Assumptions in Software Development and Evolution

IDI
Oslo
16 March 2006
Meir (Manny) Lehman
School of Computing
Middlesex University
The Burroughs]
London NW4 4BE, UK
mml@mdx.ac.uk
http://www.cs.mdx.ac.uk/staffpages/mml

Abstract: A brief summary of the results of many years of study of $E$-type system evolution leads to a focus on the inevitability and crucial role of assumptions and their potential impact. The societal and economic consequences of failure in this respect clearly indicates that their adequate management is increasingly vital.
**Facts of Life**

- RAEng/BCS report quoted study predicting UK 2003/4 software expenditure as £B22.6 - that of other countries proportional as modified by state of economy

- Long known that 50% to 95% of E-type software life cycle costs incurred after introduction into use, that is on maintenance and evolution

- Hence UK expenditure on software evolution that year between £B11.3 and £B21.5 - probably ≥ £B15

- Both levels growing as problem and environmental complexity, inter-system integration, communication, functionality and data detail all increasing

- Human and indirect costs in addition

*Effective planning, management, control of software evolution a major challenge but can yield significant returns*
Opportunities for Process Improvement

• Major potential benefit from development and application of discipline, methods and tools for evolution planning and management

• Potential UK saving from improvement of only 10% could yield up to £b2 benefit annually on direct software costs alone

• Human, social, economic saving potential many times that

First, brief historical overview of basic observations
Review of Continuing Software Evolution Studies

- 1968 - 69  IBM Programming Process study the seed
- 1969 - 71  First recognition of the role of feedback
- 1969 - 02  Collection, analysis of empirical data on evolution of industry-developed systems
- 1974 - 86  Laws of Software Evolution
- 1979      SPE program classification schema
- 1988      Key role of assumptions
- 1989      Principle of Software Uncertainty
- 1996 - 01  FEAST/1 and /2 projects
- 2004      SPE+

• Has yielded significant elements of a theory of software evolution with feedback key mechanism, assumptions key factor

• Basic observation: evolution, particularly of software, inherent to real world use of computers


Historically, laws were first step in formulation of theory
**Origin of Laws**

- Derived from behavioural **observation** and empirical **data**

- Subject to **extension** and **generalisation**

- **Relationships** between them not investigated

- Eighth, **feedback**, law suggests that first seven reflect feedback properties

- Hence, **feedback-systems** theory likely contributor to software evolution theory

---

**Today's focus Assumptions:** presence **in** and **effect** on real world **in** E-type computing
Computing in the Real World

- **E-type software, systems, applications** - address **problems, activities** and/or **operate** in real world **domains**

- System **acceptability** depends on stakeholder **satisfaction** with **results** of and **behaviour** in execution

- **Correctness** second level issue

- In fact, **not meaningful** in precise - mathematical - sense

**Implication:** E-type systems can never be **guaranteed fault free** - that is entirely **satisfactory**, utterly **reliable**
**Fundamental Requirement**

- **E-type** computer application a multilevel, multi-loop feedback system adapting system to external change by driving and controlling software change

- For results to be optimum system must reflect relevant application and domain properties to level of precision, detail as required at each moment

- **Must** continue to hold as domains, application properties, expectations change

- Such change inevitable as result of installation, use, external - real world - changes

- Software must be continually adapted to continue to reflect all relevant properties and needs so as to maintain stakeholder satisfaction,

**Evolution inescapable if useful service is to be maintained**
Application and Domain Changes

• Keep system in tune with real world states, needs, opportunities, ambitions

• Without system (software) evolution there would be inevitable decline in functionality, validity, performance, utility

• In practice evolution also exploited to increase these

Software evolution a process that maintains E-type systems satisfactory
**Human Limitations**

- Properties of $E$-type domains and applications **unbounded** in number

- **Human knowledge, understanding, mastery** is bounded hence relevance of all properties cannot be **determined, considered, even known**

- **Every** property not recognised or addressed implies **assumption of irrelevance**, that non-reflected properties are **irrelevant** for satisfactory execution

- **Unbounded** number of application, domain properties consciously or unconsciously **excluded** during design, implementation, evolution by **abstraction** process

Every $E$-type program reflects **unbounded** number of **assumptions**
Uncertainty

• Inescapable element of arbitrariness in determination of system properties

• Completeness and correctness of software artefact is essentially uncertain from conception

• In addition, real world is dynamic, its properties appear, change, disappear

• A set accepted as satisfactory at some point, level of detail and precision may, possibly unknowingly, cease being so as a consequences of such changes

