Interactive Learning Environments
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/nile20

Evaluation of learning environments for object-oriented programming: measuring cognitive load with a novel measurement technique

Murat Pasa Uysal[a]
[a] Department of Computer Technologies, Ufuk University, Ankara, Turkey
Published online: 18 May 2015.

To cite this article: Murat Pasa Uysal (2015): Evaluation of learning environments for object-oriented programming: measuring cognitive load with a novel measurement technique, Interactive Learning Environments, DOI: 10.1080/10494820.2015.1041400

To link to this article: http://dx.doi.org/10.1080/10494820.2015.1041400

Interactive Learning Environments, 2015
http://dx.doi.org/10.1080/10494820.2015.1041400

Routledge
Taylor & Francis Group

Evaluation of learning environments for object-oriented programming: measuring cognitive load with a novel measurement technique

Murat Pasa Uysal*

Department of Computer Technologies, Ufuk University, Ankara, Turkey

(*Received 18 November 2013; final version received 6 April 2015*)

Various methods and tools have been proposed to overcome the learning obstacles for Object-Oriented Programming (OOP). However, it remains difficult especially for novice learners. The problem may be not only adopting an instructional method, but also an Integrated Development Environment (IDE). Learners employ IDEs as a means to solve programming problems and an inappropriate IDE may impose additional cognitive load. Therefore, this quasi-experimental study tried to identify the cognitive effects of a more visually supportive and functional IDE. It was explored by the functional near-infrared spectroscopy method, which is a relatively new physiological tool for measuring cognitive load. Novice students participated in the study in two experimental groups and they were required to write a Java application using two different IDEs. The results indicated a significant difference between the experimental groups and the findings are discussed in view of the principles of Cognitive Load Theory and Multimedia Learning.

Keywords: integrated development environment; object-oriented programming; cognitive load; fNIRS; multimedia learning

1. Introduction

Object-oriented programming (OOP) has become a common mode of introductory computer programming, and therefore, almost every university puts it somewhere in its curriculum. This is partly because OOP is widely advocated paradigm. It is also similar to our view of the real world and allows quickly building programs from reusable components. The three-decade research indicates various methods and tools for teaching OOP (Eckerdal, 2006). However, it remains difficult, especially for novice learners. In addition to inherent complexity, the problems on teaching OOP can be grouped into several categories, such as instructional methods, programming contexts and learners' attitudes to OOP (Xinogalos, 2010). It is thought that the problem may be not only adopting an instructional approach, but also an Integrated Development Environment (IDE). Programming requires the use of IDE, and therefore, an inappropriate environment may introduce additional complexities to learning OOP (Kolling, 1999; McIver, 2002; Miller, Pane, Meter, & Vorthmann, 1994; Pane & Myers, 1996).

An IDE can be defined as one piece of software environment where a programmer interacts with that, and the editing, compilation, debugging and visualization functionalities are

*Emails: mpuysal@gmail.com, murat.uysal@ufuk.edu.tr

provided. In early days, a text editor and a compiler could be sufficient for the practice part of a course. Developments in computer technologies have much contributed to the pedagogy of OOP. Today, IDEs can vary from simple text editors and command-line compilers to fully interactive IDEs. Contemporary courses can directly start with teaching complex skills from the very beginning, such as object-oriented design, testing and code reusing. IDEs require interactions, use of multiple interfaces and manipulation of different types of files. Therefore, learning context has become more complex, especially for beginners. Although some studies indicate the importance of IDEs for learning (Whittle & Cumming, 2000), the majority of current IDEs still focus on the expert programmers' needs or software development processes. It is indisputable that the priority should be on supporting instruction IDEs can play an important role during this process. To that aim, the main emphasis should be put on its instructional support when selecting or evaluating an IDE for an introductory course (Kordaki, 2010).

During evaluation process, it is explored how an IDE (1) simplifies the programming process, for example, language and typing code; (2) provides support for learners, for example, by structuring code or visualization; and (3) creates a meaningful and learner-centered context, such as problem-solving and social learning. Most of the evaluative techniques address the mechanics of programming after the completion of a task (Kelleher & Pausch, 2005). They usually focus on the interaction between a user and the software itself by surveying, which is lacking an insight into the real-time experiences on IDEs. However, OOP requires effective use of cognitive processes, and therefore, learners mainly employ IDEs as a means to solve a programming problem. Thus, approaching IDEs from a cognitive dimension, and exhibiting the cognitive consequences, can also help in making design decisions on introductory courses. Furthermore, learners' limited cognitive capacity and the cognitive load concept is an effective factor for learning, and therefore, a research focus should be given to the cognitive issues, specifically to the measurement of cognitive load that can be imposed by IDEs.

As relevant, different cognitive load measurement techniques have been utilized, of course, each of them adds to the understanding of learners' cognitive state. However, a survey of literature on cognitive load measurements shows that most of the studies include subjective or self-reporting techniques. There are a few attempts made to measure cognitive load using direct methods. Fortunately, the latest developments in technology improved the understanding of cognitive load notion, and they are presented in the following sections of this paper. However, being a relatively new and promising method, neuroimaging is one of them with a potential to provide direct, precise and objective information for the evaluation of IDEs. Since it can be applied in learning contexts, a variety of publications advocate the integration of neuroimaging and instructional studies (Berninger & Corina, 1998). IDEs have been usually explored by using subjective and indirect measurement techniques, such as self-reporting and performances. Therefore, neuroimaging methods can enable researchers to gain a different insight not only into the cognitive processes, but also into the evaluation of learning environments with direct and precise measures.

Another important point is the multimedia aspects of IDEs and their cognitive implications. As being the combination of texts, pictures and graphics with interactive functionalities, an IDE can be regarded as a kind of multimedia learning (ML) environment. They extensively make use of visual and verbal information, and the interference of this information may present additional source of cognitive overload as a fundamental challenge. While the principles of Cognitive Load Theory (CLT) have been applied with considerable success in the associated field of ML, they appear to have not received much interest in the

research area of computer science education and IDEs (Shaffer, Doube, & Tuovinen, 2003). Thus, it would make sense to incorporate ML into the evaluation of IDEs based on CLT.

Different studies have demonstrated the importance of IDEs for novice learners and their instructional assessments (Kordaki, 2010; Pane, & Myers, 1996). However, the cognitive support of OOP IDEs has yet to be determined with direct, objective and precise measures. Therefore, the main focus of this study is to investigate students' cognitive load by using a direct measurement method when participants use two different Java IDEs. Our argument is that an instructional IDE should not impose extraneous cognitive load for OOP learners. Within this context, neuroimaging techniques would provide the empirical evidences needed for the objective evaluation of instructional IDEs in view of CLT and ML. Thus, this is explored by using functional near-infrared spectroscopy (fNIRS), which is a relatively new physiological method for measuring cognitive load. The following sections present the related work, background theory, research method, results and discussion parts of this paper.

2. Related work

Studies reporting the cognitive aspects of OOP can be divided into several categories. The first has been addressed to teaching or learning approaches to computer programming, which aims at the effective processing of information (van Gog, Kester, Nievelstein, Giesbers, & Paas, 2009). The second category explores the tools supporting cognitive processes and it includes the methods for mental effort measurement (Paas, van Merrienboer, & Adam, 1994). The third relates computer programming with different cognitive processes, and focuses on the psychology of programming and its implications (Renumol, Janakiram, & Jayaprakash, 2010). When it comes to IDEs, there are models or frameworks proposed for evaluating programming environments (Green & Petre, 1996). Evaluation is usually made by collecting the data on usability, performances, or responses to the IDEs as it can be either comparative or standalone evaluation (McIver, 2002). Standalone evaluation provides considerable aid for the decisions on a particular environment. However, it is usually valid in its own context and more often used for the improvement of IDE itself. Comparative evaluation of multiple environments is difficult as there may be interacting variables, such as course design, difference in instructors and so on. This type of evaluation is suggested for the studies working with a small sample in well-controlled settings.

