International Journal of Engineering & Technology, 7 (4.19) (2018) 1036-1040

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

A Novel Approach for Building Adaptive Components using Top-Down Analysis

Sampath Korra[1]*,D.Vasumathi[2],A.Vinaybabu[3]

[1]*Research Scholar, Dept of CSE, JNTU College of Engineering, JNT University, Kakinada,India*
[2]*Professor, Dept of CSE, JNTUH College of Engineering, JNT University, Hyderabad, India*
[3]*Retd Professor, Dept of CSE, JNTUH College of Engineering, JNT University, Hyderabad, India*
**Corresponding author E-mail: sampath_korra@yahoo.co.in*

Abstract

Developing reusable components are one of the main objectives of component-based software engineering. They play a crucial role in the field of application development and support. CBSE use certain architectural patterns and infrastructures of standard software to increase overall product quality. CBSE apply two parallel engineering activities, domain engineering and component-based development (CBD). Domain analysis explores the application domain with the intent of finding functional, behavioural, and data components that are candidates for reuse and places them in the reuse repository. Strategies for developing adaptive reusable components using top-down domain analysis leads to good quality in the component. Domain analysis promotes strategies and models that have been developed for their specific areas. Therefore, these models are suitable for their own domain, but may not be entirely suitable for domain analysis of other domains. So, developing the reusable components using the top down domain analyses existing components. This paper describes how to build a domain to use top-down analysis of reusable software components.

Keywords: *component; reuse; adaptive; domain analysis; software; top-down; bottom-up;*

1. Introduction

In software engineering, product line analysis or domain analysis is the process of analyzing related software systems in the field to find common and variable parts. However, this domain analysis, which is purely bottom-up, gives a view of the domain that also limits what has been developed in the past. What we also need has to be developed in the future to determine which components provide reuse. In order to obtain the views of the future, we can analyze the business model as part of the name resolution process. In order to promote large-scale reuse, we should consider how to re-use the method of a software component. Although recycling is the most common type, and sometimes do not turn out to be a very difficult part, we should try to reuse software component architecture, specification, and design of a higher level, because they are less dependent on the future change [1]. In addition, the design of the reusable component using top-down analysis should always be able to solve technical problems with it [2].

Software developers use "plug and play" approach, in order to promote development and integration of reusable software components. Software architects and designers will create a top-down layered architecture and interface that will use for the development of reusable components [3]. It will result in software solution products, adaptability, and scalability. This article discusses the use of the field of component based software engineering for developing reusable software components using top-down domain analysis [4].

By reusing software development projects, we can build the solution of new products and processes. Over the reuse, we can build the solution development of software, the formation of recent products and advanced technologies. On the product side, we have to make sure that the delivery forms can support reuse in the process. Our approach requires the improvement and application of the products. In the process, we must able to handle both abstract problem areas and build reusable solutions. The system may be developed within the solution area and then used. The processes and products may be started for successful reuse. In the first process, to adapt existing systems to meet the new needs generally used. In the second, new approach, the process of determining ownership, supporting, and customizing parameters is often a unique requirement. The third method to abstract the underlying engineering approach and find use of a common software-based software system that is adaptive [5].

What is Adaptability?

Adaptability is the properties of software that refers to how much work is needed to change the running program or the ability of software to adapt and changes in the environment. Software with certain adaptability is called adaptive software. Software programming techniques suitable for supporting adaptable software are referred to as Adaptive Programming Technology (APT). Adaptive software systems have a great advantage over traditional software systems. Traditional software systems adapt to external changes: the system is difficult to adapt to external changes.

Whenever external conditions change, they need to be recoded and tested. They may even affect the detailed design, overall design, and requirements analysis. Changes in the system's life cycle may involve higher levels, requiring the involvement of operators, programmers, designers, and even analysts.

Adaptive software systems adapt to external changes. External changes only affect the maintenance cycle and they do not affect the other cycles of the life cycle. Therefore, maintenance costs can be greatly reduced.

With the continued growth in the field of software engineering, researchers and practitioners to bring new perspectives and insights into the field of software reuse [6].
Here the essential focus on application software products that have been particularly in the field of model and software architecture analysis. Accepted domain analysis in software reuse is a very important first step in the decision [7].
Top-down domain approach

- Hierarchically organized component architecture,
- All the necessary components are arranged in ascending order.

