

Software Evolution (Part I)

Towards a Theory of the Phenomenon

STRL Annual Distinguished Lecture
De Montfort University, Leicester, U.K.
20 Dec 2001

M M Lehman

Dept. of Computing
Imperial College
180 Queen's Gate
London SW7 2BZ
mml@doc.ic.ac.uk
<http://www.doc.ic.ac.uk/~mml>

J F Ramil

Dept. of Computing
Faculty of Maths and Computing
The Open University
Walton Hall, Milton Keynes MK7 6AA, U.K.
j.f.ramil@open.ac.uk
<http://mcs.open.ac.uk/jfr46/>

Introduction

- **Past** - some highlights
 - 1968: IBM software **process** study - including **empirical data analysis**
 - 1969: **The Programming Process** report
 - 1971: First **discussion** of software process as **feedback system**
 - 1974-86: **Laws** of Software Evolution
 - 1979: **SPE-type program classification schema**
 - 1989: Principle of **Software Uncertainty**
 - 1993: **FEAST** hypothesis
 - 1996 to 2001: **FEAST/1** and **/2** projects
- Assumes software evolution is **phenomenon worthy of study** and **can** be studied
 - supported by results
- **Future**
 - **explore** FEAST concepts, results in context of **open source, component, COTS** based systems
 - further **develop operational system dynamics models** of **global** software processes
 - initiate **development** of **formal theory** of software process evolution
- **Observations** over 30 year period support generality of **continuous evolution** of **E-type** systems in distinction to **S-type** systems, **static by definition**

What are these software types?

S-type Systems

Definition and Properties

- Program with properties **formally specified** in **abstract, closed, isolated** domain
- **Problem, program, solutions: fully formally specified**
- Criterion of **acceptability** is **verified correctness** relative to specification
- **Unsatisfactory**, that is **incorrect** or **inappropriate** behaviour/results in execution after verification implies **unsatisfactory specification** -
- System **validation** a matter of **opinion** or **judgement**, implies **assumptions**
 - a **beauty contest**, with beauty **in the eyes of the beholder**
- Fixing a program found to be **unsatisfactory** in use requires change to **specification** followed by derivation of **new program** - **new** by **definition** even when obtained by modification of original code
- **S-type** programs: **bricks** for construction of software systems

Programs in the **real world**

E-type Systems

- **Solve problem, address application** in **real world**
- Definition of *E*-type **applications** follows
- *E*-type software is **model-like reflection** of **real world** of **interest** and must remain so as both the **real world** and **software change**
- **Acceptability criterion** for *E*-type systems is **satisfaction** of **stakeholder needs**
 - in contrast to *S*-type systems where **correctness** is **criterion**
- **Correctness meaningless** in **real world context**
- **Behaviour** during and **results** of **execution** determine **acceptability**
- **Overwhelming number** of applications and programs in **regular use** of type *E*
- **Intrinsic** property of *E*-type systems is that they must be **continually evolved** to remain **satisfactory**

What is software **evolution** and **why is inevitable?**

General Definition of "Evolution"

- **Process** of **progressive change** over **time** in **characteristics**, **attributes**, **properties** of some **material** or **abstract**, **natural** or **artificial**, **entity** or **system** or of a **sequence** of these
- Changes are **progressive** when they result in **definable trend** of, for example, **increasing value**, **growing precision** or **better fit** to a changing **domain** or **context**
- **Entities** include **objects** or **collections** of objects (e.g., **population**) such as **natural species**, **societies**, **cities**, **artefacts**, **concepts**, **theories**, **ideas** or **systems** of these
- **Change process** will, in general, be **continual** with relatively **slow** rate of change, or **discrete** with **individual incremental** changes, **small** relative to entity as a whole

Further discussion limited to **software context**

Concerns Arising from Study of Evolution in Software Context

- The **what** and **why** - "**evolution as a noun**" - studies its **nature** and that of evolution **process** and its **products**
 - **what** evolves?
 - **why** does it evolve?
 - what **drives, controls, constrains** evolution process?
 - do **process** and its **product**, the **evolving system**, have identifiable **attributes** or **patterns**?
 - what is **impact** of the above on **cost, effectiveness, timeliness, value** of evolution **processes** and their **products**?
 - etc., etc
- The **how** - "**evolution as a verb - to evolve**" - considers **implementation** of evolution process
 - what is **goal** of evolution activity?
 - **where** or which **part** must be evolved to attain **goal**?
 - what **methods, techniques** are likely to be **appropriate, effective**?
 - what **tools** would be **appropriate, effective**?
 - what **resources, skills** are required?
 - etc., etc.
- Each is important
 - **how** reflects, for example, **interests** of **business world**
 - mastery of **what** and **why** critical for **disciplined, directed how**
 - **without** that understanding, **how technology advances** tend to be **ad hoc**

Areas of software evolution

Areas of Evolution in E-type Software Context

I System **ab initio** development, fixing, enhancement, adaptation, extension, **Quality** - code and documentation levels

II **Sequence of releases** or **versions** of a software system over its lifecycle

III **Applications**

IV **Operational domain** of an application - as it impacts relevant evolution process

V **Interacting domains** - e.g. **organisational, market place, technology**

VI **Software** (and **systems**) evolution **processes** as:
I - **system development**, II - **release planning and management**

