

APP: Academic Planning Program

**Allison Deford and Tyler Hoffman
Computer Science
University of Evansville
April 30, 2014**

Abstract

At the University of Evansville, currently there is no system that adequately keeps track of what courses a student has completed and which ones they must complete before graduation. The College of Engineering and Computer Science commissioned an application that can be used to assist advisors and advisees in this process, especially as the graduation requirements become more complicated. The application should allow students to view classes they have already taken and to plan future classes based on their current major/minor(s). Advisors should be able to view their advisees' plans to allow them to do their jobs more effectively. The developers will create a web-based application that will fulfill these requirements by presenting the user with an easy-to-use system for tracking their academic experience.

Acknowledgements:

The developers would like to thank Dr. Deborah Hwang for acting as both a sponsor and advisor for this project. They would also like to thank Dr. Don Roberts for acting as an advisor.

Introduction:

One of the biggest problems a university must handle is how to track the classes each student takes on their path towards a degree. At the University of Evansville, there have been numerous attempts to create systems to track this adequately, but as it stands, there is no good tool to allow students to plan how they will complete their degrees. The job of making sure students meet all requirements falls to their advisor until their final year. This ultimately leads to some oversights, occasionally even forcing students to graduate later than they expect to. Tracking which students have completed what graduation requirements has been a problem in the College of Engineering and Computer Science for some time. Dr. Deborah Hwang presented the need for an application that solves the advising problem for students and faculty. This project solves this problem by creating a web-based application for students and their faculty advisors that tracks the classes they have taken and allows them to check these classes against the graduation requirements for their given program. This program focuses specifically on the needs of the College of Engineering and Computer Science, with the hope that it can be adapted easily for other colleges within the university.

The Problem:

Graduation requirements at the University of Evansville come in three parts: general education requirements, major requirements, and electives. General education requirements are classes that the university requires every student to take. Most of these classes must be outside the student's major. This program consists of eleven Outcomes that the student must meet by taking courses that have been approved to meet that Outcome. Each Outcome requires a certain number of credit hours before it is considered to be completed. For example, Outcome 2 states that students show "Engagement with imaginative expressions of the human identity" and requires that students take three hours. The Registrar's office maintains a list of classes that fulfill each Outcome. Additionally, each student must complete four Overlays. Overlays contain broader themes than Outcomes that span across the curriculum, such as Global Perspective: International Diversity. Each Overlay has its own set of acceptable courses to complete it and requires a different number of hours, similar to the Outcomes. Many Overlay courses also fulfill Outcome courses, though this is not always the case. Details of the general education program are given in Appendix A.

Each major maintains its own set of requirements that a student must complete to earn a degree. These courses vary in content and quantity from major to major. Each department at the university decides what the requirements will be for the majors that it covers. Within each major there generally is a set of core required classes that must be completed by every student, as well as major-related electives, which gives each student the choice to tailor the major to their interests. An example of the Computer Science degree requirements are given in Appendix B.

The last part of the degree requirements is free electives. Different majors have different numbers of free electives that must be taken. Free electives are defined as classes that fall outside of general education classes and major-related classes. Some programs restrict what courses can be counted as free electives and some require students to take some portion of their free electives at the 300/400 level.

There are a few problems this project needs to solve. Currently, each student or advisor is in charge of keeping track of completed classes by hand. Each student or advisor maintains a checklist containing all the classes the student must complete in order to earn their desired degree. They fill out this sheet as the years progress, making sure that they earn all the credits they need. Because this was tricky, the advisors suggested a digital planning tool, which would allow students to map out what classes they want to take over their four years. This also could handle the problem of handling different sets of requirements for each student. Currently, if a student is pursuing two majors, or even a minor, they must keep track of multiple sheets, cross-checking those papers to make sure they have completed all requirements.

Requirements:

Through conversation with several students, faculty, and staff about what is needed in this system, a list of requirements was created. Broadly, the application needs to be accessed easily from a variety of platforms, to allow for the greatest number of users. This accessibility allows for the greatest number of users. Additionally, a number of different kinds of users need to be able to use the application to complete different kinds of tasks. The application allows for three different kinds of users, each with different abilities.

Students are the first group of users to be considered. They should be able to view each year of their education and the classes they have, are, or are planning to take in a readable format. The application also should allow students to plan their future semesters by moving classes around to explore different possible plans. Additionally, they should be able to declare one or more majors and zero or more minors. These majors and minors will determine their compiled list of classes, which they should be able to easily view. When planning the upcoming semester, students should be able to view a list of all available classes. Those classes in the upcoming semester that meet requirements the student has yet to fulfill will be noted.

