

Lecture I

Software Evolution and the Real World

University of Sannio

2nd May 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>

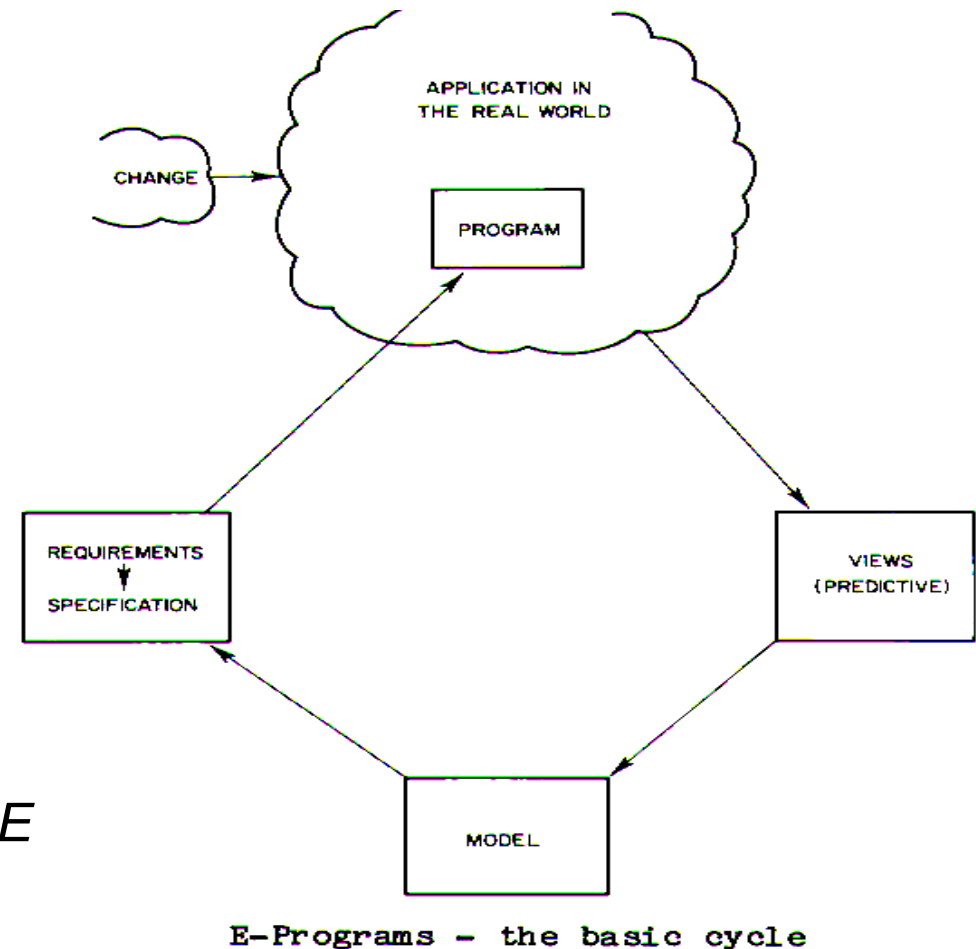
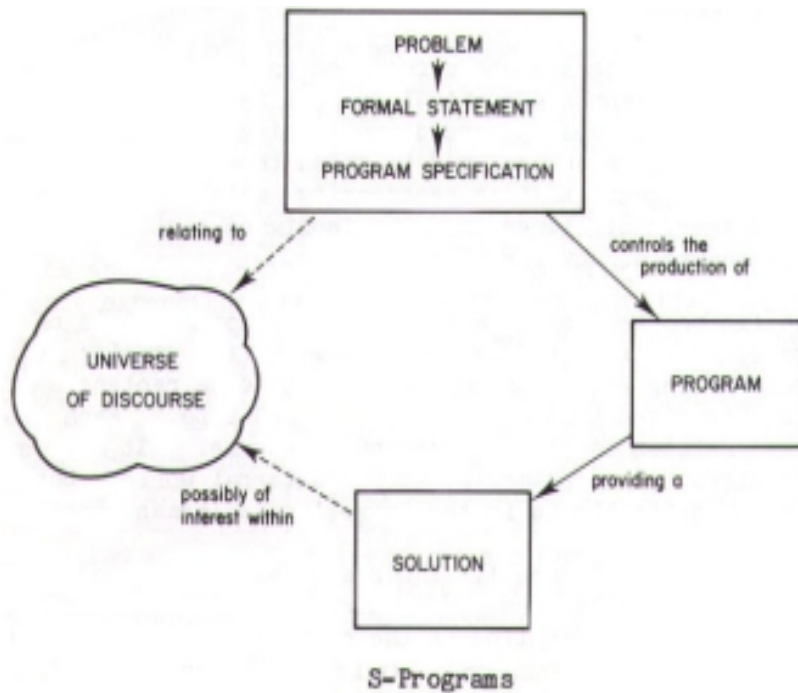
Introduction

