Implementing System Dynamics Models in Java

C. Caulfield[†], D. Veal[††], S. P. Maj[†††]

Edith Cowan University, Perth, Western Australia

Summary
For a research project into the value of serious games — games that teach and educate — in software engineering and project management education, a game called Simsoft was developed. Two keys parts of Simsoft were the system dynamics engine that captured the fundamental causal relationships of the software project being modelled; and the Java dashboard through which the players entered their project decisions. Java also provided a means of saving the players individual decisions so these could later be analysed and replayed. While there are currently no Java libraries for implementing system dynamic models, a system dynamics model is simply a collection of non-linear differential equations, and open-source Java libraries for these do exist. Therefore, it is possible to implement a system dynamics model in Java and take advantage of the features of a powerful, general purpose programming language. This paper describes how the model behind Simsoft was created using system dynamics modeling tool called iThink and how the model was subsequently implemented in Java using the Apache Commons Mathematics library.

Key words:
system dynamics, Java, iThink, serious games

1. System Dynamics

1.1 Background and History

In the late 1950s, Jay Forrester of the Sloan School of Management at the Massachusetts Institute of Technology (MIT) was asked by General Electric to review the operations of their Kentucky appliance parts plant. The company was concerned about the oscillating nature of their production cycles that often saw periods of intense activity followed by times of virtual dormancy during which workers had to be laid off. Fluctuating demand and normal business cycles did not seem to adequately explain the situation. Coming from an electrical engineering background and with a keen interest in management science, Forrester approached the problem systematically, but with just a pencil and a note pad. Starting with columns for inventory, employees and orders, and factoring in:

the policies they were following, one could decide how many people would be hired in the following week. This gave a new condition of employment, inventories, and production [1].

Forrester's calculations amounted to a simulation of the system operating at General Electric's plant.

Stemming from this first analysis came an article for the *Harvard Business Review* in 1958 entitled "Industrial Dynamics - A Major Breakthrough for Decision Makers" with the theme being developed and expanded in the seminal work, *Industrial Dynamics* [1, 2]. Industrial dynamics became system dynamics as it came to be used in areas other than industry.

For some time following the publication of *Industrial Dynamics*, system dynamics was used as a tool for looking at big-picture issues such as urban decay, major sociological conditions and world economics [3-5]. In more recent times, system dynamics has been finding a purpose for itself in a range of business and social applications. Instrumental in this change have been Peter Senge's *The Fifth Discipline* [6], and the development of intuitive, graphical software packages that have made system dynamics modelling more accessible by hiding the computer source-code look of traditional models. System dynamics has also found a place for itself in a number of primary, secondary, and tertiary institutions in the United States of America, Australia and Europe, well beyond its ground zero at MIT.

To more formally define system dynamics, it could be said that it:

…is concerned with creating models or representations of real world systems of all kinds and studying their dynamics (or behaviour). In particular, it is concerned with improving (controlling) problematic system behaviour… The purpose in applying System Dynamics is to facilitate understanding of the relationship between the behaviour of the system over time and its underlying structure and strategies/policies/ decision rules [7].

A key element of this definition is the need to build a computer model of the system under consideration. The model is used to help understand the patterns of change or dynamics that a system exhibits over time and to identify the conditions that cause these patterns to be stable or unstable. This knowledge of the system can then suggest what kinds of prescriptions for governing it will work and what kinds may not.

Manuscript received July 5, 2011
Manuscript revised July 20, 2011

However, building system dynamics models demands persistence. Translating real-world information into model elements is still an inexact science - trial and error can be just as valid as considered judgment based on experience. Perhaps a useful parallel can be drawn with that other hard, inexact activity: finding object-oriented classes. Bjarne Stroustrup, the creator of C++, notes that in design and programming there are no cookbook methods that can replace intelligence, experience and good taste; even he just tries things [8]. The lesson for system dynamics modellers would seem to be the same: just start, try things, take advice of experienced modellers and then keep iterating.