With respect to comparative evaluation, Klinea and Seffah (2005) presented the results of three successive empirical studies on IDE usability. They conducted unstructured interviews, used questionnaires and observed the novice behaviors when trying to solve common types of OOP tasks in the laboratory. Through the use of cognitive walkthrough technique, the visually unsupportive IDEs did not assist in program comprehension and poor affordances in the IDE user interfaces (UIs) were identified as the primary usability problems. Additionally, Green and Petre (1996) presented a cognitive dimensions framework as a standalone evaluation method for visual programming environments. The dimensions were defined as the discussion tools and the descriptions of an artifact–user relationship. The primary purpose of this framework was to lay out the cognitivist's view of the design space in a coherent manner, and to exhibit cognitive consequences of making particular design choices for a programming environment.

As being highly relevant to our research, Girouard et al. (2010) reported the findings of an experimental research, in which fNIRS was used as a means to measure the cognitive load experienced by participants working on a task using the software with different UIs. Their study also investigated the feasibility of recognizing mental cognitive states with

the fNIRS technology, and this method's practicality and applicability in desktop environments. Thus, they proposed a conceptually separate cognitive load notion for a task requiring the use of a computer. That is, the total cognitive load was composed of the task load itself plus the load attributable to the complexity of the UIs. Consequently, they found that the uninformed hyperspace location UIs caused participants to experience higher cognitive loads than the informed hyperspace location in UIs.

The literature on the studies exploring the IDEs in a way that would facilitate novice learning based on the principles of CLT and ML is also insufficient (Moons & Backer, 2013). For example, Pane and Myers (1996) report the usability issues in the design of novice programming systems, and they provide important ML and cognitive guidelines by organizing the research about the IDEs for novice programmers. For minimizing cognitive load when using IDE, they emphasize recognition rather than recall. Thus, objects and actions in an IDE should be visible and easily retrievable for cognitive support so that the learners do not have to remember information from one phase of programming to another. Hence, novices need a process to guide programming, and therefore, programmers should be allowed to work directly in plan terms. They point out that an IDE, which assists in these issues, can yield improvements in support for program generation as well as reducing cognitive overload.

The review of fNIRS-related works also shows that there is a considerable body of knowledge supporting the idea that the functional hemodynamic changes (increase and decrease in neural activities) are associated with the working memory and cognitive processes (Ayaz et al., 2012; León-Carrióna et al., 2010). The ones particularly similar to our experimental study explored the physiological measures, and associated them with different variables, such as performance, tool and task difficulty. These studies also indicate that human performance, maintenance of ongoing information and use of a software system can be assessed directly by the fNIR technology (Ferrari & Quaresima, 2012; Hirshfield et al., 2009).

3. Background theory

3.1. *Cognitive load theory*

The cognitive load concept is extensively addressed in the CLT which provides guidelines for the presentation of information to optimize learners' cognitive performances (Sweller, van Merriënboer, & Paas, 1998). Human mind is composed of three types of memory: sensory, working and long-term memory. The stimuli received through the sensory organs like eyes, ear, and so on are processed in the working memory. Then, the long-term memory stores the information permanently in the form of schemas or mental models and so they can be recalled. For example, the long-term memory contains OOP semantics, syntax and concepts as the building blocks of programming plans in schemas. The CLT suggests that the instructional goal should be the construction and automation of these schemas in a useful way. However, the working memory has a limited capacity and it filters both the contents and functioning of the long-term memory. The cognitive load affects this capacity, and therefore, it represents the load imposed on the cognitive system when performing cognitive tasks (Paas, Tuovinen, Tabbers, & Gerven, 2003).

There are three types of cognitive load: intrinsic, extraneous and germane cognitive load. The intrinsic cognitive load involves the inherit complexity of instructional contents, and it is not directly manipulated. Structures of a programming language, paradigm or the syntax rules form the intrinsic load. The germane cognitive load includes the mental effort

contributing to knowledge acquisition, such as reviewing worked examples or code snippets. The extraneous cognitive load is generated by an ineffective instruction with poorly designed strategies, activities, or tools. The intrinsic, extraneous and germane cognitive load is additive in working memory. If an IDE possesses design elements that adds extraneous load to the intrinsic load, then there will be little capacity left for the germane load, and therefore, it will take longer to acquire desired skills and knowledge (van Merriënboer & Paas, 1990).

3.2. *Cognitive load measurement*

Determining the cognitive load is challenging due to its multidimensional character and the complex interrelationships between performance, cognitive load and cognitive effort. Sweller et al. (1998) specifies three major categories for cognitive load measurement: (1) task and performance-based measurement, (2) subjective measurement and (3) physiological measurement. The subjective measurement is based on the assumption that people can introspect on their cognitive processes. The task and performance-based techniques use task characteristics (number of tasks, elements, etc.) and performance levels (number of errors, execution time, etc.) to obtain information on cognitive effort. The physiological techniques include the measures of brain activity or heart rate, which assume that the changes in cognitive functioning are reflected in physiological measures.

Brünken, Plass, and Leutner (2003) classifies these techniques into two groups: (1) objective/subjective and (2) direct/indirect methods. In the first, the observations of behaviors, that is, the heart rate or the signals from the brain, are defined as objective measures while self-reported data are accepted as the subjective measures. The second group is based on the relationship between the phenomenon observed by the measure and the actual attribute of the research interest. For example, the heart rate is a measurement, which is indirectly linked to cognitive load, may be the result of a learner's emotional response to a learning material. As being a physiological technique, fNIRS is a direct and objective measurement method since it straightway measures and monitors the brain activities.

3.3. *fNIRS method*

Theoretical foundation of the fNIRS system goes back to the Jobsis (1977) study, which reported that the light in near-infrared range (NIR) could be used for measuring the brain activities. If the light in NIR diffuses through scalp and skull, the functional state of tissue is influenced by the changes in electrochemical activities and blood levels. Thus, this affects the optical properties of the human brain. When the NIR range of spectrum is introduced at the scalp, the injected photons follow different paths. Some of them are absorbed by skin, skull and brain, and the others follow different patterns as a result of the scattering effect of tissue. The spectrum of light is analyzed, and the backscattered photons are interpreted as changes in blood chromophores. Therefore, the information about the blood volume and tissue oxygenation can be an evidence of functional hemodynamic activities in the dorsolateral prefrontal cortex of the brain (Izzetoglu, Bunce, Onaral, Pourrezaei, & Chance, 2004). That is, these changes indicate the increase or decrease of neural activities, and they are interpreted as the outcomes of cognitive activities.

There are other optical techniques to monitor the changes in human brain. Magnetoencephalography (MEG), functional magnetic resonance imaging (fMRI) and positron emission tomography (PET) are well-known methods (Strangman, Culver, Thompson, & Boas, 2002). However, these techniques may possess important constraints. For example, the participants

can be subjected to potentially harmful equipment. Some of the techniques may be highly expensive. Moreover, they confine participants to restricted positions since the equipment is highly sensitive to motion artifacts, and so these techniques may not be applicable to the classroom. Therefore, fNIRS is accepted as more functional than the other physiological methods (Ayaz et al., 2012; Izzetoglu, Bunce, Izzetoglu, Onaral, & Pourrezai, 2007).

As can be seen in Figure 1, the fNIRS system used in this study is composed of online and of offline components (Izzetoglu et al., 2004). During online measurements, subjects wear a headband receiving the light reflected from the tissue under forehead. This is a flexible sensor with a circuit board covering the entire forehead of a participant, and it is connected to a control box with a white cable (Figure 1). The control box and its power supply form the data acquisition component. The computer for the analysis software with a big screen monitor constitutes the data analysis and presentation system. The processing software visualizes and records the raw data by calculating the values of oxygenated and deoxygenated hemoglobin molecules relative to a baseline. Finally, an offline testing and analysis platform is used for filtering, processing and presentation of post-experimental data (Izzetoglu et al., 2007).

