Dasar Pengembangan Sistem Informasi

Maintenance and Evolution

Tim Pengampu Mata Kuliah

FILKOM
FAKULTAS ILMU KOMPUTER

Maintenance

Maintenance Definition

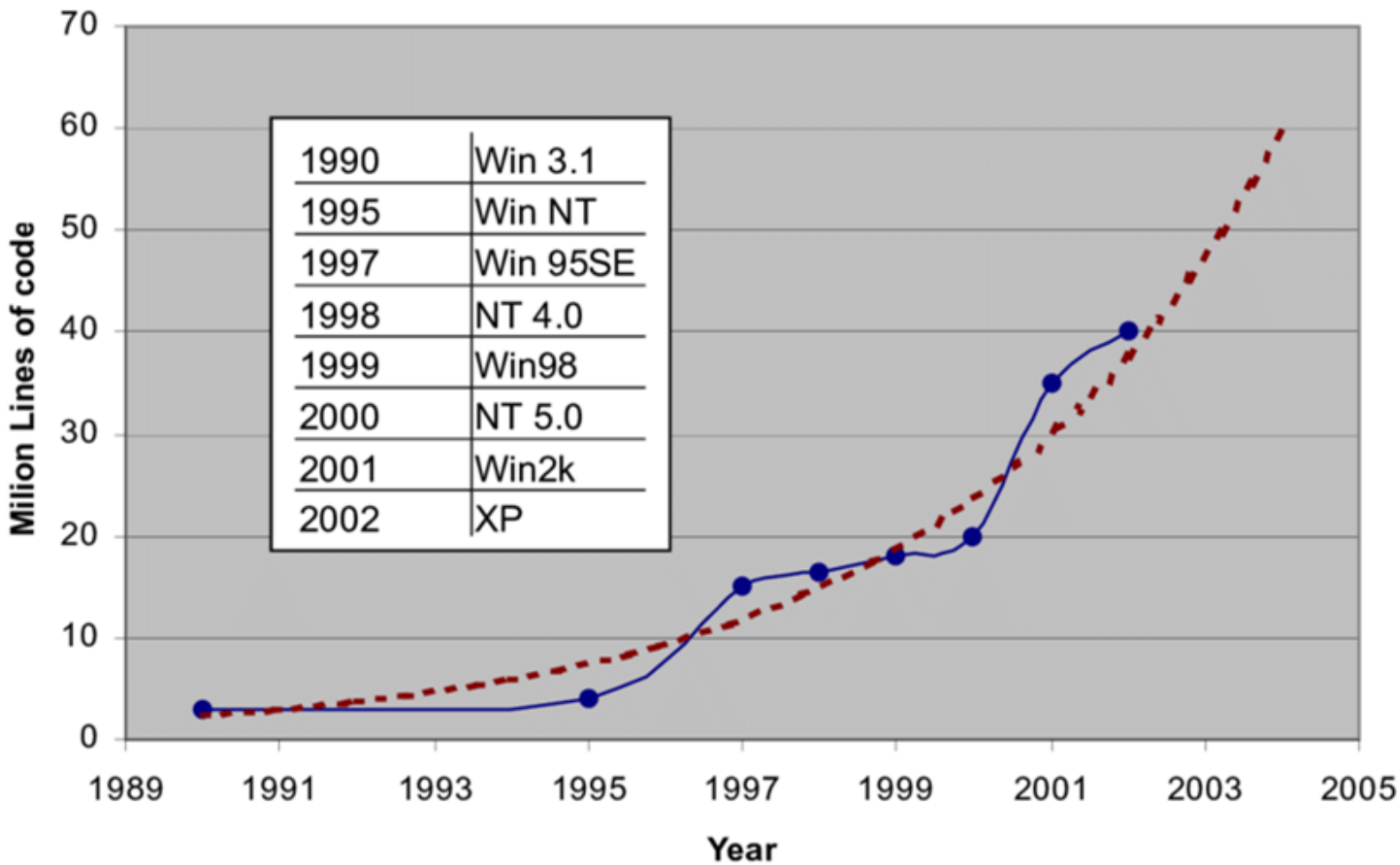
- The **modification** of a software product **after delivery** to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment. (IEEE)
- The **modification** of a software product **after delivery** to correct faults, to improve performance or other attributes. (ISO/IEC)
- The software product undergoes modification to code and associated documentation due to a problem or the need for improvement. The objective is to **modify** the existing software product while preserving its integrity.

