Adaptive access to examples in NavEx-2

Michael V. Yudelson
School of Information Sciences
University of Pittsburgh

1. NavEx-1

NavEx-1 (Navigation to Examples) is a system that explores the idea of providing adaptive navigation support for accessing programming examples. NavEx helps the student in selecting the "right" example by applying adaptive prerequisite-based annotation, a specific adaptive navigation support technology [1]. This technology was first explored in ISIS-Tutor system [2] and found very efficient. Since that, it was used in ELM-ART [3], InterBook [4], ACE [5], and other systems. With this technology, each learning object is described in terms of domain concepts. Some concepts serve as learning goals and others as prerequisites to understanding the object. Tracking the student current level of knowledge, an adaptive system can dynamically classify examples as being relevant or not (i.e., to simple or not ready to be studied). The status of each object is shown by adaptive annotation using a traffic light metaphor.

1.1 NavEx-1 interface

The interactive window of the NavEx system is divided into 3 frames (Figure 1). The leftmost frame contains a list of links to all examples/dissections available for a student in the current course. The links are annotated with colored bullets. Red bullet means that the student has not mastered enough prerequisite concepts to view the example. The link annotated with the red bullet is thus disabled. Green bullet means that the student has enough knowledge to view the example. Green check mark denotes example that has already been seen by the student. Green "play" bullet denotes the example that is currently being viewed. The order of links to examples is fixed, so that students can find them at the same place in spite of their progress in the course.

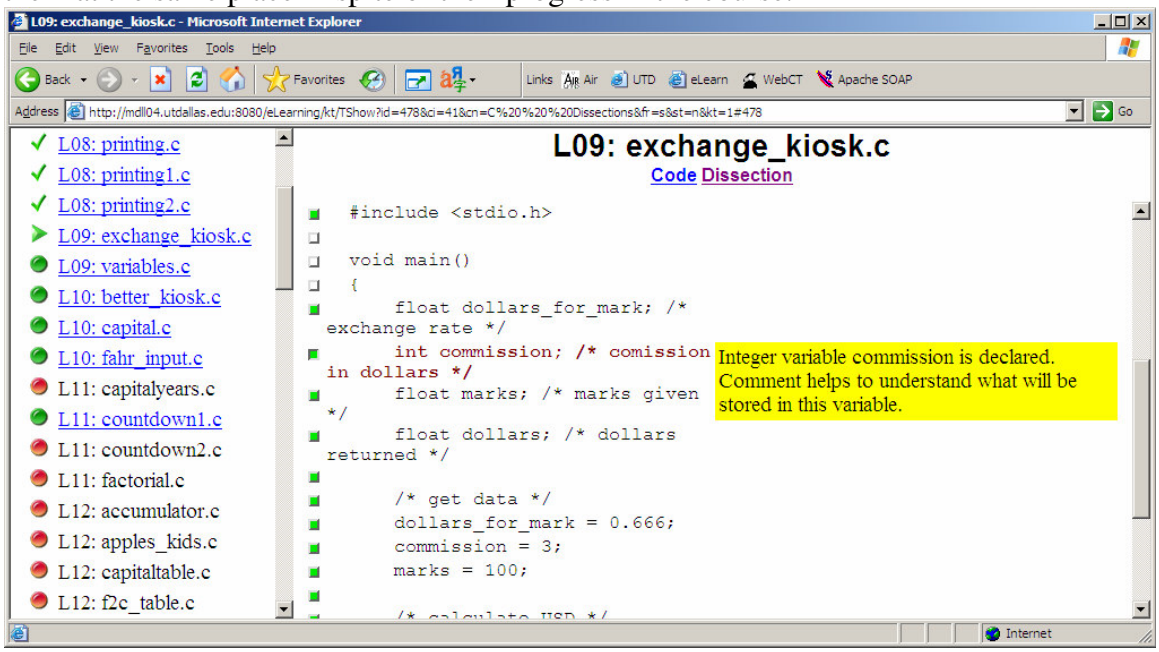

Figure 1. The interface of NavEx-1

The upper frame displays the name of the current example. Underneath are two links: one loads the source code of the example to the central frame (to be copied, compiled, and explored), the other – loads interactive example *dissection* (served directly by the WebEx

system that is now a component of NavEx). Dissection is the same source code commented by the author. These comments address the meaning and the purpose of this line of the code and help the student to understand the example better. Extended comments are shown to the left form the code and can be activated by clicking on the bullet next to the line of the code. If the comment is available the bullet is green and is white otherwise.
NavEx interface is implemented as a server-side solution written in Java. All knowledge and data are stored in a relational database. NavEx is considered as a value-added service of the KnolwedgeTree architecture [6] and implements several common protocols including student modeling and transparent authentication.

