Software Evolution and Maintenance

Mojtaba Shahin

Week #11: Lecture - Part 2

Topics

- Part 2
  - Software Maintenance
  - Software Reengineering

Notes and Acknowledgements

- Slides/images come from the following main sources:
  - **Chapter 9**: Ian Sommerville, Software Engineering, 10th Edition, 2015.
    - https://iansommerville.com/software-engineering-book/slides/

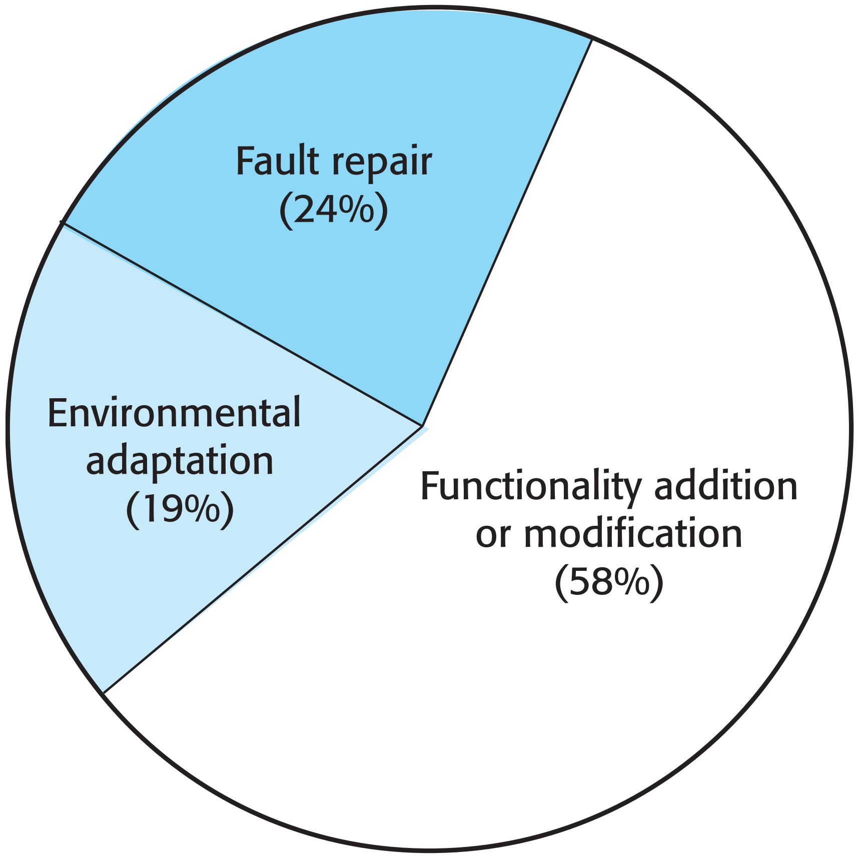
Software maintenance

- **Modifying a program** after it has been put into use.
- Maintenance does not normally involve **major changes to the system's architecture**.
- Changes in maintenance are implemented by modifying existing components and adding new components to the system.

Types of maintenance

- **Fault repairs [Corrective maintenance]**
  - Changing a system to fix bugs/vulnerabilities/errors.
    - **Coding errors** are usually relatively cheap to correct;
    - **Design errors** are more expensive because they may involve rewriting several program components.
    - **Requirements errors** are the most expensive to repair because extensive system redesign may be necessary.
- **Environmental adaptation [Adaptive maintenance]**
  - Maintenance to adapt the 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 [Perfective maintenance]**
  - Modifying the system to satisfy new requirements.

Maintenance effort distribution



Maintenance costs

- Usually greater than development costs (2* to 100* depending on the application).
- It is usually **more expensive** to add new features to a system during maintenance than it is to add the same features during development because
  - –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

Legacy Systems and Maintenance

- Software maintenance involves
  - (1) understanding the program that has to be changed and
  - (2) then implementing any required changes.
- However, many systems, especially older legacy systems, are difficult to **understand** and **change**.
- To make legacy software systems easier to maintain, you can re-engineer these systems to improve their structure and understandability

