XION IT SYSTEMS
AKTIENGESELLSCHAFT
Dresdnerstaße 81-85/8.Stock
A-1200 Wien
Tel: 0664-8242-600
E-mail: office@xion.at
Web: xion.at
Festnetz: +43/1/333 91 99-0
Fax: +43/1/333 91 99-199

x i o n . it systems ag

Software Wartung und Evolution

Dipl.-Ing. Dr. techn. Johannes Weidl-Rektenwald
Xion IT Systems AG

XION IT SYSTEMS
AKTIENGESELLSCHAFT
Dresdnerstaße 81-85/8.Stock
A-1200 Wien
Tel: 0664-8242-600
E-mail: office@xion.at
Web: xion.at
Festnetz: +43/1/333 91 99-0
Fax: +43/1/333 91 99-199

x i o n . it systems ag

Chapter 5

Chapter 5

- Inhalte
  - Software Evolution
    - Definition
    - Types of Programs
    - Laws of Software Evolution
    - Change Patterns and Evolutionary Narratives

Fallbeispiel: „Adaption auf der falschen Abstraktionsstufe“

- Gegeben: Datenbankmodell
  - Datenbankfeld: „SA_EINZEL“
    - Vermutete Semantik
      - SA ... Sportart
      - EINZEL … Einzelsportart
  - Typ: char(1), keine Constraints
  - Wertbelegung in der Datenbank
    - „J“ ... Folgerung: Ja, Sportart ist eine Einzelsportart
    - „N“ ... Folgerung: Nein, Sportart ist keine Einzelsportart
    - **„T“ ... Überraschung!: Sportart ist eine „Tennissportart“** ☹

Software Evolution

Evolution: General Definition 1/2

- Evolution is the process of *progressive change* over time in characteristics, attributes, properties of some material or abstract, natural or artificial, entity or system or of a sequence of these
  - Changes are progressive when they result in a definable trend of, for example, increasing value, growing precision or better fit to a changing domain or context

© J. Weidl-Rektenwald 02-08

Evolution: General Definition 2/2

- Entities include objects or collections of objects (e.g. population) such as natural species, societies, cities, artefacts, concepts, theories, ideas or systems of these
- Change process will, in general, be continual with relatively slow rate of change, or discrete with individual incremental changes, small relative to entity as a whole

- Source: Lehman and Ramil 2001

Software Evolution: Definition 1/2

- Keine genormte Definition
- Nach Lehman/Ramil
  - Software Evolution is the process of continual fixing, adaption, enhancement to maintain stakeholder satisfaction
    - In response to changes in domains, needs, expectations
- Nach Bennet/Rajlich
  - Maintenance means general post-delivery activities
  - Evolution refers to a particular phase in the staged model where substantial changes are made to the software

Software Evolution: Definition 2/2

- Nach Godfrey
  - Evolution is what happens while you are busy making other plans
    - Maintenance is the *planned* set of tasks to effect changes
    - Evolution is what actually happens to software

