Software as Learning: Quality Factors and Life-Cycle Revised

José Hernández-Orallo

M^{a}José Ramírez-Quintana

Universitat Politècnica de València
Departament de Sistemes Informàtics i Computació
Camí de Vera s/n, E-46022, València, Spain.
E-mail: {jorallo,mramirez}@dsic.upv.es.

Foundational Aspects of Software Engineering (FASE 2000)
European Joint Conferences on Theory of Practice of Software (ETAPS 2000)
Berlin, 27 March - 1 April 2000

Introduction

Classical View of SW Engineering:

$\boxed{\text{“From specification to final product”}}$

Requirements Engineering:

$\boxed{\text{“From needs to specification”}}$

However, this does not equal to:

$\boxed{\text{“From needs to final product”}}$

- Requirements must constantly be revised.

Classical Goals

Minimise the Maintenance Cost:

$$\boxed{f\,(\text{modifiability, modification prob.})}$$

Modifiability has been addressed by:

- Evolution of Software Technology: structure, modularity, encapsulation, polymorphism…
- *New* banners: Adapt(at)ive and Intelligent Software.

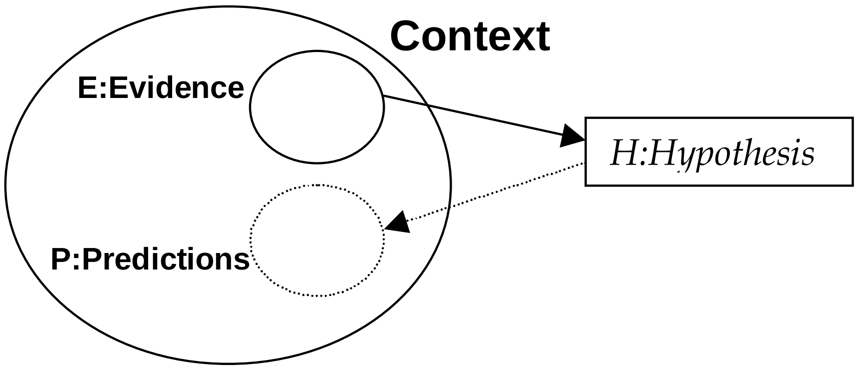
Modification probability has been less studied from a SW engineering point of view.

Classical Tools

However, the issue has been thoroughly addressed by the *sciences of induction*:

- Machine Learning (ML).
- Philosophy of Science (Ph. Sc.).

Induction:



Especially for the selection of hypotheses.

Background from ML & Ph. Sc.

Hypotheses Evaluation:

- verisimilitude (consistency with E).
- predictability (extrapolability to P).

Some evaluation criteria:

- Bayesian approaches (if the prior is known).
- Occam's Razor (MDL principle).
- Consilience, coherence or intensionality.

If the evidence is given incrementally:

Incremental learning:
H must be revised due to:

- anomalies: new evidence is wrongly covered.
- novelties: new evidence is not covered.

