21

Techniques Used for Reverse Engineering

Shah Nawaz,
Lecturer, Department of Software Engineering,
Lahore Garrison University, shahnawaz@lgu.edu.pk

Decompilation Versus Reverse Engineering

Decompilation Versus Reverse Engineering

Decompilation

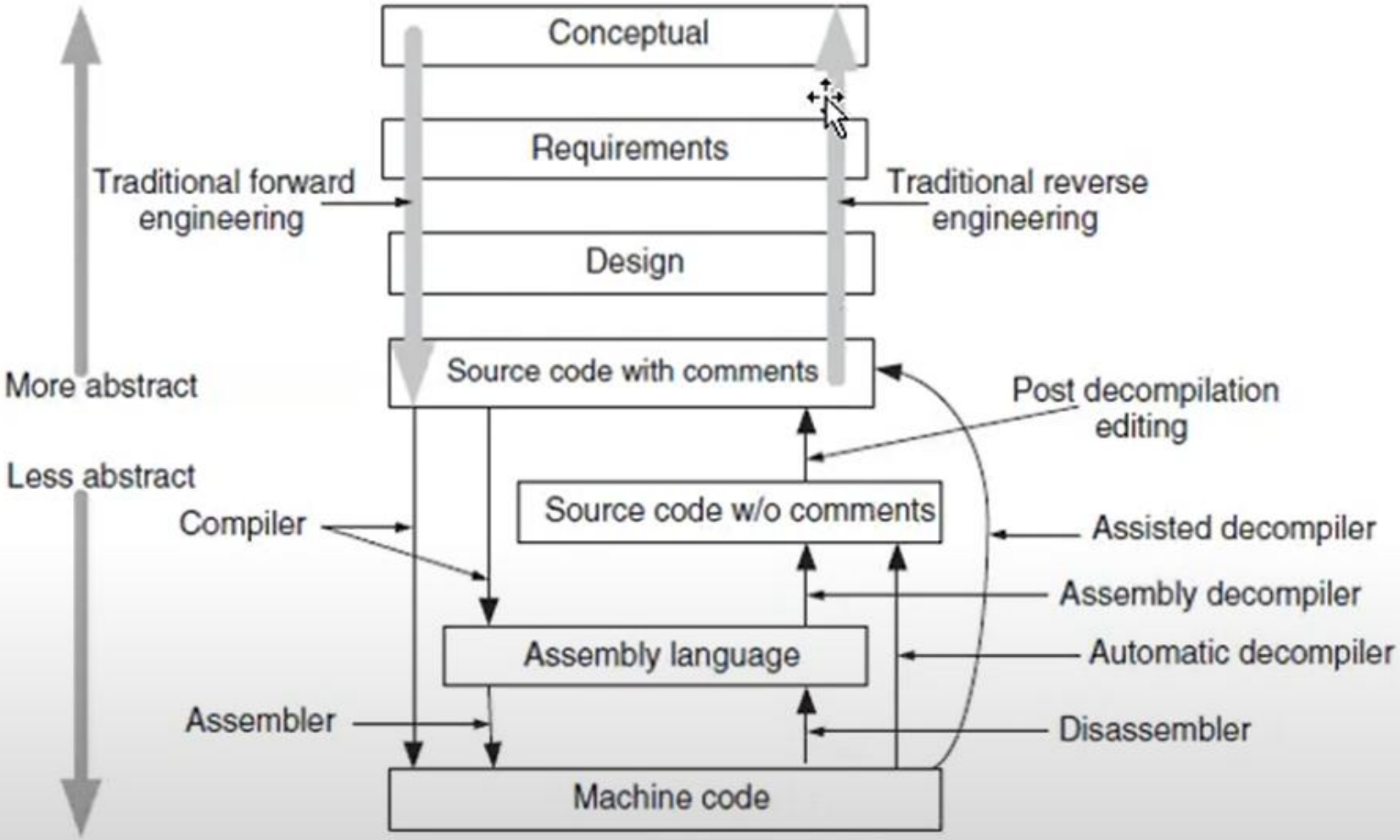
- Decompilation is a form of static analysis which is investigating a program by not running it.
- A decompiler takes an executable binary file and attempts to produce readable highlevel language source code from it.
- The output will, in general, not be the same as the original source code, and may not even be in the same language. Actual recovery of the original source code is not really possible.
- There are a number of different reasons for decompilation or disassembly, such as **understanding a program, recovering the source code for purposes of archiving or updating, finding virus es, debugging programs, and translating obsolete code.**

- Decompilation is sometimes used unethically, to reproduce source code for reuse or adaptation without permission of the copyright holder. Programs can be designed to be resistant to decompilation through protective means such as obfuscation .

- **decompiler (dotpeek for .Net, Hex Rays for C/C++)**

Decompilation Versus Reverse Engineering

Decompilation



Relationship between decompilation and traditional reengineering.

