Software Maintenance

Isaac Griffith

CS 3321
Department of Computer Science
Idaho State University

Topics covered

- Legacy Systems Continued
- Software maintenance

Environment assessment factors

- **Supplier stability**
  - Is the supplier still in existence?
  - Is the supplier financially stable and likely to continue in existence?
  - If the supplier is no longer in business, does someone else maintain the systems?
- **Failure rate**
  - Does the hardware have a high rate of reported failures?
  - Does the support software crash and force system restarts?
- **Age**
  - How old is the hardware and software?
  - The older the hardware and support software, the more obsolete it will be. It may still function correctly but there could be significant economic and business benefits to moving to a more modern system.

Environment assessment factors

- **Performance**
  - Is the performance of the system adequate?
  - Do performance problems have a significant effect on system users?
- **Support requirements**
  - What local support is required by the hardware and software?
  - If there are high costs associated with this support, it may be worth considering system replacement.
- **Maintenance costs**
  - What are the costs of hardware maintenance and support software licenses?
  - Older hardware may have higher maintenance costs than modern systems. Support software may have high annual licensing costs.
- **Interoperability**
  - Are there problems interfacing the system to other systems?
  - Can compilers, for example, be used with current versions of the operating system? Is hardware emulation required?

Application assessment factors

- **Understandability**
  - How difficult is it to understand the source code of the current system?
  - How complex are the control structures that are used?
  - Do variables have meaningful names that reflect their function?
- **Documentation**
  - What system documentation is available?
  - Is the documentation complete, consistent, and current?
- **Data**
  - Is there an explicit data model for the system?
  - To what extent is data duplicated across files?
  - Is the data used by the system up to date and consistent?
- **Performance**
  - Is the performance of the application adequate?
  - Do performance problems have a significant effect on system users?

Application assessment factors

- **Programming language**
  - Are modern compilers available for the programming language used to develop the system?
  - Is the programming language still used for new system development?
- **Configuration management**
  - Are all versions of all parts of the system managed by a configuration management system?
  - Is there an explicit description of the versions of components that are used in the current system?

Application assessment factors

- **Test data**
  - Does test data for the system exist?
  - Is there a record of regression tests carried out when new features have been added to the system?
- **Personnel skills**
  - Are there people available who have the skills to maintain the application?
  - Are there people available who have experience with the system?

System measurement

- You may collect quantitative data to make an assessment of the quality of the application system
  - The number of system change requests; The higher this accumulated value, the lower the quality of the system.
  - The number of different user interfaces used by the system; The more interfaces, the more likely it is that there will be inconsistencies and redundancies in these interfaces.
  - The volume of data used by the system. As the volume of data (number of files, size of database, etc.) processed by the system increases, so too do the inconsistencies and errors in that data.
  - Cleaning up old data is a very expensive and time-consuming process

Software Maintenance

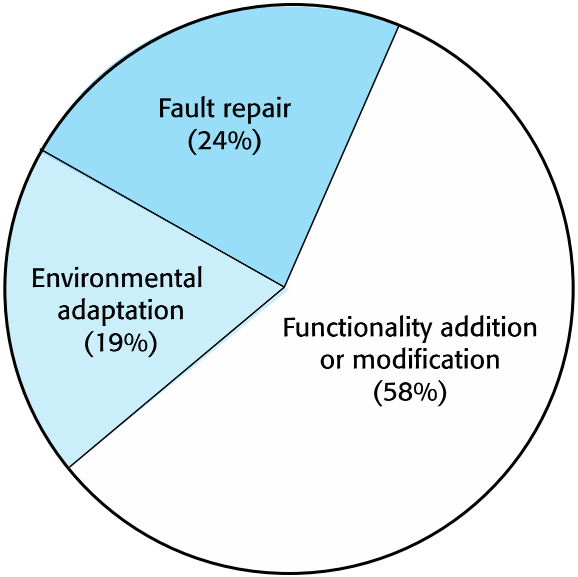
Software maintenance

- Modifying a program after it has been put into use.
- The term is mostly used for changing custom software. Generic software products are said to evolve to create new versions.
- Maintenance does not normally involve major changes to the system's architecture.
- Changes are implemented by modifying existing components and adding new components to the system.

Types of maintenance

- Fault repairs
  - Changing a system to fix bugs/vulnerabilities and correct deficiencies in the way meets its requirements.
- Environmental adaptation
  - Maintenance to adapt software to a different operating environment
  - Changing a system so that it operates in a different environment (computer, OS, etc.) from its initial implementation.
- Functionality addition and modification
  - Modifying the system to satisfy new requirements.