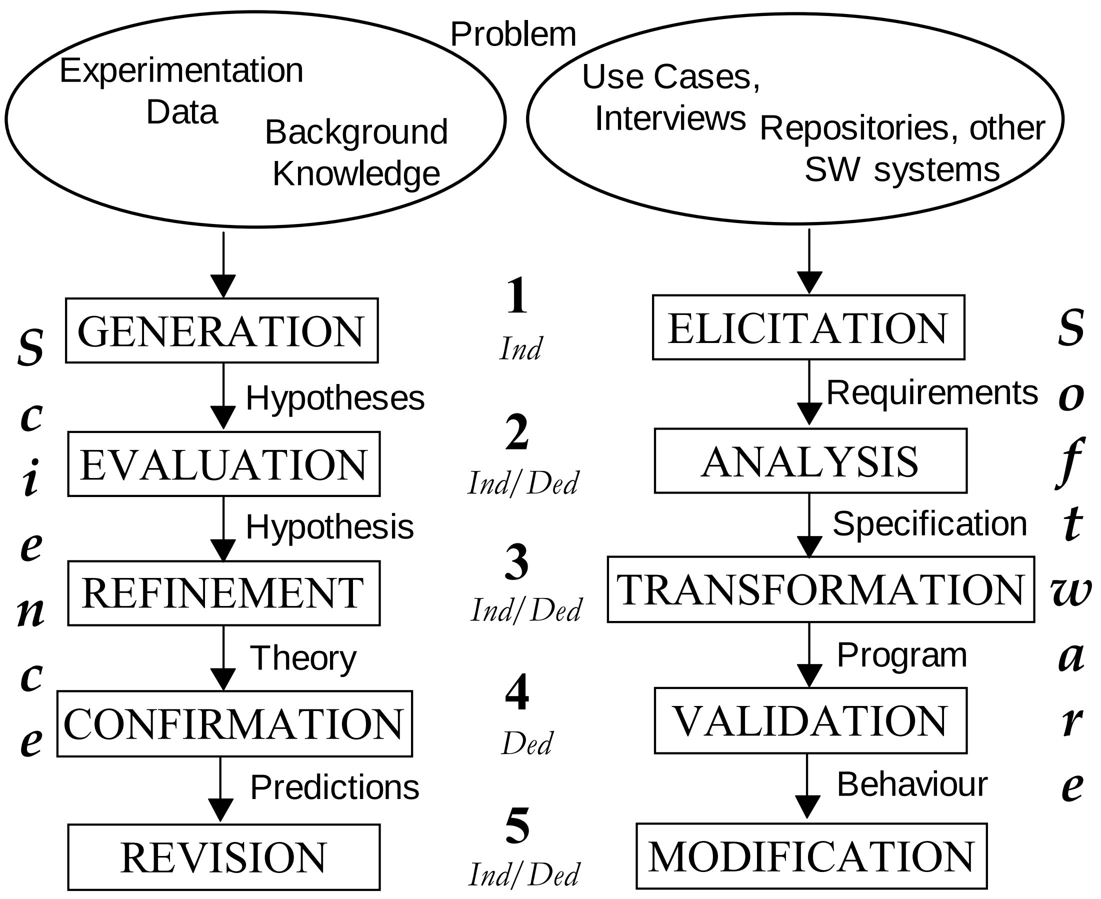
Programs as Scientific Theories

New Analogy:

Science		***Programming***
reality	↔	requirements context
problem	↔	problem
experimentation data	↔	cases / interviews / scenarios
construed evidence	↔	requirements
evaluation	↔	analysis
best hypothesis	↔	specification
validated subhypothesis	↔	prototype
refinement	↔	transformation
theory	↔	program
verisimilitude	↔	correctness
anomalies	↔	exceptions
confirmatory experiments	↔	testing
confirmation	↔	validation
revision	↔	modification
background knowledge	↔	SW. repositories / components
technical books	↔	technical/programmer's doc.
science text books	↔	user documentation

Comparison of Stages

Deduction and Induction have the same role in each side of the analogy:



Revision of SW Quality Factors

The main factors are defined in terms of “the specifications” or “requirements”:

- ***Functionality***: the degree to which a system “satisfies stated or implied needs”.
- ***Completeness***: degree to which a system implements all required capabilities.
- ***Correctness***: degree to which software *meet specified requirements* (classical view) or *meet user needs and expectations,* whether specified or not (modern view).
- ***Reliability***: “the ability to perform its required functions under stated conditions”.
- ***Robustness***: “the degree to which a system functions correctly in the presence of invalid inputs or stressful environmental conditions”.

Predictiveness

Software quality is evaluated assuming that specifications are perfect.
This is almost never the case.

The previous factors depend on how good the requirements elicitation has been made in order to know how accurate the factors can evaluate.

Let us measure this primary factor:

> **SOFTWARE PREDICTIVENESS**
> Predictiveness is the degree to which the software system predicts present and future requirements in the context where the requirements are originated.

Functionality as Predictiveness

Under the analogy, predictiveness matches with *functionality*, which includes:

- *correctness* (prediction for normal situations),
- *robustness* (prediction for environment or abnormal situations),
- *reliability* (minimisation of anomalies), and
- *completeness* (minimisation of novelties).

Predictiveness and Maintenance

Prediction Errors:

- Lack of reliability → modifications.
- Lack of completeness → extensions.

modifiability: easiness to make modifications and extensions (scope of each change).

Maintainability

$$\boxed{f\,(\text{modifiability, predictiveness})}$$

An interesting question is whether and how modifiability and predictiveness are related.

Implications: Validation

- Predictiveness covers all life-cycle errors: elicitation errors, design errors or implementation errors.
- Validating a software system with respect to the specification is neglecting part of the possible errors.
- Predictiveness includes any error since any error change the model of the hypothesis, consequently changing predictiveness.

Implications: Reusability and Modifiability

- Reusability: coherence and simplicity.
  - Uniform coverage (generality).
  - Avoidance of extensional patches.

  These features are included in the evaluation criterion called intensionality in Ph. Sc.

