Gazi Eğitim Bilimleri Dergisi (GEBD)
Gazi Journal of Education Sciences (GJES)
Cilt. 5; Özel Sayı; 2019; 39~54
ISSN: 2149-4932 E-ISSN: 2149-9381

Transforming Traditional Courses Through Web 2.0 Philosophy: The Case of an Introductory Programming Course

Hakan TÜZÜN

Hacettepe Üniversitesi

htuzun@hacettepe.edu.tr

Gönderilme Tarihi: 21/07/2019 **Kabul Tarihi**: 18/09/2019

Yayınlanma Tarihi: 10/10/2019 **DOI:** 10.30855/gjes.2019.os.01.003

Article Info

Keywords:

Programming and programming languages,

Improving classroom teaching,

Postsecondary education,

Teaching/learning strategies,
Pedagogical issues

ABSTRACT

In this study, the use of ubiquitous Internet tools utilized in an undergraduate introductory computer programming course at a large state university in Turkey are examined. Course web space, blogs, discussion board, e-mail, and personal web pages were among the Internet tools utilized by the participants of the course. The framework for using these tools has been conceived as "Learning Programming 2.0," which was inspired and shaped by "Web 2.0" principles and practices conceptualized by Tim O'Reilly. Web 2.0 characterizes second generation Internet services, such as wikis, blogs, and other tools that encourage interaction and participation through the Internet. Essentially, Web 2.0 framework includes not only the tools, but also a set of principles for using these tools such as harnessing collective intelligence or providing rich user experiences. In this sense, the focus of the study is not on the tools, but the pedagogical principles followed to transform the course, the impact of these principles on the context, and implementation problems. Although the Learning Programming 2.0 framework puts extra load on the shoulders of facilitators, it is concluded that this approach contributes to formation of a learning context in which learning programming is meaningful, effective, and life-long for learners.

Tüzün, H. (2019). Transforming traditional courses through web 2.0 philosophy: The case of an introductory programming course. *Gazi Eğitim Bilimleri Dergisi, 5(Özel Sayı),* 39-54. DOI: https://dx.doi.org/10.30855/gjes.2019.os.01.003

***Dergi Web Sayfası**: http://dergipark.gov.tr/gebd*

Geleneksel Derslerin Web 2.0 Felsefesiyle Dönüştürülmesi: Bir Programlama Dersi Örneği

Makale Bilgileri

Anahtar Kelimeler:

Programlama ve programlama dilleri,

Sınıf öğretiminin geliştirilmesi, Üniversite eğitimi,

Öğretme ve öğrenme stratejileri, Pedagojik unsurlar

ÖZET

Bu çalışmada, yaygın İnternet araçlarının Türkiye'de bir devlet üniversitesinde lisans seviyesindeki bir programlama dersinde kullanımı incelenmiştir. Ders web alanı, bloglar, tartışma listesi, e-posta ve kişisel web sayfaları, ders katılımcıları tarafından kullanılan İnternet araçları arasındadır. Bu araçları kullanma çerçevesi, Tim O'Reilly tarafından kavramsallaştırılan "Web 2.0" ilke ve uygulamalarından ilham alan ve biçimlendirilen "Programlamayı Öğrenme 2.0" olarak ifade edilmiştir. Web 2.0; wiki'ler, bloglar ve İnternet üzerinden etkileşimi ve katılımı teşvik eden diğer araçlar gibi ikinci nesil İnternet hizmetlerini karakterize eder. Temel olarak, Web 2.0 çerçevesi sadece araçları değil aynı zamanda bu araçları kullanmak üzere ortak aklın kullanılması veya zengin kullanıcı deneyimlerinin sağlanması gibi bir takım ilkeleri de içerir. Bu anlamda; çalışmanın odağı araçlar değil, dersi dönüştürmek için izlenen pedagojik ilkeler, bu ilkelerin bağlam üzerindeki etkisi ve uygulama sorunlarıdır. Programlamayı Öğrenme 2.0 çerçevesi, öğretim elemanlarının omuzlarına fazladan yük getirmesine rağmen, bu yaklaşımın programlama öğrenmeyi anlamlı, etkili ve yaşam boyu sürer hale getiren bir öğrenme bağlamının oluşumuna katkıda bulunduğu sonucuna varılmıştır.

INTRODUCTION

Software has become very vital for the modern society. At the dawn of the 21st century, most aspects of everyday life are controlled by software, including banking, transportation, and medical systems. Closer examples at a personal level would be elevators, air conditioning, electricity, and water in office buildings and homes, air bags in cars, traffic lights on streets, and all types of communication among people other than snail mail. As more and more routine and critical tasks are controlled by software, the society needs more and more programmers to fulfill this increasing demand. As an example, while the Apollo 11 mission to the Moon contained 145,000 lines of code, this increased to 400,000 lines in Space Shuttle missions, and 2.5 million lines in Curiosity mission to Mars. Further, the International Space Station's operations run through 2.3 million lines of computer code ("NASA facts and figures," 2019). With programming being this much crucial, cultivating programmers is more important than ever to meet the demands of society (European Schoolnet, 2015). However, creating software is a complex process (Kalelioğlu, Gülbahar & Doğan, 2017) and a literature review reported many difficulties in teaching and learning programming in formal courses (Saygıner & Tüzün, 2018).