Types of Programs

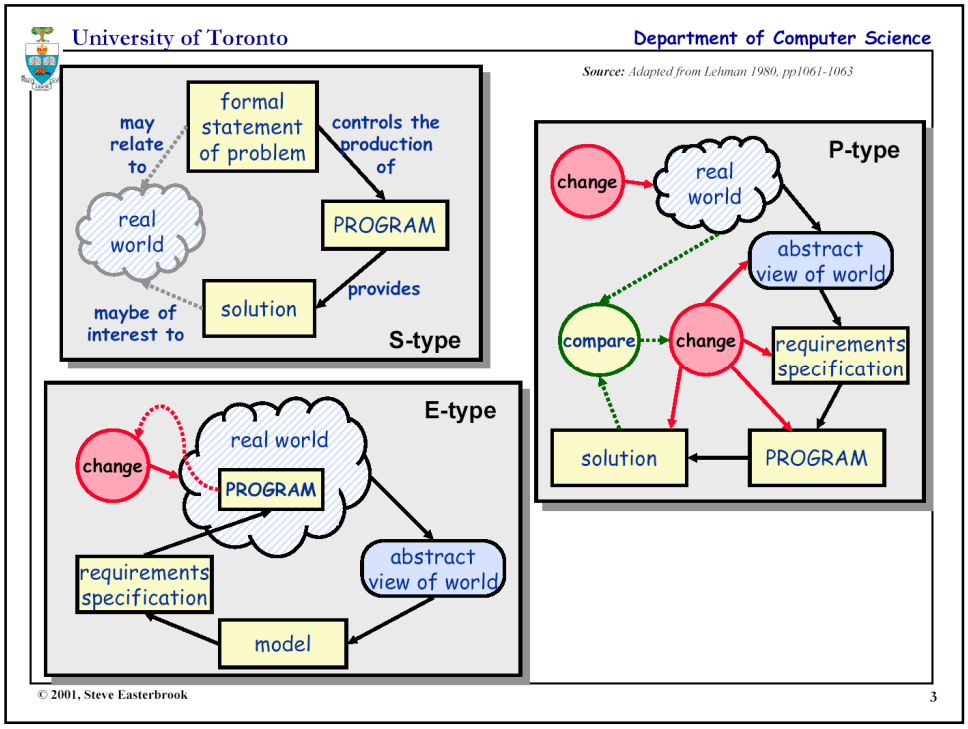
- *Nach Lehman, Belady 1980, pp. 1060-1076*
- S-type Programs („Specifiable“)
  - Problem can be stated formally and completely
  - Acceptance: Is the program correct according to its specification?
  - This software does not evolve
    - A change to the specification defines a new problem, hence a new program

Types of Programs

- P-type Programs („Problem-solving“)
  - Imprecise statement of a real-world problem
  - Acceptance: Is the program an acceptable solution to the problem?
  - This software is likely to evolve continuously
    - Because solution is never perfect, and can be improved
    - Because the real-world changes and hence the problem changes

