Applying Contextual Design to Build a Course Scheduler: A Case Study

Jean D'Amico
Teresa Hübscher-Younger
Roland Hübscher
N. Hari Narayanan

Technical Report CSSE03-01

January 14, 2003

Intelligent & Interactive Systems Laboratory
Department of Computer Science & Software Engineering
Auburn University
Auburn, Alabama 36849-5347, USA

Applying Contextual Design to Build a Course Scheduler: A Case Study

Jean D'Amico, Teresa Hübscher-Younger, Roland Hübscher and N. Hari Narayanan
Intelligent & Interactive Systems Laboratory
Dept. of Computer Science & Software Engineering
Auburn University
Auburn, AL 3849-5347
{damico,teresa,roland,narayan}@eng.auburn.edu

Abstract This report describes the development of a course-scheduling tool that assists professors in planning course events such as tests, assignments and lectures. This system takes university calendar information (e.g. semester start and end dates, and various holidays and breaks) along with user inputs of the number of tests, assignments, and extra-credit assignments, and translates these into algebraic constraints, which are then solved by a linear programming algorithm to generate a feasible schedule. It was designed using the Contextual Design approach (Beyer & Holtzblatt, 1998). We report on the application of this design methodology to generate requirements and architecture of this tool. First, a contextual analysis was conducted to ensure that specifications were adequately developed. This included conducting and coding interviews with potential users, and gathering and analyzing work artifacts. Then a prototype was implemented and evaluated by means of a cognitive walkthrough. This report, by describing the design process and the resulting system, makes two contributions. First, it illustrates the contextual approach to interactive system design in an educational domain. Second, it presents the architecture of a course scheduling tool, with a constraint-solver based on the linear programming algorithm at its heart, that allows the automatic creation of feasible event schedules.

Keywords: contextual design, interactive systems, scheduling, constraint satisfaction, linear programming.

1. Introduction

Planning and scheduling the various events (such as quizzes, exams, assignments and extra-credit activities) involved in a semester-long college course can be a tedious and time-intensive task. This is especially true if the professor has several constraints that the schedule must meet. For example, the professor may need one week to grade assignments, wish to give one week to students for doing each assignment, and plan to have two assignments assigned, graded and returned at least one week prior to the first exam, which has to take place no later than the fifth week of classes. The more events and constraints there are, the more tedious manual scheduling becomes. Scheduling adds to the course preparation burden that already involves several other activities such as determining the course content, selecting a book, assembling lectures, and developing tests and assignments. There are also individual differences in how professors approach the scheduling task. Some may initially develop, and then strictly adhere to, a highly structured schedule, while others may develop a vague outline at the beginning of the semester and fill in the gaps throughout the semester.

This report describes a project on developing a course event scheduling tool to assist professors. Our goal was to develop a tool that professors can use to create and examine different feasible schedules that conform to given constraints, once the course events that the professor wants to schedule have all been specified. This meant that the tool had to be interactive, allowing the input of events and constraints, the computation and display of feasible schedules, and subsequent examination, selection and modification of the schedules. Given this fact, and that there exist major individual differences among professors in how they schedule course events, we chose to employ a design method known as *Contextual Design*. Contextual Design is a method for developing products that focuses on how the user performs tasks within the context of the work environment itself (Beyer & Holtzblatt, 1998). This approach was employed to (1) develop a detailed picture of the process by which professors plan a college course and schedule its activities, and the factors that influence this process, and (2) subsequently derive system requirements from this picture.

The resulting system, called the Constraint-Based Course Building Tool (CB^{2}T) uses a series of user-defined and internal system constraints in order to assist in course scheduling. These constraints consist of dates (specifically course start and end dates, break start and end dates, holidays, and mid-semester) as well as the number of events (tests, assignments, lectures, and extra-credit assignments) that will occur during the semester. It uses a linear programming algorithm to determine a class schedule that minimizes conflicts between events and allows for adequate spacing between events. The intents of this report are to describe the design process, to serve as a case study in the application of the Contextual Design technique, and to present the architecture of the system that provides a much needed functionality to professors.

2. Contextual Design

Contextual Design is a method for developing products that focuses on how the user performs tasks within the context of the work environment itself (Beyer & Holtzblatt, 1998). In our case, it meant looking at the process of course development as it existed within a university culture by gathering information from people who were potential users of the tool. It was crucial to fully understand the process and meaning of “creating a college course” from the viewpoint of these potential users. From a broad perspective, it was also important to understand the essential attributes of a course and the course planning process.

The first step in the information-gathering process was interviewing potential users of the CB^{2}T system. The purpose of these interviews was to determine how a professor developed a course plan, how a syllabus was created and used, what role calendars played in course development, and other time management and course planning issues. The interview process consisted of a short (less than 30 minutes) informal discussion about what types of planning go into creating a college course. Figure 1 shows sample interview questions. We conducted five interviews with faculty members known to have different teaching styles. We also collected and analyzed artifacts used in planning and managing courses that further demonstrated their procedure, primarily calendars, syllabi, websites, schedules, and other related materials. The information gathered was then coded and compiled into useable models. Coding is a process that is used in qualitative research to extract relevant data from observations (Denzin & Lincoln, 1994). Coding schemes allow the data to be categorized and managed into useful information themes. Figure 2 shows an extract from the 7-level coding hierarchy developed from the interviews.

How do you go about building a course from scratch?
Do you use the textbook to determine some of the requirements?
Is your schedule pretty much set in stone?
How do you know when to schedule tests and assignments?
Overall, how much planning would you say goes into creating a course?
How do you prepare a syllabus?
I see that you have your tests scheduled on the syllabus, are these flexible?
What goes into preparing for a course?
How do you prepare your schedule?
Do you construct a day-by-day schedule?
How do you schedule assignments?
How much time do you usually try to allot for grading a test or an assignment?
What tools do you use to construct a course? Pen and paper? Calendar?
How flexible is your schedule?
Do you keep a personal schedule as well?
What would you want in a course scheduling tool?