Related Literature and Theoretical Framework

The purpose of a programming course is usually to teach about basic programming concepts, data and control structures, and good programming style (Malik, 2000). According to Gries (2006), influential innovators of programming such as Niklaus Wirth made the following contributions in 1970s to the teaching of programming languages, but these are not emphasized

in programming courses anymore: 1) Simplicity, beauty, and language are important in teaching programming to beginners, 2) Principles of developing and refining programs are more important than mere code writing, and 3) A program should be developed in context while the context providing opportunities for program development. Gries (2006), an instructor teaching programming for forty years, summed these up by emphasizing that the purpose of programming education should not be conveying facts into students, but to teach them to think.

Researchers and practitioners have utilized innovative pedagogies such as pair programming (Zin, Idris, & Subramaniam, 2006), or innovative software environments such as Alice (Bishop-Clark, Courte, & Howard, 2006), or block-based programming environments (Berland, Martin, Benton, Smith, & Davis, 2013) for teaching programming languages. A rising trend is the use of narrative (Burke & Kafai, 2010), language, and online methods. An example of the use of language is the practice of Panell (2003). Based on his observation of the correlation between the best programmers in his class and their use of everyday language, Panell concluded that teaching an Introductory Programming Course (IPC) is all about teaching a new language. For that reason, he recommended using language related activities. Panell utilized unusual teaching methods in his programming courses such as using poetry and bringing drama into programming. He concluded that students learnt more from these types of experiences than they did by taking notes in lectures.

An example of the use of online methods is the study of Boyle, Bradley, Chalk, Jones, and Pickard (2003). When these researchers experienced problems in student achievement rate and retentions in their IPCs, they followed a blended learning approach to deal with this problem. Their blended environment included both face-to-face and online components, provided support to students through online resources, and assessed the performance of the learners through continuous online methods. Results indicated a widespread use of online components by learners. Overall, the learning environment blended with online components resulted in improvements in pass rates and positive student attitudes in the IPCs studied in this research.

When Kelleher and Pausch (2005) reviewed programming languages and environments designed for making programming accessible to novice programmers, they identified the problems with learning a programming language as not only the mechanical barriers such as syntax and rules of a language but also as sociological barriers such as learners not seeing the relevance of programming or their perceiving programming as an individualistic and isolated act. They identified two kinds of sociological barriers in programming: the absence of "social" and "compelling contexts" for learning about programming. They added sociological barriers were harder to address than the mechanical ones since it was harder to identify them. Consequently, most of the environments they reviewed did not address this issue. They suggested that students could help each other's learning, look and use each other's work, and work on projects together in future programming courses.

This study focuses on a formal learning context in which ubiquitous Internet tools were integrated to overcome mechanical and sociological barriers in students' learning in an undergraduate introductory computer programming course. Towards this purpose, the pedagogical principles followed to transform the course, the impact of these principles on the context, and implementation problems are examined.

METHOD

An ethnographic method conceptualized by Ge, Thomas, and Greene (2006) as "Technology-Rich Ethnography" (TRE) was utilized in this study. TRE is a compilation of data gathering methods, which are appropriate for examination of technology-rich learning environments in order to see what is going on in the context. While there were 13 cases between 2006 and 2018, a single case was selected to fit data to the journal limitations. The case from 2006 was selected to be illuminative of the data for the reason that it was the transition period in terms of using the pedagogical principles.

Research Context and Participants

The research context was a Computer Education and Instructional Technology department at a major research university in Ankara, Turkey. The department aims to educate prospective computer teachers who will teach Information Technology (IT) related courses in K-12 institutions. Major components of the curriculum in the department include domain related courses (50%), pedagogy related courses (25%), and general courses (25%).

"Programming Languages I" is a 5-hour per week (3 hours for lecture and 2 hours for lab activities) mandatory course in the curriculum to be taken at sophomore level. In the curriculum this course is defined as an IPC introducing students to such topics as general structure of programming, data types, variables, sub-programs, and standard functions and procedures. The course is the first programming course students encounter in the curriculum and precedes another mandatory programming course about visual programming languages in the sophomore year. There were 82 students enrolled in two sections of this course in the Fall semester. One of the boys dropped out of the course early in the semester. Therefore, 81 students (30 girls, 51 boys) completed the course.

Transformation of the Course

An analysis of the enrollments showed that 45 (56%) students were taking the course for the first time, while 36 (44%) students were repeating it. The repeating students were asked what they thought were the reasons for repeating the course. They indicated the following difficulties: 1) They directly experienced program codes without understanding the meaning of programming, 2) and as a result they had motivational problems and related drop-outs, and 3) they had difficulties with showing in the exams what they learnt in the course. Overall, all repeaters pointed to the problematic didactic teaching method. The instructor would teach programming in this context for the first time, and the experience of the repeaters demonstrated the need for more active teaching methods.