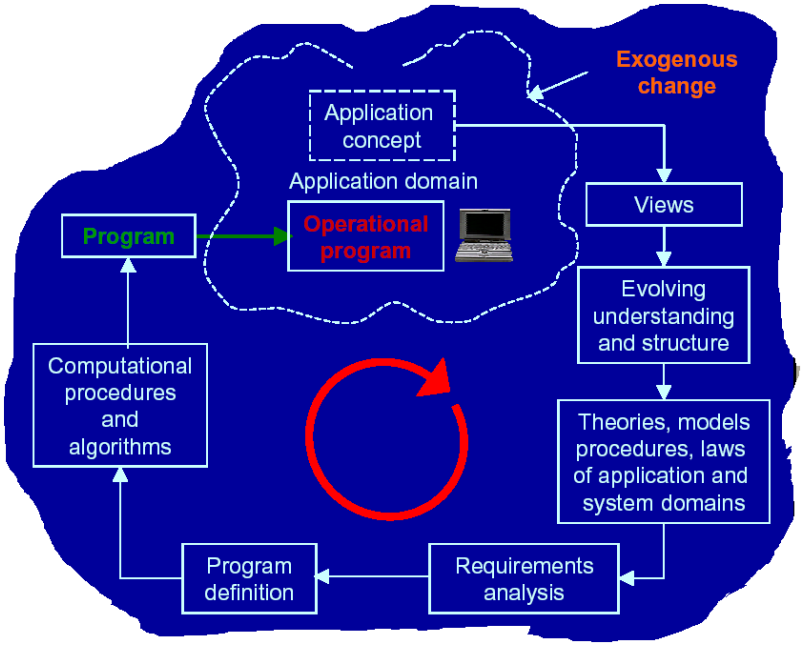
Types of Programs

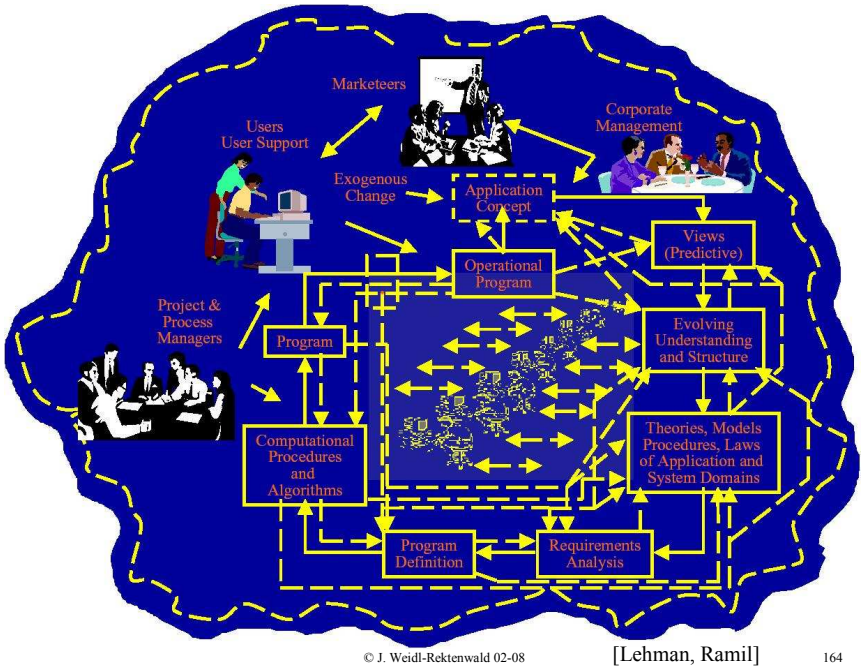
- E-type Programs („Embedded“)
  - A system that becomes part of the world it models
  - Acceptance: Depends entirely on opinion and judgement; criterion is the satisfaction of stakeholder needs
  - This software is *inherently* evolutionary
    - Changes in the software and the world affect each other



Software Systeme als komplexe Feedback Systeme

- Der Entwicklungs- und Evolutionsprozess eines Software Systems wird von Lehman als
  - Multi-level
  - Multi-loop
  - Multi-agent
  - Feedback System bezeichnet.
- Feedback technisch: Die Rückführung eines (transformierten) Teils des Ausgangssignals als Eingangssignal in ein System
  - („Feedback: The return of a portion of the **output**, or processed portion of the output, of a (usually active) device to the **input**“)


Exogenous change
Application concept
Application domain
Program
Operational program
Views
Evolving understanding and structure
Computational procedures and algorithms
Theories, models procedures, laws of application and system domains
Program definition
Requirements analysis

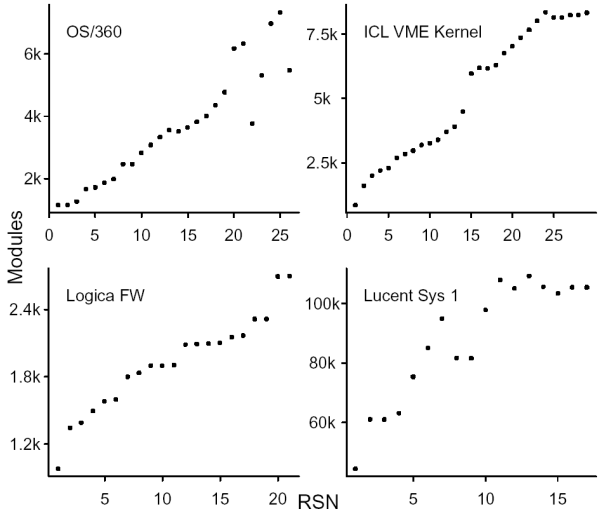

Marketeers
Users
User Support
Corporate Management
Exogenous Change
Application Concept
Views (Predictive)
Operational Program
Project & Process Managers
Program
Evolving Understanding and Structure
Computational Procedures and Algorithms
Theories, Models Procedures, Laws of Application and System Domains
Program Definition
Requirements Analysis

Laws of Software Evolution

- In den späten sechziger Jahren untersuchen *Lehman und Belady* die *Release History Daten* von IBM OS/360 mittels bestimmter Metriken und stoßen auf Eigenschaften im Evolutionsprozess, die bei anderen Systemen in späteren Untersuchungen ebenfalls nachvollzogen werden können
- Diese Eigenschaften scheinen Gesetzmäßigkeiten zu folgen und wurden als „Laws of Software Evolution“ postuliert
- Die „Laws of Software Evolution“ ergeben sich aus der Beobachtung von *E-type programs*

© J. Weidl-Rektenwald 02-08

Laws of Software Evolution



Laws of Software Evolution

- Warum „Gesetze“?
  - Die entdeckten Phänomene der Evolution werden als Gesetze bezeichnet, da sie technologie- und prozessunabhängige Mechanismen bezeichnen

Laws of Software Evolution

- *Nach Lehman, Belady 1980, pp. 1061-1063 und spätere Publikationen*
- 1) Law of continuing change
  - “A system that reflects some external reality undergoes continuing change or becomes progressively less useful
    - The change process continues until it becomes more economical to replace it by a new or restructured system.”
- 2) Law of increasing entropy (or: complexity)
  - “The entropy of a system increases with time unless specific work is executed to maintain or reduce it.”