Maintenance effort distribution



Maintenance costs

- Usually greater than development costs (2* to 100* depending on the application).
- Affected by both technical and non-technical factors.
- Increases as software is maintained. Maintenance corrupts the software structure so makes further maintenance more difficult.
- Aging software can have high support costs (e.g. old languages, compilers etc.).

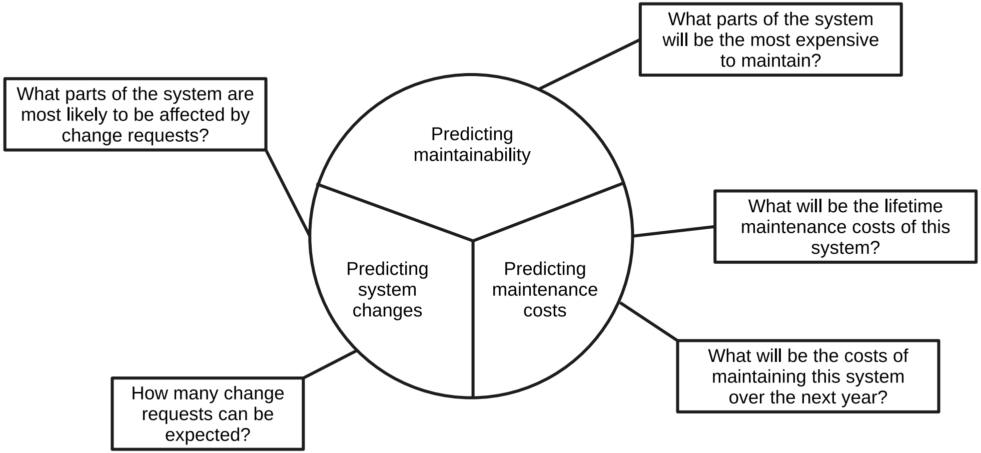
Maintenance costs

- It is usually more expensive to add new features to a system during maintenance than it is to add the same features during development
  - A new team has to understand the programs being maintained
  - Separating maintenance and development means there is no incentive for the development team to write maintainable software
  - Program maintenance work is unpopular
    - Maintenance staff are often inexperienced and have limited domain knowledge.
  - As programs age, their structure degrades and they become harder to change

Maintenance prediction

- Maintenance prediction is concerned with assessing which parts of the system may cause problems and have high maintenance costs
  - Change acceptance depends on the maintainability of the components affected by the change;
  - Implementing changes degrades the system and reduces its maintainability;
  - Maintenance costs depend on the number of changes and costs of change depend on maintainability.

Maintenance prediction


What parts of the system will be the most expensive to maintain?
What parts of the system are most likely to be affected by change requests?
Predicting maintainability
What will be the lifetime maintenance costs of this system?
Predicting system changes
Predicting maintenance costs
What will be the costs of maintaining this system over the next year?
How many change requests can be expected?

Change prediction

- Predicting the number of changes requires and understanding of the relationships between a system and its environment.
- Tightly coupled systems require changes whenever the environment is changed.
- Factors influencing this relationship are
  - Number and complexity of system interfaces;
  - Number of inherently volatile system requirements;
  - The business processes where the system is used.

Complexity metrics

- Predictions of maintainability can be made by assessing the complexity of system components.
- Studies have shown that most maintenance effort is spent on a relatively small number of system components.
- Complexity depends on
  - Complexity of control structures;
  - Complexity of data structures;
  - Object, method (procedure) and module size.

Process metrics

- Process metrics may be used to assess maintainability
  - Number of requests for corrective maintenance;
  - Average time required for impact analysis;
  - Average time taken to implement a change request;
  - Number of outstanding change requests.
- If any or all of these is increasing, this may indicate a decline in maintainability.

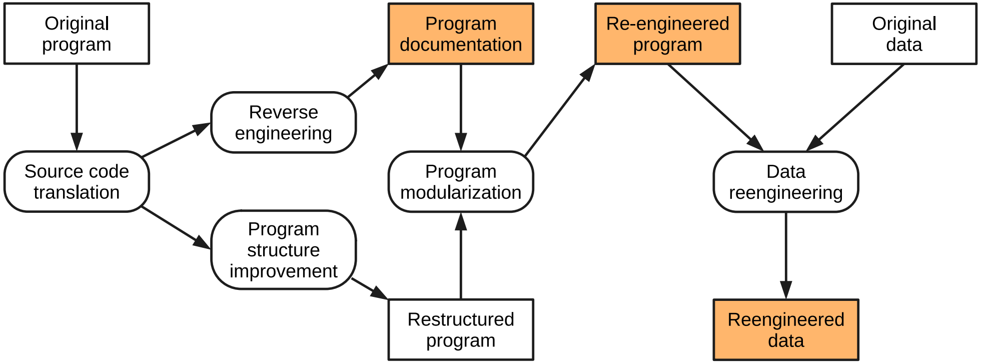
Software reengineering

- Restructuring or rewriting part or all of a legacy system without changing its functionality.
- Applicable where some but not all sub-systems of a larger system require frequent maintenance.
- Reengineering involves adding effort to make them easier to maintain. The system may be re-structured and re-documented.

