A Comparative Study of Forward and Reverse Engineering

Dhaval Jha[1] Bansal Shah[2] Dhwanish Shah[3] Saloni Shah[4] Yash Shah[5]
[1,2,3,4,5]Department of Computer Engineering
[1,2,3,4,5]Nirma University

Abstract— With the software development at its boom compared to 20 years in the past, software developed in the past may or may not have a well-supported documentation during the software evolution. This may increase the specification gap between the document and the legacy code to make further evolutions and updates. Understanding the legacy code of the underlying decisions made during development is the prime motto, which is very well supported by Reverse Engineering. In this paper, we compare the Transformational Forward engineering, where a stepwise abstraction is obtained with the Transformational Reverse Methodology. While the forward transformation process produces overlap of the decisions, performance is affected. Hence, the use of transformational method of Reverse Engineering which is a backwards Forward Engineering process is suitable. Besides the design recognition obtained is a domain knowledge which can be used in future by the forward engineers.

Key words: Software development, Specification document, Legacy code, forward engineering, Reverse engineering

I. INTRODUCTION

Software Engineering is a process of creating, enhancing and maintaining a well-defined system to meet the specific requirements of the client with the support of tools required by the same. The Software Development Life Cycle (SDLC) is not just the prime feature of software engineering but also required is a round the clock maintenance and support to meet the changes as asked by the client.

But often it happens that the software doesn't have well defined document to support changes made in the software. This increases the task of the developer to understand the code of its purpose and the design decisions made. Hence the adaptability of the software to changes is a question.

Having a semiformal specification can lead to a more varied interpretations for the coders and hence can lead to a limited usability. Therefore a formal description with a rationale can help bridge the communication problem.

II. MOTIVATION

Among all plausible methods defined for plan recognition, Programmer's Apprentice is one of the most advanced and sophisticated, representing plans with the support of control and data flows. It uses the recursive method to match the codes with the templates and maximize the ability and independence. For the ongoing research in reverse engineering 'the big-problems' are not being divided into its sub modules, for which we were inspired to carry out the comparative study between Forward & Reverse Engineering.

III. LITERATURE SURVEY

A. Forward Engineering

This approach is channelized by collecting the informal requirements which are converted into semi-formal requirements in the present scenario of software development. This gives the programmer and developer the option/choice to make underlying decisions not provided in the document.

Hence, hidden in this document and code are those underlying decisions made to make the implementation more efficient. Example would implement a binary search rather than linear search to reduce response time of the system, which is not known to the client.

Besides post development and usage phase, errors and bugs detected in the system will be rectified. But these changes may not be reflected in the program but not in the document. Hence, the program is mismatched with the document which leads to understandability problems. To bridge the gap between specification and the program its required to first reflect the changes in the design and then make hardcore changes in the program to induce flexibility and provide maintainability. Document including the rationale generates relationships between the same. Two fundamental approaches are channelized as: First and the conventional method involve the stepped filter and precede process wherein underlying intermediate decisions and steps which provide the formalized and understandable structure of the software, are added to the document to reduce the gap. However, this conventionality introduces certain drawbacks to the software and the client. It still involves a non-automatic implementation as they still don't have any rationale nor any underlying decisions taken by the programmer. Besides, a serious inefficiency is observed as piles of documents have to be maintained for every change incorporated in the software.

With the previous approach inducing inefficiency, developers use a much better and faster approach referred as the transformational method wherein a more specific system and user requirements are required. Detailed requirements may be considered to contain a domain notation like UMLs, object diagrams or data-flow diagrams. Based on these specifications provided, a transformed program code is generated to change the previous code or generate a new software code which also rationale for each transformations, unlike the previous approach. This helps convert the given specification to a more specific and updated document. It involves three standard steps to carry out the transformations: [1]

Refinement, Optimization and Integrating the filtration & optimization.

Hence, these steps generate an understandable code with a rationale mentioned for each. Transformational design generated, achieve the performance requirements along with new selections made by the programmer but at each level this details are documented.