Yet the effort of building a system dynamics model has some benefits:

- Modelling brings about an understanding of the system because of the analytical and critical thinking process it calls for. It helps bring to the surface the mental models driving the current situation - those models

  ...that one carries around in one's head for dealing with a problem or situation. Such a model maybe based on experience or intuition, or on folklore and myth; it may be influenced by politics and a wide spectrum of human emotions [9]

  Mental models may also be totally inappropriate or counter-productive, or equally priceless. But unless they are turned into something more tangible, one may never know.

- System dynamics models make room for both quantitative or hard variables— things that can be measured directly like program size, staffing numbers or dollars spent—; and qualitative or soft variables— such as motivation, commitment, confidence or perceptions. Soft variables have traditionally been left out of engineering models because they are difficult to measure and their importance may have been underestimated. Yet,

  ...if you omit soft variables you run the risk of failing to capture something essential to driving human affairs. Leaving out something so essential is the only hypothesis that you can reject with absolute certainty! [10].

  A system dynamics model can therefore be more informed about its problem space.

With a system dynamics model in hand and George Box's tongue-in-cheek caution in mind (all models are wrong, but some are useful), the model can be run. Certain variables can be held steady while others are changed, it can be placed under stress and tested for sensitivities and leverage points. In short, the model can be experimented with to better understand the present situation and to search for alternatives for improvement. It has been stated that:

The alternatives may come from intuitive insights generated during the [initial analysis], from experience of the analyst, from proposals advanced by people in the operating system [or in the] experience, art, and skill for imagining the most creative and powerful policy alternatives [11].

Peter Senge points out that the causes of many problems

...lay in the very well-intentioned policies designed to alleviate them. These problems were actually systems that lured policy makers into interventions that focused on obvious symptoms not underlying causes, which produced short-term benefit but long term malaise, and fostered the need for still more symptomatic interventions [12].

By simulating a problem space using a system dynamics model, it is possible to potentially make more informed decisions about events beyond our bounded rationality safe from the dangers of real-world experimentation.

1.2 Stock and Flow Diagrams

At its lowest level, a system dynamics model looks like computer source code, but even from the earliest days there were graphical representations to help modellers visualise their problem space. The stock-and-flow notation (Fig. 1), first described by Forrester [1], consists of a small number of symbols that together form a grammar telling a story:

- Stocks or levels can be thought of as nouns since they represent an accumulation of something (money, inventory, staff, morale, etc.) at a point in time.
- Flows or rates determine how the stocks will be filled or drained and so are analogous to verbs. Stuff (again money, inventory, staff, morale, etc.) flows through the pipe of the flow in the direction of the arrow and at a rate determined by the flow regulator in the middle. The flow regulator is fitted with a spigot that can be conceptually tightened or loosened by other variables within the model. The cloud at the end of the flow represents the boundary of the model.
- Converters modify flows within the system, just as adverbs modify verbs. They are often used to break

out the detail of the logic that might otherwise be buried within a flow and might be used to represent constant values. Converters typically influence the behaviours of the regulators on the flows.

- Connectors tie the other three building blocks together. They represent inputs and outputs, not inflows and outflows. Connectors do not take on numerical values— they merely transmit values taken on by other building blocks.

Behind these symbols are stored the functions and values (the 'source code' of the model) that drive the simulation and ultimately produce the output. For a system dynamics model, the output is a multi-scale graph (see Fig. 4 later) that shows how certain variables of interest change over time and in relation to each other.

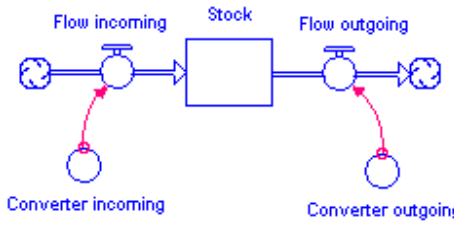

Fig. 1: Stock and flow format of system dynamics models.

2. Basic Mathematics of System Dynamics

The basic mathematics of a system dynamics model is a set of coupled non-linear first-order differential equations [1, 13]. The advance of time is broken into small intervals of equal length (typically called delta time or DT), which is small enough that we can assume change will be constant over that period. For each DT, the model's stocks, rates, converters and auxiliary variables are evaluated to yield a new value, and this value is used as input for continuing calculations.

