CS 123
Introduction to Software Engineering

08: Post-Delivery Maintenance

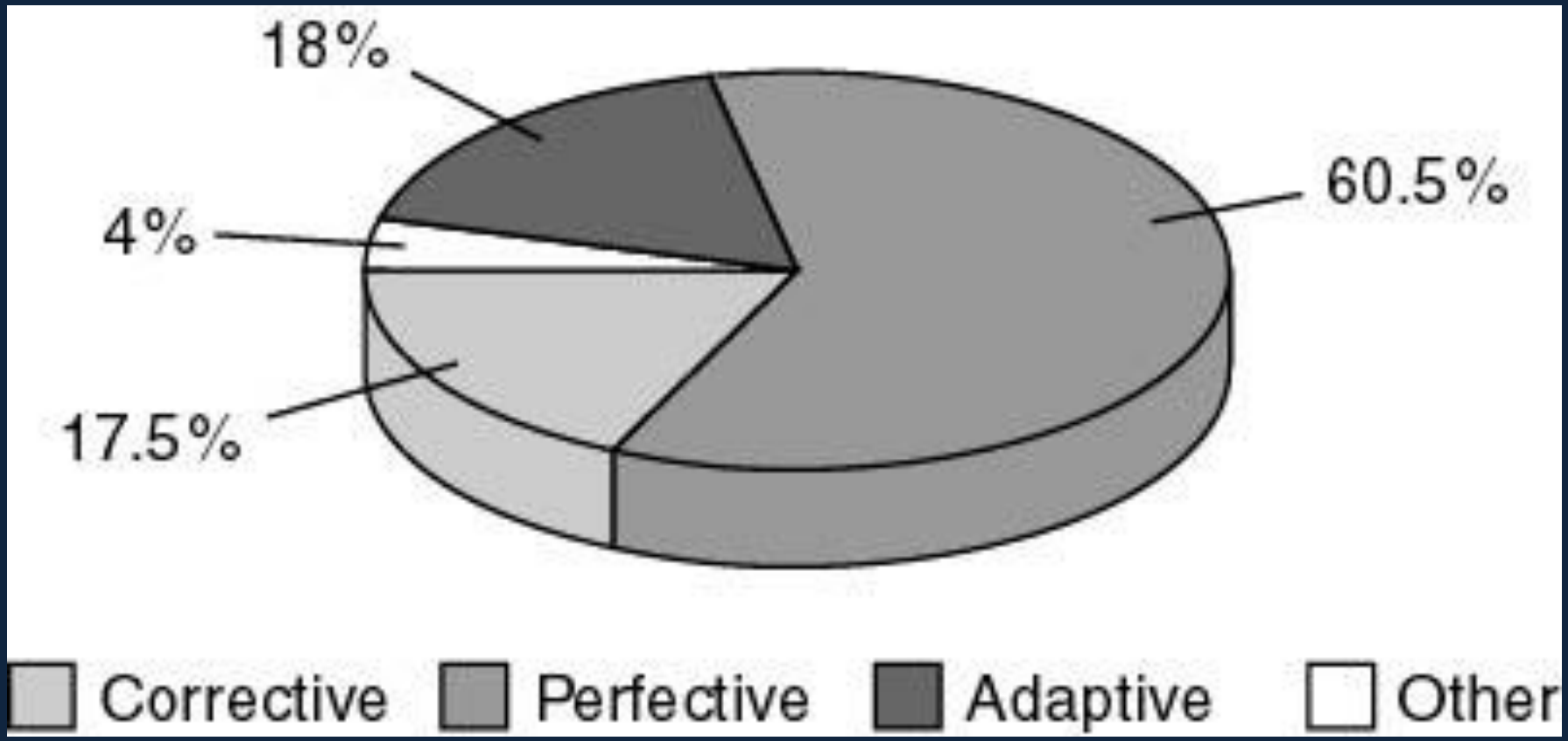
Overview

- Why maintenance is necessary
- What is required of maintenance programmers
- Maintenance case study
- Management of maintenance
- Maintenance of object-oriented software
- Maintenance skills versus development skills
- Reverse engineering
- Testing during the maintenance phase
- CASE tools for the maintenance phase
- Metrics for the maintenance phase
- Challenges of the maintenance phase

Maintenance Phase

- Post-delivery Maintenance
  - *Any* change to *any* component of the product (including documentation) after it has passed the acceptance test
- This is a short chapter
- But the whole course is on maintenance
- Why is maintenance necessary?

