Software Evolution
Basic Concepts

Roberto A. Bittencourt
Based on Tripathy and Naik's slides and also on Rajlich's slides

Basic Concepts

- Software evolution and maintenance
- Software change classification
- Software change processes
- Staged software evolution model
- Legacy system
- Reengineering

Software Evolution and Maintenance (generally are used interchangeably)

- Lowell Jay Arthur distinguishes the two terms as follows:
  - "Software **evolution** means a continuous change from lesser, simpler, or worse state to a higher or better state."
  - "Software **maintenance** means to preserve from failure or decline."
- Keith H. Bennett and Lie Xu use the term:
  - "**maintenance** for all post-delivery support and **evolution** to those driven by changes in requirements."

Software Evolution and Maintenance

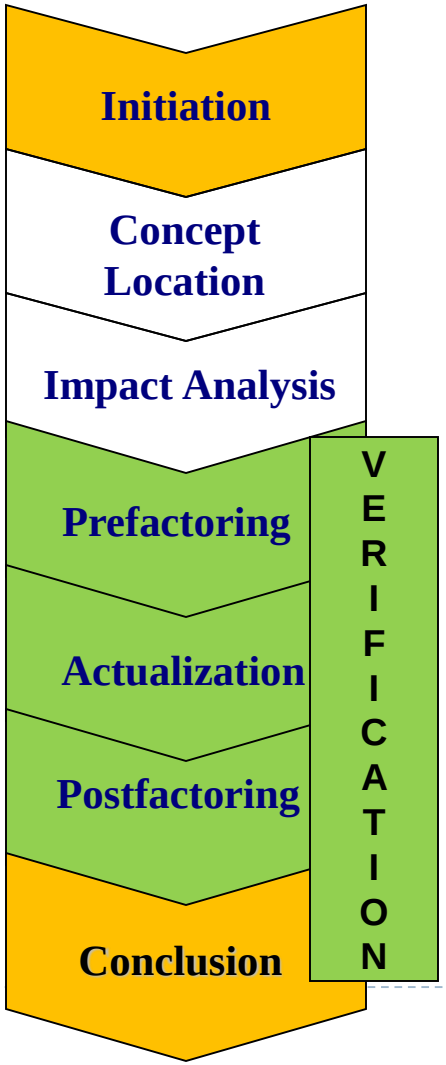
- Software Evolution
  - In 1965, Mark Halpern used the term evolution to define the dynamic growth of software
  - The term evolution in relation to application systems took gradually in the 1970s
  - Lehman and his collaborators from IBM are generally credited with pioneering the research field of software evolution
  - Lehman formulated a set of observations that he called laws of evolution
- Software Maintenance
  - There will always be defects in the delivered software application because software defect removal and quality control are not perfect
  - Software maintenance is needed to repair these defects in the released software
  - E. Burton Swanson defined three types of software maintenance:
    - Corrective, Adaptive & Perfective
    - ISO/IEC 14764 Introduced a fourth category called preventive

Classification of Software Changes

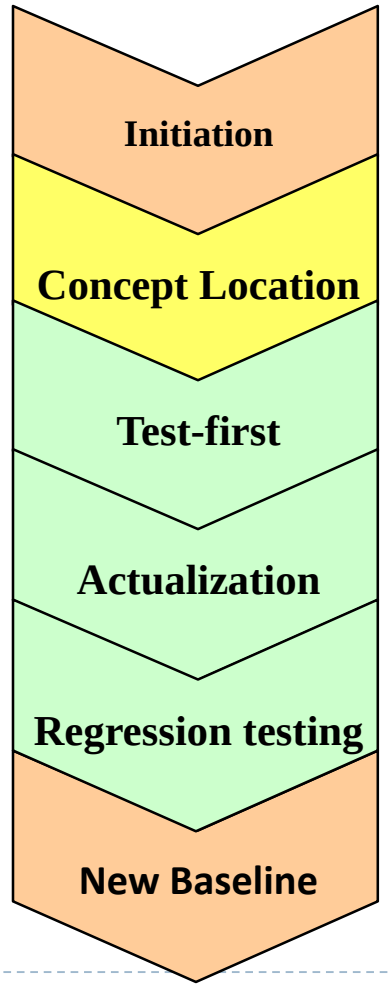
- Evolutionary
  - Include a new feature or improve an existing feature
- Perfective
  - make a variety of improvements, namely, user experience, processing efficiency, and maintainability
- Corrective
  - Fix bugs that cause processing and performance failures
- Adaptive
  - enable the system to adapt to changes in its environment
- Preventive
  - prevent problems by fixing latent bugs that have not yet caused failures to the users
- Refactoring
  - Improve code structure by preserving its behavior