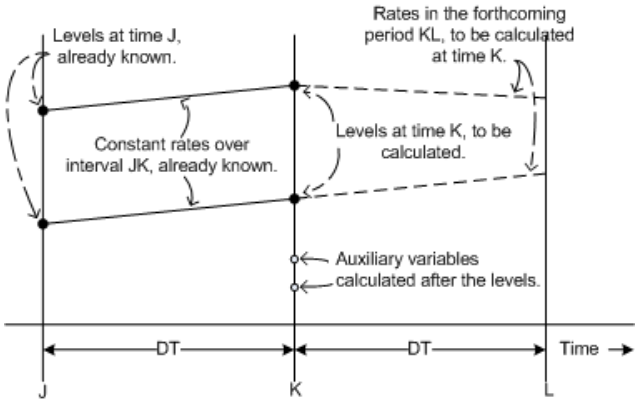

Fig. 2: Calculations at time K

In Fig. 2 [1], J, K, and L represent successive points in time with K being the present. Stock equations are evaluated first and the values are then available for use in the rate equations. Using the simple stock-and-flow diagram in Fig. 1:

$$Stock = \int_0^t (Flow\ incoming - Flow\ outgoing)dt \quad (1)$$

That is, the present value of *Stock* at time *K* is equal to the value of *Stock* at time *J*, plus the difference between the inflow rate and the outflow rate, multiplied by DT.
Flows or rates determine how stocks are filled or depleted. To cater for the delay characteristics of information-feedback systems, a rate equation is given by the outflow rate of a first-order exponential delay. For example:

$$Stock\ at\ time\ L = \frac{Stock\ at\ time\ K}{Converter\ outgoing} \quad (2)$$

There are many other specialised functions available to system dynamic modellers, but those for stocks and rates represent the majority of most models.

3. System Dynamics and Java

3.1 Simsoft

For a research project into the value of serious games as teaching tools for software engineers and software project managers, a game — Simsoft — was developed that had as its engine a system dynamics model. Physically, Simsoft comes in two pieces:

- An A0-sized printed game board around which the players gather to discuss the current state of the project and to consider their next move. The board shows the flow of the game while plastic counters are used to represent the staff of the project. Poker chips represent the team's budget, with which they can purchase more staff, and from which certain game events may draw or reimburse amounts depending on decisions made during the course of the game.
- A simple Java-based dashboard through which the players can see the current and historical state of the project through a series of simple reports, messages, and other information; and can adjust the project's settings, for example to recruit new staff, before advancing the game's time to create the state of the project.

The aim of the game is to complete the project on time and with funds (poker chips) left over. At the start of the game there is a pool of work to do. This pool is represented on

the game board with small plastic counters in the *Work To Do* box. These counters can be thought of as Use Cases or items in a work breakdown structure; whatever is most familiar to the players. Depending on the resources available to do the work, the units of work (the counters) move from the *Work To Do* box to a *For Review* box, where the work is reviewed before passing to the *Completed Work* box. Not unexpectedly, some work will fail the review and go to the *Rework* box, before passing back to *For Review* and trying again to get to *Completed Work*.

The work-to-do, review, rework, work-completed cycle is a fundamental project work structure first discussed and modelled by Roberts [14]. Roberts' initial work has been expanded greatly by subsequent researchers who have added rich details based on actual projects (see [15] for a comprehensive survey of the field), but the underlying work structure remains unchanged.

Of interest here is the Java dashboard and the system dynamics model that implements the work-to-do, review, rework, work-completed cycle. The original design of these two components called for a simple but attractive graphical user interface on top of the stock-and-flow plumbing of the system dynamics model; and a means of capturing the decisions made by the players for later analysis. While there are a number of software packages that can create a graphical user interface for system dynamics models [16-18], some problems were encountered:

- There were limited features for creating attractive, interactive user interfaces.
- All packages required some sort of proprietary software to run the model.
- None provided a means to save the individual decisions of multiple teams in a single database so that the decisions could be later analysed or replayed.
- There is a .NET software development kit allows system dynamics models to be integrated with custom-designed software, but this limits further development and deployment to Windows PCs, plus the initial purchase cost and ongoing licensing fees were relatively expensive.

