Software Lifespan Models

1. Software lifespan models
   1. Stages through which software goes, from conception to death
   2. Stages may be very different
   3. Software = product
      1. Stages are similar to the lifespan of other products
   4. When software is a product
      1. Sales go through the same lifespan
   5. Unique proprietary software
      1. Value follows the same curve
   6. Different names of stages
2. Staged Model
   1. Initial development
      1. First running version
   2. Evolution
      1. Evolution changes
      2. Code decay
   3. Servicing
      1. Servicing patches
      2. Servicing discontinued
   4. Phase-out
      1. Switch-off
   5. Close-down
3. Initial development
   1. Requirements
   2. Design
   3. Implementation
   4. Fundamental commitments
      1. Technology
      2. Architecture
      3. Program domain knowledge
4. Evolution
   1. Adapts the application to the ever-changing user and operating environment
   2. Adds new features
   3. Corrects mistakes and misunderstandings
   4. Responds to both developer and user learning
   5. Program grows during evolution
   6. Both software architecture and software team knowledge make evolution possible
5. Code decay
   1. Positive feedback between
      1. The loss of software architecture coherence
      2. The loss the software knowledge
         1. Less coherent architecture requires more extensive knowledge
         2. If the knowledge is lost, the changes will lead to a faster deterioration
   2. Loss of key personnel == loss of knowledge
   3. Challenge: eliminate or slow code decay
6. Servicing
   1. The program is no longer evolvable
      1. It either decays or managers decide not to support evolution
   2. Changes are limited to patches and wrappers
      1. Less costly, but they cause further deterioration
   3. Process is very different from evolution
      1. No need for senior engineers
      2. Programmer is assigned only part of the software to support
      3. The process is stable
         1. Well suited to process measurements
   4. Challenges:
      1. Making the change without unexpected additional effects
      2. Program comprehension
      3. Documentation management
      4. Delivery of service patches
         1. Upgrading software without the need to halt it
7. Reversal from servicing to evolution (reengineering)
   1. Every expensive, very rare
   2. Not simply a technical problem
      1. The knowledge of the software team must also be addressed
   3. For all practical reasons, the transition from evolution to servicing is irreversible
8. Phase-out
   1. No more servicing is being undertaken
      1. But the system still may be in production
   2. The user must work around known deficiencies
9. Close-down
   1. The software use is disconnected
      1. Current life of successful software
         1. About 10 to 20 years
   2. The users are directed towards a replacement
   3. Exit strategy is needed
      1. Changing to another system is expensive
      2. What od do with long-lived data
10. Versioned staged model
    1. Used by software with many users
    2. Evolution is the backbone of the process
       1. Evolution produces versions
       2. Versions are serviced, phased-out, closed down
11. Incomplete lifespans
    1. Discontinued projects
       1. Stopped during initial development
    2. Stable domain
       1. No need for evolution
12. Lifespan vs. Lifespan model
    1. Lifespan
       1. Common terminology
       2. Incorrect: there is no cycle
          1. Some software discontinued without a replacement
    2. Lifespan model
       1. Better terminology
       2. Less commonly used
13. V-Model
    1. Parallel run after implementation and testing
    2. Unit design 🡨 🡪 Unit testing
    3. System design 🡨 🡪 System testing
    4. Requirements 🡨 🡪 functional testing
    5. Maintenance
14. Prototyping model
    1. Requirements
    2. Prototype
    3. Corrected requirements
    4. Design
    5. Implementation
    6. Maintenance