

Approach to a Theory of Software Evolution

IWPSE 2001

Vienna

10 September 2001

M M Lehman J F Ramil

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

- **FEAST** projects have **substantiated**, refined, extended **observations** that first emerged in 70s as a result of a study of **IBM programming process**
 - Had revealed **growth**, **evolutionary behaviour**, of **OS/360**
 - FEAST and previous studies led to formulation of **characteristic behaviour**, properties, behavioural **patterns** and **regularities** encapsulated in
 - SPE program classification
 - laws of software evolution
 - a principle of uncertainty
 - FEAST hypothesis
 - and elsewhere
- which, together with **observations** and **patterns** identified by others, appear to provide sufficient **inputs** for initial development of **theory** of **software evolution**

What theory, why?

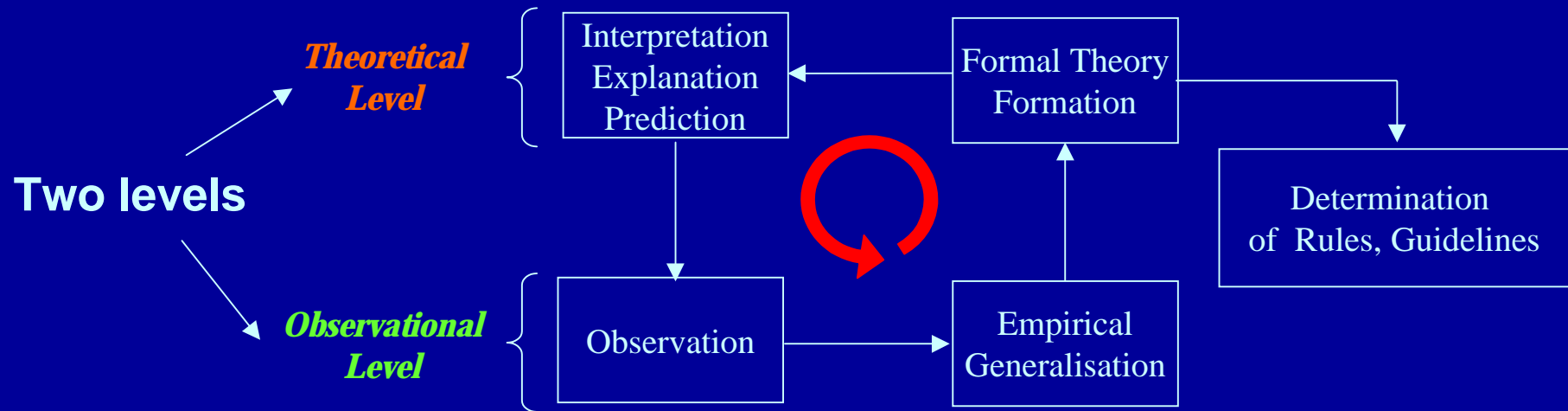
Why a Theory?

- Any useful scientific theory is expected to **explain**, **systematise**, **encapsulate** observed **phenomena**, providing **insight** into observed **facts and patterns**
- Helping to clarify **definitions** via formalisation or otherwise
- And also to provide basis for **prediction**, **control** of **phenomena**
- **Need** for theory already identified in Garmisch conference in 1968
- In general, **existing** theory addresses programming methodology, not **full industrial process**
- Most approaches to software **process improvement** *ad hoc*, individually conceived, based primarily on **experience**, **intuition**
- Theory seeks, in general, to establish **links** between **descriptions** of facts and **prescription** - when achieved, offers basis for **justification** of *good practice*

If funded, proposed project, **SETh**, will seek to investigate **approaches** and develop foundations of a theory

What Theory? - Basic Two-Level Approach to Formal Theory Formation

- **Observations**, behavioural invariants, **phenomenology**, the starting point
- A basis for **informally expressed empirical generalisations** and **definitions**



- From **empirical generalisations** one can initiate formation of theory stated in a **formal notation**
- **Predict behaviours** on basis of **emerging theory** and **validate** against **observations**
- **Iterative extension**, yields **theorems** - further **generalisations** and corresponding **definitions** may also emerge from further **observations** suggested by **emerging theory**
- Derive **rules** and **guidelines**