- **Past** - some highlights
 - 1968-9: IBM software **process** study - including **empirical data analysis** and "**The Programming Process**" report
 - 1971: first **discussion** of implications of **feedback system** nature of software process
 - 1974-86: **laws** of software evolution
 - 1979: **SPE-type program classification** schema
 - 1989: principle of **software uncertainty**
 - 1994: **FEAST** hypothesis
 - 1996 to 2001: **FEAST/1** and **FEAST/2** projects
- **Present** - **FEAST** studies - **F**eedback, **E**volution **A**nd **S**oftware **T**echnology
 - study of **impact** of **feedback** on **global** software **process** and **evolution** of its **product**
 - in collaboration with **ICL**, **Logica**, **MoD-DERA**, **Lucent Technologies**, **Matra-BAe**, **BT**
 - **evolution** of systems of **different: size, application area, development, environments, processes**
- **Future**
 - development of **formal theory** of software process evolution
 - **operational system dynamics models** of **global** software processes
- Thirty years of **observations** confirms generality of continuous **disciplined evolution** of **E-type** systems in distinction to **static** nature of **S-type** systems

***What** are these software **types** and **why** must the **E-type** evolve?*

SPE Program Classification

- 1980 Visualisation



- P*-type **class redundant** since every such program is either of type *S* or type *E*
- Classification **extended** to include **systems, applications, processes**

•What are their specific **characteristics**?

S-type

Definition, Properties

- Program properties **formally specified** - formal to prevent ambiguity
- Criterion of **acceptability** - **verified correctness** relative to specification
- **Unsatisfactory** behaviour/results in execution after verification implies **unsatisfactory specification** - **incorrect** or **inappropriate**
- **Open loop** by definition
- System remains **unchanged** (static) until change is **triggered** by **external force**
- System **validation** a matter of opinion/judgement assumption - **beauty contest**
- Fixing requires **change to specification** followed by derivation of **new program**
- this could well be obtained by modification of the original code

