ABSTRACT
In general, decomposition methods can facilitate the process of solving sophisticated and heterogeneous problems in the area of software development and engineering. These approaches are assisting to decompose problems based on different disciplines, characteristics and functionalities that is results into increasing the computational efficiency (e.g. parallel processing/computing) and accelerate the software changing process, software modifications and error tracking. Essentially, these approaches contribute to the degree of modularity to decompose a complex problem into different sub-problems and to focus on local objectives. There are different approaches that are used to decompose a problem into the smaller ones by keeping/improving the accuracy and efficiency in software engineering. Three major decomposition approaches that have been used in software engineering are decomposition based on aspects, objects and views. Each of these has their own characteristics and limitation. This research paper aims to address some of these important issues regarding to the decomposition approaches that are used in the software engineering discipline. In the next step, a new decomposition paradigm called multi-level decomposition will be introduced and a proper vision in terms of the key characteristics, differences and analogies between this decomposition method and others will be addressed. At the end, an example problem will be decomposed based upon the presented approach to show the potential capability of the approach.

Categories and Subject Descriptors
D.2.0 [Software Engineering]: General; D.2.9 [Software Engineering]: Management

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

1. INTRODUCTION
In general, one of the first steps to solve a complicated problem is to decompose the complex problem into smaller problems such that they are easier to grasp and comprehend. The main motivation behind the decomposition is to facilitate the modeling and understanding of the problem that are involved with many items/parameters as well as to solve multiple goals that may contradict with each other. Decision making methodologies such as mathematical optimization can utilize the problem solving process. In general, a software intensive system is composed of many different features, functionalities and characteristics that involve people from different disciplines with different goals and objectives. Decomposition can be used in different aspects or stages of software engineering from life cycle design and process modeling to actual development and implementation. In each aspect the purpose of decomposition is to create an understandable and efficient framework to solve problems. Moreover, it has also specific benefits. For instance, decomposition in the process model can help management to divide the project between different developers in order to obtain an efficient timely project delivery based on the capability of the developers. Management can also consider different disciplines related to the software system and define the joint interaction between developer’s team. Decomposition can also help in review process in terms of selection of review meeting representative as well as the material required to be presented in the meeting. In the software design and development, decomposition is considered as one of the important factors. The most conventional and classical decomposition strategy in mathematical/computer science is functional decomposition [1,2]. This is basically means that a complex function that has too many variables and parameters can
be decomposed into some sub-functions known as constituents (parts) which are inter-related but have less variables compare to the original problem. Therefore, the original function can be reconstructed from those constituents. This concept is explicitly used in software development community in order to represent a complicated function using simpler or basic function. This decomposition helps to manage the complexity and helps to achieve the modularity to some degree. In the early decades of computer programming, this methodology is called "art of subroutining," by some pioneers [2]. The concept of functional decomposition is extended in multiple diagrammatic techniques such as functional flow diagrams. By the discovery of object oriented programming [3], programmers are facilitated with a dramatic improvement of modularizing tool provided by this framework. In this approach, program is decomposed into different objects consisting of data field and methods along with their interactions. The fundamental concepts and feature in this paradigm are class, instance, method, message passing, inheritance, abstraction, encapsulation, polymorphism and decoupling. Aspect-oriented decomposition which emerged from object-oriented approach is another method to decompose the problem. The main characteristic of the aspect-oriented software development is to modularize a software system in order to isolate the secondary or supporting functions from the main program business logic. In general, the traditional software development is focusing on the modularization of the software systems into different functions and the secondary functions will be addressed based on the developers judgment wherever is applicable. In aspect orientated decomposition particular attention is paid to the functionalities or concerns that cannot be clearly decomposed and may result into scattering and/or tangling of the problem. These concerns are known as cross-cutting concerns. Cross-cutting concerns are part of the program or process that rely or affect on multiple parts of the system. The aim of aspect-oriented decomposition is to encapsulate cross-cutting concerns into aspects to retain modularity. In programming context, aspect-oriented programming provides a clean isolation and reuse of code dealing with the cross-cutting concerns. Aspect-oriented software development can be implemented through Subject-Oriented Programming, Feature Oriented Programming, Adaptive Programming, and Aspect-Oriented Programming. [5-7]

As previously discussed, the decomposition paradigm can be implemented in many fields of software engineering and computer science in general. From architectural point of view the software architecture can be represented from different viewpoints. From management point of view the main and foremost important issue is to manage and deliver the project within a timely and cost effective manner. A user may look at the software system from different point of view. Developers are looking at the efficient way of preparing a code while considering the constraint related to the reusability and modularity or any other goal which may be imposed by the management. View models are basically an umbrella to consider the project stakeholders’ view, goal and concerns. Therefore, view models are used just as a way to describe the whole system decomposition according to specific viewpoints.

