students were asked to complete one simple assignment a day, with a

cap of four assignments a week. Every subsequent assignment of a chapter/course built on the previous assignment and carried an incremental cognitive load (see Appendix A). Apart from testing students on new concepts, the subsequent assignment reused the concepts learned/applied in the previous assignment. The approach (Dawar, 2021) can be summarized as:

1. Students will ideally do one assignment per day.
2. Opening assignments of the chapter will test students on very basic skills like writing a method stub. Subsequent assignments will gradually increase in complexity keeping in mind the cognitive load asserted by the assignment. This mechanism is in part based on the study conducted by Alexandron et al. (2014).
3. There will not be more than four assignments per week. Deadlines may be relaxed on a case-to-case basis.
4. As an exception, and depending upon the cognitive load, an assignment may be completed in two or more days rather than a single day.

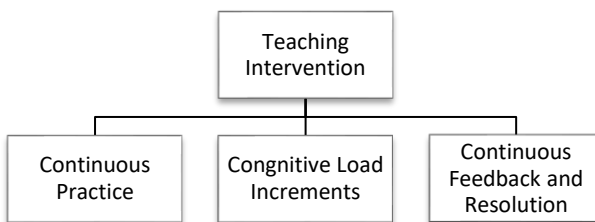The technique rests on three central pillars, as shown in Figure 1.

Figure 1: AAAD Interventional Technique

This study aims to address two research questions:

a) What is the effect of mandatory continuous engagement with cognitively germane testing on student outcome and instructor load?
b) How can instructor load be minimized while maintaining the sanctity of the technique?

The author could foresee at least two significant issues that could derail the potential acceptability of this technique:

a) Will the high number of assignments, albeit of germane cognitive load, dissuade students from participating, thereby compounding the very problem the author is trying to tackle, i.e., lack of motivation due to learned helplessness? Constant testing has been associated with high student anxiety (Kaplan et al., 2005). An easy way to make students dislike programming is to put them under unnecessary stress (Goold & Rimmer, 2000). Strict enforcement of everyday deadlines may easily overwhelm these students. The only chance of overcoming this hurdle was providing germane load assignments.
b) Even if the intervention shows promising results with students, what does that mean for the instructor load? More assignments would naturally elicit more questions, requiring additional instructional and tutoring presence, and more grading time, besides other externalities. Massive overload and instructor fatigue become apparent. Some follow-up questions are warranted. For example:
1. Is it prudent or even feasible to run a potentially beneficial instructional intervention while risking instructor overload simultaneously?
2. If the intervention is proven to be beneficial, how can instructor support be increased so that the outcome is better for students (in terms of motivation) as well as the instructor (in terms of course load)?
3. Do the system and tools required for instructor support already exist, or would they need to be allocated/constructed?
4. Are these support systems course-specific, or can they be reused within courses?

These questions are vast and may need multiple solutions at multiple levels. As a preliminary solution, a conversational agent or a chatbot is proposed to assist the instructor. The essential function of this agent is to answer repeatedly asked student questions in the course when access to the instructor is not available.

The rest of the paper is structured as follows. Section 2 discusses the perceived need for the intervention and the conversational agent and builds a case for their integration. Section 3 touches upon the operational aspects of natural language processor systems (NLP) and illustrates the parts of the conversational agent. Section 4 discusses the preliminary results for the accuracy of the conversational agent.

Section 5 concludes the paper and briefly presents the foundations of future research.

2. A Case for Integration of a Conversational Agent With Scaffolded Instructional System

In this section, justification for building and employing the AAAD technique is presented. It is

then argued that while this might be a good idea for student motivation and performance, it can overload the instructor who lacks access to dedicated resources like graders and tutors. A case is then built for the construction and use of a conversational agent/chatbot to take some load off the instructor while not jeopardizing the instructional technique. The terms conversational agent and chatbot are used interchangeably throughout the paper.

A Case for AAAD Approach
Students' belief in their success is vital if they are to be motivated to learn. There are many causes of student demotivation, but the one suspect that the author can categorically point towards in their classrooms is high cognitive load. Cognitive load theory (Paas, Renkl, & Brünken, 2010; Sweller, 1988, 1994) throws light on the aspects of load placed on working memory while a task is being executed. Computer programming requires balancing numerous interactive tasks simultaneously. For example, it involves juggling numerous details like problem domain, the current state of the program, language syntax, strategies (Winslow, 1996).

Procrastination is extremely prevalent in students studying in a university setup. Some estimates suggest that 80 to 95 percent of students engage in procrastination (Steel, 2007). The longer the students wait to turn in the assignment, the worse their grades become (Kim & Seo, 2015). Procrastination has also been linked to higher levels of anxiety, stress, and fatigue (Beutel et al., 2016). After having taught multiple programming courses over multiple years, the author encountered similar patterns.

AAAD was designed keeping these factors in mind. The intervention made continuous targeted interaction between the material and students – somewhat mandatory. It was opined that this would:

- Establish a clear study pattern for students to counter procrastination.
- Potentially **improve student's** expectations owing to germane cognitive loads.
- Make them practice programming every almost every day. The inspiration for this operation came from strong evidence suggested by psychological studies (Brown & Bennett, 2002; Glover, Ronning & Bruning, 1990; Moors & De Houwer, 2006) done on variable student populations. Constant practice can improve student motivation and make them want to learn more (Moss & Case, 2001).

The technique AAAD was administered to two experimental groups (E1 and E2), and the study was spread over three semesters. The control group (C1) was asked to complete one assignment and one lab work per week. Quizzes were given at the end of every chapter. This is the usual approach followed at our institution for introductory programming classes. E1 and E2 were taught with the interventional approach for the subsequent two semesters.

Both experimental groups were asked to complete 37 assignments over the course of 12 weeks. 10 days were meant for chapter quizzes and exams. Other details like student population comparison of the groups, determination of germane load mechanism can be found in (Dawar, 2021).

All groups were administered the same module quizzes and final exam, and their average scores were compared to measure the impact of this technique on overall grades if any.

