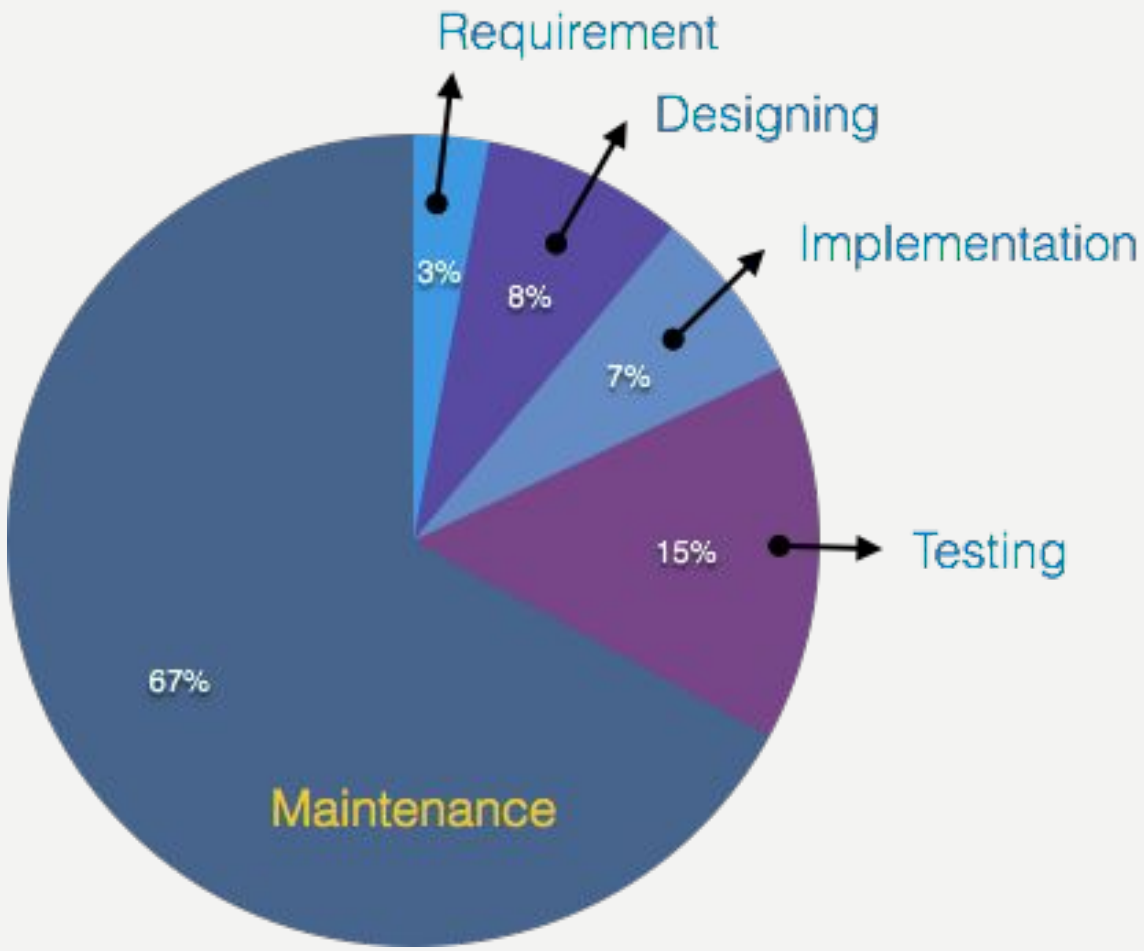
SOFTWARE MAINTENANCE

SOFTWARE MAINTENANCE

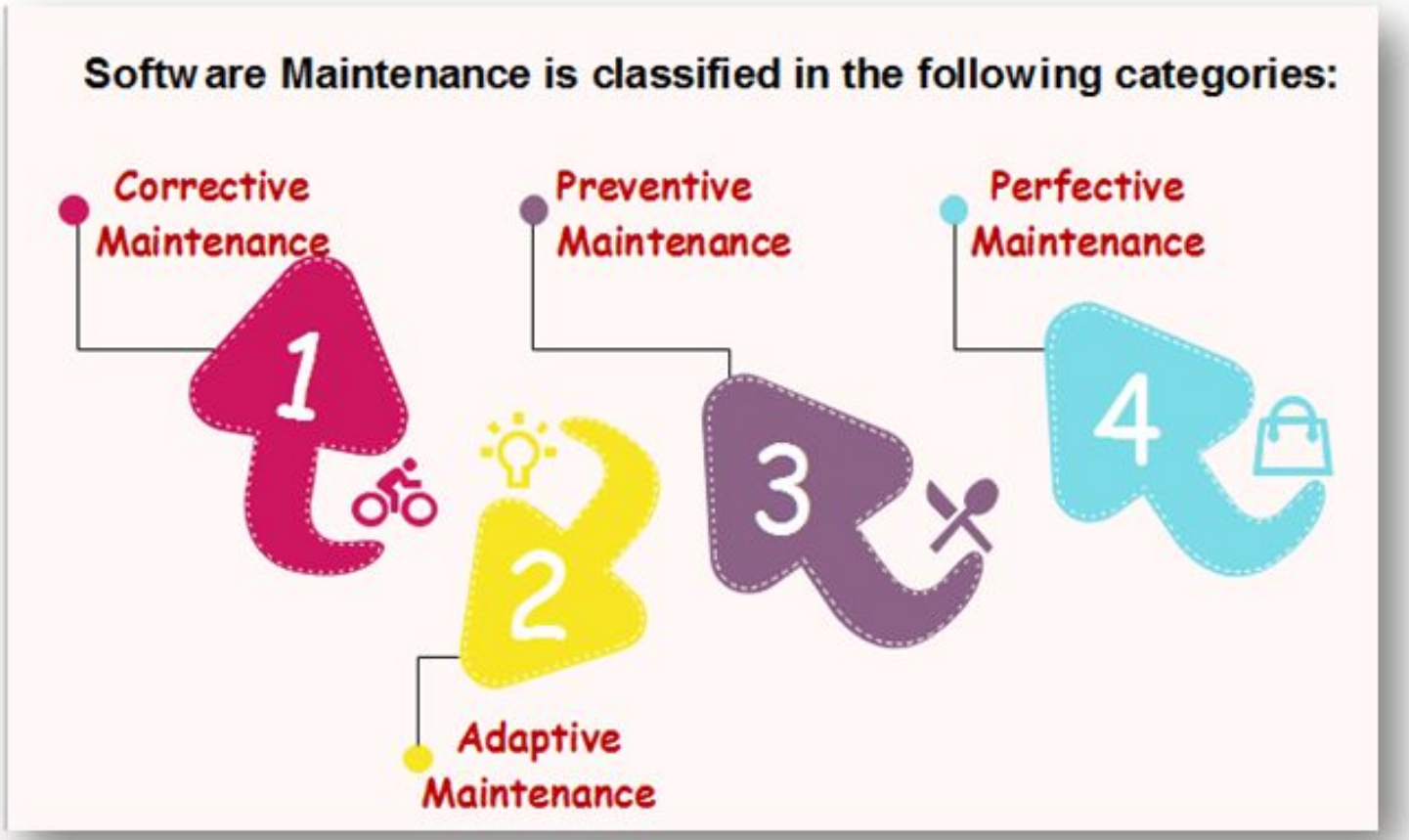
THIS IS WHERE THE MONEY GOES

- Software maintenance: Modification of a software product after delivery to correct faults, to improve performance or other attributes
- Maintain control over systems day to day functions
- Maintain control over system modifications
- Perfect existing functionality
- Prevent system performance from degrading to unacceptable levels
- Well over half of the organizations resources expended on a software system are spent on maintenance
- Design for ease of maintenance should be a primary consideration in the design of any software system

COST OF MAINTENANCE



TYPES OF SOFTWARE MAINTENANCE



TYPES OF SOFTWARE MAINTENANCE

- ***Corrective maintenance***: - repair of errors in the software reported by users
- ***Adaptive maintenance***: - modify software interface to changing environment
- ***Perfective maintenance***: - modify software to increase functionality, improve internal structure or improve performance
- ***Preventive maintenance:*** - modify or rewrite software to improve structure and reliability and to enhance future maintainability

MAINTENANCE MANAGEMENT

- Change control as described previously
- Keep documentation up to date with changes
- Archive systematically
- Do preventive maintenance on software that is in really bad shape
- Monitor maintenance effort to track maintenance costs

Beware of steadily rising maintenance costs, software rot is setting in

MAINTENANCE ACTIVITIES

- Understanding the system
- Locating information in system documentation
- Keeping system documentation up to date
- Extending existing system functions to accommodate new or changing requirements
- Adding new functionality to the system
- Finding the source of system failures or problems
- Locating and correcting faults
- Answering questions about the way the system works
- Restructuring system design and system code
- Deleting design and code components that are obsolete
- Managing changes to the system as they are made