VII Software process **models**

- A **simplified structure**

Our work has focused on a **sub-area** of II
Evolution of **E-type systems** in active use

The Focus

- **Sequence** of **releases** of a software system
- **Satisfies** stated **definition** of evolution, **evolving** as discrete **sequence of entities**
- Evolution **concepts** and **phenomenological detail** emerged over thirty years from **analysis, modelling, interpretation** of **empirical data**
- Data related to number of **industrially developed** and **evolved** systems from **variety** of **application, evolution** environments
- Study provides **compelling empirical evidence** that the *why* and the *what* can be **usefully** studied, revealing **intrinsic** properties of evolution **phenomenon**

Some **conclusions**

High Level View of FEAST Results

- **Evolution characteristics** and **behavioural patterns** of release-based systems studied, **qualitatively similar** to those observed in the 70s
- **Support** for at least 6 of the 8 **laws of software evolution** as modified over the years
- **Extended** and **new insights** into and **conclusions** about **process**
- Study of **influential** feedback **loops** in long-term **evolution context** and **assessment** of their **impact**
- Black box and system dynamics **models** to support **process planning, management, control, tuning** and **improvement**
- Some 35 **management** and **planning** rules and guidelines
- **Empirical generalisations**
- **Inputs** to **formal theory of software evolution**

Relationships between **areas** of software **evolution?**

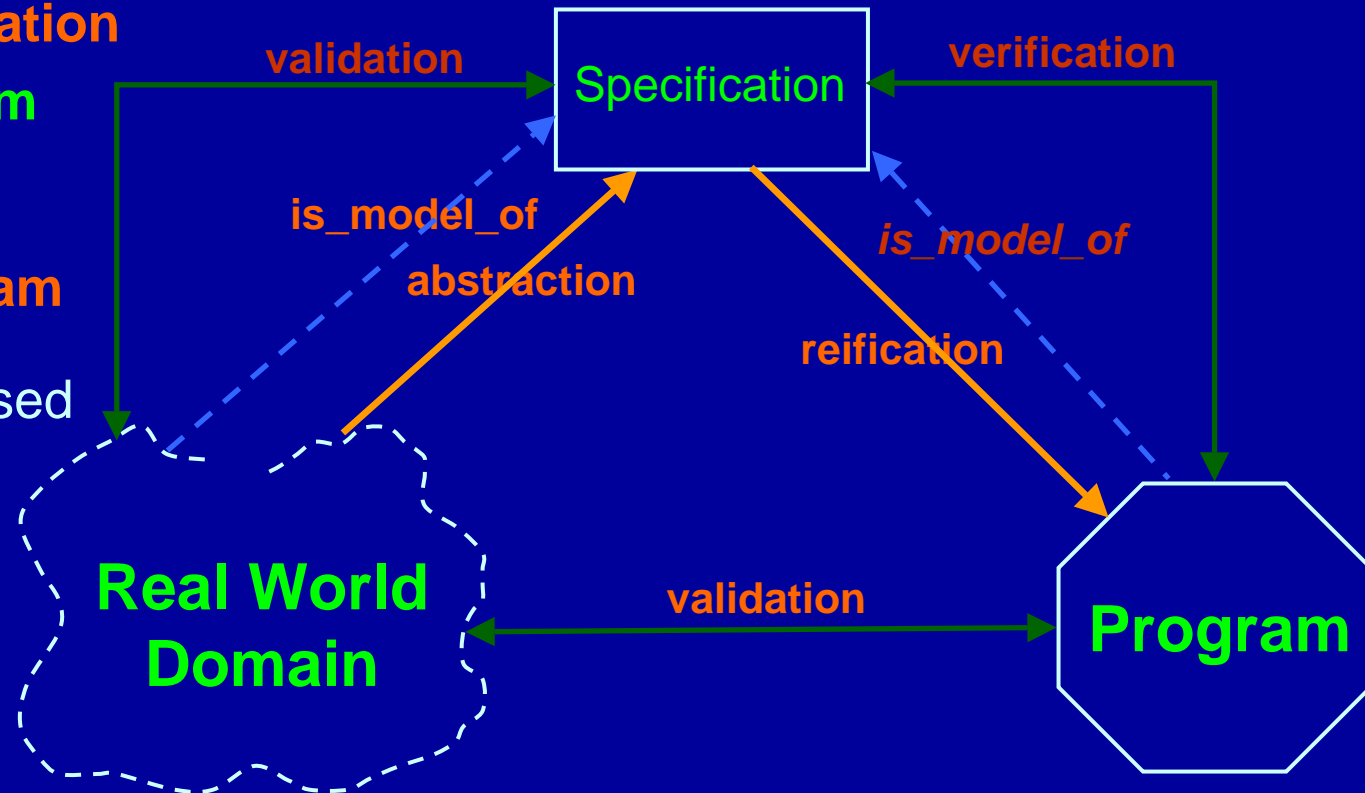
Some Further Observations

- Have introduced seven **areas of evolution** in **software context**
- List presented a **simplification** in that there are **sub areas, relationships, linkages, interdependencies** between them
- **Alternative** breakdowns, perhaps based on other criteria, are certainly **possible**
- **FEAST** studies **focussed** on **area II** evolution and extent to which it's **identified attributes** are shared by other areas **not known**
- On the contrary, **reasoning** indicates that even within software field, **stark contrast**, for example, between **area I, II** evolution on the one hand and **area VII** on the other