Module	C1 (20 students)	E1 (22 students)	E2 (20 students)
1	71% (3.72)	75% (2.05)	**75% (2.22)**
2	79% (2.08)	71% (2.33)	**78% (3.32)**
3	73% (3.19)	73% (2.55)	**73% (3.68)**
4	62% (3.72)	66% (2.49)	**71% (3.01)**
5	74% (4.26)	75% (2.44)	**75% (3.10)**
6	67% (3.41)	67% (1.78)	**76% (1.95)**
7	56% (3.48)	65% (2.50)	**61% (3.30)**
Average	68% (3.40)	70% (2.30)	**73% (2.94)**

Table 1: Mean grade points (with standard deviations) scored on the quiz by all groups

As shown in Table 1, seven chapters/modules were taught to all the groups. A quiz was given at the end of every chapter. Columns C1, C2, and E2 depict the average class scores (with standard deviations) of the quiz. The final exam consisted of a quiz that covered all seven modules, and a Java problem. Table 2 shows the average achieved by the class in the final exam.
Though there was no significant difference between module quiz scores (see Table 1), the experimental groups performed much better in the final exam (Table 2).
Even though the gains in the final quiz are marginal, the experimental groups outperformed the control group by 20 percentage points or more in JAVA program writing. The overall cumulative improvement in the final exam mean score was 16% and 19% for E1 and E2, respectively.

These numbers may insinuate that – for the experimental groups – the increased practice led to an improvement in final exam score, though it is too early to say anything with a high degree of confidence due to such a small sample size. Nevertheless, the final exam numbers are encouraging.

Group	Average Final Quiz Score	Average JAVA Program Score	Cumulative Average
C1	66%	51%	56%
E1	74%	71%	72%
E2	78%	74%	75%

Table 2: Final exam score for all groups

An end-of-course survey (see Appendix C) was conducted for both E1 and E2. The number of participants was 22 and 13 respectively, i.e., 35 students in total. One of the questions asked the students about how they felt about the utility and effectiveness of this intervention in completing the course satisfactorily. A surprising 90% of the students in E1 and 84% in E2 answered that they felt positive/better about using this technique, while 10% in E1 and 9% in E2 reported that they felt slightly worse while working with this technique.

A cumulative 45% of the students answered that working every day on assignments made it easy for them to manage stress. Students remarked that the process made it easy to manage overall stress as the assignments were gradually increasing in difficulty. 39% said it increased their stress levels as they had to do many more assignments, and 15% choose that it made no difference. The final exam results, along with the student survey responses, instilled confidence in the instructor that this technique was worth exploring.

There was one glaring and unavoidable cost of these improvements – instructor overload.

A Case for the Conversational Agent
The improvements in final exam scores, though encouraging, came at a high price as far as the instructor load was concerned. The frequency of questions asked increased in number, indicating more students were interested in asking questions. Replying to these questions consumed a significant amount of time. This load grew as the course progressed because assignments were due almost every day of the week and had to be graded quickly to provide timely feedback to students. Since every assignment was built on top of the previous one, delayed grading could mean students had no previous feedback available while attempting the current assignment. This delay is just not an option when working with AAAD. Hence, it can be seen how quickly the instructor load can increase to the point of exhaustion.

There was undoubtedly a need for support structures for the instructor. One way would be to hire a dedicated tutor and a grader. However, many instructors, due to numerous reasons, do not have access to such support. Another way would be to create a scripted expert system containing scripted question-answers. The script is a decision tree modeled by domain experts that determines which path to take in response to a question. These are static systems that may be unsuitable in circumstances where a single question can be asked in multiple ways.

Instead, a Natural Language Processing (NLP) based conversational agent/chatbot capable of answering course-related questions is chosen for bot construction in this work. The reasons for implementing such a conversational agent are multifold:

1. Many students ask the same question in different ways: Questions asked by students may be divided into two parts; text-based and knowledge-based (Scardamalia & Bereiter, 1992). Text based questions refer to queries generated as part of reading a text, while knowledge-based questions are generated through a deep interest in the topic to extend knowledge. Through the years of teaching introductory programming courses observed, the author of this work observed that many questions asked by multiple students were text-based and strikingly similar. In those cases, only the semantics and structure of the question differed, while the context of the question was the same. Hence, a system capable of understanding the context of a text based question could effectively classify multiple questions from multiple students into the same bucket and respond with a specific predefined answer. Directing these questions to an NLP-based conversational agent can save the instructor much time, which can be utilized in other areas such as mentoring. Predefined responses may not be suitable for knowledge-based questions, though.

2. Quick resolution of trivial queries: Many text-based questions asked by the

students are simple and straightforward in nature. These can be easily handled by the conversational agent, saving precious time.

3. **Student's expectation of a quick response**: Interaction between instructor and student is critical for student success, more so in an online environment (Chang 2009). Many studies (Li et al., 2010; Chang et al., 2015) have confirmed that students prefer asynchronous modes of communication like email or chat while interacting with instructors. A well-designed conversational agent can easily fulfill this task. Given these findings and **the author's own experiences in the** classroom, it is opined that the quicker a query is resolved, the stronger the **student's conviction** there is merit in asking questions, as they will be resolved quickly. This could lead to a reinforcement loop, making students more comfortable asking questions.

4. Long-term potential: As society goes increasingly digital, the current model of fixed classrooms, printed textbooks, and static lectures clearly fall short of fulfilling the expectations the society has of the educational establishment. Digital generation tends to learn at short or twitched speeds through parallel processing while simultaneously connected to others (Beavis, 2010). It is reported that students learn more when they immediately apply what they learned and receive help from human tutors who respond quickly (Colvin, 2007; Anwer et al., 2015). A conversational agent which is always ready to respond to student queries can be a great add-on in the toolkit of instructors.

Given all these factors, it was decided to pursue the integration of a conversational agent with the AAAD technique to create CASSA.