Transformation of the course was framed by insights, inferences, and attitudes the instructor has developed throughout his programming experience. He started computer programming at the age of eleven with Commodore 64 computers. He developed amateur programs on this simple platform. During this period, he concluded that *one could learn programming on an individual effort; however, English would be a must for that person since most of the original resources were available in English*. His curiosity on programming guided him to IT field and he studied computer systems education at undergraduate and graduate level, during which he practiced such computer languages as Basic, Pascal, Assembly, and C. During this period, he concluded that one's programming knowledge and skills could become better with formal education; however, it would not be possible to learn all aspects of a programming language in the limited time frame of a programming course. For that reason, *learners would have to keep on*

improving their knowledge and skills outside the course. Upon completing the undergraduate education, he became a computer systems teacher and taught courses about programming at a vocational high school for one year and at a university for another year. During this period, he considered that *students had to learn programming simply because they were present in class*. Most of the time, his pedagogy was limited to didactic teaching methods, isolated learners from each other, and assessed their learning according to the written program codes on paper.

When the instructor started and continued his doctoral degree in an instructional technology program, he became enmeshed with the importance of meaning, motivation, social interaction, and culture in learning. At the same time with his doctoral education, when he started to work as a professional programmer in a National Science Foundation project, his vision on learning and teaching programming was supported by additional insights. First, he had to extend his repertoire by learning about such programming languages and technologies as Perl, JavaScript, and MySQL since these were the backbone of the project. He did not have time to attend formal courses on these languages and technologies; therefore, he had to improve his knowledge just-in-time. This constructed his view that *programming education is a life-long endeavor*. Second, while working in the project, he experienced that programming was not a straightforward process that only included coding. *Programming actually was about solving emergent problems in the context of a relevant project by using contextual tools*. These contextual tools included not only printed resources such as books but also a community of programmers who dynamically contributed to others by using discussion boards, blogs, and knowledge bases. And third, he had previously taken numerous formal courses about programming, but he still had to re-learn or remember much of this previous knowledge. In doing this, he did not have to memorize anything since these were available and accessible anytime in the contextual tools previously mentioned. Therefore, he recognized that *some programming concepts and skills were consolidated after numerous exposures to the same content*. All these previous experiences of the instructor contributed to the transformation of the IPC.

Research assignments, lectures, lab assignments, group discussions and interactions, and personal reflections formed the teaching methods of the restructured IPC. A typical week's schedule in the course syllabus is presented in Table 1.

Table 1.
A Typical Week's Schedule in the Syllabus

Week	Topics	[Tasks] & {Lab Activities} for students
Week #8	- Week08.ppt lecture (The process of converting a problem into a program)	- [Send topic research for Week 9] - {Complete the activities in the document Lab08.doc} - {Complete weekly blog entry about what you learnt in the lecture}

One of the learners reflected how these components functioned in the context of the course:

> *We were coming to the lecture with prior knowledge because of the research assignment [about that week's topic]. We were attending the lecture, some of which included participatory activities. We were doing a variety of activities [structured by the instructor] in the lab related to the topic, and at the end of the lab we were blogging [about what we learnt that day]. If we examine this practice as a whole, we would have covered that week's*

topic for four times and the last iteration also gave us a chance for interpretation. This practice didn't give us a chance other than learning.

Data Sources

The following components of TRE were employed in this study: 1) observation records and reflections from the course instructor, 2) end-of-semester reflection reports from learners, 3) weekly blog entries by learners, 4) discussion board postings by learners, and 5) demographics of learners. End-of-semester reflection reports from learners constituted the main TRE data source that provided the richest and the most useful information. Other data sources such as instructor's observation records and reflections served in triangulating this primary data and providing trustworthiness.

Data Analysis

In the reflection reports, the learners were asked the following question: "To what degree Internet tools utilized in the course such as e-mail, blog, and discussion board have made an impact on your learning?" Responses were categorized among the Internet tools by their impact and utilization problems by following open coding (Glaser & Strauss, 1967). Quantitative data regarding the use of e-mail, blog, and discussion board were tabulated to obtain the total, minimum, maximum, and average utilization of these tools. The categories were explained by tabulated statistical data and data drawn from the reflection reports. As recommended by Creswell (1998), a reviewer who had research and teaching experience related to programming, acted as an external audit to identify and eliminate any biases in results and interpretations.

RESULTS

Impact of the Internet Tools

E-mail. The instructor sent out 87 messages over the semester. 16 of these messages were directed to all learners while 71 of them were directed to specific individuals. The instructor received 1002 messages, 85 of which included general questions and 917 of which included assignments. Learners indicated the use of e-mail linked the learners and the instructor out of class. The use of e-mail was necessary to deliver many class assignments, and it provided sustained motivation:

Since you were sending your announcements and feedback through this channel, checking my e-mails frequently became a habit.

[The use of] e-mail was both necessary and made it easy to deliver research assignments and exams. Handing these over would be difficult and time consuming.

My being informed by e-mail on what I needed to do exclusively by the course instructor, our continual communication with him, and his immediate feedback [by e-mail] when I made a mistake increased my interest towards the course. Students like the course the most when they are paid attention to.