Principle of Software Uncertainty follows
Principle of Software Uncertainty

However often a system has executed satisfactorily, satisfaction on its next execution is uncertain, cannot be guaranteed.
Source of Uncertainty

• One way or another external changes will continually invalidate assumption set

• Hypothesis: assumptions that are or become invalid are major source of project or computer failure during development and use

• Inevitably, some will remain unknown, even unsuspected, until they cause incorrect results, misbehaviour or worse

Examples
Examples of Initial or Eventual Invalid Assumptions that Led to Major Problems

- **CERN Accelerator** - possibly unconscious assumptions, valid during development, eventually became invalid

- **London Ambulance System** - invalid initial design assumption

- First launch of **Ariane V** - earlier conscious and valid operational application and domain (hardware) assumptions became invalid

- Sinking of the **Sheffield (18 lives)** - initially valid, predictably invalid, application assumption

Daily examples on Google's **Alerts for Software Failure** provide major source of material for further **investigation**

17 March 2006
**True Meaning of "Software Maintenance"**

- **Maintenance** as used in other areas refers to **restoration** of subject to its state of **pristine beauty** or as near as possible thereto

- Entirely inappropriate for software which **does not** in itself **deteriorate** from wear, tear or **other** causes

- Decline in **satisfaction** is due to external changes in **needs, application** or **domain**, reflected in invalidated assumptions

- System must be **evolved** to correctly **address** and **satisfy** changed or new needs

**What is maintained is validity of assumption set, stakeholder satisfaction**
Assumptions and their Implications

• Embedding of assumptions in system during abstraction, reification, bounding intrinsic to E-type computing system development, evolution, usage

• Assumptions may be explicit or implicit, conscious or unconscious, by commission or omission, recorded or unrecorded, (initially) valid or invalid

• Real world changes during use cause invalidation of assumptions

• Essential to search and review, at intervals determined by system criticality, to ensure safe, reliable, efficient, controlled, that is satisfactory, computer usage

Assumption management vital but neglected development, maintenance and evolution responsibility
Assumptions Management

Good practice requires one to:-

- Identify, capture, structure, document, update, continually review and revalidate rationale, assumptions and decisions underlying them

- Improve questioning of assumptions by, for example, using independent specification, implementation, validation teams

- Periodic and, event-triggered reviews to update assumption set, its validity and evaluation of its exposure to and likelihood of change

- Extend process to make search for and questioning of assumptions in reviews, inspections, validations etc. at each step a specific and required activity

- Develop and use tool support for all of the above

In general: search for, examine, control assumptions in all activities
Assumption Management, Control

Determine, develop and, where possible, formalise method and tool support for:

- Assumption management and control at all stages of development, evolution

- Identify, question assumptions in problem and requirements definition, architecture, design, review, implementations, inspections, validation etc.

- Whatever process is followed, all activities must include disciplined examination of possible impact on assumptions and their potential consequences

Examples in specific process activities
Other Areas Ripe for R & D, Industrial Application

- Development of strategies, techniques to ensure timely changes in advance of failure

- Likelihood analysis of changes and their impact on application and domains

- Structured, indexed, mechanised data-base system for assumption management

- Periodic and event-triggered reviews to update assumption sets at all stages of system definition, development, evolution

- Search for guidelines to minimise likelihood of assumption invalidation, maximise structural flexibility, increase changeability, reduce cost of system changes

Exemplify further by considering requirements
Requirements

Must explicitly address and rule on issues such as:
• Explicit and implicit assumptions relating to each requirement throughout process

• Likelihood and potential impact of changes that will impact either

• Continually determine timings for review of all aspects of system to ensure timely changes, forestall undesirable behaviour

• Linkage of architecture, process, design, code, documentation, procedures, etc to requirements to simplify change and minimise assumption dependent errors

• Full structured, indexed documentation to facilitate organised, timely searches for missing or invalidated assumption statements

General conclusions
Last Word: Theory of Software Evolution

• Concepts and interpretations outlined provide base and framework for development of theory of software evolution

• Its development non-trivial but development could have major impact

• Could provide direction and discipline to conception, development, integration of methods and tools, improvement of process for software evolution

Powerful tool to reduce cost and threat implied by need for continual change and evolution with major technical, economic and social implications in a society ever more critically reliant on software
Relevant Papers