Java was chosen because it addressed each of the above problems. Even so, there are currently no Java libraries for implementing system dynamic models. But, a system dynamics model is simply a collection of non-linear differential equations, and open-source Java libraries for these do exists, therefore it is possible to implement a system dynamics model in Java.

3.2 Model Design in iThink

Building a system dynamics model by hand-coding equations is time-consuming and error prone. Therefore, the model behind Simsoft was first built and tested using a graphical modelling package called iThink [16]. The final model included almost a hundred stocks, flows and their associated equations, so the aim here is to focus on a small part of the model— that of worker burnout as described by Homer [19].

Burnout begins when a person working on a project (in the case of Simsoft, a software engineer on a development project) tries to meet unmet expectations by working longer hours. By working longer hours they are exposed to more of the normal stress of work and consequently their finite store of "adaptive energy" [20, 21] is depleted more quickly and they also have less time to recover. This depleted energy level may leave the person even less capable of meeting their expectations, or may cause them to make mistakes that have to be fixed at the expense of real progress. In response, they may try to work harder, which will deplete their energy levels still more. Unless the person is granted some respite, this viscous cycle may continue until they are leave in frustration or are they burned out and no longer able to contribute to the project.

Fig. 3 shows how burnout can be modelled in iThink.

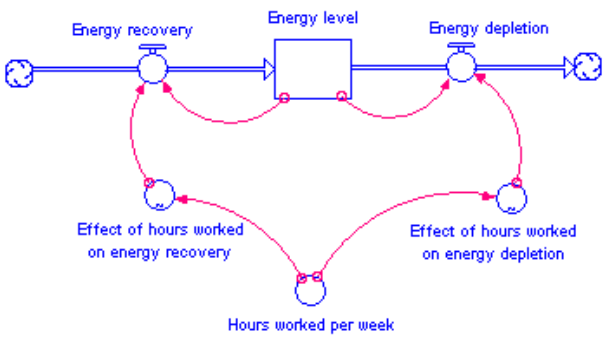

Fig.3: Worker burnout modelled with iThink

Here, a person has a stock of energy available to do work that is depleted or recovered depending on the number of hours they work each week. The effect of hours worked each week on energy recovery and depletion are given in Table 1.

The recovery and depletion rates are nominal values normalised around a 40-hour week. As the number of hours worked each week beyond this point increases, the depletion rate increases; because the person is working

longer days, evenings, or even weekends, there is less time to recover, so the recovery rate slows.

Table 1: Effect of hours worked on energy recovery and depletion

Hours per week	Recovery rate	Depletion rate
0	1.30	0.30
20	1.20	0.60
40	1.10	1.00
60	0.70	1.50
80	0.50	2.00
100	0.35	2.50
120	0.25	3.00

It should be noted that the recovery and depletion rates are known as soft or qualitative variables because they are not based on precise numerical data; such data does not exist [19]. However, the compass of a system dynamics model means that the rules by which it is calibrated and validated will be slightly different from other modelling techniques. For example, the output of a system dynamics model is meant to be read, not for particular time-point predictions, but for qualitative behavioural patterns such as growth, decline, oscillation, stability, and instability [22]. This goal of understanding general dynamic tendencies means that the model's parameters are less reliant on highly precise numerical data:

As long as the purpose of your model is not to predict the numerical magnitude of particular soft variables, you can greatly benefit from including them in your models. Doing so will cause you to think in a rigorous manner about the relationships the variables bear to other variables in the system.[10]

The calibration of soft variables may also seem an arbitrary process in which the model is 'made' to respond in a certain manner. However, the way in which the soft (and hard) variables react must be internally consistent, that is, they must generate behaviour that matches what is observed in the actual system [10].

With this in mind, when this burnout model is run, a multi-scale graph is produced (Fig. 4).

Viewed over a 13-month period, the person starts out by working a 40-hour week. Every couple of weeks there is a spike and they have to work 50-hour weeks for a short time (this pattern can, of course, be changed to model any real-world circumstance). The graph shows that the person's energy levels rise and fall in line with oscillations in the work week, but the overall trend is downwards because the constant spikes in work never allow enough time for proper recovery.