1.2 Semantic indexing of examples in NavEx-1

To provide learning material for the student in adaptive way, a system needs to have knowledge about the material itself. Indexing learning material with domain knowledge is a time-consuming process requiring expertise in both the domain of learning and knowledge engineering [7]. In NavEx we explored an alternative automated approach. All examples in the scope of the course are indexed in terms of used concepts, after then the concept taxonomy, based on inter-concept-example relations "prerequisite-outcome" is built. Current section describes this process in details.
There are no universally accepted recommendations on which level it is better to define a concept in programming domains. Some authors suppose that it has to be done on the level of programming patterns or plans [8], other believe, that the concepts is to be closer to the elementary operators [9]. From the first point of view the notion of pattern is more adequate to what is the real goal of learning programming and what programmers really use. However, the second way is more straightforward and makes the burden of indexing more feasible. With a notable exception of ELM-PE [10], all adaptive sequencing systems known to us work with operator-level concepts. Current implementation of the indexing algorithm for NavEx also uses operator-level approach.
Traditionally the extraction of a grammatically meaningful structure from programming code and determination of a concept on its basis is the task for the special class of programs named parsers. For NavEx system we have developed the indexing component, which parses C code of each example and generates the list of used concepts. This component is build with the help of UNIX well known utilities lex and yacc.

Fifty-one concepts have been determined for the subset of C language learnt during the course. Each programming structure in an example can be indexed by one or more concepts depending on the amount of knowledge student needs to involve for its understanding.
It is necessary to mention that each concept here represents not simply a keyword, found in code, but a grammatically complete programming structure. For instance, concept *while* is recognized by the indexing algorithm only after the whole loop structure, including the keyword *while*, the iteration condition and the loop body, is found. This is why concept *while* in the index list above follows concepts *not_equal_expression* and *pre_increment*.
The next stage is dividing concepts related to each example into prerequisite and outcome concepts. Prerequisites are the concepts that the student needs to master before starting to

work with the example. Outcomes denote concepts that the example helps to learn, i.e. the pedagogical goals of the example. As any automatic knowledge extractor, our grammar-based extractor is unable to distinguish prerequisites and outcomes - it simply produces the list of all concepts. For adaptive sequencing and navigation support, however, clear separation of prerequisites and outcomes is vital.

NavEx uses an original algorithm for automatic identification of outcome concepts (Figure 2). This algorithm adapts to an instructor-specific way of teaching the course. The source of knowledge for this algorithm is a sequence of example groups. Each group is formed by examples introduced in the same lecture. Groups are ordered according to the order of lectures in the course. The design of our prerequisites/outcomes division algorithm is based on the following assumptions:

- While analyzing examples from some lecture, concepts corresponding to examples from all preceeding lectures are considered to be already learnt.
- In each example, all concepts introduced in the previous lectures are considered as prerequisites while the concepts first introduced in the current lecture as outcomes
- The set of new concepts found in all examples associated with the lecture is the pedagogical goal of the lecture

$$
\begin{array}{l}
learnt\_concepts = \varnothing \\
for\ i = 1\ to\ no\_of\_chapters \\
\{ \\
\quad for\ j = 1\ to\ chapter[i].no\_of\_examples \\
\quad \{ \\
\qquad chapter[i].example[j].prereq = learnt\_concepts \cap chapter[i].example[j].all\_concepts \\
\qquad chapter[i].example[j].outcome = chapter[i].example[j].all\_concepts \setminus learnt\_concepts \\
\quad \} \\
\quad for\ j = 1\ to\ chapter[i].no\_of\_examples \\
\qquad learnt\_concepts = learnt\_concepts \cup chapter[i].example[j].all\_concepts \\
\}
\end{array}
$$

Figure 2. Pseudocode for prerequisites/outcomes identification.

The direct outcomes of this algorithm are a fully indexed set of lecture examples and a sequence of learning goals associated with the course lectures. This sequence represents a specific approach to teaching C programming employed by the instructor [7]. Once the course examples are indexed and the goal sequence is constructed, any programming example can be properly indexed by the algorithm and even associated with a specific lecture in the course. More exactly, an association with a specific lecture is the first step in this process. The example is associated with the last lecture that introduces example concepts (i.e., the lecture with the largest number that has at least one example concept in its pedagogical goal). After that, the example is indexed as belonging to this lecture. It is important to stress again that outcome identification is adapted to a specific way of teaching a course "mined" from the original sequence of examples. I has been known for a long time that different instructors presenting the same programming language may use

very different orders of concept presentation [11]. Naturally, example sequencing in a course has to be adapted to the instructor's way of teaching.
Although, the described indexing approach using parsing component is specific for the programming, we believe that the proposed general idea is applicable for the broad class of domains. In less formalized domains, where concepts do not have salient grammatical structure, classic information retrieval approach could be used instead of parsing. Current evaluation shows that implemented indexing algorithm along with the NavEx mechanism of building inter-concept-example hierarchies provide meaningful recourse for adaptive example navigation support.