How does this relate to **software** in the **real world?**

Programs in the Real World

- **Real world operational domain** is **model** of a **specification** derived by process of **abstraction**
- **Specification** must be **validated**
- **Program** reached by a process of **reification** also **model** of **specification**
- **Verification** of a **program** may be possible in **whole** or in **part**
- Both **domain** and **program** have, in general, **properties not** addressed by **specification**
- **Specification, program** must be **valid** relative to **real world**



The real world - **E-type system** relationship

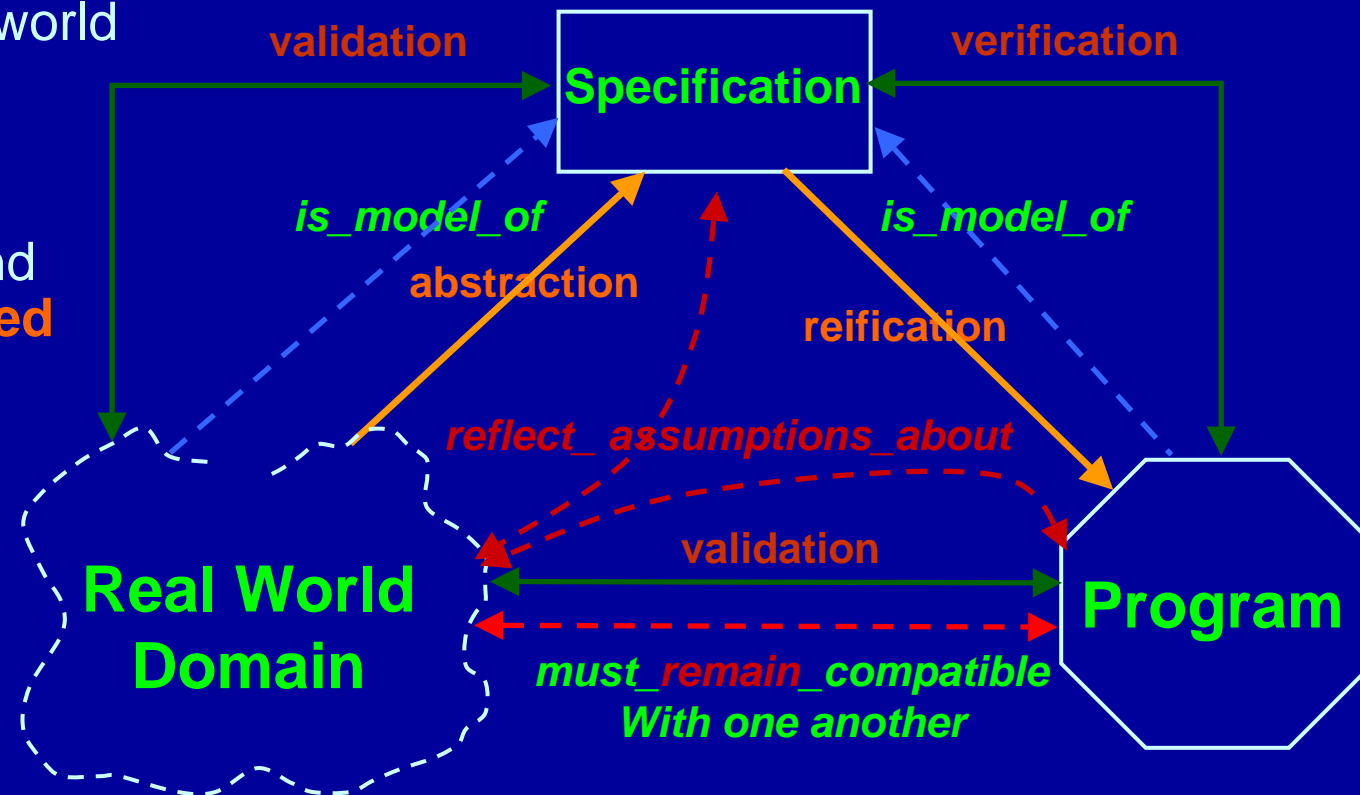
Some of Properties of the

- Universe with its **unbounded** number of properties is ultimate **operational domain** of *E*-type systems
- As **sub-domain** of **universe**, **actual operational domain** also has **unbounded** number of **properties**
- **Application** is equally **unbounded** in number of its **potential properties**
- **System** is **finite**

Gap between **finite** *E*-type system and **unbounded operational domains** bridged by **assumptions**

Crucial Role of Assumptions

- Assumptions **reflected** in specification and program accepted as **valid reflections** of real world properties
- Must be **captured** and periodically **reviewed**
- Program maintained **compatible** with **real world** over system life time



Assumptions and, hence, **program** must reflect real world as it is **now**

Adoption of Assumptions

- Adopted **throughout global** development and usage **processes**
 - by**
 - corporate management
 - local management
 - marketeers
 - vendors
 - users at all levels
 - etc., etc.
 - during**
 - conception
 - verbalisation
 - requirements statements
 - specification
 - designs at all levels
 - validation at all levels
 - integration
 - release
 - installation
 - usage
 - etc., etc.
- Adoption by **commission** or **omission**, **explicit** or **implicit**, **conscious** or **unconscious**, **recorded** or **unrecorded**
- Individual assumptions **become invalid** as **changes** occur in **operational domains**, **application**, **technology** etc.
- Software **maintenance maintains validity** of **assumption set** and, thereby, **stakeholder satisfaction**