CORRECTIVE MAINTENANCE CYCLE

- User discovers a possible failure of the software
- User calls support hot-line and describes failure
- Front line support personnel create a trouble ticket describing the call
  - Identify the release of the software involved
  - Try to get as much information as possible about the cause of the failure. Has user customized the software?
- Trouble tickets are sent to second level support personnel for assessment
- Trouble ticket assessment
  - Determine if trouble ticket corresponds to a known problem
  - Try to group trouble tickets that appear to be the same fault together
  - Try to determine if the user is using the software correctly
  - Determine severity of the failure(preliminary prioritization)

- Maintenance management monitors outstanding trouble tickets. Assign tickets to maintenance programmers in priority order
- Maintenance programmer receives new trouble ticket.
  - Receive documentation for specified release (from release archive)
  - Try to identify software modules responsible for failure
  - Retrieve relevant source code (using system model and release archive)
  - Create test environment for suspect modules
  - Try to replicate failure on test system
  - Iterate steps above until failure is reproduced or it appears that failure is user caused
  - From failure try to isolate fault causing the failure
  - Prepare change request describing the fault and the probable steps required to correct it.

- Change committee prioritizes change requests and assigns change requests to support programmers in priority order
- support programmers implement and test changes
- Change is integrated into next release of the software. Quick patch may be distributed to affected users
- Updates required
  - Software design documents
  - User and international documentation
  - System model
  - Archive all modified software in version archive
  - Add test cases to test suite
  - Status of trouble ticket and change request.

INFORMATION TO AID MAINTENANCE

- Complete and exact description of each software release. System model and version control archive.
- Complete and correct description of the process used to build the software
- Complete and correct internal and external documentation for each release
- Tools and hooks to discover user customization and/or modification of the software
- Test suite for release from release archive
- Debugging and testing output built into the system
- Ability to compare system models for different releases

SOFTWARE MAINTENANCE ISSUES

- Understanding current state of software after many versions and releases. Changes inadequately documented.
- Hard to determine the process that was used to create the software.
- Very hard to understand someone else's program. Especially if code is poorly documented.
- Originator of the software may not be available for consultation.
- Documentation may be poor.. Non-existent. Documentation may not be current
- Software was not designed to be maintainable.
- Source files may be difficult to locate.
- Maintenance is unglamorous, maintenance group can become a dumping ground for least effective programmers.

MAINTENANCE PROBLEMS – STAFF RELATED

- Limited understanding of software being maintained.
  - Maintenance programmers don't understand the system
  - Users don't understand the system. Increases maintenance workload
- Management priorities
  - May not get enough time/budget to do maintenance well
  - Sloppy maintenance causes problems in the future
- Morale
  - Maintenance staff have low social/economic status in the organization
- **Possible cures**
  - good documentation, kept up to date
  - Educate management on importance (cost/benefit) of doing maintenance right.
  - Reward maintenance staff generously. Rotate staff between maintenance and development.

