A C++ PROGRAMMING SHELL TO SIMPLIFY GUI DEVELOPMENT IN A NUMERICAL METHODS COURSE

Greg Mason and Robert Cornwell
Department of Mechanical Engineering
Seattle University
Seattle, WA 98122

Abstract

One of the difficulties with teaching an introductory programming course using C++ is that there is no simple way to create a C++ program with a graphical user interface under the Windows operating system. While the Microsoft Foundation Classes (MFC) exposes the full Windows graphical user interface with C++ classes, MFC has a steep learning curve and is not practical for an introductory programming course. The traditional alternative is to develop simple command line programs. Research has shown that students are more enthusiastic about programming if they can create programs that include graphics and an interactive interface. This paper presents a C++ programming shell which allows the students to develop fully functional form based programs with little of the programming difficulties associated with the traditional MFC approach. Since switching to the shell, faculty have reported a substantial increase in the quality of the programs the students develop and a decrease in the amount of time that must be spent in class explaining how to create the user interface to get data into and out of a program.

Introduction

Computers are an essential tool in modern engineering and, as a result, are an important component of an engineering curriculum. While there are many commercial engineering software products for design and analysis, inevitably, engineers encounter problems that require some programming. In these cases, the engineer may be required to tweak code or write a script to enhance existing software. AutoCAD users have long been Lisp programmers and Matlab users regularly produce custom scripts. While these are not traditional programs, from the computer science view point, they do require engineers to have a working understanding of programming.

At Seattle University, all mechanical and civil engineering students are required to take a one quarter numerical methods and programming course. The course uses C++ for the programming language. C++ was chosen for several reasons. First, C++ allows students to create fast standalone programs which do not require support from secondary packages such as Matlab. Thus students can create software which they will be able to use on a wide range of computers. Second, C++ requires students to understand fundamental software concepts such as memory use, pointers and classes. These concepts are obscured in high level languages such as Matlab or even Java. Third, C++ is used in a required junior level data acquisition course since it is one of the few languages supported on the real-time hardware used in this class. This programming course is a prerequisite for the data acquisition course.

One of the difficulties with teaching C++ is that there is no simple way to create a C++ program with a modern user interface under the Windows operating system. While Microsoft Foundation Classes (MFC)[1] exposes the full Windows graphical user interface (GUI) with C++ classes, MFC has a steep learning curve and is not practical for a ten week course. We did not, however, want to relegate our students to creating simple command line programs. Research has shown that students are more enthusiastic about programming if they can create programs that include graphics and an interactive interface, Park[2].

The mechanical engineering program's solution was to develop a C++ shell program for use in the programming course. The shell simplifies the creation of the user interface and handles many of the tedious tasks associated with GUI programming. This lets students focus on the technical details of programming numerical methods while still having the ability to create a complete Windows GUI for their programs with minimal effort.

The remainder of the paper is divided into four sections. The first provides an overview of the topics covered in the course and lists the types of projects completed by the students. The second includes a technical description of the shell, an outline of its key features, and a simple example program. The third shows some examples of programs complete by students. Conclusions and recommendation are provided in the final section.

Course Overview

The programming course is taken by mechanical and civil engineering students during their sophomore year. In the quarter system, the class is a 4-credit course. The course provides students with fundamental programming skills and introduces them to numerical methods commonly used in engineering. The course builds on analytical methods the students have already used in their prerequisite math, physics and engineering courses. Using C++ to implement the analytical methods numerically reinforces the analytical methods, illustrates how analytic methods can be implemented numerically, and introduces basic programming practices all at the same time. The course is the only programming and numerical methods course the students are required to take. The course covers the following topics:

Linear and Polynomial Interpolation

Students use both linear and polynomial interpolation, primarily Lagrange polynomials, to extract intermediate data points from a three dimensional data set. This is typical of problems encountered in thermodynamics where experimental data is available for a gas at several operating conditions, but you need the data at an intermediate condition.

Statistics and Linear Regression

Students calculate the mean and standard deviation of a data set and plot the least squares fit to the data using a straight line and a power curve. Students also use computer graphics to plot a two dimensional data set.

Numerical Differentiation and Integration

Students compute numerical derivatives using backward, central and forward difference algorithms for a set of data points. Students develop integration functions using both Trapezoidal and Simpson's Rule techniques. These functions are then used in the student's shear and bending moment diagrams programs. An example of student work is provided in the student examples section.