Figure 1. Sample of interview questions

Usually, the Contextual Design process involves building a series of five work models that are used to characterize the task that is being completed. The five work models are: the Flow Model, the Sequence Model, the Artifact Model, the Cultural Model, and the Physical Model (Beyer & Holtzblatt, 1998). Each of these has a direct impact on aspects of system design. A Flow Model is used to describe how information flows between people, and to define the roles that people use while completing a task. As CB^{2}T was intended to be a single-user tool, the Flow Model was deemed not necessary. The Physical Model is used to describe the physical environment in which work takes place. This is crucial when designing a piece of machinery or having a user complete a repetitive task. The environment of use of CB^{2}T would be an office setting or a home setting. So we did not develop a Physical Model either. The Cultural Model investigates the overall culture of the workplace. It looks at factors that impact the activity in question. The Sequence Model is used to show how a sequence of events unfolds over time. The Artifact Model looks at how actual objects are used in the work process. These three models were developed.

The Cultural Model (Figure 3) encoded how various people, institutions, and influences impacted the course planning process. Obviously, professors and students most heavily impact a college course. Beyond these influences there are other, less obvious, components that impact a course. For instance, there is the immediate influence of the department and the university on the class. They influence how the class is structured through factors such as the

- Type of process
    - Formal
    - Informal
- Planning steps done prior to the semester
    - Identify objectives
    - Define instructional methods used to achieve requirements
    - Determine calendar for course
    - Determine lecture schedule
    - Draft syllabus
    - Determine student activities
    - Create planning documents
- Evaluate course during semester
    - Schedule adjusted during semester
    - Maintain separate schedules
    - Maintain a single schedule
- Grading
- Revise course after semester

Figure 2. Coding hierarchy extract

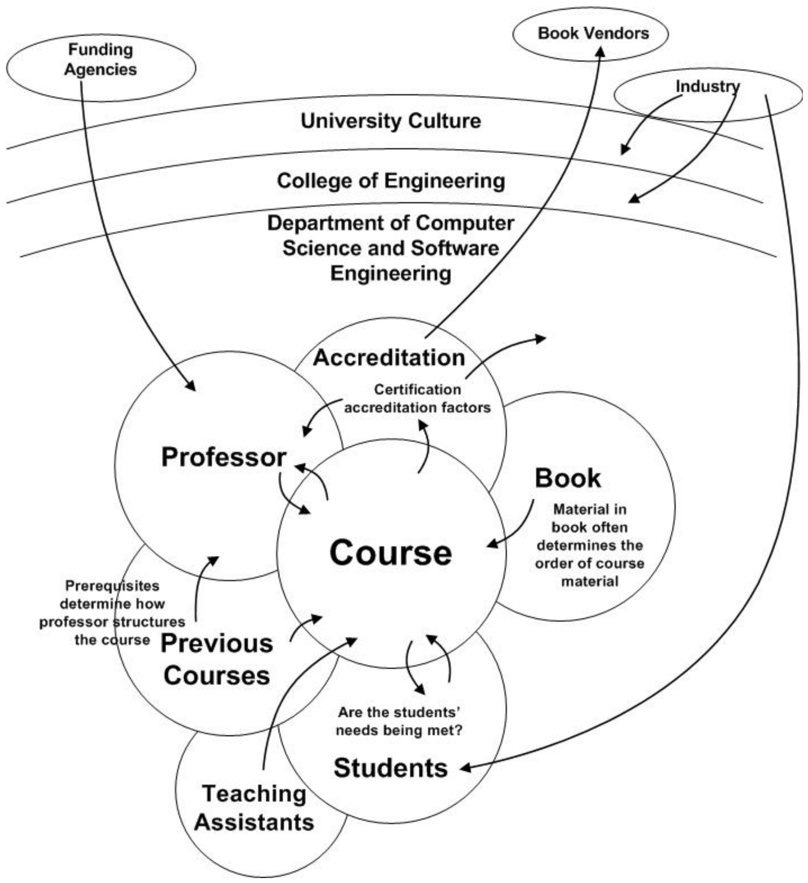

Funding Agencies
Book Vendors
Industry
University Culture
College of Engineering
Department of Computer Science and Software Engineering
Accreditation
Certification accreditation factors
Professor
Book
Material in book often determines the order of course material
Course
Prerequisites determine how professor structures the course
Previous Courses
Are the students' needs being met?
Students
Teaching Assistants

Figure 3. The cultural model

schedule for the school year, vacations, and the criteria by which the course will be evaluated. Other factors include accreditation boards, who determine what information should be covered in a course; the book vendors, who often impact which book is chosen which in turn determines the sequence of the material to be covered; teaching assistants, who influence how quickly grades are returned to students and the course load the professor is able to assign; and industry trends, which impact, over the long-term, the types of courses that are taught in universities.

The Sequence Model is used to show how a sequence of events unfolds over time. From the interviews, we could determine a generalized method that professors were using to develop courses. This series of events was broken down into three main parts:
1. Developing initial course material.
2. Developing a syllabus and class schedule.
3. Teaching the class and managing/modifying course content.

Developing initial course material involved such things as researching and choosing the appropriate book, readings and/or materials relevant for teaching the class. From this information users would typically develop a set of requirements (either formally or informally) in order to determine and schedule a combination of activities that are most appropriate for teaching the material, such as lectures, quizzes, in-class exercises, tests, and assignments.