MAINTENANCE PROBLEMS - TECHNICAL

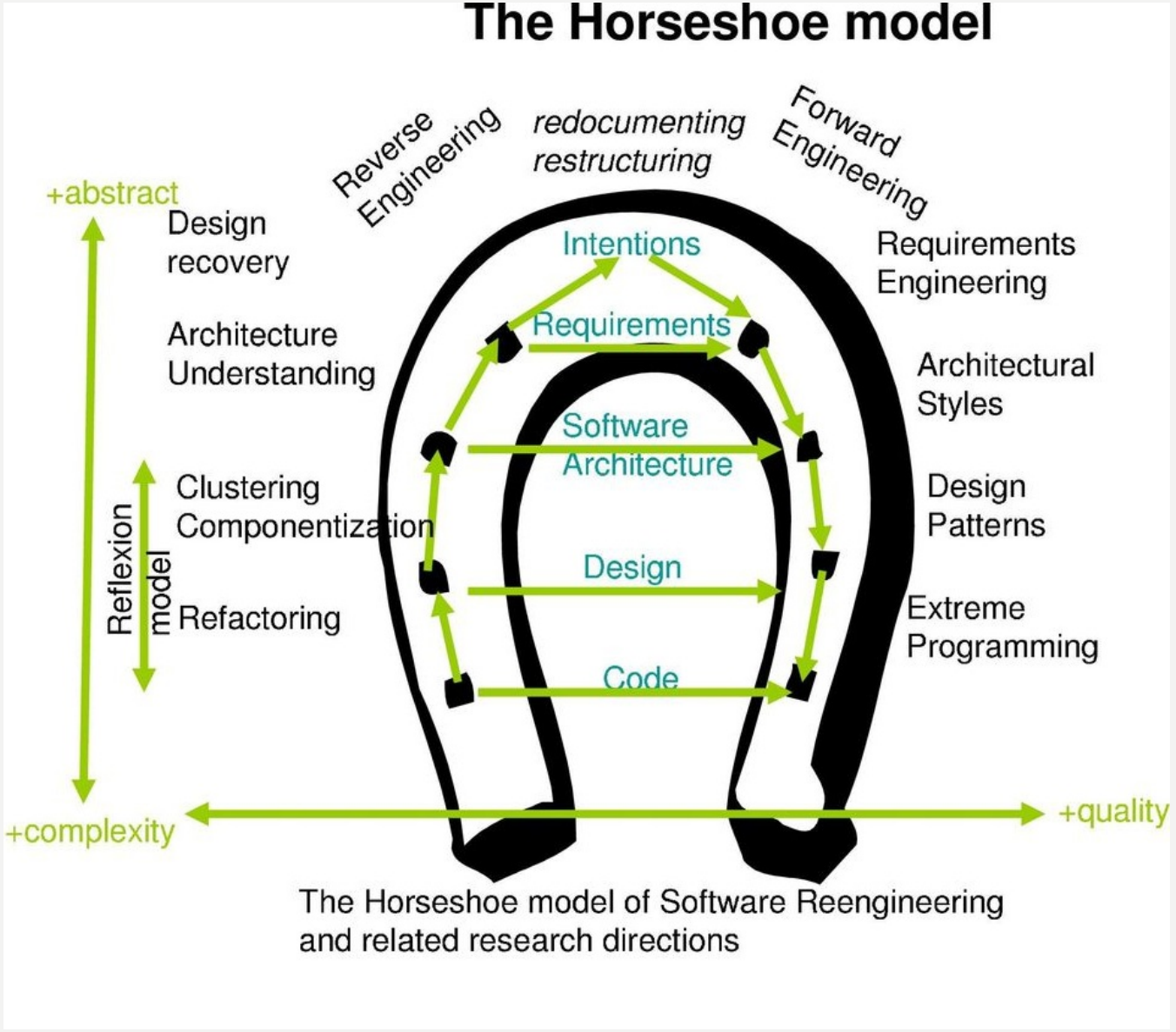
- Artifacts and paradigms
  - Original design logic convoluted and/or hard to understand
  - Designers failed to participate (correctly) future changes
    - Eg: Y2K problems
  - Object oriented programs harder to maintain due to intricate object interconnectivity
  - Very intricate programs (eg: high coupling, low cohesion) are inherently more difficult to maintain (correctly).
- Testing issues
  - May need expensive duplication of production environment for testing
  - May be difficult to generate test data for new hardware
  - May be difficult to determine how to devise tests for new change

FACTORS AFFECTING MAINTENANCE EFFORT

- Type of application
- Novelty of the software
- Turnover and maintenance staff availability
- Expected system life span
- Dependence on a changing environment
- Hardware characteristics
- Design quality
- Code quality
- Documentation quality
- Testing quality

MAINTAIN VS REPLACE

- Is the cost of maintenance too high?
- Is the system reliability unacceptable?
- Has the system lost the ability to adapt to further change at a reasonable cost and schedule?
- Is the system performance beyond prescribed constraints?
- Are system functions of limited usefulness?
- Can other systems do the same job better, faster, cheaper?
- Has hardware maintenance cost become excessive, so that hardware should be replaced?



The Horseshoe model of Software Reengineering and related research directions

SOFTWARE REDOCUMENTATION

- Develop new documentation to assist maintenance efforts
- Use static analysis tools to process the source code
  - Component calling relationships
  - Data dictionary information
  - Data flow information
  - Control flow information
  - Design recovery
  - Possible test paths
  - Cross reference information, functions & variables

SOFTWARE RESTRUCTURING

- Restructure software to make it easier to understand and change
- Use tools to analyze source code
  - Improve modularity through clustering analysis
  - Try to reduce coupling and increase cohesion
  - Try to impose good structure on ill-structured code

SOFTWARE REFACTORING

- Restructuring existing code by altering its internal structure without changing its external behavior.
- You begin with a program that runs correctly, but is not well structured, refactoring improves its structure, making it easier to maintain and extend.
- Are following activities a refactoring technique?
  - Adding new functionalities
  - Fixing functionality bugs
  - Tuning performance
  - Patching security