1.3 Adaptive navigation support in NavEx-1

Adaptation of navigation in NavEx is done on the basis of the overlay user model [12]. User knowledge is represented as a binary vector k, where k_i=1 means that the user has successfully mastered concept i, and k_i=0 means the opposite.
When the user logs into the system a new session is created and information about current state of his/her knowledge is retrieved from the user records. This information contains concepts the user has mastered and examples the user has reviewed.
Knowing the user knowledge of each domain concept and the prerequisite-outcome profile of examples, the sequencing mechanism can dynamically compute current educational status for each example. The current status of each example is presented to the user in the leftmost frame of the system window (Figure 1) in the form of adaptive annotations. Already mastered examples are annotated with green check marks, available examples are annotated with green bullets, unavailable – with red bullets.
When user clicks on example link and reviews the example code the outcome concepts of the example change their state to "learned" (k_i=1). The changes in the knowledge state of the user are then propagated to the user records. The availability of new examples is determined by checking whether any of the previously unavailable examples now have all of their prerequisite concepts mastered. As user reviews examples more new examples become available. The knowledge-based adaptive annotation approach used in NavEx is a variation of a popular adaptive annotation approach introduced originally in ISIS-Tutor system. This approach is known to be very efficient [2].

1.4 Identified problems in NavEx-1

1.4.1 'Clicked–knows' problem

One of the most serious NavEx-1 problems is that the user is considered to "know" the example as soon as s/he clicks on it. This oversimplification does not take into consideration many factors such as: whether user just clicked on the example or thouroughly went through it, whether this is user's first visit or user returns to an example. Besides, clicking once on an example is surely not enough. Some of the examples have many lines of code and a single click wouldn't be sufficient for the example to be considered known and the according concepts learnt.

1.4.2 Adaptive annotation problem

The adaptive annitation bullets used in NavEx-1 have some room for improvement. The existing set of green ball, red ball, green check mark does not reflect the progress of the

user with an individual example. Besides icons are not suitable for color blind or partially color blind persons.

1.4.3 Navigation constraint

Navigation through examples in NavEx-1 is cosntained: examples that do not have all of the prerequisite concepts learned are not available for viewing. This is not quite correct since such constraint doesn't allow users make their own decisions. Adaptive navigation is should provide only guidance and users should me their own judges in making decisions whether to follow that guidance.

2. NavEx-2

NavEx-2 is a further development of the ideas of adaptive guidance to the code examples indroduced in NavEx-1. NavEx-2 is a deep modification of its predecessor designed to overcome the shortcomings discussed above.

2.1 NavEx-2 interface

The interface of NavEx-2 sticks to the same 3-frame design. The design of the frames, however, is a bit changed (Figure 3).

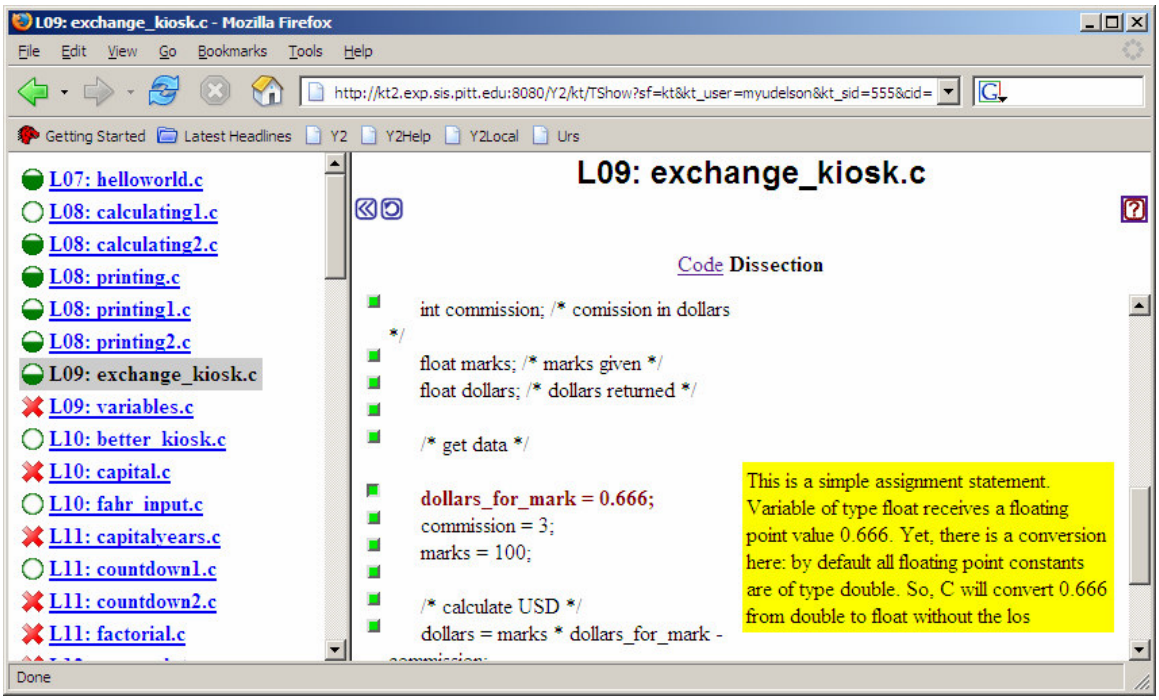

Figure 3. The interface of NavEx-2

Each link to an example in the left frame is supplied with an icon that conveys information about (1) 'readiness' of the student to browse the example, and (2) the student's progress within the example. If the student is 'not ready' to browse the example then a red X bullet is displayed (Figure 4). If the student is 'ready' to browse the example then a green round bullet is shown. Depending on the student's progress, the green bullet will be empty, partially or wholly filled. There are 5 discreet progress measures from 0% to 100%, with 25% increments (Figure 4). An empty green bullet denotes examples that