What Theory? Detailed Approach

- Alternative **implementations** of two-level approach to **theory formation**, for example, those inspired in terminology and reasoning of:
 - *Experimental Sciences*, as discussed by Carnap and hence termed here **Carnapian**
 - *Abstract Theories* as, for example, for Geometry and termed here **Euclidean**
- At this stage, both **approaches** considered **promising**, seen as **complementary** with potential to yield evolution management rules, tools, guidelines
- **Evaluation, balance** between them - part of initial tasks, when project is launched

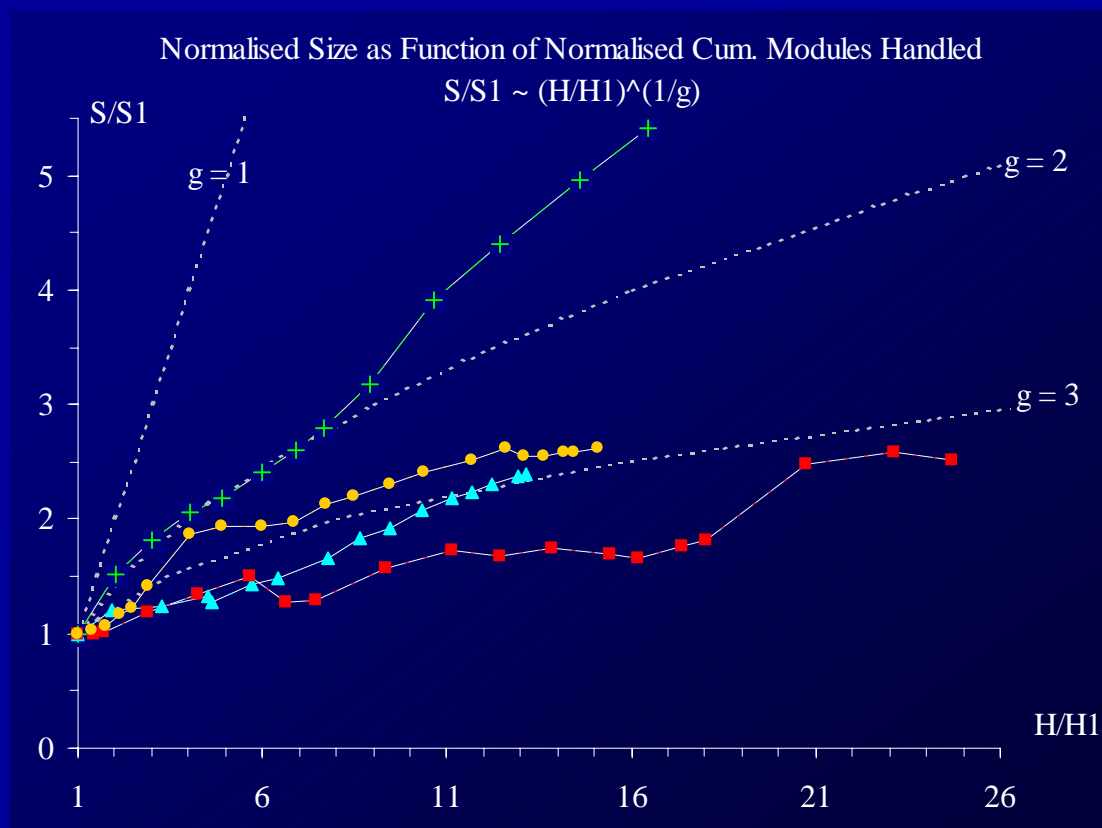
Both briefly discussed

Carnapian Implementation of the Two-Level Approach to Theory Formation

- From observations in the form of empirical data one derives **empirical generalisations** such as a **relationship** between two or more **attributes** of entities in software evolution - for example, the software, its operational domain, the process, implementing organisation
- Transition to the theoretical level is exemplified by **formalisation** and **explanation** of the empirical generalisations - with explanation involving determination of **mechanisms** that MAY underlie generalisations
- On basis, for example, of data for the evolution of four systems, **relationships** between system size and two other attributes were determined and - with segmentation - yielded **growth models** with prediction accuracy from 2 to 17 percent (MMRE)