Blog. The least number of blog entries was for second week that included the response of 66 learners while the most number of blog entries was for ninth and tenth weeks that included responses from 79 learners. On average, 71 learners reflected about their learning each week. When collective entries for weeks were imported to a word processor, they occupied between 18 pages (second week) and 55 pages (fourteenth week) of reflections, while the weekly average collective blog contribution was 37 pages. The blog tool impacted the learners at personal and

interpersonal levels. At personal level, blog tool increased the concentration of learners within lectures:

The blog assignments being related to what was covered in the lectures made me fully concentrate within the lectures.

Blogging helped learners with recalling previous topics:

I was reviewing the topics I forgot about by following the blogs [I wrote].

Another impact of blogs at personal level was consolidating and critiquing that week's content:

Blog was a repetition and evaluation of that day's topic from our own perspectives. Objectively, I tried to write about everything I did and experienced that day (including things like if the lecture bored me and if I slept in the lecture). This showed me how much I achieved in the course that day, and also showed how much more I should achieve. It was also increasing my attention to the course.

I guess blogging was a useful self-evaluation tool. This way, I knew what I achieved and what I was missing related to that week's topic, I was linking that week's topic with previous weeks', and I was speculating about how my learning in that week would benefit myself in the future.

Blogging helped with correcting learners' general misconceptions about specific programming topics:

Another positive outcome of blogging was that it afforded the instructor correcting any misunderstandings that we thought we understood in the lectures.

The blog tool also compensated the learners when they were not able to attend a specific lecture:

I was not able to attend the lecture a week, and by going through [my friends'] blogs I compensated for this deficit.

At the interpersonal level, the blog tool allowed for sharing and paying attention to others' perspectives:

I was supplementing the issues I missed in the lecture and also the issues I didn't understand about by reading through my friends' blogs.

In my opinion, the public availability of blogs [to each other] supported the course. Because, it allowed others to see what they learnt in the class and how they related the topics to previous topics. This way, we had a chance to study a topic from multiple perspectives.

Discussion board. Excluding 842 messages under entertainment topic and 627 messages under technology news topic, learners posted a total of 1626 messages related to course content in the discussion board. 1300 of these messages were under 13 topics initiated by the instructor, and 326 messages were under 8 topics initiated by learners. For topics that were initiated by the instructor and required mandatory participation, 36 learners (topic of "what would happen?") to 76 learners (topic of "why binary system?") contributed. The average number of learners participating in each mandatory topic was 52. The biggest impact of the discussion board was that it allowed for communication among course participants out of class:

Every course should definitely have a discussion board. I was following all students' perspectives there, I was able to learn through the resources sent, and I was able to direct questions and get answers. It

was like a library. Since there were many students taking the course, the participation was immense. The topics in the discussion board on "entertainment" and "technology news" added an extra flavor.

One of the learners creatively explicated the overall pedagogical impact of the discussion board:

In our pedagogy relates courses, we constantly touch upon an issue. Our instructors always indicate that students learn better from each other; they communicate the ideas to each other the way they understand. Discussion board served in this purpose by presenting this opportunity to learn from each other's learning. I benefited a lot in terms of programming and IT from the topics of discussion board.

Characterizing the Use of Internet Tools: Analytical Web 2.0 Framework

The crash of the dot-com bubble in 2001 was a turning point for the Internet. It was observed that the software companies that survived this crash had some common characteristics. These characteristics were conceptualized as Web 2.0, and the concept embraces a set of principles for the design of next-generation software for the Internet (O'Reilly, 2005). According to Web 2.0 framework, Internet applications could be at a varying distance to these principles. The more principles an application contains, the more Web 2.0 it is. The Web 2.0 framework was utilized in this study as an analytical lens to examine the activity of learners in the IPC through Internet tools. The concept of "Learning Programming 2.0" was inspired by "Web 2.0" principles and practices conceptualized by O'Reilly (2005). It denotes a set of principles for using in programming education. While Web 2.0 defines seven principles, Learning Programming 2.0 framework contains seven transformed principles. Below, Web 2.0 principles are explained based on O'Reilly (2005) and their corresponding equivalents in Learning Programming 2.0 framework are explicated. An advanced organizer is presented in Table 2.

Table 2.

Principles of Web 2.0 and Learning Programming 2.0

Principles identifying "Web 2.0"	Principles identifying "Learning Programming 2.0"
1) The web as platform	1) The context as platform
2) Harnessing collective intelligence	2) Harnessing collective intelligence
3) Data is the next intel inside	3) Added value
4) End of the software release cycle	4) End of the exam-based assessment
5) Lightweight programming models	5) Lightweight course management models
6) Software above the level of a single device	6) Learning above the level of brick-and-mortar
7) Rich user experiences	7) Rich learner experiences

The web as platform (the context as platform). Web 2.0 applications treat the web as a platform; different web services scattered over the Internet function together seamlessly to deliver an integrated page to a reader on another computer on the Internet. In the Programming 2.0 framework, this principle takes the form of "the context as platform;" i.e., multiple tools and pedagogies are utilized in multiple contexts as part of the programming course to support students in their learning.

The Web 2.0 framework points to the importance of data in Internet applications: "The value of the software is proportional to the scale and dynamism of the data it helps to manage" (O'Reilly, 2005). Google, for example is a specialized database with its extensive database

management capabilities. In the IPC, the course Web space was utilized to provide this dynamism of the content. The course Web space brought together such tools as lecture notes, lab activities, and blog and discussion tools. These tools dynamically and organically increased the content and value of the course over the semester.

