In memory of Professor Meir (Manny) Lehman

Nazim H. Madhavji
University of Western Ontario
Canada
madhavji@gmail.com

Manny’s contributions

Most know “Manny” as:

- the “Father of Software Evolution”

or as the proponent of:

- the “Laws of Software Evolution”.

Laws of Software Evolution

No.	Brief Name	Law
I 1974	Continuing Change	Unless continually adapted, an *E*-type system must decline in use and become ever more difficult to maintain satisfactory
II 1974	Increasing Complexity	As an *E*-type system is evolved its complexity increases unless work is done to maintain or reduce it
III 1974	Self-regulation	Global *E*-type system evolution is feedback regulated
IV 1978	Conservation of Organisational Stability	Rate of work of organisation evolving *E*-type software tends to be constant over phases of the operational lifetime of system
V 1978	Conservation of Familiarity	Growth rate trend of *E*-type systems constrained by need to maintain familiarity
VI 1991	Continuing Growth	Functional capability of *E*-type systems must be continually enhanced to maintain user satisfaction
VII 1996	Declining Quality	Quality of *E*-type systems declines unless rigorously evolved to take into account changes in the operational environment
VIII 1971, 1996	Feedback System (Recognised 1971, formulated 1996)	*E*-type evolution processes are multi-level, multi-loop, multi-agent feedback systems

Derived from observation and metrics

Provides base for empirical Theory of Software Evolution

Why the term "Laws"?

- Lehman's use of the term is often misunderstood.
- Unlike laws in sciences (e.g., physics), Lehman's laws do not specify precise invariant mathematical relationships between directly observable quantities, and were never intended to.
- Their purpose is to capture knowledge about the common features of frequently observed behaviour in evolving software systems.
- As this knowledge becomes more detailed and reliable, it is likely that future versions of the laws may be expressed in more precisely quantified terms.

Why the term "Laws"?

- Lehman's use of the term is similar to that of *social scientists* -- to describe general principles that are believed to apply to some class of social situation.
- For example, Say's Law in economics describes a general principle about the relationship between demand and supply, which may need to be modified when it is applied to particular situations.
- Since the theory of software evolution is similarly describing social situations that are extremely variable in practice, this use of the term 'law' is appropriate.

-- [Ch. 5 in Madhavji, F-Ramil, Perry: Wiley 2006]

On the term “Evolution”

- **Dictionary definition**:

“a gradual process of change and development.”

- **Lehman**:

‘a . . . process of discrete, progressive, change over time in the characteristics, attributes, [or] properties of some material or abstract, natural or artificial, entity or system or of a sequence of these [changes]’.

-- [Ch. 5 in Madhavji, F-Ramil, Perry: Wiley 2006]

Lehman’s definition of “Evolution”

- It captures important characteristics of evolution in many situations, including software systems.
- It is applicable to both natural and artificial systems, and to abstractions such as ideas.
- It provides a very general, universal definition of evolution that can be specialised for particular domains, such as software, natural languages and genes.

-- [see Ch. 5 in Madhavji, F-Ramil, Perry: Wiley 2006]

Darwin's definition of "Evolution"

- The idea of organic evolution was proposed by some ancient Greek thinkers but was long rejected in Europe as contrary to the literal interpretation of the Bible.
- Lamarck: a theory that organisms became transformed by their efforts to respond to the demands of their environment.
- Darwin: a theory of gradual evolution over a long period by the natural selection of those organisms slightly better adapted to the environment and hence more likely to produce descendants.

-- [Online Oxford dictionary]

Lehmanian vs. Darwinian definitions

- Lehman leaves it open as to who actually does the "evolving". For example:
  - Humans can change the software system themselves (e.g., through modifications).
  - Software's behaviour changes due to automatic recognition of conditions by autonomic systems (constrained/guided by policies and learning mechanisms).

- Darwin focuses on natural selection.

Evolving Laws

- Lehman's Laws capture knowledge of the time but are open for refinement and improvement:

'Though termed "laws" , always recognised as initial hypotheses subject to change ....'

[Int. Workshop on Software Evolvability, 2005, Budapest]

Our History: excerpts from Proc. 1968 NATO SE Conf., (eds.) P. Naur and B. Rendall.

- In Autumn 1967, a Study Group on Computer Science, established by the NATO Science Committee proposed a working conference in 1968 -- the NATO Conf. On Soft. Eng. (Garmisch, Germany, 7th to 11th October 1968).
- The following were the three key concerns:
  - Design of Software
  - Production of Software
  - Service of Software

Life-cycle Strategy

- Today we tend to go on for years, with tremendous investments to find that the system, which was not well understood to start with, does not work as anticipated.
- We build systems like the Wright brothers built airplanes build the whole thing, push it off the cliff, let it crash, and start over again.

  – Professor R.M. Graham, Project MAC, M.I.T. Cambridge, Massachusetts, USA., (1968 NATO SE Conf.)

12 3. Software Engineering

20

Maintenance and Follow-up

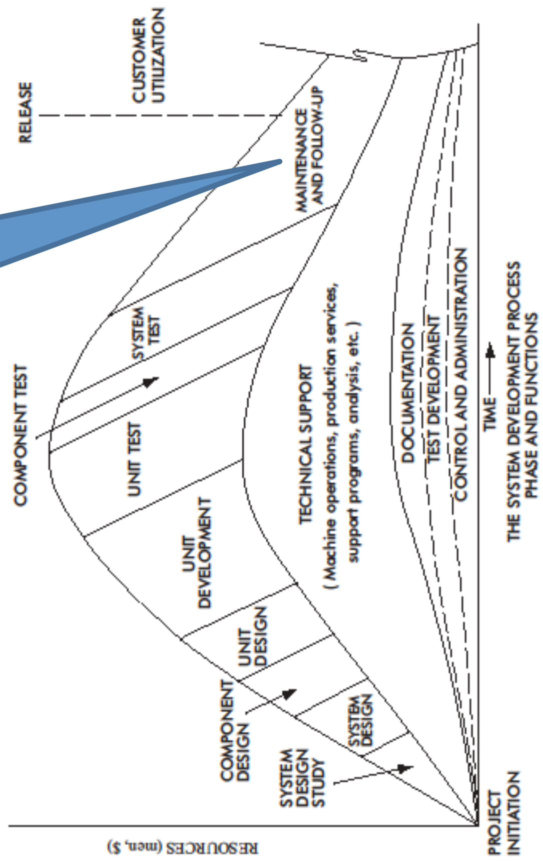


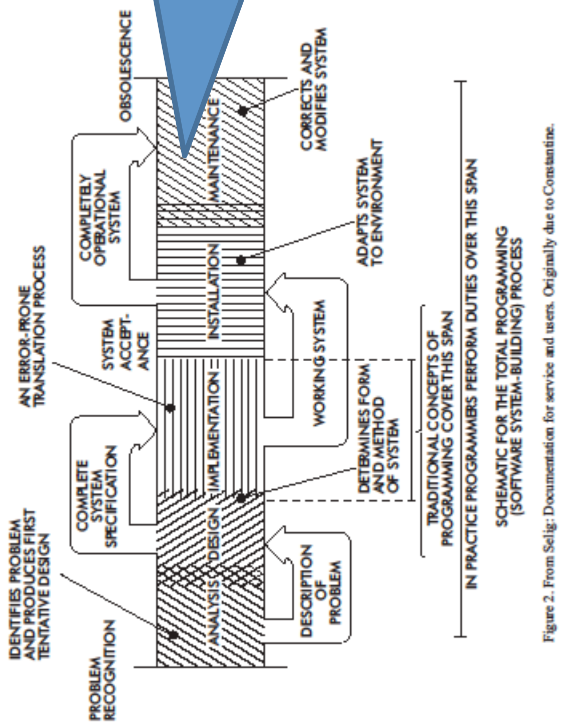
Figure 1. From Nash: Some problems in the production of large-scale software systems.

Mr. J. Nash, IBM UK Laboratories, Hursley Park, England.

NATO SOFTWARE ENGINEERING CONFERENCE 1968

Corrects and Modifies System

13



PROBLEM RECOGNITION
ANALYSIS
DESIGN
IMPLEMENTATION
INSTALLATION
MAINTENANCE
OBSOLESCENCE
IDENTIFIES PROBLEM AND PRODUCES FIRST TENTATIVE DESIGN
COMPLETE SYSTEM SPECIFICATION
AN ERROR-PRONE TRANSLATION PROCESS
SYSTEM ACCEPT-ANCE
COMPLETELY OPERATIONAL SYSTEM
DESCRIPTION OF PROBLEM
DETERMINES FORM AND METHOD OF SYSTEM
WORKING SYSTEM
ADAPTS SYSTEM TO ENVIRONMENT
CORRECTS AND MODIFIES SYSTEM
TRADITIONAL CONCEPTS OF PROGRAMMING COVER THIS SPAN
IN PRACTICE PROGRAMMERS PERFORM DUTIES OVER THIS SPAN
SCHEMATIC FOR THE TOTAL PROGRAMMING (SOFTWARE SYSTEM-BUILDING) PROCESS

Figure 2. From Selig: Documentation for service and users. Originally due to Constantine.

Dr. F. Selig, Mobil Research and Development Corporation, Dallas, Texas, USA.