Such maintenance at **core** of system **evolution process**

E-type System Evolution Process

- **Ab initio development** or **change** for whatever reason

Trigger - **Need/Demand/Opportunity** not satisfied by current system



Preliminary, generally broad, **statement** of required **system** or **change**



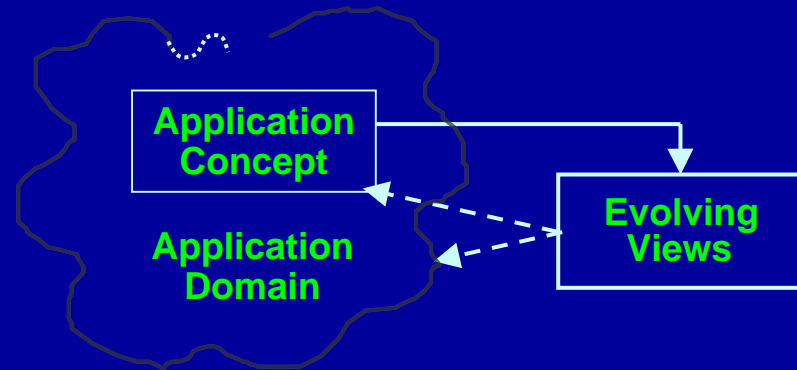
Yields - **new** or **changed application concept** and/or **domain**

- In general, **both concept** and **domains** initially:
 - **ill defined**
 - **not** fully or precisely **verbalised**
 - **not** explicitly **bounded**
- **Authoritative statement of purpose** provides **base** for process evolving proposed change, providing, *inter alia*, initial functional and **domain bounds** for **application**
- **Bounding** a **critical** and **continuing** process **activity**

Continuing revision throughout **process, system lifetime**

Bounding Process

- **Elicit, reconcile, merge, limit** restricted, biased views of individual stake holders:
 - all levels of **management**
 - organisational and individual, direct or indirect **users**
 - **marketeers, suppliers, procurement** personnel
 - domains and application **experts**
 - system, software **engineers, developers**
 - etc., etc.
- **Process** blends **viewpoints**, determines **broad goals**, agrees initial **assumptions**



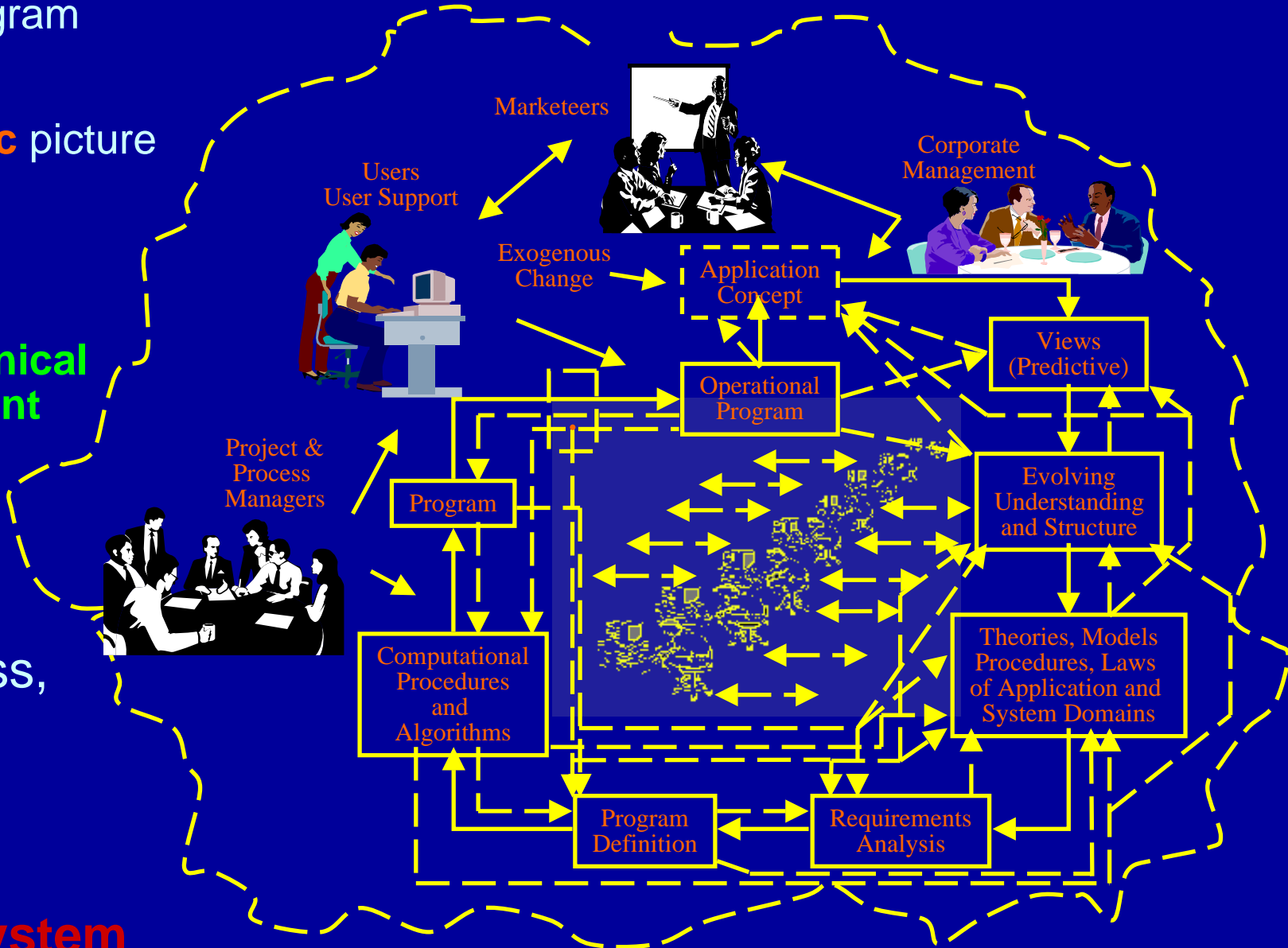
- **Iterative convergence** to consensus changes **application, bounds**