The Web 2.0 framework advices to "leverage customer self-service and algorithmic data management to reach out to the entire web, to the edges and not just the center, to the long tail and not just the head" (O'Reilly, 2005). Here, the long tail refers to the collective power of tiny Web sites that constitute the majority of the Internet. As an example, with its AdSense technology, Google invented a way to push its commercials to hundreds of thousands of these tiny Web sites. In the IPC, all learners were reached out by e-mails to connect them to the course when they existed out of class. Reminders about the upcoming assignments and emerging issues were shared regularly through e-mail messages.

Another key Web 2.0 principle indicates that "the service automatically gets better the more people use it" (O'Reilly, 2005). Most point to point (P2P) applications are an example of this principle. In these applications, every client in the network also participates as a server. The implication of this principle for Learning Programming 2.0 is that every learner is a potential instructor at the same time. Learners shared their knowledge and skills through the blog and discussion tools in the course for the benefit of their peers.

Harnessing collective intelligence (harnessing collective intelligence). Major Web 2.0 applications are able to harness the collective intelligence of their users. eBay, for example, is really about the collective activity of its users. This service provides a context for its users and it organically grows in response to users' activities (O'Reilly, 2005).

The blog and discussion tools enabled harnessing collective intelligence of learners in the IPC. There were structured activities scattered through the semester for this purpose. Such a regular learner task was learners' posting to the discussion board weekly about what they did not understand about that week's topic and whether they wanted to follow-up any additional issue out of curiosity. As an example, the topic for week thirteen was about debugging, and postings about this topic generated 228 messages in a single week. These messages contained 40 questions initiated by 30 different individuals and 188 answers responded by 47 different individuals. All 40 questions for this week were answered. Clearly, it was almost impossible for the instructor to offer this opportunity to 81 students because of time constraints but collective power of learners created an extended learning opportunity.

Harnessing collective intelligence of users in Web 2.0 applications requires the encouragement of users towards participation. There are three methods for such encouragement. The first one is the method of P2P services in which every participant is a natural contributor. Another method is to pay users for their contribution. A third method is to find volunteers for their contribution just like occurring in open source communities. The learners in the IPC were encouraged to share their collective intelligence by a combination of these three methods. The participation of learners was frequently asked and encouraged both in class and through e-mail messages. In addition, an assessment method that is based on participation was followed. For example, 16% of the overall course grade for a learner was assessed based on the contribution of that learner to the discussion board topics.

Learners were asked to put together a personal web page at the beginning of the semester to foster a feeling of collectiveness and community. These pages included a learner's name, e-

mail address, prior experience related to programming and using various software, prior courses taken related to programming, and learner's expectations from the course, along with a picture. These personal web pages were published in the course web space and they were accessible to all learners.

Data is the next intel inside (added value). Database management is a core prerequisite of Web 2.0 applications. As an example, multiple services provide mapping services through the Internet, such as MapQuest, Yahoo, Microsoft, and Google. Other companies can license the same map data through providers and provide a similar service. In this sense, the extension of data through added value is essential for Internet applications to gain competitive advantage over others (O'Reilly, 2005). For example, although MapQuest was the first mapping service emerged on the Internet, it obliged to share the users with other similar emerging service providers. Google Earth, one of these emerging services, on the other hand added value to the same data with such extensions as community layers, in which users could tag locations on the maps. Had the MapQuest added this kind of value, it would have been very difficult for the competitors to enter the market just by licensing the map data.

This kind of value was added to the IPC by blog and discussion tools. Useful information was presented by learners especially through the discussion board. As an example, one of the learners informed other community members in the fifth week of the course about the availability of a very rich code searching service that even the course instructor was unaware of. After learning about the availability of this service, the instructor immediately e-mailed other learners to ripple this added value.

End of the software release cycle (end of the exam-based assessment). Software companies working according to the Web 2.0 principles deliver a service and not a product. For that reason, the software needs to be constantly updated. As an example, Flickr releases new versions in every half hour (May, 2005). In this continuous release cycle, users are treated as co-developers. Their participation is constantly monitored to see which new features are used, and how they are used. This software release model produced the concept of "perpetual beta," in which software is released on a very frequent basis (O'Reilly, 2005).

In Learning Programming 2.0 framework, learning of a student in the IPC continued on a frequent basis since the performance of the learners were assessed not just through a single end-of-the-semester final exam but also through such "perpetual" means as research assignments, weekly blog entries, discussion topics, and lab assignments. Besides, the nature of the exams measured more than memorization. For example, the mid-term exam was take-home type, spanned to a whole week, and required reporting the creation of an authentic programming task. The final exam included many different types of questions while avoiding "write code on paper that will do this" type of questions. In addition, the grade weight of the final exam over the course grade was minimal.