Different professors used various approaches to determine the dates for scheduling various activities. The Sequence Model outlined three different approaches that could be taken regarding event scheduling:
Method 1: Tests occur after a given period of time and are static.
Method 2: Tests occur after a given period of time but can be moved within reason.
Method 3: Tests occur after a given amount of material has been covered.

The final step of the Sequence Model outlined how professors would perform routine maintenance on a course. This included making daily changes to the schedule and format of the class for teachings of the same course the next time.

The Artifact Model was created by analyzing artifacts collected from the faculty members who were interviewed, coding this information, and creating a single document that explicated the purpose, characteristics and components of the three types of artifacts that we discovered were being used by the interviewed faculty: planning documents, syllabi and calendars. These documents brought to light a great deal of information that was often not available directly in the interviews, in addition to illustrating how professors tangibly laid out their courses. For instance, one planning document was a flow chart of a step-by-step iterative process that thoroughly described how a course schedule was constructed by a particular professor. The resulting model thus provided additional insights into how a typical user carried out the task under analysis. An extract from this Artifact Model, detailing with the purpose of calendars and schedules, appears below for illustrative purposes.

> *Purpose*: Calendars and schedules are used by both professors and students in order to show how the course will be structured during the semester. The purpose of a calendar is to show a specific day-by-day agenda of the events occurring in a course. A schedule, while similar to a calendar, gives an all-purpose course outline and a more general direction for the course. A calendar displays detailed accounts of when events will occur accompanied by specific dates and times. A schedule, on the other hand, is meant to present the overall outline of the course. This may be high-level, such as topics to be covered over an entire course. It may also be more specific, including broad weekly schedules.

Planning documents revealed interesting individual differences. Some professors, when building a course, use a highly structured method in which every class period is planned out at the beginning of the semester and leaves very little margin for change within the schedule. Such professors develop what we call a static calendar. It is developed at the beginning of the semester and defines all lectures, assignments, tests, and extra-credit assignments. It is often included as a part of the syllabus. Other professors take a more flexible approach in which the topics are described generally within the syllabus, which provides a general outline for the course. They create what we call a dynamic calendar, which is actually a schedule of events that is created with the deliberate intent of being altered as necessary during the semester. Such a calendar might schedule specific events at the beginning of the semester, but with the implicit understanding (by professors and students) that the events may be moved as the course progresses. A particular characteristic of schedules is that professors will often update them as the course progresses in order to make a more accurate schedule for the following semester.

3. Requirements Specification

The steps of user interviews, artifact collection, information analysis and coding, and model development yielded several system requirements.

Requirement 1:The system should include daily, weekly, monthly and semesterly calendar views of the course plan, with a provision for entering and flexibly displaying (i.e. with more or less details) event information such as title, time, location and description.
One interesting aspect revealed by the artifacts was that types of calendar views the professors created took many different forms. Some professors would use a weekly or monthly calendar view with only the event name and title (i.e. lecture 12 or exam 1). Other professors would use a daily view that would show the details of every class event. Still other professors would just have a course outline that showed the sequence of topics to be covered with no details, defined events, or dates.

Requirement 2: The system should support the creation and maintenance of private (can be seen only by the user) and public (can be made publicly viewable, say, over the web) calendars, with facilities for easy migration of events from the private to the public calendar.
The Artifact Model indicated that a calendar or schedule could either be a privately held document by the professor alone or a public document that is shared between the students and professor. In many cases, professors will keep two separate documents: a public one that is openly available to the class and a private one that is used for planning and revising before adding items to the public one. The private calendar may contain course information that is intended for the personal use of the professor and not for consumption by the class at large. Some events first appear

in the private calendar and later appear in the class calendar. Further analysis revealed that this is an indication of background planning and revision that takes place before information is posted publicly.

Requirement 3: The system should allow the user to hide and show course activities by type and by detail.
The interviews and the Artifact Model indicated that users would like to view their daily, weekly or monthly schedules in different ways based on type and desired level of detail. For instance, sometimes they may want to see all tests that are scheduled for the current day, week or month. They may also just want to see the titles of all lectures scheduled for a week or see the detailed descriptions of all lectures scheduled for the day.

Requirement 4: A course calendar should be easily changeable at any time.
The Sequence and Artifact Models indicated that the content and timing of course events are often changed as the semester progresses, sometimes affecting the current offering of the course and sometimes with an eye toward the next offering of the course. These maintenance activities indicated the need for highly customizable course content while the course is being taught.

Requirement 5: Allow course plans to be saved as reusable templates.
From the Cultural Model we identified the impact of previous offerings of a course on its current content and schedule. This indicated the desirability of a template feature that would allow course information to be saved as a template for future modification and/or reuse.

Requirement 6: The system should allow the user to create and maintain multiple calendars for the same semester, and calendars of the same course over several semesters.
The Sequence Model indicated that professors created and maintained separate plans for all the courses they were teaching, and also revised and maintained course plans over several semesters.

Requirement 7: The system should separately maintain data for each user and each course.
Since it is common for the same course to be taught by different professors at the same or different semesters, and for the same professor to teach separate courses in any given semester, the data need to be kept separate and access needs to be controlled through a password-protected user login facility.

Requirement 8: The system should maintain an easily searchable and printable database of all course information.
The Cultural Model also highlighted the importance of accreditation concerns to course planning. Typically, an accreditation board will visit a university and ask to see syllabi and other course related material from the courses that have been taught since the last certification. So a system that captures and archives the complete schedule, syllabus and content (i.e. topics covered in each lecture) information from multiple offerings of a course in a uniform format can be invaluable in preparing reports for accreditation boards.