3. SYSTEM DESIGN

Figure 2 presents an abstracted view of CASSA. The student initiates a query through a text dialogue/message. If the conversational agent is capable of answering the query, it is annotated as **"Simple," and the response is returned.** Otherwise, the query is automatically sent to the instructor via email through the agent and is **annotated as "Complex." When the instructor is** notified of an unanswered query, they update the knowledge base of the conversational agent with

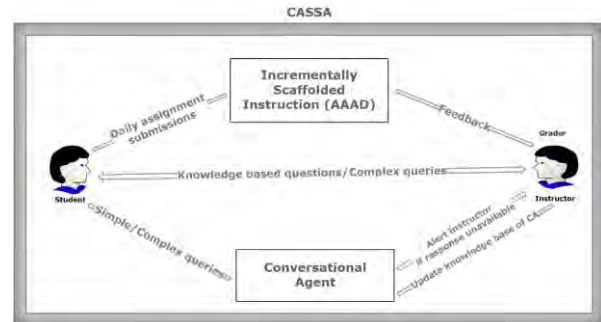a potential response while relaying the same answer/solution to the student.

Figure 2: CASSA – An Abstraction (see Appendix B for expanded view)

Design Considerations
The retrieval process of many modern conversational agents makes use of advances in machine learning in which responses are based on predefined rules as well as analysis of the web searches. Some prominent contemporary **examples are Amazon's Echo, Microsoft's Cortana, and Apple's Siri to name a few** (Weinberger, 2017). The agents on the other side of the spectrum use generative algorithms and assemble responses using statistical machine translation techniques. One popular example of such mechanisms is Seq2Seq, which uses **recurrent neural networks (RNN's) to accomplish** the response generation.

For this work, the former approach of predefined rules aided with natural language processing algorithms was chosen. There are at least three reasons for this choice:

a) The landscape of questions asked by students in a particular course may be large, but the questions would certainly be limited by the domain of the course. This can be achieved through rule-based or information retrieval methods more efficiently since generative methods tend to be reasonably much more complex to construct.

b) By defining a rule-based template, it would be a lot easier to use the same template as a basis for another course, thereby possibly achieving re-usability in the future.

c) Generative algorithms like Seq2Seq and systems that use them tend to be relatively complex in construction and operation. Hence, it was deemed fair to use a rule-based system as a pilot.
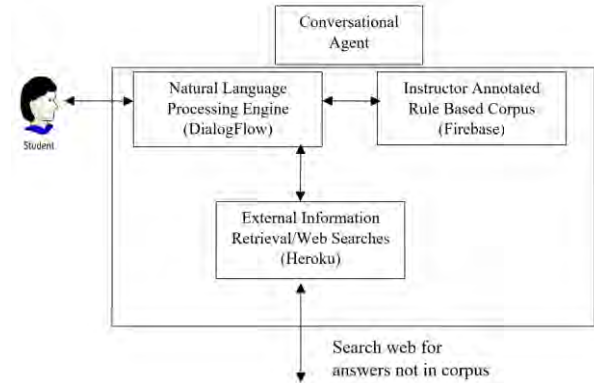
Figure 3: Conversational Agent Architecture

Figure 3 presents an abstracted view of the conversational agent used in this work. Its sub-parts are discussed below.

a) Student: Students can initiate a dialogue through three interfaces – Instructor provided web link, Dialogflow messenger, and Telegram. The student's questions are presented to the natural language processing (NLP) engine of the conversational agent (CA). It is assumed that in this day and age, students have access to the internet and should have the ability to initiate a conversation from an interface of their choice. More integrations like Facebook Messenger, Slack are possible in the future.

b) Natural Language Processing Engine (NLP): NLP can be defined as manipulation of natural language like text or speech, using mathematical representations and software. The main goal of any NLP system is to take in an unstructured input and provide a structured output. This work makes use of Dialogflow, a Google product, and a commercially available NLP platform for developing chatbots. It provides a powerful natural language processer capable of handling contextual conversations. It uses deep parsing techniques and is mainly used as integration between a conversational interface (Telegram, Slack, etc.) and the chatbot.

c) Knowledge Base: The accuracy and final employability of conversational agents depend greatly on the quality and quantity of training data. This statement is true for both generative (machine learning classifiers) and information retrieval (or rule-based) agents.
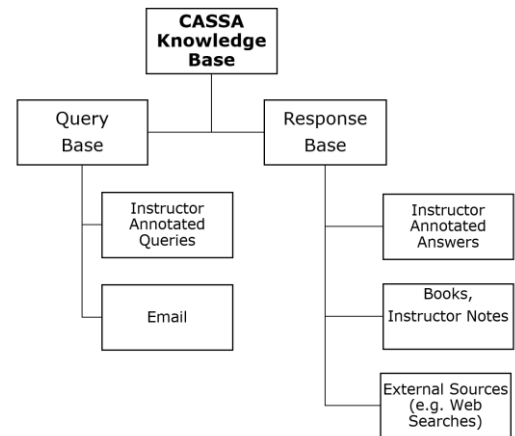

Figure 4: Conversational agent knowledge base

Though there are many ways of collecting, storing, and using the training data, this work relies upon a simplified version depicted in Figure 4.

I. Query Base: The instructor – to some extent - predefines what questions students are likely to ask in the course and creates a data set of such question-answer pairs. All the possible questions that might lead to the same response are coded under an Intent, and every Intent will have multiple questions/user examples under it. Basically, an intent categorizes the **user's intention, and the agent** contains possible hundreds of such intents (231 in this work). When a user writes or says something, the NLP engine (Dialogflow in this case), matches the user expression with the best Intent. Students are also very likely to ask questions over email. This can act as a rich source of query data that the agent would need to improve its accuracy of response. This work also verifies the fact that on a single topic, many students ask the same question in different ways and formats.