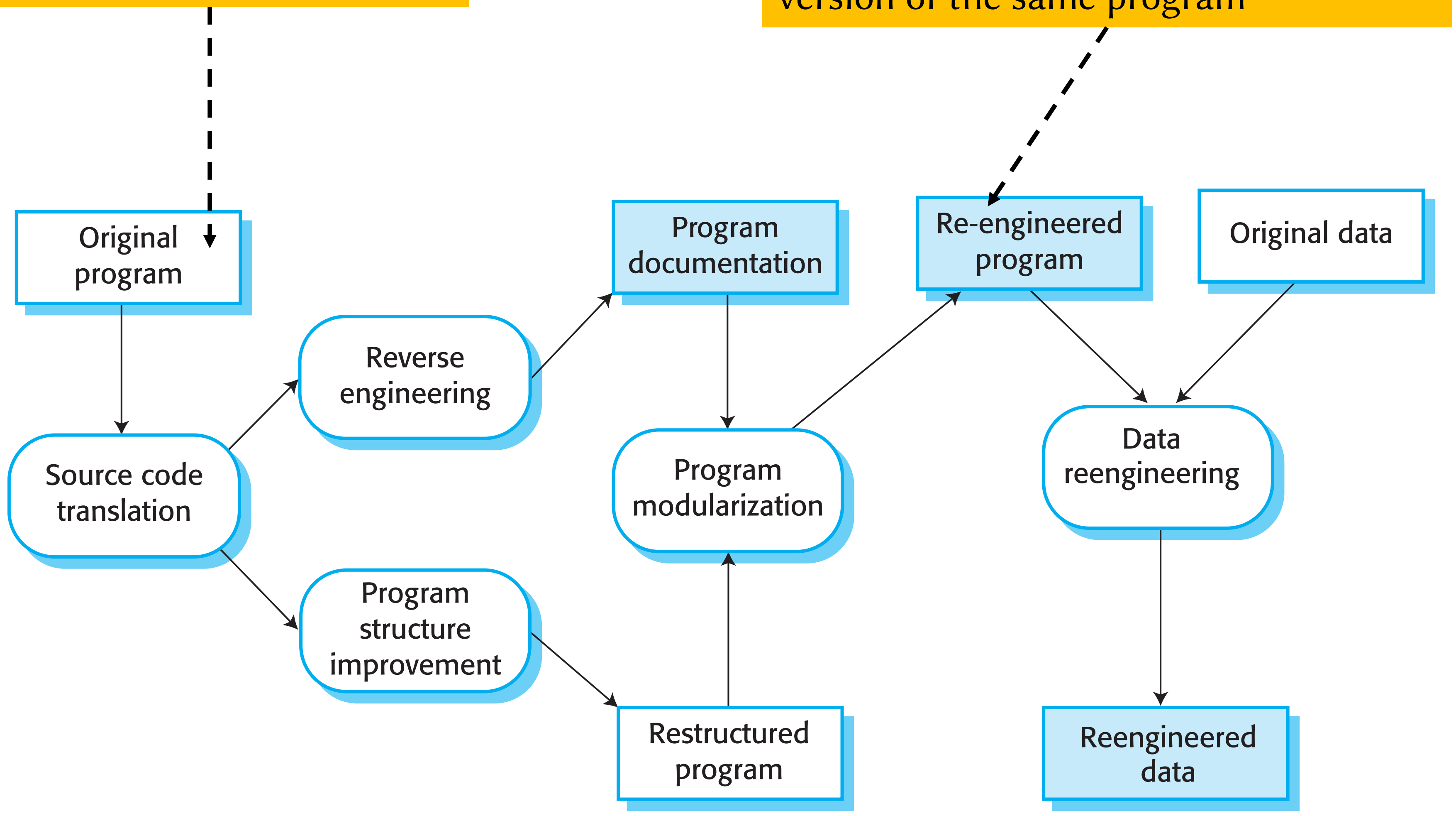
Software Reengineering

- Restructuring or rewriting part or all of a legacy system **without changing its functionality**.
- Reengineering may involve
  - redocumenting the system,
  - refactoring the system architecture,
  - Translating programs to a modern programming language, or
  - modifying and updating the structure and values of the system's data

The reengineering process

Input: a legacy program

Output: An improved and restructured version of the same program



Reengineering process activities

- **Source code translation**
  - Using a translation tool, you can convert the program from an old programming language to a more modern version of the same language or to a different language.
- **Reverse engineering**
  - The program is analyzed and information extracted from it. This helps to document its organization and functionality.
  - This process is usually completely automated.
- **Program structure improvement**
  - The control structure of the program is analyzed and modified to make it easier to read and understand.
  - This can be partially automated, but some manual intervention is usually required.

Reengineering process activities

- **Program modularization**
  - Reorganize the program structure;
  - In some cases, this stage may involve architectural refactoring (e.g., a system that uses several different data stores may be refactored to use a single repository).
  - This is a manual process.
- **Data reengineering**
  - Clean up and restructure system data.

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

- **Reengineering** takes place after a system has been maintained for **some time** and maintenance costs are **increasing**.
- **Refactoring** is a continuous process of improvement throughout the development and evolution process.

References

- **Chapter 9**: Ian Sommerville, Software Engineering, 10th Edition, 2015.
  - https://iansommerville.com/software-engineering-book/slides/

Thanks!

Mojtaba Shahin

mojtaba.shahin@rmit.edu.au