Example of Input to Carnapian Theory Development

- This yielded following candidate for an **empirical generalisation**:
“Size of future releases predictable at a high level of precision by a model, possibly segmented, of form $S_i/S_1 = H_i/H_1^{1/g}$ where i is release sequence number, S_i is size at release i , H_i is the *work rate* in cumulative number of modules handled and g is a parameter”



- Its **explanation** triggers theory development...
- For example, **case $g \sim 3$** is linked to Turski's **inverse square model** and to **complexity constrained growth** - details in Turski 1996 (inverse square model) and in Lehman *et al* Proc. ICSM 2001

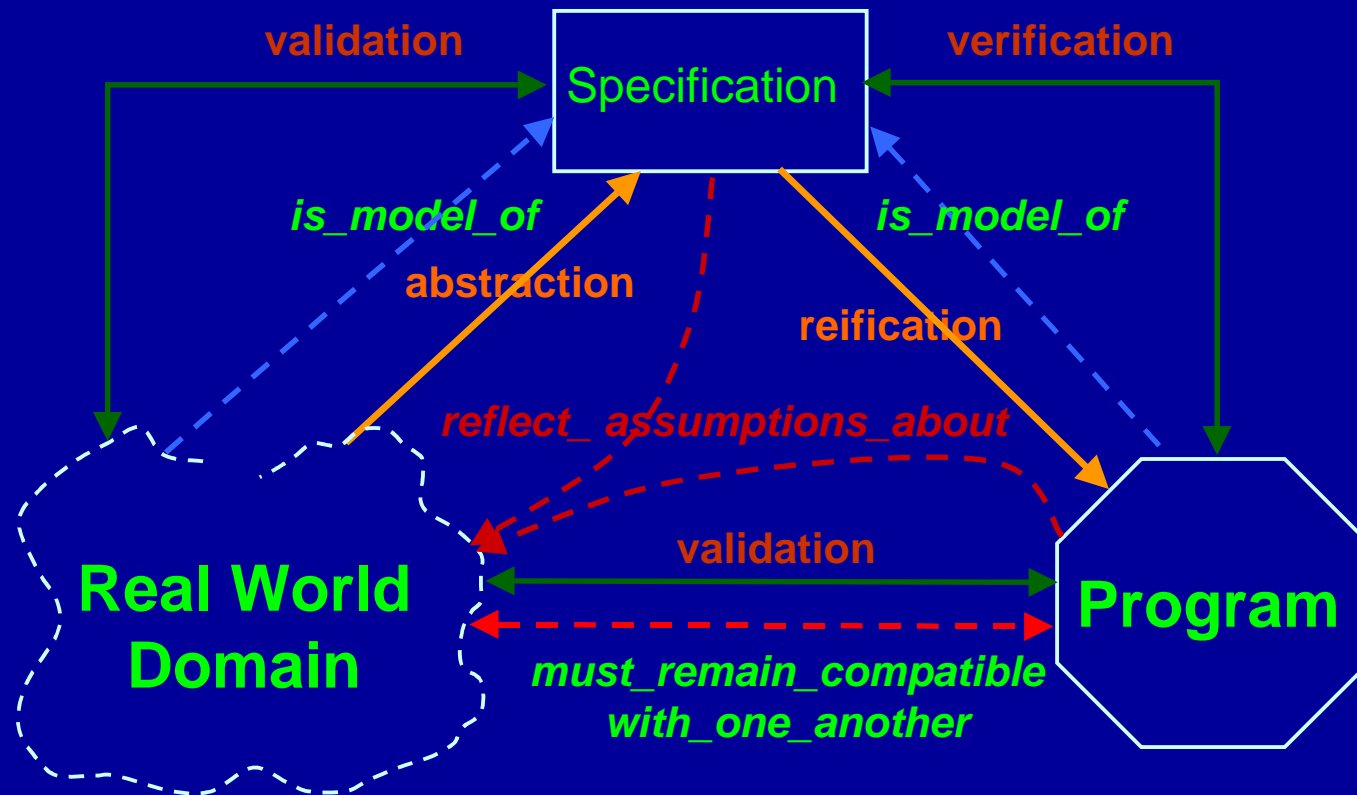
Outline Implementation of Based on Modified Euclidean Terminology