Laws of Software Evolution

- 3) Fundamental law of software evolution
  - Software evolution is self-regulating with statistically determinable trends and invariants
- 4) Conservation of organisational stability (invariant work rate)
  - During the active live of a software system the *average effective global activity rate* is roughly constant

Laws of Software Evolution

- 5) Conservation of familiarity
  - In general, the average incremental growth rate (growth rate trend) tends to decline
    - As an E-type system evolves all associated with it, developers, sales personnel, users, for example, must maintain mastery of its content and behaviour to achieve satisfactory evolution. Excessive growth diminishes that mastery.
- 6) Continuing growth
  - The functional content of E-type systems must be continually increased to maintain user satisfaction

Laws of Software Evolution

- 7) Declining quality
  - The quality of E-type systems will appear to be declining unless they are rigorously maintained and adapted to operational environment changes
- 8) Feedback System
  - E-type evolution processes constitute multi-level, multi-loop, multi-agent feedback systems and must be treated as such to achieve significant improvement over any reasonable base

Lehmans Approach: Formal

- Lehman describes software evolution on a formal level
  - Based on observations
    - Empirical generalisations are made
    - They provide basis for axioms in a formal theory
  - Possible inferences are proposed
    - Derived from the formal models
    - Basis for potential theorems in formal theory
  - Try to fully prove theorems

Formal Models of Software Evolution: Growth

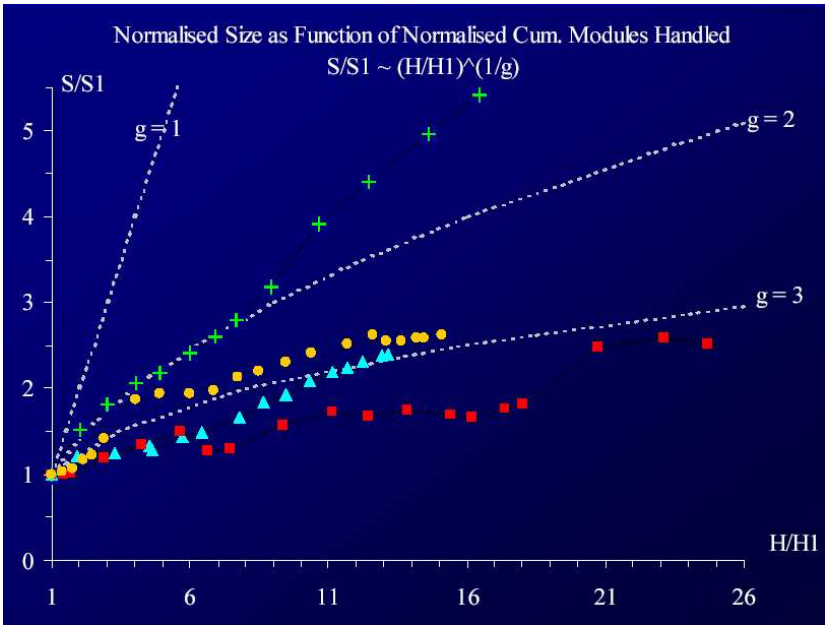
- Inverse Square Model *[Turski 1996]*
  - $\underline{S_1} = S_1$
  - $\underline{S_i} = \underline{S_{i-1}} + e / (\underline{S_{i-1}})^2$
    - *S ... Size (often number of modules)*
    - *i ... Release sequence number (1..n, n = max release nr.)*
    - *e ... Model parameter*
    - *S_i and $\underline{S_i}$ stand for actual and predicted size at release i*
- *Other model: Normalised size as a function of the normalised work rate [Lehman 2001]*
  - $\underline{S_i} / S_1 = (H_i / H_1)^{1/g'}$ *for* $i >= 1$
    - *H ... Work rate as indirect effort indicator (e.g. elements handled)*
    - *g´ ... Model parameter*

© J. Weidl-Rektenwald 02-08

Formal Models: An Example

- Next slide shows the *normalised size as a function of the normalised work rate*
  - size measured in number of modules
  - work rate measured in modules handled
  - For four industrially evolved systems
  - Three different organisations
  - Three different applications domains
- Data taken from release data history

© J. Weidl-Rektenwald 02-08



Formal Models: Use?