Various techniques or approaches can be applied by the developer to attain the transformational design:

1) Flow Generation:

Generalization or specialization as required is structured. To achieve higher efficiency a better decision is to use specialization.

2) Algorithm Selection:

Algorithm selection based on the different implementations possible can be done to be able to properly document the implementations.

3) Interleaving:

Various different modules or functionalities can be integrated into same code or section to achieve higher performance.

4) Classification into Subsystems:

Hierarchy of subsystems can be achieved to organize the different modules/designs as per usage in the code.

5) Representation Changes:

To attain the required specification in a particular language it might be possible to change the representations. As a structure of arrays or array of structures [1], representing the parsed data as an indexed single dimensional array or a parse tree for better understandability and usage.

6) Resource Sharing:

To avail the same resource to various functionalities and modules.

7) Caching:

In order to preserve the intermediate results to be used later in the program, these results are cached.

Besides the transformations can be optimized using various techniques such as Elimination of common sub-expression, Dead Code Elimination, Frequency Reduction Strength Reduction Compile Time Evaluation [2]

Irrespective of the implementation, manual or semi-automated tools understandability is affected as an overlap of the decisions occurs. However, if the code implementation is carried out considering the performance motive, the recorded decisions provide a faster way to develop a new system and modify the existing systems using the CASE tools.

B. Reverse Engineering

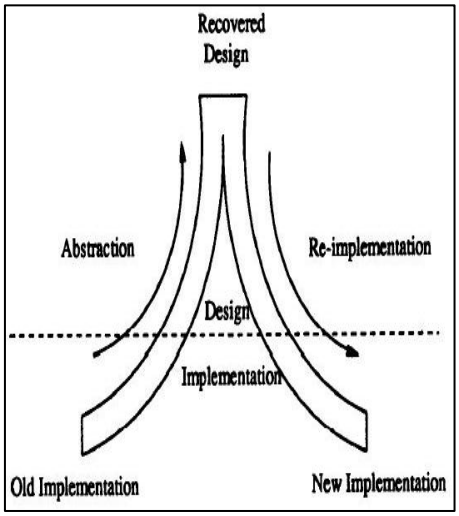

Fig. 1 Flow from old to new implementation [3]

Executing the forward engineering to all systems is not feasible as providing a rationale and formal specific document for each developed software is next to impossible. Over time each software demands an evolution to meet the new client requirements. Removing the bugs and faults, enhancement demands incorporating the different approach than the conventional Forward Engineering.

In the most recent and currently implemented method, recognizing the plans is a process to understand the code fragments. This is a bottom-up design to analyses the code as matching is done of the code fragments with the defined templates to recognize the concepts by the reverse engineer. However this is carried out using certain procedural steps: [3]

1) Software Data Collection:

Involves aggregating the specifications document, source code, all entities relating the software.

2) Information Interpretation:

All collected information in the previous step is analyzed for further implementation.

3) Structure Identification:

Based on the information analyzed, structures of code are created as a chart to represent the calls and returns to map them properly.

4) Functionality Identification:

Structural points in the chart provide the details regarding the code executed in each structure. Appropriate higher level language can be used to describe the functionality extracted.

5) Data-Flow Identification:

Transformations among the data and its related processing and execution are identified and it's recorded using the data-flow diagrams.

6) Control-Flow Identification:

Control structures affecting the overall operation of the system are identified and represented using appropriate flow diagrams.

7) Review Recovered Plan:

Check the correctness and the pattern extracted with the available plans for consistency and fill in the holes by checking the representations.

8) Document Specification:

Recording every step of reverse engineering to cache the pattern extracted, even the rationale derived for each.