Requirement 9: The system should not enforce a single approach to planning and scheduling; instead, it should be as flexible as possible.
The Sequence Model revealed an inherent difficulty in developing a tool like CB^{2}T: there are large individual differences and no single approach fits everyone. The Artifact Model suggested that a course schedule must be flexible and dynamic, to be used in different ways depending on the nature of the user. For example, there are detailed planners who construct a complete and detailed schedule at the beginning of a semester and adheres to it as much as possible through the semester, and there are flexible planners who construct just an outline at the beginning and fill in details and dates as the semester progresses. In order for a system like CB^{2}T to fit the variety of approaches that people could potentially use, this flexibility requirement needs to be further refined as:
9.1. The system should allow a user to input as many, or as few, course events and constrains as he/she wishes.
9.2. It should allow the user to choose between automatic scheduling of the events entered or manually constructing a schedule.
9.3. For automatic scheduling, instead of generating and showing the "optimal" schedule or a single feasible schedule, the system should support the user in developing a schedule that best fits his or her approach to course planning.
9.4. This means that the system should support an iterative approach to scheduling:

> *9.4.1. Calculate and display all or several feasible schedules.*
> *9.4.2. Then allow the user to choose (and modify if desired) one of these schedules.*
> *9.4.3. It should also allow the user to reject all offered schedules and instead change some of the events and constraints for the next iteration.*

Requirement 10: The system should incorporate, and allow the user to enter and modify, a variety of constraints or relationships among various course events. The Artifact Model provided information regarding several types of constraints that are applied during course planning. First, there are hard constraints. For example, no formal course event can be scheduled on a Sunday or a university holiday. Similarly, universities generally require final exams to be held on specific dates and times. Second, there are soft constraints. For example, a professor may want the initial schedule to meet constraint that the default length of time to grade an assignment is a week (i.e. the events of an assignment being turned in and its grades being announced are separated by 7 days). Another example is that an assignment due date cannot be scheduled within a week of a test. These are soft constraints because they are user-defined and therefore can be modified or relaxed by the user. Given this distinction between hard and soft constraints, this requirement can be further refined as:
10.1. The system should automatically acquire hard constraints such as the start and end dates of the semester, scheduled holidays, final exam schedule etc.
10.2. The system should allow the user to enter additional soft constraints and modify or relax them as needed during iterative scheduling.

Requirement 11: The system should support manual and automatic emailing of students with course information and reminders about upcoming deadlines.
The interviews revealed that this is a highly desired functionality that paper-based calendars cannot provide.

4. System Architecture

Based on the requirements generated from the Contextual Design process, we developed an architecture for CB^{2}T. The system consists of three separate levels, with each performing a separate task. The highest level (level 3) is a PHP/HTML interface level that presents information to, and collects information from, the user. This includes such features as user login and course calendars. The middle level (level 2) is a translation level that consists of a Java program, which converts course information into a simplified format for the constraint solver, which resides at the lowest level (level 1). The middle level also interprets the output of the constraint solver into a form that can be used by the interface level. The constraint solver at is a linear programming tool known as lp_solve. It is the heart of the course scheduling algorithm. All of the data that is used throughout the levels are stored and retrieved by a MySQL database. Figure 4 illustrates this architecture.

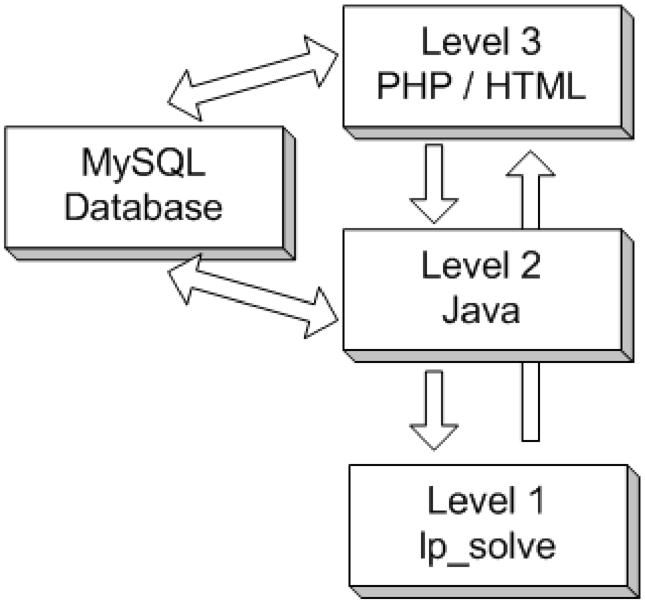
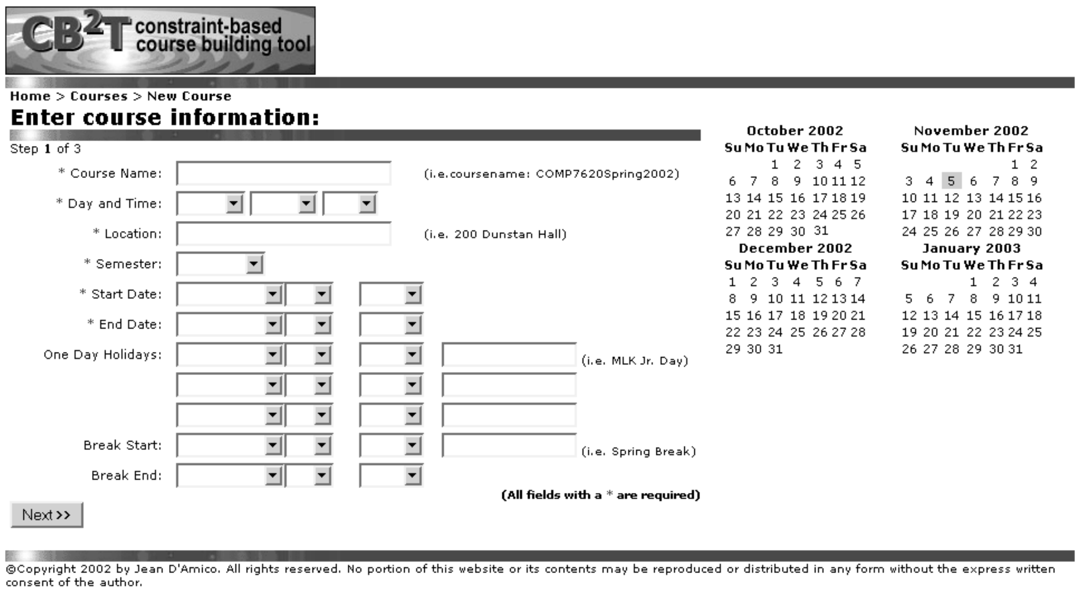