All these similar questions can be represented by the same Intent to generate a single, unified response. Figure 5 illustrates this process.
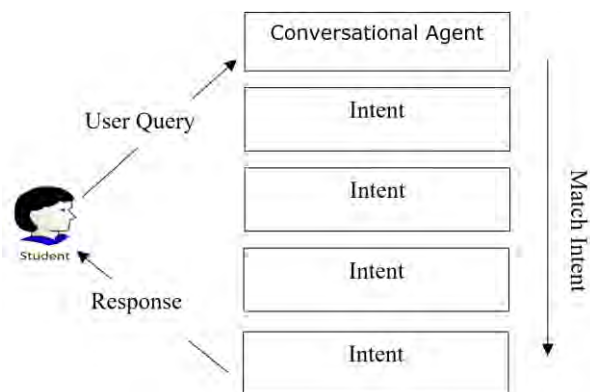
Figure 5: Intent matching

II. Response Base: This is where the responses/answers to the queries are stored. The responses to simple to mildly complex queries are stored in instructor annotated form, i.e., Intents where the instructor predefines the answer for a set of queries. The secondary source is searched if the response is not found in Intents, which includes textbook and instructor notes. Failing to find an answer in the first two sources, the agent sends the query to an external webhook.

d) External Search/Data Retrieval: The agent, as a last resort, also has the ability to query the web if it determines that a suitable response may not be present within it. This service was hosted on a web hosting platform named Heroku. Currently, only Google searches are supported. The agent extracts the **relevant entities from the student's** query, forms a search string, and relays it over to Heroku - a container based Platform as a Service (PaaS) - which runs a node.js service with Google search API enabled. Google search API responds with multiple URLs, and the first three URLs are presented to the student as a reference. This is not a sophisticated functionality at all. Students could easily search the web themselves and see the same URLs listed. The intention is to minimize student distraction; keep students engaged with the agent, and **improve the agent's knowledge base. This** query is moved to instructor annotated answers later on.

4. Conversational Agent Preliminary Evaluation

Evaluation of a chatbot is a complex problem. Many perspectives and methods, many of them subjective and often conflicting, can be utilized for its evaluation. For example, a chatbot can be evaluated on the basis of:
1. User experience
2. Information Retrieval Performance
3. Linguistic accuracy
4. Business perspective

As a direct result of a multitude of evaluation methods, numerous metrics, not necessarily mutually inclusive, have been proposed. SASSI, PARADICE, MIMIC are but a few such evaluation systems (Venkatesh et al., 2018). Some are lenient in awarding scores, while others are punitive. For example, Walker et al., 1997, proposed an attribute value matrix (AVM) to measure chatbot effectiveness. In this method, a script is created and is run through the chatbot. The desired responses are cataloged in a **"scenario key,"** while the **bot responses are** recorded in the AVM. A confusion matrix (M) is then constructed as:

$$\kappa = \frac{P(A) - P(E)}{1 - P(E)} \qquad (1)$$

where:
P(A) = proportion of AVM aggress with the correct response
P(E) = probability of agreement by chance
κ = kappa coefficient; bot that provides random answers, κ=0; for a human κ would ideally be 1.

Other subjective methods of chatbot evaluation are presented in other studies on chatbots (Bates, & Ayuso, 1991), (Kuligowska, 2015). It becomes readily evident that no single system is able to deliver a universal framework for chatbot evaluation. Moreover, catering to so many different perspectives is an expensive endeavor and out of the scope of this work. Hence, this work focuses on the evaluation of the chatbot from the perspective of information retrieval performance only.

Specifically, this work uses a confusion matrix similar to the one suggested by Walker et al., 1997, but instead of using κ as a metric, precision, recall, and F1-scores are calculated to evaluate the chatbot.
A confusion matrix visually answers questions like - when a student asks a question X which has an actual answer Y, what was actually predicted?

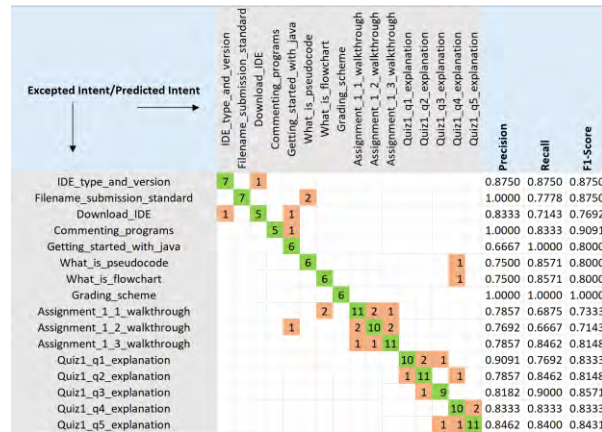The expected Intents are shown as rows, and the predicted Intents are shown as columns.

Figure 6: Confusion Matrix for Module 1

Figure 6 shows the confusion matrix for module 1 that has 16 Intents. Every Intent has multiple user examples, which are nothing but different ways of asking the same question. For example, a student can inquire about the IDE for the course. There are many ways this question be asked. Some of them are:

1. What is the IDE we are using?
2. **What's the IDE name?**
3. Can I use Netbeans IDE?
4. Tell me the IDE for this course?
5. What is the software to run Java programs?
6. What is the software we are using for this course?

These user examples are sent to the agent, and whenever the expected and predicted Intent is a match, the diagonal cell value is increased by 1, and these are called successful test cases. All other cell values that are not on the diagonal are failed test cases. Again, it must be noted that this method leaves out many other vital facets like evaluating chatbot looks, appearance, personality. These aspects may be evaluated in the future as the work on this system progresses.

At the time of writing, the agent had access to had 231 instructor annotated Intents, instructor class notes compiled as a .pdf, and a freely available Java textbook as a .pdf. Out of the 231 Intents, 104 were predefined by the instructor, and the rest were compiled from the questions asked by students on email over years of teaching this programming course. It should be noted that every Intent contains examples/queries that are written in different formats/ways but point towards the same response/answer. The distribution of Intents among different chapters/modules is listed in Table 3.

Module	No. of Intents
1	16
2	27
3	37
4	40
5	41
6	39
7	31
Total	231

Table 3: No. of Intents per module

The instructor annotated Intents if correctly matched with the user query, are the first line of **response. If the response isn't found in those** intents, the query is referred to instructor notes or the textbook, and then the web, in that order. The more such intents the agent has access to, the better the potential accuracy of the agent. Ideally, the number of intents should progressively expand as the course is taught multiple times over, and the new questions by the students, and previously unknown questions to the agent, are fed into the knowledge base.