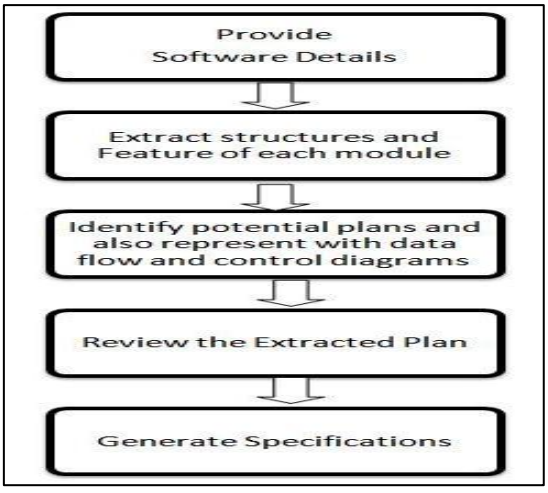

Fig. 2 Reverse Engineering Execution Flow-Diagram

However this induces a drawback as all necessary and required plans need to be known priory. Besides different implementations of the recognized plans is possible which can create a discrepancy. Example a pass by value or

pass by reference in case of function arguments. To identify the correct underlying decision all plans need to be known.

Problems identified can be resolved mainly using alternative approaches:

First method is an extension to the original method. It requires the reverse engineer to add more plans, patterns, templates and correlating rules to detect with more better semantic and syntactical specifications. This increases the probability to recognize the correct plan and make appropriate design decisions. Still, rationale is also required to properly trace back the code with the extracted pattern. Appropriate valid reasoning and transformations are required for the code to be matched to the recorded data set.

Compared to the convention, an approach is also defined for the same. For each set of code, small transformations in reverse order are carried out compared to Forward Engineering, and suitable abstraction is identified. Besides it may happen that the plan abstracted may not exist in the current libraries and hence new domain dataset is obtained. This dataset can further be used by reverse engineers to maintain and modify the existing software, by the forward engineers for develop new systems.

Also, it's not so that plan recognition serves no advantage, it provides a basis for certain transformations too. While performing a match of a natural no from a string value property of hashing is recognized, which helps indicates that a hash table type transformation is required.

Besides from both the reverse engineering processes certain levels of abstraction are obtained. Purposes served are

- maintenance purpose
- evolution purpose
- reengineering purpose

Levels achieved for abstraction are:

- Application: Application concepts, business rule, policies
- Function: Logical and functional specification, non-functional requirement
- Structure: Data and control flow, dependency graph Structure and subsystem charts Architectures
- Implementation : Symbol tables, source text

IV. Conclusion

After analyzing four methods of software development and modification, it is concluded that irrespective of the type of checking performed, we require domain knowledge in each to extract the system design and plan. But Reverse Engineering provides an external support not just to identify the bugs and rectify them but also help generate new domain datasets for future use by the Forward engineers.

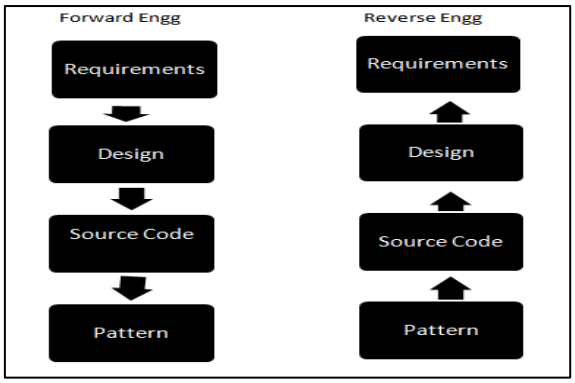

Fig. 3 Forward versus Reverse Engineering Flow

References

[1] D. M. Dhamdhere, Systems Programming, New Delhi: McGraw Hill Education (India) Private Limited, Fifth Reprint 2013.

[2] M. M. Ira D. Baxter, "Reverse Engineering is Reverse Forward Engineering," in Fourth Working Conference on Reverse Engineering, Amsterdam,Netherlands, 1997.

[3] E. J. Byrne, "Software Reverse Engineering: A Case Study," SOFTWARE—PRACTICE AND EXPERIENCE, vol. 21(12), no. December(1991), pp. 1349-1364.