As described above, decomposition techniques are conventionally used in the area of software engineering, computer science and programming. In this paper, a new decomposition paradigm known as target cascading is introduced. This methodology is more suitable for the problems that illustrate some level of hierarchy. The functional dependency table is utilized the decomposition and partitioning process. Different features of this decomposition methodology will be explained and an example problem related to requirement engineering and software design will be discussed. In order to design a quality based dynamic process model for a vehicle control software system, the target cascading approach is used. A discussion on the coordination strategy, integration and internal interaction of the decomposed system will be provided.

2. DECOMPOSITION

In order to have an effective decomposition, we need to know when to decompose the problem. The second step is to identify the components in which address specific functionality and then have proper model to represent each component. Component identification can be achieved by separating the complex software system into different disciplines, characteristics or different functionality. This approach facilitates the quality implementation of proper model that can be more efficiently delegated to the experts in each fields/disciplines. In order to decompose a system, one conventional way is to tackle the problem with divide and conquer. The problem is partitioned to some subsystems that are roughly have the same levels of details, each problem can be solved independently and the solution of each problem is equivalent to the solution of the original problem called all-at-once problem.

The main advantage of the decomposition is that each subsystem is focused on specific needs and can be solved separately that results into better parallelization and facilitate the software maintenance and modifications.

A. Multi-level Decomposition

Complex software systems, such as those encountered in aerospace and automotive industries, can typically be considered as a hierarchy of individual elements. This hierarchy is reflected in the software systems that are used to analyze and control the physical characteristics of the system as a whole.

Many complex software systems can be considered a single component when viewed from far and a collection of components when closely observed. An aircraft or automobile control software system can be seen as a single element if it is being considered as an all-at-once problem. Zooming in, the vehicle control system consists of different system controls such as engine control software system. Each element is built from smaller components, e.g. the engine system consists of a spark control system, adaptive injection control system, etc. Each of these sub-level control software is built from some subsystems as a single component when viewed as an element, however when closely observing the control system, it is built up of several components. In the subsystem level, analysis models have been developed to describe a control system that focuses on single task. Explicit connections and links between the multiple levels or components are crucial to develop an efficient multi-level sketch of a software system’s behavior.

The analyses of elements at different levels are typically conducted independently if the control system can be decoupled to some degree. Therefore, the output from one analysis forms the input to the neighboring element. When elements are part of a hierarchy and the higher elements in the hierarchy pass output to elements lower in the hierarchy, approaches are called hierarchical top-down approaches. The analysis covering general
characteristics of the structure is conducted and results are passed into the lower levels in the hierarchy.

Likewise, a hierarchic bottom-up approach is used that starts at the smallest scale of observation and passes the results to higher levels in the hierarchy. In cases where independent analysis of phenomena at different levels is not possible, coupled approaches are required. The designer may decompose the problem over various levels of a hierarchy. Elements of the hierarchy are formulated as individual modules that are coupled to individual modules elsewhere in the hierarchy. The designer can focus on changing smaller and more manageable parts of the design without neglecting the combined performance of the entire hierarchy. Such an approach is successful, if many variables and/or constraint functions and/or objectives can be assigned to individual hierarchical elements [12]. Typically, the system will be initiated at the top or the bottom of the hierarchy and level by level the individual functionalities are conducted. Cycling over the levels is necessary if the results of a single element largely affect the optimal functionality of neighboring elements.

