Teaching Formal Methods in Software Engineering

Gavriel Yarmish, Brooklyn College
Danny Kopec, Brooklyn College
Jim Aman, Xavier University

Innovations in Engineering Education I

Abstract

Today we live in a world where computers and software are ubiquitous. In effect they run our lives -- with applications in industry, business, education, finance and a lot more. In the short history of computers and software there have been several known near catastrophes including the Three Mile Island Nuclear Power Station, the North American Air Defense System (NORAD), Air Traffic Control and airplane accidents, The London Ambulance System just to name a few.

Many methods within the framework of Software Engineering have been developed to facilitate both the programming and management of these systems. Some are general rules of thumb while others are more formal and rigorous.

In general Software Engineering courses have focused less on formal methods and more on general concepts. We describe a course that incorporates more formal methods without excluding the more typical general concepts. These will be highlighted by presentation of case studies illustrating human errors and general good design and maintenance methods to follow. A significant focus of the course, though, will be formal methods.

The course will include three major components:
1) The continued deep study and documentation of complex systems failures, particularly those which involve computer software.
2) The study of software development methods and tools which have been (or may be) used to design and develop complex systems (e.g. CMM, Jelinski-Miranda, McCabe Complexity Measure) and
3) A group project involving the construction of a software system using the tools studied (above) for the purpose of design, diagnosis, security and prevention of failures in complex systems, e.g. in medical information systems, transportation systems, etc.

Introduction

With the new millennium upon us, it is clear that the world is dedicated to technological advancement. There is no turning back to the relatively slower life of yesteryear. It is often heard that the number and impact of technological advances in a recent decade (for example the 1990's) exceeds the total number of technological advances in the previous century. Nonetheless,

the fundamental question which perpetually confronts us is:

"**Despite the tremendous and rapid technological advances of the past century, do we live in a safer world?"**

Traditional measures of safety would suggest that we do indeed live in a safer world. For example, if average life-span is a measure of safety then we can rest assured that the world is safer than a century ago. If safety is measured in terms of, for example, the likely success of surgical or medical procedures, or the safety of travel via car or plane, then certainly we would conclude that life is safer today. We can travel more miles, faster, over a longer period of time today, without an accident than in anytime before in history. Jerome Lederer (1982) suggests that safety be measured in terms of fatalities per 100,000 hours of exposure, and with this measure concedes that highway travel would be the safest mode of travel.

Software has become ubiquitous. It affects nearly every aspect of our lives, yet nowhere is there a control agency which can regulate every conceivable application of software. There are too many possible applications and permutations of software -- especially when one considers embedded systems as well. Software complexity today is immense. Software systems employed to control and fly a plane, a satellite, a space-orbiting vehicle, or a nuclear power station, are very difficult to evaluate in terms of safety.

In computer science it is known that there is a *crossover point* where an Algorithm's *utility* versus its *theoretical complexity* will be bounded. That is, although the algorithm may in theory grow more quickly (e.g. an algorithm whose growth rate is sublinear is known to eventually grow more quickly than an algorithm whose growth rate is logarithmic) the domain space of the algorithm (i.e. the range of values where the most abundant data will occur) will never (or rarely) occur in practice. This illustrates a concept known as "The Practicality Window" (Rawlins, 1992). Referring to the case of sublinear algorithms, where N (the number of data items) is ~~less~~ more than one million; the algorithm with sublinear growth rate will grow more quickly, both in theory and in practice. ~~((Question: isn't a logarithmic algorithm a subset of asublinear programs?))~~ However, in practice, cases whereby N equals or exceeds one million may occur only relatively infrequently. This depends on the application area of the algorithm. Hence given that within the Practicality Window the sublinear algorithm will indeed grow more slowly than the logarithmic growth algorithm, the sublinear growth algorithm may be preferred.