Data Reverse Engineering

i- Data Structure Extraction

ii- Data Structure Conceptualization

Data Reverse Engineering

- Database reverse engineering is the process through which the logical and conceptual schemas of a legacy database, or of a set of files, are reconstructed from various information sources such as DDL code, data dictionary contents, database contents or the source code of application programs that use the database.
- DRE is defined as "the use of structured techniques to reconstitute the data assets of an existing system"

- The two vital aspects of a DRE process are
  - **(i) recover *data assets* that are valuable and**
  - **(ii) *reconstitute* the recovered data assets to make them more useful.**

Data Reverse Engineering

In practice, the purpose of DRE is as follows:

- ***Knowledge acquisition.*** Knowledge acquisition is a method of learning. It includes elicitation, collection, analysis, modeling, and validation of information for software projects. The need for knowledge acquisition is pivotal to reverse engineering of *data-oriented applications*. The data portion—be it flat files or a relational database—must be clearly understood in a reverse engineering process.
- ***Tentative requirements.*** DRE of an operational system can identify the tentative requirements of the replacement system. DRE ensures that the functionality of the current system is not forgotten or overlooked.
- ***Documentation.*** DRE improves the documentation of existing systems, especially when the original developers are no longer available for advice. Maintenance of legacy software is assisted by the new documentation.
- ***Integration.*** DRE facilitates integration of applications, because (i) a logical model of encompassed software is a prerequisite for integration and (ii) a logical model of encompassed software presents a plausible model of how the program will function in certain environmental conditions.

Data Reverse Engineering

- ***Data administration.*** As data are increasingly used as information, the data owner must be able to perform data administration easily and pragmatically. DRE allows companies to manage data correctly and efficiently. Data administration, also known as data resource management, is an organizational function working in the areas of information systems that plans, organizes, describes, and controls data resources.

- ***Data conversion.*** One needs to understand the logical connection between the old database and the new one before converting the old data. Data conversion is the migration of the data instance from the old database to the new one.

- ***Software assessment.*** DRE can be used to assess the database management system (DBMS) schema of the software. A relational database contains a **catalog** that describes the various elements in the system. The catalog divides the database into sub-databases known as schemas. Within each schema are database objects—tables, views, and privileges. The catalog itself is a set of tables with its own schema name.

Data Reverse Engineering

- ***Quality assessment.*** The overall quality of a software system can be assessed with DRE techniques, because a flawed design of a persistent data structure is likely to lead to faults in the software system.

- ***Component reuse.*** The concept of reuse has increasingly become popular amongst software engineers. DRE tools and techniques offer the opportunity to access and extract software components. Quite often these components need to be modified one way or another before they can be reused.

Data Reverse Engineering

Forward design process of a database

- The forward design process of a database comprises three basic phases as follows:
- ***Conceptual phase.*** In this phase, user requirements are gathered, studied, and formalized into a conceptual schema. The phase of conceptual schema has no impact on reverse engineering.

- ***Logical phase.*** In this phase, the conceptual schema is expressed as a simple model, which is suitable for optimization reasoning. Independent of the target DBMS, the model can be optimized. Next, it is translated according to the target model. The model can be further optimized according to data management system (DMS)-dependent rules.

- ***Physical phase.*** Now the logical schema is described in the data description language (DDL) of the DMS and the host programming language. The views needed by the application programs are expressed partly in DDL and partly in the host language.

Data Reverse Engineering

The process is divided into two main phases, namely:

i - Data Structure Conceptualization

- The complete DMS schema, including the structures and constraints, are recovered in this phase.
- In general, database systems provide a description of this schema in some machine processable form. Schemas are a rich starting point to begin a DBRE process. Schemas can be refined through further analysis of other components of the application such as subschema, screen and report layouts, procedures, and documentation.

Data Reverse Engineering

i - Data Structure Conceptualization

In this methodology, data structures are extracted by means of the following main processes:

- ***DMS–DDL text analysis.*** Data structure declaration statements in a given DDL, found in the schema scripts and application programs, are analyzed to produce an approximate logical schema.
- ***Program analysis.*** This means analyzing the source code in order to detect integrity constraints and evidences of additional data structures.
- ***Data analysis.*** This means analyzing the files and databases to (i) identify data structures and their properties, namely, unique fields and functional dependencies in files and (ii) test hypothesis such as "could this field be a foreign key to this file?"
- ***Schema integration.*** The analyst is generally presented with several schemas while processing more than one information source. Each of those multiple schemas offers a partial view of the data objects. All those partial views are reflected on the final logical schema via a process for *schema integration*.

Data Reverse Engineering

ii - Data Structure Conceptualization

- The process within reverse engineering that aims at deriving a plausible conceptual schema from the logical schema of a database.
- In this phase, one detects and transforms or discards redundancies, DMS-dependent constructs, technical optimization, and nonconceptual structures.

Data Reverse Engineering

ii - Data Structure Conceptualization

The phase comprises two sub-phases:

- ***Basic conceptualization.*** In this sub-phase, relevant semantic concepts are extracted from an underlying logical schema, by solving two different problems requiring very different methods and reasoning: *schema untranslation* and *schema de-optimization*.
- ***Conceptual normalization.*** The basic conceptual schema is restructured for it to have the desired qualities, namely, simplicity, readability, minimality, extensibility, and expressiveness. Examples of conceptual normalization are (i) replace some entity types by relationship types; (ii) replace some entity types by attributes; (iii) make the *is–a* relation explicit; and (iv) standardize the names.