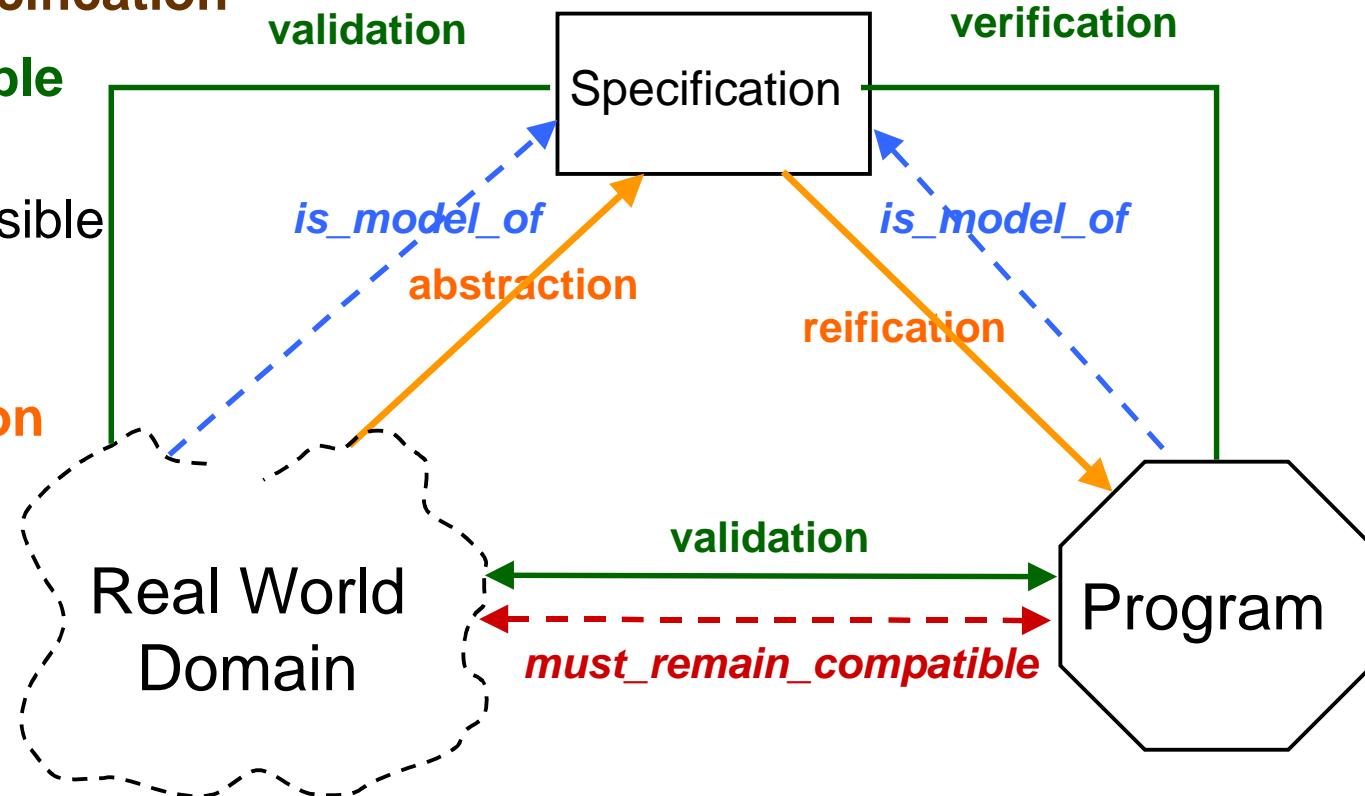
Process Role

- For effective **maintenance make-up** of programs must be precisely **known**
- **Individuals** and small **group** assignments must be **precisely specified**
- **S-type** programs the **bricks** for **construction** of **E-type** systems

S-type **programs** and **systems** of **programs in the real world**

Programs in the Real World

- **Computing system and its real world operational domain** are **models** of a **specification** reached by processes of **abstraction** and **reification** respectively
- Both have, in general, **properties not addressed** by **specification**
- May become **incompatible with one another**
- **Verification** may be possible in **whole** or in **part**
- Continuing **validation** to ensure user **satisfaction**
- **Both program and specification must be maintained valid**



E-type systems

E-type Systems

Definition, Properties

- **Model** of the real world - or at least of **part of it**
- **Systems** solving a **problem** or addressing an **application** in real world
- **Correctness** of system is **relative to real world**, **cannot** be **demonstrated**
- **Behaviour** and/or **results** of **execution** determine **acceptability**
- But system must **remain** satisfactory **model** of application in **operational domain**
- True **validation criterion** is satisfaction of **stakeholders' current needs**
- **Intrinsic** need for **continuing application**, **system evolution**
- **Needs, desires, opportunities**, operational **domain all evolve** - system must satisfy **current needs** and **interests** of **application**, **operational domain**, **stakeholders**