Figure 4. System architecture

4.1 The Interface Level

The user interface provides five main functionalities: user login, creating a course, entering constraints, viewing a course schedule by day, week or month, and creating, editing and deleting individual course events.

In order to create a new course, the user can enter the following information: course title, instructor name, instructor email address, instructor telephone number, instructor office location, instructor office hours, GTA name, GTA email address, GTA office location, GTA office hours, course description, location (classroom and building), and lecture time (Figure 5)., They can also input (implicit) constraints: semester begin date, semester end date, holidays, and any breaks (e.g., spring break), class rotation (Monday/Wednesday/Friday or Tuesday/Thursday), the number of assignments, the number of extra-credit assignments, and the number of tests. From this, the system computes information such as the actual number of lecture days in the semester.



Home > Courses > New Course

Enter course information:

Step **1** of 3

* Course Name: (i.e.coursename: COMP7620Spring2002)
* Day and Time:
* Location: (i.e. 200 Dunstan Hall)
* Semester:
* Start Date:
* End Date:

One Day Holidays: (i.e. MLK Jr. Day)

Break Start: (i.e. Spring Break)

Break End:

(All fields with a * are required)

Next >>

October 2002

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

November 2002

Su	Mo	Tu	We	Th	Fr	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

December 2002

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

January 2003

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

©Copyright 2002 by Jean D'Amico. All rights reserved. No portion of this website or its contents may be reproduced or distributed in any form without the express written consent of the author.

Figure 5. A course creation screen

The system defines four types of events. An assignment is an event that is defined as a class activity with a due date attached to it. A test event is an activity that occurs during a lecture on a date assigned by the professor. An extra-credit assignment is an optional assignment. Lectures are hour-long events that take place during the class period. These occur every class day with the exception of days that tests are scheduled. Events have associated features such as type, name, date, description and notification schedule (i.e. to schedule automatic email reminders a certain number of days before the event). The user can manually add a new event and edit or delete an existing one at any time. The system automatically checks each new or modified event against the course calendar for a variety of conflicts. Conflicts include events being held on weekends, holidays, breaks, or overlaps with existing events. If a conflict is detected, the user is asked to modify the conflicting feature. After all conflicts have been resolved, the event is added to the master course calendar.

The course calendar is viewable by day, week, or month. All views, by default, show events associated with all courses for the current semester created by the current user. The user can toggle individual courses on and off, thereby restricting the views to events associated with a specific course or courses. The default view is the current day view (Figure 6). Events can be clicked to bring up full descriptions. Figures 7 and 8 illustrate the week and month views. In these views few details of individual events are provided. However, each day has a 'Go >' button that will bring up that particular day's view.

October 29, 2002

Today's lectures:
COMP1234 – Lecture 17: Complex systems.
COMP5678 – Lecture 22: Chapter 7 continued.

Today's tests:
No tests are currently scheduled for today.

Today's assignments:
COMP9876 – Assignment 1 is due today.
COMP1234 – Assignment 6 is due today.

Today's extra-credit assignments:
No extra-credit events are currently scheduled for today.

Figure 6. Day view

October 20-26, 2002						
Sunday	**Monday**	**Tuesday**	**Wednesday**	**Thursday**	**Friday**	**Saturday**
20	**21**	**22**	**23**	**24**	**25**	**26**
		Lecture **COMP1234** Lecture 12 GO >		**Lecture** **COMP1234** Lecture 13 **COMP9876** Assignment 1 GO >		
			Tests **COMP5678** Test 1 GO >			

Figure 7. Week view

October 2002						
Sunday	**Monday**	**Tuesday**	**Wednesday**	**Thursday**	**Friday**	**Saturday**
		1 **Lecture** **COMP1234** Lecture 12 GO >	2	3 **Lecture** **COMP1234** Lecture 13 GO >	4	5
6	7	8	9 **Tests** **COMP5678** Test 1 GO >	10	11	12
13	14	15 **Lecture** **COMP1234** Lecture 14 GO >	16	17 **Lecture** **COMP1234** Lecture 15 GO >	18	19
20	21	22	23	24	25 **Assignments** **COMP1234** Assignment 4 GO >	26
27	28 **Extra-Credit** **COMP9876** Extra-credit 1 GO >	29	30	31		

Figure 8. Month view

4.2 The Translation Level

This level contains a Java program that receives, as input from the interface level using shell scripting through PHP, all valid information (validity checking is done at the interface level) entered by the user. Java is particularly well suited for this because it is a good programming language for parsing and manipulating string sequences. From the user inputs this level determines constraints such as the available number of class days in the semester after holidays and breaks have been accounted for. These are then combined with a set of predefined constraints. An example will clarify this translation process.

