Some issues in the engineering of widely deployed software intensive systems-functional variation

by Assoc. Prof. Karl Reed,FACS, FIE-Aust., MSc,ARMIT

Chair IEEE-Computer Society Tech. Council on Software Engineering
Governor, IEEE-Computer Society(1997-1999,2000-2002),
Director, Computer Sys. & Software Engineering Board, ACS,
Department of Computer Science & Computer Engineering, La Trobe University

The Problem..

1. **We have a system consisting of a collection of interacting (statically or dynamically) bound components in which either a single component or sub-system can be replaced with another…**
2. **If the functionality represented by the replaced "part" is changed,**
   - **A/What mechanisms can be provided to allow this to be propagated to the user level?**
   - **B/ How shall we (or should we) control the functionality-variation at the user level?**
   - **(Normally I'd be arguing that we need systems stability, and that this is a measure of quality???)**

Why are surprises important?

-Will customers pay for software intensive systems if they continue to be difficult to use

If not, there'll be no research funding!!

Normalised Time to Breakeven-No. learning times T/T_c	Productivity Increase Required
1	50%
1.5	30%
2.00	20%
2.5	15%
3.33	10%
7.00	5%

Agenda

1. Dynamically "varying" systems of (autonomous) dynamically linked components can lead to variant *functional* of *non-functional* behavioural variation.

2. If *functional* variation is allowed, the ability to somehow control it, and determine what is both ***acceptable*** and ***allowable*** is needed

3. **In terms of what is *acceptable*,** we can make use of a number of properties of real systems (Shaw 1999) where the internal states/transitions are quite "fuzzy", and where a user actually accepts a range of outcomes (hence, varying functionality). In fact, humans may not work with precise systems often.

   -Also, humans work with large systems whose **apparent ambiguity** (Reed, 2000) can appear as *functional* variation

4. **What is *allowable*** .. An (functional) operational envelope could be defined in principal. Functionality out-side this envelope could be rejected- BUT RECORDED, AND PRESENTED TO THE USER, WHO COULD ADD IT TO THEIR OPERATIONAL ENVELOPE (or a systems administrator could)

5. Pt.4. Constitutes a "controlled mutant adoption" process.

6. Pt.3. May involve a kind of reverse HCI,or a human-centred fault tolerant approach.

Surprise(we already deal with functional variation)….!!! *(nsf report on s/w research 1998)*

"F1. Current software has too many surprises. The sources of surprise are poorly understood."

F2. Key sources of software surprise include immature or poorly integrated software domain sciences, construction (product) principles, and engineering processes.

Sources of surprises... Real and apparent ambiguity in the means of representation of systems, e.i. Languages *(cf 3 pages of c++ with 3 pages of government regulations)*

Real and apparent unpredictability in behaviour...

"Teenagers have less trouble with PC software because they are adept at playing computer games" *Charles Wright, editor Melbourne Age "green pages" computer section 2000*

"Building 'bots' that play computer games with near human competence is not that hard" US researcher in AI….

LARGE-SCALE (UBIQUITOUS) SOFTWARE SYSTEMS CANNOT HAVE TO MANY 'SURPRISES'

WHAT ARE SURPRISES --- WHO KNOWSABOUT THEM AND WHAT CAN WE DO ABOUT THEM?

what are they?

-A ***"surprise"*** *(for our purposes) is some behaviour of a system's which causes or could cause a user to make an error*[1]*, or excessive stress and discomfort in resolving the behaviour*

-Occur inherently in Lehmann's E-type systems

-Occur WHEN developers BELIEVE they are building E-type systems

Examples..*introduction of new expense claim s/w suddenly impacts the work-loads, stress and financial security of 100's of people*

…no logical relationship between functions in s/w and semantics of menus they are in

..almost un-usable web-sites

..SCS system failures are better known

-INCLUDE some un-expected functionality (that requires some effort to interpret?)

WHAT ARE SURPRISES --- WHO KNOWSABOUT THEM AND WHAT CAN WE DO ABOUT THEM?

Who knows about them?

-The SCS community places great effort on identifying "unexpected behaviours"and controlling their impact.

-Those working on systems with adaptive behaviour and user-error recovery.

-Fault-tolerant community (already been mentioned)

-Games technologists (one of the Fraunhofer Institut's has looked at this)

-Extreme Programmers and iterative developers "think" they are dealing with "surprises"

-Product-line strategists

-lateral thinkers (here, the "surprise" is an unrecognised "use" which allows functional substitution

What can we do about them?

-Elevate their definition and management to a high-level design feature rather than an implementation problem to be avoided

-Those working on systems with adaptive behaviour and user-error recovery.

-Study the behaviour of people working in/with systems whose behaviour changes

-at the design level- deal with the adoption and control of autonomous functional variation

LA TROBE UNIVERSITY

adoption of autonomous functional variation of due to some type of component "change"

Where functionality has been removed..

-Maybe analogous to a fault-tolerant situation (however, the system could actively seek a replacement component)

-User may agree that they can manage without this (operational envelope).

-User may seek-out a replacement function--rejecting the original

-Fault-tolerant community (already been mentioned)

-Games technologists (one of the Fraunhofer Institut's has looked at this)

-"Grace-full degradation" of OS in the 70's..

Where functionality has been changed

-Need to define "conformant" and "non-conformant" changes, e.g., a data representational change

-Need to recognise unacceptable (I.e. unadoptable at the system level) change (op-envelope issue)

-Can we define functionality extractors-correctors? (recognise and correct a functional miss-match?- a re-use - "glue-code" problem?)

LA TROBE UNIVERSITY

Dealing with changing function..

Formal approaches….

-Component-contract specifications specify..

1. What semantic variation in service-component that can be tolerated

2. Semantic definition of service's actual functionality

-Semantic reasoning about aggregated functionality, propagating the changes upwards until either they reaches the user, OR violate some semantic constraint.

Informal Approaches.. (may be an operational approach?)

-Examine examples of functional variation visible to users,

-Using these, develop rules for specifying functional variations at the component level, and for their transmission upwards (an operational approach to the formal)

Suggest there is actually a lot of data and examples to examine..

e.g. behaviour of pc-desk top s/w applications present to the user as having unpredictable changes in functionality

relationship between knowledge, experience,skill and tools is relevant

Operational Envelope approach…

Assumes mechanisms for adopting functional variation and propagating it through a design

An (functional) operational envelope could be defined in principal. Functionality out-side this envelope could be rejected- BUT RECORDED, AND PRESENTED TO THE USER, WHO COULD ADD IT TO THEIR OPERATIONAL ENVELOPE (or a systems administrator could)

Constitutes a “controlled mutant adoption” process.

May involve a kind of reverse HCI,or a human-centred fault tolerant approach.

Use approaches from security (although of conceptual value only)--perhaps consider form of intrusion (e.g. audit trail monitoring)

Adaptive user profile generation could be part of this

Needs..

mechanisms for describing the added functionality to users and administrators

roll-back mechanisms

Research Agenda (one component)

1. Study the way humans work with large systems with **apparent ambiguity**

2. Develop the operational envelope approach..

- Develop a “controlled mutant adoption” process.