- Modifiability: redundancy must be avoided.
  - Compression is a criterion for predictive models. However, extremely compressed models are cryptical.
  - But there are many infinite non-redundant models.

  The notion of avoidance of redundancy makes modifiability and predictiveness compatible.

This is the explanatory paradigm of ML-Ph.Sc.

Software Development as an Incremental Learning Session.

New *adaptive* and *intelligent* software systems include revision techniques from ML and non-monotone reasoning. After each error, the system modifies itself (revises its model).

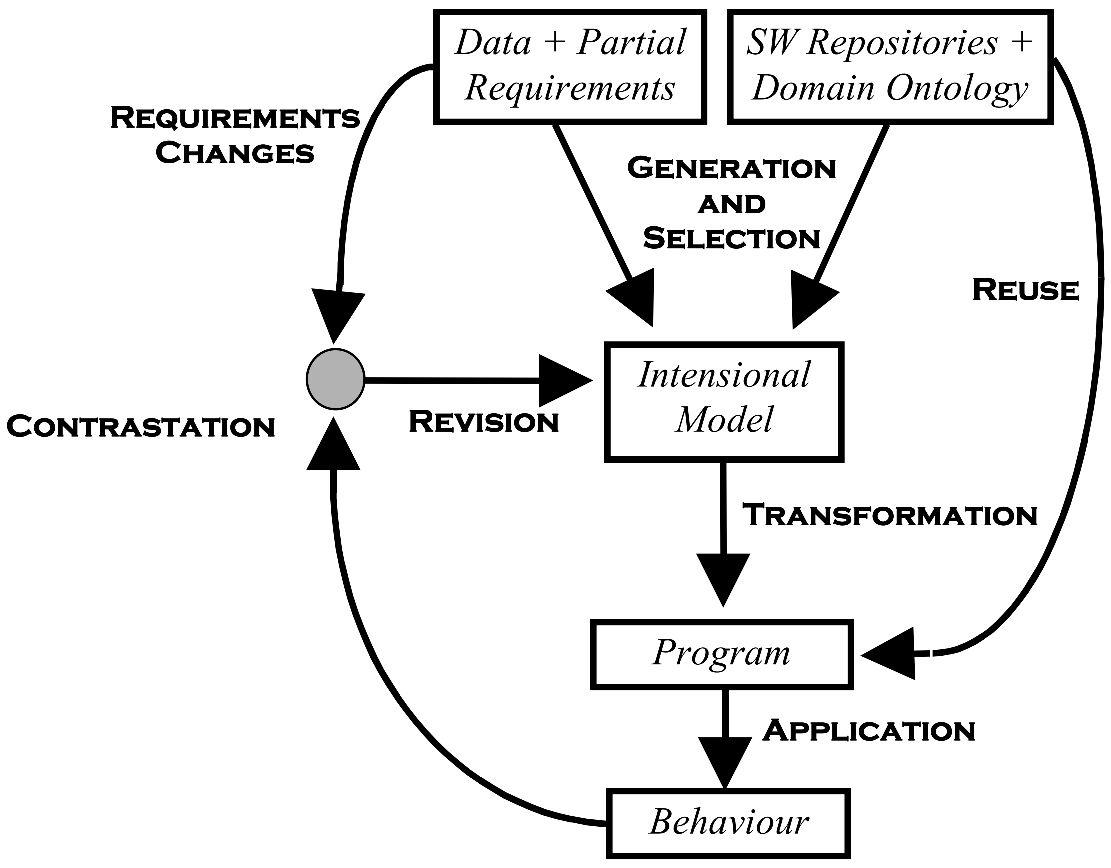
However, this is far from new if:

Software Development is seen as an Incremental Learning Session.

The relevance is now put on the inductive phases (generation and selection) and revision.

Predictive Software Cycle

By using a combination of terms and cycles from ML-Ph.Sc and Software Engineering:



On Automation

The previous analogies and cycle highlight that the automation of software development relies on the automation of induction.

GOOD NEWS:

The selection of hypotheses can be made automatically. Evaluation criteria can and should be applied to the analysis stage.

BAD NEWS:

The generation of hypotheses has been addressed by ML, but,
ML is not yet prepared for addressing so complex problems such as those software engineering deals with.