Faculty members are the second set of core users. Faculty users have different needs than students, so they should be able to perform different tasks. The main ability of the faculty will be to view information about their advisees. They should be able to view the classes each of their advisees has taken and is planning to take so that they can make sure their students are taking the classes they need to in order to graduate on time. Next, faculty advisors should have the ability to set the catalog of their advisees. Initially, students will be assigned the default catalog of their freshman year, but occasionally, changes have to be made based on the changing circumstances of some students. Along those same lines, advisors should be able to grant a variety of exceptions to their students, including allowing them to bypass certain prerequisites or major classes. This allows the system to meet the needs of all students, even the ones who have special circumstances.

The final group of users is the administrator group. These users are the ones who control the system itself. They should have the ability to add majors and minors to the system, as well as modifying already existing majors and minors. Administrators also should have the ability to add new requirements to the system, including general education requirements and other university-ordained requirements. Finally, administrators should have the ability to modify some aspects of currently existing accounts, as may be required from time to time.

For the backend, the application must interface with and add to the University of Evansville's already existing Datatel server. The application should draw much of its necessary information from the university's Datatel server, including which classes a student has already completed and what classes are listed in the current catalog. The backend of the system will extend this, allowing departments to specify that a student must complete a certain number of hours from a given set of classes to fulfill a prerequisite. This important feature is needed for Civil Engineering and Mechanical Engineering majors, who have requirements that they have to complete before they will be allowed to take upper-level courses.

Design:

Technologies Used

This application is built using Ruby on Rails as the backend. Ruby on Rails is a web development platform. A web platform was selected because of the requirement that the application be easily accessible to as many people as possible. A web application can be accessed on any computer and on other web-enabled devices, such as smart phones. Ruby on Rails specifically was chosen because it is a free, fully matured platform. It allows the programmer to easily create and manage databases by allowing the user to define objects and builds the database off of those objects. This allows for a very object oriented approach to be used. Additionally, Ruby on Rails supports RubyGems, which are add-ons to the language that allow the user to quickly and easily add new features to their site, such as a simple, secure, login page.

Using Ruby on Rails makes it easier to create all the databases that will need to hold information such as degree checklists, prerequisites of classes, and student transcripts as Ruby on Rails does most of the work itself. Ruby on Rails creates databases from the objects that are created, so once all the objects for the application are made in Ruby on Rails, Ruby on Rails will do the rest in creating all the databases needed for the application by itself. Ruby on Rails also makes the needed connections between databases through the information that each object holds. For example, if a faculty object has his ID number and his advisee's ID number and the student object has an advisor ID number and her own ID number, then Ruby on Rails will make a connection between the teacher and student databases using these ID numbers. These databases are accessed much the same way as an object. Ruby on Rails allows you to access tables the same way as objects, with just their name variable instead of with a Query statement to the table.

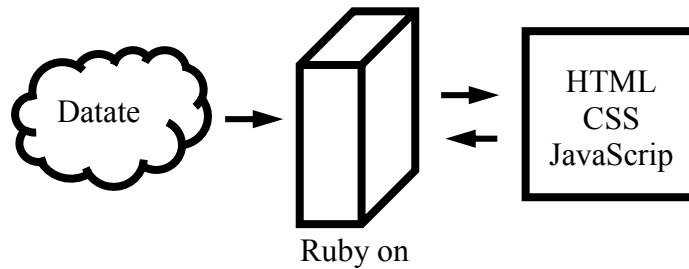


Figure 1: Interaction Diagram

The frontend design was completed using a combination of modern web technologies. The structural design is done using HTML5, the new modern standard version of HTML. The styling of pages and content will be done using CSS. User interaction is handled by Javascript, which runs user-side, instead of server-side. These technologies, as a set, were chosen because they represent modern web standards for frontend development and design. Figure 1 summarizes how the frontend and backend technologies will work together.

The code base of this system is managed by GIT, the open source version control system. GIT was chosen because of the developers' familiarity with it. Additionally, the use of GIT allows the developers to use Capistrano, an application that manages the live server of the deployed version and allows for simple updating with a single command that can be issued remotely.