- **Intuitive definitions**
 - initial **formulation**
 - require **refinement** and **precision** via **formalisation**
- **Identify** a number of **observations**
 - should eventually be linked to empirical generalisations
 - basis for **axioms** in **formal theory**
- **Propose** possible **inferences**
 - derived from current **models** and **interpretations**
 - basis for potential **theorems** in **formal theory**
- **Iterate**
 - **test** candidates sets of definitions, observations, inferences
 - **refine** until a useful and satisfactory set is found
- Sketch of **approach** to achieve one **proof**, follows - **full proof** must await **formalisation**
- As must expression at theory level

Brief example illustrates possible transition between levels

Entities and Relationships Involved

- Diagram illustrates entities and relationships at the heart of example that follows



Some Definition Candidates

- **Definition 1:** An ***E*-type program** is one that supports an **activity** or solves a **problem** in the **real world**
- **Definition 2:** A **specification** is a statement of the **properties** of a program believed to be necessary to ensure that the program will be **acceptable** to its **stakeholders**, as-is and in execution
- **Definition 3:** As a **specification** of an *E*-type program, an ***E*-type specification** is an **abstraction** of a **problem** defined in a **real world domain**
- **Definition 4:** The **real world domain abstracted** into a specification is a **model** of that specification
- **Definition 5:** **Properties** not addressed in the *E*-type specification are, ***de facto***, declared to be of **no concern**. The program may or may not possess them
- **Definition 6:** An **explicit assumption** is a **statement** in a specification or program that is, in some sense, **arbitrary** and does not follow from the remainder of the assertions in the specification or program
- **Definition 7:** **Omission** of a real world property from a **specification** of an *E*-type program is an **implicit assumption**
- **Definition 8:** An **assumption** reflected in a specification or program is **invalid** if **execution** of a program satisfying the specification (or embedding the assumption) is **unacceptable** for reasons related to the assumption

Some Observations - candidates for axioms to be formulated

- **Observation 1:** The real world may be partitioned in an **unbounded** number of ways into domains that, in general, each possess an **unbounded** number of **properties**
- **Observation 2:** The real world is **dynamic**. Its **properties** change over time
- **Observation 3:** *E*-type programs as such (as distinct from such programs in execution) are **bounded** in size

Potential Inferences at the Observation Level - candidates for theorems to be **proven**

- **Inference 1:** As the real world changes, **assumptions** reflected in the specification may become **invalid**, and the specification is no longer a valid abstraction of the (changed) execution domain
- **Inference 2:** An *E*-type program is a **model-like** reflection of the execution domain identified by its specification
- **Inference 3:** The **real world** domain abstracted in the *E*-type specification has an **unbounded** number of properties
- **Inference 4:** The **number** of **distinct properties** of an *E*-type program implied by its specification is **finite**
- **Inference 5:** An *E*-type program (in execution) is **essentially** incomplete since it cannot reflect an **unbounded** number of real world properties
- **Inference 6:** *Behaviour* of an *E*-type program in execution is **inherently uncertain** and **cannot be guaranteed** to be **acceptable**

Principle of Software Uncertainty (1989-1990)

- These **inferences 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**
- **Precise statement, formal proofs** remain to be **developed** at the **theory level**
- Nevertheless, the **principle of software uncertainty**, first stated in 1989, is a **candidate theorem** as per the sixth inference
- **Example** of how **theory of software evolution** might be developed

Theorems have direct **practical implications**

Example of Expected Outputs - Rules and Tools for Evolution Management

- **Guideline 1:** Continually **capture** and perform regular reviews of validity and impact analysis of **assumption sets**
- **Guideline 2:** **Co-evolve specification and program** to ensure compatibility with real world domain, as properties of the latter change

Some other 30 guidelines have been identified

Rules and Tools for Evolution Management - Areas Covered

- **Plan**, **management** and **control** of system evolution
- Management of **Assumptions**
- **Release** Management
- **Metrics** and **Modelling**

Charts with example of what can be derived available follow

Detailed discussion of rules and tools in **Annals of Software Engineering**
special issue on Software Management, vol. 11, 2001, in press