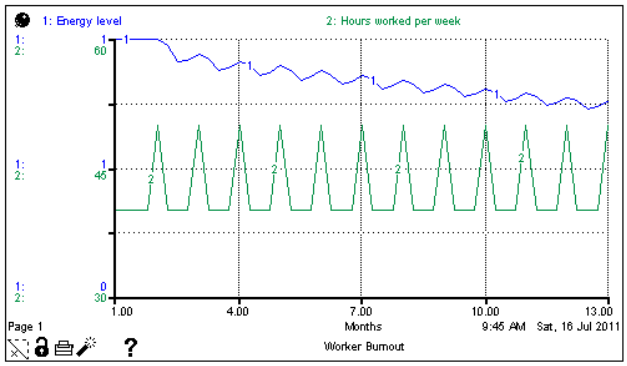

Fig. 4: Worker burnout over a 13-month period

With this portion of the model defined, it only remained to implement it in Java.

3.3 Implementation in Java

The model behind Simsoft was implemented in Java by using the open source Apache Commons Mathematics library [23]. Among its many functions, this library provides a programming interface for solving differential equations.

To implement the system dynamics model shown in Fig. 3, we need to:

- Create a class, *EnergyEquations*, that implements the *FirstOrderDifferentialEquations* interface..
- Pass the class to an integrator to calculate values at different time points. The Apache Commons Mathematics library provides a range of integrator, but for system dynamics models, the Euler or Runge-Kutta methods are most often used.

The key method in *EnergyEquations* is *computeDerivatives*— the one that evaluates the stock equation given at (1).

```
/**
 * Get the current time derivative of the state vector.
 *
 * @param t current value of the independent time
variable
 * @param y array containing the current value of the
state vector
* @param yDot placeholder array where to put the time
derivative of the state vector
*/
     public void computeDerivatives(double t, double[] y,
double[] yDot) throws DerivativeException {

     yDot[0] = y[0] * (recoveryRate - depletionRate);
}
```

EnergyEquations is covered by the following unit test:

```
public void testEnergyEquationsDerivatives() throws
Exception {

FirstOrderIntegrator integrator = new
EulerIntegrator(0.25);
FirstOrderDifferentialEquations energy = new
EnergyEquations(0.7, 1.50);

StepHandler stepHandler = new StepHandler() {
   public void handleStep(StepInterpolator interpolator,
boolean isLast) throws DerivativeException {
     double t = interpolator.getCurrentTime();
     double[] y = interpolator.getInterpolatedState();
     System.out.println(t + "\t" + y[0]);
     }
   public boolean requiresDenseOutput() {return false;}
};
   integrator.setStepHandler(stepHandler);
   integrator.integrate(energy,
         0.75,              // start time
         new double[]{1.0}, // initial stock value
         13,                // end time
        new double[1]);           // storage
}
```

First, an Euler integrator is defined with a step size of 0.25. Then, the EnergyEquations class is constructed and initialised with recovery and depletion rates of 0.7 and 1.50 respectively, being values from Table 1 that equate to a 60-hour week. (An inner StepHandler class is created so we can see the output at each step). Finally, the EnergyEquations instance is passed to the integrator along with the initial conditions of the run. The first data items are shown Table 2.

Table 2: Initial data items from

Time	Energy Level Value
1.00	0.80
1.25	0.64
1.50	0.512
1.75	0.4096
2.00	0. 32768
2.25	0. 26214400000000004
…	

In essence, the same pattern can be followed for all stocks.

4. Conclusions

System dynamics is concerned with building quantitative and qualitative models of complex problem situations and then experimenting with and studying the behaviour of these models over time. Often such models will demonstrate how unappreciated causal relationships, dynamic complexity, and structural delays may lead to counter-intuitive outcomes of less-informed efforts to improve the situation. System dynamic models also make room for soft factors such as burnout so that problem spaces can ultimately be better understood and managed.