NATO SOFTWARE ENGINEERING CONFERENCE 1968

Iterative Design

- The design process is an iterative one.
  - 1. Flowchart; 2. Write code; 3. Go back and re-do the flowchart; 4, Write some more code and iterate to what you feel is the correct solution.
- In a large production project, ... You know you are going to iterate, so you don't do a complete job the first time.
- This is why there is version 0, version 1, version N.
- If you are ...writing specifications, you don't have the chance to iterate, the iteration is cut short by an arbitrary deadline. This is a fact that must be changed.

Mr. H.A. Kinslow, Computer Systems Consultant, Connecticut, USA.

Correctness vs. Relevance

- The most deadly thing in software is the concept, which almost universally seems to be followed, that you are going to specify what you are going to do, and then do it.
- And that is where most of our troubles come from.
- The projects that are called successful, have met their specifications. But those specifications were based upon the designers' ignorance before they started the job.

-Mr. D.T. Ross, Electronic Systems Laboratory, M.I.T., Cambridge, Massachusetts, USA.

Problems of Scale of Systems

OS/360 growth:
1--5+ MLOC – 1966-68+

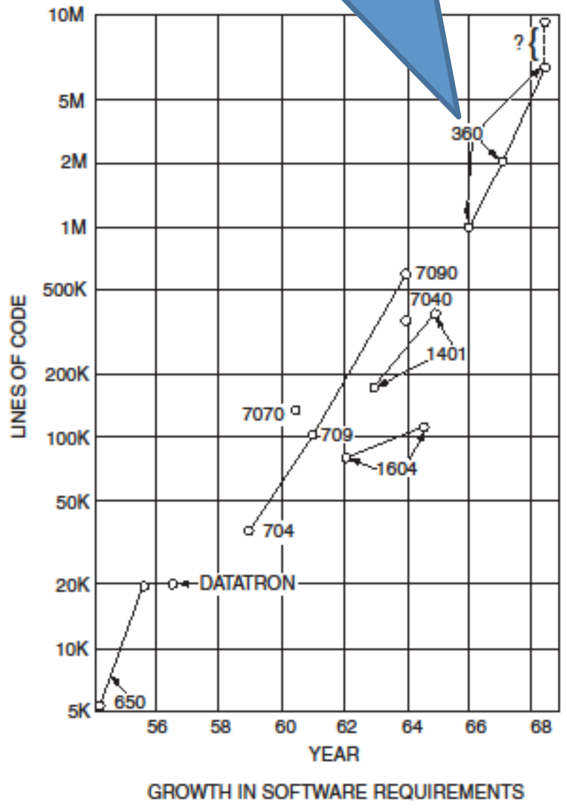


Figure 6. Provided by McClure

5. Production

The rate of growth of the size of different software systems (code) for a variety of computers. Logarithmic scale.

Note the OS/360 growth!

Dr. R.M. McClure, Southern Methodist University, Dallas, Texas, USA.

System Releases -- 1

- *Kolence*: Large systems must evolve, and cannot be produced all at one time.
- *Babcock*: Fewer releases, containing major functional improvements ... that work well are more desirable than frequent releases of versions containing only minor improvements.

System Releases -- 2

- *Opler*: The latest release of OS/360 was intended to introduce 16 changes (many complex), and to correct 1074 errors.
- Current policy is 90 day release intervals (i.e., over 11 corrections per day between versions).
- While it is obviously better to batch improvements, customers need quick responses. Once a year would be much too infrequent.

System Releases -- 3

- *Pinkerton*: With less frequent releases there would be increased stability, and more opportunity for users to generate responsible feedback .....
- *Galler*: OS/360 has had 16 releases in two and a half years. We should have frequent updates for corrections, but decrease the frequency of traumatic upheavals.

Maintenance

- *Gillette:* ... cost of maintenance frequently exceeds that of the original development.
- ... a basic OS [and system utilities] ... > 250 KLOC ... requires about 2-3 year effort. Maintenance ... an unending process which lasts for the life of the machine...
- In summary, then, the maintenance ...involves corrective code, improvement code, and extensive code.

User vs. Developer Responsibility

- *Babcock*: As a user, I think it is a manufacturer's responsibility to generate systems to fit a particular user's need, but I haven't been able to convince my account representative of that fact.
- *Kolence*: Users should ... ensure that it works in their environment. A large manufacturer cannot test out his software on all the environments in which it will operate. It is for this reason that manufacturers typically provide an on-site representative .....

Three Feedback Control Loops

- *Haller*: There is feedback of different entities, on different paths, leading to three separate control loops with different time-lag:
  - 1. ...correctness ... to the maintenance group; up to a week.
  - 2. ... performance, to the production group; within, say, a month.
  - 3. ... extra facilities to the design group; and, if accepted, a year.

Some observations

- *Focus in the late '60s was more on problems of the day:* Escalating costs, Completion, Reliability, Estimation and other management problems, Scale of systems, Tools, Design problems, Release quality, and system maintenance, etc.
- *Little focus on studying scientific properties* of: system growth (evolution), feedback control, etc., and impact on practice.

Single release vs. Multiple releases

- If there is anyone in our community who has been driving with the "high beams" on, ..., then in my mind this is ... Lehman.

- While most of us were either in our infancy or fire-fighting software problems, Lehman had his sight set far ahead – on the *programming process*, as evidenced by a seminal paper published in 1969 [1].

-- Madhavji, ICSM 2003 Panel session on Lehman's Laws.

[1] Lehman M.M. (1969), "The Programming Process", IBM Research Report RC 2722, IBM Research Centre, Yorktown Heights, NY, September.

Single release vs. Multiple releases

- For about thirty-five years now, Lehman has been concerned about, ..., software's long term health, ..., while most others...have had "low beams" on as if the next release is the final release of the software product or system.

- It needs no further explanation as to why we encounter surprises when we drive in the pitch-dark roads of software engineering.

-- Madhavji, ICSM 2003 Panel session on Lehman's Laws

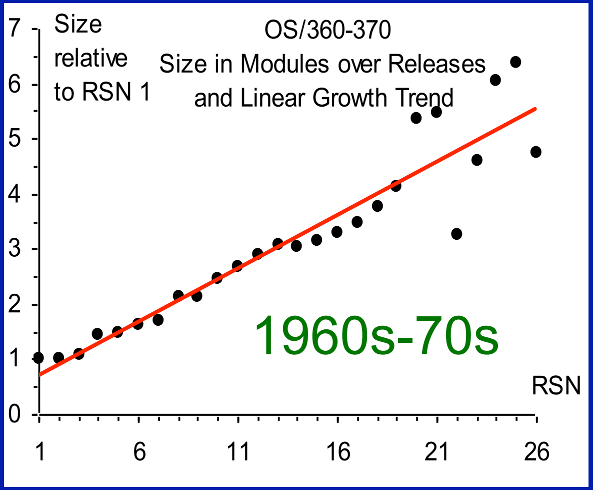
Empirical Approach

- The "Laws" and recognition of software process as a feedback system have roots in empirical observations as early as 1968-69 of IBM's OS/360-370.
- Later studies (Belady and Lehman 1972 and Lehman 1974, Lehman 1980, Lehman and Belady 1985) and the FEAST projects (1994-2002 with Turski, Perry and Ramil) involved other systems.
- This triggered studies of *Program Growth (or Evolution) Dynamics.*
- Refined versions of the Laws were subsequently proposed as tools for planning, management and control of sequences of releases (Belady and Lehman 1972, Lehman 1974, 1980).

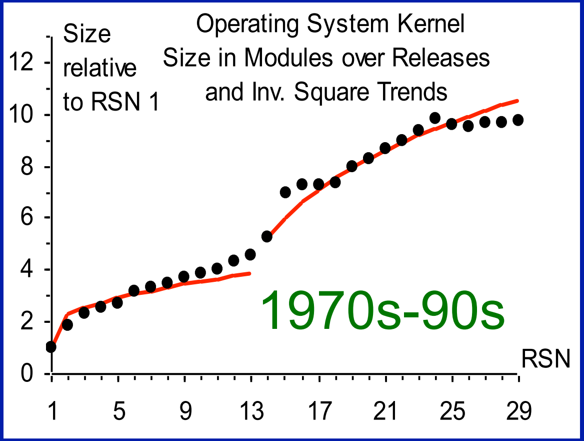
-- [see Ch. 1 in Madhavji, F-Ramil, Perry: Wiley 2006]

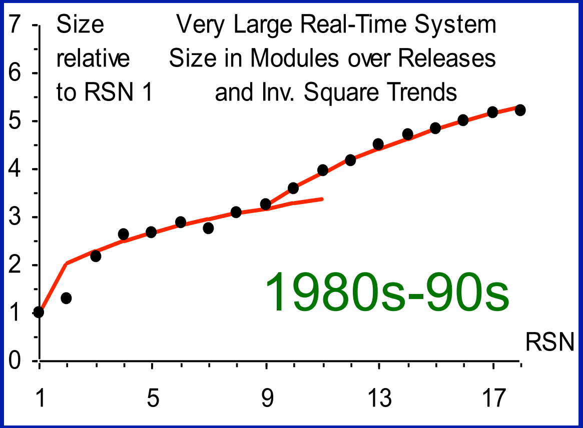
Continuing Growth - Linear or Inverse Square?