Backend Design:

The application has several scripts for populating the tables within the application. The University of Evansville uses a Datatel server to securely hold all student information and, as students, the developers were not granted direct access to the server because of the sensitive data it contains. Instead, the developers worked with faculty and staff to pull the information from the server. The data was then sanitized by randomizing all of the names and hashing all of the student id numbers so that no identifying information could be connected to its original owner, protecting the privacy of each student's school records. Once this was completed, all of the information was put into a comma separated values file and given to the developers to use. A number of scripts were created to parse the given file. The first set of scripts is used to populate the users, advisors, and majors tables. These scripts use the data that we received in the csv file by parsing and separating the data into three separate csv files, one for each table. Then the new csv files are imported into their respective tables so that the application has data up to the day that the data in the csv file was received. The other two scripts are for populating the prerequisites table and the available classes table, which both have their own Excel spreadsheets, where the administrator can add and remove classes and prerequisites. They each have their own Excel spreadsheet that is updated because none of the information they store is currently being stored on Datatel and needed to be kept track of by the application. Two different scripts are used to update the tables with the new information in the Excel spreadsheets.

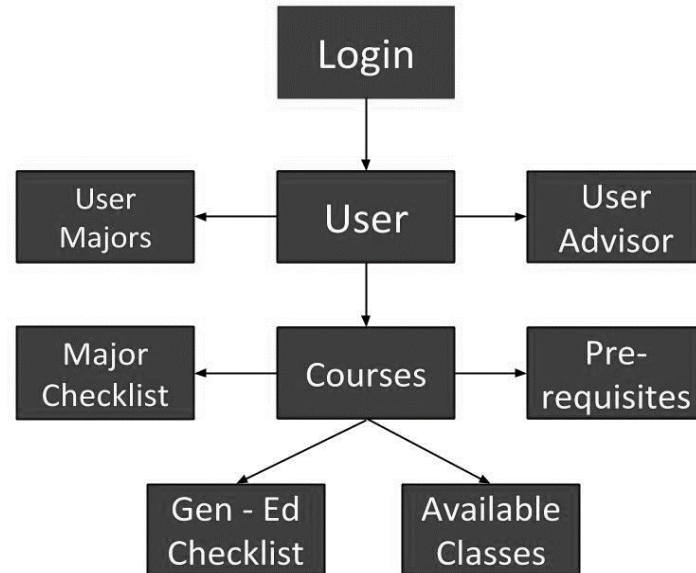


Figure 2: Class Diagram

Within this application, the Ruby on Rails objects became tables within the database. Figure 2 shows the basic layout of the objects and how they connect. The first object is the login object which holds students' username, password, what type of account it is meaning student, faculty, or administrator, and what that user's id number. The students use their student id numbers that were given to them by the University. There also is a user object to hold each student's basic information such as their name, grade point average, and catalog; it also holds the necessary variables to connect the student table to almost all of the other objects within this application, as seen in Figure 2. An advisor object also holds each student's advisor names and a majors table holds each student's major and minor.

The last group of objects is the five objects called courses, major checklists, general education checklists, prerequisites, and available classes that just hold information for the application to use. The major checklists object and the gen-ed checklists object each hold what classes need to be completed in order to get a degree in a particular major or to have completed all of the gen-ed requirements. The courses object holds what courses each student has completed, the credits, and the grade for that class and it also holds which classes the student plan on taking in the future. The courses object is connected to a tags object which can be used by the other objects to see if a certain requirement has been fulfilled. A course can have multiple tags since classes can fulfill many different requirements such as CS415 having a CS415 tag but also having a CS upper level elective tag since it could be used to fulfill either requirement depending on which major the student is in. These tags within the courses can be used by the other objects when determining if the student has fulfilled all of the requirements within the gen-eds object, major checklists object, or prerequisite object. Lastly, the prerequisites object holds what the prerequisites for each class are before a student is allowed to take that class and the available classes object will hold all the classes available at the University and also which semester they will be available.

Frontend Design:

Broadly speaking, the visual look of the site is the same for all users, but it will be populated with different content based on the user's personal data. Each task is contained in a page. Students have pages for viewing and editing their current proposed schedule, viewing their major and minor checklists, searching for classes, and changing personal options, such as major. Faculty users have pages for searching for and viewing an advisee's schedule, viewing rosters for current classes being taught, and adding exceptions for a given student. Administrators have pages for adding new majors and minors, modifying existing data, and other similar pages.