Examples

- Y2K
  - many, many systems had to be updated
  - language analyzers (find where changes need to be made)
- Anti-Virus Software
  - don't usually have to update software, but must send virus definition updates
- Web-based applications
  - 73% of total cost of e-commerce used to re-design the website after its first implementation

Examples

- Windows NT, 30 million LOC added within 6 years

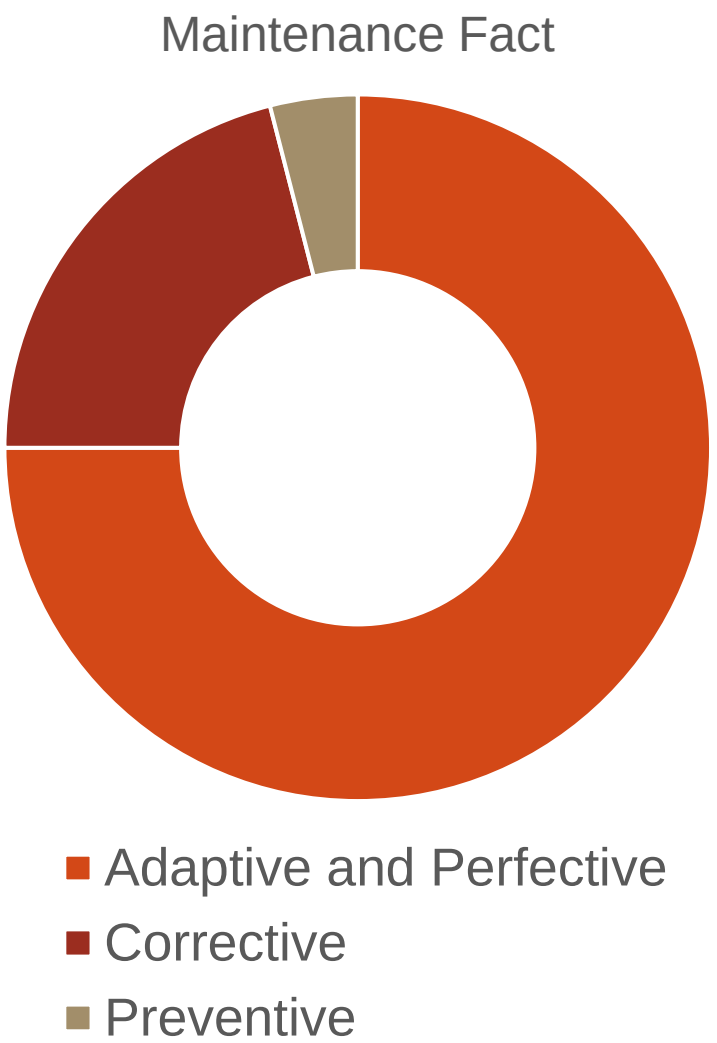


Types

- Adaptive: changes in the software environment (DBMS, OS)
- Perfective: new user requirements
- Corrective: diagnosing and fixing errors
- Preventive: prevent problem in the futures (increasing maintainability and reliability)