Advantages of reengineering

- Reduced risk
  - There is a high risk in new software development. There may be development problems, staffing problems and specification problems.
- Reduced cost
  - The cost of re-engineering is often significantly less than the costs of developing new software.

The reengineering process



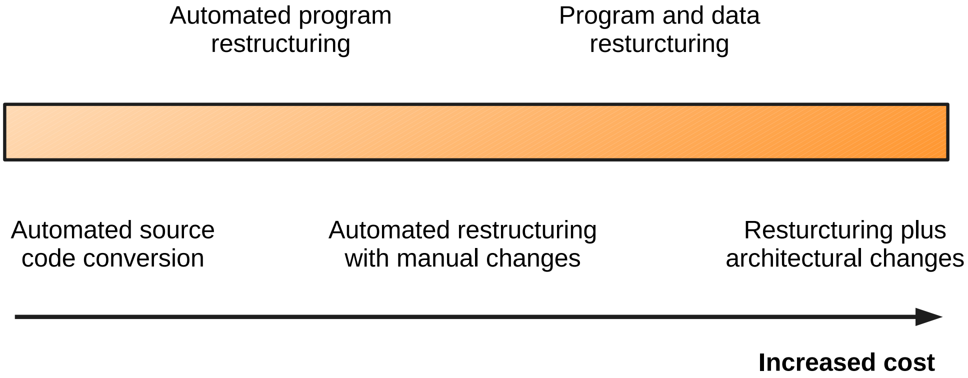
Idaho State University
Computer Science

Reengineering process activities

- Source code translation
  - Convert code to a new language.
- Reverse engineering
  - Analyse the program to understand it;
- Program structure improvement
  - Restructure automatically for understandability;
- Program modularisation
  - Reorganise the program structure;
- Data reengineering
  - Clean-up and restructure system data.

ROAR

Reengineering approaches


Automated program restructuring
Program and data resturcturing
Automated source code conversion
Automated restructuring with manual changes
Resturcturing plus architectural changes
Increased cost

Reengineering cost factors

- The quality of the software to be reengineered.
- The tool support available for reengineering.
- The extent of the data conversion which is required.
- The availability of expert staff for reengineering.
  - This can be a problem with old systems based on technology that is no longer widely used.

Refactoring

- Refactoring is the process of making improvements to a program to slow down degradation through change.
- You can think of refactoring as ‘preventative maintenance’ that reduces the problems of future change.
- Refactoring involves modifying a program to improve its structure, reduce its complexity or make it easier to understand.
- When you refactor a program, you should not add functionality but rather concentrate on program improvement.

Refactoring and reengineering

- Re-engineering takes place after a system has been maintained for some time and maintenance costs are increasing. You use automated tools to process and re-engineer a legacy system to create a new system that is more maintainable.
- Refactoring is a continuous process of improvement throughout the development and evolution process. It is intended to avoid the structure and code degradation that increases the costs and difficulties of maintaining a system.

‘Bad smells’ in program code

- Duplicate code
  - The same or very similar code may be included at different places in a program. This can be removed and implemented as a single method or function that is called as required.
- Long methods
  - If a method is too long, it should be redesigned as a number of shorter methods.
- Switch (case) statements
  - These often involve duplication, where the switch depends on the type of a value. The switch statements may be scattered around a program. In object-oriented languages, you can often use polymorphism to achieve the same thing.

‘Bad smells’ in program code

- Data clumping
  - Data clumps occur when the same group of data items (fields in classes, parameters in methods) re-occur in several places in a program. These can often be replaced with an object that encapsulates all of the data.
- Speculative generality
  - This occurs when developers include generality in a program in case it is required in the future. This can often simply be removed.

Key points

- There are 3 types of software maintenance, namely bug fixing, modifying software to work in a new environment, and implementing new or changed requirements.
- Software re-engineering is concerned with re-structuring and re-documenting software to make it easier to understand and change.
- Refactoring, making program changes that preserve functionality, is a form of preventative maintenance.

Are there any questions?