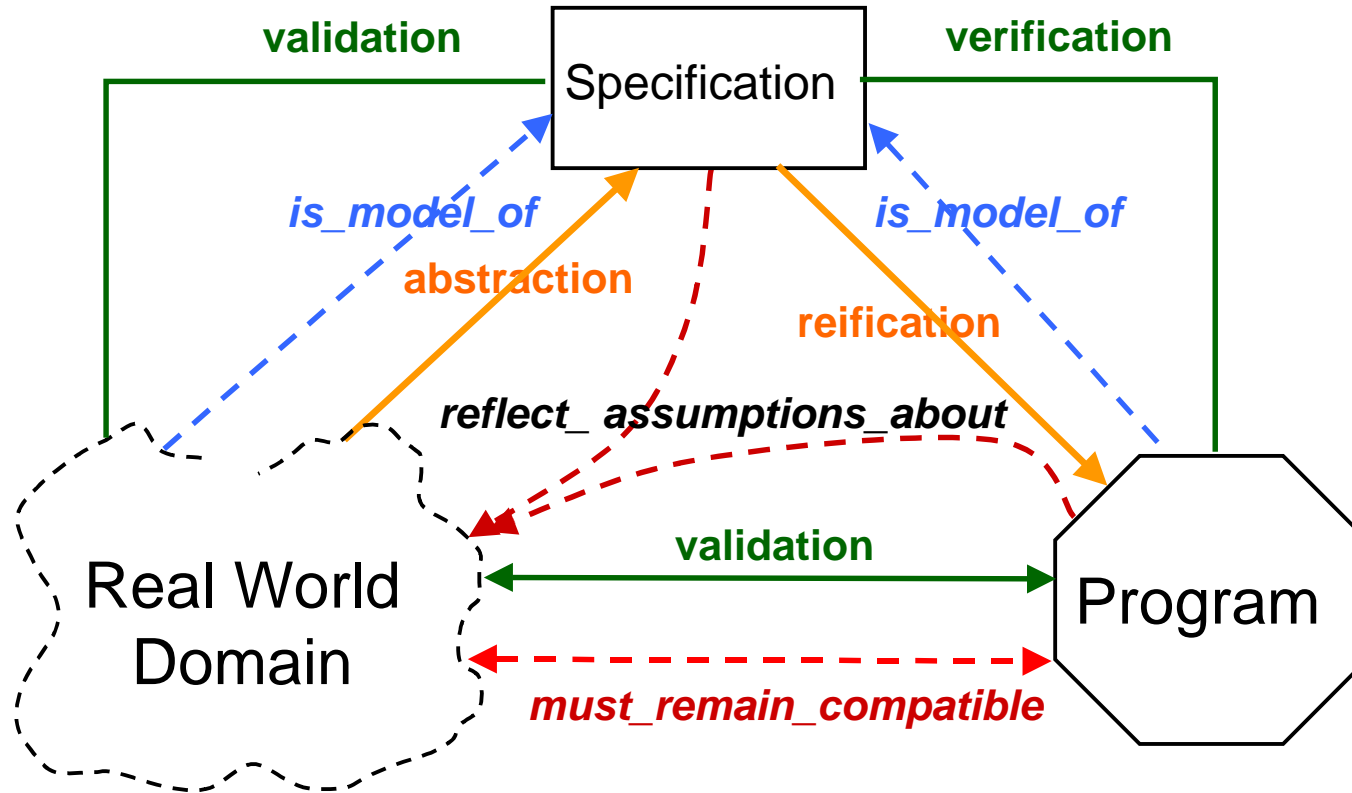
Process Challenge

- **Maintain** system, that is **specification**, **programs**, **documentation set**, as **valid**
- that is **acceptable** - **model** of application in its operational domain

How can all this be achieved?
How are E-type systems **evolved** - **developed, maintained**?

E-type Program as a Model

- More complete **visualisation** of **real world/specification/program** relationship



Program reflects real world

Real World/*E*-type System Relationship

- Conclusion: **program** is **model** of a **specification** that also has **operational domain** as a **model**
- **Universe**, is the ultimate **operational domain of** *E*-type systems
- And is **unbounded**, at least in its **properties**
- **Operational domain** of an *E*-type system is a **sub-domain** of the **universe**
- **Operational domain** also **unbounded** in its **properties**
- **Application** is equally - **potentially** - **unbounded** in its **properties**
 - can always add another **bell** or **whistle**
- **System** is **finite** - e.g. human creators, finite space, finite resources including time

E-type system **is**, perforce, a finite **model** of a finite **specification** of **two unbounded domains**

The System – Real World Gap

- An **E-type system** is **bounded reflection** of **application** in **operational domain**
- It is **intrinsically** an **incomplete reflection**
- **Gap** between **system** and **real world operational domain** bridged by **assumptions embedded** in **global system**
 - e.g. **implementation**, **procedures**, **documentation**, etc.
- **Number of assumptions** is unbounded
- **Assumptions** are adopted **throughout global** development, 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 may be 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**

Lecture II

The Global Software Process

University of Sannio

2nd May 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>

E-type System Evolution Process

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



Preliminary **statement** of required **change**



Yields - **new** or **changed application concept** (high level **requirement**)
and/or - **new** or **changed application domains**

- **Initially both concept and domains:**
 - **ill defined**
 - **not** fully or precisely **verbalised**
 - not explicitly **bounded**
- E-type system **is**, perforce, **finite** model-like **reflection** of **finite specification** of **multiple unbounded domains**
- **Bounding of application and domain** must become a **conscious** process step

Activity that **continues** throughout **process** and **system lifetime**

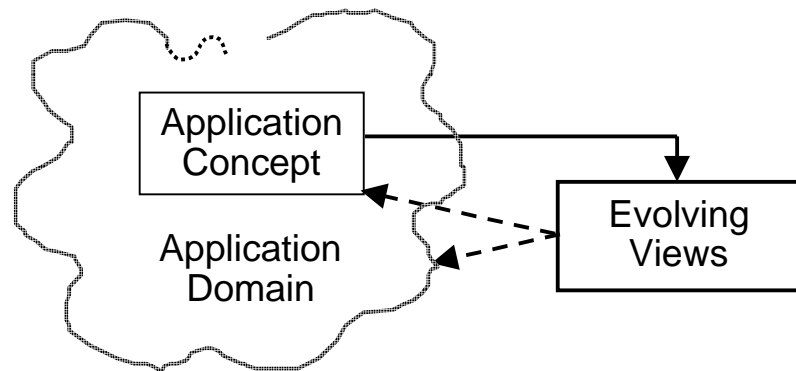
Bounding in E-type Evolution

- **Bounding** an **abstraction process** that involves **adoption of assumptions**
- Latter **bridge gap** between **unbounded domains** and **finite system**
- Assumption **set** relating to any *E*-type program **unbounded** since an **unbounded number of attributes** are **excluded**
- Of these an unbounded number are **unknown, unrecognised, not agreed** and/or **not validated** - many also not necessarily relevant, **initially** or **intrinsically**
- As world **changes** some previously **valid assumptions** become **invalid**
- **System** and/or **documentation** must be **corrected** to remove **non-validity**
- In **maintaining validity** of relevant **assumption set one maintains satisfaction** of - at least some - **stakeholders**