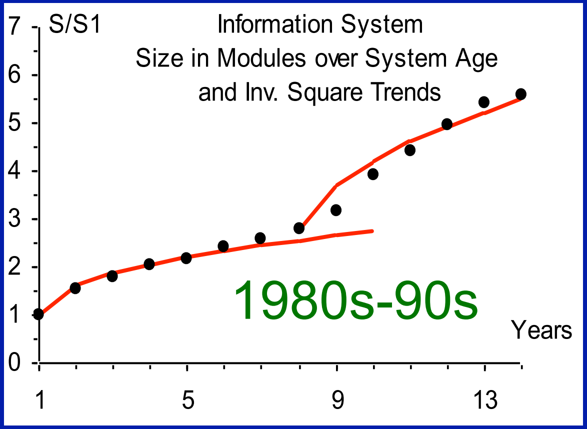
OS/360



- **Linear** growth to release 19
- **End result** - **instability** leading to **fission**
- **Trigger** - **excessive growth?**







- **Inverse square growth** : $S_{i+1} = S_i + \hat{E}/S_i^2 \quad (1<= i <= n-1)$
  - where ***i*** are **sequence numbers** (rsn) of the ***n*** **releases** for which **data** is available

- Suggests **complexity constrained** growth

Empirical Approach

- From the very start (1968-69) the study demonstrated that software evolution is a phenomenon that can be observed, measured and analysed, with *feedback* playing a major role in determining the behaviour.

- Though there was empirical evidence of system growth (McClure) and thoughts on feedback (Haller) and evolution (*Kolence*) at the 1968 NATO SE Conf., it was Lehman who relentlessly pursued to find the truth for over 35 years.

- This makes Lehman the "Galileo" of our time.

Communication with Vic Basili – 16 August, 2011

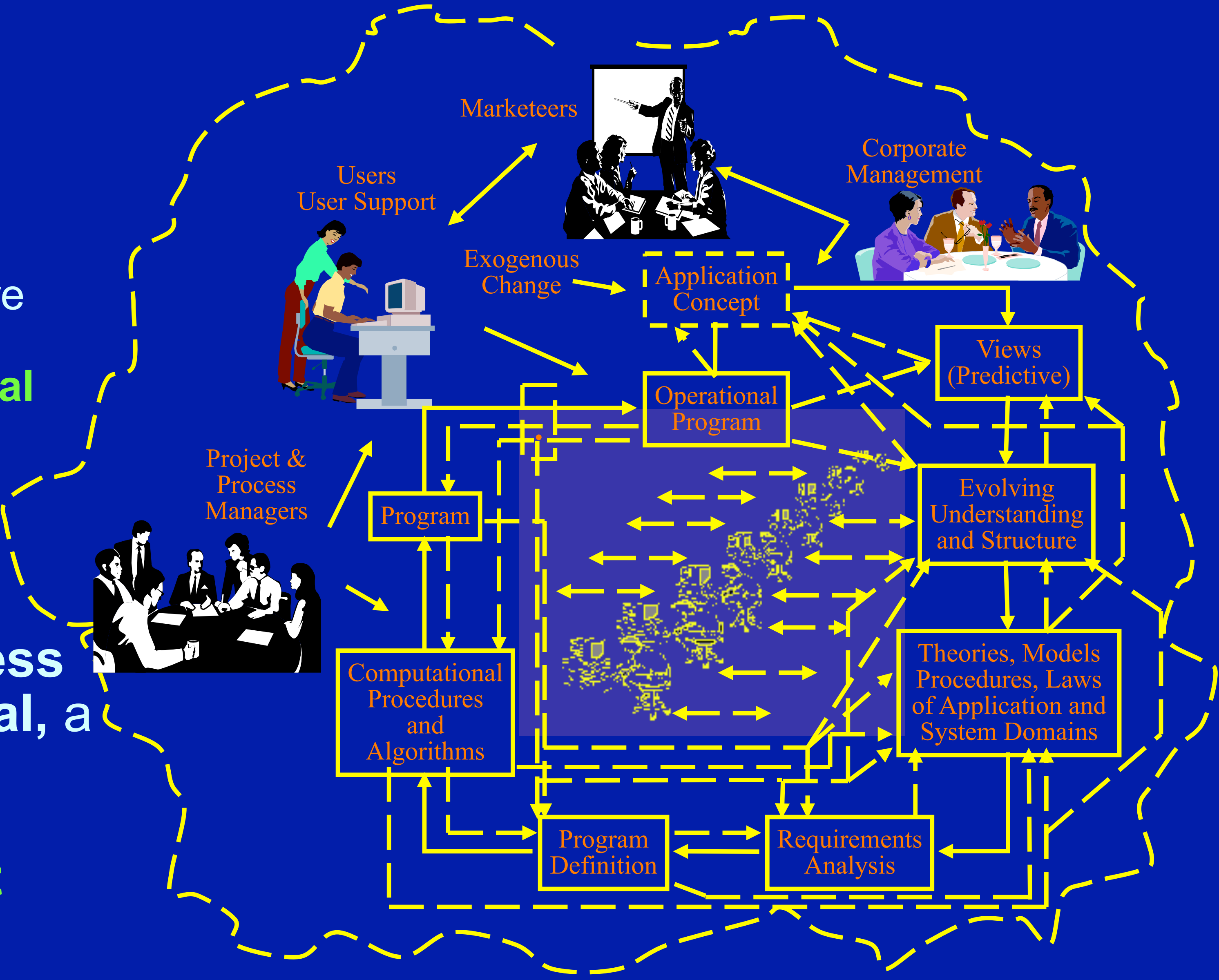
The Belady-Lehman work had a big influence on my thinking, leading me to be a proponent of observation.

- I remember Manny as a dedicated, driven observer, trying to identify rules and laws. He was the first one out there. He was committed and had an enormous amount of energy, even in his waning years. He coined and made real the concept of evolution of systems, rather than maintenance.

The Global Software Process

- Incorporates activity of **all** involved
- Process **not sequential**
- Involves more than just **technological activity**

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



Lehman’s Multi-level loops vs. Basili’s QIP

- There is a relationship between QIP and Manny's multi-level loops, but they are not the same.
- Manny was trying to connect to control theory while I was closer to Deming's Plan Do Check Act, with the distinction being that there were two loops in QIP.
  - The project loop and the learning loop which tried to create models with a sparsity of data, unlike the situation in manufacturing.
- In fact QIP is really about observing your own activities to see how to characterize, understand, evaluate, predict, and improve.

-- Vic Basili: 16 August, 2011

Global Software Process and Feedback

- Two broad levels of feedback in the process:
  - Product-level (e.g., code inspections) to developer.
  - Process-level (e.g., product-quality data, resource consumption, time consumed, etc.) to process group.
- Feedback can be used to improve the product or the process.

Regulatory vs. Step-function Feedback Control

- An interesting question is whether (say) product-feedback control is:
  - "regulatory" (as in a home thermostat), i.e., can be increased/decreased with a knob, or
  - a "step-function", i.e., difficult to "turn back" controllably once the process-change has been made.

[see Ch. 17 in Madhavji, F-Ramil, Perry: Wiley 2006]

SPE classification of systems

- E-type includes all programs that '*operate in, or address, a problem or activity of the real world*'.
- This type must be continually adapted and changed if it is to remain satisfactory in use.
- Very central to the notion of "evolution".

SPE classification of systems

- A program is defined as being of S-Type if it can be shown that it satisfies the necessary and sufficient condition that it is correct in the full mathematical sense relative to a pre-stated formal specification (Lehman 1980, 1982) – e.g., a function in home appliance.
- There is no good reason for changing it subsequently. The program cannot be improved since, by definition, it already completely satisfies its acceptance criteria.

SPE classification of systems

- In P-type systems, the satisfaction of its stakeholders depends on the system maintaining consistency with a single paradigm over the system's lifetime. (e.g., external standard to be followed).
- Thus, the evolution of a P-type system is constrained by the strategic decision of its stakeholders to keep the system consistent with a paradigm. For example:
  - It will prevent some kinds of change that might otherwise have occurred.
  - It may also induce change, either when the paradigm is updated or when opportunities arise, for example, through technological change, to improve the system's consistency with its paradigm.

[see Ch. 5 in Madhavji, F-Ramil, Perry: Wiley 2006]

Assumptions & Software Uncertainty

- Assumptions and uncertainty in software development are extremely difficult to manage.
- A major cause for system problems.

Fact of Life

- Real world is dynamic with changes in application and domains continual, inevitable
- Systems kept in tune by being evolved
- Adaptation attempts to regain any loss in validity, utility, performance, functionality - may in fact increase the last three
- Adaptation and extension is an inescapable, almost natural, phenomenon intrinsic to *E*-type systems
- *Effective direction, management, control of evolution, reduction of cost, impact, requires understanding of phenomenon and its causes*

Adapted from MML

Assumptions

- Unbounded number of real world (domain) properties must be rejected or ignored in system development and evolution.

- This leads to the principle of software uncertainty.

Adapted from MML

Principle of Software Uncertainty

No matter how often a system executed satisfactorily, satisfaction on its next execution is uncertain.

