JOMO KENYATTA UNIVERSITY OF AGRICULTURE AND TECHNOLOGY

MASTERS OF SCIENCE IN INFORMATION TECHNOLOGY

PETER IRUNGU MWANGI

SCT321-C004-2079/2018

MIT3101 OBJECT ORIENTED PROGRAMMING AND SOFTWARE DEVELOPMENT

ASSIGNMENT

--

A discussion paper on Component Based Development, Aspect Oriented Programming and use of RESTFUL web APIs approaches to software development.

INTRODUCTION

Software development is the process of creating new software solutions or modifying existing software solutions(Young, 2013). It is an iterative logical process that aims at creating a software code that will address a unique personal or business objective, process or goal. It is a complicated process that requires careful planning and execution of tasks so as to meet the set goals. Consequently, it involves processes that include but not limited to requirements findings, data flow design, process flow design, designing and building flowcharts, technical documentation, software testing debugging and other software architecture techniques(Hull, Taylor, Hanna, & Millar, 2017). A standard software development life cycle (SDLC) is as shown in the figure 1.

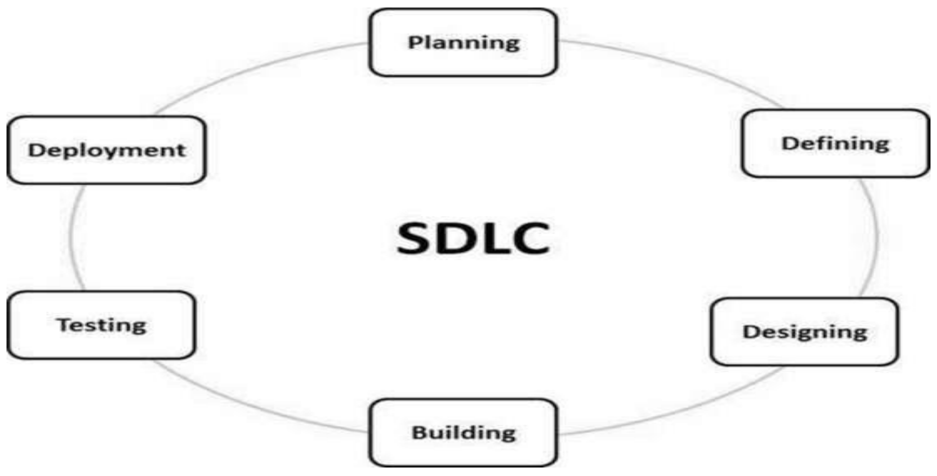

Figure 1: Software Development Life Cycle

(Source:(Mohammad, Munassar, & Govardhan, 2010))

Software development approaches define how the various tasks related to a software development can be organized. The various approaches vary from very simple limited planning approaches to very detailed, formal and structured approaches depending on the scale of the system to be developed. There exists roughly fifty-five (55) software development approaches(Zhang, Hu, Dai, & Li, 2019). Some of the examples include waterfall methodology, agile, prototyping, aspect oriented programming, extreme programming, objected oriented design, component based development, pair programming with iterative and use of RESTFUL web API among other approaches. In this paper, the component based development, aspect oriented programming and use of the RESTFUL web API, will be discussed in detail.

Keywords: Component based development, aspect oriented programming, RESTFUL web APIs.

COMPONENT BASED DEVELOPMENT

A component is an independent executable entity that can be composed of one or more executable objects(Broy et al., 2015). Councill and Heinmann defined a software component as a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard(Councill & Heineman, 2001). Additionally, Szyperski in defines a software component as a unit of composition with contractually specified interfaces and explicit context dependencies only(Szyperski, 2002). He further states that a software component can be deployed independently and is subject to composition by third-parties. The characteristics of a component are as described in the table 1.