The exact layout and function of the pages was determined through a scenario-based user-interface design process. Initially, the developer created some hypothetical stakeholders for the project. These stakeholders are fictional future users of the system. From this, the developer wrote design scenarios using the stakeholders, which describe a single action the user will complete using the system. For example, one of the hypothetical stakeholders is Mindy Right. Mindy is a sophomore computer science major. She is trying to figure out if she has time to take a minor and still graduate on time. She needs help laying out her classes to see if she has enough time. In one design scenario, Mindy wants to add a class to her proposed schedule for her junior year. She should be able to select a class from a drop-down and add it to her schedule in the right block.

These design scenarios influenced the design of several prototype pages. These prototypes cover important features, as well as the overall design on the site. The prototypes do not hook into the backend, but instead contain static data for testing purposes. For example, Figure 3 contains some mockups of layouts for the overall site design that were tested. Figure 4 shows the completed prototypes that were developed from the mockups in Figure 3. As can be seen, each mockup informed the prototype with Mockup A corresponding to Interface 1 and Mockup B to Interface 2. The two prototypes are fully completed sites, though they share some common features. Each prototype has a different overall design, as well as different functionality on some of the pages.

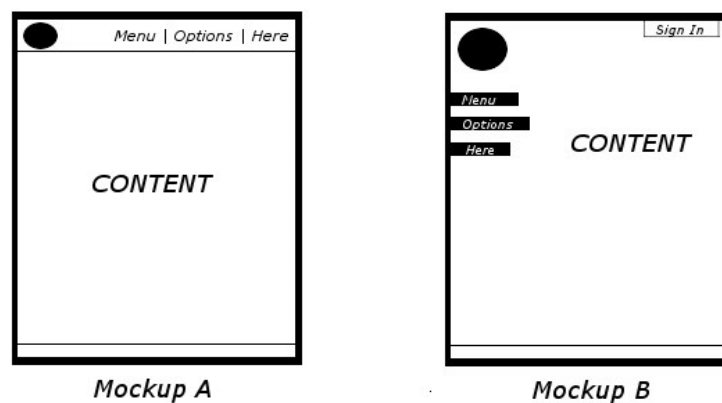


Figure 3: Mock layouts

Advising Planning Program

Home | Checklist | Schedule | Search Classes | Map

Schedule

Freshman	
FALL	SPRING
CS 101	CHEM 118
CS 210	PHYS 210
PHYS 112	ANTH 207
Add Row	Add Row

Sophomore	
FALL	SPRING
ANTHRO 215	MATH 221
Add Row	Add Row

Junior	
FALL	SPRING
Add Row	Add Row

Senior	
FALL	SPRING
Add Row	CS 375
Add Row	Add Row

Interface 1

Advising Planning Program

Home | Checklist | Schedule | Search Classes | Map

Schedule

Freshman	
FALL	SPRING
CS 101	CHEM 118
CS 210	PHYS 210
PHYS 112	ANTH 207
Add Row	Add Row

Sophomore	
FALL	SPRING
ANTHRO 215	MATH 221
Add Row	Add Row

Junior	
FALL	SPRING
Add Row	Add Row

Senior	
FALL	SPRING
Add Row	CS 375
Add Row	Add Row

Major Classes:
[Dropdown]

Minor Classes:
[Dropdown]

General Education Classes:
[Dropdown]

All Classes:
[Dropdown]

Interface 2

Figure 4: Schedule page from each completed prototype

Once the prototypes were completed, the developer completed a usability test on the system. The developer chose to use an empirical test, which observes real potential users of the system instead of an analytical test, which simply simulates the actions of a future user. This was chosen because, although it is more complicated to complete, it gives a better understanding of the usability of the system [1]. Specifically, the developer chose to use a laboratory usability test because it can be completed on a prototype, whereas a long-term field study, where the developer would spend several days watching the normal work flow of users, requires a completed program. The test used within-subject design, which means that each participants will be exposed to all of the independent variables, instead of each user being exposed to only a single independent variable. This is to say, all subjects completed tests on both prototypes and following the same procedure, as opposed to each subject only looking at a single prototype [1]. During the test, both numerical data, in terms of number of errors and time taken to complete tasks, as well as subjective user thoughts was gathered. These kinds of studies are common in the field of Human-Computer Interaction.