are available, yet not browsed by the student. The relevance of the example is marked by the font style. If the example is relevant its link is displayed in bold font, otherwise it is in regular font (Figure 3). The fact that the example is 'not ready' or 'not recommended' doesn't prevent the user from actually browsing it. All of the annotated examples are available for exploration and it is up to a student as to whether to follow the suggestions expressed by annotations or not.


Not ready to be browsed Ready to be browsed

Figure 4. Annotation of the examples

2.2 Adaptive navigation support in NavEx-2

The annotation of examples is compiled, based on the domain model concepts. Each of the examples is indexed with such concepts before it is added to the system. The indexing goes through two stages. First, concepts are extracted from each of the examples by a fully-automatic operation-level parser. Second, for each of the examples, the set of concepts is split into prerequisite concepts and outcome concepts. The splitting algorithm, besides example-concept pairs, requires examples to be grouped by lecture. Indexing algorithms are discussed in more detail in [13]. Supplying each example with two sets of concept - prerequisites and outcomes – plays a two-fold role. First, the concept separation helps to define the learning goals (focus) of the examples in terms of outcomes. Second, concept separation is used for partial ordering of the examples. Thus, an example that has a certain concept as a prerequisite will be placed after an example that has the said concept as outcome.

Once the example is in the system, its annotation for the current user is determined by counting whether or not the current user has mastered the prerequisite concepts. If all of the prerequisite concepts are mastered (or the example simply has no prerequisite concepts) – the example is considered 'ready to be browsed.' If the prerequisite concepts are not mastered – the example is marked as 'not ready to be browsed'. The progress of the student within the example is measured by counting the number of clicks on annotated lines of code example code the user has done with the example.

The relevance of the examples is calculated based on the 'threshold' parameter. The 'threshold' (calculated for each of the examples individually) is the amount of clicks that has to be done by student for the system to conclude that s/he 'knows' the example and declare all of concepts corresponding to example to be mastered. The threshold amount of clicks is calculated as:

$$threshold = 0.8*[(all\_concepts - mastered\_concepts)/all\_concepts]*clicks\_possible$$

Namely, the total number of clicks possible (for current example) is multiplied by the ratio of currently not-mastered concepts (of the current example) to all concepts (of the current example). This gives the number of clicks 'left' for user to do and he has to make 80% of those to 'master' the example. Only clicks on distinct code lines are counted. E.g. if there are 10 clicks possible on the lines of the code example and there are 10 concepts assigned to the example: 5 prerequisite (all mastered) and 5 outcomes (none mastered), then the user has to make 0.8 * (5/10) * 10 = 4 clicks to 'master' the example. As soon as

some concepts are declared mastered the 'readiness' of all other examples is recalculated and the mastery of the concepts is propagated further.
The threshold is only used to determine the minimal amount of work the student has to do with the individual example to learn the underlying concepts. The annotation of the examples reflects the absolute amount of student's work and is not related to the threshold. Since all of the examples share the pool of concepts, it might turn out that at some point there will be one or more examples whose concepts are mastered, yet the student has never browsed those. As mentioned in a previous section, students can browse examples that are annotated as 'not ready to be browsed'. In extreme cases, the student can browse an example, which contains only concepts that are not yet mastered. To master those concepts while browsing such an example, the student will have to do an extensive amount of clicks, as determined by the threshold.
The NavEx interface is implemented as a server-side solution written in Java. All knowledge and data are stored in a relational database. NavEx is considered to be a value-added service of the KnowledgeTree architecture [14], and uses several protocols, including student modeling and transparent authentication. As a typical value-added service, NavEx resides between E-Learning portals and reusable content objects, providing additional value for teachers and students who use this content through the portal. Unlike other kinds of value-added services, such as annotation services, the value added by NavEx is the ability to adapt to the course goals and student knowledge. With NavEx, teachers can bypass the time-consuming process of selecting examples for each course lecture that meet goal and prerequisite restrictions. Students receive adaptive guidance in selecting examples that are most relevant to their learning goals and knowledge.

3. Classroom study of NavEx-2

A classroom study of NavEx was performed in the context of an undergraduate programming course in the Fall 2004 semester in the School of Information Sciences at the University of Pittsburgh. NavEx was made available to all students taking this course in the second half of the semester, after the midterm exam. Before the introduction of NavEx the students were able to explore code examples with the original WebEx (i.e., without adaptive guidance) directly through the Knowledge Tree portal. After the introduction, they were able to use both methods of access – with adaptive navigation support through NavEx and without it through the portal and WebEx. User activity collection procedures does not depend on the way students access code examples. Student work with both WebEx and NavEx was equally considered for user modeling.