An earlier version available from:

http://www.doc.ic.ac.uk/~mml/feast/papers/pdf/611_2.pdf

Some Rules to Plan, Manage & Control System Evolution

Observation that **evolution** is **inevitable** if *E*-type system is to remain **satisfactory** suggests that one:

- **Plans evolution** in **advance taking system** and **domain volatility** into account
- **Creates, maintains** comprehensive **documentation** of **specification, design, implementation, evolution patterns, metric data** - to **control cost** of future evolution
- Provides ready **access** by evolution teams to **domain specialists**
- **Validates** changes by addressing **interaction** with and **impact** on all parts of system including those **not touched**
- **Prepares, updates, formalises** comprehensive **specification** - long term goals
- **Provides tools** for **capture**, structured **recording**, of **specifications** and **assumptions** with **changes** to them classified by **categories**
- **Establishes baselines** of **key measures** over **real time** and **releases**
- **Collects, plots, models, interprets** historical data to determine **patterns, trends, rates of change** and **growth**

Inevitability of system **change** and **growth** - 1st and 6th laws - a **lifetime challenge**

Management of Assumptions

Observations that **invalid assumptions**, a **source** of evolution activity suggests:

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 **status**
- **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 is **changed**

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

Release Management

- **Constrain**, **scope**, **size** of **release increments** based on models of **past** system **evolutionary behaviour**
- **Observe** safe **change-rate limits**
- Use '**m+2s**' or similar **criterion** to determine whether growth increments are **safe**, **risky**, **unsafe**
- Follow established **software engineering principles** - e.g., information hiding - to minimise **spread of change** through system
- **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**

Release must be **planned** as a **whole**

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, 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 models** to **plan** further **work, identify interactions, improve planning, optimise policies, control strategies**
- Use **dynamic models** to **discipline** and **improve process improvement** process
- Consider, model, manage **global process** that **embeds** both **technological process** and associated **organisational** and **environmental links** and **activities**

Metric based, **interpretation, empirical analysis** an **essential tool** for **evolution management**

Final Remarks

- **Rules, tools, guidelines** identified during FEAST
- Many are **self evident** or **reflect currently recognised good practice**
- Their **refinement** and **extension** will constitute a **significant advance** for that technology providing for the first time, for example, a **conceptual framework** for software **process improvement**
- **SETh** proposal has attracted a number of **industrial collaborators** who have recognised its potential and awaits funding decision