Future student users of the system were recruited for this study. The participants came from all majors and grades within the College of Engineering and Computer Science. Testers entered the Computer Science Project Lab one at a time. After reading a consent waiver, subjects were asked to sit at one of the computers that had been prepared by the developer for the test. Subjects were randomly assigned to begin with either Interface 1 or Interface 2. To them, the first one they complete was known as Interface A, and the second, Interface B. This random assignment is to prevent familiarity with the questions and the system as being a deciding factor in which interface appears the most usable. The test subjects will be asked to complete a certain set of tasks, as seen in Appendix C, using each of the prototypes while the developer observes their actions. The developer noted what tasks they seem to complete easily, and which ones the subjects struggle with, specifically noting the number of errors that occur that occur while completing each task and noting the total amount of time it took to complete each task. After completing the tasks using Interface A, the subjects were asked to switch to Interface B and complete the same tasks. When they finished, the subjects were given a survey where they voiced any overall thoughts they had about each prototype. The survey also can be found in Appendix C. The developer then analyzed the results of the testing by comparing collecting and analyzing the results of the survey, as well as noting which parts the participants struggled with in order to determine what could be made clearer. The developer then decided on the final design based on all the given input. The most usable features of each prototype will be combined and turned into a single, working application.

Results:

The database that was developed stayed pretty close to the original design. The database contains all of the objects as was explained above with Figure 2 and can perform the queries that the application needs. The queries can be grouped into three main categories: student queries, faculty queries, and application queries. There are three main student queries. The first is getting all of the student's information. This information would be needed for the user's profile page and includes the student's name, grade point average, what catalog they are using for their checklists, and also which year the student is in. The second query is getting a list of all of the student's advisors. The last two queries for this group are a query to get which classes each student has completed, which includes the grade for the class, how many credits, and which term the class was completed, and a query to get which classes the student plans on taking while at the University of Evansville.

The second group of queries is the faculty queries. These queries are used by faculty users. The first query is to get a list of all of the professor's advisees. The second query gets the information for one of those students, in particular the course the student has completed and also plans on taking.

The last group of queries is the program queries. The program queries are used by the program to provide pertinent information to the user. The first query is to access the major checklists. The query would get all of the classes required for the user's major and then compare this to the courses that the user has completed from the courses table using the tag table. If any class has not been completed then the user would be told which requirements they still need to take. The next query is to access the general education checklists and would do pretty much the same as the last query but instead it checks to see if the student has completed all of the general

education. The last two queries would work together. For each class, they would get the list of prerequisites and then check if the student has completed each of the prerequisites and is allowed to take the class. Any class that they did not have all the prerequisites for would be taken off the list and then the final list with only classes the student is able to take would then be shown to the student as available classes for the student to take.

The final participants of the usability study were 10 students, across all 5 majors within the College of Engineering and Computer Science and across all 4 standard years. The usability tests produced some definitive results, which can be seen in Table 1 and Table 2 below.

	Clear Intent	Checklist	Schedule	Search Classes	About Me	Overall Experience
Interface 1	5.50	5.70	5.80	4.60	6.50	5.70
Interface 2	4.90	5.10	4.10	4.60	6.00	4.50

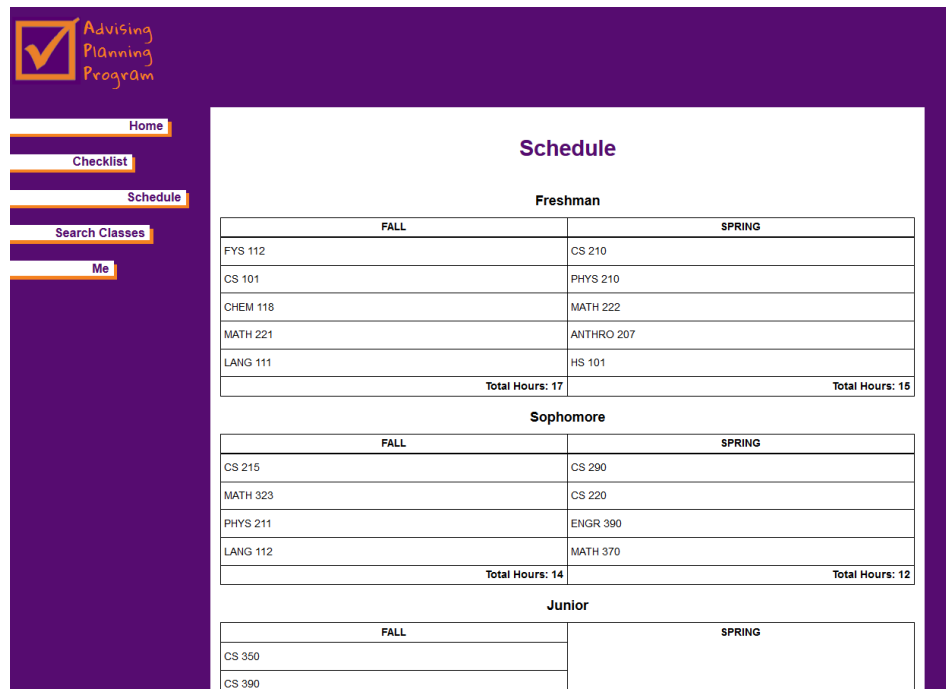
Table 1: Ranking of individual features (scale 1-7, n=10)