Lastly, a hierarchy can be created because the model is too large to solve or it is more convenient to solve it in parts. Such techniques search for weak links between the functionalities or characteristics that allow for decomposition. A hierarchy might not seem present at first, but due to analyzing the properties of the lower level problem, blocks of local analysis equations or functions can be distinguished such that decomposition can take place. In the process of identification of elements of hierarchy, it is important that to know whether they are naturally present or artificially identified. It is also important to know if those problems can be considered locally. These responses and/or design problems might weakly or strongly dependent on responses elsewhere, however their main computational effort is local. The influences of computed local responses and/or design problems are communicated to neighboring elements through boundary. Hence, the actual responses resulted from each elements are mapped onto the neighboring domain and vice versa.

B. Functional Dependency Table (FDT)

In this section, a guideline for the identification of a hierarchical problem is illustrated using so called functional dependence table. This approach is originated from multi-level decomposition problem solving [13]. In this section, the overall methodology of decomposition technique using functional dependence table (FDT) is conceptually described and then this approach will be adapted to be used in dynamic process design in automotive software system. The functional dependence table (FDT) is a matrix representation of the structure of the objective and constraint functions in optimization. As shown in Fig 1, rows are labeled with function names, and columns with variables names, element (i, j) of the FDT is non-zero when function i depends on variable j, otherwise the element is zero. Shading is often used to indicate non-zero elements of the matrix [11]. The linking variable vector $s$ contains three components in Fig 1.

To illustrate the use of the FDT, consider that a function can be decomposed into four functions that are inter-related. First function called $f_0$ that is dependent on $s_1$ and $x_0$ and $f_1$ is dependent on $s_1,s_2$ and $x_1$, $f_2$ is dependent on $s_2$, $s_3$, $x_3$ and $f_3$ is dependent on $s_3$ and $x_3$ the resulting FDT representing this problem is shown in Fig 1. As it is illustrated, this problem can be represented by flowchart shown in Fig 2. This figure shows that the elements are related to each other through shared variable $s$.

This approach will be used to design a dynamic process model based on the functional and non-functional requirements and cross-cutting or shared variables or concerns. Fig 3 shows that by listing non-functional requirements in different rows and having functional requirements in columns we can design a proper dynamic process for our software system. Each box in this paradigm represented non-functional requirements together with functional requirements to be able to achieve the quality and goals.

C. Coordination Strategy

As it is shown in Fig 3, the pipes that link different components do not have any direction and it is not clear how to start or initialize the problem or how the information passes
between different levels. This issue will be addressed through coordination strategy. The coordination strategy will be responsible to ensure consistency among system and/or subsystem components during an iterative process. On the other hand the concerns related to the stakeholders’ need will be prioritized by coordination strategy [13, 14]. As shown in Fig 4 parent’s response depend on children’s response. This coordination will be decided based on the project need and the constraint coming from iterative practice to find a proper match for each component. This will be discussed more in the example problem.

**Figure 4. Coordination Strategy [15].**

D. Target Cascading

Target cascading is a decomposition approach enabling top level design targets to be cascaded down to the lowest level of the modeling hierarchy [10]. The steps involved are: (i) appropriate modeling, (ii) partitioning, (iii) formulating the target cascading problems for each element of the partition, and (iv) solving the partitioned problem through a coordination strategy to capture the targets. As it is clear the main feature of target cascading is that the responses in one component will be used as target for the other component. Once a proper coordination strategy is selected the starting point will be determined. Then the responses of the starting component will be used as targets for the components that have connections with this element. This process will be iterated till a proper balance achieved with responses and targets in each element. (Fig 5)

**Figure 5. Target Cascading, All-at-once and Multi-level Approaches [14].**

3. EXAMPLE PROBLEM: VEHICLE ELECTRIC CONTROL SOFTWARE

In this section, we apply multi-level decomposition methodology to design a proper dynamic process model for vehicle electric control software system. This approach shows how to consider the ultimate quality values described as non-functional requirements from user/management/standards viewpoint. This approach enables the software to be easily adaptive to bring a proper trade-off between different systems imposed by user, management or standards.

Nowadays, vehicles are utilizing different software systems that are responsible for controlling different features. The main motivation behind developing a software system is to increase the cost-effectiveness of the product and at the same time deliver a more reliable and safe product for the customers.

