

Software Evolution: From Observations to Theory

Seminar
University of Sannio, Benevento, Italy

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>

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?*

S-type

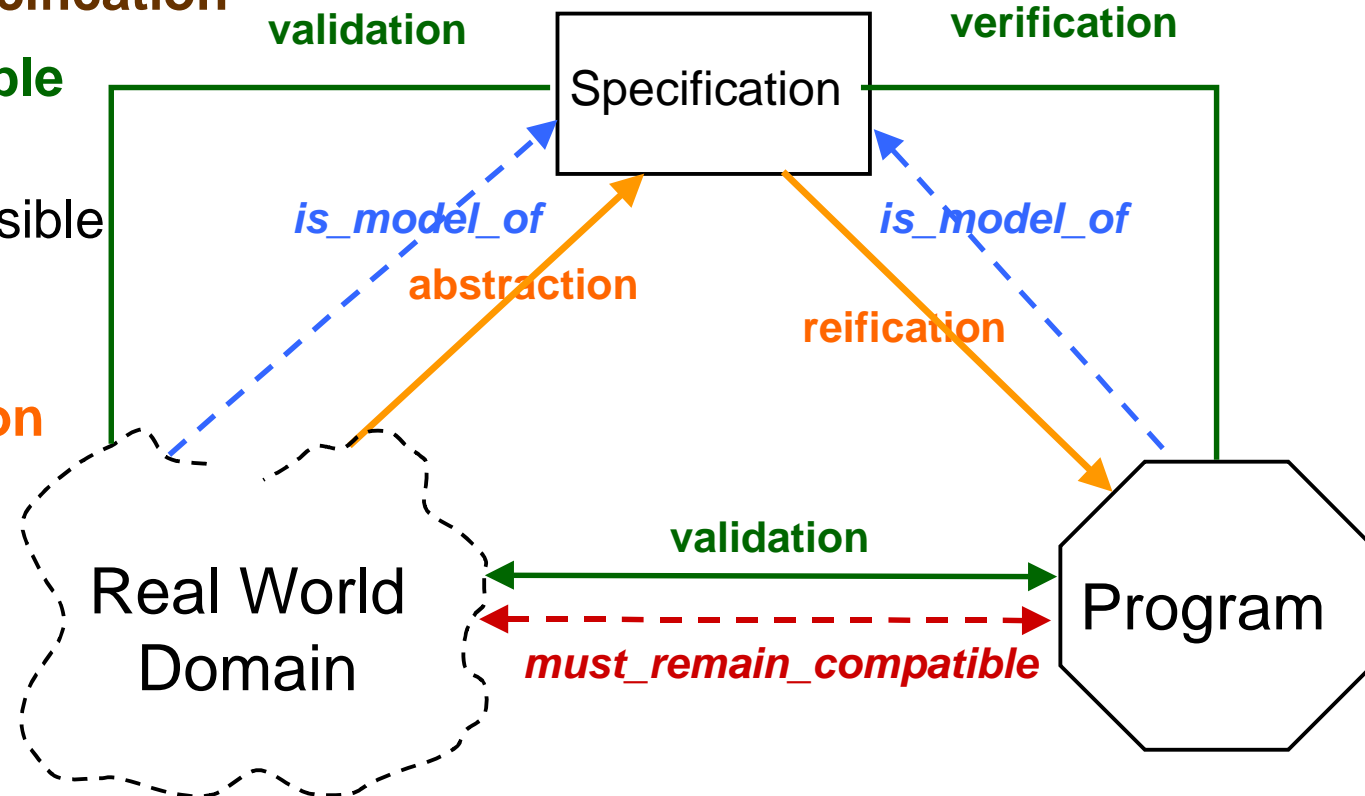
Definition, Properties

- 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
- System **validation** a matter of opinion/judgement assumption - **beauty contest**
- Fixing a program found to be **unsatisfactory** in use requires **change to specification** followed by derivation of **new program**, **new** by **definition**
- could well be obtained by modification of original code
- **S-type** programs the **bricks** for **construction** of **E-type** systems

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** relative to **real world meaningless**, **cannot** be **demonstrated**
- **Behaviour** and/or **results** of **execution** determine **acceptability**
- But system must **remain** satisfactory **model** of application in **operational domain**
- **Satisfaction** of **stakeholders' current needs** is ultimate **criterion** of **validity**
- **Needs, desires, opportunities**, operational **domain all evolve**
 - system must satisfy **current needs, interests** of **application, operational domain, stakeholders**
- **Intrinsic** need for **continuing application, system evolution**