3.1 Log Analysis

Our main source of data for the study was the user activity log. The log recorded every user click (i.e., every example and code line accessed). Note that the log data gave clear evidence as to whether a student accessed a specific example through NavEx or through WebEx. Since students used WebEx and NavEx in parallel (the use of NavEx was not enforced), a natural way to evaluate the influence of adaptation was to compare the usage profiles of WebEx and NavEx. Analysis of the data showed that NavEx, though introduced late in the course, was considered as a strong alternative to WebEx. After the

introduction of NavEx, 56% of example browsing activity was generated by NavEx users. Only 30% of the students didn't use NavEx at all.

Since different students used different "mixtures" of WebEx and NavEx through the course, we decided to assess the added value of the adaptive navigation support by comparing these two systems on a session-by-session basis. A session is counted as a sequence of examples browsed by the student without any sizeable break. The result of this comparison demonstrated clearly the value of adaptive navigation support in increasing the amount of student work with examples.

First, the average session of non-NavEx users was 9.4±0.97 clicks, while NavEx users made an average of 29.6±4.65 clicks per session. That means that navigation support provided by NavEx encouraged students to click on 3.14 times more annotated code lines. Second, the average number of examples browsed per session of non-NavEx users was 1.78±0.15, while NavEx users browsed 2.95±0.46 examples per session. Thus NavEx motivates students to see an average of 1.66 examples more per session. And thirdly, the average length of the non-NavEx user session is 225±33 seconds, while NavEx users have average session length of 885±266 seconds. Hence NavEx keeps students focused on examples 3.9 times longer.

Further evidence can be derived by comparing the example browsing statistics of Fall 2004 semester, when students could use adaptive guidance and Spring 2004 when they could not. Examples set in the Spring 2004 semester had only minor differences from the set of examples available in the Fall 2004 so we can assume that the students had the same external (i.e., tool-independent) motivation to use the tool. The only significant difference was that in the Fall 2004 semester students were able to use NavEx.

The comparison of student activity data of the two semesters demonstrated that the introduction of NavEx boosted the motivation of the students to work more with annotated code examples. The number of code lines accessed per session increased by about 11% from 14.22 in the Spring 2004 semester to 15.8 in the Fall 2004 semester (if we consider only NavEx users the number of clicks per session almost doubled). The average number of line accesses by students over a semester grew by 35% from 323.3 lines in the Spring 2004 semester to 435.9 in the Fall 2004 semester.

Thus, adaptive navigation support succeeded as a tool that encourages the students to work more with examples. It appears that the students were simply more motivated to work with examples when adaptive navigation support was provided. We think that such increase of students' motivations can be attributed to the following reasons. First, navigation support allows students to see 'the big picture' – visualize their current progress with all of their examples and estimate whether the progress they made is enough to move further. Second, since students had all the examples grouped together, they were able to switch from one example to another in fewer clicks and were interested in exploring more examples.

3.2 Survey results and analysis

The user survey was conducted using a questionnaire containing 13 multiple choice questions, 1 multiple answer question and 1 open ended question. The goal of the questionnaire was to judge the effectiveness and the usabilityof the system in providing adaptive guidance and to judge the effectiveness of the interface design features implemented.

The questionnaire was largely based on the questionnaire used to evaluate the WebEx. New questions were added to evaluate the features of NavEx. The system was evaluated by the students of the C Programming class offered at the School of Information Sciences at University of Pittsburgh. Out of the 15 students that took the class, 12 used WebEx, 11 used NavEx, and 10 of those students responded to the survey.

3.2.1 Survey results 'in-the-raw'

Below is the detailed summary of the survey results.

Question 1 Multiple Choice Average Score: 0 point(s)

Annotated examples can significantly improve my understaning of class material

Answers	Percent Answered
Strongly agree	50.00%
Agree	40.00%
No opinion	0.00%
Disagree	10.00%
Strongly disagree	0.00%
Unanswered	0.00%

Question 2 Multiple Choice Average Score: 0 point(s)

I think that interface of the annotated examples was good

Answers	Percent Answered
Strognly agree	30.00%
Agree	70.00%
No opinion	0.00%
Disagree	0.00%
Strongly disagree	0.00%
Unanswered	0.00%

Question 3 Multiple Choice Average Score: 0 point(s)

The content of annotations in the examples was good and helpful

Answers	Percent Answered
Strognly agree	30.00%
Agree	50.00%
No opinion	10.00%
Disagree	10.00%
Strongly disagree	0.00%
Unanswered	0.00%

Question 4 Multiple Choice Average Score: 0 point(s)

The interacivity of the dissections (ability to click on selected lines and see

comments for those lines) was important for me.