In perpetual assessment of learners, their participation in tasks and activities were valued and learners were perceived and treated by the course instructor as co-developers of the course through the artifacts they produced and contributed. For this purpose, the organization of the course was loosely structured in order to flexibly integrate contributions by learners. For example, one of the learners posted to the discussion board articles related to programming algorithms in the eleventh week of the course. Coming out in a national computer magazine, the articles provided a useful knowledge base in an appropriate technical language suitable for novices. Although the topic of algorithms was covered in the sixth, seventh, and eight weeks of

the course, the instructor decided to utilize these artifacts to consolidate learners' knowledge and skills. For this purpose, the learners were assigned the task of reading these articles and posting a one-page summary and critique to the course blog by the following week. One of the learners indicated at the end of the semester reflection that "*reading these articles changed [his] perspectives into programming*." The instructor has continued to use these articles in the following iterations of the IPC with a similar effect on learners.

Lightweight programming models (lightweight course management models). Web 2.0 applications usually utilize simple programming technologies. These simple technologies allow for hackability and remixability with other technologies, since they don't prevent re-usability. As an example, the application of housingmaps.com was built simply by merging together two existing services. Web 1.0 components are everywhere; but Web 2.0 applications create value simply by assembling them in novel or effective ways (O'Reilly, 2005).

In the IPC, a lightweight course management model was embraced for the utilization of Internet tools. A fragmented structure was utilized to accomplish this instead of using a Learning Management System (LMS). Blogging and discussion board tools and other resources such as course syllabus, PowerPoint lectures, lab assignments, library resources, online books, and personal web pages were accessible through the course Web space (Figure 1). The course Web space was actually a browsable directory structure on the instructor's personal homepage on the university's web server. The lack of a course Web site, Web page, or course LMS prevented the complexity of the course management; the instructor was easily able to extend the tools and resources available to learners by dragging and dropping them to the course Web space. This remixing of the simple existing tools and content created a synergy without creating extra hassle for the learners:

"Programming Languages I" course was such a planned course from head to toe. I conceptualize this course as a very well ordered home context. We could find anything we needed since everything was in place. Catering our needs just-in-time did not exhaust us since we did not have to search for them room by room. Fragments in the course Web space were like rooms. With its discussion board, blog, and library, the course Web space resembled the kitchen, living room, and hall of a house.

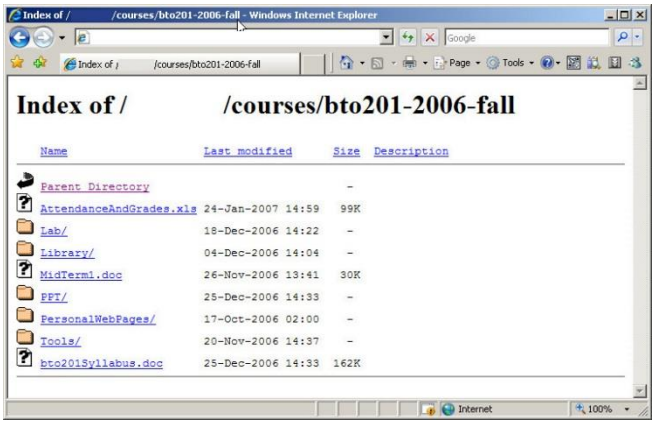

Figure 1. The course web space

Software above the level of a single device (learning above the level of brick-and-mortar). Web 2.0 applications are not limited to just the PC platform. iTunes is a good example of this principle. iPod/iTunes combination spans multiple devices (O'Reilly, 2005). In Programming 2.0

framework, this principle takes the form of "learning above the level of brick-and-mortar." In the IPC, learners participated in activities both in class and out of class.

The impact of this extended participation was clearly visible. One learner indicated that "*they extended the course into other times through the opportunities provided by the Internet.*" Another one indicated "*the course broke the time limitations and provided opportunities out of class such as searching solutions to problems and getting feedback.*" Another one summarized the overall impact of these Internet tools: "*All these tools connected me to the course even at home.*"

Rich user experiences (rich learner experiences). Web designers are able to build user interfaces for Web 2.0 applications as rich as local PC applications (O'Reilly, 2005). Similarly, the Internet tools afforded a rich and active learning environment advocated by most recent learning theories:

The course was not built on a single learning resource, it guided us to research. We were not used to this method, and it was very different from learning from a single resource that we were used to. I believe this method was quite constructive and useful; because while searching through these resources you are trying to find what you need by carefully reading them, which makes you think.

Implementation Problems

An analysis of learners' access to IT outside the school revealed that the majority (90%) had access to a computer with Internet access while a small number (10%) did not. Those learners without IT access solved this problem by using public Internet cafes:

Since I did not have a computer at home, I was not able to use the discussion board effectively. For my assignments, I had to drink many teas at Internet cafes :)

Most of the learners critiqued getting credit towards discussion board participation, and indicated this strategy resulted in redundancy of information:

I think the discussion board diverted from its main purpose. Students sent the same information over and over again to get credit. I witnessed many times that a question was answered by the same multiple responses.

Some learners however developed strategies to overcome redundancy in the discussion board:

I think the discussion board was effectively used. Many friends shared [with each other]. I was not able to read them all, but by being selective I learnt a lot.

A common concern was about the submission of assignments by e-mail:

Sometimes the e-mail system has problems, like the server going down, and etc. As a result, our assignments sent by e-mail don't go to their target, and although we sent it, it is as if we did not, and this creates an anxiety on us.