Matrices and Systems of Equation

Students use Gauss Elimination with partial pivoting to solve a truss problem. Their programs are used to solve for the force in each truss member and create a 2D plot of the truss where the truss members are color coded based on their internal force.

Numerical Solution of Differential Equations

Students use Eulers and Runga-Kutta methods to solve for the time response of a second order spring-mass-damper system. Their programs generate a position versus time graph of the motion and show an animated simulation of the system's movement. An example of student work is provided in the student examples section.

Roots of Equations

Students use Marching, Bisection and Regula Falsi methods to solve for the zeros of a nonlinear equation. An example of student work is provided in the student examples section.

In addition to learning to apply numerical techniques to the solution of engineering problems, students learn fundamental programming concepts. These concepts include, structured programming[3], branching, looping and functions. Variables are discussed in the light of their storage requirements and binary format. String and arrays are used to organize data. Files are introduced as a way to provide persistent storage for data. Classes are utilized in the course, but students are not required to develop their own class structures.

Programming Shell

The amount of computer work required by the students in the ten week numerical methods course is substantial. As discussed earlier, we did not want students to be encumbered with interface design and not have the time, or motivation, to focus on the numerical methods presented in the course. To aide the students, a programming shell was developed that simplifies the creation of Windows programs. The shell hides the Windows Application Programming Interfaces (APIs) from the students, but does not restrict their ability to create form based Windows programs. The shell is based on a Win32 program that does not require any DLL's, ActiveX or COM components, so students do not need to worry about component registration or installations issues. The finished program is standalone executable and does not require an installer.

The shell includes features that make it easy for student to quickly create Windows programs. These features include:

1) Access to common window interface items, such as pushbuttons, radio buttons, edit boxes and dropdown lists. These items are accessed through a set of classes predefined in the shell. Students use a set of common methods, such as setvalue() or getvalue() to interact with the interface items. Methods are also included for enabling/disabling, hiding/showing items and changing the item's font. The available GUI classes and their methods, are summarized in Table 1 (next page).

2) Text parsing functions that can be used to extract data values from edit boxes. These methods make it easy for students to enter large amounts of data into their programs without using files. Students can copy and paste data into an edit box and then use the text parsing methods to extract data into arrays or variables. The text parsing function allows the file access topics to be moved towards the end of the course, since students do not need to read data from files.

3) A double buffered graphic subsystem. This system automatically takes care of refreshing the screen in response to Windows OS requests and simplifies plotting and programs with animation. This relieves the students from the substantial amount of work that would normally be involved in implementing graphics. The shell also includes simple graphics management features that let students create flicker-free animations.

4) Integration with Visual Studio's resource editor. Students design the GUI layout of their program using the graphical resource editor included with Visual Studio. The programming shell allows the students to create form based programs using an approach very similar to that required when using Visual BASIC.NET or C#.NET.

5) A set of graphing commands. These let students use engineering units to specify graph ranges and data. Students can have multiple graphs in their program and can respond to user clicks on a graph. All scaling and graph updates initiated by the operating system are handled by the shell.

Table 1 User interface class summary.

Class	Included Methods	Descriptions
BUTTON	getValue, enableItem, showItem, setFont	Use to manage push buttons. Includes methods to hide and show buttons and change fonts.
CHOICEBOX	getValue, setValue, enableItem, showItem	Use to manage check and ratio buttons. Includes methods to check and change button states.
COMBOBOX	getValue, enableItem, showItem, setListText	Use to manage drop lists such as menus. Includes methods to change items in the list.
EDITBOX	getValue, enableItem, showItem, initializeScan, getNextItem, getArray, getList	Use to manage edit boxes. Includes methods to set or retrieve text box entries, and methods for parsing text.
FILENAME	getOpenFileName, getSaveFileName	Access to standard open and save dialog boxes.
GRAPH	clear, drawBitmap, drawIcon, drawSymbol, drawText, drawAxis, setScale, moveTo, lineTo	Use to create and manage graphs. Includes methods for plotting and scaling graphs, and responding to user clicks in the graph area. Graphs can also be used to create simple animations.
LABEL	setValue, enableItem, showItem, setFont	Use to manage static text. Includes methods to change text output and font.
MESSAGE	askMessage, showMessage	Creates popup message boxes.
SLIDER	setValue, getValue, enableItem, showItem, setRange	Use to manage slider interface items.
TEXTFONT		Use to manage fonts and change the look of interface items.