Reusable software components, typically the designer will provide software and hardware architecture system design in the reusable framework or model.

2. Problem statement

Component-based development, also known as the development of "reuse", involves application building. By reusing existing components new applications are developed. If necessary, anybody can develop new components that can even be obtained from a third party. An important issue in CBSE is that even if the requirements are not fully present, it is more important to reuse existing components rather than develop new components. It reduces costs and time to market but requires a search mechanism to access all available components. Therefore, in order to be effective, CBSE must provide a way to find, connect and adjust the existing components. It must also support the addition of developed components during this period.
The component combination integrates the component (whether qualified, adapted or designed) into the work system. Integration of components is solved by establishing an infrastructure that binds components of the operating system. This infrastructure is usually a specialized component library. It provides components and specific service correlative models that facilitate to coordinate components with each other and common tasks perform. Many of the functions of the software system come from the interaction of their components.
The functionality created from the component architecture can be understood as a careful component. At present, there are two standard mechanisms for the component. Components interconnected languages and standard interfaces.
The interconnection between programming languages:
Interworking language is a special purpose description language that describes how a component is connected. At least, the interconnected language expresses the general structure of the system and an advanced linkage language that describes the functions of the system likewise the communication protocol and connector properties. Interworking languages help to abstractly specify components in an isolated modeling language. While this enhances the clarity of the model, it increases the exploration of updating the system specification when the component definition changes or the combination of components is different. The most primary category of interconnected languages is Module Interconnection Language (MIL).
Component Adaptation Techniques

- **White-box wrapping** - Integration conflicts were removed when code-level changes are made to the code.
- **Grey-box wrapping** -Used when the component library contains a component extension language or API that allows to remove or mask conflicts
- **Black-box wrapping** – Require the introduction of pre-treatment and post-processing in the component interface, so that conflicts can be removed or covered

Standardized interfaces:
The formation of standard interface methods and language interconnection distinction. In an interconnected language, the connection between components is explicitly indicated in a higher level of language. Standardization allows a component to absolute appeal to a service defined in a canonical interface and then consists of any component that implements the interface. Develop a standardized interface for each region. Finally, the development component implements a specific area of the interface. It is said that these areas are parameterized to clearly identify components of other standardized interface areas. A component combination is a process that replaces a domain parameter with a specific component of the field.
Adaptor specification:
In this paper, we present a simple notation to express the specification of the adapter with intent feature interoperability of one or more components. The adapter specification consists of a series of correlation table actions and parameters of two components. The distinctions part of the entry is that it generates a high level, the partial specification of the adapter. The meaning of the adapter specification can be formalised in a set of properties and those properties are discussed in the below algorithm.
Pragmatic programmer:
Pragmatic developers have a practical and unknowable component approach because he does not have any particular brand, type of component and is, therefore, superior with other components in the general sense manner. The software is nothing but software, whether it is a good or bad development, in the professional development of closed business software industry has a lot of bad software examples, because a community has driven open source project with the hacker world. An approach may not exclude the option, but rather than selecting the most appropriate component for any particular task or case.

3. Proposed work

1. How to improve the adaptive method for a software system

Enhancing the adaptability of the software system means that the software system can adapt to changes in business performance and changes in an external environment. Improving the adaptability of the software system can be done from the following points: [22].
Software interoperability improvement:
Software design relies on business requirements that can meet the needs of time and have certain versatility. In other words, various changes that may be encountered during the system design phase should be considered. When external circumstances change, the system should be better able to adapt and maintain a stable operation;
Improvement of software self-description:
The system has a certain ability of self-description. It can describe the change of external conditions as much as possible through the form of parameters. When the conditions change, only the corresponding parameters need to be adjusted, but no large-scale software modification is required;
Software as a tool to improve:
Some basic, unchanged or often referred to as modules, they can be considered as an instrument only in the design phase, which improves system flexibility and call efficiency;
Software modularization improvements:
Create a reasonable, efficient and independent module with functions to call different functions for the modules. These modules make every effort to use coupled data to minimize the complexity of the program;

2. Domain analysis method

The domain analysis method is to perform a software system for collecting information that has a common set of features and data. In addition, domain Analysis will determine precisely in those areas and software products Domain is a good resource for reuse [8]. Overview formulated in a top-down approach of the system but does not specify any detail about the component. Each subcomponent is then further purified in more detail.