Assumptions direct **consequences** of **bounding processes**

Bounding Process

- **Elicit, reconcile, merge, limit** biased, partial views of individual stake holders:
 - organisational and individual **users**, i.e. all those that use system directly or indirectly
 - **marketeers, suppliers, procurement** personnel
 - domains and application **experts**
 - system, software **engineers, developers**, etc., etc.
- **Process** blends **viewpoints**, agrees **assumptions** to provide development **base**
- All **participants** must be alert to **capture, record assumptions** about **application, domain, interfaces** and **human behaviour** - in development, application **processes**
- **Convergence** to consensus changes **application, bounds**
- Involves **objective** technical criteria, **pragmatic** considerations, **assumptions**
- Often determined by **individuals** or **local groups**, taken in **isolation, not documented**

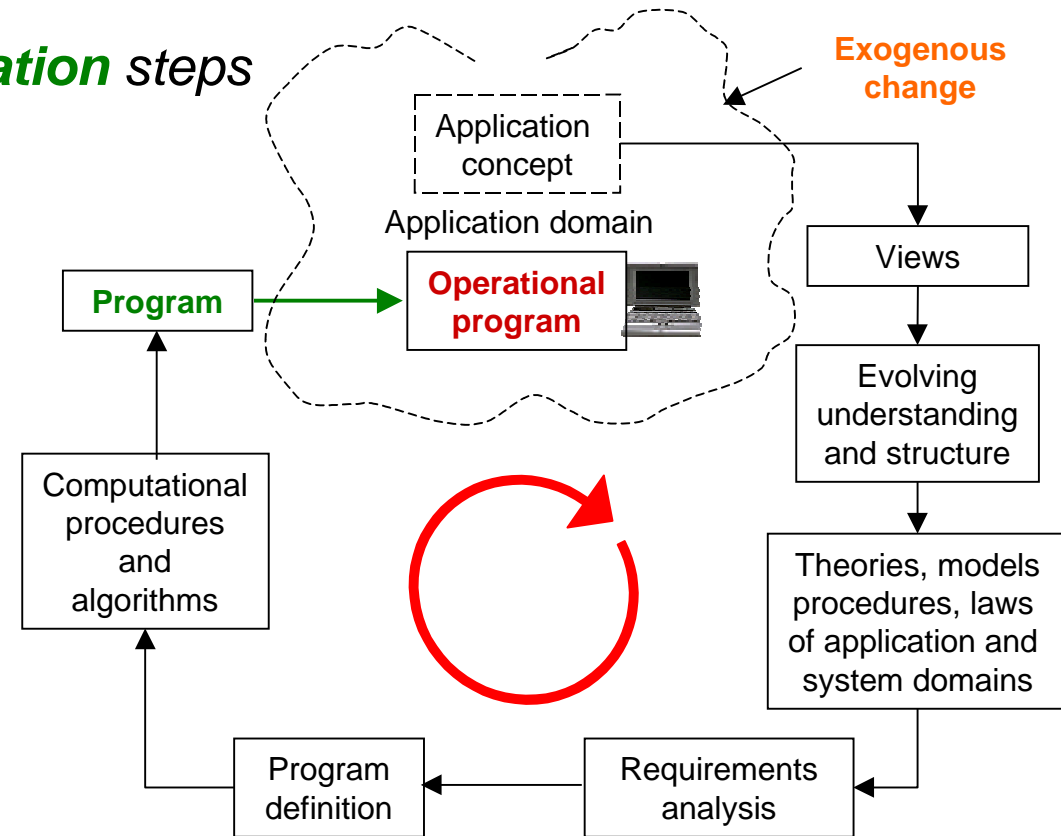


First step in *E*-type **Evolution** - **Development** and **Maintenance** - **Process**