Answers	Percent Answered
Strognly agree	50.00%
Agree	50.00%
No opinion	0.00%
Disagree	0.00%
Strongly disagree	0.00%
Unanswered	0.00%

Question 5 Multiple Choice Average Score: 0 point(s)

For every lecture of the course I want to have:

Answers	Percent Answered
Dissections of all classroom examples	10.00%
Dissections of all classroom examples and a few extra dissections	10.00%
All of the above and dissections of howework solutions	80.00%
I do not need any dissections	0.00%
Unanswered	0.00%

Question 6 Multiple Answer Average Score: 0 point(s)

If I would have a handheld computer with wireless internet access, I'd be interested to work with example dissections

Answers	Percent Answered
when I am doing my homework	60.00%
right in class during the lecture when examples are presented	70.00%
whenever I have some spare time	50.00%
when I meet and talk with other students from my class	50.00%
I am not interested to work with dissections on a handheld computer	20.00%

Question 7 Multiple Choice Average Score: 0 point(s)

Imagine that dissections adapt to your current level of knowledge (similar to the way WADEIn does) by highlighting the lines with comments that are most important to you to read (thus decreasing the amount of comments to explore). This feature would be very useful.

Answers	Percent Answered
Strognly agree	50.00%
Agree	40.00%
No opinion	10.00%
Disagree	0.00%

Strongly disagree	0.00%
Unanswered	0.00%

Question 8 Multiple Choice Average Score: 0 point(s)

What do you think about an idea of students to be able to create some dissections using a good Web interface?

Answers	**Percent Answered**
I don't like this idea at all	10.00%
The idea is interesting, but I do not think anyone will be interested in it	10.00%
The idea is interesting and this can even be offered as an extra credit assignment	70.00%
The idea is interesting and this can even be offered as one of the regular assignments	10.00%
Unanswered	0.00%

Question 9 Multiple Choice Average Score: 0 point(s)

What do you think about an idea to let students add their own comments to lines of examples' code and view comments of other students?

Answers	**Percent Answered**
This isn't a good idea at all	0.00%
The idea is interesting, but I doubt anyone in class will do this	20.00%
The idea is good and this option can be offered as an extra credit assignment	70.00%
The idea is great and should be offered as a regular assignment	10.00%
Unanswered	0.00%

Question 10 Multiple Choice Average Score: 0 point(s)

I think that interactive program dissections should become one of the key tools in programming classes (including IS12)

Answers	**Percent Answered**
Strognly agree	30.00%
Agree	50.00%
No opinion	20.00%
Disagree	0.00%
Strongly disagree	0.00%
Unanswered	0.00%

Question 11 Multiple Choice Average Score: 0 point(s)

The ability to access all dissections from all the lectures using the joint list of

dissection on the left side of NaveEx interface was helpful

Answers	Percent Answered
Strognly agree	50.00%
Agree	50.00%
No opinion	0.00%
Disagree	0.00%
Strongly disagree	0.00%
Unanswered	0.00%

Question 12 Multiple Choice Average Score: 0 point(s)

The ability to see my own progress in NavEx (the percentage of each example that I have already analyzed that was shown using fillable green bullets) was helpful

Answers	Percent Answered
Strognly agree	30.00%
Agree	40.00%
No opinion	10.00%
Disagree	10.00%
Strongly disagree	0.00%
Unanswered	10.00%

Question 13 Multiple Choice Average Score: 0 point(s)

NavEx estimated whether you were ready to understand a specific example and warned you about not ready to be explored examples using a red X icon. I think that NavEx estimation was mostly correct (i.e., examples shown with red X were, indeed, too complicated, while examples shown with a green bullet were just right)

Answers	Percent Answered
Strognly agree	20.00%
Agree	40.00%
No opinion	10.00%
Disagree	10.00%
Strongly disagree	10.00%
Unanswered	10.00%

Question 14 Multiple Choice Average Score: 0 point(s)

NavEx estimated whether you were ready to understand a specific example and warned you about not ready to be explored examples using a red X icon. Regardless of the correctness of this estimation in the current version of NavEx, I think that it is useful to see "not ready" warning next to too complicated examples.

Answers	Percent Answered
Strongly agree	20.00%
Agree	30.00%

No strong opinion	20.00%
Disagree	10.00%
Strongly disagree	10.00%
Unanswered	10.00%

Question 15 Essay Average Score: 0 point(s)

Please add any comments or suggestions regarding functionality or interface of both WebEx or NavEx.
Add your concerns here too.

Given Answers

0 Unanswered Response(s)

I did not use the NavEx program so I was not able to complete those multiple choices.The WebEx programs were quite useful however.From all the abundant tools in class I did not feel that I had to use the NavEx program as the others were quite useful and plentiful at that.Comments added to dissections could be useful from students who are on the same level and pace however these would have to be monitored to make sure a faulty comment isn't placed that could possibly lead someone astray.Personally, I used the WebEx a bit to analyze and further understand the examples that I did not fully grasp in class so I felt that it complimented the class-room examples quite well.Interactive programming dissections worked best for me when I had trouble understanding a specific program but otherwise I would rarely use them so I personally don't think they should become a center tool for the IS12 class but more so as help for the various examples and assignments.Also, I'm sure many students would benefit if there was a homework section added as well to the WebEx with the previous homeworks dissected.