3.4. *Multimedia learning*

Multimedia is defined as the combination of text, picture, sound or video, and it is suggested that ML occurs when learners can construct effective mental models from words and pictures. Mental models are accepted as the internal representations of the external world, and human uses them for making decisions in different circumstances. Johnson (1983)

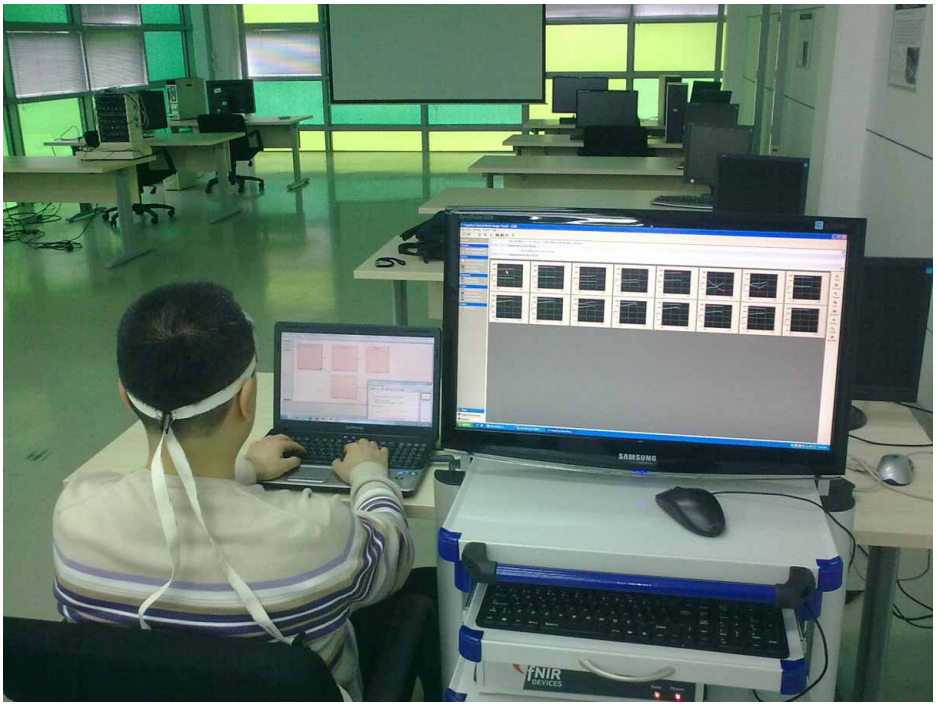


Figure 1. The fNIRS system and experimental environment. Source: Author

describes a mental model as the process of solving deductive reasoning problems. They provide people with information on how systems work, and they are built on prior knowledge and experiences, and rely on cognitive problem-solving skills. Thus, mental models' power of expressing abstractions can make computer programming understandable.

As being relevant to the mental model notion, Cognitive Theory of Multimedia Learning (CTML) is based on three assumptions (Mayer, 2009). First, the dual-channel assumption accepts the idea that humans have separate channels for processing visual–pictorial and auditory–verbal representations. Second, the limited capacity assumption is the idea that only a small piece of information can be actively processed at one time in each channel. Third, the active processing assumption suggests the idea of meaningful learning. By using different information processing channels, the humans (1) select relevant words or images for processing in working memory; (2) they organize the selected words or images into a verbal or pictorial model; and finally, (3) they integrate the verbal and pictorial mental representations with prior knowledge. In view of CTML, multimedia instruction is regarded as the presentation of words and pictures to foster learning. It takes the idea that meaningful connections are built between words and pictures for understanding.

According to CTML, "students learn more deeply than they could have with pictures or words alone" (Mayer & Moreno, 2003). It also requires learners' attendance to the significant aspects of learning materials, and mentally organizing them into coherent cognitive structures. However, one challenge is the potential for cognitive overload and so the design principles have been proposed to reduce this overload. In an example, one is the "spatial contiguity" principle, which proposes that "people learn better when corresponding words and pictures are placed near each other rather than far from each other on the screen" (Mayer, 2009). The "temporal contiguity" is the presentation of corresponding words and pictures at the same time. Another principle asserts that learning is better from words and pictures than from words alone. Meaningful learning occurs when a learner mentally organizes the presented material into coherent cognitive structures, and "it is reflected in the ability to apply what was taught to new situations." The verbal and visual working memories play the central role at this process. However, their limited capacity and interferences between visual and verbal memory present extraneous load. Thus, exceeding the available capacity can lead to cognitive overload in which some of the information may not be processed or the cross-domain information may be filtered out.

Computer-based ML environments can "offer a potentially powerful venue for improving student understanding" (Mayer & Moreno, 2003). IDEs make use of texts, pictures and graphics extensively, and therefore, they can be regarded as a type of ML environment. Although the prescriptions of CTML may not directly address IDEs, it is thought that they may provide important guidelines. The visually more supportive IDEs, such as BlueJ, may fit the principles of ML design better than mainly text-based IDEs, such as JCreator LE; however, their cognitive aspects are often neglected. Consequently, it may be suggested for approaching the visually supportive IDEs in view of the principles of CTML, and this may provide a deep insight into the evaluations of instructional IDEs.

4. Method

4.1. *Research design*

This was a quasi-experimental research with a post-test-only design. The research hypothesis was: "when compared to a mainly text-based IDE, a more visually supportive and

functional IDE would make significant differences between students in terms of their cognitive load represented by average oxygenation changes." A cluster random sampling method was used, and therefore, the students were randomly assigned to two existing experimental groups. The average oxygenation changes were investigated when participants were using BlueJ or JCreator LE IDE. As representing the total cognitive response to the IDEs, these measures constituted the dependent variable of the research design.

4.2. *Participants*

The participants in this study initially consisted of a total of 20 male students, with age ranging from 25 to 28. They voluntarily participated in the research in two study groups. They attended a graduate program and completed the "OOP with Java" course in the spring semester of 2011–2012 academic year. Although the students were familiar with BlueJ and JCreator LE, supplementary lectures were given to make sure that they mastered the IDEs. They signed consent forms and were allowed to leave the study at any phase. However, one student left the study, and the fNIRS experiment was conducted with 19 participants.

4.3. *IDEs for teaching OOP*

The educational tools developed for supporting the instructional process of the OOP was in the scope of this study. Thus, the selection of the experimental IDEs was grounded on the results of a previous research (Uysal, 2014). In addition to exploring the attributes, this study mainly aimed to identify an instructional IDE for an introductory OOP course. Within the study, (1) a list of instructional IDEs was formed (BlueJ, DrJava, JCreator LE, jGRASP and Geany); (2) the evaluation criteria visual nature (VN), functionality (FN), ease of comprehension (EC), paradigm support (PS) were determined and applied to these IDEs (Kiper, Howard, & Ames, 1997); (3) BlueJ and JCreator LE were selected, and then two groups of students experienced them; and finally, (4) semi-structured interviews were conducted to explore how these IDEs were perceived by the students. The data were analyzed by Verbal Analysis Technique (Chi, 1997), and the results were discussed in view of the evaluation criteria. According to the results, only for the visual nature criterion was there enough evidence to conclude a difference in the means at $\alpha = .05$ level of significance ($z = -2.398$, $p = .016$). Although the BlueJ interviewers' mean ranks for other criteria (FN, EC and PS) were also higher than the JCreator LE interviewers', the differences were not statistically significant. The findings implied that the learners considered the visual nature of BlueJ as relatively more supportive for learning.

As being one of the experimental IDEs, BlueJ (Appendix 1) has been specifically designed for introductory teaching OOP (Kolling, 1999). It places a special emphasis on interaction and visualization to create an interactive environment that encourages exploration and testing objects. Its wizards help learners to create classes and implement interfaces. The unique nature of BlueJ may be its UIs supporting a much greater degree of visual interaction than other IDEs. The UML-like interfaces help learners apply complex OOP concepts, for example, inheritance and polymorphism, to their programs before talking about the detailed Java syntax. Learners can directly interact with single objects of any class, and execute methods using the interfaces. Objects can be directly instantiated from classes without writing code and their states can be inspected.

The other experimental IDE, JCreator LE (Appendix 2), is also for Java programmers of every level, and focuses on programming rather than rapid application development. It is

designed to provide developers with an easy-to-use environment for creating applications. The tab-based interface allows learners to move from one file to another. It provides necessary tools for editing and makes code writing easy. Code snippets, keyword completion and automatic suggestions improve coding speed. The Class Wizard enables creating new classes and implementing interfaces. Learners can manage breakpoints; debug files and project. It is possible to view variables, and monitor the threads to ensure that the code is working as it should. The customizable UIs make JCreator LE one of the most preferred instructional IDEs.