6) Open source code. Students have access to the full shell source code. Ambitious students can modify the shell or add features as needed.

To create a shell based program, students open the shell template using Visual Studio. The shell is made up of several source, header and resource files. Most of these files exist to provide core functionality for the shell and do not need to be edited by students. Students design their program interface using Visual Studio's resource editor. This is a graphical drag and drop editor for placing and sizing interface items used in their program. Interface design is similar to designing a form in Visual Basic or C#.

Once the interface is designed, each interface item is declared in the main program. This creates an instance of a C++ class for the interface item. These objects can then be used within the program, using class members, to interact with the interface. The class constructor for interface items uses the name of the resource item to associate a particular instance of the class to a particular interface item. In order to programmatically respond to user actions, the students add case statements to the ProcessButtons() function. This function is called by the shell whenever the user interacts with an interface item. Case items are associated with interface items using the same resource identification as used in the interface object construction. The ProcessButtons() function can also be used to perform actions automatically when the program starts or stops.

While the ProcessButton switch/case statement structure is not as elegant as the method used by MFC, it does lead to simpler code. MFC uses overwritten member function to respond to user interface actions. The problem with this approach is that it requires the use of preprocessor message maps to associate the actual Windows OS message with the overridden class member. These message maps are automatically generated by the editor when the programmer creates code to respond to user actions. Experienced Windows programmers understand this process. However, automatic code generation and preprocessor based message maps are outside the scope of our introductory programming course. As a result our shell was designed using a simple switch/case structure. This structure also closely matches how the operating system actually processes user events, and so also helps students understand how messages are processed using a single thread.

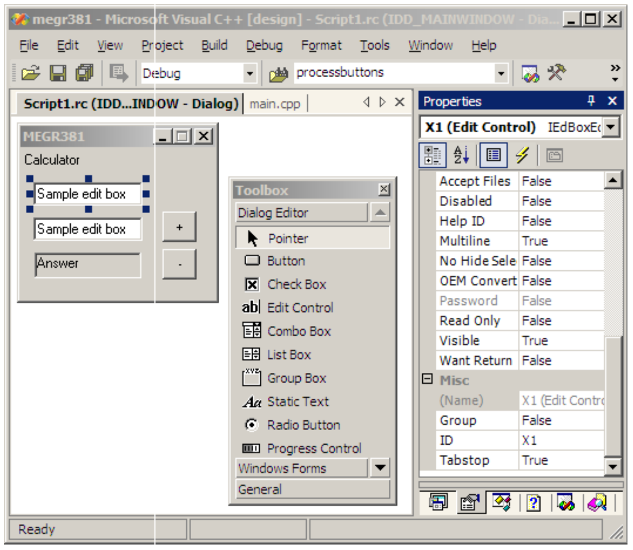
An example program, that adds or subtracts two numbers using the shell, is shown in Fig. 1. User interface items are declared at the top of the source code. In this example these correspond to two edit boxes and one label placed on the interface as shown in the resource editor, Fig. 2. Events are processed in the ProcessButtons() function. The switch/case structure used in the function handles four events. The START_PROGRAM and QUIT_PROGRAM cases are predefined in the shell and are used to initialize or destroy values. The ADD and SUBTRACT cases correspond to the Add and Subtract buttons on the user interface, Fig. 3. Here interface items are accessed and processed to provide program functionality.

Student Examples

The success of the shell is best demonstrated by the programs that the students are able to create. Because creating a sophisticated user interface is fairly simple using the shell approach, students are able to focus more of their time working on the engineering aspects of their programs. The software examples