	Interface 1	Interface 2	Tie
Easy	7	3	0
Checklist	6	4	0
Search	8	2	0
Me	4	3	3
Design	4	6	0

Table 2: User preferences for 1 or 2 (n=10)

As can be seen in the above tables, Interface 1 won for every feature in both types of evaluation. Interface 2, however, won for design. When asked the rate on a scale of 1-7, most features were within approximately 1 point of each other. The most notable difference was on the schedule page. That page featured, on Interface 1, an add class button at the bottom of each semester and, on Interface 2, a drag and drop feature for adding classes. Because students had problems figuring out how the drag and drop worked, they significantly preferred Interface 1. The smallest difference was on the Search Classes page, which ranked the same in both cases because it worked the same way. The developer also noted that participants tended to struggle with the same tasks. For example, the developer selected a set of categories that could be used to search for classes, but students found these categories to be unintuitive. Each participant struggled to decide which category they should use for the different types of searches.

The results were combined to produce the final interface, which can be seen in Figure 5.



Freshman	
FALL	SPRING
FYS 112	CS 210
CS 101	PHYS 210
CHEM 118	MATH 222
MATH 221	ANTHRO 207
LANG 111	HS 101
Total Hours: 17	Total Hours: 15

Sophomore	
FALL	SPRING
CS 215	CS 290
MATH 323	CS 220
PHYS 211	ENGR 390
LANG 112	MATH 370
Total Hours: 14	Total Hours: 12

Junior	
FALL	SPRING
CS 350	
CS 390	

Figure 5: Schedule page in final design

Conclusion:

There is still work to be done on the application. The developers were unable to link the database to the front end due to a lack of time. Additionally, the developers would like to expand the application to include all majors and minors that exist within the university. With the cooperation of the school, the developers would also like to be able to access information directly from Datatel instead of the current system, which requires a file to be download and imported once a semester. Finally, developers would like to develop a feature that allows faculty to check that everyone enrolled any of the classes they teach has met the necessary prerequisites. The developers hope that this application can be completed and put into use in the future.

References

- [1] Rosson, Mary Beth., and John M. Carroll. Usability Engineering: Scenario-based Development of Human-computer Interaction. San Francisco: Academic, 2002. Print.
- [2] University of Evansville. (2013). *2013-2015 Undergraduate and Graduate Catalog*. Retrieved from <http://www.evansville.edu/registrar/downloads/CourseCatalog2013-2015.pdf>

Biography

Allison Deford is a senior Computer Science major at the University of Evansville, with minors in Mathematics and Anthropology. She is originally from Lafayette, IN. While not hanging out and working in the Computer Science lab, Allison spends much of her time working with UE PRIDE and working to promote diversity within the university and beyond. While she enjoyed her time at the University of Evansville immensely, she has big plans for the future. Upon graduation, Allison will use the knowledge she gains from her senior project to attend graduate school at the University of Washington in Seattle to obtain a master's degree in the field of Human-Computer Interaction.

Tyler Hoffman is a senior Computer Science and Mathematics double major at the University of Evansville. He is from Cincinnati, OH and is a member of Sigma Phi Epsilon Fraternity. While he was a member Sigma Phi Epsilon at the University of Evansville he held several executive positions which include Vice President of Communication for his chapter as well as Vice President of Recruitment for the Inter Fraternal Council. While in these position he worked to promote brotherhood not only within his own fraternity but also worked to build a sense of brotherhood between each of the fraternities on campus. After graduation, Tyler plans on moving back home to Cincinnati and searching for a job there.