Three primary activities are framed for representing the process:
Scope: It will determine the domain analysis
Domain modeling: It will meet the software domain provided description requirements.
Architectural modeling: To achieve a solution to the problem domain creates software architecture.
Domain Engineering is subdivided into 2 parts
Top-down Domain Analysis
Bottom-up Domain Analysis

2.1. Top-down domain analysis

A top-down approach for developing a reusable component scheme is known as an enumerative classification scheme of domain analysis [9] [10]. Domain analysis and research, through the system top down, from the future business plans of top domain model analysis. To target for developing reusable components using top-down analysis model, it will determine how many components are before the next component of the system to establish a common architecture [11] [12].
Top-down analysis of the domain is the proposed system planned for analyzing the domain business model. Its purpose is to decide and build reusable components. Those can be reused for finding common system identification requirements and regular system behavior/function**.** If the regular software architecture has enough organized, then this is a good sign, re-use should be achieved with a new approach.
Experience in the analysis of applications shows that the domain definition and scope of analysis is not easy in many cases. The range of areas or product lines often depends more on the inside analysis [13].
Commonalities can find for duplicate system requirements, functional and regular software architecture of the system behavior on a daily basis to determine. If the ratio is high enough, then this is a good sign in the field, if we use these strategies. So always top-down domain analysis has high priority in the design of reusable components [14].
Selection of component within the boundary region can significantly affect the success and domain analysis but is not uniform throughout the method that can minimally resolve in some cases [15] [16].
There is no strategic planning and business models to choose from, often are not performed in the bottom-up analysis. But it also requires a corresponding top-down [17]. But frame a strategy for developing reusable components using a top-down approach the product development time will be reduced [18]. Some of the key points to be observed while using top-down domain analysis are shown in below table.

Table 1: Top-down domain analysis

Required specifications in order to improve quality
Accelerate reuse systems throughout the software life cycle
Facilitate future reuse of the product life cycle
Development life cycle assists in the early stages
Better use of existing components

2.2. Bottom-up domain analysis

A bottom-up domain analysis scheme which is also known as a faceted classification scheme. Some studies, a typical example of a conventional bottom-up systems domain method, the components in these systems for the identifying component. By examining how the use of common components present in the system. This sample, we can learn how to create a reusable component [19].
The method is used to determine the recurrence of the whole system that is looking for the same or similar names, the same or similar input and output that are similar to the same common component flow graphs, the same or similar data structure [20].

2.3. Why we are using top-down analysis?

Establish and maintain a database of reusable formal semantics is described and the use of semantic information, we can focus on the components do. This method is part of software development and software reuse. Reusable results are an integral part of the research domain analysis that can significantly benefit from the work of other areas.
Classification is nothing more than a combination of similar components, that is, all members of the group share a feature that is part of other groups [21]. The Java component, its function is classified. Evaluating each function may have the same function for similar parts.
Several domain name resolution of the problems in other disciplines. Information is a reusable knowledge and extensive research and technical services. Various components are classified to meet the overall requirements of the frame to reuse [22].
Here developed a heuristic approximate matching reuse method to identify a subcomponent of the relevant library specification which is a more detailed assessment that is shown in figure 1. Indexing of component is based on the use of the classification function on the component and allows for efficient reuse. Classification is achieved in a top-down manner; the features that may be described by the defined automation components[30].
Experts to define the control of the classification system, rather than a person in the field [23][24].To ensure the most likely component features to match the standard feature set similar to reuse. Coincides with the full-size scale can use automated logic provides components accurately assess applications [25].
For software reuse, it is an essential component which can be composed without having to know each other. It makes the component composition[28], without changing the member (Dynamic Control). For example, the functions call functional classification, calling a function, rather than assembling, has been modified in the program text called function [26].
System analysis also provides a complete technology and proven methods to help us understand the domain name resolution process. Domain analysis should try to reuse existing research from other disciplines [27].
Object-oriented programming provides greater flexibility through dynamic binding. It is very easy to re-use the components, where each component in the environment may be generated in response to the other events that are not aware of the receiver member to create a new event [25] [29].