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

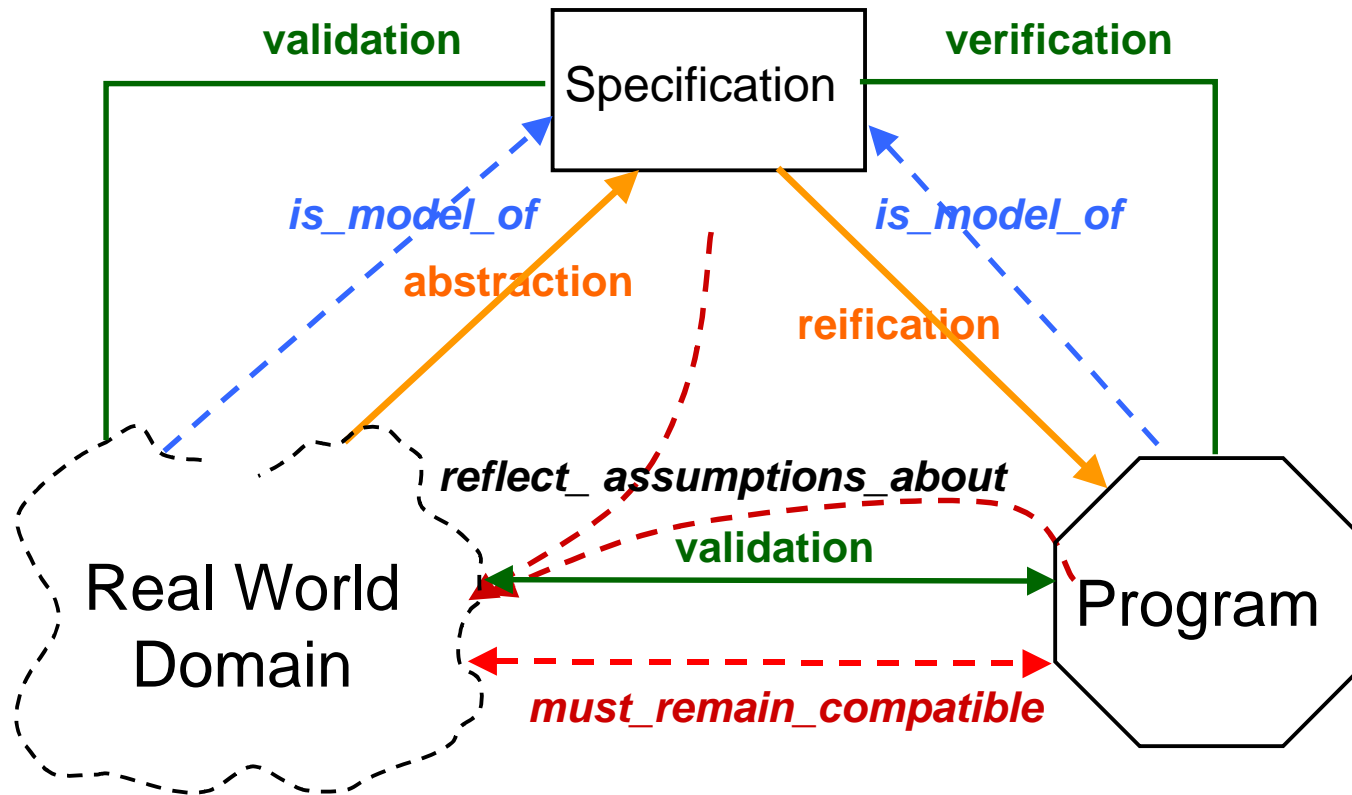
Real World/*E*-type System Relationship

- Previously concluded that **program** is a **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 **attributes**
- **Application** is equally - **potentially** - **unbounded** in its **attributes**
 - 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**

E-type Program as a Model

- Previous view of the **real world/specification/program** relationship



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

Program reflects real world

The System – Real World Gap

- An **E-type system** is **bounded reflection** of application in **operational domain**
- **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**

E-type System Evolution Process

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



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



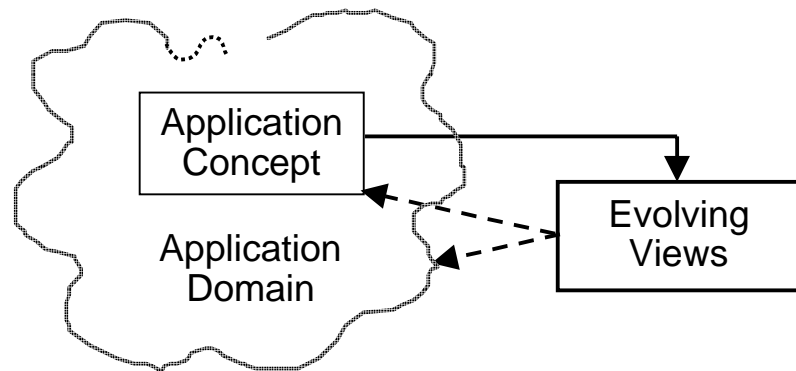
Yields - **new** or **changed application concept** (high level **requirement**)
and/or - **new** or **changed application domains**