Examples of Software Change Processes

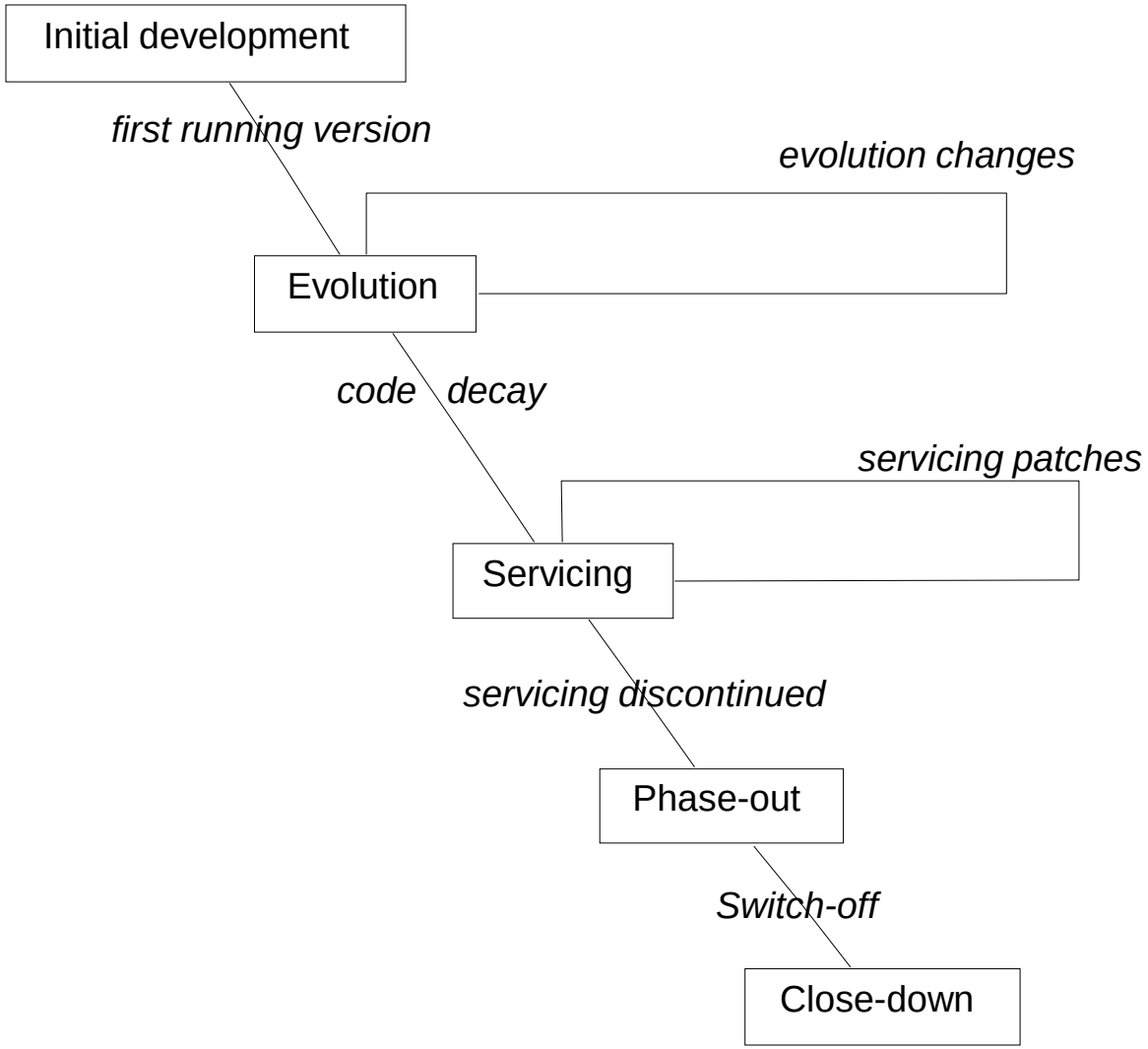
- Rajlich SC Model



- Test-Driven Development



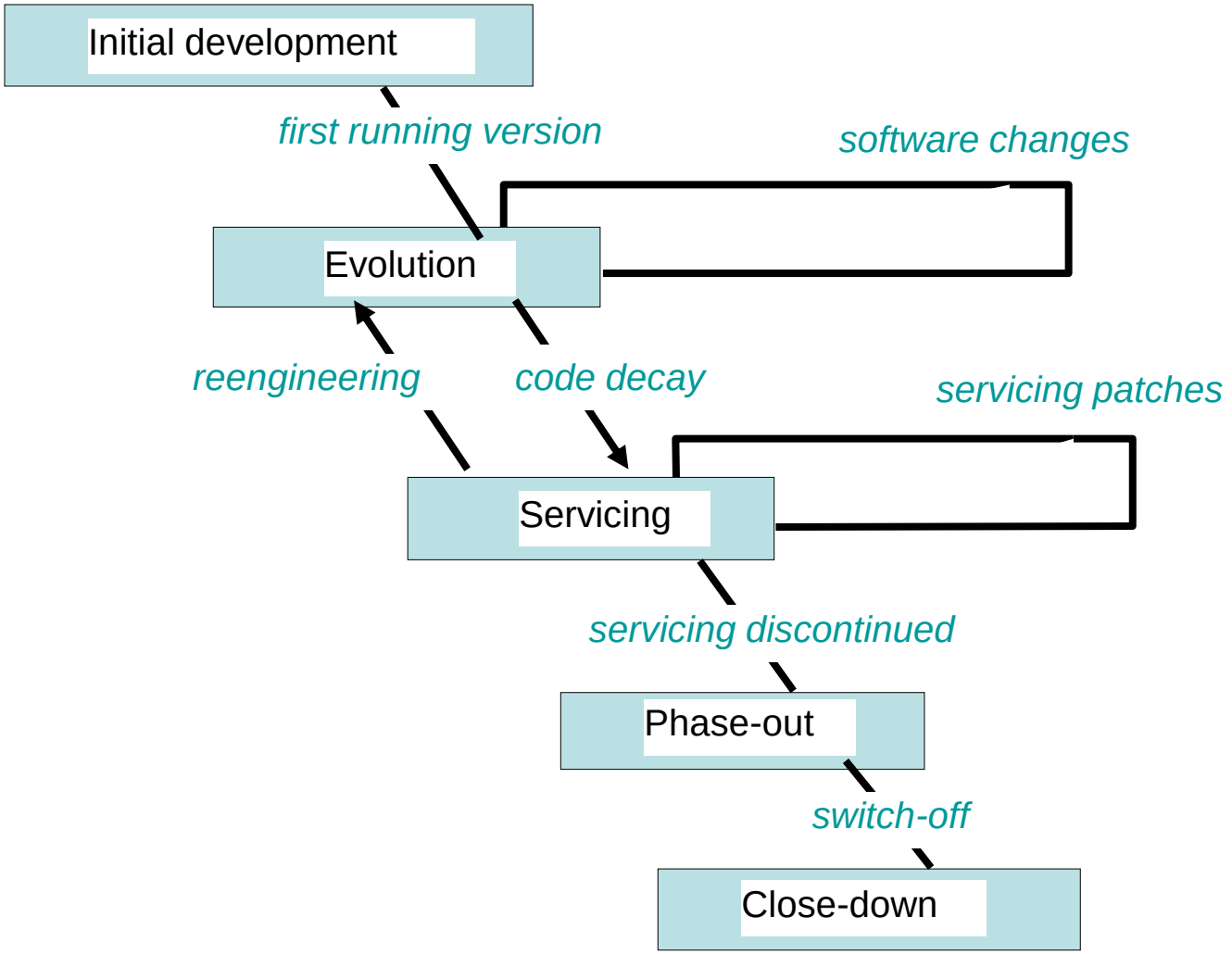
Rajlich & Bennett's Software Evolution Model



Legacy System

- A legacy system is an old system which is valuable for the company which often developed and owns it.
- It is the phase-out stage of Rajlich and Bennett's software evolution model
- Bennett used the following definition:
  - "large software systems that we don't know how to cope with but that are vital to our organization."
- Similarly, Brodie and Stonebraker define a legacy system as:
  - "any information system that significantly resists modification and evolution to meet new and constantly changing business requirements."

Can developers revert code decay? If so, Reengineering is the term



Reengineering

- Reengineering is done to transform an existing "lesser or simpler" system into a new "better" system.
- Chikofsky and Cross II defines reengineering as:
  - "the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form."

Reengineering steps

- Jacobson and Lindstorm defined the following formula:
  - Reengineering = Reverse engineering + Δ + Forward engineering
- **Reverse engineering**: defining a more abstract, and easier to understand, representation of the system
- **Forward Engineering**: the traditional process of moving from a high-level abstraction to the physical implementation of the system
- The second element Δ captures the alteration that is the effective change of the system