

# Software Maintenance and Evolution

**Keith H. Bennett**

Research Institute for Software Evolution  
University of Durham

**Václav T. Rajlich**

Department of Computer Science  
Wayne State University  
rajlich@cs.wayne.edu  
<http://www.cs.wayne.edu/~vip>

1

## Aims and Objectives

- present a new model of software lifecycle called the *staged model*
- describe research issues within this framework.
- describe agenda for research in software maintenance and evolution over the next ten years

2

## Basic definitions

- Software maintenance defined in IEEE Standard:  
*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.*
- The term software evolution lacks a standard definition
  - some researchers use it as a substitute for maintenance.
- Our approach:
  - *maintenance* means general post-delivery activities
  - *evolution* to refer to a particular phase in the *staged model* where substantial changes are made to the software

3

## Empirical data of software maintenance

- Software maintenance represents 67- 80 % of software costs
- Survey by Lientz and Swanson
  - late 1970s, very widely cited
  - maintenance activities divided into four classes:
    - Adaptive – changes in the software environment
    - Perfective – new user requirements
    - Corrective – fixing errors (21% of all changes)
    - Preventive – prevent problems in the future.
  - incorporation of *new user requirements* is the **core problem** for software evolution and maintenance (79% of all changes)

4

## Grand challenge of software maintenance

- incorporation of new user requirements quickly and reliably
- **If** changes can be anticipated at design time
  - they can be built in by a parameterization, encapsulations, etc.
  - the problem solved
- **However** 40 years of hard experience confirms:
  - many changes cannot be even *conceived* of by the original designers
  - inability to change software quickly and reliably means that business opportunities are lost
  - our solution: base the software lifecycle on the fact that many changes cannot be predicted

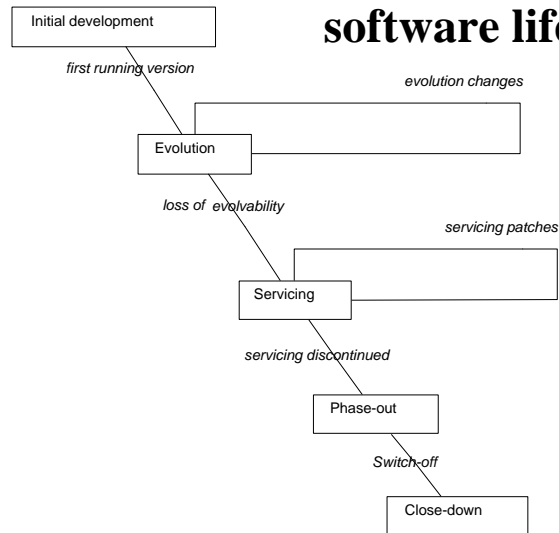
5

## Additional empirical data

- Cusumano and Selby reported that requirements during each iteration may change by 30% or more, as a direct result of the team learning process during the iteration
- Lehner, Pigoski described yearly variations in the frequency of the changes of a long lived system
  - frequency peaks, then declines
  - identifiable differences between phases
- facts:
  - inability to predict changes
  - several different stages, not a uniform “maintenance”

6

## Staged model of software lifecycle



7

## Initial development

- first version of the software system is developed
  - may be lacking some features
- software architecture is established
  - likely to persist through the rest of the life of the program
    - in one documented instance, we studied a program that underwent substantial changes during its 20 years of existence, but it still possesses the architecture of the original first version.
- programming team acquires the *knowledge* of
  - application domain, user requirements, role of the application in the business process, solutions and algorithms, data formats, strengths and weaknesses of the program architecture, operating environment, etc.
  - crucial prerequisite for the next phase of *evolution*.

8

## **Research challenges for initial development**

- To build evolvable software
- in the evolvable architecture, 'the cost of making the change is proportional to the size of the change, not to the size of the overall software system'
- evolvable software can handle unanticipated changes
- 'design for change' should be predominantly aimed at strategic evolution, not code level servicing

9

## **Evolution**

- goals
  - to adapt the application to the ever-changing user and operating environment
  - to correct the faults in the application
  - to respond to both developer and user learning
- inevitability of evolution [Lehman]
- program grows during evolution [Lehner, Lehman]
- business setting of evolution
  - user demand is strong
  - the organization is supportive
  - return on investment is excellent
  - both software architecture and software team knowledge make evolution possible

10

## Code decay

- There is a positive feedback between the loss of software architecture coherence, and the loss of the software knowledge
  - less coherent architecture requires more extensive knowledge in order to evolve it
  - if the knowledge necessary for evolution is lost, the changes in the software will lead to a faster deterioration of the architecture
- Example of loss of knowledge:
  - loss of key personnel
- Research challenge: eliminate or slow code decay

11

## Servicing

- the program is no longer evolvable
- changes are limited to patches and wrappers
  - they are less costly
  - they further deteriorate the architecture.
- Senior designers and architects do not need to be available
- Tools and processes are very different from evolution
- A typical engineer will be assigned only part of the software to support
  - will have partial knowledge of the system.
- The process is stable, well understood and mature.
  - it is well suited to process measurement and improvement

12

## Research issues in servicing

- Making the change without unexpected additional effects
- Program comprehension
- Impact analysis and ripple effect management.
- Concept identification, location and representation.
- Automated tool for code improvement
- Documentation management
- Delivery of service patches
  - Upgrading software without the need to halt it.
- Program health checkers