- **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 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**

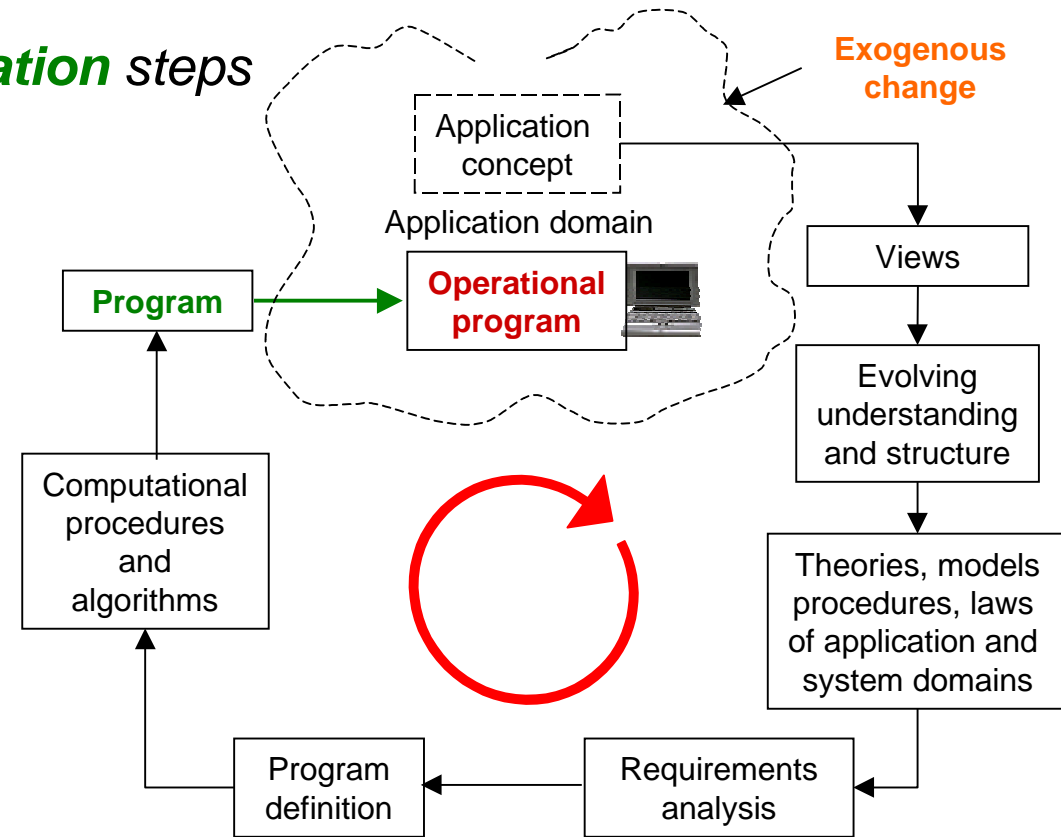


First step in *E*-type **Evolution** - **Development** and **Maintenance** - **Process**