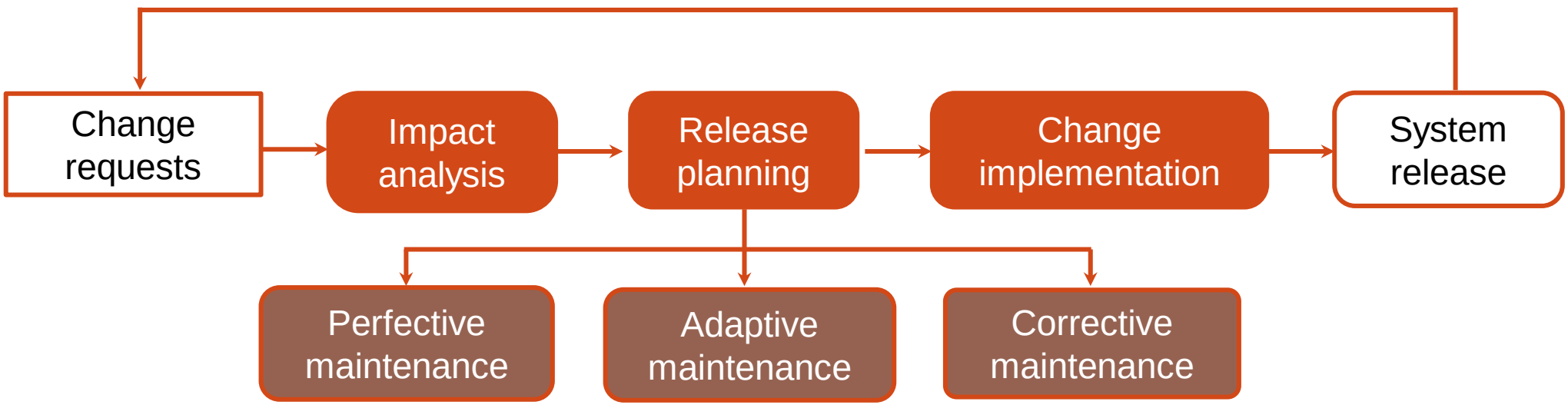
Facts

- 75% on adaptive and perfective;
- 21% on corrective;
- 4% on preventive



Process

- Depending on the types of software being maintained, the development processes used in an organization and people involved in the process



Difficulties

- Factors adversely effect maintenance
  - Lack of models or ignorance of available models (73%)
  - Lack of documentation (67.6%)
  - Lack of time to update existing documentation (54.1%)
- Other factors (1994 study)
  - Quality of original application
  - Documentation quality
  - Rotation of maintenance people

Difficulties

- More factors (Yip ’95 study)
  - Lack of human resources
  - Different programming styles conflict
  - Lack of documentation and tools
  - Bad maintenance management
  - Documentation policy
  - Turnover

Maintenance Cost Factors

- Team stability: maintenance costs are reduced if the same staff are involved with them for some time
- Contractual responsibility: The developers of a system may have no contractual responsibility for maintenance so there is no incentive to design for future change
- Staff skills: Maintenance staff are often inexperienced and have ***limited*** domain ***knowledge***
- Program age and structure: As programs age, their structure is ***degraded*** and they become harder to understand and change

Evolution

Evolution: Definition

- Changing a software product after delivery to adapt such system to the ever-changing user requirements and operating environment
  - The system operates in or address a problem or activity of a real world
- Software maintenance
  - bug fixing or error correction
- Software evolution
  - adaptive and perfective

Software Change

- Software change is inevitable:
  - New requirements emerge when the software is used;
  - The business environment changes;
  - Errors must be repaired;
  - New computers and equipment is added to the system;
  - The performance or reliability of the system may have to be improved.
- A key problem for all organizations is implementing and managing change to their existing software systems

Why Evolve?

- Organisations have huge investments in their software systems: they are critical business assets.
- To maintain the value of these assets to the business, they must be changed and updated.
- The majority of the software budget in large companies is devoted to changing and evolving existing software rather than developing new software.

Lehman's Law

The size and the complexity of a software system will continually increase in its life time; on the other side, the quality of a software system will decrease unless it is ***rigorously*** *maintained and adapted.*

thorough and careful way

Lehman's Law Software Categories

- An *S*-program is written according to an exact specification of what that program can do.
- A *P*-program is written to implement certain procedures that completely determine what the program can do (example: a program to play chess).
- An *E*-program is written to perform some real-world activity; how it should behave is strongly linked to the environment in which it runs, and such a program needs to adapt to varying requirements and circumstances in that environment.

8 Lehman's Law of E-type System

1974

- "Continuing Change" — System must be continually adapted or it becomes progressively less satisfactory.
- "Increasing Complexity" — As system evolves, its complexity increases unless work is done to maintain or reduce it.
- "Self Regulation" — System evolution processes are self-regulating with the distribution of product and process measures close to normal.

8 Lehman's Law of E-type System

1978

- "Conservation of Organisational Stability (invariant work rate)" — the average effective global activity rate in an evolving system is invariant over the product's lifetime.
- "Conservation of Familiarity" — As system evolves, all associated with it, developers, sales personnel and users, for example, must maintain mastery of its content and behavior to achieve satisfactory evolution. Excessive growth diminishes that mastery. Hence the average incremental growth remains invariant as the system evolves.

8 Lehman's Law of E-type System

1991

- "Continuing Growth" — the functional content of a system must be continually increased to maintain user satisfaction over its lifetime.