Types of maintenance



Types of maintenance (contd)

- Corrective maintenance
  - To correct residual faults
    - Specification, design, implementation, documentation, or any other types of faults
  - On average, 17.5% of maintenance

Types of maintenance (contd)

- Perfective maintenance
  - Client requests changes to improve product effectiveness
    - Add additional functionality
    - Make product run faster
    - Improve maintainability
  - On average, 60.5% of maintenance

Types of maintenance (contd)

- Adaptive maintenance
  - Responses to changes in environment in which product operates
    - Product ported to new compiler, operating system, and/or hardware
    - Change to tax code
    - 9-digit ZIP codes
  - On average, 18% of maintenance

Difficulty of Maintenance

- About 67% of the total cost of a product accrues during the maintenance phase
- Maintenance is a major income source
- Nevertheless, even today many organizations assign maintenance to
  - Unsupervised beginners, and
  - Less competent programmers

What is Required of Maintenance Programmers?

- Maintenance is one of the most difficult aspects of software production because
  - Maintenance incorporates aspects of all other phases
- Suppose a fault report is handed to a maintenance programmer
- What tools does the maintenance programmer have to find the fault?
  - The fault report filed by user
  - The source code
  - And often nothing else

What is Required of Maintenance Programmers?

- Maintenance programmer must have superb debugging skills
  - The fault could lie anywhere within the product
  - The original cause of the fault might lie in the by now non-existent specifications or design documents

Corrective Maintenance

- Suppose that the maintenance programmer has located the fault
- Problem
  - How to fix it without introducing a regression fault

Corrective Maintenance (contd)

- How to minimize regression faults
  - Consult the detailed documentation for product as a whole
  - Consult the detailed documentation for each individual module
- What usually happens
  - There is no documentation at all, or
  - The documentation is incomplete, or
  - The documentation is faulty
- The programmer must deduce from the source code itself all the information needed to avoid introducing a regression fault

Corrective Maintenance (contd)

- Now the programmer changes the source code
- Test that the modification works correctly
  - Use specially constructed test cases
- Check for regression faults
  - Use stored test data
- Add specially constructed test cases to stored test data for future regression testing
- Document all changes

Corrective Maintenance (contd)

- Major skills required for corrective maintenance
  - Superb diagnostic skills
  - Superb testing skills
  - Superb documentation skills

Adaptive and Perfective Maintenance

- The maintenance programmer must go through the phases of
  - Requirements/Analysis
  - Design
  - Implementation and integration
- Using the existing product as a starting point

Adaptive and Perfective Maintenance (contd)

- When programs are developed
  - Specifications are produced by specification experts
  - Designs are produced by design experts
  - Implementation is performed by implementation experts
  - Integration is performed by integration experts
  - Testing is performed by testing experts
  - Documentation is produced by documentation experts

Adaptive and Perfective Maintenance (contd)

- But every maintenance programmer needs to be an expert in
  - Specifications
  - Design
  - Implementation
  - Integration
  - Testing
  - Documentation

Adaptive and Perfective Maintenance (contd)

- Conclusion
- No form of maintenance
  - Is a task for an unsupervised beginner, or
  - Should be done by a less skilled computer professional

The Rewards of Maintenance

- Maintenance is a thankless task in every way
  - Maintainers deal with dissatisfied users
  - If the user were happy,the product would not need maintenance
  - The user's problems are often caused by the individuals who developed the product, not the maintainer
  - The code itself may be badly written
  - Maintenance is despised by many software developers
  - Unless good maintenance service is provided, the client will take future development business elsewhere
  - Maintenance is the most important phase of software production, the most difficult—and most thankless

The Rewards of Maintenance (contd)

- How can this situation be changed?
- Managers must assign maintenance to their best programmers, and
- Pay them accordingly

Maintenance Case Study

- Temperate Fruit Committee
- Software was developed for exactly 7 temperate fruits
  - Apples, apricots, cherries, nectarines, peaches, pears, and plums
- It was extended to include kiwi fruit, with little difficulty
- The product now needed to handle 26 additional fruits
- “Just do the same thing 26 times”

Maintenance Case Study (contd)

- Lessons to be learnt from this
  - The problem was caused by the developer, not the maintainer
  - A maintainer is often responsible for fixing other people's mistakes
  - The client frequently does not understand that maintenance can be difficult, or all but impossible
  - This is exacerbated when previous apparently similar perfective and adaptive maintenance tasks have been carried out
  - All software development activities must be performed with an eye on future maintenance

Management of Maintenance

- We need a mechanism for changing a product
- If the product appears to function incorrectly, the user files a fault report
  - It must include enough information to enable the maintenance programmer to recreate the problem
- Ideally, every fault should be fixed immediately
  - In practice, immediate preliminary investigation

Management of Maintenance (contd)