To overcome this anxiety, the instructor sent out an e-mail message to the whole class after the due date of an assignment, acknowledging received and absent submissions. Related to this regular interaction between the instructor and learners, the single biggest difficulty of Learning Programming 2.0 framework from the perspective of instructor was the extensive time requirements for guidance and assessment of this immense learning community. Learners were also aware of this requirement and acknowledged the burden of the instructor:

Our instructor insistently avoided [following] traditional didactic teaching method, and I think this was good. I can say he wore himself out; he was constantly giving us feedback.

Following, controlling, and grading all of our activities should be very time consuming and tiresome.

I deem the real difficulty was on the shoulder of the instructor. Addressing the community and accurately guiding them is a quite difficult task.

DISCUSSION AND IMPLICATIONS

The purpose of this study was to integrate ubiquitous Internet tools in an undergraduate introductory computer programming course to overcome mechanical and sociological barriers in students' learning. Course web space, blogs, discussion board, e-mail, and personal web pages were among the Internet tools utilized by the participants. Web 2.0 framework was used as an analytical lens for characterizing the use of Internet tools in the IPC. The findings point towards the issues discussed below.

The use of Internet tools in the IPC has been successful in addressing the previous underlying problems like motivation, attendance, and achievement. In terms of motivation, Perkins, Hancock, Hobbs, Martin, and Simmons (1989) classify novice learners of programming as "stoppers" and "movers." When faced with a problem, stoppers discontinue the task while movers persist in spending effort. The use of Internet tools in the IPC impacted the learners to have a positive attitude towards learning about programming, which in turn prevented them from becoming stoppers. In terms of attendance, just one learner dropped-out early in the semester and this was a student dropping-out all of his other courses. In terms of achievement, just one learner was graded less than 50% (failed according to university grading criteria), 7 learners were graded between 50% and 65% (passed if learner's overall GPA was over 1.80 out of 4.00), and 73 learners were graded over 65% (passed). Considering that some contexts of programming education have an achievement rate of as low as 32% and a drop-out rate of as high as 52% (Beise, Myers, VanBrackle, & Chevli-Saroq, 2003) these results are quite remarkable. The use of Internet tools in parallel to the Web 2.0 framework created a field of attraction in the course context. As a result of this attraction, learning programming was meaningful for learners, attendance increased almost to perfection, and learners' achievement rate increased.

Learning programming is a complex process that requires mastering concept of programs, relationship between computer hardware and programs, mechanics of programming languages, general structures such as conditionals and loops, and additional skills such as problem solving, testing, and debugging (DuBoulay, 1989). It is so difficult to cover all these components at once in the limited time frame of an IPC. For that reason, having incremental opportunities for exploration is essential in programming education to handle the complexity of learning about many components (Green, 1990). The Internet tools extended the boundary of the formal IPC in terms of time and place while the learners had incremental and distributed opportunities for exploring and mastering many programming components through these tools.

Two of the Internet tools deserve special consideration with their pedagogical affordances for teaching and learning programming. The blog tool afforded the learners the opportunity to reflect on their understandings. Learners used their natural language while explicating about the learning activities, which is deemed as important in learning programming languages (Gries, 2006, Panell 2003). While blogging, the learners speculated about why they were learning and how to use this learning, and also made connections with their past learning. Their discourse in

the blogs also informed the instructor about any misconceptions they constructed, and provided an opportunity for modification. Both the discussion board and blog tools not only increased the interaction among learners, but also afforded learners helping each other's learning, and looking and using each other's work as suggested by Kelleher and Pausch (2005). This way, they helped with the formation of a Community of Practice (CoP, Wenger, 1998) among the learners of programming. The perception of community was such evident in learners' reflection responses that most of them answered the questions from the first person plural perspective (i.e., we). Further, the community aspect was so strong among the learners that it eliminated the barrier between two different sections of the course that were on different days of the week. Additionally, the instructor being a part of this community and his becoming familiar with the members through the trails in blogs, discussion board, and personal web pages not only motivated learners but also afforded the instructor to personalize learning to many different personalities.

Approximately ten years are required to turn a novice programmer into an expert one. For that reason, a novice programmer becoming an expert is not possible in a four-year IT program (Winslow, 1996). Related to this limitation, literature suggests that the focus in programming education should be on student's learning instead of instructor's teaching to foster reflective and life-long learners (de Raadt, 2007; Robins, Rountree, & Rountree, 2003). The Internet tools utilized in the IPC facilitated the formation of a learner-centered context while increasing the awareness of learners into the importance of being life-long learners.

CONCLUSION

This study showed that, in addition to being useful for distance education, ubiquitous Internet tools can increase the quality and effectiveness of face-to-face "proximal education." Although Learning Programming 2.0 engages the facilitator with many tasks and commitments over just lecturing in class and just organizing a mid-term and a final exam, the framework pays off by helping with the formation of a learning context in which programming education is meaningful, effective, and life-long for learners. As the continuity and advancement of modern society is becoming more and more dependent upon software and people programming them, innovative frameworks and pedagogies for programming education needs to be conceptualized, applied, verified, and extended. Learning Programming 2.0 framework is such an effort in this direction and it is expected that future work will verify and extend this grounded framework.