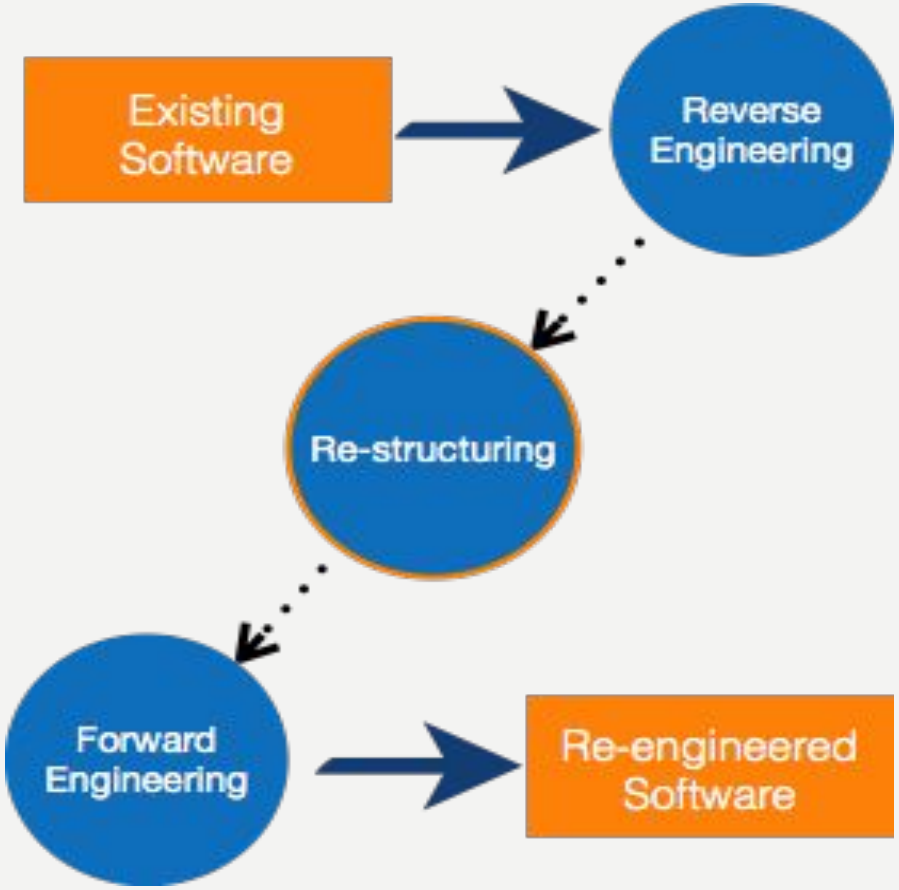
REFACTORING RHYTHMS

- one of the most difficult things to get over is the rhythm of refactoring.
- This is the way you do small step by small step, slowly improving code quality.
- Small steps=
  - Extract methods
  - Move methods
  - Rename methods
  - Replace temp with query
  - Replace conditional with polymorphism
  - Replace type code with state/strategy
  - Self encapsulate field
  - …..

SOFTWARE RE-ENGINEERING

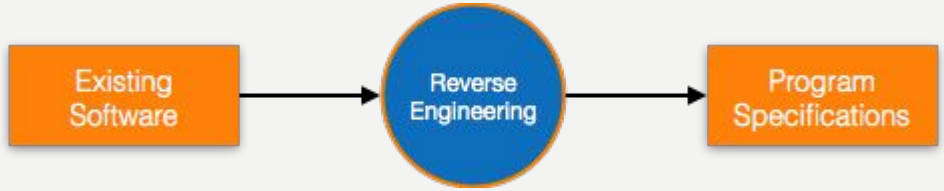
- Software quality and structure decays over time
  - Maintenance by "patching" the software
  - Documentation of patches may be poor or non-existent
  - Source code gets lost due to programmer turnover, inadequate archives, organizational changes
  - Long term maintenance may introduce errors and inefficiencies into the software
- Software re-engineering involves the implementation of all or key parts of important software systems
- Goal of re-engineering is to reduce maintenance costs and to improve software quality
- Re-engineering may require *design recovery* from *binary(!!)* programs
  - Because source code or original design have been lost
  - Because patched software doesn't correspond to any documentation

SOFTWARE RE-ENGINEERING



SOFTWARE REVERSE ENGINEERING

- Extract specification and design information from existing source code
- Analyze data usage and calling patterns
- Attempt to identify and specify underlying functionality
- Reverse engineering difficulty increases with complexity of the software



FORWARD ENGINEERING

- Obtaining desired software from the specifications in hand which were brought down by means of reverse engineering