In this paper, vehicle control system is defined based on three major disciplines that are actually representing three different disciplines in automotive engineering. The functional requirement of the software is defined to have a quality control software system related to engine, adaptive suspension and automatic transmission. Each of these subsystems is supposed to control different systems related to their disciplines and the related software systems are developed within separate teams. The functional requirements for this system are to have proper engine control software, adaptive suspension control software, and automatic transmission control system.

The quality attributes are represented through non-functional requirements as the fuel efficiency, drivability, ride quality, performance, emission, and noisiness. Fuel efficiency is the main concern of the automotive industry to reduce the fuel cost and energy conservation. Drivability is coming from the fact that automatic transmission should be properly calibrated to be able to change the gear based on the signals coming from the shaft. Ride quality is defined as proper suspension system to reduce the road imposed vibration due to bumps and small hurdles. The performance and emission is directly coming from the engine so that the purpose is to have an adaptive software system to change the engine inputs such that proper performance is achieved while considering the emission and reducing the air pollution. Noisiness is also considered to reduce the amount of noise generated from road bumps through suspension system as well as noise coming from engine. Noise reduction helps to reduce driver fatigue while driving long distances.

Based on the functional dependency table (FDT) described in previous section a list of non-functional requirements are arranged in each row of FDT whereas the functional requirements are shown in each column in Fig 6. In this Figure, the cross-cutting concerns are circled by red showing that features are shared in defining two quality attributes. For instance, transmission system will be a concern for both fuel economy and drivability and adaptive suspension system is a concern for fuel economy, ride quality and noise reduction.
Figure 6. Example functional and non-functional requirements of vehicle control software system in functional dependency table (FDT).

Based on FDT in Fig 6 the dependency of each quality attribute is clearly addressed so that a proper dynamic process model can be structured as a proper design for the present software system. This flowchart is shown in Fig 7. As previously discussed, the coordination strategy will determine the starting point and flow of information between each component defined in Fig 6.

Figure 7. Dynamic Process Design based on FDT.

Coordination can be performed as cyclic coordination, top-down and bottom-up. This will be decided based on stakeholders’, management or standards and can be easily altered based on prioritization. This can be also fixed by manufacturer or can be done by drivers. This capability can also be helpful through possible recall to upgrade the software system. It is worth noting that the upgrade is going to change just the coordinator part of the software system and the rest of the system will not be changed. Obviously, the software system will not be changed if refactoring is performed correctly and none of the external output or inputs is changed. Fig 8 represents two main coordination strategies defined as cyclic and top-down coordination. In top down coordination, fuel economy is considered as a driving factor to control the system this module generates responses that are targets for drivability and performance. The drivability and performance modules try to come close to the targets defined by the fuel efficiency module while providing a proper quality for their own levels. Once the targets are met the responses of drivability will be considered as target for ride quality and similarly, the Responses from performance is going to be considered as targets for noisiness. Since noisiness and ride quality are sharing a response from adaptive sustention control system, they will have a linking variable that should be iterated to satisfy both modules in this level.

4. Summary and Conclusion

Several decomposition techniques that are available in software engineering, computer science and programming are discussed briefly. Decomposition approaches are discussed through software development, process modelling, and life cycle model and in general problem solving paradigm. It is also discussed that how these approached can be fitted into different viewpoints and view models in software architecture. In specific, functional decomposition, object-oriented and aspect-oriented software development paradigms are also briefly discussed. Then decomposition approaches in general introduced and in specific multi-level decomposition is introduced and adapted from multi-level optimization. Target cascading is introduced as an alternative to perform multi-level decomposition. It can be concluded that the target cascading decomposition can be categorized as a functional decomposition approach. The distinctive feature of the presented decomposition is that the decomposed elements are interrelated in a way that the response of parent is going to be considered as targets for the child/children. This gives us a proper tool for partitioning problems that may have too many inter-relations. As an example problem, basic functional and non-functional requirements in automotive software system are taken into account and a proper decomposition is performed to prioritize the quality attributes and design a dynamic process model.

Although this paper is briefly discussed the main features of multi-level decomposition, it is seen that the presented methodology has a proper potential to be used in different phases of software engineering especially from project management to software developers point of view.

5. REFERENCES