Three performance metrics, namely precision, recall, and F1-score, were measured for every Intent. As can be seen, there are numerous ways of asking the same question. These ways are the instructor annotated queries or user examples. All these questions should match the same Intent, which in this case should be IDE_type_and_version. However, it is tough to achieve such perfect performance. For the sake of brevity, Figure 6 only displays the performance of the agent for Module 1 having 16 intents. The precision, recall, and F1-score are also shown in the three rightmost columns.

Averages of all 231 Intent performance scores were computed to mark the final performance measures of the agent. The results are listed in Table 4.

Performance Metric	Average Metric Scores for Seven Modules
Precision	0.7981
Recall	0.7856
F1-Score	0.7923

Table 4: Preliminary performance score of conversational agent

F1-Score below 0.80 is less than desirable, and F1-Score above 0.90 is considered good.

As a work in progress, the author believes that an F1-Score of 0.7923, though only slightly comforting, is a reasonable milestone in the preliminary agent development while acknowledging that a lot more training data and improvements are required to make this agent usable in live courses. See Appendix B for example conversations between the chatbot and a student. The integration with Dialogflow Messenger and Telegram is shown.

5. CONCLUSION AND FUTURE WORK

At an anecdotal level, the results indicate that it may be possible to affect the motivation levels of novice programmers using incrementally scaffolded instruction. Though there were no significant differences in the individual chapter quiz scores between the control and experimental groups, the experimental groups performed significantly better in the final exam. This came at the price of significant instructor overload. The integration of a helper chatbot with this technique is expected to reduce the instructor load. The initial preproduction performance of the conversational agent is undoubtedly below expectations but is expected to improve with more data and time. One of the ways the author intends to collect more data/user examples is to use the course chat forums and discussion boards for more questions asked by students to each other. The next step will be continuous training of the chatbot to achieve an F1-Score of at least 0.85, after which it will be opened for students to use.

To further mitigate the load on the instructor while maintaining the integrity of the technique, integrating an automatic grading system with the CASSA is proposed. An abstract schema of this system is shown in Figure 7.
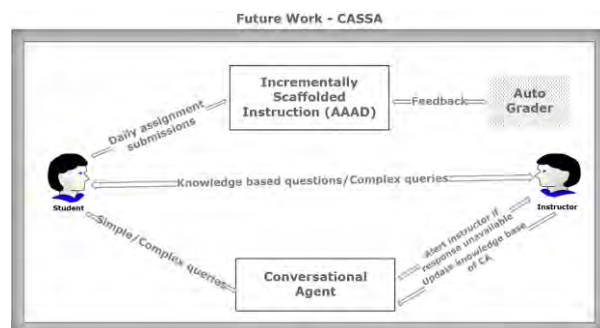

Figure 7: Integration of Auto Grader with CASSA

In closing, it would be too premature to consider the CASSA system as a workable method for affecting student motivation, given the significant challenges this system entails presently. The preliminary results, nevertheless, are encouraging and provide a solid direction for future research.

6. REFERENCES

Alexandron, G., Armoni, M., Gordon, M. & Harel, D. (2014). Scenario-based programming: Reducing the cognitive load, fostering abstract thinking. In Companion Proceedings of the 36th International Conference on Software Engineering pp. 311–320.

Ali, N., Anwer, M., & J., Abbas. (2015). Impact of Peer Tutoring on Learning of Students. Journal for Studies in Management and Planning, 1(2), 61-66.

Allan, V. H. & Kolesar, M. V. (1997). Teaching computer science: a problem solving approach that works. ACM SIGCUE Outlook, 25(1–2), 2–10.

Bates, M., & Ayuso, D. (1991). A proposal for incremental dialogue evaluation. Proceedings of the workshop on Speech and Natural Language - HLT '91.

Beaubouef, T. B. & J. Mason (2005). Why the High Attrition Rate for Computer Science Students: Some Thoughts and Observations. Inroads – The SIGCSE Bulletin, 37(2), 103–106.

Beavis, C. (2010). Literacy, Learning, and Online Games: Challenge and Possibility in the Digital Age. In Proceedings of the IEEE 3rd International Conference on Digital Game and Intelligent Toy Enhanced Learning. Piscataway, NJ: Institute for Electrical and Electronics Engineers.

Bennedsen, J. & Caspersen, M. E. (2007). Failure rates in introductory programming. ACM SIGCSE Bulletin, 39(2), 32–36.

Beutel, M. E., Klein, E. M., Aufenanger, S., Brähler, E., Dreier, M., Müller, K. W., Quiring, O., Reinecke, L., Schmutzer, G., Stark, B., & Wölfling, K. (2016). Procrastination, Distress and Life Satisfaction across the Age Range - A German Representative Community Study. *PloS one*, *11*(2), e0148054.

Brown, S. W., & Bennett, E. D. (2002). The role of practice and automaticity in temporal and nontemporal dual-task performance. *Psychological Research*, 66, 80–89.

Chang, C-W. (2009). Efficacy of interaction among college students in a web-based environment. Journal of Educational

Technology Development and Exchange, 2(1), 17-32.

Colvin, J. W. (2007). Peer tutoring and social dynamics in higher education. Mentoring and Tutoring, 15(2), 15-181.

Chang, C-W., Hurst, B., & McLean, A. (2015). **You're got mail: Stu**dent preferences of instructor communication in online courses in an age of advancing technologies. Journal of Educational Technology Development and Exchange, 8(1), 39-47.

Dawar, D., (2021). Towards Improving Student Expectations in Introductory Programming Course with Incrementally Scaffolded Approach. Information Systems Education Journal 19(4), 61-76.

Glover, J.A., Ronning, R.R. and Bruning, R.H.: 1990, Cognitive Psychology for Teachers, Macmillan, New York.