Adapted from MML

Sources of Uncertainty

- **Assumption** that excluded properties are irrelevant at desired level of precision and
  detail may **become** progressively **invalid** because of domain/application changes

- Similarly, previously **valid** assumptions about properties reflected in software may
  become invalid

- In general, application and domain **changes** may **falsify** assumptions reflected in
  system

- **Impact** and **consequences** will range from **trivial** to **disastrous**

Hypothesis: ***assumptions*** *that are or become invalid* ***major source*** *of project or computer* ***failure*** *during development and use*

Assumptions and their Implications

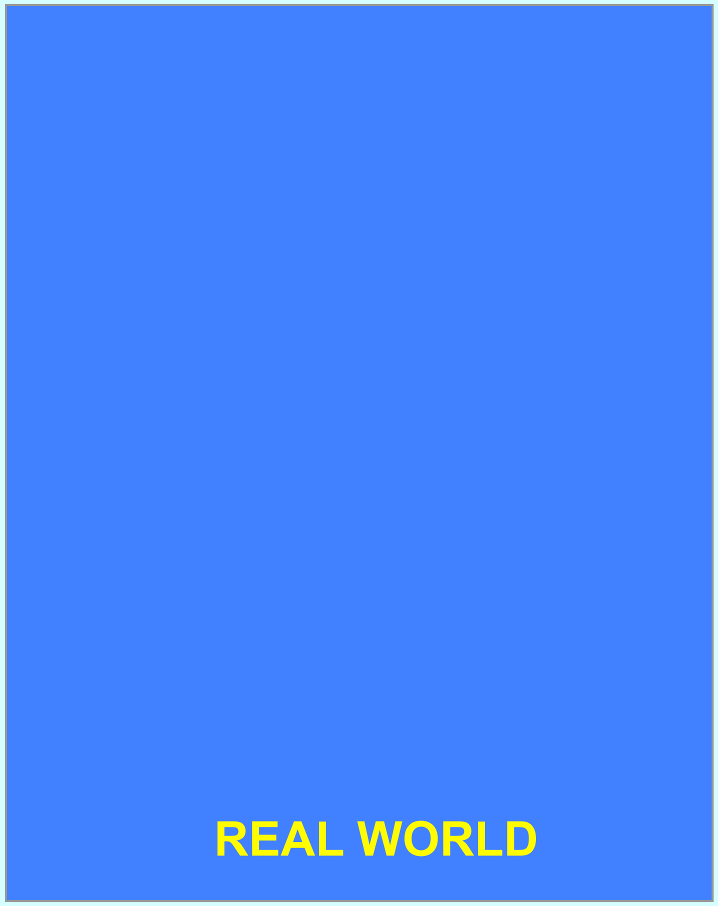
- Embedding assumptions during system development intrinsic to *E*-type systems
- Also, usage, itself, implies real world changes and hence drives application and domain evolution
- Continual monitoring of assumptions key to satisfactory (i.e., correct, safe, reliable, efficient, and controlled) computer usage

Assumption management vital for development and evolution management, stabilisation, and control.

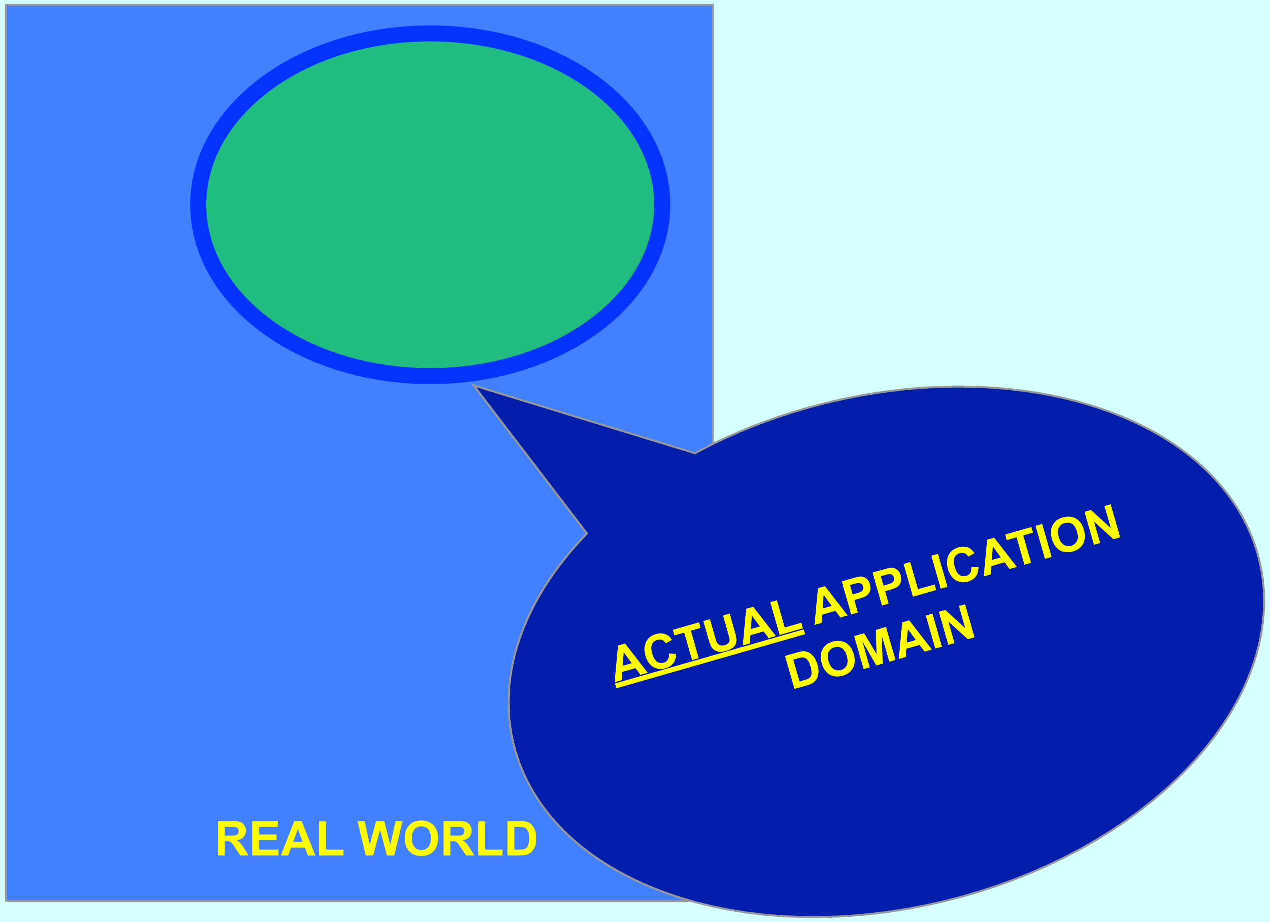
Adapted from MML

Software evolution, assumptions, and software uncertainty

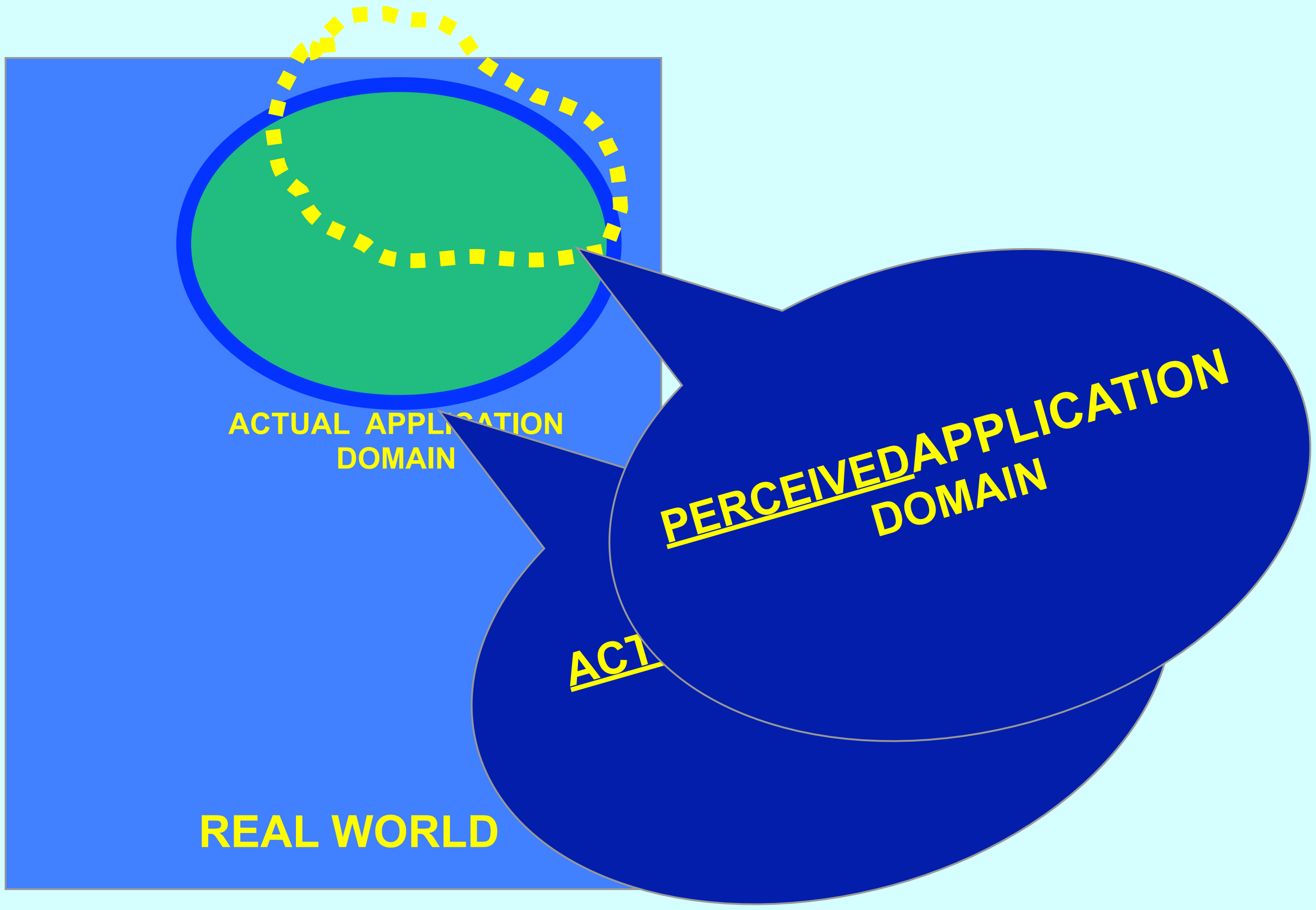
Software evolution, assumptions, and software uncertainty



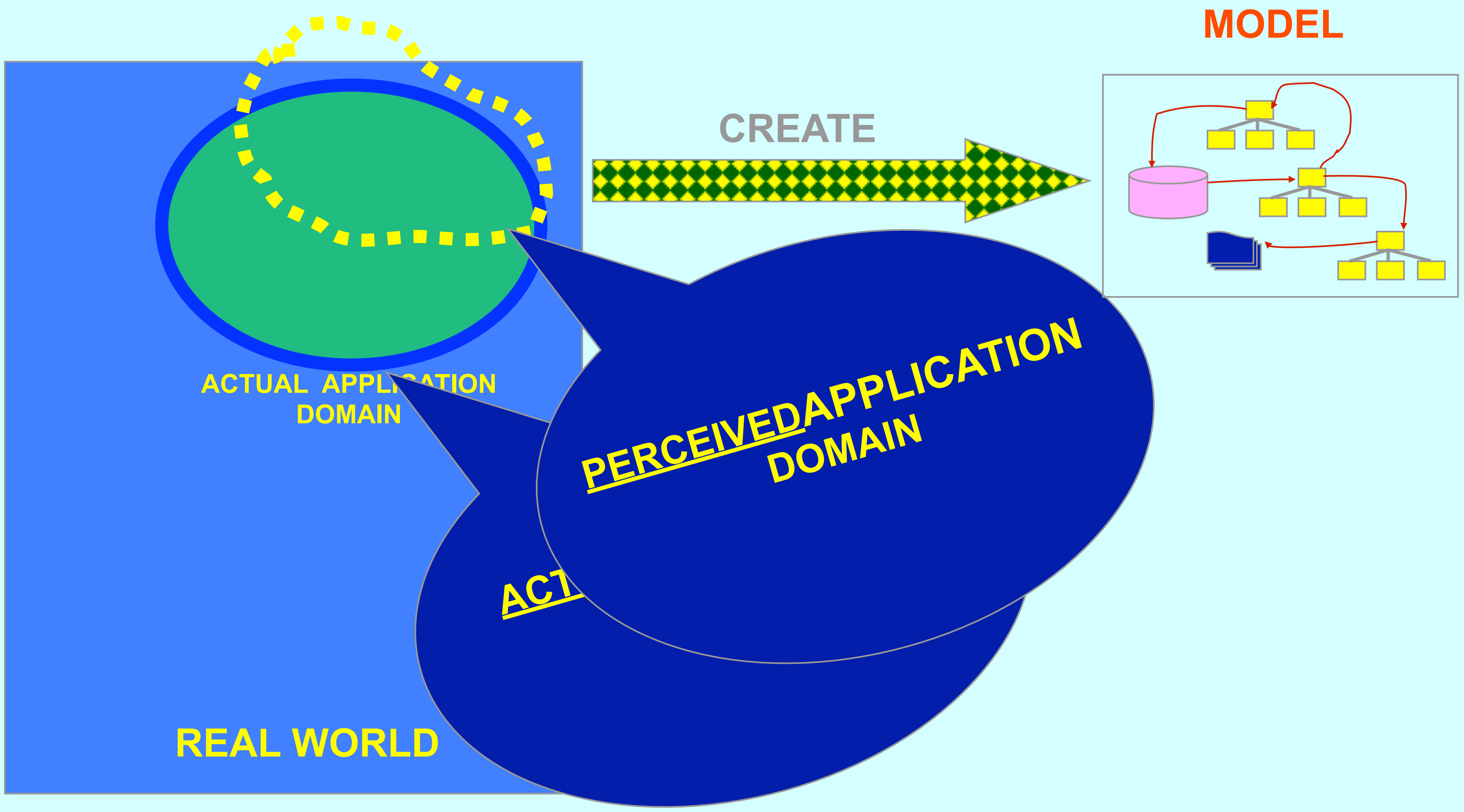
Software evolution, assumptions, and software uncertainty



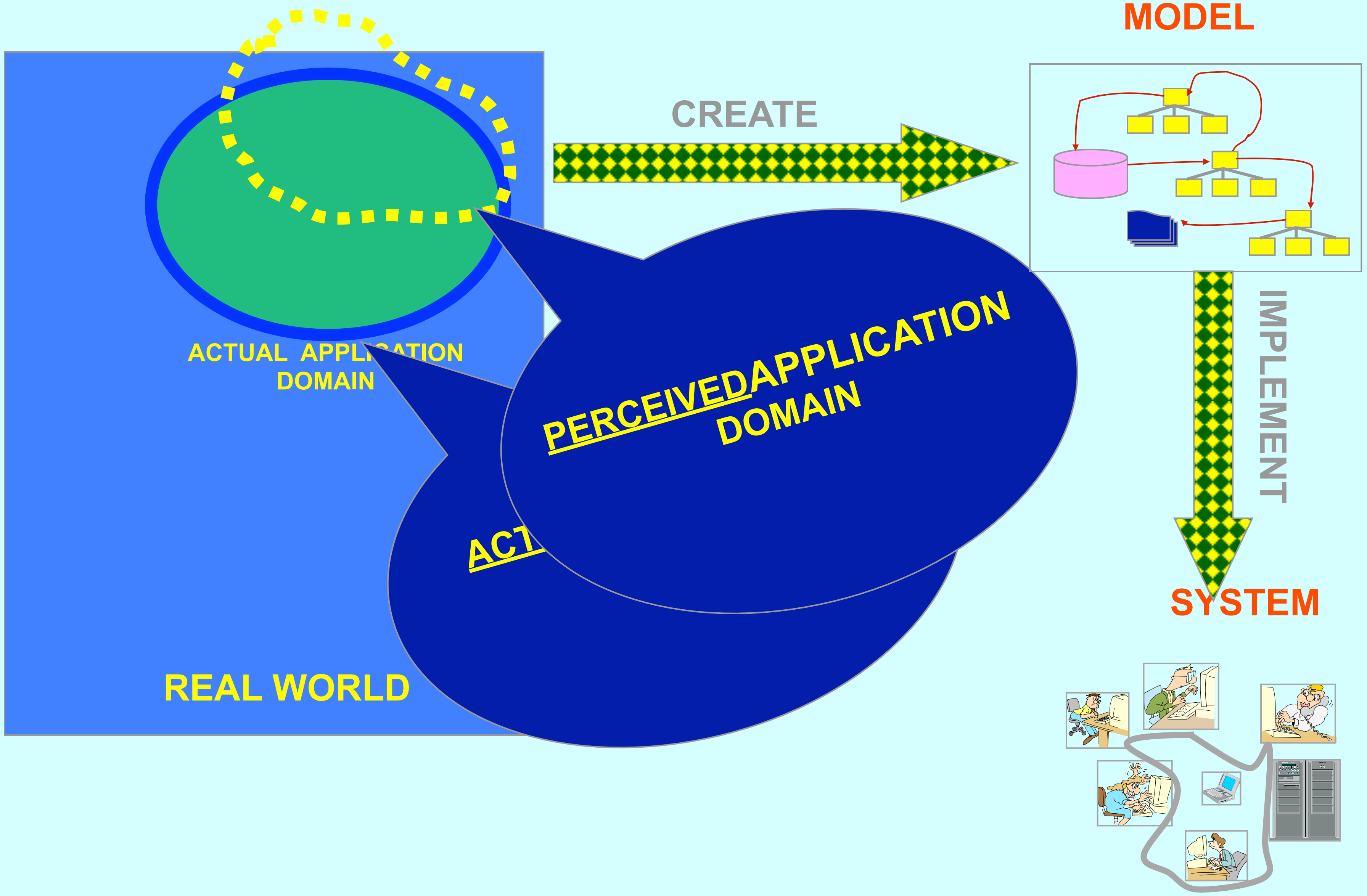
Software evolution, assumptions, and software uncertainty



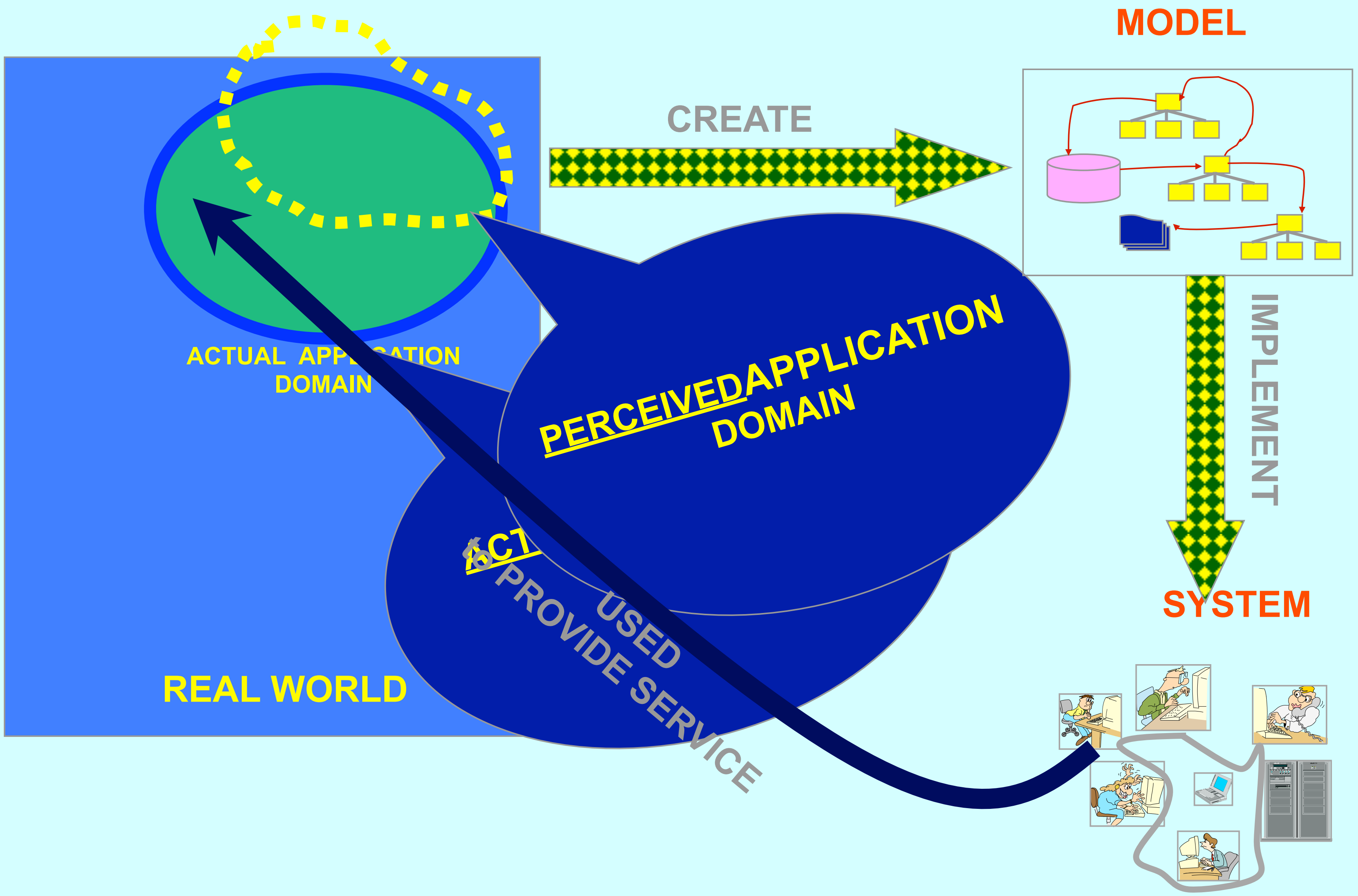
Software evolution, assumptions, and software uncertainty



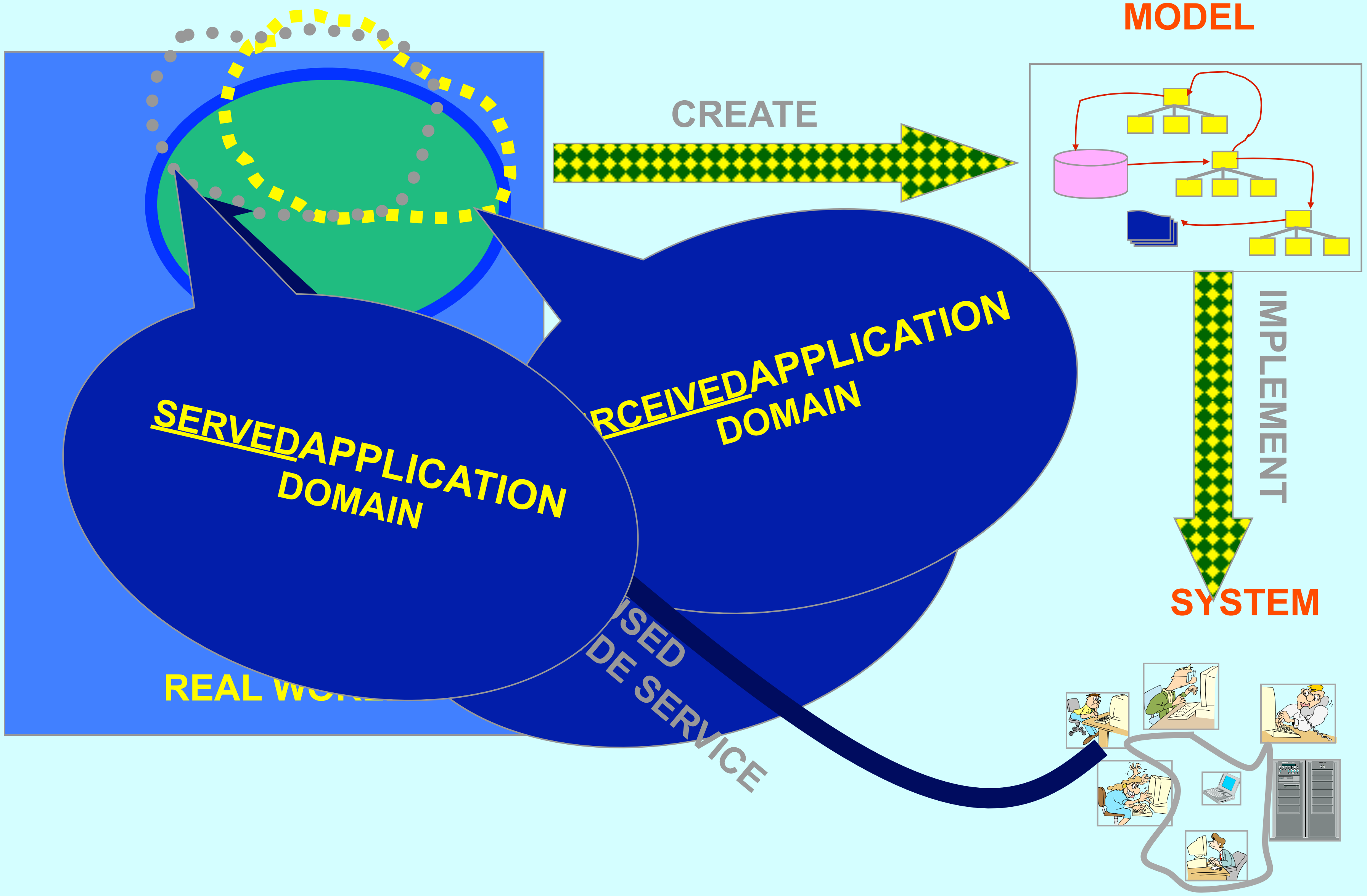
Software evolution, assumptions, and software uncertainty



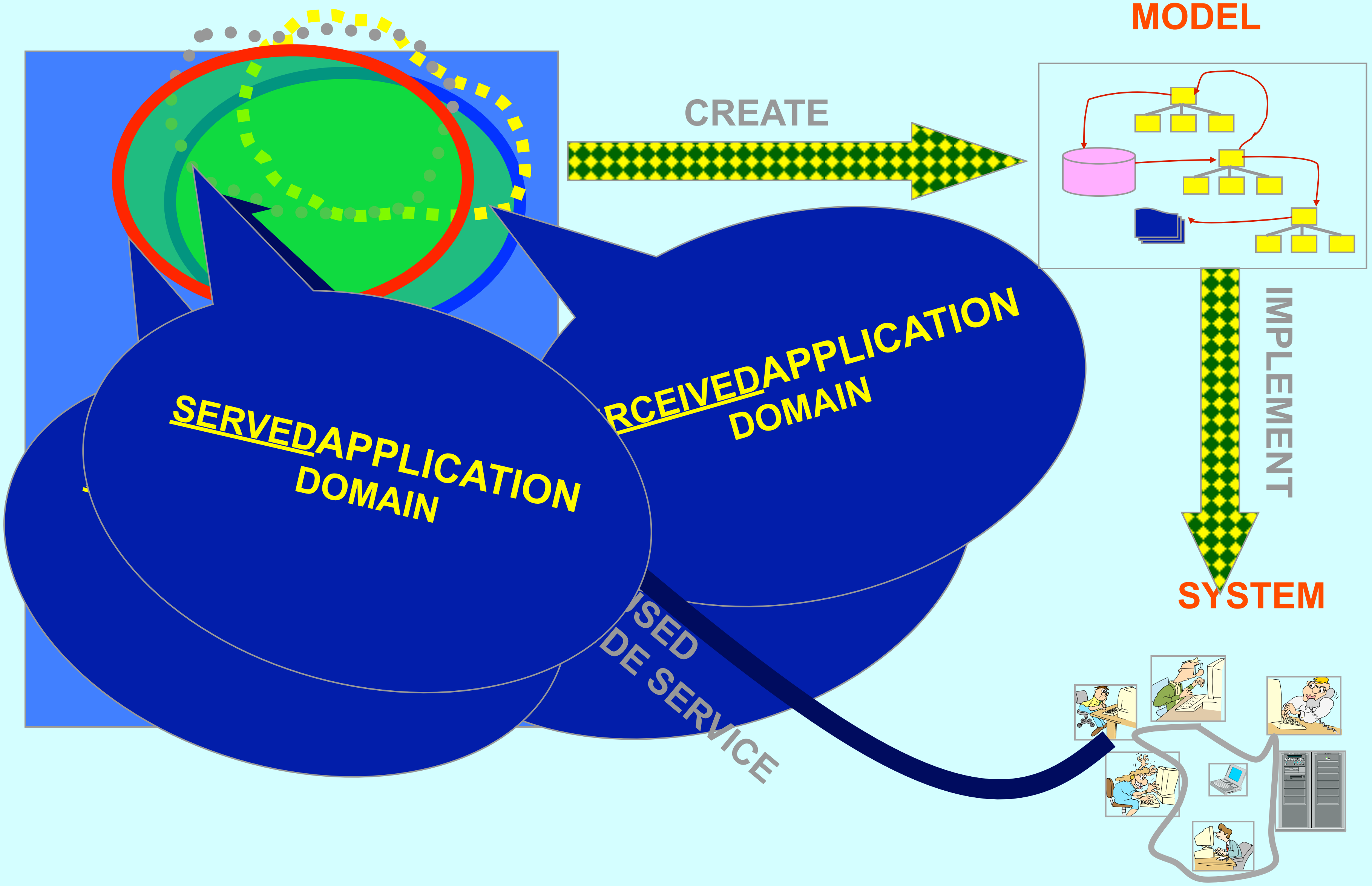
Software evolution, assumptions, and software uncertainty