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

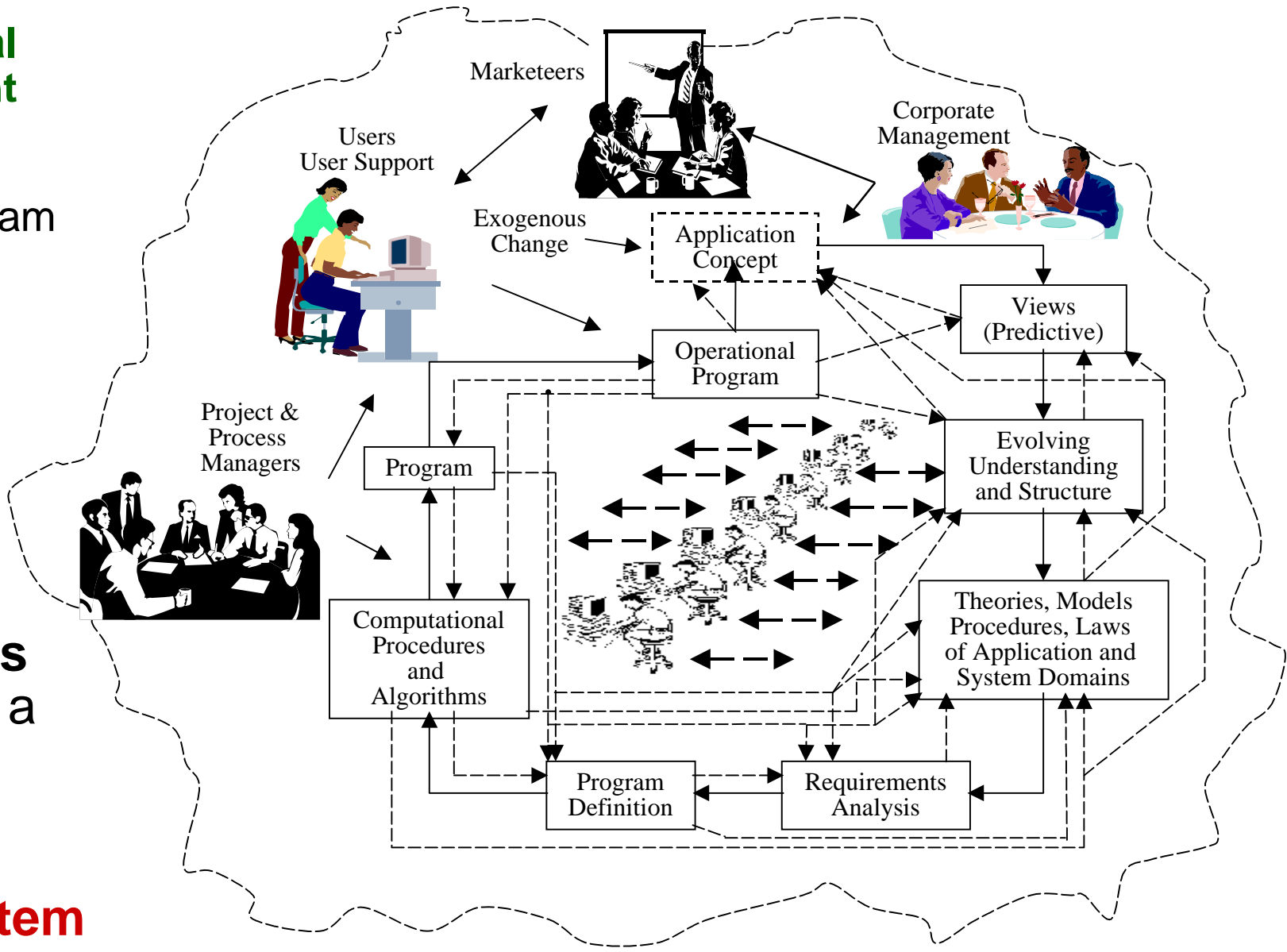


Driver of **continuing** software **evolution**

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** as **empirical generalisations** on **software evolution**

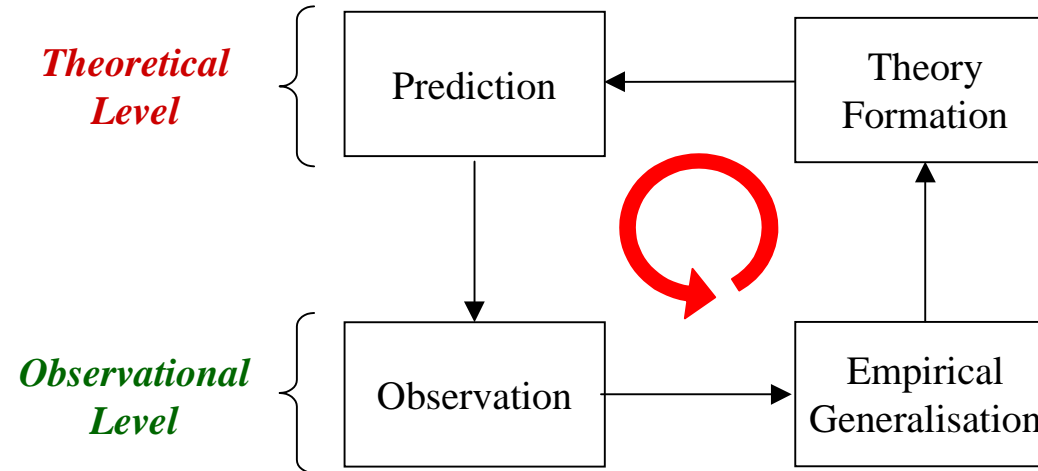
Also known as ***laws of software evolution***

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

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

Apply approach to develop a theory of **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**
- Provide foundations for 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
- Theory has **practical implications** in the form of **rules**, **tools** and **guidelines**
- Detailed discussion in **Annals of Software Engineering**, special issue on Software Management, Spring 2001
- Current status: **awaiting funding**

To what **observations** do these conclusions **relate**?