- The maintenance programmer should first consult the fault report file
- All reported faults not yet fixed, and
- Suggestions for working around them

Management of Maintenance (contd)

- Previously reported fault
- Give information in fault report file to user

Management of Maintenance (contd)

- New fault
- Maintenance programmer should try to find
  - Cause of fault
  - A way to fix it
  - A way to work around problem
- New fault is now filed in the fault report file, together with supporting documentation
  - Listings
  - Designs
  - Manuals

Management of Maintenance (contd)

- The file should also contain the client's requests for perfective and adaptive maintenance
  - Contents of file must be prioritized by the client
  - The next modification is the one with the highest priority
- Copies of fault reports must be circulated to all users
  - Including: estimate of when the fault can be fixed
- If the same failure occurs at another site, the user can determine
  - If it is possible to work around the fault, and
  - How long until it can be fixed

Management of Maintenance (contd)

- In an ideal world
  - We fix every fault immediately
  - Then we distribute the new version of product to all sites
- In the real world
  - We distribute fault reports to all sites
  - We do not have the staff for instant maintenance

Making Changes to the Product

- Corrective maintenance
  - Assign a maintenance programmer to determine the fault and its cause, then repair it
  - Test the fix, test the product as a whole (regression testing)
  - Update the documentation to reflect the changes made
  - Update the prologue comments to reflect
    - What was changed,
    - Why it was changed,
    - By whom, and
    - When

Making Changes to the Product (contd)

- Adaptive and perfective maintenance
  - As with corrective maintenance, except there is no fault report
  - There is a change in requirements instead

Making Changes to the Product (contd)

- What if the programmer has not tested the fix adequately?
  - Before the product is distributed, it must be tested by the SQA group
- Maintenance is extremely hard
- Testing is difficult and time consuming

Ensuring Maintainability

- Maintenance is not a one-time effort
- We must plan for maintenance over the entire life cycle
  - Design phase—use information-hiding techniques
  - Implementation phase—select variable names meaningful to future maintenance programmers
  - Documentation must be complete and correct, and reflect current version of every module

Ensuring Maintainability (contd)

- During the maintenance phase, maintainability must not be compromised
  - Always be conscious of the inevitable further maintenance
- Principles leading to maintainability are equally applicable to the maintenance phase itself

The Problem of Repeated Maintenance

- Moving target problem
  - Frustrating to development team
  - Frequent changes have an adverse effect on the product
- Apparent solution
  - Construct a rapid prototype
  - Change it as frequently as the client wants
  - Once the client is finally satisfied, the specifications are approved and product is constructed
- In practice
  - The client can change the requirements next day
  - If willing to pay the price, the client can change the requirements daily

Problem of Repeated Maintenance (contd)

- The problem exacerbated during the maintenance phase
- The more changes there are
  - The more the product deviates from its original design
  - The more difficult further changes become
  - Documentation becomes even less reliable than usual
  - Regression testing files are not up to date
  - A total rewrite may be needed for further maintenance

Moving Target Problem (contd)

- Clearly a management problem
- In theory
  - Developers must explain the problem at start of the project
  - Specifications can be frozen until delivery
  - Specifications can be frozen after perfective maintenance
- In practice
  - It does not work that way
  - A client with clout can order changes and they are implemented
- “He who pays the piper calls the tune”

Warning

- It is no use implementing changes slowly
- The relevant personnel are replaced
- Nothing can be done if the person calling for repeated change has sufficient clout

Maintenance of Object-Oriented Software

- The object-oriented paradigm promotes maintenance
  - The product consists of independent units
  - Encapsulation (conceptual independence)
  - Information hiding (physical independence)
  - Message-passing is the sole communication
- Reality is somewhat different!
- Three obstacles
  - The complete inheritance hierarchy can be large
  - The consequences of polymorphism and dynamic binding
  - The consequences of inheritance