8 Lehman's Law of E-type System

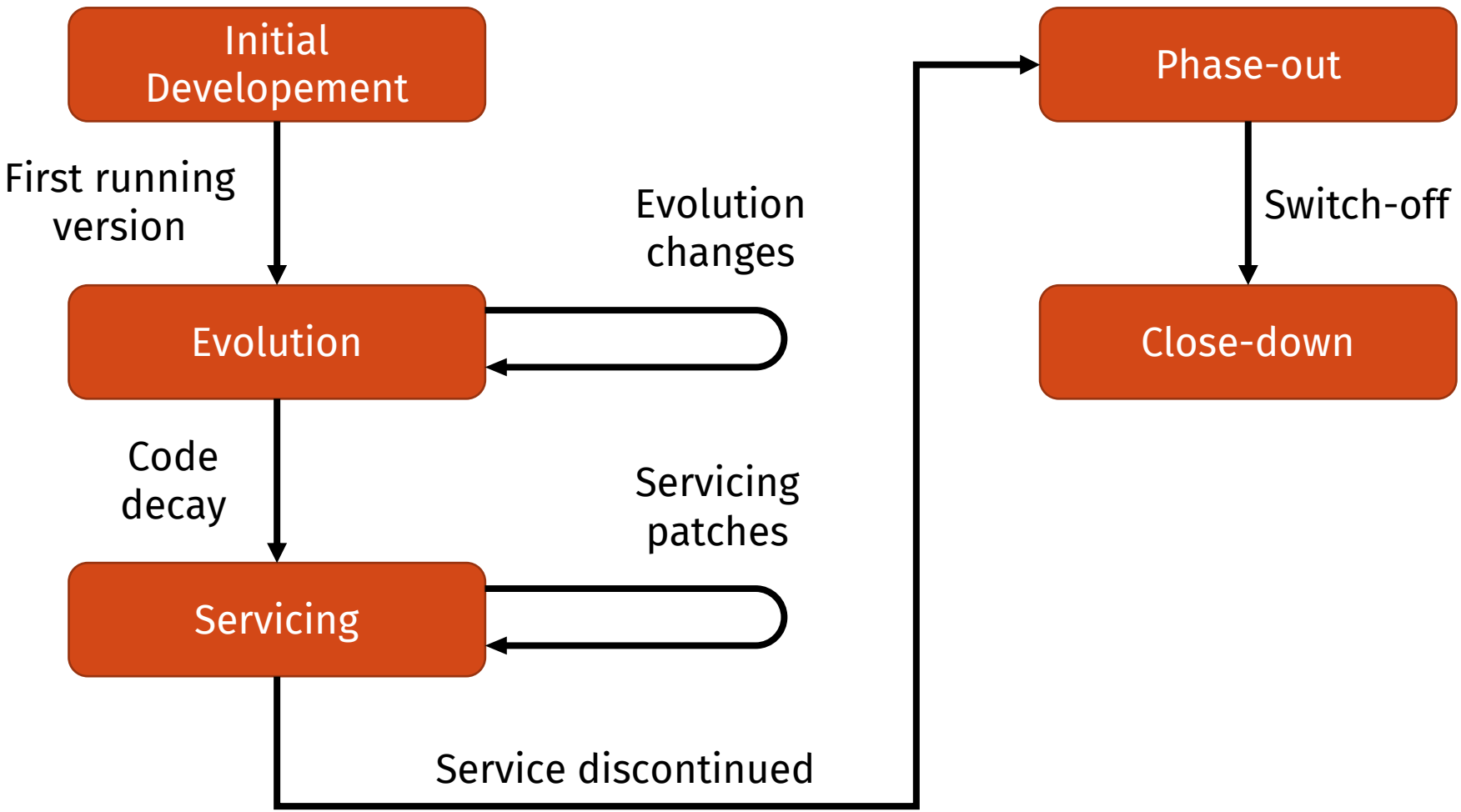
1996

- "Declining Quality" — the quality of a system will appear to be declining unless it is rigorously maintained and adapted to operational environment changes.
- "Feedback System" — 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.

FILKOM
FAKULTAS ILMU KOMPUTER

Evolution Model

Stage Model: Software Life Cycle



Initial development

- Developing the first version
- System architecture proposed even with some lack features
- System knowledge acquired: application domain, user requirements, data formats, algorithms, operating environment, etc.
  - crucial for the evolution

Evolution

- The stage in a software system's life cycle where it is in operational use and is evolving as new requirements are proposed and implemented in the system
- The first version already released successfully:
  - Successful in the marketplace
  - User demand is strong
  - Revenue streams are buoyant
  - The organization is supportive
  - ROI is excellent

Evolution

- Adapts the application to the ever-changing user and operating environment
- Adds new features
- Corrects mistakes and misunderstandings
- Responds to both developer and user learning
- Program usually grows during evolution
- Both software architecture and software team knowledge make evolution possible