First step in *E*-type **evolution** - **development** and **maintenance** - **process**

More Realistic Picture

- Previous diagram a **fiction**
- More **realistic** picture
- Process **not sequential**
- Not just **technical development**

Global process, in general, a **multi-level multi-loop multi-agent feedback system**



The Global Process

- **Aggregated** effect of **all activities** that **transform concepts, ideas, needs, resources** into executable **code** and other **deliverables**
- Involves many **activities, groups, people** with diverse **responsibilities**
- **Complex, non-linear**, possibly **time varying**, loop **structure**
- Satisfies engineering **definition** of **feedback** - *output from part of system or process modifies one or more of its inputs*
- All these **factors** influence **process & product properties & behaviour** and must be considered when **planning, executing, controlling** evolution
- These and other **observations encapsulated** in **generalisations** relating to **software evolution**

Laws of software evolution

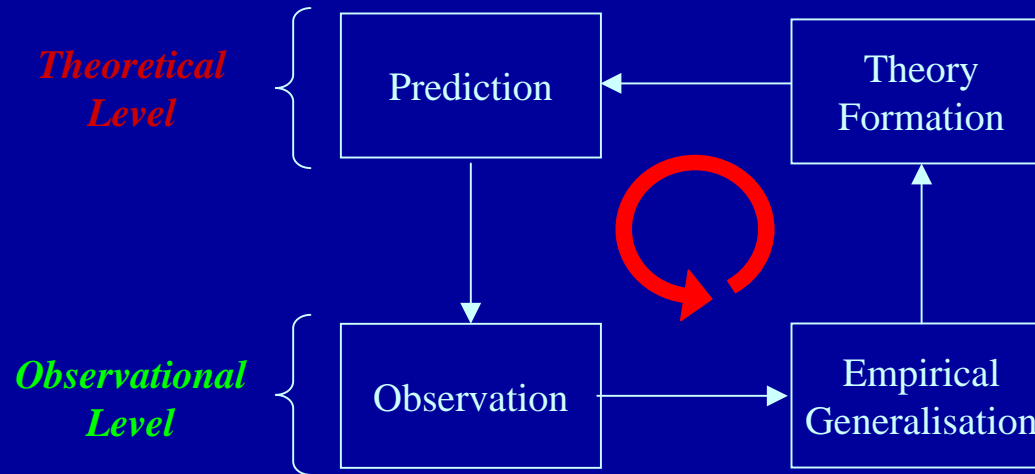
The Laws and Their Observational Base

No.	Brief Name	Law	Support
I 1974	Continuing Change	An <i>E</i> -type system must be continually adapted else it becomes progressively less satisfactory in use	yes
II 1974	Increasing Complexity	As an <i>E</i> -type system is evolved its complexity increases unless work is done to maintain or reduce it	yes (indirect)
III 1974	Self Regulation	Global <i>E</i> -type system evolution processes are self-regulating	yes
IV 1978	Conservation of Organisational Stability	Average activity rate in an <i>E</i> -type process tends to remain constant over system lifetime or segments of that lifetime	yes
V 1978	Conservation of Familiarity	In general, the average incremental growth (growth rate trend) of <i>E</i> -type systems tends to decline	yes
VI 1991	Continuing Growth	The functional capability of <i>E</i> -type systems must be continually enhanced to maintain user satisfaction over system lifetime	yes
VII 1996	Declining Quality	Unless rigorously adapted to take into account changes in the operational environment, the quality of an <i>E</i> -type system will appear to be declining as it is evolved	theory based
VIII 1996	Feedback System (Recognised 1971, formulated 1996)	<i>E</i> -type evolution processes are multi-level, multi-loop, multi-agent feedback systems	yes (indirect)

The laws as part of a **Theory of Software Evolution**

Development of a Formal Theory

- Based in **observations**, **behavioural invariants**, **wider phenomenology**
- Provides basis for **formation** of informally expressed **observational level** theory
- As a set of derived **empirical generalisations**



- From **axioms suggested** by the **empirical generalisations** one derives a **theoretical level** theory stated in a **formal notation**
- **Predict behaviours** on basis of **emerging theory** and **validated** against **observations**
- **Iterative extension**, yields **theorems** - further **generalisations** and corresponding **axioms** may also emerge from **observations** suggested by **evolving theory**