Suppose a user enters the following course information:
Course dates: August 19 2002 - December 6 2002
Holidays: September 2 2002
Break: November 25 2002 - December 1 2002
Schedule: MWF
Tests: 2
Assignments: 2
Extra-credit Assignments: 0
Mid-term date: October 9 2002

The total number of days between the start and end date, 109, is first calculated. Then a determination of which are weekdays and which are actual class days (Mondays, Wednesdays and Fridays excluding holidays and break) is made. Then this level constructs algebraic equations representing constraints implicit in the information entered by the user and several predefined system constraints. One predefined system constraint is that a test and an assignment event cannot occur on the same day. This results in 4 equations of the form *Ti-Aj!=0* where *Ti* is the day of the ith test and *Aj* is the day of the jth assignment. The system also has predefined constraints of the form event-i must occur before event-i+1 for any event type. This generates equations *T1-T2<0* and *A1-A2<0*. Another predefined constraint is that the first test cannot occur within the first seven days of the semester. This generates the equation *T1>7*. A fourth constraint is that assignments need 7 days for students to do them and for the professor to grade. A fifth constraint is that at least one assignment must precede a test. A sixth predefined system constraint is that any outstanding assignments must be graded and returned at least 7 days before a test. These three constraints give rise to the equation *A1+21-T1<0*. A seventh constraint is that a test must be given and returned after grading to students at least a week before the mid-term, which is the last possible date for dropping a course. In the present example, the mid-term is day 51 out of a 109-day semester. This yields the equation *T1+14<=51*. This represents that test 1 must occur at least 14 days prior to the mid-term.

Variables of the form *eiej* stand for the number of days between Event-i and Event-j, where the letter *e* is replaced with *S*, *E*, *T*, *A*, *L* or *X* to represent the start of the semester, the end of the semester, a test, an assignment, a lecture or an extra-credit assignment respectively. This, in the present example, results in six equations *T1-S-ST1<=0*, *T2-T1-T1T2<=0*, *E-T2-T2E<=0*, *A1-S-SA1<=0*, *A2-A1-A1A2<=0* and *E-A2-A2E<=0*. Note that in our example *S=1* and *E=109*. An eighth predefined constraint is that tests be scheduled symmetrically (i.e. as equidistant from each other and the start and end of the semester as possible). This gives rise to the equations *ST1=T1T2* and *T1T2=T2E*. The constraint solver will attempt to maximize these values subject to their sum being less than or equal to the total number of days in the semester. The translation level passes the set of equations developed to the constraint solver level below. It will solve these constraints and return the corresponding variable values back to the translation level.

A solution for the present example is *A1=7; A2=58; T1=37; T2=73; SA1=51; A1A2=51; A2E=51; ST1=36; T1T2=36; T2E =36*. These values stand for the number of days from the start of the semester (i.e. day 1). The translation level translates these numbers into actual dates and checks them to ensure that they do not fall on Saturdays, Sundays, non-class days, holidays, or breaks (if they do, the closest class days are picked instead). Then these are written into the database and passed to the interface level for display. Subsequently, the user is presented a screen that shows him or her the events that have been scheduled with the option to edit each by clicking on it.

4.3 The Constraint Solver Level

Generating feasible schedules by developing and solving constraints is a feature of CB^{2}T that separates it from commercially available course management tools such as WebCT. The Artifact Model indicated that professors created course schedules from relatively few constraints, notably course start and end dates, holidays, break start and end dates, and the number of tests, assignments, and extra-credit assignments. Initially we considered network flow algorithms (Skiena, 1997) for constraint solving. But it was found that it would be difficult to manage the complexity of the network flow graph derived from constraints. As the example above showed, both implicit (i.e. the semester calendar) and explicit (i.e. regarding relationships among various events to be scheduled) constraints can be expressed as algebraic inequalities. Therefore linear programming seemed quite suitable. A linear programming problem can be written in the following form (Standard Form): *minimize cx subject to Ax = b and x >=0*. Here *x* is the set of variables that is being solved for, *A* is a set of defined coefficients, and *c* and *b* are vectors of defined coefficients. The *minimize cx* function is known as the objective function while the equation set *Ax = b* is referred to as the constraints. Depending on the nature of the problem at hand, the objective function could be a maximizing or minimizing function, and *Ax* and *b* could be related by any of the inequalities =, <, >, <=, or >= (Fourer, 2000).

We chose an open-source implementation of the linear programming algorithm called "lp_solve", available from ftp://ftp.ics.ele.tue.nl/pub/lp_solve, for our constraint solver level. It was originally written in C by Michael Berkelaar and has since been ported over to Java. It is one of the most successful linear programming solvers currently available as open-source software on the Internet. The translation level formulates the inequalities that it generates in this standard form, with an objective function to maximize the date of the first test, and passes the problem to lp_solve. The results are then passed back up to the translation level.

5. User Interface Evaluation

The information gathered during the Contextual Design process allowed us to construct profiles of typical users of CB^{2}T. From these profiles we developed use cases and used them in a "cognitive walkthrough" evaluation of the interface. This section provides details of this evaluation.

5.1 User Profiles

Information gathered from interviews suggested that a "typical" user of the system can be characterized as falling between two extreme kinds of users, whom we call the Schedule-driven User (SU) and the Content-driven User (CU).

