Evolution of Software and Research Infrastructures: A Report from the Trenches

John Mylopoulos
University of Trento

ECSS 2011, Milano ITALY
November 8, 2011
Abstract

The first part of this presentation focuses on the main objectives and preliminary results of the Lucretius project (ERC advanced grant, 2011-16), whose aim is to develop concepts, techniques and tools for supporting software evolution. Evolution may involve adaptation of a system to a new behaviour that better copes with the system's operational environment, or mutation of a system to deal with unforeseen requirements (e.g., derived from a new regulation). Moreover, evolution may be automatic (e.g., self-adaptation), or manual, or something in-between.

The second half of this presentation focuses on academic research funding infrastructures in Europe, both indicative (as-they-are) and optative (as-they-ought-to-be), from the perspective of a researcher who recently moved from Canada to Europe (Italy).

Based partly on material from [Lucretius11].
I. Software evolution

- Known hard problem since the very beginning of Software Engineering (SE).
- Laws of software evolution put forward a theory of how software (code) evolves over its lifetime [Lehman80]
  e.g., “The complexity of a program increases, unless … “
- Software Reengineering, Software Maintenance study aspects of software evolution, and constitute important areas within SE -- largely code-focused.
Requirements-driven software evolution

We want to develop design principles for software that evolves gracefully, in response to changes to its requirements and/or its domain (= operational environment).

The principles need to address questions such as

What are the primitive concepts in terms of which we conceive and design such software?

What runtime support (with/without humans in the loop) do we need to enact/facilitate evolution?

Basic premise: Requirements models constitute (part of) the hereditary material (DNA) of software. Throughout its lifecycle, a software system consists of a requirements model and code that operationalizes the requirements (OK, and other things ...).
The Requirements problem (J&Z)

In its original formulation [Jackson95], a requirements problem (RP) consists of finding a specification $S$ for a given set of requirements $R$ and indicative environment properties $E$ such that

$$E, S \models R$$

meaning: “… satisfaction of the requirements can be deduced from satisfaction of the specification, together with the environment properties…” [Jackson95]

*Behaviours* are implementations of specifications.

We prefer a formulation where environment properties are replaced by *domain assumptions* ($D$) and inference is replaced by entailment

$$D, S \models R$$
Evolution mechanisms

- **Adaptation**: In response to changes in the environment, system changes its behaviour – but not its requirements -- at runtime in order to continue to fulfill its requirements.
  
  Change: $D \rightarrow D'$, Problem: $D', S'(S) \models R$
  
  E.g., flight control system changes automatic->manual mode

- **Mutation**: System changes its specification in response to changes to its requirements.
  
  Change: $R \rightarrow R'$, Problem: $D, S'(S) \models R'$
  
  E.g., new requirements added because of a new law.

- **Composition (Reproduction)**: Requirements models of two systems are composed to produce a new requirements model within the same software “species”.

  ...
Requirements problem spaces

At the heart of every evolution there is a changed requirements problem.

Consequences:

- Need to characterize requirements problem spaces
- Need efficient algorithms for searching such spaces with different search criteria, e.g., (i) new solution should be best wrt new RP, (ii) new solution should be as similar to old as possible.
Requirements as goals (GORE)

Requirements are stakeholder goals and (requirements) problem solving amounts to incremental goal refinement.
Goal problem spaces

Here, specifications consist of tasks, domain assumptions and quality constraints that together satisfy requirements, e.g., for G:ScheduleMtg, one specification is \{T:Collect, T;Schedule, D:RoomsAv, QC: ‘>70% participation’\}

Unlike J&Z, goal refinement generates a space of possible specifications (problem space) and the requirements problem amounts to finding those that satisfy R, assuming D.

The GORE version of the requirements problem can be reduced to SAT solving [Sebastiani04].

This algorithm constitutes the backbone of goal-oriented requirements problem solving.
Control variables and indicators (CV&I)

Success of a behaviour to fulfill its requirements depends on **control variables** and degree of success is measured by **indicators** (gauge/quality variables).

FhM - From how many?  
RfM - # of Rooms for Meetings

Success Rate

Schedule meeting

Choose schedule

Good quality mtg

Rooms available

Collect timetables

Collect

Find free room

Get free room

By person

By system

By system

>70% participation

©2011 John Mylopoulos
The CV&I version of the RP

Control variables and indicators determine resp. resource allocation, quality of performance.

The requirements problem (CV&I) consists now of finding a solution that maximizes/does well enough with respect to one or multiple indicators.

Now we have an optimization problem that seems to lie somewhere between OR-style optimization and SAT solving ...

See [Letier11] for work along this direction.
The CV&I version of the RP

- Control variables and indicators determine resp. resource allocation, quality of performance.
- The requirements problem (CV&I) consists now of finding a solution that maximizes/does well enough with respect to one or multiple indicators.
- Now we have an optimization problem that seems to lie somewhere between OR-style optimization and SAT solving ...
- See [Letier11] for work along this direction.
Controlling CV&I software systems

To control a CV&I system, we need to add information about how control variables influence indicators

e.g., \[ F(FhM,RfM,\text{SuccessRate},\text{CostPerMtg}) = 0 \]
\[ G(FhM,RfM,\text{SuccessRate},\text{CostPerMtg}) \geq 0 \]

Such constraints -- in the form of differential equations and the like -- constitute the backbone of Control Theory.
Dealing with unknown unknowns (MT)

Suppose now we have implemented several specifications and a running (=old) solution, and a requirement changes ...

- Get manager approval
  - OR cp0
  - OR

- Collect timetables
  - OR cp1
  - OR

- By person

- By system

- Rooms available

- Schedule meeting
  - AND
  - AND
  - AND
  - AND

- Find free room
  - OR cp2
  - OR

- Get free room

- Choose schedule
  - OR cp3
  - OR

- By person

- By system

- Good quality mtg

- >70% participation
The incremental requirements problem

- All we need to do is run our GORE/CV&I/... algorithm for solving the requirements problem, right? ...

- Not quite, if we want to:
  - Maximize familiarity – use as much as possible of the old solution (user perspective)
  - Minimize effort – minimize the number of tasks that need to be implemented (vendor perspective)

- We seem to need algorithms here that use a “minimum repair” principle to find new solutions.
- [Ernst11] studies a class of such algorithms using AI Truth Maintenance Systems.
II. Research funding infrastructures

Context: Recent move from Canada to Europe (Italy), after 39 years of academic research activity in Canada.

Canadian research funding model (Science & Engineering)

Funding programmes

- Basic research: About 50% (*) of proposals funded, with small grants (€15K-75K/yr), no overhead applied.
- Industry-University funding has different models.

Features:

- Inclusiveness
- Continuity in funding
- Stability of programmes, rules
- Low overhead (researcher time, decision period, ...)
- Strictly peer-reviewing system
European realities

- Funding of basic research (+)
- Inclusiveness (-/+)
- Continuity (-)
- Stability (--) [after factoring Italian context (--*)]
- Overhead (--), for application, administration and deliverables.
- Peer reviewing (+/-), factors other than peer-reviewing play an important role in some funding decisions.
Conclusions

- There is room for streamlining/rationalizing research funding in the EU.
- There is also room for more emphasis on
  - Basic research
  - Core CS areas, as opposed to fringe/emerging ones
  - Better interfaces with national funding programmes and rules
... In summary

- We have sketched a requirements-based approach to the design of software evolution mechanisms.
- We have also taken stock of EU research funding infrastructures, with Canadian glasses on...
References


References (cont’d)


References (cont’d)