SETh proposal to **initiate theory development** submitted

Formal Theory of Software Evolution

- Previous slide outlined **two level approach** to general **theory formation**
- FEAST **observation, modelling, interpretation** led to identification of **behavioural invariants, mathematical models** and eventually **empirical generalisations**
- **Empirical generalisations** provide foundations for application of **approach** to development of **formal theory of software evolution**
- **Exploratory steps** indicate **feasibility** of **observation level theory**
- Every **reason** to **believe** that **theory level theory** may be derived
- Current status: **awaiting funding**

The SETH proposal

Outline Example

- **Intuitive definitions**
 - **initial formulation**
 - require **refinement** and **precision** via **formalisation**
- Identify some **observations**
 - Seek to reflect in **empirical generalisations**
 - Provide basis for **axioms** in **formal theory**
- Propose possible **inferences**
 - derived from current **models** and **interpretations**
 - basis for potential **theorems** in **formal theory**
- Sketch out **approach** to one **proof**
- **Full proofs** await **formalisation**

Example

Definitions

- **Definition 1:** *Real world* encompasses **entire universe** and all **happenings** in it
- **Definition 2:** An *operational domain* is a **real world domain** with **attributes** a **sub-set** of the attributes of the **unbounded set** of **real world attributes**
- **Definition 3:** An *E-type program* is one **solving problem** or **implementing application** in an **operational domain**
- **Definition 4:** A *specification* is an **abstraction** of an **operational domain** that is believed **valid** and **sufficient** at some **point** in **time**
- **Definition 5:** An *E-type program* **implements** an **application** in its **operational domain**
- **Definition 6:** To be **satisfactory** an **E-type program** must **reflect** all **required** attributes of the operational domain and **none incompatible** with it
- **Definition 7:** *Validation* is a **process** demonstrating that an **E-type program** can be accepted as **satisfactory**

Observations

- **Observation 1:** The real world is **dynamic** and its **attributes** are **changing** continually
- **Observation 2:** It may be partitioned in an **unbounded** number of ways into domains that, in general, each possess an **unbounded** number of **attributes**
- **Observation 3:** Attributes of an *E*-type **program** are an **abstraction** of the **attribute sets** of the **operational domain** and the **application** to be supported
- **Observation 4:** The attribute set of an *E*-type **program** as such (as distinct from such programs in execution) is necessarily **finite**
- **Observation 5:** **Attributes** of **application** in its **operational domain** that make it **satisfactory** must be reflected in *E*-type **program** implementing it

Potential Inferences at the Observation Level

- **Inference 1:** **Assumptions** underlying a **specification** or **reflected** in **program** may become **invalid** so **invalidating** program
- **Inference 2:** **Application domain** from which specification of an *E*-type program is abstracted has **unbounded** number of properties
- **Inference 3:** The **number** of **distinct behavioural properties** of an *E*-type program implied by its specification is **finite**
- **Inference 4:** An *E*-type program is **essentially incomplete** since it **cannot** reflect an **unbounded** number of real world properties
- **Inference 5:** Though **models** of the same **specification**, a **problem/application** in its operational **domain** may become **incompatible** with its **implementation**
- **Inference 6:** **Outcome of execution of an** *E*-type program in its **operational domain** entails a degree of **uncertainty**, outcome of execution **cannot be absolutely predicted**

Outline proof of Principle of Software Uncertainty

The FEAST Project as Scientific Endeavour

- Triggered by **observation** of **empirical data** - in early seventies
- **Patterns** and **trends** recognition
- **Empirical Generalisations**
- **Further hypothesis**
- Applicability to other **paradigms** - e.g. **OO**, **component based**, **cots**, **open source**, **extreme programming**
- **Theory generation** and **validation** - planned
- **Iteration** and **extension**

Expectation that **approaches** and **results** can be **extended** to **wider systems studies**

Brief Summary

- FEAST studies focused on **area II**, currently **most widespread process paradigm**
- How do extend to **new paradigms**: - e.g. **OO**, **component based**, **cots**, **open source**, **extreme programming**
- Message has been of **interdependencies** between all areas
- Need to develop **outline agreement** on **definitions**, **concepts**, **analysis**
- Then achieve degree of **understanding** of **all** areas and their **interactions**
- Growing **dependence** on computers makes society ever more dependant on **up to date**, **continually evolved** software which, despite growing **complexity**, must remain compatible with **world as it is** and with co-evolving **domains**
- Process **improvement** for **planning**, **management**, **execution**, **control** of **evolution**
- Theoretical **base** and **framework**, **theory of software evolution**, one **key** for success

Models and the Support they Provide

- Extensively recorded in **literature**
- **Data, black box and white box (SD) models, phenomenology, hypotheses**
- Cannot discuss today
- Part II will now provide an **instance** of the **models** and the **methodological approach** it is based on

Some FEAST Publications - See <http://www.doc.ic.ac.uk/~mml/feast>

A complete listing of FEAST papers is provided at and some may be accessed via <http://www.doc.ic.ac.uk/~mml/feast>