4.4. *Experimental controls*

As stated before, there are studies providing the empirical evidences for fNIRS as a measurement method of cognitive load experienced by users working on a task with software and UIs (Girouard et al., 2010). However, experiencing or detecting high cognitive load when a learner works with a system may not necessarily mean a bad thing. It could indicate that the user is deeply involved with the task, or s(he) has become overwhelmed by the task complexity. Therefore, several experimental controls were employed both to eliminate the differential influence of extraneous variables and to make the experimental groups the same on the variables that might have an influence on the cognitive load.

Three controls were consisted with the Shneiderman's semantic/syntactic model that describes how people interact with computers (Shneiderman & Plaisant, 2005). The syntactic knowledge is the device-dependent details of how to use system, which also presents the cognitive effort required for the use and interpretation of the UIs. The semantic knowledge involves the cognitive effort expended by a user to complete a given task (Girouard et al., 2010). The total cognitive load is composed of these two variables. However, we elaborated this model based on the principles of the CLT. Accordingly, the semantic cognitive load notion was divided into the intrinsic load and the germane load while the syntactic cognitive load notion was associated with the extraneous load (Hirshfield et al., 2009). Thus, three additive variables (van Merriënboer & Paas, 1990) were determined for the total average oxygenation changes as shown in the formula (1): The experimental OOP task (intrinsic load); the participants' conscious cognitive activity based on their programming knowledge (germane load); and the external constraints (extraneous load), such as the IDE or hardware.

$$V_{\text{Total}}(\text{fNIRS measures}) = V_{\text{Intrinsic}}(\text{task}) + V_{\text{Germane}}(\text{knowledge \& activity}) + V_{\text{Extraneous}}(\text{IDE}). \quad (1)$$

As one of possible influencing variables, the participants' knowledge could be a confounding variable. Therefore, an academic achievement test was given before the sessions to see whether they acquired required OOP skills and knowledge. The test was prepared by the researcher and it included the application of tasks and basic concepts, and it was also similar to that in the fNIRS experiment. The faculty members evaluated the test for the content and face validity, and the results indicated the participants' desired levels of competencies in OOP (Table 3). This meant that the participants acquired the knowledge required for the experimental task, and therefore, possible differences in the average oxygenation changes would not be attributed to the participants' expertise level.

Additional controls were also used to avoid possible extraneous load during the experiment. The first was the presentation of fNIRS system to the participants as a noninvasive

tool in the introduction sessions before conducting the experiment. The second control was the design of a simple experimental task to enable the participants to focus mainly on the IDEs. The third was the participants' individual programming styles that might be adopted during the experiment. Usually, novice learners may not prefer a linear or straightforward style and they can jump from a high level, that is, class design, to a low level, that is, method implementation. Beginners frequently revise what they have written so far, and their programming styles can easily deviate from a top-down to bottom-up style vice versa (Bellamy & Gilmore, 1990). Therefore, the order of coding was left to the participants to provide flexibility.

The fourth control was that the participants used their own notebooks in the experiment as in the course. The reason for using personally owned devices was to prevent extraneous load that would stem from a possible anxiety due to unknown hardware and software. This decision was grounded on Technology Acceptance Model (TAM), specifically on the use of laptops in education (Davis, 1989; Moran, 2006). According to the TAM, the perceived ease of use is defined as "the degree to which user expects the system to be free of effort." Unplanned or inappropriate changes in instructional tools can easily affect students' perceptions on the perceived usefulness of a tool, let alone its perceived ease of use. Finally, the experimenter controlled some of the attributes of laptops before each fNIRS session. This was to ensure that any of the laptops would not significantly violate the visual interface principles, though they might not be considered problematic by the participants. For example, unnecessary scrolling during program development would cause extraneous cognitive load because of a change in the locus of focus, and then the participants even might not be aware of that.

4.5. *fNIRS procedures*

The experimental protocol required the participants to write a Java application simulating a calculator with a simple UIs. The participants had to apply core concepts of OOP (Appendix 3). They were initially seated in front of their laptops installed with related Java IDEs. They had to complete the experimental task with no time constraint. The experimenter read the OOP task, and answered the questions without giving any clue pertaining to the programming task and techniques. This was to clarify everything before the start of fNIRS measurements. For avoiding possible motion artifacts, the task document was also placed at an eyesight level to provide a comfortable reading position. Correct application of the sensor headband on the forehead was critical to the measurement quality and success. The experimental procedures lasted more than three weeks, and the following steps were taken in each participant's session:

(1) The cognitive optical brain imaging (COBI) software was started, and then it was waited for the signal traces to stabilize.
(2) If the signal values were high (>4000 mV) or low (<400 mV), then the tightness of the headband was checked and adjusted.
(3) Seeing the baseline signal levels as acceptable, the data were recorded to a specified file after the baseline ended automatically.
(4) The participant performed the experimental task.
(5) The data acquisition process ended after completing the task.

The first-five steps were iterated for all participants.

4.6. *Data collection and analysis*

The fNIRS system collected the raw data from a 16-channel sensor headband worn by the participants. The system recorded 48 measurements of 16 voxels for each sampling period to a text file. Each of the 16 voxels included 3 columns containing the light intensity data in 3 wavelengths (Appendix 4). The COBI software used the baseline signal values to calculate the oxygenations and to record them to the output files. Later, the artifacts were low-pass filtered by the testing and analysis software to obtain the filtered data. Consequently, some of the data were excluded due to distortion (equipment noise, respiration and motion artifacts), and the actual data of each participant were recorded to new text files. Each of these files was converted to a spreadsheet to calculate the cumulative oxygenation changes for each participant. Then, the data were aggregated to a single value, which represented the average oxygenation change and the participants' total cognitive response. Finally, they were imported into the SPSS v.18 software for the statistical analysis procedures.

5. Results

Descriptive and inferential statistical techniques were used to test the hypothesis and to analyze the experimental data. The results were depicted in tables, and their interpretations were presented in the corresponding paragraphs of relevant sections below. Based on the results of the normality test (Shapiro–Wilk), the average oxygenation measurements and the academic achievements were not normally distributed ($p < .05$). Therefore, nonparametric tests were employed for the inferential statistical analysis. A summary of descriptive statistics is given in the following tables. As can be seen in Table 1, the average oxygenation change of the BlueJ group (Mean = 0.8676) is lower than that of the JCreator LE group (Mean = 2.4895).

The descriptive data about the academic achievement scores are presented in Table 2. It is found that the average score for the BlueJ group (Mean = 86.6000) is very close to that of the JCreator LE group (Mean = 85.6667). Moreover, the Mann–Whitney test results did not indicate a statistically significant difference between the experimental groups in terms academic achievements ($z = 0.511$; $p > .05$).

Table 3 presents all of the descriptive data of this study in summary that includes the participants' average oxygenation changes and academic achievements.

Table 1. Means and standard deviations for the fNIRS measurements.

Groups	*n*	Mean	Std. deviation
BlueJ	10	0.8676	0.8475
JCreator LE	9	2.4895	1.3481
Total	19	1.5268	1.2948

Table 2. Means and standard deviations for academic achievements.

Groups	*n*	Mean	Std. deviation
BlueJ	10	86.6000	7.6912
JCreator LE	9	85.6667	6.9282
Total	19	86.1579	7.1512

Table 3. The descriptive data of the study.

Learners	Groups	Achievement test	Average oxygenation change
Learner-1	BlueJ	80	0.0017
Learner-2	BlueJ	94	1.9034
Learner-3	BlueJ	71	1.5286
Learner-4	JCreator LE	76	3.3254
Learner-5	JCreator LE	91	2.4895
Learner-6	BlueJ	83	0.6774
Learner-7	JCreator LE	77	3.7285
Learner-8	BlueJ	90	1.0801
Learner-9	BlueJ	92	0.3072
Learner-10	JCreator LE	90	3.1880
Learner-11	JCreator LE	78	0.1091
Learner-12	JCreator LE	87	1.6727
Learner-13	BlueJ	80	2.4204
Learner-14	JCreator LE	90	0.7261
Learner-15	JCreator LE	95	3.7394
Learner-16	BlueJ	93	0.6694
Learner-17	BlueJ	93	0.0883
Learner-18	JCreator LE	87	1.3532
Learner-19	BlueJ	90	0.0002

Table 4. The Mann–Whitney test results of average oxygenation changes.