- Formal Models provide means for
  - Evolution planning
    - Simulation, visualisation, release planning
  - Process Management and Control
  - Long term prognosis
  - Overall process improvement
  - Tools

© J. Weidl-Rektenwald 02-08

Research Areas in Software Evolution

- The driving force guiding the work will be the search for formally supported techniques:
  - logic-based declarative description and reasoning techniques
  - formal models for software evolution based on rewriting systems
  - software metrics
  - visualisation techniques
  - generation of design documents and source code
  - extraction of design and analysis documentation
  - migration to component-based and web-based systems
  - the use of metamodels as a general integration technique

© J. Weidl-Rektenwald 02-08

Change Patterns and Evolutionary Narratives

Change patterns and evolutionary narratives

- Cathedral style [Raymond]
  - careful control and management
  - debugging done before committing code
  - evolution is slow, planned, rarely undone
- Bazaar style
  - lots of low-level changes, frequent fixes
  - lots of "building around" rather than wholesale changing, occasional redesigns
  - creeping feature-itis, "complete" dependency graph

Change patterns and evolutionary narratives

- Band-aid evolution (just add a layer)
  - quick & dirty way to add new functionality, esp. if system is not well understood
    e.g. Y2K fixing, adding portability, new features
- "Vestigial features"
  - design artifact persists after rationale dies
    e.g. whale fin bone structure resembles hand

Change patterns and evolutionary narratives

- "Adaptive radiation" [Lehman]
  - when conditions permit, encourage wild variation for a while
  - later, evaluate and let "best" ideas live on.
    e.g. Linux kernel evolution
- "Convergent evolution"
  - compare similar systems to reference architecture (or to each other)
    e.g. everyone grows an XML generator in response to market pressure

© J. Weidl-Rektenwald 02-08
[Godfrey 2001]

Change patterns and evolutionary narratives

- Radical redesigns (localized and global)
  - aka "refactoring"
  - little new functionality added, but structure changes significantly, legacy cruft dissipates
  - likely "goodness" (design metrics) improves
- Migration patterns
  - look out for known translation idioms, especially if migration is not one big bang
    e.g. procedural-to-OO idioms

© J. Weidl-Rektenwald 02-08
[Godfrey 2001]

Change patterns and evolutionary narratives

- Reuse patterns
  - components are (re)used in different systems
    e.g. build COTS interface, throw out homebrew DB

[Godfrey 2001]

Ausgewählte Kapitel der Software Wartung

- Program Comprehension
  - Span of Understanding
- Maintainability (Wartbarkeit)
- Design for Change
  - Design Metriken
    - Coupling und Cohesion

Span of Understanding

- Span of Understanding
  - Nennt man die Zeitspanne, die der Programmierer zum Verstehen eines definierten Programmstückes benötigt
- Ziel der Forschung: Wie kann man den *Span of Understanding* verkürzen bzw. minimieren

Definition: Maintainability

- Maintainability is the ease of maintenance
- Can be decomposed as
  - Repairability
    - Ability to correct defects in reasonable time
  - Evolvability
    - Ability to adapt software to environment changes and to improve it in reasonable time

Design Metrik: Cohesion

- Beschreibt den Grad der logischen Abhängigkeiten innerhalb eines Software Moduls
- Je größer die Cohesion desto besser das Software Design
- Hinter einem Interface sollte die Cohesion maximal sein
- Große Cohesion erlaubt die einfache Wiederverwendung von Software Modulen

Design Metrik: Coupling

- Coupling beschreibt den Grad der logischen Abhängigkeiten zwischen verschiedenen Software Modulen
- Je größer das Coupling desto schlechter das Software Design
- Das Coupling über Interfaces hinweg sollte minimal sein
- Großes Coupling verhindert die einfache Wiederverwendung von Softwaremodulen

Computer science...

- Computer Science is the discipline that believes all problems can be solved by adding one more layer of indirection.
  - *Dennis DeBruler*

Teufelsquadrat