13

## Reversal from servicing to evolution

- worthy research goal
- in practice:
  - very hard, very rare
- not simply a *technical* problem
  - the *knowledge* of the software team must also be addressed
- for all practical reasons, the transition from evolution to servicing is irreversible

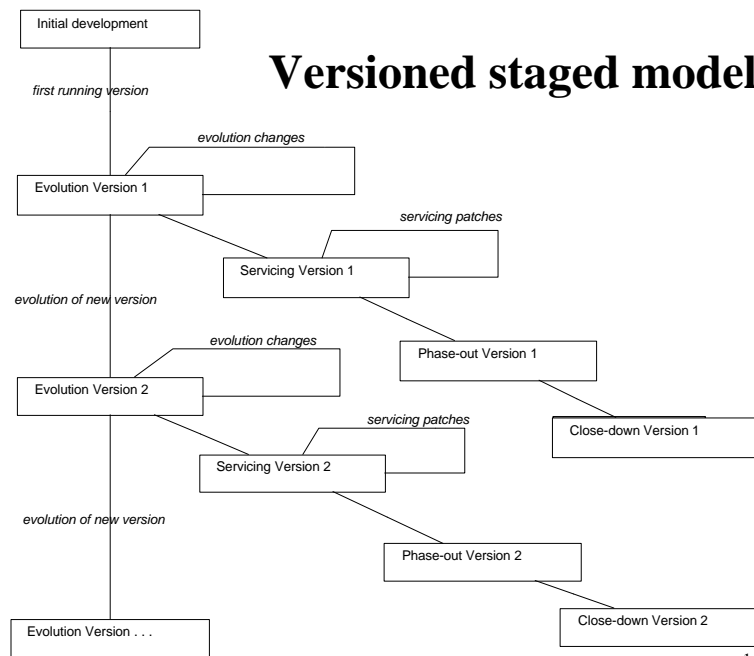
14

## Phase-out and close down stages

- phase-out
  - no more servicing is being undertaken, but the system still may be in production
  - the users must work around known deficiencies
- close-down
  - the software use is disconnected
  - the users are directed towards a replacement.
- business issues:
  - Can any of the software be re-used?
  - 'exit strategy' is needed.
    - once an organization commits to a system, changing to another is expensive, technically difficult, and time consuming.
    - What to do with corporate data?

15

## Versioned staged model



16

## Software change

- basic operation of both software evolution and software servicing
- change mini-cycle consists of the following phases:
  - Request for change
  - Planning phase
    - Program comprehension
    - Change impact analysis
  - Change implementation
    - Restructuring for change
    - Change propagation
  - Verification and validation
  - Re-documentation

17

## Software change

- Program comprehension
  - prerequisite of any change
  - it consumes more than half of all maintenance time
- Change impact analysis
  - it assesses the extent of the change
    - the components that will be impacted by the change
  - it indicates how costly the change is going to be
- Change propagation
  - change may consist of several steps, each visiting one specific software component
  - modified component may no longer fit with the rest
  - neighboring components may need to be changed

18

## Delocalized change

- Not supported by the architecture
- concepts of the application domain relevant to the change are delocalized in the code
- In the case of delocalized changes, an advisable strategy is:
  - to transform the architecture so that the change will be localized
  - then to make the change itself
- research challenge:
  - behavior preserving transformations do not change the behavior of the program, but change the architecture.

19

## Redocumentation

- change is not complete without the update of documentation
- if the documentation is missing or incomplete, the end of the mini-cycle is the opportunity to record the comprehension acquired during the change
  - program comprehension is a very valuable commodity (more than half of resources of software maintenance)
  - in current practice, that value is thrown away when the programmer completes the change and turns his/her attention to new things
- in order to avoid that loss, incremental and opportunistic redocumentation effort is called for.
  - After a time, substantial documentation can be accumulated
- research challenge: structure of documentation, process

20

## **Emergent organizations**

- time-to-market for software has become the top priority for many business applications
  - A finance house may create a new financial product; it must be implemented and launched within 24 hours; and then has a life of only two more days.
- A group of senior software engineering academics and industrialists in UK met regularly to explore and frame visions of the future of software
  - level of abstraction of software engineering will continue to rise.
  - the focus of research will change from technology to the interface of the software with business
  - software will move from product oriented view to service oriented view

21

## **Service oriented view**

- Already some suppliers are making software available on central servers on a pay-per-use basis.
- future: a change in the way the software itself is constructed
  - a federation of services which are only bound together at execution
  - an analogy: making an international telephone call.
    - The caller pays for the use of a range of third party facilities.
    - when a call is made to the same number over a period of time, the telecommunications operator will route the call in different ways on each occasion in order to optimize cost, network traffic and performance

22

## **Conclusions**

- Grand challenge for software maintenance, software engineering: to incorporate new (and unexpected) requirements into software quickly and easily
- We presented a novel model of the software life-cycle, called the staged model
- Staged model was used to describe the research agenda
- We expect software evolution to be at the center of software engineering
- Software evolution needs to be addressed as a business issue as well as a technology issue
  - fundamentally interdisciplinary
- Long-term view of software evolution is based on a service model not a product model

23