Size of Inheritance Hierarchy

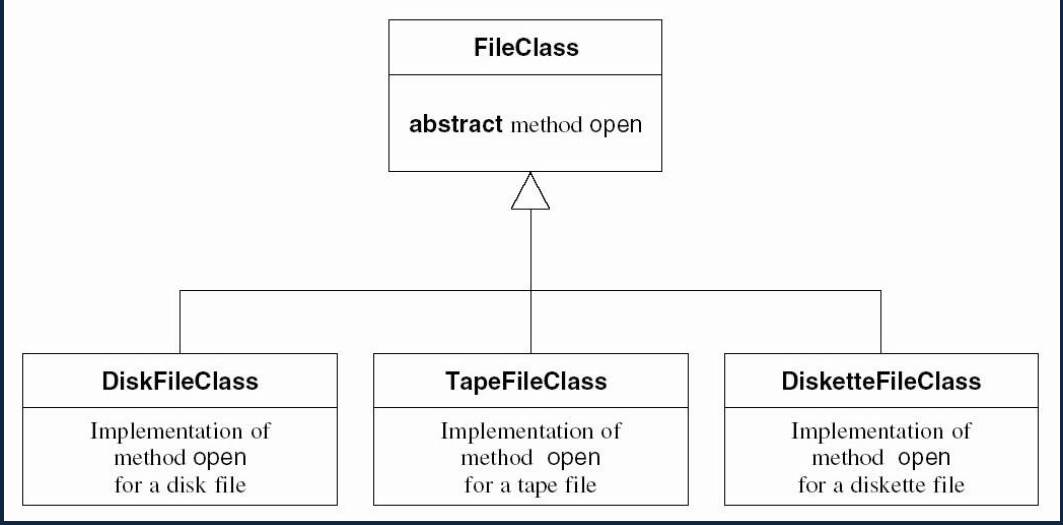
```
{
   ...
   void displayNode (Node a);
   ...
}// class UndirectedTree

class DirectedTree : public UndirectedTree
{
   ...
}// class DirectedTree

class RootedTree : public DirectedTree
{
   ...
   void displayNode (Node a);
   ...
}// class RootedTree

class BinaryTree : public RootedTree
{
   ...
}// class BinaryTree

class BalancedBinaryTree : public BinaryTree
{
   Node          hhh;
   displayNode (hhh);
}// class BalancedBinary Tree
```

Size of Inheritance Hierarchy (contd)

- To find out what displayNode does in BalancedBinaryTree, we must scan the entire tree
  - Inheritance tree may be spread over the entire product
  - A far cry from "independent units"
- Solution
  - A CASE tool can flatten the inheritance tree
  - Eiffel flat

Polymorphism and Dynamic Binding

- Product fails on the invocation myFile.open ()
- Which version of open contains the fault?
  - A CASE tool cannot help (static tool)
  - We must trace



Polymorphism and Dynamic Binding (contd)

- Polymorphism and dynamic binding can have
  - Positive effect on development
  - Negative effect on maintenance

Consequences of inheritance

- Create new subclass by inheritance
  - Does not affect superclass
  - Does not affect any other subclass
- Modify this new subclass
  - Again, no affect
- Modify a superclass
  - All descendent subclasses are affected
- Inheritance can have
  - Positive effect on development
  - Negative effect on maintenance

Maintenance versus Development Skills

- Skills needed for maintenance include
  - Ability to determine cause of failure of large product
    - Also needed during integration and product testing
  - Ability to function effectively without adequate documentation
    - Documentation rarely complete until delivery
- Skills in specification, design, implementation, testing
  - All four activities carried out during development
- Skills needed for maintenance same as those for other phases

Maintenance versus Development Skills

- Key Point
  - Maintenance programmers must not merely be skilled in broad variety of areas, they must be *highly* skilled in *all* those areas
  - Specialization impossible for the maintenance programmer
- Maintenance is the same as development, only more so

Reverse Engineering

- When the only documentation is the code itself
  - Start with the code
  - Recreate the design
  - Recreate the specifications (extremely hard)
  - CASE tools help (flow charters, other visual aids)

Reverse Engineering (contd)

- Definitions
  - Reengineering
    - Reverse engineering, followed by forward engineering
    - Lower to higher to lower levels of abstraction
  - Restructuring
    - Improving product without changing functionality
    - Prettyprinting, structuring code, improving maintainability

Reverse Engineering

- What if we have only the executable code?
  - Treat as black box

Testing during the Maintenance Phase

- Maintainers view a product as loosely related modules
  - They were not involved in development of the product
- Regression testing is essential
  - Store test cases and outcomes, modify as needed

CASE Tools for the Maintenance Phase

- Version, revision control tools
- Reengineering tools
  - Battlemap, Teamwork, Bachman Product Set

Challenges of the Maintenance Phase

- The development-then-maintenance model is unrealistic today
  - The client's requirements frequently change before the product is delivered
  - Faults often have to be fixed before the product is delivered
  - Development from scratch is almost unknown today.
    - Instead, products are built from reused components.
- Products are modified before delivery

Future of Software Engineering

- Most people have 20-20 hindsight
  - Forecasting the future is much harder!
- Possible scenario 1
  - Emphasis on formal techniques
  - Formal object-oriented techniques
- Possible scenario 2
  - Intensive reuse of hundreds of standard classes
- Possible scenario 3
  - Totally different to anything currently envisioned