References

- [bel72]* Belady LA and Lehman MM, An Introduction to Program Growth Dynamics, in Statistical Computer Performance Evaluation, W. Freiburger (ed.), Academic Press, NY, 1972, pp. 503-511
- [ben00] Bennett KH and Rajlich VT, Software Maintenance and Evolution: A Roadmap, in Finkelstein, A. (ed.), The Future of Software Engineering, 22nd ICSE, Limerick, Ireland, Jun. 2000, ACM ord. no. 592000-1, pp. 73-87
- [car66] Carnap R, Philosophical Foundations of Physics, Basic Books Inc. 1966
- [feast2000] Preprints of FEAST 2000 International Workshop on Feedback and Evolution in Softw. and Business Processes, Ramil JF (ed.), Dept. of Comp., Imp. Col., London, 10-12 Jul. 2000, 124 pps. Available via links at <http://www.doc.ic.ac.uk/~mml/f2000> <July 2001>
- [hae01] Haeberer AM and Maibaum TSE, Scientific Rigour, an Answer to a Pragmatic Question: a Linguistic Framework for Engineering, ICSE 2001, Toronto, Canada, May 12-19, 2001
- [kem99] Kemerer CF and Slaughter S, An Empirical Approach to Studying Software Evolution, IEEE Trans. Softw. Eng., v. 25, n. 4, Jul./Aug. 99, pp. 493-509
- [kit99] Kitchenham B et al, Towards an Ontology of Software Maintenance, J. of Softw. Maint., v. 11, 1999, pp 365-389
- [leh69]* Lehman MM, The Programming Process, IBM Res. Rep. RC 2722, IBM Res. Centre, Yorktown Heights, NY, Sept. 1969.
- [leh74]* id., Programs, Cities, Students, Limits to Growth?, Inaug. Lect., May 1974, Imp.Col., Dept. of Comp., v. 9, 1970-74, pp. 211-229. Also in Gries D., (ed.), Programming Methodology, Springer, 1978, pp. 42-62
- [leh85] Lehman MM and Belady LA, Program Evolution - Processes of Software Change, Academic Press, London, 1985
- [leh89] Lehman MM, Uncertainty in Computer Application and its Control through the Engineering of Software, J. of Softw. Maint., Research and Practice, v. 1, 1 Sept. 1989, pp. 3-27
- [leh90] id., Uncertainty in Computer Application, Technical Letter, Comm. ACM, v. 33, n. 5, pp. 584, May 1990
- [leh94] id., Feedback in the Software Evolution Process, Keynote Address, Proc. CSR Eleventh Annual Wrksh. on Softw. Ev. - Models and Metrics. Dublin, 7-9th Sep. 1994, Also in Info. and Softw. Tech., spec. iss. on Software Maint., v. 38, n. 11, 1996, Elsevier, 1996, pp. 681-686
- [leh96] Lehman MM and Stenning V, FEAST/1: Case for Support Part 2, Dept. of Comp., Imp. Col., London, UK, Mar. 1996. Available via links at [feastwww]
- [leh97] Lehman MM, Laws of Software Evolution Revisited, EWSPT96, Oct. 1996, LNCS 1149, 1997, pp. 108-124
- [leh98] id., FEAST/2: Case for Support Part 2, Dept. of Comp., Imp. Col., London, UK, Jul. 1998. Available from [feastwww]
- [leh00b] Lehman MM, Approach to a Theory of Software Process and Software Evolution, FEAST 2000 Pre-prints, Imp. Col., London, 10-12 Jul. 2000, and with Ramil JF as Towards a Theory of Software Evolution - And Its Practical Impact, invited lecture, in Katayama T et al. (eds.) Proceedings ISPSE 2000, Intl. Symp. on the Principles of Software Evolution, Kanazawa, Japan, 1-2 Nov. 2000, pp. 2 - 11, IEEE CS Press
- [leh00c] Lehman MM and Ramil JF, Software Evolution Phenomenology and Component Based Software Engineering, IEE Proc. Softw., sp. issue on Component Based Software Engineering, v. 147, n. 6, Dec. 2000, pp. 249 - 255, earlier version as Tech. Rep. 98/8, Imperial College, London, Jun. 1998
- [leh00d] Lehman MM, An Approach to a Theory of Software Evolution - TheSE, EPSRC project proposal, DoC, Imperial College, Dec. 2000
- [leh01a] Lehman MM, Rules and Tools for Software Evolution Planning and Management, FEAST 2000 Pre-prints, Imp. Col., London, 10-12 Jul. 2000. A revised version with Ramil JF to appear in Annals of Softw. Eng., vol. 11, special issue of Softw. Management, 2001
- [leh01b] Lehman MM (with Ramil JF), Software Evolution, keynote lecture, in this volume
- [mai00] Maibaum TSE, Mathematical Foundations of Software Engineering: a Roadmap, in A. Finkelstein (ed.), The Future of Software Engineering , ICSE 2000, June 4-11 Limerick, Ireland, ACM ord. no. 592000-1, pp. 161-172
- [nato69] Naur P and Randell B (eds.), Software Engineering, Report on a conference sponsored by the NATO Science Committe, Garmisch, 7-11 Oct. 1968, Jan. 1969, 231 pps.
- [shr90] Shrager J and Langley P (eds.), Computational Models of Scientific Discovery and Theory Formation, Morgan Kaufmann Pubs., Inc, San Mateo, CA, 1990, 498 pps.
- [tha92] Thagard P., Conceptual Revolutions, Princeton Univ. Press, Princeton NJ, 1992 pp. 285
- [tur87] Turski WM and Maibaum T, The Specification of Computer Programs, Addison Wesley, UK, 1987, 278 pps.
- [wir71] Wirth N, Program Development by Step-wise Refinement, Comm. ACM, v. 14, n. 4, Apr. 1971, pp. 221-227