Software evolution

background, theory, practice

Alexandre Bergel
abergel@dcc.uchile.cl

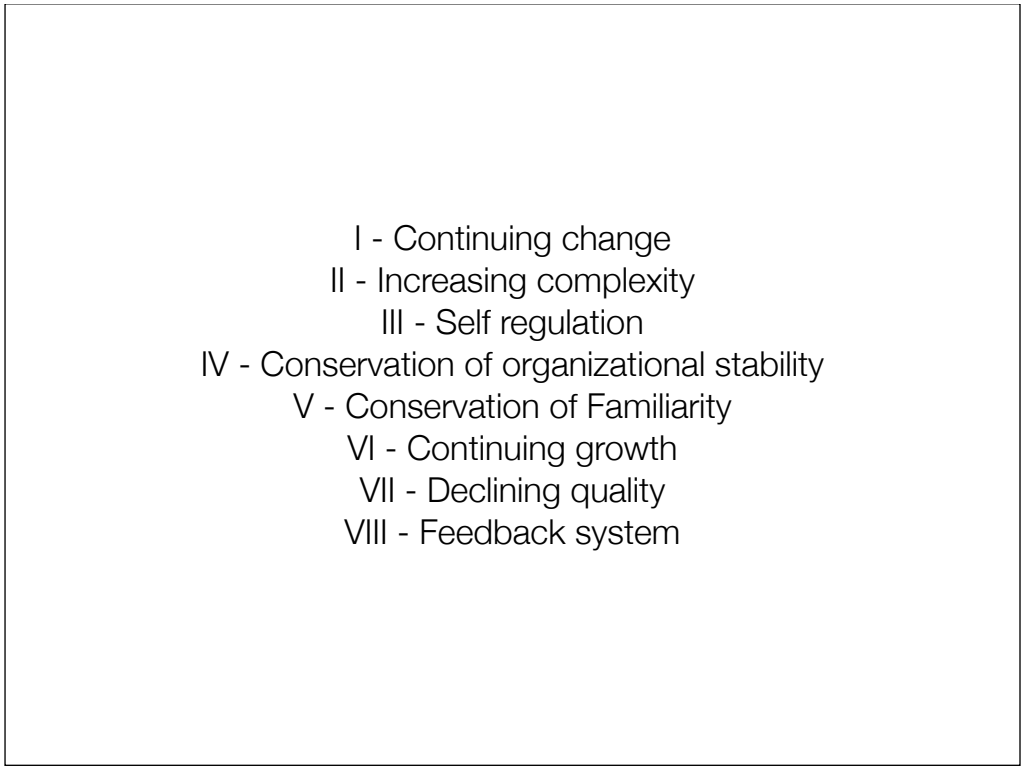
During a 1968 study, Lehman studied the evolution of OS/360, and over a 20 years period he formulated 8 statements knows as *Laws of Software Evolution*

Lehman was an engineer at IBM during 60's, and was deeply involved into the OS/360, which was an operating system. According to Wikipedia: "OS/360, officially known as IBM System/360 Operating System,[1] was a group of batch processing operating systems developed by IBM for their then-new System/360 mainframe computer, announced in 1964. They were among the earliest operating systems to make direct access storage devices a prerequisite for their operation."

Lehman's original work is founded on the analysis of the programming process at IBM.

Meir M Lehman,
id., Programs, Cities, Students, Limits to Growth?, Inaugural Lecture, May 1974. Publ. in Imp. Col of Sc. Tech. Inaug.l Lect. Ser., vol 9, 1970, 1974, pp. 211 - 229. Also in Programming Methodology, (D Gries ed.), Springer, Verlag, 1978, pp. 42 - 62

Meir M Lehman,
Laws of Software Evolution Revisited, 1997



In “Laws of Software Evolution Revisited” [Lehman97], an historical background is provided.
"The first three of a total of now eight laws of software evolution were formulated in the mid seventies [Leh74] arising from analysis of data first acquired during a study of the IBM programming process [Leh69]. These three were discussed in somewhat greater detail in 1978 [Leh78]. Two further laws were introduced in a 1980 paper [Leh80a] with the sisth introduced in a footnote [Leh91]. [...] All relate specifically to E-type systems [Leh80b] that is, broadly speaking, to software systems that solve a problem or implement a computer application in the real world."

I - Continuing change

"An E-type program that is used must be continually adapted else it becomes progressively less satisfactory."

Software programs behave in an organic way

The first Lehman's law suggests that the growth of an E-type software system share similarities with an organism. "Growth results from feedback pressures caused by mismatch between the software and its operational domain.

This need for continuing adaptation and evolution is intrinsic to software applications and systems. It is, in part, due to the fact that development, installation and operation of the software changes the application and its operational domain so creating mismatch between the two.

Evolution is achieved in a feedback driven and controlled maintenance process. If the consequent pressure for evolution to adapt to the new situation is resisted the degree of satisfaction provided by the system in execution declines with time."

II - Increasing complexity

"As a program is evolved its complexity increases unless work is done to maintain or reduce it"

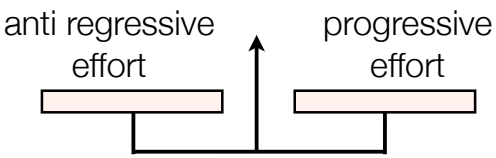
"This law may be an analogue of the second law of thermodynamics or an instance of it"

"It results from the imposition of change upon change upon change as the system is adapted to a changing operational environment. As the need for adaptation arises (Law I and VII) and changes are successively implemented, interactions and dependencies between the software component increase in an unstructured pattern and lead to an increase in system entropy. "

One definition of "Entropy": lack of order or predictability; gradual decline into disorder
This definition is important in Software Engineering, we will come back to it when we will talk about Software Maintenance.

"If the growth in complexity is not constrained, the progressive effort needed to maintain the system satisfactory becomes increasingly difficult. "

II - Increasing complexity



=> determined by feedback

In 1974, Lehman [Leh74] showed that “if anti regressive effort is invested to combat the growth in complexity, less effort is available for system growth. Given that resources are always limited the rate of system growth declines as the system ages whichever strategy is followed. In practice the balance between progressive and anti-regressive activity is determined by feedback.”

III - Self regulation

"The program evolution process is self regulated with close to normal distribution of measures of product and process attributes"

Positive and negative feedback regulate the evolution process

IV - Conservation of Organisational Stability (invariant work rate)

"The average effective global activity rate on an evoling system is invariant over the product life time"

"the level of activity is not decided exclusively by management but by a host of feedback inputs and controls"

V - Conservation of Familiarity

"During the active life of an evolving program, the content of successive releases is statically invariant"

> "The more changes and additions are associated with a particular release, the most difficult it is to for all concerned to be aware, to understand and to appreciate what is required of them"

VI - Continuing growth

"Functional content of a program must be continually increased to maintain user satisfaction over its lifetime"

"The E-type system inevitably grows with time driven by feedback from user and marketeers"

Different but not unrelated to the first law

VII - Declining quality

"E-type programs will be perceived as of declining quality unless rigorously maintained and adapted to a changing operational environment"

> "A system that has performed satisfactorily for some period of time may suddenly exhibit unexpected, previously unobserved, behaviour."

> "... uncertainty increases with time unless successful attemps to detect and rectify the embodiment are taken as part of the maintenance activity."

VIII - Feedback system

"E-type programming processes constitute multi-loop, multi-level feedback systems and must be treated as such to be successfully modified or improved"

Popularity of Scrum and Extreme Programming illustrate this law