Appendix A:

University of Evansville General Education Requirements 2013-2015 [2]

Outcomes

Outcome 1: Critical reading and thinking (FYS)- 3 hours

Outcome 2: Engagement with imaginative expressions of the human condition- 3 hours

Outcome 3: Knowledge of human history and the historical context of knowledge- 3 hours

Outcome 4: Engagement with fundamental beliefs about human identity, core values, and humankind's place in the world- 3 hours

Outcome 5: Understanding of human aesthetic creation and artistic creativity- 3 hours

Outcome 6: Linguistic and cultural competence in a language other than one's own- 6 hours

Outcome 7: Quantitative literacy- 3 hours

Outcome 8: Scientific literacy- 7 hours

Outcome 9: Understanding of core concepts of society, human behavior, and civic knowledge- 6 hours

Outcome 10: Knowledge and responsibility in relation to health and wellness- 1 hour

Outcome 11: (Capstone) Ability to think critically and communicate effectively, orally and in writing- 3 hours

Overlays

Global Perspective: International Diversity (two course-equivalents)

Global Perspective: Domestic Diversity (one course-equivalent)

Social Responsibility (one course-equivalent)

Writing-Intensive (four course-equivalents)

Appendix B: Major Requirements

Below is a sample of the requirements for computer science.

Enduring Foundations General Education Requirements			
(43 hrs)	Hrs	Grade	Overlay
Outcome 1: Critical Reading and Thinking (3 hrs)			
FYS 112: First Year Seminar	3		E
Outcome 2: Engagement with Imaginative Expressions of the Human Condition (3 hrs)			
	3		
Outcome 3: Knowledge of Human History and the Historical Context of Knowledge (3 hrs)			
	3		
Outcome 4: Engagement with Fundamental Beliefs about Human Identity, Core Values, and Humankind's Place in the World (3 hrs)			
	3		
Outcome 5: Understanding of Human Aesthetic Creation and Artistic Creativity (3 hrs)			
	3		
Outcome 6: Linguistic and Cultural Competence in a Language Other than One's Own (6 hrs)			
	3		
	3		
Outcome 7: Quantitative Literacy (3 hrs)			
Math 221 – Calculus I	4		
Outcome 8: Scientific Literacy (8 hrs)			
(Lab) BIOL 107 or CHEM 118	4		
PHYS 210 – Calculus Physics I	4		
Outcome 9: Understanding of Core Concepts of Society, Human Behavior, and Civic Knowledge (6 hrs)			
	3		
	3		
Outcome 10: Knowledge and Responsibility in Relation to Health & Wellness (1 hr)			
	1		
Outcome 11: Ability to Think Critically and Communicate Effectively, Orally and in Writing (3 hrs)			
CS 495 - Senior Project Phase I	3		E
TOTAL CREDITS			
TECHNICAL ELECTIVES (12 hrs minimum)			
<i>Students must choose from CS 350, 355, 375, 376, 415, 430, 440, 475, 478, 480, 499; EE 310, 311, 354, 454, 456.</i>			
	3		
	3		
	3		
	3		
TOTAL CREDITS			

BASIC LEVEL REQUIREMENTS (33 hrs)	Hrs	Grade	Overlay
CS 101 – Intro to Computer Science	3		
CS 210 – Fund'ls of Programming I	3		
CS 215 – Fund'ls of Programming II	3		
CS 220 – Logic Design & Machine Org	3		
CS 290 – Object-Oriented Design	3		
ENGR 390 or MATH 365 or MATH 341	3		
MATH 222 – Calculus II	4		
MATH 323 – Calculus III	4		
MATH 370 – Discrete & Combinatorial Mathematics	3		
<i>Computer Science majors must complete a two-semester sequence in biology, chemistry, or physics.</i>			
BIOL 109; CHEM 240, 280; or PHYS 211	4		
TOTAL CREDITS			

UPPER LEVEL REQUIREMENTS (21 hrs)	Hrs	Grade	Overlay
CS 315 – Algorithms & Data Structures	3		
CS 320 – Computer Architecture	3		
CS 380 – Programming Languages	3		
CS 381 – Formal Languages	3		
CS 390 – Software Engineering	3		
CS 470 – Operating Systems	3		
CS 494 – Senior Project Seminar	0		
CS 497 – Senior Project Phase II	3		E
TOTAL CREDITS			