High Level View of E-type Process

Series of **development** and **implementation** steps

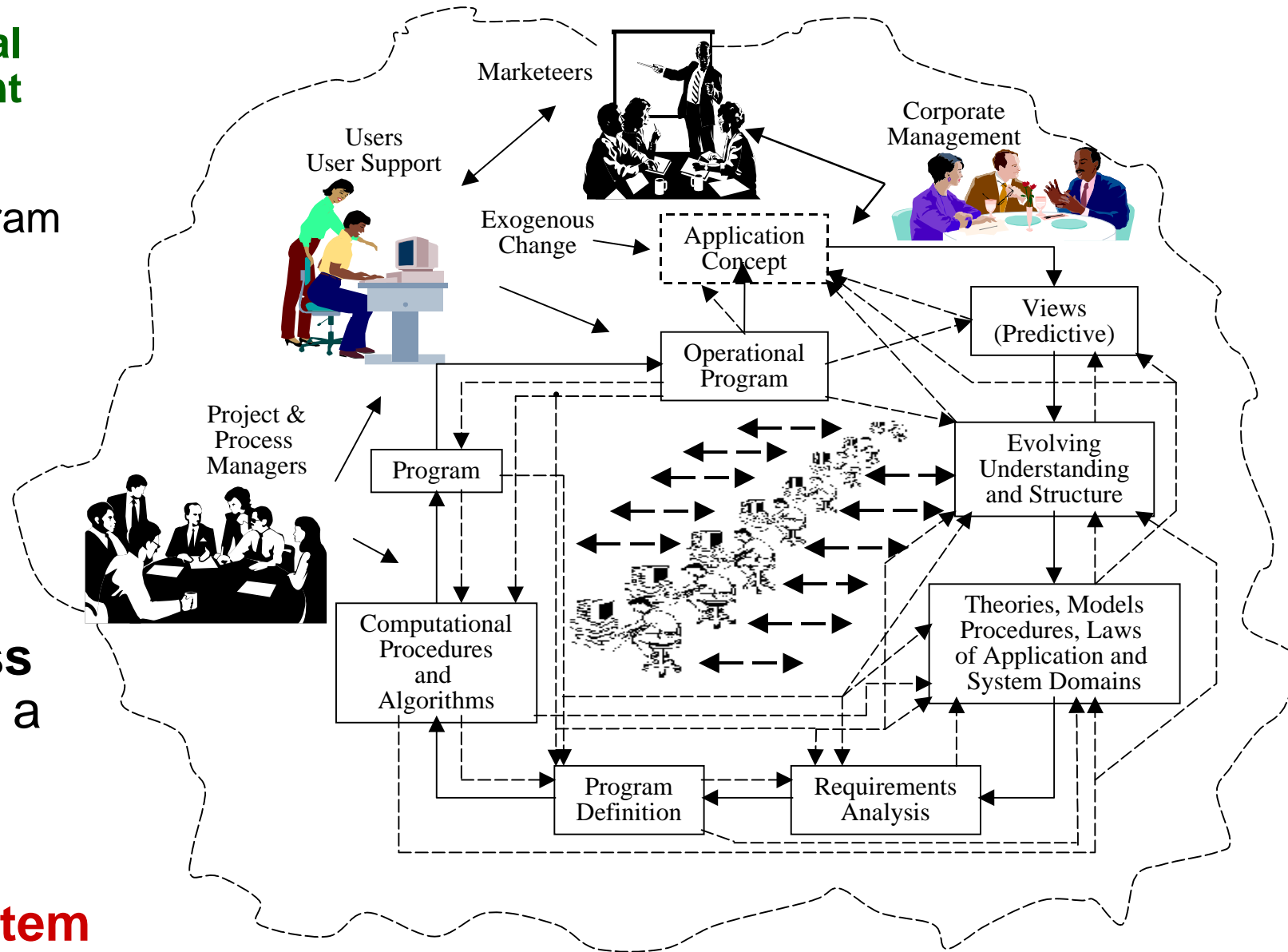
- Culmination: **validated program**
- Final step: **installation, operation**
- **Installation changes domain**
- **Usage changes application**
- In a **changing real world** domain
- System **contains** implicit **model** of **itself**
- **Installation** and introduction into **usage** closes major **feedback loop** that drives **iterative process releasing** a sequence of **upgrades** to users



Driver of **continuing** software **evolution**

More Realistic Picture

- **More** than just **technical development** activity
- Previous diagram a **fiction**
- Process **not sequential**



Global process is, 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 **people, groups, activities** with diverse **responsibilities**
- **Complex, non-linear**, possibly **time varying**, loop structure
- Satisfies engineering **definition** of **feedback** - “**Output from part of system (process) modifies one or more of its inputs**”
- All these **factors** influence process **behaviour** and **product**
- Must be considered when **modelling, planning, assessing, predicting, controlling, improving** their **properties, behaviour**

These and other **observations encapsulated** in **empirical generalisations** on **software evolution**

Laws of Software Evolution

- The **empirical generalisations** had their **roots** in the **earliest evolution study**
- Follow on of 1968 **programming process study** revealed **disciplined** long term **evolution** of OS/360
- Subsequent study of three other operating systems revealed very similar evolutionary behaviour
- **Models** of **observations** and their **interpretation** suggested that they reflected **human** and **organisational structure, behaviour** and **processes** rather than low level aspects of software technology
- Hence they were **encapsulated** in series of, now, 8 **statements** termed **laws**, behaviour abstracted appearing to be **outside** purview of **software engineers**
- FEAST/1, /2 **extended** study by analysis of data from **several sources**, **repeated** and **extended** earlier observations so increasing **confidence** in their **validity**

The **Laws of Software Evolution** as **currently stated**

The Laws

No.	Brief Name	
I 1974	Continuing Change	An <i>E</i> -type system must be continually adapted else it becomes progressively less satisfactory in use
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
III 1974	Self Regulation	Global <i>E</i> -type system evolution processes are self-regulating
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
V 1978	Conservation of Familiarity	In general, the average incremental growth (growth rate trend) of <i>E</i> -type systems tends to decline
VI 1991	Continuing Growth	The functional capability of <i>E</i> -type systems must be continually enhanced to maintain user satisfaction over system lifetime
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
VIII 1996	Feedback System (Recognised 1971, formulated 1996)	<i>E</i> -type evolution processes are multi-level, multi-loop, multi-agent feedback systems