SU is a highly structured individual who creates detailed plans. He completely plans his course schedule at the beginning of the semester and follows it closely throughout. He uses a highly formalized process with well-defined steps when creating a course schedule. The first step is to analyze the requirements for the course. These requirements come from various sources including the university, literature regarding the topic to be taught, and his professional expertise. After defining the requirements for the course, he refines these into a set of general concepts that all students taking the class should know by the end of the semester. These concepts are translated into specific course objectives. From these objectives, he determines what topics to teach and how many lectures to give on each. This process is time consuming and tedious, but the end result is a detailed daily lecture schedule for the course. SU would use CB^{2}T to assist him in scheduling events once course requirements and objectives have been defined and the lectures planned out. He is quite likely to manually schedule course events such as tests and assignments, or at least significantly modify the automatic schedule that CB^{2}T generates for him. However, once the schedule is thus finalized at the beginning of the semester, he will strictly follow it with minimal modifications during the semester.

CU, on the other hand, is the type of user who describes her course preparation methods as more content-based. She is generally not concerned with planning out her schedule to the minute detail at the very beginning of the semester. She is interested in thoroughly covering the course material, and scheduling tests and assignments based on how much gets covered as the semester progresses. Upon deciding to teach a course, she decides what major topics should be covered. She may also assign a time period, say, about a week per topic. However, this period of time is highly flexible. It is more important to her to cover the topics thoroughly even if this takes longer than expected, than to stay with a predetermined schedule. So she makes only approximate plans at the beginning of the semester, and fills in details later as needed. She wants to have a feasible schedule generated for her by CB^{2}T once she decides on the important course events. This provides her with a view of the semester with tests and assignments fairly evenly distributed throughout the course. It will also provide some insight about the pacing of lectures to cover sufficient material before tests and assignments. But she also knows that this will not necessarily be the schedule that she will follow. Instead, as the semester progresses, she will adjust the schedule based on the actual course content she covers from lecture to lecture. Thus, CU is more likely to keep both a private and a public calendar, moving items from the private one to the public one as she finalizes events. She is also the type of user who is likely to require that multiple feasible schedules be generated.

5.2 Developing Use Cases

Once user profiles had been constructed, the next step was to develop well-defined use cases to demonstrate how the users would interact with the system and how the system would respond. Use cases are an important part of the Unified Modeling Language, or UML. UML, sometimes referred to as the Unified Process, is an object-oriented software design methodology that has become popular in the software development community. Within this methodology, a use case is defined as a “sequence of actions, performed by one or more actors (people or non-human entities outside of the system) and by the system itself that produces one or more results of value to one or more of the actors” (Scott, 2002). Use cases provide detailed specifications of actions that the user performs and the system’s responses. We developed use cases for the four tasks that all users will perform with CB^{2}T: logging in, create a course schedule, manually creating a course event and modifying an already created event. One of these, the use case for creating a course schedule, is provided below. See (D’Amico, 2002) for the others.

Use Case:	Create course
Actor:	Course instructor
Purpose:	Create a new course
Overview:	The instructor creates a new course by entering relevant information
Type:	Primary and Essential

Typical Course of Events – Create Course Use Case

User Action	**System Response**
1. User chooses the ‘Create new course’ menu option	
	2. System displays the ‘Create new course’ Screen 1 of 3 (see Figure 5).
3. User types the course’s name in a field labeled course name, chooses the appropriate class rotation – MWF or TTh and the time – from the drop-down menus labeled class day and time, types the name of the classroom in a field labeled class location, chooses the semester, start and end dates from the appropriate drop-down menus labeled semester, course start and end dates, chooses appropriate holiday dates from the drop-down menus and types in the holiday names in the field labeled holiday name, and chooses appropriate dates from the drop-down menus labeled break start and end date and then clicks on the button labeled “Next”.	
	4. The system checks if any required fields are left blank, and also does error checking on the values. If errors or blank fields are encountered, a corresponding error message is shown. Otherwise, it displays the next screen (Screen 2 of 3).
5. User enters the text book name, text book ISBN, author name, GTA name, GTA location and GTA office hours in the appropriate fields, and clicks on the button labeled “Next”.	
	6. System displays the next screen (Screen 3 of 3).
7. User chooses number of tests, assignments and extra-credit assignments from drop-down menus with the same labels and clicks on the button labeled “Confirm”.	
	8. If user inputs zero tests, zero assignments, and zero extra-credit assignments, a course is created without generating a schedule. The user is returned to the main screen. Otherwise, a feasible schedule is generated and displayed for user’s approval and/or editing.

5.3 Cognitive Walkthrough

The cognitive walkthrough (Preece et al., 1994) is an evaluation process that involves constructing task descriptions from a system specification, and then obtaining critical feedback from one or more usability experts who interact with the system to carry out these tasks. A task description is a brief explanation of a task that the evaluator should complete. It does not give explicit instructions about which buttons to press and other interface-specific directions. Instead, it provides a general description of the task, and it is left to the evaluator to decide the best way to proceed. While completing the tasks, the expert reviewer attempts to predict how a potential user will view the system and potential problems they may face, and suggests design changes. This allows the interface's usability to be evaluated and improved.

We developed two sets of task descriptions from the use cases, one matching the Schedule-driven User profile and the other matching the Content-driven User profile. Both involved logging in, entering information regarding a new course, having a schedule automatically generated, and then rejecting and/or modifying various course events to create a final course schedule. We had one usability expert, who held a Master's Degree in Human-Computer Interaction and who was a doctoral student in the same area, carry out these tasks and provide feedback on interface improvements. Her suggestions included improving the feedback from the system upon user login, more informative feedback from the various course creation screens, more flexibility in entering holiday and break information, more flexibility in accepting or rejecting the entire schedule or parts of a schedule that the system generates, more direct means of editing individual events that the system automatically schedules, and an automatic "copy" facility by which a series of similar events can be quickly created. Implementing these changes is part of our ongoing work.