Possibly a slightly cleaner interface would be useful, although the simplistic approach is quite effective. Retaining simplicity, but maybe adding a way to aggregate(by lecture or content similarity) examples together and then expand them somewhat like the quizGuide, at least for improved navigation.
IN ANSWER TO THE QUESTIONS ABOVE ABOUT LETTING STUDENTS ADD THEIR OWN DISSECTIONS I THINK THAT IS A EXCELLENT BECAUSE ANY BODY CAN JUST COPY SYMBOLS FROM LECTURE IN A PROGRAM BUT UNDERSTANDING WHAT THOSE SYMBOLS DO AND HOW THEY ARE USED IS WHAT IS IMPORTANT IN PROGRAMMING SO I THINK THAT WOULD BE AN EXCELLENT EXTRA CREDIT PROJECT TO DO LETTING STUDENTS DO THEIR OWN DISSECTIONS

I think that interface should have been incorporated into the classroom a long time ago. For example, the beginning of the semester could have been the focus area for reading and learning "C" through the lessons that were online. Then having us do dissections would be a very good idea for extra credit, as well as for learning too!

i felt that it helped me out alot

The webEx and Naxex provided me with a good what of seeing how the prgrams ran by dissecting them. they were also good tools

the only other suggestion I would have is just to break them up from topic to topic or break up the programs by the principals they use instead of from lecture to lecture, at least have this as a sorting option.

It works pretty good.However, disections of the homework would be extremely helpful

I think it would be helpful to have computers in the classroom to work through some of the dissections together.

3.2.2 Survey results analysis

Our secondary source of evaluation data was a non-mandatory questionnaire administered at the end of the term that solicited students' opinions about key features of the system. Out of 15 students in the class, 10 completed the questionnaire.

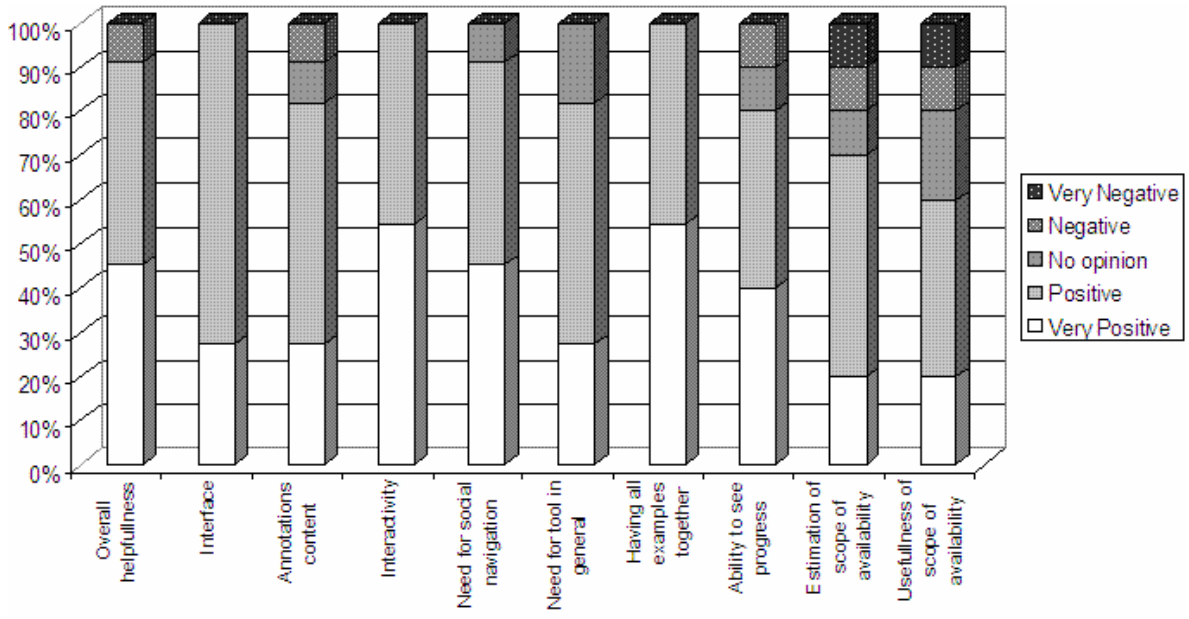

Figure 5. Subjective student evaluation of different features of NavEx

Some of the data obtained from processing the answers is shown in Figure 5. As it can be seen, 90% of students considered annotated examples with or without adaptive guidance

helpful. 80% percent of students feel positive or strongly positive about the need for such a tool in general. All of the respondents positive or strongly positive evaluated the convenience to have all of the annotated code examples together. 100% of students positively or strongly positively evaluated the interface and the interactive nature of examples.

Two principal features of NavEx: progress indicator and the scope of availability ('readiness') were evaluated positively or strongly positively by a solid fraction of the students (80% for progress indicator and 60-70% for the scope of 'readiness'). The slight downfall of positive response about the scope of 'readiness' of examples' annotation we account to the fact that students started with NavEx in the middle of semester. At the time of their first logon, all of the examples were 'not ready to be browsed', yet at that time students were already familiar with almost half of them and had literally to 'get through' the red X's. Nevertheless, they did appreciate the scope of 'readiness' on the whole.