Goold, A., and Rimmer, R. (2000). Factors affecting performance in first-year computing. SIGCSE Bulletin 32, 39–43.

Howles, T. (2009). A study of attrition and the use of student learning communities in the computer science introductory programming sequence. *Computer Science Education*, 19(1), 1–13.

Kalchman, M., Moss, J., & Case, R. (2001). Psychological models for the development of mathematical understanding: Rational numbers and functions. In S. M. Carver & D. Klahr (Eds.), Cognition and instruction: Twenty-five years of progress (pp. 1-38). Mahwah, NJ, US: Lawrence Erlbaum Associates Publishers.

Kalchman, M., Moss, J., & Case, R. (2001). Psychological models for the development of mathematical understanding: Rational numbers and functions. In S. M. Carver & D. Klahr (Eds.), Cognition and instruction: Twenty-five years of progress (pp. 1-38). Mahwah, NJ, US: Lawrence Erlbaum Associates Publishers.

Kaplan, D. S., Liu, R. X., & Kaplan, H. B (2005). School related stress in early adolescence and academic performance three years later: The conditional influence of self-expectations. *Social Psychology of Education*, 8, 3-17.

Keller, J. M. (1983). Motivational design of instruction. In Instructional-Design Theories and Models: An Overview of their Current Status, C. M. Reigeluth, Ed. Lawrence Erlbaum Associates, pp. 383–434.

Kim, J. & Lerch, F. J. (1997). Why is programming (sometimes) so difficult? Programming as scientific discovery in multiple problem spaces. *Information Systems Research* 8(1) 25–50.

Kim, K. R. & Seo, E. H. (2015). The relationship between procrastination and academic performance: A meta analysis. Personality and Individual differences, 82, 26-33.

Kinnunen, P. & Malmi, L. (2006). Why students drop out CS1 course?. In Proceedings of the Second International Workshop on Computing Education Research (pp. 97–108). New York, NY: ACM.

Kuligowska, K. (2015). Commercial Chatbot: Performance Evaluation, Usability Metrics and Quality Standards of Embodied Conversational Agents. Professionals Center for Business Research, 2(02), 1-16. doi:10.18483/pcbr.22

Li, L., Finley, J., Pitts, J., & Guo, R. (2010). Which is a better choice for student faculty interaction: Synchronous or asynchronous communication? Journal of Volume 9, No. 1, September, 2016 11 Technology Research, 2, 1-12.

Mendes, A. J., Paquete, L., Cardoso, A. & Gomes, A. (2012). Increasing student commitment in introductory programming learning. In Frontiers in Education Conference (FIE) (pp. 1–6). New York, NY: IEEE.

Moors, A., & Houwer, J. D. (2006). Automaticity: A Theoretical and Conceptual Analysis. Psychol Bull, 132(2), 297-326.

Newman, R., Gatward, R. & Poppleton, M. (1970). Paradigms for teaching computer programming in higher education. WIT Transactions on Information and Communication Technologies, 7, 299–305.

Paas, F., Renkl, A., & Sweller, J. (2010). Cognitive Load Theory and Instructional Design: Recent Developments. Educational Psychologist, 38 (1), 1-4.

Robins, A. V., Rountree, J. & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education* 13(2) pp. 137–172.

Rogalski J. & Samurçay R. (1990). Acquisition of programming knowledge and skills. In J. M. Hoc, T. R. G. Green, R. Samurçay & D. J. Gillmore, eds., Psychology of Programming. London: Academic Press, pp. 157–174.

Scardamalia, M. and Bereiter, C. 1992. Text-based and knowledge-based questioning by children. *Cognition and Instruction*, 9: 177–199.

Sheard, J. & Hagan, D. (1998). Our failing students: a study of a repeat group. ACM SIGCSE Bulletin, 30(3), 223–227.

Steel, P. (2007). The nature of procrastination: A meta-analytic and theoretical review of quintessential self-regulatory failure. *Psychological Bulletin, 133*(1), 65–94.

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science*, 12(2), 257–285.

Sweller, J. (1994). Cognitive load theory, learning difficulty, and instructional design. *Learning and Instruction*, 4(4), 295–312.

Venkatesh, A., Khatri, C., Ram, A., Guo, F., Gabriel, R., Nagar, A., Raju, A. (2018). On Evaluating and Comparing Conversational Agents. ArXiv:1801.03625 [Cs].

Walker, M. A., Litman, D. J., Kamm, C. A., & Abella, A. (1997). Paradise. Proceedings of the 35th annual meeting on Association for Computational Linguistics.

Watson, C. & Li, F. W. (2014). Failure rates in introductory programming revisited. In Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education (pp. 39–44). New York, NY: ACM.

Weinberger, M. (2017). Why Amazon's Echo is totally dominating - and what Google, Microsoft, and Apple have to do to catch up.