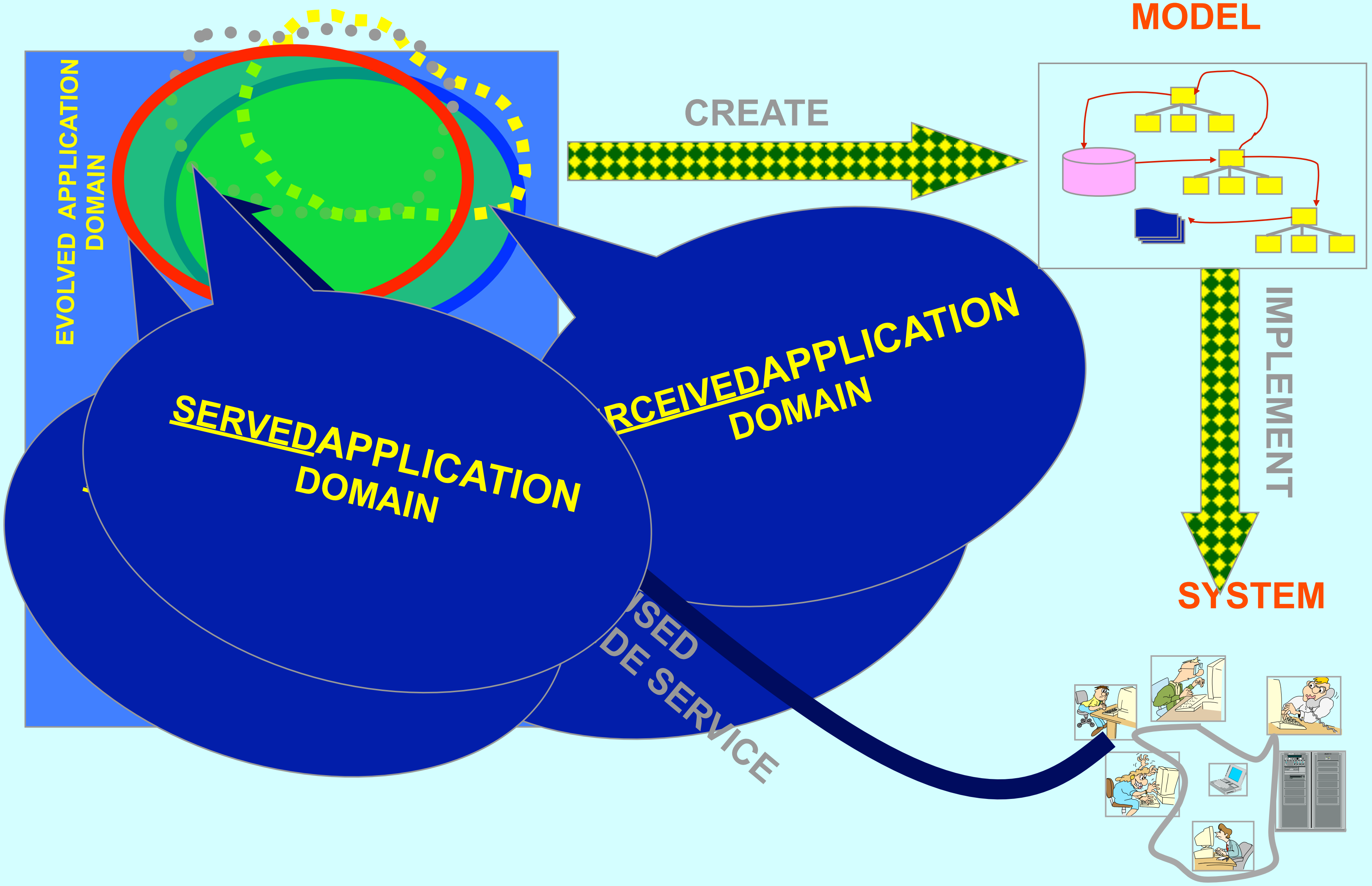
Software evolution, assumptions, and software uncertainty



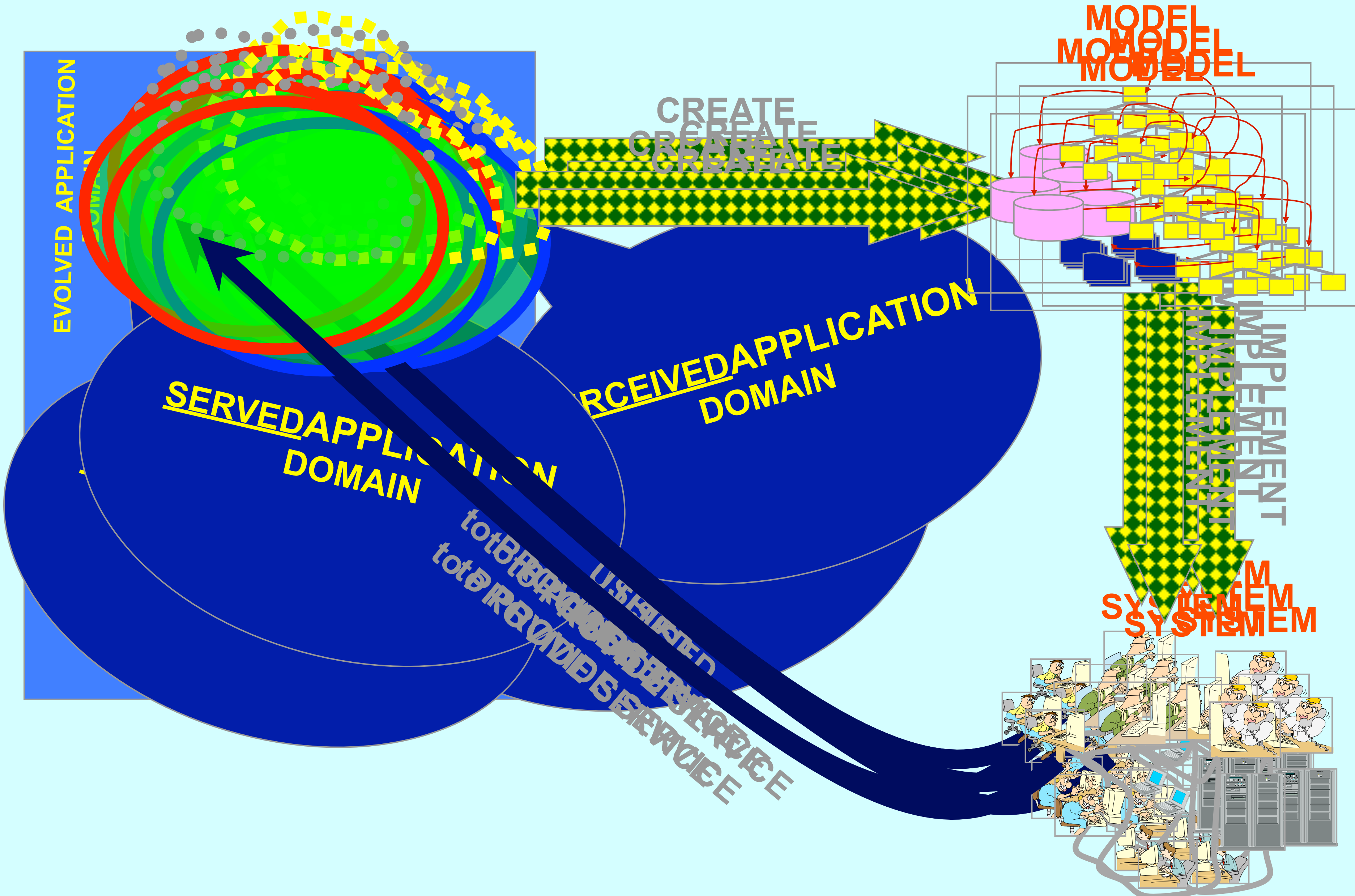
Software evolution, assumptions, and software uncertainty



Software evolution, assumptions, and software uncertainty



Software evolution, assumptions, and software uncertainty


EVOLVED APPLICATION DOMAIN
SERVED APPLICATION DOMAIN
PERCEIVED APPLICATION DOMAIN
CREATE
MODEL
IMPLEMENT
SYSTEM
USE
to PROVIDE SERVICE

Process and Model evolution

- Processes are often the cause of many delays; with their products displaying defects and deficiencies.
- Processes thus need to be improved:
  - Process change: CMMI, SPICE, etc. methods; or
  - A process model can first be changed *in vitro* prior to changing the process itself.