Significance: If we view a piece of software as a type of algorithm to achieve some task in a given application domain, the implications may be similar: there will be problem spaces in the task domain where the software system's performance cannot be tested. Yet, with the increasing reliance of society on software such "unknown and untested" spaces will continue to increase. This demands careful, continuous, investigation.

Course Design

Most courses in Software Engineering today concentrate on the various standard topics such as requirements, specification, design, and numerous methods for developing software systems as distinct from smaller programs which students may write to develop their programming skills. We believe that it is time that theory should be joined more intrinsically with practice. In this way a software engineering course would address recent trends suggesting the need to effectively measure software. Our course would be a nice blend of practical case studies, tools to develop software, as well as practical and theoretical methods for software development.

Our course would include three major components:

1) The continued deep study and documentation of complex systems failures, particularly those which involve computer software.

2) The study of software development methods and tools which have been (or may be) used to prevent complex systems failures and

3) A group project involving the construction of a software system using the tools studied (above) for the purpose of design, diagnosis, security and prevention of failures in complex systems. Examples include medical information systems, transportation systems amongst others.

Component 1) Kopec has been investigating the general area of complex systems failures for a number of years (Kopec & Michie, 1982), (Kopec, 1983), and (Kopec, 1990). Some of the systems which have been studied in recent years in addition to the earlier mentioned TMI, NORAD, Air Traffic Control, and Royal Dutch Steel (Hoogovens) cases, include: The Case of KAL Flight 007 (1983), The Bhopal Chemical Plant Disaster (1984), The Space Shuttle Disaster (1986), The Chernobyl Nuclear Power Station (1986), The Therac-25 Radiation Therapy Machine (1986-87), The Estonian Ferry Disaster (1994), The London Ambulance Service (1994), TWA FL-800, SWISSAIR FL-111, The Y2K Problem. I hope to continue to my investigation and documentation of these accidents particularly as they relate to software and systems failures in general with my research assistant(s). It is hoped that the net result of these investigations would be a theory (or theories) which would help in understanding when and how complex systems fail (particularly those involving software), in addition to the theories of Leveson (1995) Perrow (1999).

Component 2) During recent years a number of expert system shells and systems modeling tools have been developed. Some examples are diagnostic systems like MYCIN, INTERNIST, INTERPRET, SOPHIE, which have been applied to such diverse domains as medicine, mechanics, and electrical circuits (Jackson, 1999). In addition several system modeling tools such as STELLA, UML, and RATIONAL ROSE have been developed. STELLA is a tool developed by High Performance Systems for developing Models of Dynamic Systems. It was developed in an attempt to resolve the so called "method wars" as users sought the perfect object

modeling language. The Unified Modeling Language (UML) is graphical language for visualizing, specifying, constructing and documenting the artifacts of software-intensive system (Booch, 2000). RATIONAL ROSE is one of major object modeling languages developed by the highly successful Rational Corporation. We will investigate these in general and the latter two in particular with respect to their facilitating the design of complex object oriented systems

Component 3) In this component students will put together the theory and experience derived from the first two components. They will embark on the design of a safe software system for a socially critical task. First, prototype systems will be designed. Several real application domains under consideration are medical information systems, modeling and relief of highway traffic, and practical development of software watermarking.

Based on these ideas we provide one possible syllabus:

Syllabus: Ubiquitous Computing

Overview of Ubiquitous Computing: Kopec & Michie: FAST Report Complex Systems Failures

- TMI-2
- Air Traffic Control (ATC)
- North American Air Defense System (NORAD)

Normal Accidents (Charles Perrow): CASES

- The ingredients of a "Normal Accident"
- TMI/nuclear power in detail
- Chernobyl, Therac-25
- Seveso, Bhopal
- The Space Shuttle; Air Traffic Control

Design for Safety: Lessons Learned

- The Role of Computers in Accidents
- Seven Software Myths

Computer-Related-Risks (Neumann)

- Software peculiarities
- Determining causality
- Realities we face

Systems Engineering and Safety: Foundations

- Systems Theory
- Systems Engineering
- Systems Analysis

More Recent Cases (systems)