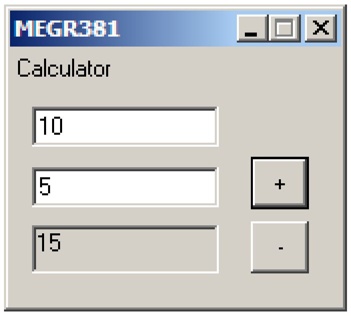
```
#include "support381.h"
#include <math.h>

// Globals --------------------
// Declare all interface items and global variables here

EDITBOX    x1input(X1);          // input textbox object
EDITBOX    x2input(X2);          // input textbox object
LABEL      answer(XOUT);         // output the answer here
// ---------------------------

// ProcessUserEvents -----------------------
//
// Handles events from Windows
// User supply event handlers for each interface item they add
//
// Events can be any of the following:
//   1) START_PROGRAM = program is starting
//   2) QUIT_PROGRAM = program is quitting
//   3) ID of a interfact item = the user pressed the item
// Inputs:  id,   the ID of the item
// Output:  none
// Return:  none
// ---------------------------------------

void ProcessUserEvents(unsigned long id){
          double x1, x2;
          switch (id) {

          // handle any user actions
          case ADD: // user tapped the add button
                    x1=x1input.getvalue();  // get the values
                    x2=x2input.getvalue();
                    answer.setvalue(x1+x2); // output the answer
                    break;

          case SUBTRACT: // user tapped the subtract button
                    x1=x1input.getvalue();  // get the values
                    x2=x2input.getvalue();
                    answer.setvalue(x1-x2); // output the answer
                    break;

          case START_PROGRAM:  // this occurs at program start
                    x1input.setvalue(0.0); // clear everything
                    x2input.setvalue(0.0);
                    answer.setvalue(0.0);
                    break;

          case QUIT_PROGRAM:  // this occurs at program end
                    break;
          }
}
```

Figure 1. Example Program Source Code.



Accept Files	False
Disabled	False
Help ID	False
Multiline	True
No Hide Sele	False
OEM Convert	False
Password	False
Read Only	False
Visible	True
Want Return	False
(Name)	X1 (Edit Contro
Group	False
ID	X1
Tabstop	True

Figure 2. Resource Editor Showing Program Layout Design.


MEGR381

Calculator

10

5

15

Figure 3. Final Program.

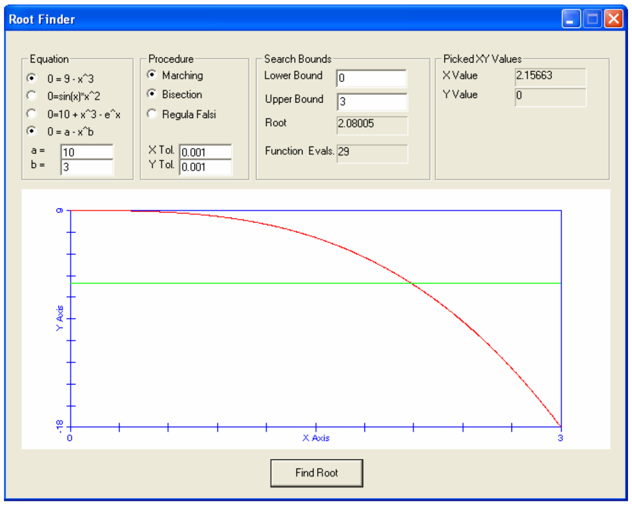
discussed below were created or extended by students as homework assignments or, as in the case of the truss analysis program, by teams of two to three students over a two week period as part of a required final project.

Root Finder

The root finder program shown in Fig. 4 was written by an engineering student for a weekly homework assignment where students were required to implement marching, bisection and regula falsi methods to find the root of an equation. The user can select an equation type, enter associated parameters, and enter the search range. The software plots the equation and outputs the root of the equation that falls between the user specified search bounds. When the user clicks on the graph, the software outputs the value of the function at the corresponding mouse location. The program illustrates the type of user interfaces students can quickly create using the shell program.

Spring-Mass-Damper Vibration

The spring-mass-damper program shown in Fig. 5 was written by an engineering student for a weekly homework assignment where students were required to implement a Runga-Kutta or Eulers method to solve a second order differential equation. The user can vary the equation parameters and then have the software simulate the response described by the


Root Finder

Equation: [✓ 0 = 9 - x^3] 0=sin(x)*x^2 0=10 + x^3 - e^x [✓ 0 = a - x^b]

a = 10
b = 3

Procedure: [✓ Marching] [✓ Bisection] Regula Falsi

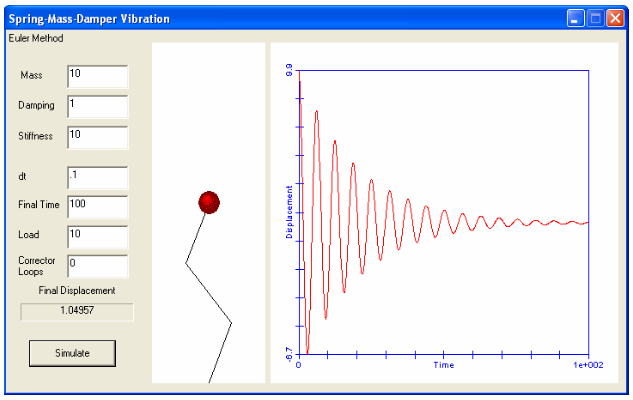
X Tol. 0.001
Y Tol. 0.001

Search Bounds
Lower Bound 0
Upper Bound 3
Root 2.08005
Function Evals. 29

Picked XY Values
X Value 2.15663
Y Value 0

Figure 4. Root Finder.