Algorithm:
(1) s1 and s2 subset of S
(2) Rb(Sn c) > Rb(Sn1 c)
(3) Rb->Reusable component. Reusing a more expensive component is more beneficial than reusing a cheaper component.
The cheaper component is Rb (Scm)
(4) Rb (Scv) > Rb (Scm) > Rb (S)
(5) Rb (Scv) ->Adaptive component reuse
(6) Rb (Sn, c) ->Reusable output.
(7) Rb (c) ->Classifications of components
S1->Selective Component 1
S2->Selective Component 2
S->Super set of S1 and S2
Rb-> Reusable components
Scv-> Expensive component
Scm-> Cheaper component
C->Classification
Rb(c)->Reusability using Adaptive Classification.

In the above algorithm, we are discussing how to classify the components using a top-down approach. Before this approach the components are unordered. That is shown in figure 1. The components are of different technologies ex-java, c++, c language etc.

Classification is nothing more than a combination of similar components, that is, all members of the group share a feature that is part of other groups [21]. The Java component, its function is classified. Evaluating each function may have the same function for similar logic so we classified using subset theory.
If two components need to be more structurally integrated, the two components can be grouped into one package component. For example, s1 and s2 are selective components and now those are a subset of S. The top-down analysis is defined for this purpose. However, the packaged component needs to delegate the request to the contained component to meet the system's requirements. The traditional approach is to define a large set of small methods on the encapsulated component and forward the message to the correct encapsulated component. Obviously, this method will lead to consider implementation costs for software engineers. In addition, the reusability of the solution is very limited.
Rb->reusable component. Reusing a more expensive component is more beneficial than reusing a cheaper component. The cheaper component is Rb(Scm)
The root cause of these problems is that the polymerization is opaque. When top-down analysis can be used as a compositing technique, the solution is to define a superimposed entity that allows combining two or more components without the above drawbacks.
Informally, the above specification states that the overlay unit openly transmits all messages in a nested component that defines a similar method. However Name conflicts or other scenarios, software developers can configure explicit interface element mappings. Explicit mapping functions override implicit definitions that are shown in figure2.

4. Results and Discussion

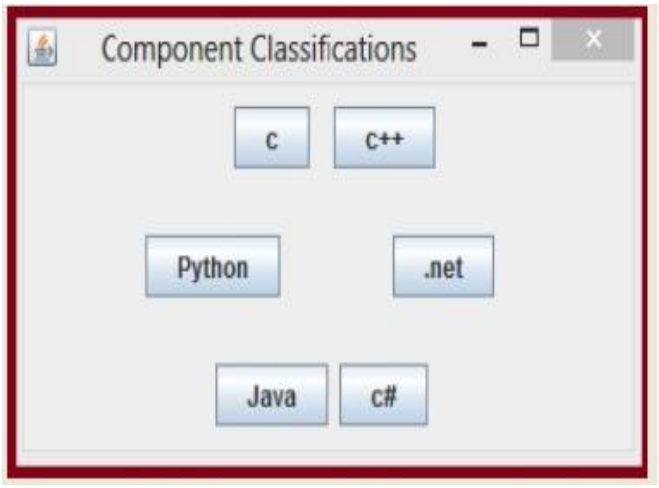

Fig. 1: Component classifications

In the above example components are of different technologies like c,c++, java,python,.net etc. And all are in unordered list. If we classify the components based on the selective components then they are going to form as a subset.
Java and C++ are the component state and behavior available to the reuse component. According to the language model, all internal aspects or only some aspects can be used to reuse components. For example, in python, all methods and instance variables defined in a superclass are available for subclasses, and in C++ it depends on methods and instance variables that use private and protected keywords. Can be used for subclasses. Inheritance provides an important advantage of code that still exists in one place. However, one of the major drawbacks of inheritance is that when software engineers rewrite superclass methods and use super class-defined behaviors to define new behaviors, they must often have a detailed understanding of the super class's internal functions.