Basis of the Laws

- Encapsulate **behaviours** and **regularities** observed over the years
- Reflect **global** process that **includes all activities**
- And impact of **system dynamics, management, organisational, marketing, user attitudes** and **actions, more than** they do **architectures, methods, tools**
- **Focus** on **organisational, human** (e.g. cognitive) factors and on **process** as a **system**
- **Complexity** of **feedback structure** and **phenomenon** and the resultant **process behaviour** identifies **empirical** and **dynamics modelling** as tools for achieving **process understanding**, and process **improvement**
- Relate to **organisational** and **human behavioural** issues rather than directly to computing issues
- **Software engineering** professionals cannot, in general, change these without a significant **broadening** and **extension** of software engineering

Must be considered as **laws**, at least for time being

Lecture III

Software Process: Theory

University of Sannio

2rd May 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>

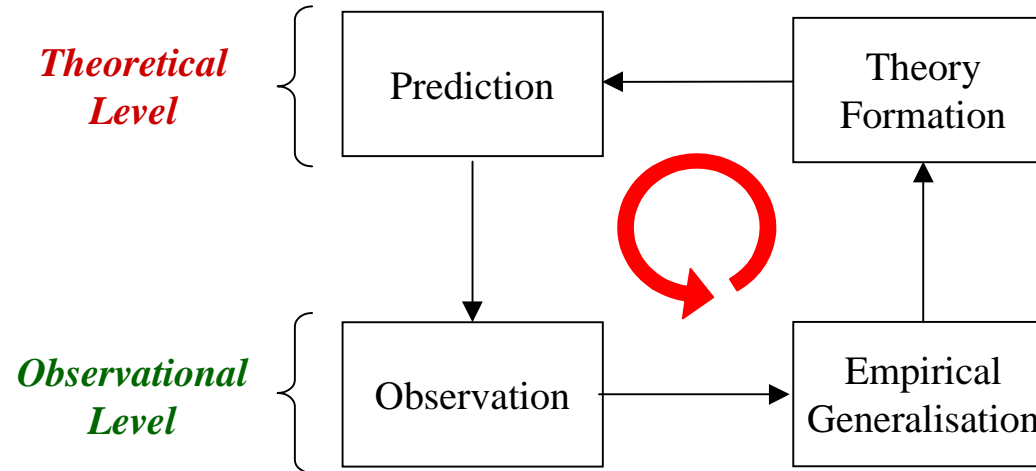
Summary of Some Fundamental Observations

- An *E*-type program is a **model** of an **abstraction** of an application with an **unbounded** number of **properties** in **real world**, therefore **unbounded**, **domain**
- As a **model**, it is essentially **incomplete**, the reification of an **abstraction** of a **problem/application** in **operational domain** with which it must be **completely compatible**
- **Gap** between **program** and **domains** bridged by **assumptions** implicitly **embedded** in program
- *E*-type programming processes are **feedback systems** and develop a **system dynamics** which **drives** and **regulates process** of **product evolution**
- Feedback system **characteristics** reflected in **common patterns of behaviour** observed over systems of many **types, sizes, organisations, processes**
- **Observed patterns** provide basis for **formulation** of the **laws** and, more generally, **empirical generalisations** in the sense of Carnap
- These, in turn, provide **basis** for development of a **Theory of Software Evolution**

Approach to 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 **validate** against **observations**
- **Iterative extension**, to yield **theorems** - further **generalisations, axioms** may also emerge

Can approach be **applied** to **software evolution**?

Formal Theory of Software Evolution

- Previous slide outlined **Carnap approach** to general **theory formation**
- **FEAST projects observations**, modelled and interpreted, led to identification of **behavioural invariants** and **mathematical models**
- Their development provides foundations for application of the approach to the development of a **formal theory of software evolution**
- **Exploratory steps** indicate **feasibility** of **observation level theory**
- Every **reason** to **believe** that a **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 a number of **observations**
 - should eventually be reflected in **empirical generalisations**
 - 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** must await **formalisation**