PROFESSIONAL DEVELOPMENT ELECTIVE - Computer Science			
<i>majors must choose one course from ECON 101; COMM 210, 382, 485; PHIL 111, 121, 231, 241, 316, 317; WRTG 330 when the topic is technical writing. This course may not be used to fulfill a general education course. (3 hrs)</i>			
	3		
TOTAL CREDITS			
FREE ELECTIVES - At least 9 hours must be 300 level or higher. It is recommended that Computer Science majors use their free electives to minor in a field of application. Courses numbered MATH 222 or lower, CHEM 10x, CS 210 or lower, PHYS 100 level, SA courses, IT 120, and English language courses may not be used as free electives. (18 hrs)			
300/400 level			
300/400 level			
300/400 level			
TOTAL CREDITS			

Appendix C: Usability Study

[All blanks for writing answers have been omitted.]

Directions:

1. Bring up the web browser and make sure the web browser is focused on the first tab.
2. Log in using the following username and password:
3. Imagine you are a computer science major trying to plan your course load for Fall 2014, which will be the fall of your junior year. First, view the checklist to determine which of the following classes you still need to complete: CS290, CS315, CS350, CS390, CS380, CS381, MATH323, MATH324
4. Navigate to the schedule page and add the classes you have not yet completed to your Fall 2014 schedule.
5. After talking to some classmates, you decide that your schedule is too full for the upcoming semester. You decide to take CS315 in the spring of your junior year instead. Remove it from your current schedule and schedule it for the spring of your junior year.
6. Add a class from Outcome 7 to your schedule for the spring semester of your junior year.
7. You hear some of your friends talking about a class they call PSYC121. Find the title and description for this class number.
8. Upon viewing your schedule, you discover that you need to take 3 more technical electives. Search for classes that would fulfill the CS technical elective requirement. Write the course number for 3 of those courses below:
9. Let's say you decide that you want to add a psychology minor to your computer science degree. Using the Me page, add this minor to your requirements.
10. Can you determine how many classes you still need to take to complete this minor? If so, write the answer here:
11. View the Checklist page. How Outcomes must you still fulfill before graduation?
12. You have a question about how to officially add your new psychology minor. Find your advisor's email address and write it below so that you can send them an email and ask:
13. Repeat steps 3-12 in the second tab, which contains an alternate user interface. Fill in your answers to the questions using the blanks labeled Interface B.
14. When you have completed these steps, please turn the page and complete the post survey.

Please answer the following questions (separate form for each interface option). [All questions will use the same scale printed in question 1. Blank space for writing answers has been omitted.]

1. I found this interface easy to use.

<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>
<i>Strongly</i>	<i>Disagree</i>	<i>Moderately</i>	<i>Neutral</i>	<i>Moderately</i>	<i>Agree</i>	<i>Strongly</i>
<i>Disagree</i>		<i>Disagree</i>		<i>Agree</i>		<i>Agree</i>

2. It was clear where I needed to click to complete each task.

3. I thought the Checklist page was easy to use.

Additional Comments on the Checklist page:

4. I thought the Schedule page was easy to use.

Additional Comments on the Schedule page:

5. I thought the Search Classes page was easy to use.

Additional Comments on the Search Classes page:

6. I thought the About Me page was easy to use.

Additional Comments on the About Me page:

With regards to both interfaces,

- | | | |
|--|--------------------|--------------------|
| 1. Which interface was easier to use? | <i>Interface A</i> | <i>Interface B</i> |
| 2. Which Checklist page was easier to use? | <i>Interface A</i> | <i>Interface B</i> |
| 3. Which Schedule page was easier to use? | <i>Interface A</i> | <i>Interface B</i> |
| 4. Which Search Classes page was easier to use? | <i>Interface A</i> | <i>Interface B</i> |
| 5. Which About Me page was easier to use? | <i>Interface A</i> | <i>Interface B</i> |
| 6. Which overall site design was easier to use? | <i>Interface A</i> | <i>Interface B</i> |
| 7. I am likely to use an application such as this one when planning my future classes. | | |

<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>
<i>Strongly</i>	<i>Disagree</i>	<i>Moderately</i>	<i>Neutral</i>	<i>Moderately</i>	<i>Agree</i>	<i>Strongly</i>
<i>Disagree</i>		<i>Disagree</i>		<i>Agree</i>		<i>Agree</i>

Please include any overall comments you might have about either design: