Developing programming skills to enhance engineering dynamics modelling

Desmond ADAIR
Australian College of Kuwait
Mishref, Kuwait
d.adair@ack.edu.kw

Martin JAEGER
Australian College of Kuwait
Mishref, Kuwait
m.jaeger@ack.edu.kw

ABSTRACT

This paper details a successful approach to developing programming skills, central to projects which simulate dynamical systems. The approach here is to include the modelling of nonlinear differential equations which are best solved numerically, so reducing the amount of model simplification and increasing accuracy. The programming language used is Java, which presents an opportunity for high quality simulation models in dynamics. A project which simulates the characteristics of an inverted pendulum, is chosen to illustrate use of the Java programming language. The path taken by students to complete projects is outlined, highlighting necessary skills they must acquire. A detailed description of the modelling of the inverted pendulum is given and the findings of a survey completed by the students are also reported.

Keywords*: Programming, modeling, evaluation, engineering dynamics.*

INTRODUCTION

An important skill in engineering dynamics is the ability to simulate and visualize dynamical behaviour. Traditionally, engineering dynamics courses at undergraduate level emphasize analysis of kinematics and dynamics using Newtonian and Lagrangian mechanics approaches, resulting in formulation of equations of motion in the form of ordinary differential equations. To analytically solve the more complicated and usually practical systems resulting from the analyses, simplification and the use of linear differential equations are necessary.

This limits the number of problems which can be considered and reduces the exposure of students to modelling practices found in the engineering profession. A better approach, and the one used here, is to include the modelling of non-linear differential equations that are best solved numerically, so reducing the need for excessive model simplification and usually resulting in increased accuracy. A prerequisite, however, for students to follow such an approach with ease, confidence and competence, is knowledge of a suitable programming environment.

The Java programming language (Liang, 2009) presents an opportunity for high quality simulation models in dynamics and related areas (Kilgore *et al.*, 1998). Java, like other programming languages, has its own strengths and weaknesses (Meehan and Lunney, 2001) and (Thimbleby, 1999), but despite some limitations and after much debate as to its merits when compared to other languages (Mittermeir and Boszormenyi, 1997, Noon, 1994), and, when it should be introduced into the curriculum (Harvey and Deitel, 2002, Tyma, 1998), it has grown in popularity. There are many already-written application-specific Java classes which are compatible and reusable simulation components. These object-orientated components can be developed using inexpensive, professional-quality Java development environments. Also, as the students are working directly with source codes, there is the capability of incorporating tools which are state-of-the-art, or most suitable, efficiently and with understanding. This is superior to modelling facilities with drop down menus. In addition Java is now being used widely as a general purpose language incorporating sophisticated graphical user interfaces on the client-side and as a robust server-side programming language on the server-side (Benander, *et al.*, 2004) so students are being exposed to a widely used useful tool.

A project which simulated the characteristics of an inverted pendulum as described by Goldstein *et al.* (2002) and Landau *et al.* (1976), was chosen to illustrate use of the Java programming language in engineering dynamics modelling. The inverted pendulum is a classical problem in engineering dynamics and is also used in control theory. An ordinary differential equation system, deduced by applying Lagrange multipliers to Newton's equations of motion is applied to solving the inverted pendulum problem. To integrate the equations of motion, use is made of a Runge-Kutta type numerical method. For simulation of the inverted pendulum, the Java programming language is used to build the model interface, control the model, measure variables, assign and display inputs and provide data output for display. The approach adopted here follows the model-view-controller pattern as detailed by Eckstein (2007).

The Java programming language was delivered initially through a basic Java programming and applications course during a previous semester and the knowledge and skills acquired were built on incrementally as students completed their engineering projects. The basic course is a combination of (seminar-style) lectures, supervised laboratories, tutorials and several assignments.

The course spans one semester and the language is taught from a beginner's level (covering basic syntax) to a more advanced level (covering areas like introducing threads and GUI programming). The expected learning outcomes are to produce students with an ability to solve design problems using OO techniques and be able to implement the design effectively using the Java language and Java API framework. While this is the ideal situation, the reality is that most students found the language (and the related OO concepts) relatively easy to comprehend and apply, but other students (a significant minority) found Java difficult at various levels.

In addition to describing the simulation process, this paper also reports on the findings of a survey of students to try to identify aspects of the language students consider the most difficult.

A TYPICAL SIMULATION PROJECT

Inverted Pendulum Model -Analysis

The modelling problem chosen to illustrate the use of the Java programming language can be described with the help of equations of motion, which lead to a system of differential algebraic equations as described by Karasözen *et al*. (1995). These equations, in descriptor form, are Lagrange equations of the first kind and describe a mechanical system of bodies with massless connections.

The equations, according to Newton, without constraint are,

$$\begin{aligned} \dot{p} &= v \\ M(p,t)\dot{v} &= f(p,v,t) \end{aligned} \tag{1}$$

with $p \in \Re$ the vector of position coordinates, $v \in \Re^n$ the vector of the velocity coordinates, $M(p,t) \in \Re^{n,n}$ the symmetric and positive definite mass matrix and $f(p,v,t) \in \Re^n$ the vector for the applied external forces. Due to the connections the following constraints occur,

$$b(p,t) = 0 \qquad b(p,t) \in \Re^q \tag{2}$$

Using Lagrange multipliers $\lambda(t) \in \Re^q$, Eqn. (1) can be transformed to the descriptor form,

$$\dot{p} = v$$

$$M(p,t)\dot{v} = f(p,v,\lambda,t) - \left(\frac{\partial}{\partial p}\right)\left(b(p,t)\right)^T \lambda \qquad (3)$$
$$b(p,t) = 0$$

Eqn. (3) is a differential-algebraic system of index three and these are the equations applied to the inverted pendulum problem. The mechanical system modelled is shown on Figure 1 and consists of a cart of mass m concentrated in the joint and one bar of length l_1, inertia I_1 and mass m_1 concentrated at its centre.

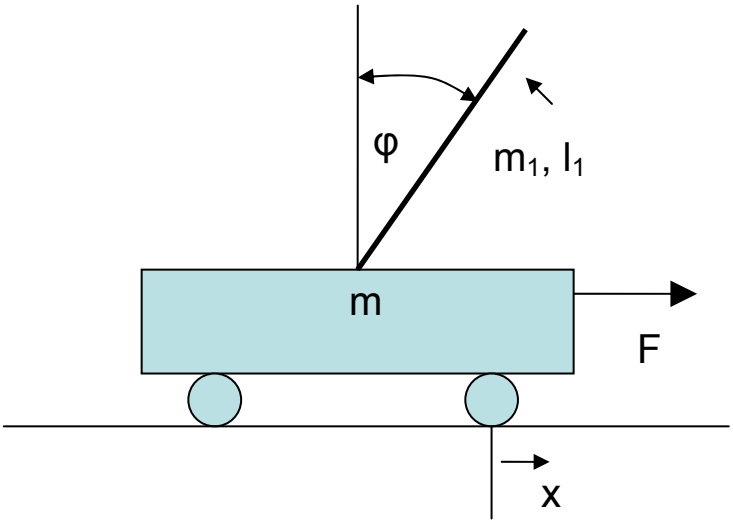

Figure 1: Cart with inverted one-bar pendulum.

The cart can move in the horizontal direction and coordinates p are, the position x of the cart, the position x_1, y_1, of the bar centre and the angle φ_1. On introducing constraint forces F_{z1}, ...,F_{z4} , the Newton formulation gives,

$$m\ddot{x}(t) = F + F_{z1} \qquad m_1\ddot{x}_1(t) = F_{z2} \qquad m_1 y_1(t) = -m_1 g + F_{z3}$$
$$I_1\ddot{\varphi}_1(t) = F_{z4} \qquad x(t) - x_1(t) + \left(\frac{l_1}{2}\right)sin\left(\varphi_1(t)\right) = 0 \qquad (4)$$
$$y_1(t) - \left(\frac{l_1}{2}\right)cos\left(\varphi_1(t)\right) = 0$$

The last two equations describe the geometry,

$$b(p) = b(x, x_1, y_1, \varphi_1) = 0$$

of the system and the inertia of a bar with length l_1 is, $I_1 = \frac{1}{3} m_1 l_1^2$.

Using D'Alembert as given by Karasözen, 1995, the descriptor form is,

$$\begin{aligned} m\ddot{x}(t) &= F + \lambda \\ m_1 \ddot{x}_1(t) &= -\lambda_1 \\ m_1 \ddot{y}_1(t) &= -m_1 g - \lambda_2 \\ I_1 \ddot{\varphi}_1(t) &= \lambda_1 \left(\frac{l_1}{2}\right) cos(\varphi_1(t)) - \lambda_2 \left(\frac{l_1}{2}\right) sin(\varphi_1(t)) \\ 0 &= x(t) - x_1(t) + \left(\frac{l_1}{2}\right) sin(\varphi_1(t)) \\ 0 &= y_1(t) - \left(\frac{l_1}{2}\right) cos(\varphi_1(t)) \end{aligned} \tag{5}$$

The state space form can be deduced from Eqn. (5) by solving for, $\ddot{x}(t)$ and $\ddot{\varphi}_1(t)$.

$$\begin{pmatrix} m + m_1 & m_1 \left(\frac{l_1}{2}\right) cos(\varphi_1(t)) \\ m_1 \left(\frac{l_1}{2}\right) cos(\varphi_1(t)) & I_1 + \left(\frac{l_1}{2}\right)^2 \end{pmatrix} \begin{pmatrix} \ddot{x}(t) \\ \ddot{\varphi}_1(t) \end{pmatrix}$$

$$= \begin{pmatrix} F + m_1 \left(\frac{l_1}{2}\right) sin(\varphi_1(t)) (\dot{\varphi}_1(t))^2 \\ m_1 \left(\frac{l_1}{2}\right) sin\, \varphi_1(t) g \end{pmatrix} \tag{6}$$

which is the classical state space formulation consisting of two second order ordinary differential equations with a symmetric mass matrix.

A description of the integration of the equations of motion and Java programming for the inverted pendulum model is given in the Appendix.

Inverted Pendulum Model – Simulation Description

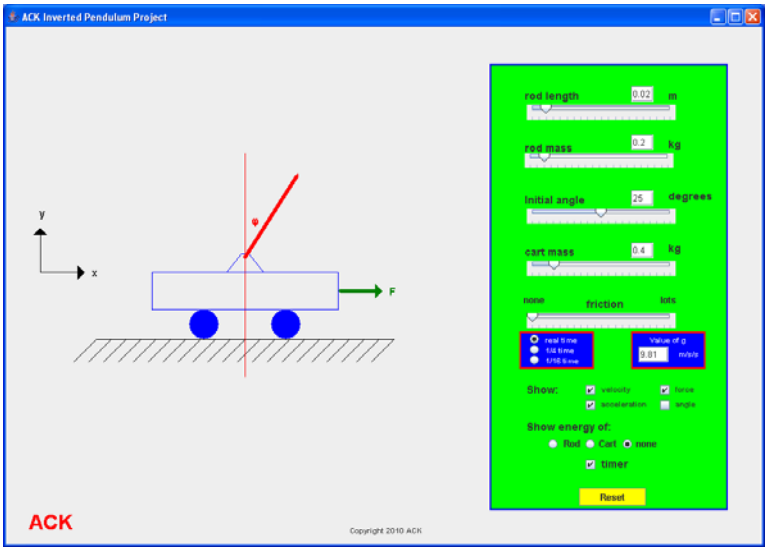

Figure 2: View of the simulation interface.

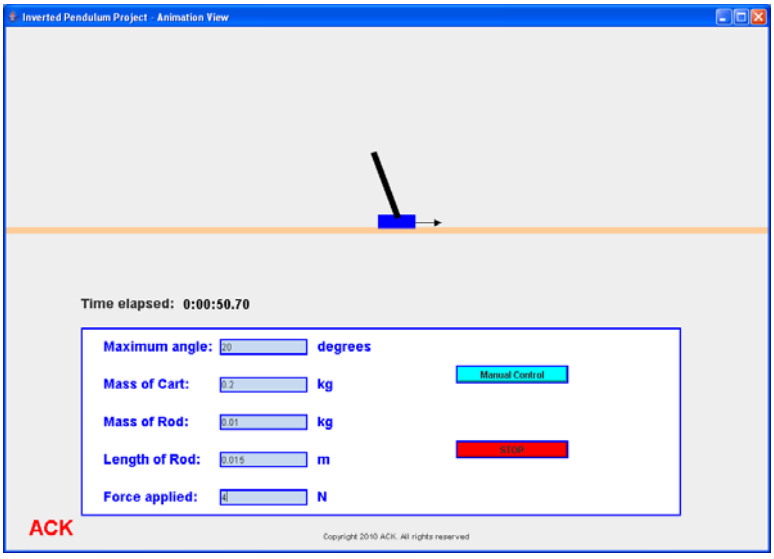

Figure 3: Snapshot of animation window.

A typical example of a simulation interface and a snapshot of an animation window are shown on Figures 2 and 3.

TEACHING SYLLABUSES

When teaching basic Java programming skills, the development environment initially used was BlueJ as described by Barnes and Kölling (2008), which was developed as part of a university research project to help novice students think in terms of objects. The BlueJ IDE, although still occasionally unstable, provides an easy-to-use teaching environment for beginners and places emphasis on visualisation and interaction techniques to create a highly interactive environment that encourages experimentation and exploration (Madden and Chambers, 2002). During the latter part of the basic course and during the development of projects, students used the JCreator IDE (JCreator, 2010), which is a simple yet professional environment and has features including project management, templates, syntax highlighting and class views. A brief outline of the basic Java programming and applications course is found in Table 1.

Table 1: Basic Java Programming and Applications Course Syllabus.

Week 1	Unit introduction, Programming terms & tools, Computing terms & tools
Week 2	Solving problems with computers. Data storage primitive types, Data storage manipulation of primitive data
Week 3	Using class libraries, Object methods, Class methods
Week 4	Flow of control branches, Planning & implementing branches, Multi-way branching
Week 5	Flow of control loops, Implementing loop algorithms, Nesting flow
Week 6	Extending existing classes, Parameters & return values, Extending classes, Graphical User Interface
Week 7	New classes simulating real world objects, Implementing and using new classes, Testing & Documenting programs
Week 8	New classes to organise tasks, Method decomposition, GUI interface adding components
Week 9	Structured data, Arrays, Declaring & filling arrays, Using arrays inheritance polymorphism
Week 10	Standard array algorithms, Sorting algorithms, Searching algorithms
Week 11	GUI interactivity event driven programs, GUI implementation events, Input from the GUI
Week 12	Run time errors exceptions, Recursion concepts, Recursion implementation
Assessments	3 Online test, 3 Practicals, 2 Assignments, Final Exam

A brief outline of the Engineering Dynamical Systems Course which follows Beer *et al*. (2007), also taught in a semester before the engineering projects begin is given in Table 2.

Table 2: Brief outline of Engineering Dynamical Systems Syllabus.

Week 1	Introduction to Dynamics
Week 2	Rectilinear motion of particles
Week 3	Curvilinear motion of particles
Week 4	Kinematics of particles, Newton's second law
Week 5	Kinematics of particles, Energy and momentum methods
Week 6	Systems of Particles
Week 7	Kinematics of rigid bodies
Week 8	Plane motion of rigid bodies, Forces and accelerations
Week 9	Plane motion of rigid bodies, Energy methods
Week 10	Plane motion of rigid bodies, Momentum methods
Week 11	Kinetics of rigid bodies in three dimensions
Week 12	Vibrations without damping, Damped vibrations
Assessments	6 Assignments, Mid-semester test, Final examination

Extra skills, necessary to complete the engineering dynamics projects, were taught using a less formal, selective and highly "hands-on" approach. By selective is meant that students were matched to Java classes and methods which suited their particular project. Students were expected to review and analyse already written classes, with and without help of a tutor, and, to either enlarge on these or write their own, obtaining similar outcomes. A strong emphasis was put on the model-view-controller approach using the modified MVC design shown on Figure 4.

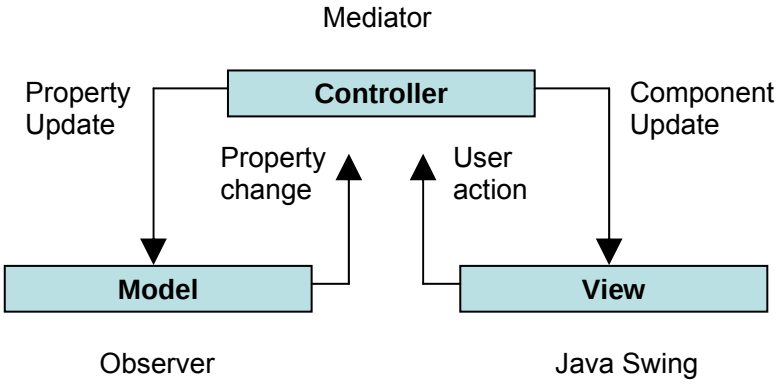

Figure 4: MVC design placing the controller between the Model and the View.

Students were invited to inspect the items shown in Table 3 and submit a list of topics they felt they would need further instruction and exploration of to help complete their projects. Advice was also available to the students to achieve a good match of topics and their needs.

Table 3: Further Java Programming Language Instruction during Project Semester.

Introduction to Model-View-Controller Approach	Model element, View element, Controller element
MVC Design	Instantiation of three components, Recognising a GUI action has occurred using a listener method, How View interacts with Controller, How Controller access and updates Model, How to notify interested listeners of change in Model, Examples and use of MC design
Model Element	review of Math class, get() and set() methods, update() method, toString() method
View Element	review of implementation of JPanel, insertion into JDialog and JFrame, review of paintComponent() and repaint() methods, initComponents() methods, JSpinner and JSlider objects, adding DocumentListeners() to JTextField components, setModel() and modelPropertyChange() methods
Controller Element	review of NoSuchMethodException handler, AbstractController, ArrayList objects, PropertyChange() method, setModelProperty() method
Issues with Application Design	how to handle modelPropertyChange() to avoid ambiguity, multiThreading
Common Swing Event Listener Methods	Further explanation and practice with: JButton, JCheckBox, JRadioButton, JToggleButton, JSpinner,JTextField, JFormattedTextField, JPasswordField, JTextArea, JComboBox, JScrollBar, JScrollPane, JPopupMenu, JSlider

DISCUSSION AND EVALUATION

Teaching Methodology

Many current methodologies to teach programming skills depend heavily on a 'syntax-driven' approach. It is argued by Covington and Benegas (2005) that this approach places undue early emphasis on language syntax and not enough on understanding and problem solving. To avoid this, a 'schema-driven' approach was used for the basic Java programming course. Here the course was designed to encourage the students to spend more time thinking about problem characteristics and problem solving problems patterns (or schema). To implement the schema-driven approach the following steps were taken,

1. Several problems from a similar class of problems were simultaneously introduced.
2. An explanation was always given identifying the similarity of the solution approaches.
3. Introduction of the relevant programming language features for the class of problems being considered.
4. Integration of the solution approaches was then made with the programming language features.

Computer programming can be perceived as dry, so endeavours were made to make the studies and work contributing to the project less formal, more friendly and inviting. This coupled with project milestones, giving presentations on progress and brainstorming sessions, when all students gathered to share experiences and hopefully, find solutions when needed, seemed to move the projects along with less delay than in previous semesters. During the project stage more use was made of the more traditional 'problem-driven approach' as described by Feldman (2005). With this approach, an engineering or scientific problem is presented to the students by the tutor, a mathematical description of the problem is derived in the form as a set of equations and the computational method is introduced and applied to the engineering dynamics problem being considered.

In addition to teaching the skills and knowledge listed in the previous section, two techniques central to the practical use of formal methods in object-orientated programming (Meyer, 1997) were mentored and emphasised during the project process. These were 'design-by-contract', which is a methodology in which programs are written together with their specifications, and the 'command-query separation principle', which allows (limited) reasoning by substitution even in imperative programs. It is a well known that after-the-fact verification offers a practical method of integrating specifications into the program development process and offers immediate benefits in terms of early error detection as well as long-term process improvement (Feldman, 2005). The specifications are given in the form of assertions such as class invariants and method pre- and post-conditions.

This allows the compiler to instrument the program to check these assertions at runtime. More important, it allows the programmer to reason about the program with relative ease, because the assertions provide important intermediate points in proofs of correctness. It was emphasised and demonstrated to students that 'design-by-contract' has particularly strong implications for inheritance in that in order to prevent sub-classes from violating the contracts of their super-classes, it is required that sub-classes only weaken preconditions and strengthen invariants and post-conditions. The tool chosen to implement 'design-by-contract' was written by Man Machine Systems (2000), and has its own scripting language and can only be used on a Windows platform with specific versions of the Java SDK. Some difficulties found, and occasionally inhibiting, were that the correct semantics of 'design-by-contract' were not fully implemented for inheritance and some of the error messages were cryptic.

The students were also mentored on 'command-query separation'. One of the major difficulties in reasoning about programs is the presence of side effects as detailed by Abelson and Sussman (1996). Each change in the contents of variables or data structures creates a new program state. The properties of this state are different from those of previous or subsequent states and explicit reasoning is required to relate successive states. The 'command-query separation' tries to minimize the confusion created through state changes by separating methods into two types, commands and queries. Only the queries are allowed to return information and may not change the program's state.

Survey scope, Description and Results

The survey was in the form of an intranet questionnaire directed at students who have taken the Java programming course and have also used Java for their projects. The questionnaire comprised 40 questions divided into four sections:

1. Learning Java where respondents were presented with a list of broad language topics (e.g. syntax, file handling, GUI) and asked to rate the difficulty of learning each.
2. Teaching techniques where the students were asked to rate the effectiveness of the teaching methods employed.
3. Course evaluation where respondents were asked to assess the course usefulness, how enjoyable it was and what they thought of the balance between theory and practical.
4. Finally the students were asked for suggestions on improvements and any other additional comments they may have.

Students, ($N = 43$), were asked to rate the level of difficulty of Java programming language topics using a scale of 'difficult', 'average' and 'easy'. Arrays, Streams, Objects, JDK and Syntax were considered relatively easy while the topics perceived most difficult were Threading, Files, OOP (object orientated programming) and Exceptions. It can be seen therefore that using and defining

objects was not considered very difficult but the topic OOP, which includes the concepts of inheritance and polymorphism was considered difficult . This means that an 'objects early' approach should not be too difficult for students.

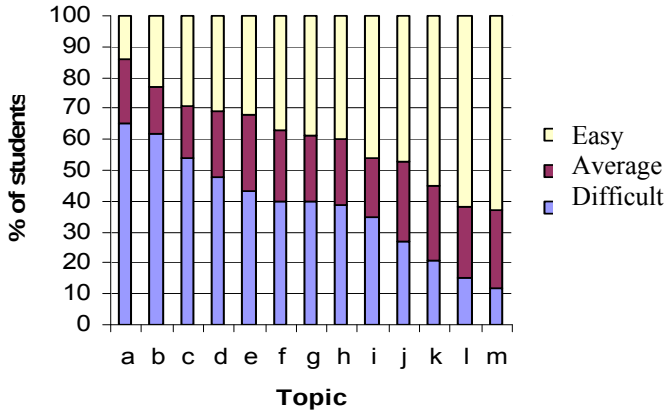

Figure 6: Perceived Difficulty of Java Language Topics.

[a Threads, b Files, c OOP, d Exceptions, e Flow controls, f GUIs, g Recursion, h Applets, i Arrays, j Streams, k Objects, l JDK, m Syntax]

It can be seen from Fig. 6 that objected-based programming was not considered to be very difficult by the students as a whole, though they did find the concepts of object orientated programming reasonably difficult. Using the results of Figure 6 the sequence of the current Java programming language syllabus could be better arranged. Students were also asked to rate the effectiveness of lectures, laboratories, tutorials assignments and informal and extra skills training using a scale of 'useful' and 'not useful'

Table 4: Perceived Effectiveness of Various Teaching Methods.

	Lectures	**Laboratories**	**Tutorials**	**Assignments**	**Extra Skills**
Useful	53.00%	56.25%	65.44%	78.45%	76.25%
Not Useful	34.56%	42.65%	32.11%	8.77%	6.55%
No opinion	12.44%	1.10%	2.45%	12.78%	17.20%

As can be seen from Table 4 the students did not consider the lectures and the laboratories to be all that useful, but as the courses became more problem focused their perception of 'usefulness' grew considerably. It may be that students do not

consider theoretical concepts behind such a 'hands-on' based subject as programming as being all that important.

Finally, students were asked to comment on the BlueJ environment and the more conventional JCreator environment. For BlueJ, 73% said it was easy to use and only 8% found it difficult while 35% found JCreator easy to use and 32% found it difficult. Although the BlueJ development environment appears to be more effective, probably because it is designed for early-stage educational use, it is also true that students in later professional life will meet environments similar to and usually more sophisticated than JCreator.

CONCLUSIONS

It has been demonstrated that, with proper planning and sequencing of topics, it is possible to integrate and hence enhance engineering dynamics modelling courses using programming skills. In general, using the Java programming language as a means of simulation proved to have a high learning curve initially, but eventually provided students with an effective and useful skill. It is thought that most beneficial to the successful outcome of the projects was acquisition of knowledge and skills during the less formal, but in ways more intensive problem solving stage of completing the project. Students benefited from mentoring in 'design-by-contract' methodology and the 'command-query separation principle'. It is clear that students consider programming as a highly 'hands-on' subject.

ACKNOWLEDGEMENTS

The authors acknowledge the contribution of Dr. Julian Dermoudy , School of Computing and Information Systems, University of Tasmania, in designing the Basic Programming and Applications Course.

REFERENCES

Abelson, H., & Sussman, G. J. (1996). *Structure and Interpretation of Computer Programs,* Cambridge, MA.: MIT Press.

Abramowitz, M., & Stegun, I.A. (1964). Handbook of Mathematical Functions, *Applied Mathematics Series*, 55. Washington: National Bureau of Standards.

Barnes, D.J., & Kölling, M. (2008). *Objects First with Java. A Practical Introduction using BlueJ*. New Jersey: Prentice Hall.

Beer, F.P., Johnston, Jr., E.R., & Clausen, W.E. (2007). *Vector Mechanics for Engineers*. San Francisco: McGraw Hill.

Benander, A., Benander, B., & Sang, J. (2004). Factors Related to the Difficulty of Learning to Program in Java - An Empirical Study of Non-Novice Programmers. *Information and Software Technology*, *46*, 99-107.

Covington, R., & Benegas, L. (2005) A Cognitive-Based Approach for Teaching Programming to Computer Science and Engineering Students, *Proceedings of the ASEE Annual Conference and Exposition*.

Eckstein, R. (2007) Java SE Application Design with MVC, java.sun.com.

Feldman, Y.A. (2005). Teaching Quality Object-Orientated Programming. *ACM Journal on Educational Resources in Computing*, *5*(1).

Goldstein, H., Poole, C.P. & Safko, J.L. (2002). *Classical Mechanics*. San Francisco: McGraw Hill.

Harvey, D., & Deitel, P. (2002). *Java How to Program*. Englewood Cliffs: Prentice Hall.

JCreator. (2010). Xinox Software, Delft, Netherlands, www.jcreator.com.

Kilgore, R.A., Kleindorfer, G.B, & Healy, K,J. (1998) The Future of Java-Based Simulation. *Proceedings of the 1998 Winter Simulation Conference, IEEE,* Washington D.C.

Karasözen, B., Rentrop, P., & Wagner, C. (1995) Inverted n-bar Model in Descriptor and in State Space Form. *Mathematical and Computer Modelling of Dynamical Systems, 1*(4), 272-285.

Landau, L.D. & Lifshitz, E.M. (1976) *Course of Theoretical Physics: Mechanics*. Butterworth Heinemann.

Liang, Y.D. (2009). *Introduction to Java Programming, Comprehensive Version*. Englewood Cliffs, NJ: Prentice Hall, 2009.

Madden, M. & Chambers, D. (2002). Evaluation of Student Attitudes to Learning the Java Language. *ACM International Conference Proceedings Series*, 25.

Man Machine Systems (2000). Design by contract tool for Java JMSAssert. mmsindia.com/JMSASSERT.html.

Meehan, A., & Lunney, T. (2001). Java Garbage Collection a Generic Solution? *Information and Software Technology, 43*(2), 151155.

Meyer, B. (1997). *Object-Orientated Software Construction*. Englewood Cliffs, NJ :Prentice Hall.

Mittermeir, R., & Boszormenyi, L.(1997). Choosing Modula3 as Mother Tongue. In Mossembock, H. (Ed.) *Modular Programming Languages, Proc., JMLC'97*, (336-350), Springer Berlin.

Noon, J.P. (Ed.) (1994). OOP in the early courses. Instructors speak out. *Computer Science Product Companion*, *3*(2), 24.

Thimbleby, H. (1999). A Critique of Java. *Software Practice and Experience (UK)*, 457-497.

Tyma, P. (1998). Why are we using Java again? *Communications of the ACM*, 41(6), 38-42.

APPENDIX

Inverted Pendulum Model - Integration of the Motion Equations

To solve Eqns. (6) involves integration, and in general, anything but the simplest differential equations cannot be integrated in closed form. Since the intention is to provide a computer simulation the choice is to use a numerical method of integration. A 4th order Runge-Kutta method (Abramowitz and Stegun, 1964) briefly described below, is used here.

Given $f'(x) = f(x, y)$ the objective is to find $y(x)$. Suppose $y = y_0$ at $x = x_0$, and a small interval h is chosen then the following can be calculated,

$$k_1 = hf(x_0, y_0)$$
$$k_2 = hf(x_0 + h/2, y_0 + k_1/2)$$
$$k_3 = hf(x_0 + h/2, y_0 + k_2/2)$$
$$k_4 = hf(x_0 + h/2, y_0 + k_3)$$

Then at, $x = x_0 + h$, $y = y_0 + (k_1 + 2k_2 + 2k_3 + k_4)/6$

Eqns. 6 are second order, so they need to be written in first order form. Noting that $\dot{x}(t) = \dot{v}(t)$ and $\ddot{\varphi}_1(t) = \dot{\omega}(t)$ four first order equations can be written

$$\begin{aligned}
&\dot{x}(t) = v(t)\\
&\varphi_1(t) = \omega(t)\\
&\dot{v}(t) = \left[2\left(I_1 + m_1 l_1^2\right)\left(F + m_1 l_1 \sin\varphi_1 \omega^2\right) - m_1^2 l_1^2 g \sin 2\varphi_1\right]/\Delta\\
&\dot{\omega}(t) = \left[2(m + m_1) m_1 l_1 g \sin\varphi_1 - 2F m_1 l_1 \cos\varphi_1 - m_1^2 l_1^2 \sin 2\varphi_1 \omega_1^2\right]/\Delta
\end{aligned} \tag{7}$$

where, $\Delta = (m + m_1) I_1 + m_1 l_1^2 (2m + m_1 - m_1 \cos 2\varphi_1)$

The equations are now in the right form for solving with the Java programming language.

Inverted Pendulum Model - Java Programming

The "model-view-controller" or MVC pattern (Eckstein, 2007) was chosen for the Java simulation. This breaks the problem into three classes, i.e. model, view and controller. The model class is responsible for setting the parameters, and simulating the behaviour of the system, with the state of the system tracked and iteration achieved through small time steps.

Within the model class, the following public methods specifically used to solve Eqns. (7) and implement the Runge-Kutta method are,

- setM() - set value of m
- setM1() - set value of m_1
- setL() - set value of l_1
- setF() - set value of F
- getX() - return current x
- getPhi - return current φ_1
- getL() - return value of l_1
- reset() - puts the model in an initial known state
- update(double delTime) moves the model through delTime using the Runge-Kutta method
- toString() - return a String representation of the model state

The view class is used to display the cart and the pendulum bar. Here the view class is a subclass of javax.swing.JPanel and the following public methods are used,

- setModel(Model) - associate the view with a specific model instance
- modelStateChanged() - tell the view to update itself to reflect a new model state

The protected method paintComponent() handles the actual drawing. This method is called indirectly via the this.repaint() in modelStateChanged().

The controller used here is simple with, at regular intervals, the controller updating the model and the view. There is no need to have a separate controller class as the functionality of the controller can be integrated into the JFrame class. The method init() creates an instance of the model class and associates it with the view class and finally starts a timer which calls actionPerformed() periodically. In turn actionPerformed() updates the model and notifies the view that the model state has changed.

Copyright © 2011 IETEC11. Desmond Adair, Martin Jaeger: The authors assign to IETEC11 a non-exclusive licence to use this document for personal use and in courses of instruction provided that the article is used in full and this copyright statement is reproduced. The authors also grant a non-exclusive licence to IETEC11 to publish this document in full on the World Wide Web (prime sites and mirrors) on CD-ROM and in printed form within the IETEC 2011 conference proceedings. Any other usage is prohibited without the express permission of the authors.