Towards Automation

Two possible (non-exclusive) ways:

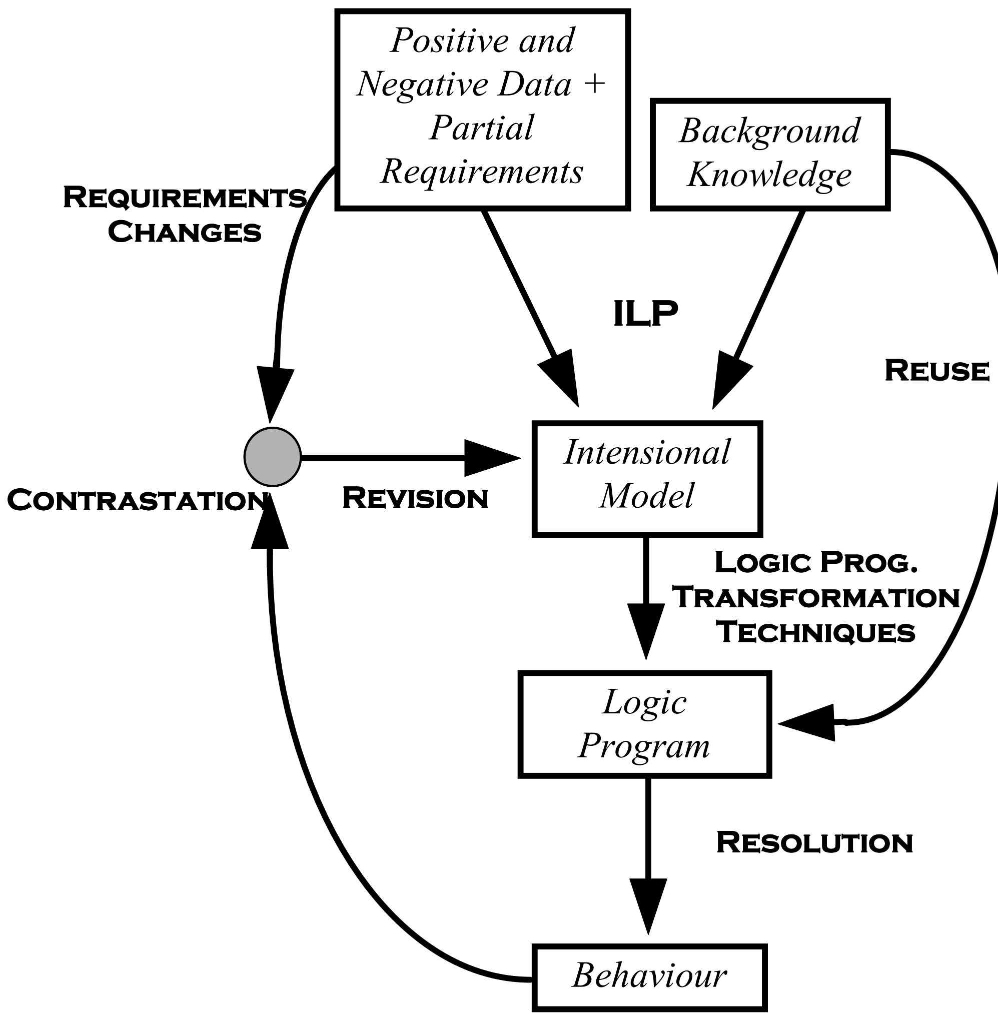
- Fully automatised development for *simple systems*. Only descriptional languages where the previous phases have been automatised (generation, transformation, revision).

- Increase the degree of partial automation of complex systems. For this, comprehensibility is indispensable. Only comprehensible descriptional languages can be used.

Consequently, whatever the approach, only declarative languages, where induction has been developed, can be used.

Presently, only (functional) logic programming is prepared for this.

Predictive LP cycle



Conclusions and Discussion

View of program as scientific theories.

Software life-cycle	=	Incremental learning session

The goal is to construct predictive software, in order to reduce the number of modifications.

However, modifiability must also be preserved. In this sense, evaluation criteria that support predictive hypotheses with easiness of revision are preferred over very compressed models.

The analogy clearly shows that until induction could be fully automatised for complex problems, automated software development will be a fallacy.

Nonetheless, evaluation can be automatised.

Current and Future Work

- The compromise between predictiveness and modifiability has been studied theoretically for different software topologies.
- Development of more powerful induction systems (e.g. the system FLIP) (`http://www.dsic.upv.es/~jorallo/flip`):
  - is able to induce recursive functional logic programs from examples and BK.
  - is designed for being used jointly with other automated stages of functional logic programming (transformation).
- Other paradigms of ML can also be applied to SW Eng. (query or interactive learning).
- Although the analogy is more general and plausible than preceding ones, more experimental support is necessary.