Students also had a chance to express their suggestions about the future use and development of the system. The idea of students being able to create their own dissections or add their own annotations to the code lines was supported by 70% of respondents (when such activity is an extra credit assignment), and strongly supported by 10% (when such activity is a regular assignment). 90% students expressed strong and very strong support for adding a social navigation feature. A substantial amount of students have also expressed certainty that NavEx should remain as one of the class tools available for students.

4. Conclusions and future work

The NavEx system implemented adaptive navigation support to encourage the students to work more with program examples. Our classroom study confirmed that adaptive navigation support can visibly increase student motivation to work with non-mandatory educational content. NavEx boosted the overall amount of work and the average length of a session. In addition, various features of NavEx were highly regarded by the students. Among two kinds of adaptive navigation support, performance-based annotation was appreciated more than zone-based annotation. However, it may have been influenced by the late introduction of the system.

We plan to perform further studies with NavEx to achieve a better understanding of the value of adaptive navigation support. In addition, we plan to extend the scope of adaptive annotation by providing an annotation of every commented line in an example – not only an example as a whole. To make it possible, we will apply social navigation techniques that we are currently exploring in the course of another project.

5. References

[1]. Brusilovsky, P.: Methods and techniques of adaptive hypermedia. User Modeling and User-Adapted Interaction 6, 2-3 (1996) 87-129.

[2]. Brusilovsky, P. and Pesin, L.: Adaptive navigation support in educational hypermedia: An evaluation of the ISIS-Tutor. Journal of Computing and Information Technology 6, 1 (1998) 27-38.

[3]. Weber, G. and Brusilovsky, P.: ELM-ART: An adaptive versatile system for Web-based instruction. International Journal of Artificial Intelligence in Education 12, 4 (2001) 351-384, available online at http://cbl.leeds.ac.uk/ijaied/abstracts/Vol_12/weber.html.

[4]. Brusilovsky, P., Eklund, J., and Schwarz, E.: Web-based education for all: A tool for developing adaptive courseware. Computer Networks and ISDN Systems. 30, 1-7 (1998) 291-300
[5]. Specht, M. and Oppermann, R.: ACE - Adaptive Courseware Environment. The New Review of Hypermedia and Multimedia 4 (1998) 141-161.
[6]. Brusilovsky, P. and Nijhawan, H.: A Framework for Adaptive E-Learning Based on Distributed Re-usable Learning Activities. In: Driscoll, M. and Reeves, T. C. (eds.) Proc. of World Conference on E-Learning, E-Learn 2002, Montreal, Canada, AACE (2002) 154-161
[7]. Brusilovsky, P.: Developing Adaptive Educational Hypermedia Systems: From Design Models to Authoring Tools. In: Murray, T., Blessing, S. and Ainsworth, S. (eds.): Authoring Tools for Advanced Technology Learning Environments: Toward cost-effective adaptive, interactive, and intelligent educational software. Ablex, Norwood (2003).
[8]. Lutz, R.: Plan diagrams as the basis for understanding and debugging pascal programs. In: Eisenstadt, M., Keane, M. T. and Rajan, T. (eds.): Novice programming environments. Explorations in Human-Computer Interaction and Artificial Intelligence. Lawrence Erlbaum Associates, Hove (1992) 243-285.
[9]. Barr, A., Beard, M., and Atkinson, R. C.: The computer as tutorial laboratory: the Stanford BIP project. International Journal on the Man-Machine Studies 8, 5 (1976) 567-596.
[10]. Weber, G. and Bögelsack, A.: Representation of programming episodes in the ELM model. In: Wender, K. F., Schmalhofer, F. and Böcker, H.-D. (eds.): Cognition and Computer Programming. Ablex, Norwood, NJ (1995) 1-26.
[11]. Moffatt, D. V. and Moffatt, P. B.: Eighteen pascal texts: An objective comparison. ACM SIGCSE bulletin 14, 2 (1982) 2-10.
[12]. Greer, J. and McCalla, G. (eds.):Student modelling: the key to individualized knowledge-based instruction. NATO ASI Series F, Vol. 125, Springer-Verlag, Berlin (1993) p.
[13]. Sosnovsky, S., Brusilovsky, P., and Yudelson, M. (2004) Supporting Adaptive Hypermedia Authors with Automated Content Indexing. In: Proceedings of Second International Workshop on Authoring of Adaptive and Adaptable Educational Hypermedia at the Third International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH'2004), Eindhoven, the Netherlands, pp. in press.
[14]. Brusilovsky, P. (2004) KnowledgeTree: A distributed architecture for adaptive e-learning. In: Proceedings of The Thirteenth International World Wide Web Conference, WWW 2004 (Alternate track papers and posters), New York, NY, 17-22 May, 2004, ACM Press, pp. 104-113.