ACKNOWLEDGEMENT

Öğretim Teknolojileri alanında A.B.D.'de doktora eğitimi için 1998 yılında kazandığım bursu değerlendirip değerlendirmemeyi düşünürken alan hakkında verdiği bilgiler ve yönlendirmeleri ile rehberlik yaparak mesleki gelişimime katkı sağlayan Sayın Prof. Dr. Halil İbrahim YALIN Hocama teşekkür eder, kendisine mutluluklar dilerim.

REFERENCES

Beise, C., Myers, M., VanBrackle, L., & Chevli-Saroq, N. (2003). An examination of age, race, and sex as predictors of success in the first programming course. Journal of Informatics Education and Research, 5(1), 51-64.

Berland, M., Martin, T., Benton, T., Smith, C.P., & Davis, D. (2013). Using learning analytics to understand the learning pathways of novice programmers. Journal of the Learning Sciences, 22(4), 564-599.

Bishop-Clark, C., Courte, J., & Howard, E.V. (2006). Programming in pairs with alice to improve confidence, enjoyment, and achievement. Journal of Educational Computing Research, 34(2), 213-228.

Boyle, T., Bradley, C., Chalk, P., Jones, R., & Pickard, P. (2003). Using blended learning to improve student success rates in learning to program. Journal of Educational Media, 28(2-3), 165-178.

Burke, Q., & Kafai, Y.B. (2010). Programming & storytelling: Opportunities for learning about coding & composition. In Proceedings of the 9th International Conference on Interaction Design and Children (pp. 348-351), Barcelona, Spain.

Creswell, J.W. (1998). Qualitative inquiry and research design: Choosing among five traditions. Thousand Oaks, CA: Sage.

de Raadt, M. (2007). A review of Australasian investigations into problem solving and the novice programmer. Computer Science Education, 17(3), 201-213.

DuBoulay, B. (1989). Some difficulties of learning to program. In E. Soloway & J.C. Spohrer (Eds.), Studying the novice programmer (pp. 283-299). Hillsdale, NJ: Lawrence-Erlbaum Associates.

European Schoolnet (2015). Computing our future: Computer programming and coding - Priorities, school curricula and initiatives across Europe. Retrieved July 20, 2019 from http://www.eun.org/resources/detail?publicationID=661.

Ge, X., Thomas, M.K., & Greene, B.A. (2006). Technology-rich ethnography for examining the transition to authentic problem-solving in a high school computer programming class. Journal of Educational Computing Research, 34(4), 319-352.

Glaser, B.G., & Strauss, A.L. (1967). The discovery of grounded theory: Strategies for qualitative research. Chicago, IL: Aldine.

Green, T.R.G. (1990). Programming languages as information structures. In J.M. Hoc, T.R.G. Green, R. Samurcay, & D.J. Gillmore (Eds.), Psychology of programming (pp. 117-137). London: Academic Press.

Gries, D. (2006). What have we not learned about teaching programming?. Computer, 39(10), 81-82.

Kalelioğlu, F., Gülbahar, Y., & Doğan, D. (2017). Teaching how to think like a programmer: Emerging insights. In H. Özçınar, G. Wong, & H.T. Öztürk (Eds.), Teaching Computational Thinking in Primary Education (pp. 18-35). Hershey, PA: IGI Global Publishing.

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

Malik, A.M. (2000). On the perils of programming. Communications of the ACM, 43(12), 95-97.

May, C. (2005). Deploy every 30 minutes: Redux, retrieved June 29, 2007 from http://blogs.warwick.ac.uk:80/chrismay/tag/flickr.

Nasa facts and figures. (2019). Retrieved July 20, 2019, from https://www.nasa.gov/feature/facts-and-figures.

O'Reilly, T. (2005). What is web 2.0: Design patterns and business models for the next generation of software, retrieved June 29, 2007 from http://www.oreillynet.com/lpt/a/6228.

Panell, C. (2003). Teaching computer programming as a language. Techdirections, 62(8), 26-27.

Perkins, D.N., Hancock, C., Hobbs, R., Martin, F., & Simmons, R. (1989). Conditions of learning in novice programmers. In E. Soloway & J.C. Spohrer (Eds.), Studying the novice programmer (pp. 261-279). Hillsdale, NJ: Lawrence-Erlbaum Associates.

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

Saygıner, Ş., & Tüzün, H. (2018). Programlama eğitimi üzerine bir inceleme: Yaşanan zorluklar, mevcut uygulamalar ve güncel yaklaşımlar. In B. Akkoyunlu, A. İşman, & H. F. Odabaşı (Eds.), Eğitim Teknolojileri Okumaları 2018 (pp. 693-710). Ankara, Turkey: Pegem Akademi.

Wenger, E. (1998). Communities of practice: Leaning, meaning, and identity. New York: Cambridge University Press.

Winslow, L.E. (1996). Programming pedagogy-a psychological overview. ACM SIGCSE Bulletin, 28(3), 17-22.

Zin, A.M., Idris, S., & Subramaniam, N.K. (2006). Implementing virtual pair programming in e-learning environment. Journal of Information Systems Education, 17(2), 113-117.