- ROYAL DUTCH STEEL
- Industrial,
- Medical,

- Financial

Fundamentals of Systems Safety
- History, Basic Concepts
- Software Systems Safety
- Cost and Effectiveness
- Novel Approaches

Software Metrics: Software Measurement Techniques
- McCabe's Complexity Measurement
- Goals, Questions, Metrics
- Capability Maturity Model

Specification Techniques
- Data Flow Diagrams
- Finite State Machines
- Petri Nets

Systems Development Methods
- Agile Development
- Object Oriented (UML)
- Dynamic Systems Modeling(STELLA)

Formal Methods
- Modularization Techniques
  - Textual Design Notation (TDN)
  - Graphical Design Notation (GDA)
- Verification: Testing
  - Empirical
  - Structural Testing (White-Box)
  - Statement Coverage
  - Edge Coverage
  - Condition Coverage
  - Black Box Testing
  - Logic Specification
  - Boundaries, in the Large
  - Correctness Proofs
- Validation

Requirements Engineering: More UML, STELLA

Analysis Modeling: Course Group Projects

Design Engineering: Course Group Projects

References

Booch, G, (2000) UML in Action. CACM. Vol. 42, No. 10, October, 1999, pp16-18.

Kemeny, John, et al. (1979) The Need for Change: The Legacy of TMI. Report of the President's Commission on the Accident at Three Mile Island. Washington, D.C.: Government Printing Office**.**

Kopec, D, Michie, (1982) *Mismatch between machine representations and human concepts:* dangers *and remedies***.** Report to the EEC under subprogram FAST (Forecasting and Assessment in Science and Technology), Brussels, Belgium**.**

Kopec, D (1983) Human and Machine Representations of Knowledge, Ph.D. Thesis, University of Edinburgh.

Kopec, D (1990) Technology Transfer Crises in the 1980's: mishaps at the man-machine interface, (1990). In Proceedings of the 15th Annual Meeting of the Technology Transfer Society (June 26-28) Dayton, Ohio, Technology Transfer in a Global Economy, ed. Robert W. Harrison pp. 173-76.

Kopec, D, Jiang, Q (1992) The Societal and Technological Problems of Computers. Computers and Artificial Intelligence Slovak Technical Institute, Bratislava, CFSR Vol. 11, No.4, pp. 409-418.

Lederer, J. (1982), Aviation Safety Perspectives: Hindsight, Insight, Foresight, New York: The Wings Club.

Relevant Texts

Jackson, P. (1999) Introduction to Expert Systems, (3rd.), Addison Wesley Publishing Company, Reading, MA, 542 pages.

Leveson, N. (1995) SAFEWARE, Addison Wesley Publishing Company, Reading, MA, 680 pages.

Neumann, P. (1995) COMPUTER-RELATED RISKS, ACM Press New York, 367 pages.

Perrow, C. (1999) NORMAL ACCIDENTS, (2nd ed.), Princeton University Press, Princeton, NJ, 451 pages.

Fenton, N, S. L. Pfleeger, Software Metrics A Rigorous & Practical Approach, Second Edition, PWS Publishing Company, 1997.

~~Fowler, M. (2003?) UML Distelled (2nd ed.) Addison-Wesley.~~
Fowler, M. UML distilled : a brief guide to the standard object modeling language. 3rd ed. Addison-Wesley, 2004.

Pressman, R. (2005) Software Engineering: A Practitioner's Approach, (6th ed) MacGraw Hill, NY 880 pages.

Authors biography

Gavriel Yarmish currently teaches at Brooklyn College...He got his PhD

Danny Kopec…..??

Jim Amen is currently an Associate Professor of Computer Science at Saint Xavier University in Chicago and is Program Director for the Masters of Applied Computer Science in Internet Information Systems.

Saint Xavier University
3700 West 103rd St.
Chicago, IL 60655
773-298-3454
aman@sxu.edu