Winslow L E (1996) Programming pedagogy – A psychological overview. ACM SIGCSE Bulletin, 28(3), 17–22.
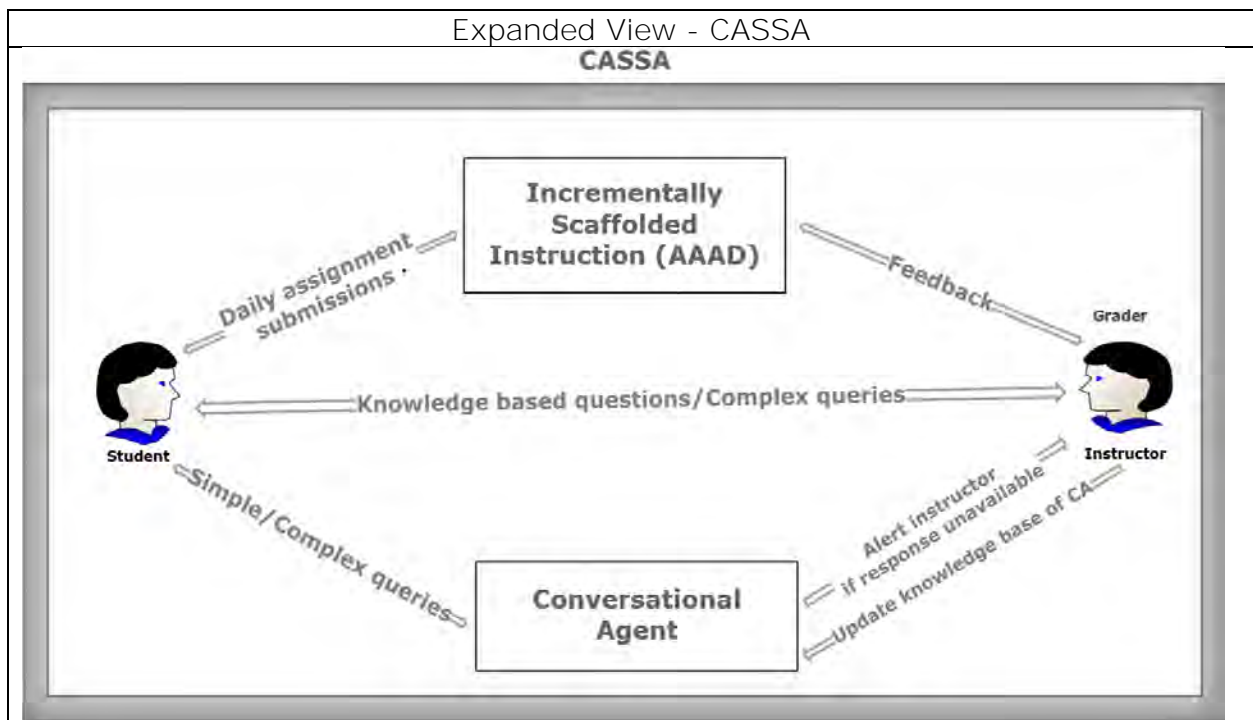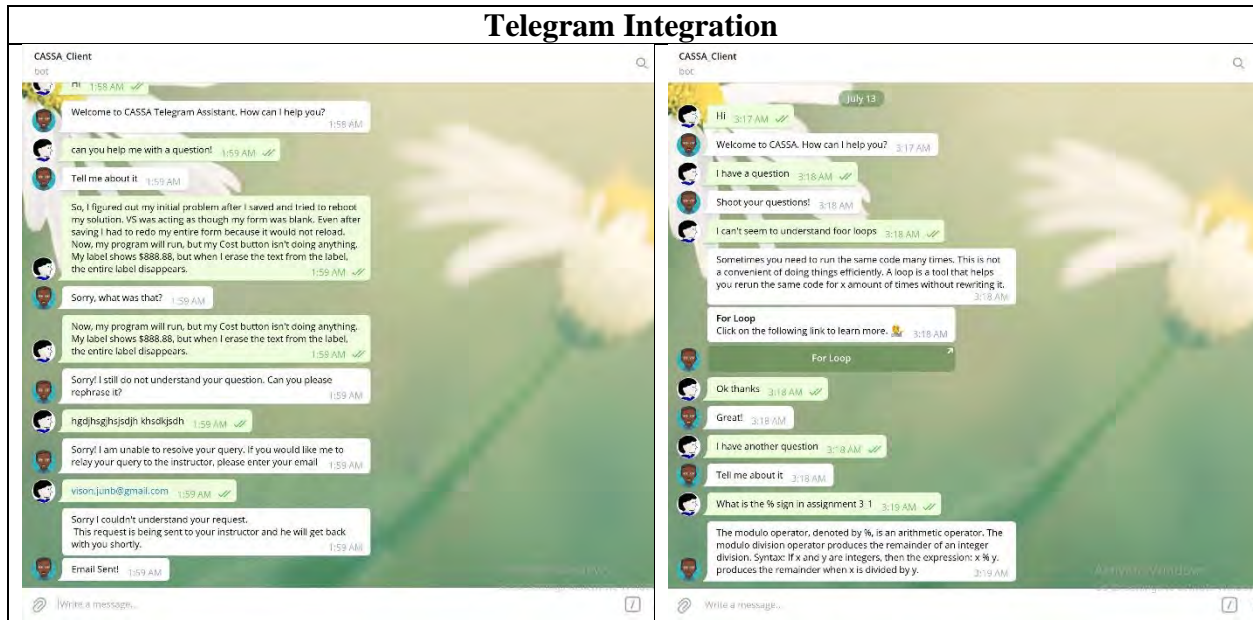
APPENDIX A

Table X: Increment in cognitive load with time

Assignment No.	Description	Concepts Tested	Cognitive Load
1	Write a method printS that takes a string as an input and prints it to the console.	Rudimentary method writing.	Low
2	Modify the above method printS and enable it to take another argument, an integer, *n*. The method then prints the string *n* times in a line.	Method writing, method calling, method modification.	Low
3	Reuse printS to print a user entered string *n×n times;* i.e., a square with each element as the string	User input, loops, method writing, method calling	Medium
4	Reuse printS method to print a right angle triangle in terms of user entered string	User input, loops, method writing, method calling, Problem solving	Medium
5	Reuse printS to print a pyramid in terms of user entered string	User input, loops, method writing, method calling, Problem solving	High