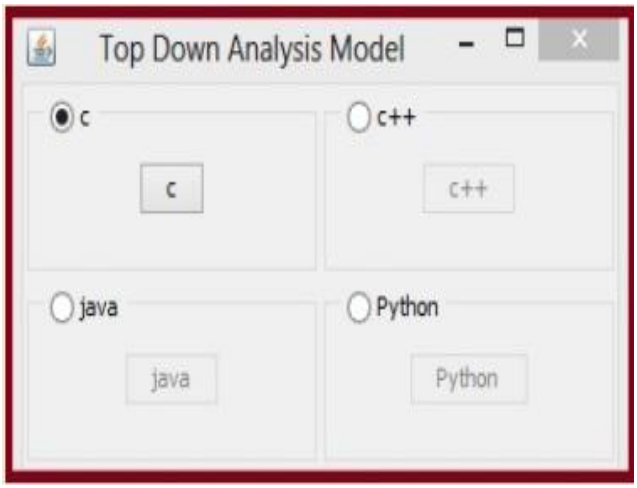

Fig. 2: Component classifications after the top-down analysis model

If two components need to be more structurally integrated, the two components can be grouped into one package component. Here S is superclass that is going to classify the components s1, and s2 and that is shown in above figure 2.
In below Table 2, an overview of the conventional adaptation techniques is presented that indicates how well each technique fulfills the specified requirements. From the table, one can see that some problems are dealt with well wrapping but not so well by the white-box techniques, i.e. copy-paste and inheritance, and vice versa,

Table 2: Conventional adaptation techniques versus the identified problems and requirements

Requirement	Copy-Paste	Inheritance	Wrapping
C	Yes	No	No
C++	No	Yes	Yes
Java	No	Yes	Yes
Python	Yes	No	No
.Net	No	No	No

Copy-paste techniques and inheritance are transparent because reuse and adaptation behaviors are merged into a single entity. However, according to other requirements, the white box adaptation technology score is not so good. The package is opaque because it encapsulates the adapted components. By definition, the package is a black box, and the package is combinable because the package components can be repackaged by another packager to accommodate different aspects of the original assembly. Traditional technologies do not support configurability and reusability well because there is no distinction between generic behavior and component-specific behavior. For this reason, it is impossible to separate common aspects and apply them to different components[18][23].

5. Conclusion

Top-down analysis is usually possessed economical issues, but not the way to take the re-use of an opportunist. To focus on short-term gains, lower costs to re-use and only those who direct the application specific system (program) is being developed. There are two main strategies to build reusable components in the software engineering field which are top down and bottom up analysis.

For adaptive software, reuse needs successful component identification, classification, storage, retrieval, and management, for the understanding of the background of all exported to such information. By developing a top-down model context, for the domain basis, maximize they assemble, wherein the reusable understand and able to infer how to reuse the value can be carried from one context to the other side. This new model-based "requirements engineering" field is accurate. Top-down domain analysis is concerned with the processing of information overload from the software industry, they do not realize that the only relevant requirements. To achieve this goal, we recommend that top-down domain analysis strategies. To achieve the application of different software architectures, code and using reusable components for reuse of the building blocks of portable applications.

References

[1] R.G. Lanergan and C.A. Grasso, "Software Engineering with Reusable Designs and Code," IEEE Transactions on Software Engineering, vol. SE-10, no. 5, September 1984, pp. 498-501

[2] William B. Frakes and Kyo Kang, "Software Reuse Research: Status and Future", 2005

[3] J.M. Boyle and M.N. Muralidharan, "Program Reusability through Program Transformation," IEEE Transactions on Software Engineering, vol. SE-10, no. 5, September 1984, pp. 574-588.

[4] C.A.R. Hoare, "Hints on Programming Language Design," In Programming Languages: A Grand Tour, E. Horowitz, ed., Computer Science Press, Rockville, MD, pp. 31-40, 1983,

[5] Korra, Sampath, A. Vinaya Babu, and S. Viswanadha Raju. "The adaptive approach to software reuse." Contemporary Computing and Informatics (IC3I), 2014 International Conference on. IEEE, 2014.

[6] Prieto-Díaz, R., Domain analysis for reusability, Proceedings of COMPSAC'87, 23–29, (1987).

[7] W.A. Hegazy, The Requirements of Testing a Class of Reusable Software Modules, Ph.D. dissertation, Department of Computer and Information Science, The Ohio State University, Columbus, OH, June 1989.

[8] H.D. Mills, M. Dyer, and R.C. Linger, "Cleanroom Software Engineering," IEEE Software, vol. 4, no. 5, September 1987, pp. 19-25.