Rules and Guidelines

MM Lehman, *Rules and Tools for Software Evolution Planning and Management*, pos. paper, FEAST 2000 Workshop, Imp. Col., 10 - 12 Jul. 2000, DoC, Res. Rep. Nov 2000, a revised version to appear in *Annals of Software Engineering, Spec. Issue on Software Management*, vol. 11., 2001

MM Lehman, *The Future of Software - Managing Evolution*, inv. contr., IEEE Software, Jan-Feb. 1998, pp. 40-44

Theory

MM Lehman and JF Ramil, *Towards a Theory of Software Evolution - And Its Practical Impact*, invited talk, ISPSE 2000, Intl. Symposium on the Principles of Software Evolution, Kanazawa, Japan, Nov 1-2, 2000

Modelling of Evolution Processes

G Kahen, MM Lehman, JF Ramil and PD Wernick, *Dynamic Modelling in the Investigation of Policies for E-type Software Evolution*, ProSim 2000, International Workshop on Software Process Simulation and Modelling, 12 - 14 Jul. 2000, London, UK

BW Chatters, MM Lehman, JF Ramil, P Wernick, *Modelling a Software Evolution Process*, ProSim'99, Softw. Process Modelling and Simulation Workshop, Silver Falls, Oregon, 28-30 Jun. 1999, also as *Modelling a Long Term Software Evolution Process* in *J. of Softw. Proc.: Improv. and Practice*, 2000 v. 5, iss. 2/3, Jul. 2000, pps. 95-102

MM Lehman, DE Perry and JF Ramil, *On Evidence Supporting the FEAST Hypothesis and the Laws of Software Evolution*, Proc. Metrics'98, Bethesda, MD, 20-21 Nov. 1998, pp. 84-88

P Wernick and MM Lehman, *Software Process White Box Modelling for FEAST/1*, ProSim '98 Workshop, Silver Falls, OR, 23 Jun. 1998. As a revised version in *Journal of Systems and Software*, Vol. 46, Numbers 2/3, 15 Apr. 1999

MM Lehman and P Wernick, *System Dynamics Models of Software Evolution Processes*, Proc. Int. Wrkshp. on the Principles of Software Evolution IWPSE-98, ICSE-20, 20-21 Apr. 1998, Kyoto, Japan, pp. 6-10

MM Lehman, DE Perry, JF Ramil, WM Turski and P Wernick, *Metrics and Laws of Software Evolution - The Nineties View*, Proc. Fourth International Symposium on Software Metrics, Metrics 97, Albuquerque, New Mexico, 5-7 Nov. 97, IEEE Comp. Soc. or. n. PR08093, pp 20-32. Also as in K El Eman and N H Madhavji (eds.), *Elements of Software Process Assessment and Improvement*, IEEE CS Press, 1999

WM Turski, *A Reference Model for the Smooth Growth of Software Systems*, IEEE Trans. Softw. Eng., v. 22, n. 8, Aug. 1996, pp. 599 - 600

Resource Estimation

JF Ramil and MM Lehman, *Exploring Cost Estimation Models in the Context of Continuing Software Evolution*, FESMA-AEMES Software Measurement Conference 2000 "Management Excellence through IT Measurement", Madrid, Spain Oct. 18-20, 2000 Lessons Learnt (e.g., Modelling Methodology)

JF Ramil and MM Lehman, *Metrics of Software Evolution as Effort Predictors - A Case Study*, Proc. ICSM 2000, Int. Conference on Software Maintenance, 11-14 Oct. 2000, San Jose, CA Component-based and Integration-intensive Processes

JF Ramil, *'Why COCOMO Works' Revisited or Feedback Control as a Cost Factor*, pos. paper, FEAST 2000 Workshop, Imp. Col., London, 10 - 12 Jul. 2000, 5 pps. Also as Res. Rep. 2000/3, Dept. of Comp. Imp. Col., Feb. 2000

Other

MM Lehman and JF Ramil, *Software Evolution Phenomenology and Component Based Software Engineering*, to appear in *IEE Proc. Softw.*, sp. issue on Component Based Software Engineering, scheduled for Dec. 2000, earlier version as Tech. Rep. 98/8, Imperial College, London, Jun. 1998

JF Ramil (ed.), *Preprints of FEAST 2000 International Workshop on Feedback and Evolution in Software and Business Process*, Imp. Col., London, 10 - 12 Jul. 2000, 124 pp.

JF Ramil, MM Lehman and G Kahen, *The FEAST Approach to Quantitative Process Modelling of Software Evolution Processes*, Proc. PROFES'2000 2nd International Conference on Product Focused Software Process Improvement, Oulu, Finland, 20 - 22 Jun. 2000, in Frank Bomarius and Markku Oivo (eds.) LNCS 1840, Springer Verlag, Berlin, 2000, pp. 311 - 325.