Group	n	Mean rank	Sum of ranks	z	p
BlueJ	10	7.10	71.00	−2.368	.018
JCreator LE	9	13.22	119.00		

As the research hypothesis indicated, there was enough evidence to conclude on a difference in the values at the $\alpha = .05$ level of significance according to the average oxygenation changes ($z = -2.368$; $p < .05$). It is possible to state that the participants using the BlueJ had lower average oxygenation changes than those who used JCreator LE (Table 4). It was investigated whether the academic achievements would correlate with the average oxygenation changes. According to the Pearson's correlation coefficient, the regression test results showed no significant correlation between the participants' average oxygenation changes and their academic achievements ($r = -0.122$; $p > .05$).

6. Discussion

In general, the findings of this research are in agreement with the current fNIR literature exploring the cognitive states when participants use software systems (Ferrari & Quaresima, 2012). The fNIR technology used in this study measured average oxygenation changes in the participants to collect information about the neural activation when using different IDEs. In order to understand the biological significance and its cognitive meaning, it is noteworthy that the two different hemodynamic activities produced the results: the decrease and increase in the mean concentration levels of oxygenated hemoglobin.

In the Girouard's et al. (2010) study, fNIRS was verified to provide a measure of cognitive load experienced by the users working on a simple task with given software interfaces. As one of the findings, visually altering UIs and changing underlying information

demanded relatively less cognitive effort. In another study, Hirshfield et al. (2009) also adopted the Shneiderman's theory, which was based on the assumption that syntactic and semantic cognitive efforts are needed to complete a task requiring the use of a UI. They designed a simple task and two user UIs to map directly to the participants' spatial and verbal working memory. By performing physiological measurements, they were able to separate and to identify syntactic and semantic cognitive loads using fNIRS technology. The results were in line with our findings, and also showed that fNIRS could be a useful method to differentiate the brain activities induced by different UIs.

When regarding the task complexity in fNIRS experiments, however, our results may contradict the findings of the León-Carrion's et al. (2008) study. It was found that the individuals, who displayed better performances, had also the highest oxygenation levels during the investigation of color–word interference by using a modified Stroop test and fNIRS. They attributed these results and the high level of measures to the participants' effectiveness of cognitive control, claiming that the differences might stem from the use of different solution strategies, attention levels, or task difficulty. In our study, however, the task simplicity notion was employed as one of the experimental controls to make it possible to concentrate on the IDEs. Therefore, these results suggest that how a cognitive task and its level of complexity affect the fNIRS measures is controversial, and it still needs empirical evidences.

Furthermore, the findings can be also explained by examining the programming processes in detail, and they can be discussed under the two headings; (1) CTL and (2) ML, though some of the aspects of these knowledge domains may be overlapped. The former may highlight the cognitive consequences of the visual support in IDEs based on fNIRS measures. The latter can help us to reflect on the multimedia aid of the IDEs in view of CTML.

6.1. *Interpretation of the results in view of CLT*

In general, novice OOP learners perceive decomposition and solution activities more difficult than understanding a programing problem (Tegarden & Sheetz, 2001). In this study, the requirements of the programming task were clearly defined and explained to the participants before the experiment, and so they did not have to spend extra cognitive effort to comprehend the task. As to the decomposition programming processes, the participants were mainly expected to design the logical and physical structure of the code. Especially, the class designs, which involved design and implementation of complex concepts, for example, inheritance and polymorphism, would demand highly cognitive effort (Rosson & Alpert, 1990). The participants had to maintain the access to the task-related information, and they had to retrieve the previously acquired knowledge in long-term memory. Since the information retrieval cues might directly affect the quality and effectiveness of the cognitive performance, the participants needed assistance to have a reliable information access and retrieval. Therefore, the BlueJ's UML-like graphical and visual interfaces were supportive for identifying the perceptual cues (Morey & Cowan, 2005) associated with the application of core OOP concepts.

The BlueJ users could explicitly put a mental emphasis on the class and object relationships, instead of concentrating on the programming details. It provided guidance throughout programming process, and connected the logical design (class design) with the physical design (coding) by visual interfaces. BlueJ users easily switched among the visual and text-based representations while revising what they had coded so far. The participants directly started with the visual design of classes, navigated to the text editor, and they tested the code without overhead. Thus, they could consciously cross between the programming phases (logical and physical) in the BlueJ's environment, and therefore, they were able to adopt

a more systematic programming approach. This development cycle required relatively less time and cognitive effort when compared to that in the JCreator LE environment.

Additionally, it is known that task automation is important in performing a cognitive task similar to computer programming (van Merriënboer & Pass, 1990). It frees up working memory, reduces cognitive load and information can be processed automatically without extra mental effort. Therefore, BlueJ was helpful for learners to construct visual templates required for both automating and implementing programming skills. On the other hand, JCreator LE's components were relatively uncoupled. The BlueJ users could easily transfer their logical and mental designs to the code by using the UML-like visual interfaces. However, the JCreator LE users had to keep the logical designs in their mind, and frequently referred to them when developing code. For that reason, they had to jump from a high level, for example, class design, to a low level, for example, method implementation. As a result, the decomposition and solution activities of OOP were interleaved. Contrary to the BlueJ environment, even a small change in the code or in the class design necessitated a complete compilation or debugging. Therefore, it was observed that JCreator LE users often deviated from their programming plans, and they went back and forward between the decomposition and solution phases during the experiment (Bellamy & Gilmore, 1990). Consequently, unnecessarily cognitive processing of the experimental task and the split-attention effect possibly resulted as one of the sources of extraneous cognitive load (Ayres & Sweller, 2005).Visualizing the complex concepts and abstract entities of object orientation also provided a powerful easy-to-understand and easy-to-implement environment for BlueJ users. Accordingly, it is possible to state that BlueJ decreased the extraneous cognitive load, and it was effective in the use of cognitive capacity.

6.2. *Interpretation of the results in view of ML*

In general, OOP learners find programming concepts and techniques difficult to envision, and they need support in forming mental representations in a concrete form. If the programming tasks are visually represented, they may be retained in long-term memory more effectively and can be processed easily in working memory (Mayer, 2009). Therefore, learners' mental models are affected by external representations of programming structures. It is required for every programmer to refer these mental models when solving problems. If a programmer cannot match her representations to a given programming task, she has to do extra work for the transition between the programming task and representations (Cañas, Bajo, & Gonzalvo, 1994). Therefore, BlueJ's graphical representations enabled alternative views of the experimental task. Users were able to control their focus by navigating to visual- or text-based interfaces, and used the abstraction mechanism and development features of BlueJ simultaneously. Our results were also consisted with the findings of Chang, Hsu, and Yu's (2011) study, which found that the learning of a programming language in a dual-screen learning environment was more effective in avoiding extraneous cognitive loads than in a single-screen environment. Thus, BlueJ possibly enabled the programmers to map their previous mental models of OOP schemata to the experimental task, and guided them throughout the programming processes (Tegarden & Sheetz, 2001).

In terms of CTML, BlueJ is compliant with the principles of contiguity aids, such that the users can understand more deeply when text (code) and pictures (class diagram) are presented simultaneously (Mayer & Moreno, 2003). It applied the spatial contiguity principle by placing the UML-like interface and the text editor near each other; and the temporal contiguity by allowing the simultaneous usage of class diagram, code and object inspection mechanism. Thus, the BlueJ users were able to make connections between corresponding

visual and text representations of the experimental task. However, JCreator LE users were mainly able to perceive and process text-based information. Except for debugging, JCreator's visual support helped the participants only for creation, modification and configuration of the files. Its users mostly attended to the text-based materials, and they were not provided with the opportunity to build connections between graphical and text representation of OOP structures. As shown in the studies of Colvin, Tobler, and Anderson (2007) and Hutchings and Stasko (2007), the use of integrated screens in a learning environment provided the BlueJ users with more learning and application space. Consequently, BlueJ is thought to have complemented the cognitive processes effectively, that is, by selecting codes as relevant words from the text editor, presenting classes as pictures from the design interface, and organizing them into consistent verbal (code) and visual (class) representations.