[9] Piho, G, Tepandi, J. and Roost, M., "Domain analysis with archetype pattern based Zachman framework for enterprise architecture." [ed.] A K Mahmood, et al. Kuala Lumpur, Malaysia, 15th-17th June 2010 : IEEE, 2010. Proceedings The 4th International Symposium on Information Technology 2010. Vols. 3 - Knowledge Society and System Development and Application, pp. 1351-1356.

[10] J. Krone, The Role of Verification in Software Reusability, Ph.D. dissertation, Department of Computer and Information Science, The Ohio State University, Columbus, OH, August 1988.

[11] D.L. Parnas, "A Technique for Software Module Specification with Examples," Communications of the ACM, vol. 15, no. 5, May 1972, pp. 330-336

[12] Daniel Gross and Eric Yu. "From Non-Functional Requirements to Design through Patterns". Requirements Engineering, 2011, 6:18-36.

[13] O. Kath, R. Schreiner, and J. Favaro. Safety, Security, and Software Reuse: A Model-Based Approach. In Proceedings of the 4th International Workshop on Software Reuse and Safety, RESAFE '09, Washington, D.C., US, September 2009.

[14] B.H. Liskov and S.N. Zilles, "Specification Techniques for Data Abstractions," IEEE Transactions on Software Engineering, vol. SE-1, no. 1, March 1975, pp. 7-19.

[15] Software Reusability: Concepts and Models, T.J. Biggerstaff and A.J. Perlis, eds., ACM Press, New York, vol. 1, 1989.

[16] Hafedh Mili, Fatma Mili, and Ali Mili "Reusing Software: Issues and Research Directions" IEEE Transactions on software engineering, VOL 21, NO. 6, JUNE 1995

[17] Schmidt, D. C., Why Software Reuse has Failed and How to Make it Work for You [Online], Available: http://www.flashline.com/content/ DCSchmidt/lesson_1.jsp, [Accessed: 18 August 2002].

[18] Douglas Eugene Harms "The Influence of Software Reuse on Programming Language Design" The Ohio State University 1990.

[19] Sullivan,K.J.;Knight,J.C.; "Experience assessing an architectural approach to large-scale, systematic reuse," in Proc. 18th Int'l Conf. Software Engineering, Berlin, Mar. 1996, pp. 220–229.

[20] M. Pat Schuler, "Increasing productivity through Total Reuse Management (TRM)," Proceedings of Technology2001: The Second National Technology Transfer Conference and Exposition, Volume 2, Washington DC, December 1991, pp. 294-300.

[21] http://www.encyclopedia.com/science-and-technology/biology-and-genetics/biology-general/taxonomy

[22] D. Lucredio, A. F. d. Prado, and E. S. d. Almeida, "A Survey on Software Components Search and Retrieval," in Proceedings of the 30th EUROMICRO Conference, 2004, pp. 152-159.

[23] M. Aoyoma, "New Age of Software Development: How Component-Based Software Engineering Changes the Way of Software Development?," in Proceedings of the 1998 International Workshop on CBSE.

[24] Constance Palmer, "A CAMP update," AIAA-89-3144, Proceedings of Computers in Aerospace 7, Monterey, Oct. 3-5, 1989

[25] Brian W. Holmgren, "Software reusability: A study of why software reuse has not developed into a viable practice in the Department of Defense," Masters Thesis, Air Force Institute of Technology, AFIT/GSM/LSY/90S-16, September 1990.

[26] yvind Hauge. Adoption of Open Source Software in Software Intensive Industry. 2010.

[27] Carl Gutwin, Reagan Penner, and Kevin Schneider. Group awareness in distributed software development. Proceedings of the 2004 ACM conference on Computer supported cooperative work - CSCW '04, page 72, 2004.

[28] J. Schneider, "Components, Scripts, and Glue: A conceptual framework for software composition,"Ph.D. Thesis, Ins. of Computer Science and Applied Mathematics, University of Bern, 1999

[29] K. Venugopal Reddy *and* Sampath Korra "Object-Oriented Analysis and Design Using UML" BS Publications, 2018.

[30] Classification-based Mining of Reusable Components on Software Product Lines, Maximiliano Arias; Alan DeRenzis; Agustina Buccella; Andres Flores; Alejandra Cechich, IEEE Latin America Transactions Vol.14, Issue:2, Feb.2016.