[see Ch. 5 in Madhavji, F-Ramil, Perry: Wiley 2006]

Process Modelling vs. Programming

- At 1987 ICSE, Osterweil proposed the idea of *Process programming* – that, like software, processes too can be programmed.
- Lehman's response was to *promote process models* as a way of understanding and guiding processes and coping with uncertainty.
- Their respective positions seemed to polarise the research community.

Rules and Tools

- Lehman and Ramil take each of the eight Laws and make practical recommendations, e.g.:
- 8th Law – Feedback system
  - Determine the organisational structures and domains within which the technical software development process operates, including information flow, work flow and management control, both forward and feedback, and monitor changes.
  - There are tons of recommendations in all.

[see Ch. 27 in Madhavji, F-Ramil, Perry: Wiley 2006]

Theory of Software Evolution

- Based on the formulation of the Laws, various observations and conceptual frameworks, Lehman and Ramil were on a quest for a theory of software evolution.
- They had elaborated the various objectives and activities that would lead them to this goal.
- They had not yet formulated the theory.
- Once established, a theory would serve several purposes, such as: providing well-founded and plausible interpretations of empirical evidence derived from observed phenomena; and provide a growing base and framework to guide further empirical work.

[see Ch. 16 in Madhavji, F-Ramil, Perry: Wiley 2006]

Communication with Dewayne Perry – 18 August, 2011

Manny's pre-FEAST work had a lot of influence on mine. It is particularly manifested in my 1993 keynote talk at ICSM - dimensions of software evolution.

[see Ch. 2 in Madhavji, F-Ramil, Perry: Wiley 2006]

- Manny and I had been a participants in the software process workshops (ISPW).
- At ICSE as well - where obviously Manny and I were on the same side in the Osterweil/Lehman discussion on process programming vs. process modeling.
- This paved the way for the work later, on FEAST. Our mutual interest obviously was in software evolution.

Communication with Wlad Turski – 18 August, 2011

We liked talking to each other which was quite amazing: an orthodox Jew of German childhood and an atheist Pole, a liberal descendant of a small-nobility family with roots in 14th century.

- In the late 70s or early 80s, we were returning from an IFIP WG 2.3 (Prog. Methodology) meeting... By chance we chose the same train, sat in the same compartment.
- On scientific side,... Manny looked for phenomenological regularities, I was interested in its calculational (formal) properties.
- Success: measurable phenomenology of software process .... Challenge: no measurable feed-back controls in software process have been discovered.

Communication with Juan Ramil – 18-25 August, 2011

- When I joined Manny in Oct 1996 he had already stated and published his 8 laws.
- The statement of the laws were [then] refined to align them to the metric data that the FEAST projects analysed.
- The most notorious refinements are in law 3, where instead of talking of "normal distributions", the refined law talks of "feedback regulation", and law 4 which recognised the existence of 'phases' in the evolution of the system.

Despite his deep thoughts and scientific aspirations, Manny enjoyed very much interacting with young researchers and students: he has himself a fresh mind and a young heart.

Communication with Juan Ramil – 18-25 August, 2011

- VERY difficult to get reliable data to test the laws. The way software systems are constructed and evolved today do not seem to help in studying their evolution. [NHM: agile processes?]
- One very interesting aspect of Manny's work is his approach to data analysis and interpretation. Empirical software engineers should be educated to follow Manny's approach. In particular, Manny avoided complicated statistics and instead promoted thinking about the data in a direct and intuitive way. This could (should?) be further developed...

Communication with Vic Basili – 16 August, 2011

Manny was truly a pioneer and a major contributor to software engineering research.

Communication with Brian Rendall – 16 August, 2011

Manny was above all a gentleman, kindly and thoughtful - and someone who thought long and hard (and successfully) about worthwhile issues in computing science.

- I met him before I left the UK in 1964 to join IBM Research where I worked on Project Y. He joined later, and after a year I was transferred to the IBM West Coast.
- After a year, I returned to IBM Research, and joined Project IMP (Manny's very innovative multiprocessor system design research project), which he was by then leading there.
- I worked on dynamic storage allocation, with Carl Kuehner and Les Belady (both were members of Project IMP).

Communication with Les Belady– 15 August, 2011

They became close friends for life ....

- 1970: Manny's project on parallel processing is terminated. Manny is asked to look into quality and cost problems in the development of IBM's OS/360 . Manny gets project data and starts examining.
- Manny approaches Les to join him in the study of the software process. They realise that this would be novel.
- The two-man project was called "Meta Dynamics of Software under dev. and continuous change."
- 1972: Presentation under new name -- Growth Dynamics.

Communication with Les Belady– 15 August, 2011

- 1972: Manny resigns from IBM to join Imperial College; works remotely with Les.
- 1974 – Les takes a sabbatical to Imperial College; several staff work with Les and Manny.
  - One staff recommends the name for the study: **Program Evolution**.
- By 1980, Les stopped contributing to evolution studies (for other initiatives).
- Manny and Les complemented each other very well. Brainstorming together Manny was pushing hard for creating a new branch of science ..... while Les was the engineer with design background looking for differences/similarities between HW and SW and the interplay between process and structure.

Process Award Committee

- With Vic Basili, Watts Humphrey, Barry Boehm, and Bill Riddle, Lehman served on the first Software Process Award Achievement Committee for several years.

"I and was always amazed by his insights and articulation of those insights."

-- Vic Basili, 16 August, 2011

Personal Memories -1

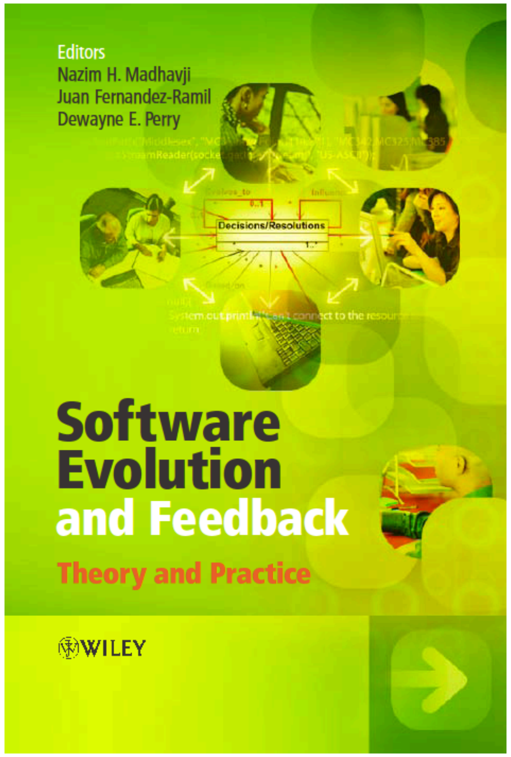
- 1985: Nazim meets Manny at ICSE at Imperial.
- Met Manny at ISPWs (process workshops) and ICSMs.
  - Got on like “House on Fire”.
- 1993: Manny attends “Process Evolution” workshop (Co-Chaired by Nazim and Bill Riddle) in Quebec.
- Mid-90s: Manny visits Nazim at McGill (Montreal).
- 2000: Nazim attends FEAST-4 workshop at Imperial.

Personal Memories - 2

- Manny invites Nazim to Edit the FEAST-4 workshop papers.
- Nazim (naively) convinces Manny to instead do a new, edited book project, a sequel to the seminal 1985 Lehman-Belady book “Program Evolution”
- Manny agrees. Perry and Ramil are invited to join the editorial team.
- Mountain to climb, with 27 chapters.
- 2000: Nazim moves to NZ.
- 2002 Manny moves to Middlesex University.

Personal Memories - 3

- 2003: Manny and Ramil visit Nazim (who had moved to) Ontario.
- Book published in 2006, Wiley.



Final thought

- Manny’s work was different from most researchers’. He didn’t do research in language notations and mechanisms, design methods, program structure, tool development, testing and analysis of systems, single release concern, and the like. He marched at the beat of his own drum.
- He was always concerned about the “big picture”: the laws, how systems change over time, feedback, uncertainty, assumptions, theory, etc.
- He was thus a “software evolution philosopher” with an acute sense for empiricism.
- For those of us researching in these areas, there are numerous inroads Manny has left behind for us to explore for “Manny” years to come.

Bio - 1

- Lehman studied mathematics at Imperial College London.
- Was involved in the design of the Imperial College Computing Engine's Digital Computer Arithmetic Unit.
- Worked for Ferranti (1956–1957), the Scientific Department of Israel Ministry of Defence (1957–1964) and IBM (1964–1972).
- 1972 to 2002: was with Imperial College, where he headed Section and later Department.
- 2002: moved to Middlesex University.
- After retiring from Middlesex, he moved to Jerusalem, Israel, where he passed away on December 29, 2010.

Bio - 2

- Manny received numerous awards, including:
  - the IEEE Computer Society 'Harlan D. Mills Award' in 2001 'For pioneering contributions to the empirical study of software processes and program evolution' and
  - the Re-engineering Forum 'Stevens Award' in 2003.

[JSME, April 2011]