6.3. *Limitations and future direction*

One limitation of this study was the design of incremental experimental task. The fNIRS literature generally includes time-varying cognitive tasks, such as n-back tasks, which are adjusted by increasing or decreasing task difficulty. This is because statistically more consistent inferences and comparison can be made if the physiological measures are simultaneously observed with the changing task patterns. However, the conceptual and programmatically complex nature of OOP made it difficult to design such a programming task. Therefore, rather than adopting a holistic approach as in this study, the combination of fNIRS measures with integrated OOP tasks is suggested for future research. Moreover, integrating these measures with some of software metrics, such as function points and lines of code, can help deeply understand the physiological meaning of individual programming processes.

The other limitation was the findings associated with the principles of CTML and fNIRS method. Although BlueJ can be compliant with some of the principles, there is still need for more empirical support to explain how these principles could be applied to lower the cognitive load in the framework of fNIRS. Finally, the sample size, which is reasonable when regarding the studies done on fNIRS, was another limitation. It is still unknown just how the results were representative in terms of the users' profile, and of the developers who use these IDEs regularly. Therefore, these limitations will direct the attention to the issues in the list of future work, which will also highlight our future directions.

7. Conclusion

The literature review on learning environments for OOP shows little effort given to the instructional evaluation of IDEs, especially the studies using direct and objective techniques. Therefore, this quasi-experimental research tried to identify whether a more visually functional and supportive IDE would lead to lower cognitive load measured by the fNIRS method. The participants were required to write a simple Java application while they were using either BlueJ or JCreator LE, and they applied the concepts of OOP to their programs. There were also several experimental controls to avoid extraneous cognitive load. The results indicated a significant difference between the experimental groups, and the findings were discussed in view of the principles of CLT and CTML. As a result, BlueJ is believed to have reduced the cognitive load and helped the participants to use their cognitive capacity.

It is worth reminding that one implicit intention was to introduce the fNIRS method and to see how it could be applicable to the research area of evaluating learning environments.

The initial experiences indicated that it is a promising and innovative tool for monitoring the cognitive tasks in various learning contexts. It is important to point out that the findings of this study should not mean that one IDE can be preferred to the other. Learners have different attitudes, strengths or preferences towards to take in and process information. The conclusions drawn from this study might vary depending on different aspects of cognitive psychology, cognitive or learning styles, motivation and individual differences. Therefore, this paper concludes with an invitation for more researches to be conducted on the fNIRS method and IDEs. It is hoped that this study would extend the previous knowledge not only by the tools it has utilized, but also by the approaches it has adopted for the evaluations of interactive learning environments.

Acknowledgement

This study was conducted in the laboratories of Modeling and Simulation Research and Development Center at Middle East Technical University. The author would like to thank the members of this center for their continual support throughout the study. The author also gratefully acknowledges Dr Murat Perit Cakir for processing of fNIRS data.

Disclosure statement

No potential conflict of interest was reported by the author.

Notes on contributor

Murat Pasa Uysal is Assoc. Prof. Dr. at the Department of Computer Technologies in Ufuk University. He holds a B.S. degree in electrical & electronic engineering from Turkish Military Academy, (TMA) an M.S. degree in computer engineering from Cankaya University, a Ph.D. degree in technology of education from Gazi University. He completed his post-doctoral studies at Rochester Institute of Technology in New York, on both software re-engineering and IT governance. He directed or served as an advisor and engineer for IT projects in TMA and Turkish Army (TA) for many years, and conducted studies addressing the problems of TA in the research areas of IT. He has been teaching IT, computer and software engineering-related courses. His research interest is in the areas of IT, instructional methods and tools for computer programming, software engineering and IT governance.

References

Ayaz, H., Shewokis, P. A., Bunce, S., Izzetoglu, K., Willems, B., & Onaral, B. (2012). Optical brain monitoring for operator training and mental workload assessment. *NeuroImage*, *59*, 36–47.

Ayres, P., & Sweller, J. (2005). The split-attention principle in multimedia learning. In R. E. Mayer (Ed.), *The Cambridge handbook of multimedia learning* (pp. 135–158). New York: Cambridge University Press.

Bellamy, R. K. E., & Gilmore, D. J. (1990). *Programming plans: Internal or external structures in lines of thinking: Reflections on the psychology of thought*. New York: Wiley.

Berninger, V. W., & Corina, D. (1998). Making cognitive neuroscience educationally relevant: Creating bidirectional collaborations between educational psychology and cognitive neuroscience. *Educational Psychology Review*, *10*(3), 343–354.

Brünken, R., Plass, J. L., & Leutner, D. (2003). Direct measurement of cognitive load in multimedia learning. *Educational Psychologist*, *38*(1), 53–61.

Cañas, J. J., Bajo, M. T., & Gonzalvo, P. (1994). Mental models and computer programming. *International Journal of Human-Computer Studies*, *40*, 795–811.

Chang, T. W., Hsu, J. M., & Yu, P. T. (2011). A comparison of single- and dual-screen environment in programming language: Cognitive loads and learning effects. *Educational Technology & Society*, *14*(2), 188–200.

Chi, M. T. H. (1997). Quantifying qualitative analyses of verbal data: A practical guide. *The Journal of Learning Sciences*, *6*(3), 271–315.

Colvin, J., Tobler, N., & Anderson, J. A. (2007). Productivity and multi-screen computer displays. *Rocky Mountain Communication Review*, *2*(1), 31–53.

Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, *13*(3), 319–339.

Eckerdal, A. (2006). *Novice students' learning of object-oriented programming* (Unpublished doctoral dissertation). The Uppsala University, Sweden.

Ferrari, M., & Quaresima, V. (2012). A brief review on the history of human functional near-infrared spectroscopy (fNIRS) development and fields of application. *NeuroImage*, *63*, 921–935[AQ9].

Girouard, A., Solovey, E. T., Hirshfield, L. M., Peck, E. M., Chauncey, K., Sassaroli, A., … Jacob, R. J. K. (2010). *From brain signals to adaptive interfaces: Using fNIRS in HCI*. Brain-Computer Interfaces, Human-Computer Interaction Series, 221–237. ISBN 978-1-84996-272-8.

van Gog, T., Kester, L., Nievelstein, F., Giesbers, B., & Paas, F. (2009). Uncovering cognitive processes: Different techniques that can contribute to cognitive load research and instruction. *Computers in Human Behavior*, *25*, 325–331.

Green, T. R. G., & Petre, M. (1996). Usability analysis of visual programming environments: A "cognitive dimensions framework". *Journal of Visual Languages and Computing*, *7*, 131–174.

Hirshfield, L. M., Solovey, E. T., Girouard, A., James, K., Jacob, R. J. K., Sassaroli, A., & Fantini, S. (2009, April 4–9). *Brain measurement for usability testing and adaptive interfaces: An example of uncovering syntactic workload with functional near infrared spectroscopy.* Proceedings of CHI 2009, Boston, Massachusetts, USA.

Hutchings, D. R., & Stasko, J. (2007). Quantifying the performance effect of window snipping in multiple-monitor environments. In C. Baranauskas, P. Palanque, J. Abascal, & S. D. J. Barbosa (Eds.), *Proceedings of human-computer interaction INTERACT 2007 Part II* (pp. 461–474). New York: Springer.

Izzetoglu, K., Bunce, S., Onaral, B., Pourrezaei, K., & Chance, B. (2004). Functional optical brain imaging using near-infrared during cognitive tasks. *International Journal of Human-Computer Interaction*, *17*(2), 211–227.

Izzetoglu, M., Bunce, S. C., Izzetoglu, K., Onaral, B., & Pourrezai, K. (2007). Functional brain imaging using near-infrared technology: Assessing cognitive activity in real-life situations. *IEEE Engineering in Medicine and Biology Magazine*, July/August, 36–44.

Jobsis, F. F. (1977). Noninvasive infrared monitoring of cerebral and myocardial oxygen sufficiency and circulatory parameters. *Science*, *198*, 1264–1267.

Johnson, P. N. (1983). *Mental models: Towards a cognitive science of language, inference and consciousness*. Cambridge, MA: Harvard University Press.

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: Taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys*, *37*(2), 83–137.