MM Lehman, *Feedback in the Software Evolution Process*, Keynote Address, CSR Eleventh Annual Workshop on Software Evolution: Models and Metrics. Dublin, 7-9 Sept. 1994, Workshop Proc., Information and Software Technology, sp. is. on Software Maintenance, v. 38, n. 11, 1996, Elsevier, 1996, pp. 681 - 686

General Bibliography

A listing of Prof Lehman's papers and other material is provided at and some may be accessed via <http://www.doc.ic.ac.uk/~mml>

References indicated with an '*' have been reprinted in Lehman MM & Belady LA, *Program Evolution—Processes of Software Change*, Acad. Pr., London 1985

- *Belady LA & Lehman MM, *An Introduction to Program Growth Dynamics*, in Statistical Computer Performance Evaluation, W Freiburger (ed), Academic Press, New York, 1972, pp. 503 - 511
- Boehm BW, *Software Engineering*, IEEE Trans. on Comp., v. C-5, n. 12, Dec. 1976, pp. 1226 - 1241
- id.*, *A Spiral Model of Software Development and Enhancement*, Computer, v. 21., May 1988, pp. 61 - 72
- id.*, *Software Engineering Economics*, Englewood Cliffs, N.J, Prentice-Hall, 1981
- Brooks FP, *No Silver Bullet - Essence and Accidents of Software Engineering*, Information Processing 86, Proc. IFIP Congress 1986, Dublin, Sept. 1-5, Elsevier Science Pubs. (BV), (North Holland), pp. 1069 - 1076
- *Lehman MM, *The Programming Process*, IBM Research Report RC 2722, IBM Research Centre, Yorktown Heights, NY, Sept. 1969, also in *Program Evolution—Processes of Software Change q.v.*
- **id.*, *Programs, Cities, Students - Limits to Growth?.*, Imperial College. Inaugural Lecture Series, v. 9, 1970 - 1974, also. in [GRI78], pp. 42 - 69, Lehman MM & Belady LA, 1985, pp. 133 - 163, also in *Program Evolution—Processes of Software Change q.v.*
- **id.*, *Laws of Program Evolution - Rules and Tools for Programming Management*, Proc. Infotech State of the Art Conf., Why Software Projects Fail, - Apr 9 - 11 1978, pp. 11/1 - 11/25
- **id.*, *Programs, Life Cycles and Laws of Software Evolution*, Proc. IEEE Sp. Iss. on Softw. Eng., v. 68, n. 9, Sept 1980, pp. 1060 - 1076
- Lehman MM, Stenning V & Turski WM, (1984). *Another Look at Software Design Methodology*, ICST DoC Res. Rep. 83/13, Jun 1983. Also, Software Engineering Notes, v. 9, no 2, Apr 1984, pp. 38 - 53
- Lehman MM & Belady LA, *Program Evolution,- Processes of Software Change*, Academic Press, London, 1985, 538 p.
- id.* *Evolution - The Cause of Iteration*, Third Process Workshop, Breconridge, CO, Nov. 1986. In *Iteration in the Software Process - Proc. 3rd Int. Process Wrkshp.*, Dowson M (ed), IEEE Comp. Soc. Press, Mar. 1987, pp. 29 - 32
- Lehman MM, *Process Models, Process Programs, Programming Support*, Inv. Resp. To A Keynote Addr. By Lee Osterweil, Proc. 9th Int. Conf. on Softw. Eng., Monterey, CA, 30 Mar. 2 Apr 1987, IEEE Comp. Soc. pub. n. 767, IEEE Cat. n. 87CH2432-3, pp. 14 - 16
- id.*, *Uncertainty in Computer Application*, Comm. of the ACM, Vol. 33, No. 5, May 1990, pp. 584-586
- id.*, *Uncertainty in Computer Application and its Control through the Engineering of Software*, J. of Softw. Maint., Res. & Pract., v. 1, 1 Sept 1989, pp. 3 - 27
- id.*, *Models and Modelling in Software Engineering*, Ency. of Softw. Eng., J Marciniak (ed), Wiley and Co, 1994, vol. 1, pp. 698 - 702
- id.*, *Software Evolution*, loc cit, vol. 2, pp. 1202 - 1208
- Osterweil L, *Software Processes are Software Too, Iteration in the Software Process*, Proc. of the 3rd Int. Proc. Worksh., Breckenridge, CO, 17 - 19 Nov. 1986, IEEE cat. n. TH0184-2, IEEE Comp. Soc. order n. 709, 1987, pp. 79 - 80
- Perry DE, *Policy and Product-Directed Process Instantiation*, Proc. of the 6th Int. Softw. Process Workshop, 28-31 October 1990, Hakodate, Japan
- Rajlich VT and Bennet KH, *A Staged Model for the Software Life Cycle*, Computer, July 2000, pp. 66 - 71
- Turski WM, *And No Philosophers' Stone Either*, Inf. Processing 86, Proc. IFIP Congr., Dublin, Sept. 1 - 5, 1986, Elsevier Sci. Pubs, London, pp. 1077 - 1080
- Wilkes M V, Wheeler D J & Gill S, *The Preparation of Programs for an Electronic Digital Computer*, Addison Wesley Press Inc., 1951, 167 pp.
- Wirth N, *Program Development by Stepwise Refinement*, CACM, v.14, n.4, Apr 1971, pp.221-227
- *Woodside CM, *A Mathematical Model for the Evolution of Software*, J. of Sys. and Softw. vol. 1, no. 4, Oct 1980, pp. 337 - 345 Zurcher FW and Randell B, *Iterative Multi-Level Modelling - A Methodology for Computer System Design*, IBM Res. Rep. RC 1938, Nov. 1967, IBM Res. Centre, Yorktown Heights, NY 10594. Also in Information Processing 67, Proc. IFIP Congr. 1968, Edinburgh, Aug. 1968, pp. D138 - 142