Complete listing provided at http://www.cs.mdx.ac.uk/staffpages/mml/

Rules, Tools and Guidelines


Theory


Process Modelling


Estimation


Lessons Learnt (e.g., Modelling Methodology)


Component-based and Integration-intensive Processes


Other


Bibliography - Others

A first listing with additional papers and other material is provided at and some may be accessed via http://www.cs.mdx.ac.uk/staffpages/mml/
Needs to be updated

id., A Spiral Model of Software Development and Enhancement, Computer, v. 21, May 1988, pp. 61 - 72
Rajlich VT and Bennet KH, A Staged Model for the Software Life Cycle, Computer, July 2000, pp. 66 - 71
Wirth N, Program Development by Stepwise Refinement, CACM, v.14, n.4, Apr 1971, pp.221-227
In Summary

- Must recognise inevitability of evolution and accept need for related social, economic, operational effort and expenditure

- Relate general and specific behavioural patterns and characteristics to application areas, domains, processes, organisation, cultures etc.

- Develop and apply understanding of evolution to support development of processes, methods, tools for, for example, its and assumption management
Reservations

• Observations, hypotheses, generalisations such as laws and principle all based on data from conventional industrial processes

• Some studies of Open Source processes appear to suggest other growth patterns but support conclusion that software evolution is predictable phenomenon

• Not aware of studies of evolutionary behaviour of systems developed using other approaches such as Object Oriented, Open Source, Agile programming

• But universal and process independent domain concepts, behaviours, in particular feedback, assumptions, suggest that phenomenon is universal - independent in principle, though not necessarily detail, of application, domain, process, degree of system integration, development organisation and so on

Urgent need for widespread R & D to determine evolution characteristics as function of these and other variables
## Laws of Software Evolution

<table>
<thead>
<tr>
<th>No.</th>
<th>Brief Name</th>
<th>Law</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Continuing Change</td>
<td>Unless continually <strong>adapted</strong>, an E-type system must decline in use and become ever more difficult to <strong>maintain satisfactory</strong></td>
</tr>
<tr>
<td>II</td>
<td>Increasing Complexity</td>
<td>As an E-type system is evolved its <strong>complexity increases</strong> unless work is done to maintain or reduce it</td>
</tr>
<tr>
<td>III</td>
<td>Self-regulation</td>
<td>Global E-type system <strong>evolution</strong> is feedback <strong>regulated</strong></td>
</tr>
<tr>
<td>IV</td>
<td>Conservation of Organisational Stability</td>
<td>Rate of <strong>work</strong> of organisation evolving E-type software tends to be <strong>constant</strong> over phases of the <strong>operational lifetime</strong> of system</td>
</tr>
<tr>
<td>V</td>
<td>Conservation of Familiarity</td>
<td><strong>Growth rate trend</strong> of E-type systems constrained by need to maintain familiarity</td>
</tr>
<tr>
<td>VI</td>
<td>Continuing Growth</td>
<td><strong>Functional capability</strong> of E-type systems must be continually enhanced to maintain user satisfaction</td>
</tr>
<tr>
<td>VII</td>
<td>Declining Quality</td>
<td><strong>Quality</strong> of E-type systems declines unless rigorously <strong>evolved</strong> to take into account changes in the operational environment</td>
</tr>
<tr>
<td>VIII</td>
<td>Feedback System</td>
<td>E-type evolution processes are <strong>multi-level, multi-loop, multi-agent feedback</strong> systems</td>
</tr>
</tbody>
</table>

*Derived from **observation** and **metrics**

Provides base for empirical **Theory of Software Evolution**
Example of E-type Evolution

• **First** observation

  ![Graph showing size in modules over release sequence numbers (RSN)](chart)

  - Size in modules over release sequence numbers (RSN)

  - Underlying **linear growth** with superimposed **ripple** to rsn 19

  - Ripple, indication of **feedback stabilisation**

  - Instability and break-up from rsn 20 triggered by excessive **positive feedback** preceding growth rate decline observed in other systems

  - Initial **faster than linear** growth
Further Example

• Large **current** telephone switch - 44M instructions 4 years ago

![Graph showing size relative to RSN 1 over RSN 17]

- Understanding has gone through three stages, **linear**, **inverse square**, **S-curve** growth patterns
- Similar faster than linear **initial** growth and sustained **ripple**
- Consistently **similar** evolution patterns observed on other systems

*Suggests empirical generalisations such as: Laws, Principle of Software Uncertainty, \[-growth\] (S-curve)*