Kiper, J. D., Howard, E., & Ames, C. (1997). Criteria for evaluation of visual programming languages. *Journal of Visual Languages and Computing*, *8*(2), 175–192.

Klinea, R. B., & Seffah, A. (2005). Evaluation of integrated software development environments: Challenges and results from three empirical studies. *International Journal of Human-Computer Studies*, *63*(6), 607–627.

Kolling, M. (1999). The problem of teaching object-oriented programming, part 2: Environments. *Journal of Object-Oriented Programming*, *11*(9), 6–12.

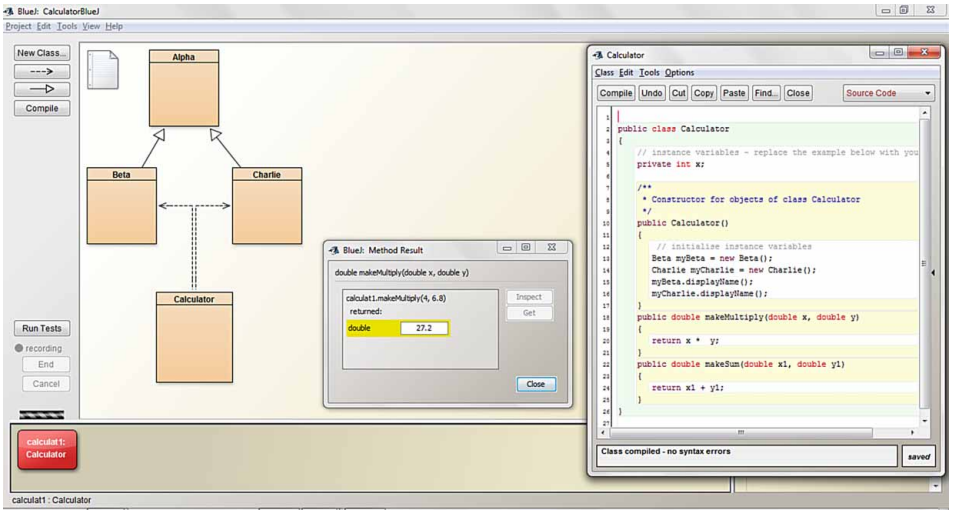
Kordaki, M. (2010). A drawing and multi-representational computer environment for beginners' learning of programming using C: Design and pilot formative evaluation. *Computers & Education*, *54*, 69–87.

León-Carrion, J., Damas-López, J., Martín-Rodríguez, J. F., Domínguez-Roldán, J. M., Murillo-Cabezas, F., Barroso, Y., & Domínguez-Morales, M. R. (2008). The hemodynamics of cognitive control: The level of concentration of oxygenated hemoglobin in the superior prefrontal cortex varies as a function of performance in a modified Stroop task. *Behavioral Brain Research*, *193*, 248–256.

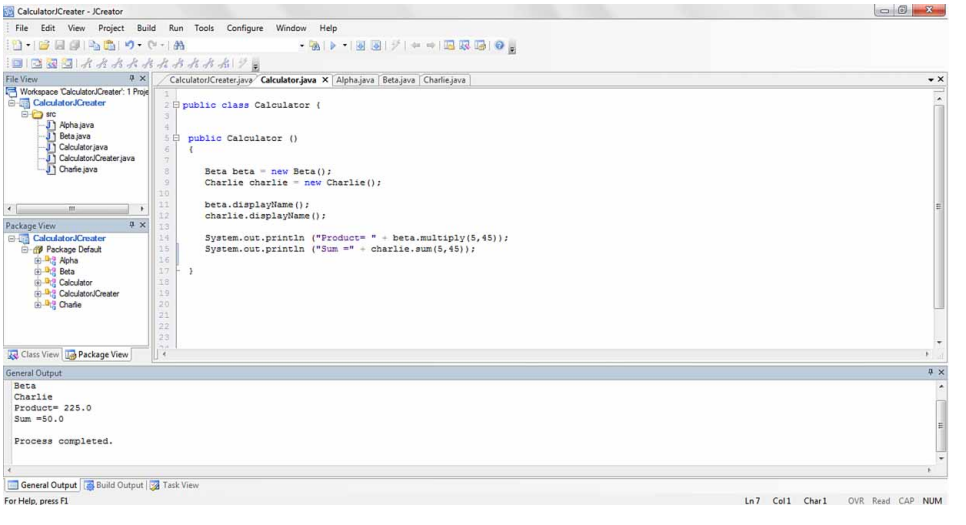
León-Carrióna, J., Izzetoglu, M., Izzetoglu, K., Martín-Rodrígueza, J. F., Damas-López, J., Barroso, J. M. M., & Morales, M. R. D. (2010). Efficient learning produces spontaneous neural repetition suppression in prefrontal cortex. *Behavioral Brain Research*, *208*, 502–508.

Mayer, R. E. (2009). *Multimedia learning* (2nd ed.). New York: Cambridge University Press.
Mayer, R. E., & Moreno, R. (2003). Nine ways to reduce cognitive load in multimedia learning. *Educational Psychologist*, *38*, 43–52.
McIver, L. (2002). Evaluating languages and environments for novice programmers. In J. Kuljis, L. Baldwin, & R. Scoble (Eds.), *Proceedings of 14th workshop of the psychology of programming interest group*, Brunel University, London, 100–110.
van Merriënboer, J. J. G., & Paas, G. W. C. (1990). Automation and schema acquisition in learning elementary computer programming: Implications for the design of practice. *Computers in Human Behavior*, *6*, 273–289.
Miller, P., Pane, J., Meter, G., & Vorthmann, S. (1994). Evolution of novice programming environments: The structure editors of Carnegie Mellon University. *Journal of Interactive Learning Environments*, *4*(2), 140–158.
Moons, J., & Backer, C. D. (2013). The design and pilot evaluation of an interactive learning environment for introductory programming influenced by cognitive load theory and constructivism. *Computers & Education*, *60*, 368–384.
Moran, M. J. (2006). *College student's acceptance of tablet personal computers: A modification of the unified theory of acceptance and use of technology model* (Unpublished Ph.D. thesis). Capella University.
Morey, C. C., & Cowan, N. (2005). When do visual and verbal memories conflict? The importance of working-memory load and retrieval. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *31*(4), 703–713.
Paas, F. G. W. C., van Merrienboer, J. J. G., & Adam, J. J. (1994). Measurement of cognitive load in instructional research. *Perceptual and Motor Skills*, *79*, 419–430.
Paas, F., Tuovinen, J. E., Tabbers, H., & Van Gerven, P. W. M. (2003). Cognitive load measurement as a means to advance cognitive load theory. *Educational Psychologist*, *38*(1), 63–71.
Pane, J. F., & Myers, B. A. (1996). *Usability issues in the design of novice programming systems*. Human-Computer Interaction Institute Technical Report, CMU-HCII-96-101.
Renumol, V. G., Janakiram, D., & Jayaprakash, S. (2010). Identification of cognitive processes of effective and ineffective students during computer programming. *ACM Transactions on Computing Education*, *10*(3), 211–232. doi:10.1145/1821996.1821998
Rosson, M. B., & Alpert, S. R. (1990). The cognitive consequences of object-oriented design. *Human-Computer Interaction*, *5*, 345–379.
Shaffer, D., Doube, W., & Tuovinen, J. (2003). Applying cognitive load theory to computer science education. In M. Petre & D. Budgen (Eds.), *Proceedings of joint conference EASE & PPIG*, 333–346.
Shneiderman, B., & Plaisant, C. (2005). *Designing the user interface: Strategies for effective human-computer interaction* (4th ed.). Reading: Addison-Wesley.
Strangman, G., Culver, J. P., Thompson, J. H., & Boas, D. (2002). A quantitative comparison of simultaneous bold fMRI and NIRS recordings during functional brain activation. *Neuroimage*, *17*(2), 719–731.
Sweller, J., van Merriënboer, J. J. G., & Paas, F. G. W. C. (1998). Cognitive architecture and instructional design. *Educational Psychology Review*, *10*, 251–296.
Tegarden, D. P., & Sheetz, S. D. (2001). Cognitive activities in OO development. *International Journal of Human-Computer Studies*, *54*, 779–798.
Uysal, M. P. (2014). Interviews with college students: Evaluating computer programming environments for introductory courses. *Journal of College Teaching & Learning*, *11*(2), 126–136.
Whittle, J., & Cumming, A. (2000). Evaluating environments for functional programming. *International Journal of Human-Computer Studies*, *52*, 847–878.
Xinogalos, S. (2010). Guidelines for designing and teaching an effective object-oriented design and programming course. *Advance Learning*, *10*, 397–422.