6. Conclusion

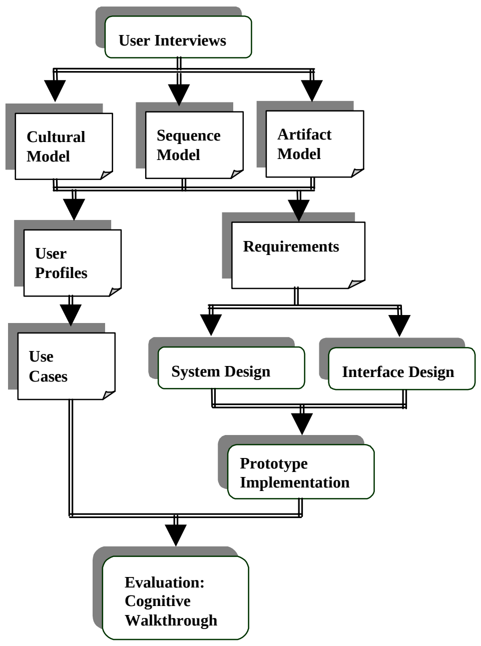


Figure 9. The design process

We presented a case study of the design, development, and evaluation of a course-scheduling tool to assist a professor in planning and scheduling events of a course such as tests, assignments and lectures. This system takes university calendar information (e.g. semester start and end dates, and various holidays and breaks) along with user inputs of the number of tests, assignments, and extra-credit assignments, and translates these into algebraic constraints, which are then solved by a linear programming algorithm to generate a feasible schedule. It was designed using the Contextual Design approach (Beyer & Holtzblatt, 1998). We described the application of this design methodology to generate models (a cultural model, a sequence model and an artifact model) and user profiles, system and interface requirements, and the system architecture. The analytical steps preceding the development of models and requirements included conducting and coding interviews with potential users, and gathering and analyzing work artifacts. Then a prototype was implemented and evaluated through developing use cases and conducting a cognitive walkthrough. Figure 9 summarizes this design process.

By describing this design process and the resulting system as a case study, this report makes two contributions. First, it illustrates the contextual approach to interactive system design in an educational domain. In particular, it shows how one can explore and formalize the tasks of interest through qualitative data collection and model building, which in turn lead to a set of system requirements. A finding, based on user needs identified through the Contextual Design approach, is that faculty users of a system like CB^2T will typically fall somewhere between the two extremes represented by the Content-driven User and Schedule-driven User profiles. So such a system requires both easy-to-use manual scheduling features and automatic scheduling features.

Second, it presents the architecture of a course scheduling tool, with a constraint-solver based on the linear programming algorithm at its heart, that allows the automatic creation of feasible event schedules. Through the use of the constraint-solver, CB^2T is able to take user-defined constraints and combine them with internal system constraints to create a schedule that adequately fulfills the needs of the user. Unlike commercial course management tools such as WebCT, CB^2T provides a novel functionality – the automatic creation of a feasible event schedule for a course – based on constraint satisfaction.

As what we have described is the first iteration of design, clearly, there is more work ahead. The prototype does not yet meet some of the requirements outlined in Section 3. At the moment, its schedules remain private and visible over the web only to the logged-in user. It needs a mechanism to display a schedule on a public web page and an associated interface for the user to easily move events from the private to the public calendar (*requirement 2*). It does not yet have the template creation functionality (*requirement 5*). A capability to schedule automatic email reminders has not yet been implemented (*requirement 6*). Another goal of future work is the design and implementation of a report generation facility by which accreditation reports for courses can be easily created and printed (requirement 8). While the current prototype meets most aspects of *requirement 9*, the constraint solver needs additional work. The current prototype displays an error message if no feasible schedule can be found, otherwise it returns one solution. This needs to be extended to include both an ability to interactively relax constraints if no feasible solution can be found, and a way of ranking and displaying different schedules if there are multiple solutions. At present, the user can implicitly specify constraints by entering various dates and times. But there is no direct way for the user to view or change the predefined constraints internal to the system. So two additional capabilities are required: the capability to automatically acquire constraints relating to semester start and end dates, holidays, etc. from the university calendar, and a constraint editor functionality by which a user can add, delete or modify the predefined system constraints (*requirement 10*). More sophisticated error checking is also desirable. Current and future research will first focus on addressing these issues and improving the interface based on results of the cognitive walkthrough. From a testing perspective, only formative evaluation has been done on the system so far. So the next step will be testing the usability of the improved version and releasing it to faculty for semester-long use in actual courses and subsequent summative evaluation. Ultimately, we plan to release the system as open-source software.
Acknowledgments: This project was supported by National Science Foundation under grant REC-9815016.

7. References

Beyer, H. & Holtzblatt, K. (1998). *Contextual Design: Defining Customer-Centered Systems*. San Francisco, CA: Morgan Kaufmann Publishers.

D’Amico, S. J. (2002). Constraint-based Course Building Tool. Unpublished Master of Software Engineering Project Report, Dept. of Computer Science & Software Engineering, Auburn University, Auburn, AL 36849.

Denzin, N., & Lincoln, Y. (Eds.). (1994). *Handbook of Qualitative Research*. Thousand Oaks, CA: Sage.

Fourer, R. (2000). Linear programming frequently asked questions. Report from the Optimization Technology Center of Northwestern University and Argonne National Laboratory. Retrieved May 22, 2002 from http://www-unix.mcs.anl.gov/otc/Guide/faq/ linear-programming-faq.html.

Preece, J., Rogers Y., Sharp, H., Benyon, D., Holland, S., & Carey, T. (1994). *Human-Computer Interaction*. Wokingham, England: Addison-Wesley.

Scott, K. (2002). *The Unified Process Explained*. Indianapolis, IN: Addison-Wesley.

Skiena, S. S. (1997). *The Algorithm Design Manual*. New York: Springer-Verlag.