differential equation. During simulation, the ball and spring, in the center of the screen, bounce up and down, in real time, according to the computed solution. The graph on the right shows a time domain plot of the ball position. The program illustrates how students can use the graphing class to create graphs and animations. This student used the drawIcon methods to draw the bouncing ball and scaled the graph with engineering units using the setScale method. Since the shell uses a double buffered graphics system, the ball animation is smooth and flicker free.


Spring-Mass-Damper Vibration

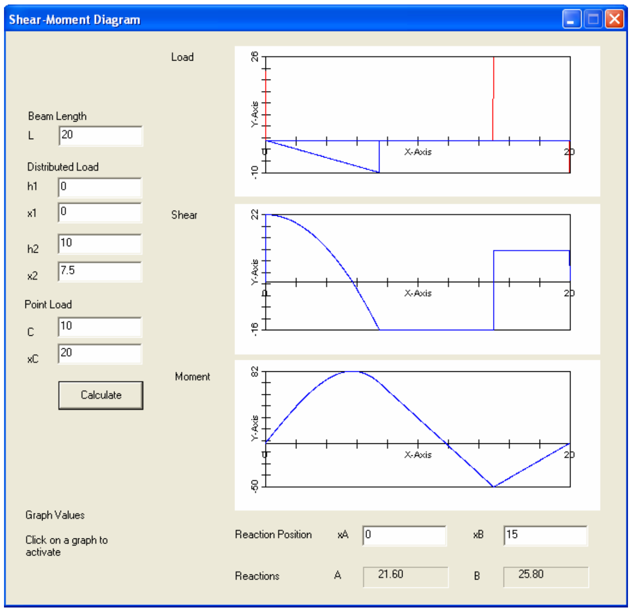
Euler Method

Mass 10
Damping 1
Stiffness 10
dt .1
Final Time 100
Load 10
Corrector Loops 0
Final Displacement 1.04957

Figure 5. Spring-Mass-Damper Vibration.

Shear and Moment Diagram

The shear and moment diagram program shown in Fig. 6 was written by an engineering student for a weekly homework assignment where students were required to use numerical integration to generate shear and moment diagrams given a load diagram. Shear is found by integrating the load, and moment by integrating the shear. Students modified a starter program which drew the initial load diagram. In the program below, the user can enter various loading cases and have the software draw the shear and moment diagrams. When the user clicks on any of the diagrams, the software displays the corresponding load, shear or moment value. The program illustrates how the graph methods can be used to create complex outputs. Each graph is a separate entity with its own scaling.


Shear-Moment Diagram

Beam Length
L 20

Distributed Load
h1 0
x1 0
h2 10
x2 7.5

Point Load
C 10
xC 20

Calculate

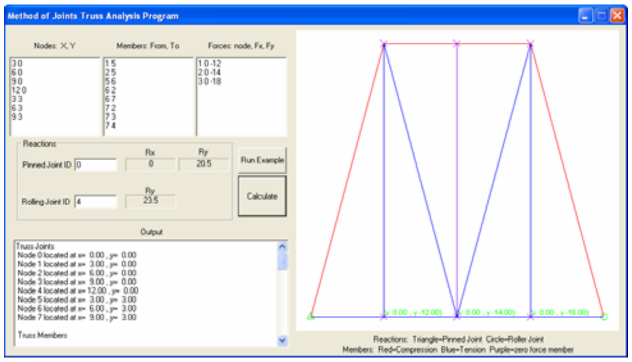
Graph Values
Click on a graph to activate

Reaction Position xA 0 xB 15
Reactions A 21.60 B 25.80

Figure 6. Shear and Moment Diagrams.