Example of the **proposed SETH development**

Definitions

- **Definition 1:** *Real world* encompasses **entire universe** and all **happenings** in it
- **Definition 2:** A real world *domain* is **sub-domain** of the **real world**, its **attributes** an **unbounded sub-set** of the unbounded set of real world attributes
- **Definition 3:** An *operational domain* is a **real world domain** whose **attributes** are a **sub-set** of the attributes of that domain
- **Definition 4:** An *E-type program* is one **solving a problem** or **implementing an application** in an **operational domain**
- **Definition 5:** A *specification* is an **abstraction** of an **operational domain** that is believed **valid** and **sufficient at some point in time**
- **Definition 6:** A *E-type program* **implements** an **application** in its **operational domain**
- **Definition 7:** To be **satisfactory** an *E-type program* must **reflect** all the required **attributes of the operational domain** and **none** that are **incompatible** with it
- **Definition 8:** *Validation* is a **process** that demonstrates 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:** Those **attributes** of an application and its **operational domain** that make it **satisfactory** must be reflected in *E*-type **program** implementing it

Some Potential Inferences at the Observation Level

- **Inference 1: Assumptions** underlying a specification or embedded in a program may become **invalid** so **invalidating** the program
- **Inference 2:** The **real world** application domain from which specification of an E -type program is abstracted has an **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** and its program **implementation** may be incompatible
- **Inference 6: Behaviour** of an E -type program when executed is **inherently uncertain** and **cannot be guaranteed** to be **satisfactory**

Principle of Software Uncertainty

- These **inferences** appear to **follow** from **stated definitions** and **observations**
- When **proven correct** at the **theory level** these, and others to be derived, will be established as **theorems** of the **theory**
- More **precise statement**, **formal proofs** remain to be **developed** at the **theory level**
- **Sixth inference**, first stated in 1989 as the **Principle of Software Uncertainty**, follows from definitions, observations and other inferences
- A first **example** of how **theory of software evolution** might **develop**

Theorems have direct **practical implications**

Lecture IV

Software Process: Practice

University of Sannio

3rd May 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>

Rules and Tools for Evolution Management

- Exemplify **rules**, **tools**, **guidelines** one might expect to derive from theory, when developed
- Detailed discussion in **Annals of Software Engineering**, Special Issue on Software Management, Spring 2001

System Evolution

- Have concluded that **evolution inevitable** if *E*-type system is to remain **satisfactory**

Some implications

- **Plan evolution** in **advance taking** system, domain **volatility** into account
- **Create, maintain** comprehensive **documentation** of **specification, design, implementation, evolution patterns, metric data** - to **control cost** of future evolution
- Provide ready **access** by evolution teams to **domain specialists**
- **Validation** of **all** changes must address **interaction** with and **impact** on parts of system **not changed**
- **Prepare, update, formalise** comprehensive specification (**long term goal**)
- Provide **tools** for **capture** and structured **recording** of **specifications** and **assumptions, changes** to them classified by **categories**
- **Establish baselines** of **key measures** over **real time** and **releases**
- **Collect, plot, model, interpret** historical data to determine **patterns, trends, rates of change** and **growth**

Metrics and Modelling

- Develop **automatic tools** to support **data collection, modelling**
- **Collect, plot, model, interpret** historical evolution data to determine, for example, **patterns, trends, growth** and **rates of change**
- **Maintain models** on basis of **new data** to **detect** and **accommodate process** and **environmental changes**
- Model **dynamics** of global process
- Use **dynamic model** to **plan** further work, identify **interactions**, improve **planning**, optimise **policies**, control **strategies**
- Use **dynamic models** to **improve** and **discipline process improvement** process
- Model, consider, manage **global process** that **embeds** technical process and associated **organisational links**

Release Management

- **Constrain**, **scope**, **size** of **release increments** based on models of **past** system **incremental growth**
- **Observe** safe **change rate limits**
- Use '**m+2s**' or similar **criterion** to determine whether growth increments **safe**, **risky**, **unsafe**
- Follow established **software engineering principles** - e.g., information hiding - to **spread of change** between system **elements**
- **Assign effort** to **control** and **reduce complexity** and its **growth**
- **Alternate** major - functional enhancement, extension - and minor - clean-up, restructuring - **releases**
- **Plan** for **clean-up releases** when major increments in functionality are unavoidable
- **distribute large functional increments** across several releases as in **evolutionary development**

Management of Assumptions

- Have observed that **invalid assumptions** are a major **source** of **defects**
- *Some implications*
- **Capture, document, structure, retain assumptions throughout** process
- **Include assumptions** in **validation** process and **re-validate** with **changes**
- **Review** assumption set **periodically** and **implement** consequent changes as necessary
- **Update, document assumptions** during all **process steps**
- **Develop** and **use tool support** for **processing** and **structured classification** of **assumptions** throughout process
- Institute **periodic** and **event-triggered reviews, assessments** to **identify** or **anticipate** changes in assumption set
- **Improve questioning** of assumptions, for example, by using **independent** implementation and validation **teams**
- **Document rationale** and **underlying assumptions** when any **aspect** of **domain, application, specification, design, implementation changes** or **are changed**

Planning, managing, controlling assumptions about **application** and **operational domains** is **key** to successful software evolution

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*, ol. 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, Maryland, 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 fo 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

Cost 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