These features made system dynamics an obvious choice for creating the model behind Simsoft because the game was trying to demonstrate some of the dynamic complexities of software development projects. However, the means for implementing system dynamic models and integrating them with custom-designed graphical user interfaces and databases are limited.

By using simple open source tools, such as the Apache Commons Mathematics library, it is possible to build system dynamics models that integrate with general purpose programming languages such as Java, meaning the models can draw upon all the features of those languages. For now this integration is largely manual: create the system dynamics model using tools such as iThink and then translate this into a matching class structure in Java. Based on the results presented here, further research is being conducted into ways of automating this translation and being able to perform round-trip translations.

References

[1] J.W. Forrester, Industrial Dynamics, Pegasus Communications, Waltham, 1961.
[2] J.W. Forrester, Harvard Business Review, 36 (1958) 37 - 66.
[3] J.W. Forrester, Urban Dynamics, Productivity Press, Portland, 1969.
[4] J.W. Forrester, World Dynamics, Productivity Press, Portland, 1971.
[5] D.H. Meadows, D.L. Meadows, J. Randers, W.W. Behrens, The Limits to Growth: A Report for the Club of Rome's Project on the Predicament of Mankind, Earth Island Ltd, London, 1972.
[6] P.M. Senge, The Fifth Discipline: The Art & Practice of The Learning Organization, Revised edition ed., Random House Business Books, London, 2006.
[7] E.F. Wolstenholme, System Enquiry: A System Dynamics Approach, John Wiley & Sons, Brisbane, 1990.
[8] B. Stroustrup, The C++ Programming Language, special edition ed., Addison-Wesley, Boston, 2000.
[9] E. Yourdon, Rise and Resurrection of the American Programmer, Prentice-Hall, Sydney, 1998.
[10] B. Richmond, Modelling "Soft" Variables, in: An Introduction to Systems Thinking, High Performance Systems, Hanover, 1999, pp. 9-1 - 9-10.
[11] J.W. Forrester, System Dynamics Review, 10 (1994) 245 - 256.
[12] P.M. Senge, The Fifth Discipline: The Art & Practice of The Learning Organization, Random House, Milsons Point, 1990.
[13] A. Ford, Modeling the Environment: An Introduction to System Dynamics Modeling of Environmental Systems, Island Press, Washington, 1999.
[14] E.B. Roberts, The Dynamics of Research and Development, Harper & Row, New York, 1964.
[15] J.M. Lyneis, D.N. Ford, System Dynamics Review, 23 (2007) 157 – 189.
[16] isee Systems (http://www.iseesystems.com/), 2011. iThink version 9.1.4.

[17] Ventana Systems (http://www.vensim.com/), 2011. Vensim version 5.
[18] Powersim (http://www.powersim.com/), 2011. Powersim version 8.
[19] J.B. Homer, System Dynamics Review, 1 (1985) 42 - 62.
[20] H. Selye, Stress Without Distress, Signet Books, Philadelphia, 1974.
[21] H. Selye, The Stress of Life, 2nd edition ed., McGraw-Hill, New York, 1978.
[22] D.H. Meadows, J.M. Robinson, The Electronic Oracle: Computer Models and Social Decisions, John Wiley & Sons, New York, 1985.
[23] Apache Commons Mathematics Library (http://commons.apache.org/math/), 2011. Version 2.2.

Craig Caulfield is a senior software engineer for a technology consulting company and PhD candidate at Edith Cowan University. His research areas include problem-based learning and the application of serious games to software engineering education and project planning.

Dr. David Veal is a Senior Lecturer at Edith Cowan University. He is the manager of Cisco Network Academy Program at Edith Cowan University and be a unit coordinator of all Cisco network technology units. His research interests are in Graphical User Interface for the visually handicapped and also computer network modeling.

A/Prof S. P. Maj has been highly successful in linking applied research with curriculum development. In 2000 he was nominated ECU University Research Leader of the Year award He was awarded an ECU Vice-Chancellor's Excellence in Teaching Award in 2002, and again in 2009. He received a National Carrick Citation in 2006 for "the development of world class curriculum and the design and implementation of associated world-class network teaching laboratories".