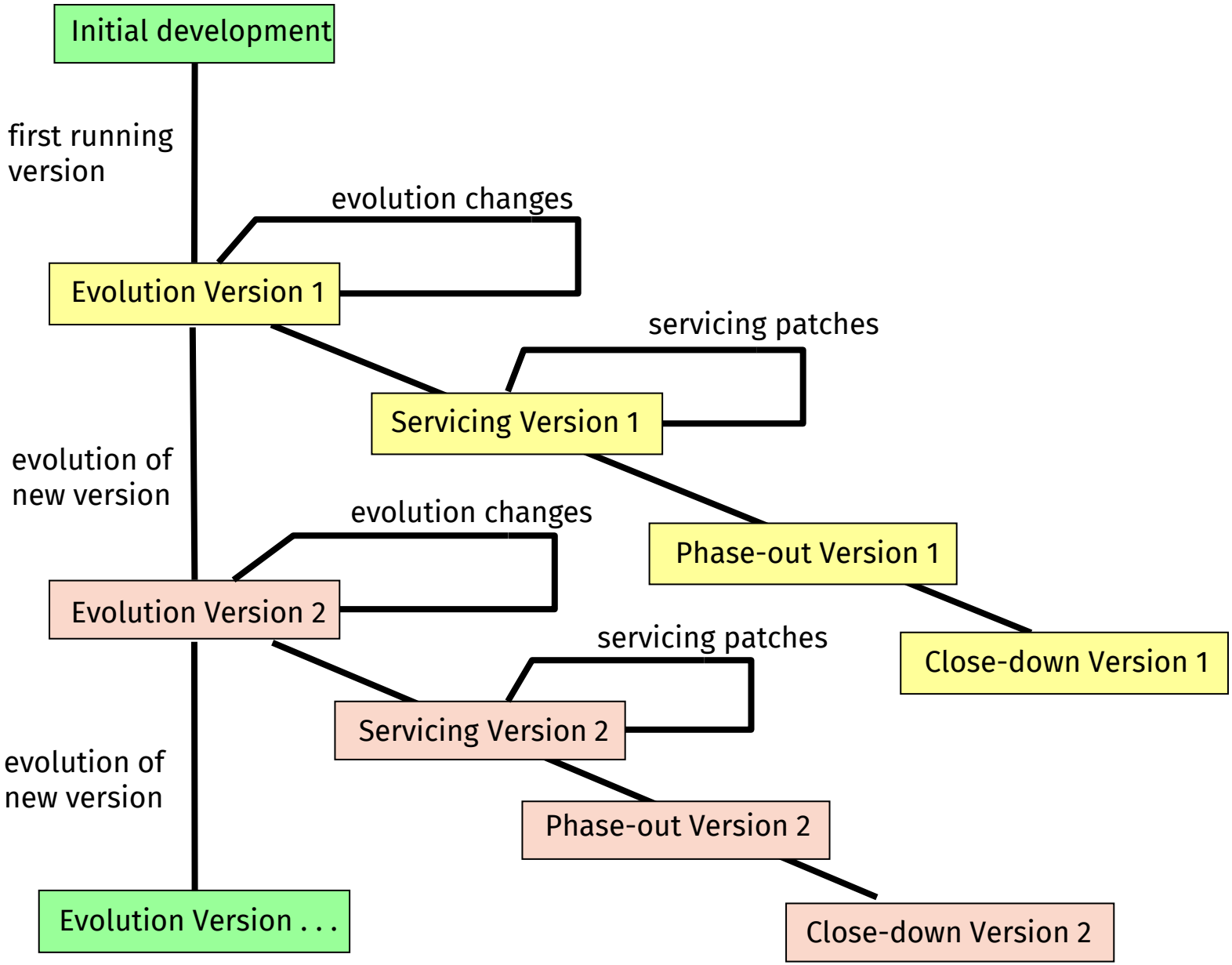
Servicing

- Software maturity
  - no longer evolvable
- The software remains useful but the only changes made are those required to keep it operational i.e. bug fixes and changes to reflect changes in the software's environment
  - small tactical change
- No new functionality is added
- Code decay
  - loss of system knowledge due to the loss of key personnel
    - system goes to servicing
- Transition from evolution is irreversible
  - technical and knowledge problems must be addressed

Phase-out and Close-down

- Phase-out:
  - No more servicing is being undertaken
  - System still may be in production/use
- Close-down:
  - The software use is disconnected
  - Towards replacement
  - An ‘exit strategy’ is needed:
    - Changing to another system requires retraining;
    - What to do with long-lived data?

Versioned Stage Model



Questions?