APPENDIX B

Dialogflow Messenger Integration

Telegram Integration

Expanded View - CASSA
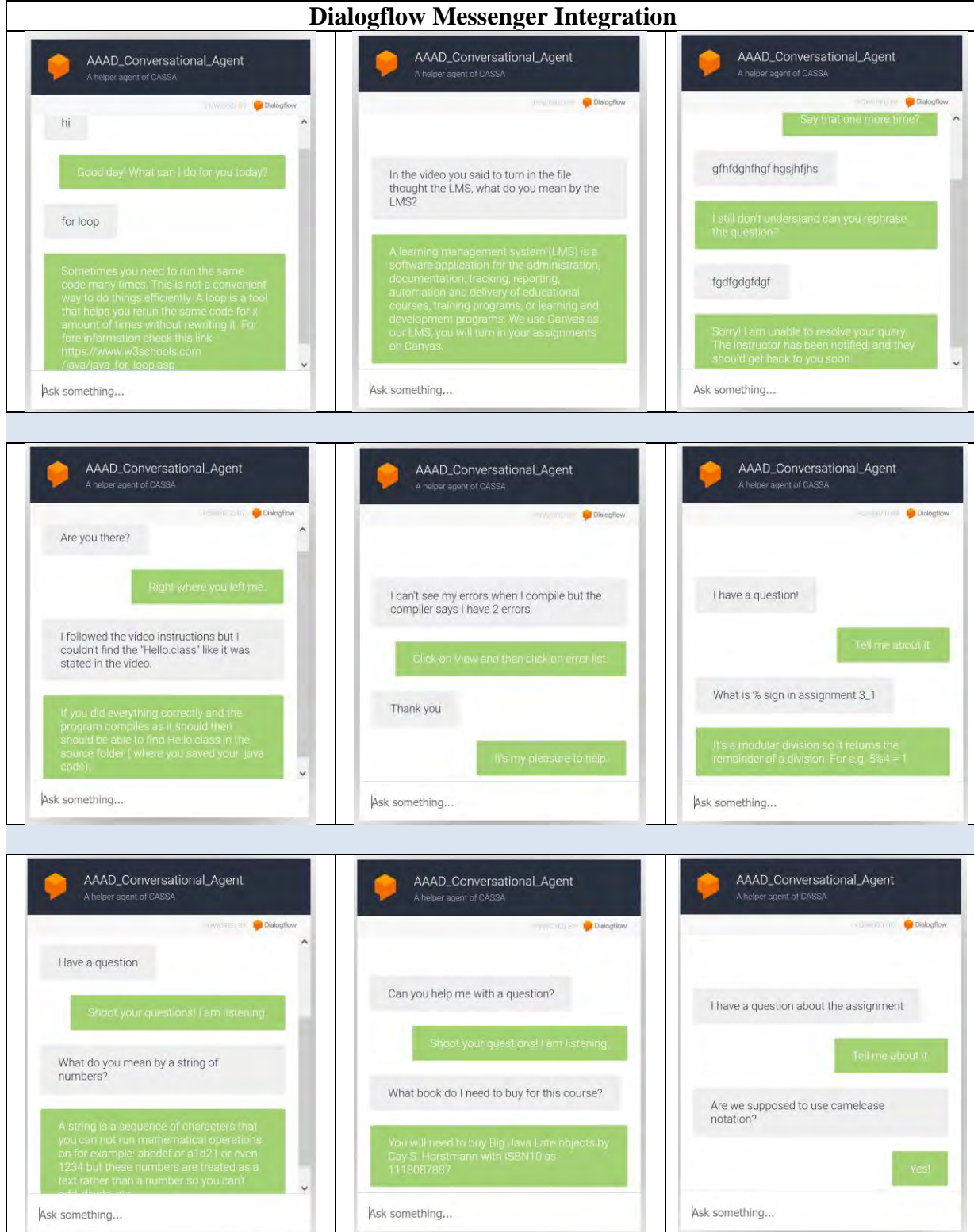
APPENDIX C

CSE 174: Student experiences with multiple assignments

| English   ▼ |

SURVEY INSTRUCTIONS

Dear CSE 174 Student,
This short survey is designed to ask you about your experiences in this course, specifically about an assignment a day (AAAD) format, where, for each chapter, you did one assignment per day (or more) depending upon the difficulty level of the assignment(s). Please consider each question carefully. Your participation is much appreciated.

Student Resources

Did the daily assignments prepare you for the last (concluding) assignment of the module?

| Definitely yes | Probably yes | May be | Probably not | Definitely not |

Did the daily assignments prepare you for the midterm and final exams?

| Definitely yes | Probably yes | May be | Probably not | Definitely not |

How difficult was it for you to *schedule time* every day to complete the daily programming assignment?

| Extremely easy | Moderately easy | Slightly easy | Neither easy nor difficult | Slightly difficult | Moderately difficult | Extremely difficult |

How difficult was it for you to *complete* the daily assignment?

| Extremely easy | Moderately easy | Slightly easy | Neither easy nor difficult | Slightly difficult | Moderately difficult | Extremely difficult |

Overall, how much time did you spend on completing the daily assignment?

A great deal   A lot   A moderate amount   A little   None at all

How did the daily assignment make you feel about your ability to complete the course satisfactorily?

Much better   Moderately   Slightly better   About the   Slightly worse   Moderately   Much worse
                  better   same   worse

Overall, how useful is a daily assignment for learning computer programming?

Extremely   Moderately   Slightly useful   Neither useful   Slightly   Moderately   Extremely
useful   useful   nor useless   useless   useless   useless

Given an option, what mode of practice work would you prefer for this course?

One long and possibly difficult assignment each week

One small and possibly easy to medium difficulty assignment every day that builds on previous concepts

No preference

Block 2

How did doing multiple assignments effect your stress levels?

It made it easy to manage overall stress as the assignments were gradually increasing in difficulty

It increased my stress as I had to do many assignments

It made no difference

Did having a programming assignment everyday format encourage you to practice more on your own?

It positively pushed   It made me practice   It made me practice   I would have   It made me practice
me to practice   moderately better   slightly better   practiced a lot   less
much better   regardless of this
                                                                         format

	Extremely well	Very well	Moderately well	Slightly well	Not well at all
Outcome 1: Use and describe a contemporary programming language and programming environment (IDE) like Dr. Java.	O	O	O	O	O
Outcome 2: Identify and eliminate errors in programs	O	O	O	O	O
Outcome 3: Specify, trace, and implement programs written in a contemporary programming language like Java that solve a stated problem in a clean and robust fashion	O	O	O	O	O
Outcome 4: Solve programming problems using a procedural approach i.e. divide your program into methods	O	O	O	O	O
Outcome 5: Describe, trace, and implement basic algorithms like linear search, binary search etc.	O	O	O	O	O
Outcome 6: Apply and communicate information that they read from technical sources such as APIs like Scanner etc.	O	O	O	O	O