Appendix 1. BlueJ IDE



Appendix 2. JCreator LE IDE



Appendix 3. Experimental task for the fNIRS measurements

The Programming Task

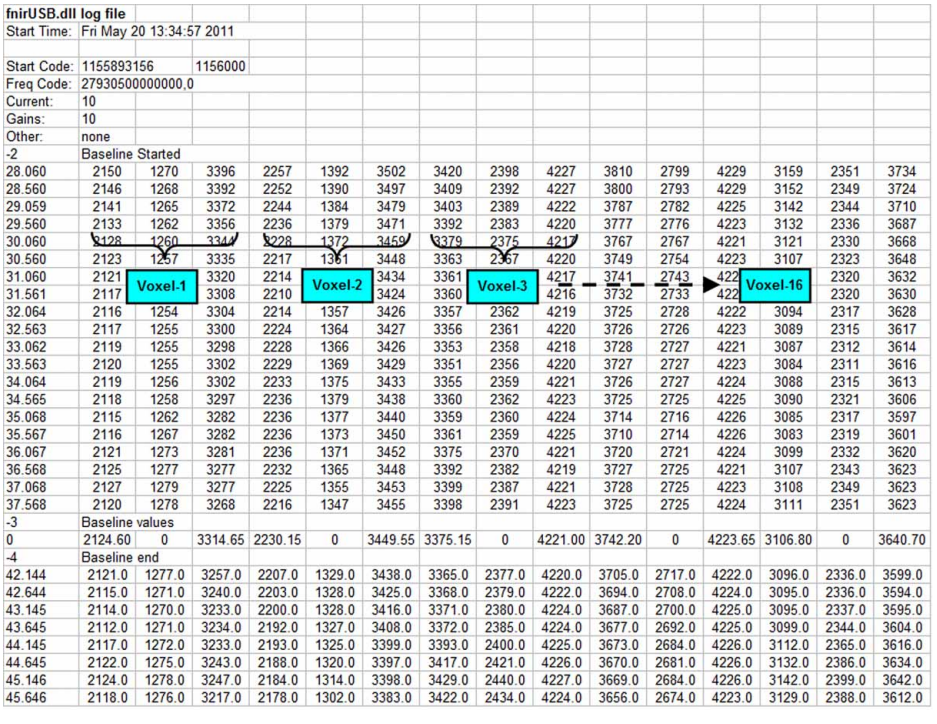
Write a Java application simulating a simple calculator using the specifications below. Your program should make "sum" and "multiplication" arithmetic operations. Users should be able to input data, and observe the outputs on a simple user interface

Requirement Specifications and Explanations

There are 3 classes. Beta and Charlie extends Alpha. How you apply the concepts of inheritance and polymorphism to your program is primarily tested in this task. It is important to meet all the requirements, and take the following explanations into account when designing and developing your application:

1 Alpha is a base class; Beta and Charlie are subclasses of Alpha
2 "DisplayName()" is a method of class Alpha, and it returns the "Alpha" as a string value
3 Class Beta overrides "displayName()" method of Alpha and it returns the "Beta" as a string value
4 Class Beta has a method named as "multiply ()".This returns a double value and it has a signature as (double, double)
5 Class Charlie overrides "displayName()" method of Alpha and it returns the "Charlie" as a string value
6 Class Charlie has a method named as "sum ()". This returns a double value and it has a signature as (double, double)

Appendix 4. Learner-1's data recorded by the fNIRS system



fnirUSB.dll log file															
Start Time:	Fri May 20 13:34:57 2011														
Start Code:	1155893156	1156000													
Freq Code:	27930500000000,0														
Current:	10														
Gains:	10														
Other:	none														
-2	Baseline Started														
28.060	2150	1270	3396	2257	1392	3502	3420	2398	4227	3810	2799	4229	3159	2351	3734
28.560	2146	1268	3392	2252	1390	3497	3409	2392	4227	3800	2793	4229	3152	2349	3724
29.059	2141	1265	3372	2244	1384	3479	3403	2389	4222	3787	2782	4225	3142	2344	3710
29.560	2133	1262	3356	2236	1379	3471	3392	2383	4220	3777	2776	4223	3132	2336	3687
30.060	2128	1260	3347	2228	1372	3459	3379	2375	4217	3767	2767	4221	3121	2330	3668
30.560	2123	1257	3335	2217	1361	3448	3363	2367	4220	3749	2754	4223	3107	2323	3648
31.060	2121		3320	2214		3434	3361		4217	3741	2743	422		2320	3632
31.561	2117		3308	2210		3424	3360		4216	3732	2733	422		2320	3630
32.064	2116	1254	3304	2214	1357	3426	3357	2362	4219	3725	2728	4222	3094	2317	3628
32.563	2117	1255	3300	2224	1364	3427	3356	2361	4220	3726	2726	4223	3089	2315	3617
33.062	2119	1255	3298	2228	1366	3426	3353	2358	4218	3728	2727	4221	3087	2312	3614
33.563	2120	1255	3302	2229	1369	3429	3351	2356	4220	3727	2727	4223	3084	2311	3616
34.064	2119	1256	3302	2233	1375	3433	3355	2359	4221	3726	2727	4224	3088	2315	3613
34.565	2118	1258	3297	2236	1379	3438	3360	2362	4223	3725	2725	4225	3090	2321	3606
35.068	2115	1262	3282	2236	1377	3440	3359	2360	4224	3714	2716	4226	3085	2317	3597
35.567	2116	1267	3282	2236	1373	3450	3361	2359	4225	3710	2714	4226	3083	2319	3601
36.067	2121	1273	3281	2236	1371	3452	3375	2370	4221	3720	2721	4224	3099	2332	3620
36.568	2125	1277	3277	2232	1365	3448	3392	2382	4219	3727	2725	4221	3107	2343	3623
37.068	2127	1279	3277	2225	1355	3453	3399	2387	4221	3728	2725	4223	3108	2349	3623
37.568	2120	1278	3268	2216	1347	3455	3398	2391	4223	3725	2725	4224	3111	2351	3623
-3	Baseline values														
0	2124.60	0	3314.65	2230.15	0	3449.55	3375.15	0	4221.00	3742.20	0	4223.65	3106.80	0	3640.70
-4	Baseline end														
42.144	2121.0	1277.0	3257.0	2207.0	1329.0	3438.0	3365.0	2377.0	4220.0	3705.0	2717.0	4222.0	3096.0	2336.0	3599.0
42.644	2115.0	1271.0	3240.0	2203.0	1328.0	3425.0	3368.0	2379.0	4222.0	3694.0	2708.0	4224.0	3095.0	2336.0	3594.0
43.145	2114.0	1270.0	3233.0	2200.0	1328.0	3416.0	3371.0	2380.0	4224.0	3687.0	2700.0	4225.0	3095.0	2337.0	3595.0
43.645	2112.0	1271.0	3234.0	2192.0	1327.0	3408.0	3372.0	2385.0	4224.0	3677.0	2692.0	4225.0	3099.0	2344.0	3604.0
44.145	2117.0	1272.0	3233.0	2193.0	1325.0	3399.0	3393.0	2400.0	4225.0	3673.0	2684.0	4226.0	3112.0	2365.0	3616.0
44.645	2122.0	1275.0	3243.0	2188.0	1320.0	3397.0	3417.0	2421.0	4226.0	3670.0	2681.0	4226.0	3132.0	2386.0	3634.0
45.146	2124.0	1278.0	3247.0	2184.0	1314.0	3398.0	3429.0	2440.0	4227.0	3669.0	2684.0	4226.0	3142.0	2399.0	3642.0
45.646	2118.0	1276.0	3217.0	2178.0	1302.0	3383.0	3422.0	2434.0	4224.0	3656.0	2674.0	4223.0	3129.0	2388.0	3612.0