Method of Joints Truss Analysis

The Method of Joints truss analysis program is shown in Fig. 7. The program was developed in approximately two weeks by a team of three students as their final project. The user supplies node, member, force and support information. The program then draws the truss and identifies the loaded and supported nodes. A color code is used to indicate members loaded in tension or compression. A summary of user supplied input, resultant reactions and member forces is also provided in a scrollable editbox.


Method of Joints Truss Analysis Program

Nodes: X,Y	Members: From, To	Forces: node, Fx, Fy
3 0	1 5	1 0 -12
6 0	2 5	2 0 -14
9 0	5 6	3 0 -18
12 0	6 2	
3 3	6 7	
6 3	7 2	
9 3	7 3	
	7 4	

Reactions
Pinned Joint ID 0 Rx 0 Ry 20.5
Rolling Joint ID 4 Ry 23.5

Run Example
Calculate

Output

```
Truss Joints
 Node 0 located at x=  0.00 , y=  0.00
 Node 1 located at x=  3.00 , y=  0.00
 Node 2 located at x=  6.00 , y=  0.00
 Node 3 located at x=  9.00 , y=  0.00
 Node 4 located at x= 12.00 , y=  0.00
 Node 5 located at x=  3.00 , y=  3.00
 Node 6 located at x=  6.00 , y=  3.00
 Node 7 located at x=  9.00 , y=  3.00

 Truss Members
```

Reactions: Triangle=Pinned Joint Circle=Roller Joint
Members: Red=Compression Blue=Tension Purple=zero force member

Figure 7. Method of Joints Truss Analysis.

Conclusions

The Mechanical Engineering Department at Seattle University has been using the shell successfully for the past six course offerings and found it to be a powerful tool for teaching numerical methods. Prior to using the shell the department taught the numerical methods course using either a command line or MFC interface. Since switching to the shell, faculty have reported a substantial increase in the quality of the programs the students develop and a decrease in the amount of time that must be spent in class explaining how to create the user interface to get data into and out of a program.

Two years ago the shell was ported to the Windows CE real time operating system for use in a data acquisition and instrumentation class. Students now use a common programming structure for both courses, greatly reducing the amount of time students must spend learning to program under Windows CE. Since the shell has a compact structure it is easy for students to implement both cooperative and preemptive threads to control timing in their data acquisition systems.

Since changing to the shell, we have noticed that students display more enthusiasm for the class. By the end of the course some students are creating software packages for their own use. The only drawback we have noted with use of the shell is that there are currently no textbooks that follow this approach. Many books exist that use either the command line interface or the full MFC programming interface. Occasionally students express frustration at not having a textbook to follow. We are studying the possibility of creating a console component that will allow the students to mimic the iostream library's cin and cout methods. This will make it easier for students to follow examples from books using the console based approach.

Resources

The shell and the example programs discussed in this paper are available online at www.seattleu.edu/scieng/me/C++Programming Shell The shell requires the C++ component of Visual Studio 2005 or better.

References

1. Prosise, Jeff, "Programming Windows with MFC", Microsoft Press, 1999

2. Park, W, "Why isn't my professor using graphics in the freshman programming course", Journal of Engineering Education, Oct 1996, pp331-336

3. Nassi, I. and Shneiderman, B., Flowchart Techniques for Structured Programming, SIGPLAN Notices 8, 8 (August, 1973).

Biographical Information

Greg Mason received a BSME from Gonzaga University, an MS in Computer Integrated Manufacturing from Georgia Institute of Technology, and a Ph.D. in Mechanical Engineering from the University of Washington-Seattle. He developed a robotics laboratory for the Department of Defense in Keyport, WA and was involved in numerous automation projects, including a robotic container welding system and a robotic torpedo fueling system. He has published papers addressing the uses of advanced controls system techniques in manufacturing. He is also active in pedagogical research. He has developed a handheld data acquisition system which uses the techniques outlined in this paper and which is currently in use at Seattle University.

Bob Cornwell received a BS in Civil Engineering and a MS in Engineering from the University of Texas-Austin. He received a Ph.D. in Engineering Mechanics from the University of Wisconsin-Madison. He has worked for Exxon Production Research where he participated in the development of optimization and reliability methods for offshore structures and the Boeing Company where he contributed to the development of analytical methods for both composite and metallic airframe structures. His interests involve the application of analytic methods to the design of mechanical systems. Since joining Seattle University he has been involved in the development of the program and the teaching machine design, mechanics and numerical methods courses.