Table 1: Component Characteristics

No.	Characteristic	Description
1	Standardized	A component has to conform to a standardized component model which may define the component interface, component, composition, component meta-data, the documentation and deployment.
2	Independent	One should be able to use and/or deploy a component without having to use or define other components.
3	Compassable	A component should be capable of providing external access to its methods

		and attributes thus all external interactions should take place from well-defined interfaces
4	Deployable	A component should operate as a stand-alone entity on a platform that implements the component model
5	Documented	The syntax and semantics of a component have to be specified so as to assist users to decide whether to use them or not

The component based development is a software development approach that emphasizes on design and development of software using reusable software components, integration being the main focus during implementation(Shivani Bahuguna,Sushil Chandra Dimri,Umesh Kumar Tiwari, 2016). It is based on the idea of developing software systems by selecting appropriate off-the-shelf components and assembling them with a well-defined software architecture. Furthermore, it is described by independent components which are specified by their interfaces, components standards that facilitate component integration, middle ware, that provides support for component interoperability and a development process that focuses on re-use. The component's interface define the services that are provided by that component to other components in addition to defining the services that are required for the component to execute as it should.

This approach emerged from the failure of the object oriented design (OOD) of not effectively supporting re-use. Components are considered to be more abstract than object classes and as a result are stand-alone service providers(Sommerville, 2005). The main objectives of the component based software engineering are: Cost and time reduction of developing large and complicated system, improving on the quality of the system by improving on the components that it consists of and faster detection of a defect within the system by debugging the small components integrated to form it.

The component based software engineering process involves five main steps for the software development as indicated in the figure 2 (Nautiyal, 2013).In the first phase, the requirement to be met are collected, outlined, analyzed and specified. After identifying the requirements to be met, the suitable components that would effectively satisfy the identified needs are then identified. This step is then followed by modifying the identified requirements according to the components that have been found. Then the architectural design of the system, based on the modified requirements, is then built and the larger system created by integrating the different components.

During the component development process, the design principles followed include:

i. Components are independent hence they should not be interfered with

ii. Component implementation is hidden

iii. Component communication is done via well-defined interfaces

iv. Component platform are shared and reduce the component costs.

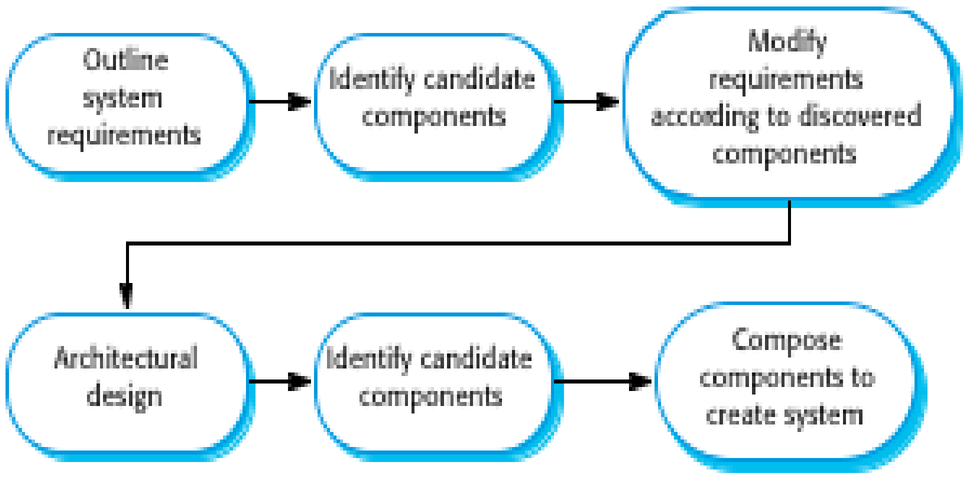
Outline system requirements
Identify candidate components
Modify requirements according to discovered components
Architectural design
Identify candidate components
Compose components to create system

Figure 2: Component Based Software Engineering cycle

(Source[(Gulia & Palak, 2017)])

During component composition process, different components are integrated with each other and with the component infrastructure. Normally, a “glue code” is written to integrate the components. There exists three types of component composition, that is:

Sequence composition; the composition is done sequentially where each of the provided interfaces are composed (Burridge, Rizzi, & Koditschek, 2016)

Hierarchical composition whereby one components calls the services of another component thus an interface of a component is composed of requirements of another component(Sommerville, 2005)

Additive composition where the interfaces of two components are combined to come up with an interface of one component(Gulia & Palak, 2017)

To achieve better and efficient results from the component based system engineering process, it is necessary to perform assessment processes that will occur separate with the development process. The assessment process includes activities such as:

i. Finding the components that might provide the required functionality
ii. Selecting the most suitable components, from those found, that is most suitable for the set requirements
iii. Component verification where the functions of each component is tested independently and also when the different components have been combined
iv. Storage of a component, which has been deemed fit for the application, in a component repository. The component repository will also include the component(s) meta-data for further exploitation of the component.

Despite the numerous advantages of using the component based software engineering process, it is associated with challenges that include but not limited to:

i. Trustworthiness as the components come with no source code
ii. Quality certification of the component
iii. Prediction of the emergent properties of the component composition
iv. Trade off analysis between the features of one component and another.

ASPECT ORIENTED PROGRAMMING

Aspect oriented programming(AOP) complements object oriented model by allowing the development process to dynamically modify the static object oriented model so as to create a system that can grow to meet the requirements(Sutton Jr, 2005). It separates the concerns, areas of interest, of a software hence improve on the modularization. The separation of Concepts (SoC) aims at making software easier to maintain by grouping features and behavior into manageable parts which all have a specific purpose(Ortin, Vinuesa, & Felix, 2011). The software concerns can broadly be categorized to high level concerns such as security and quality of service, low level concerns such as caching and buffering, functional concerns such as the system features and the business rules and the non-functional concerns that include synchronization and transactional management.

The AOP is based on aspects that implement system functionality that may be required at several different places in a program. The aspects are used alongside other abstractions such as the methods and objects. An executable aspect oriented program is created automatically by combing aspects, objects and methods according to the specifications of the source code(Gharehchopogh, Amini, & Zebardast, 2013). Additional, it contains a description of where the aspects should be included in the program as well as the code addressing the cross-cutting concerns.

The AOP software development approach involves activities shown in the figure 3.

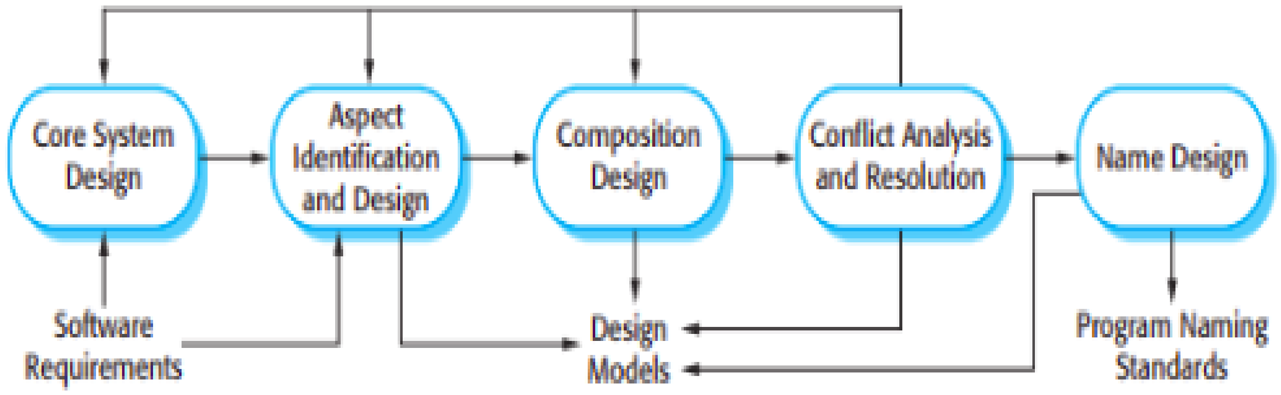

Figure 3: Aspect oriented Programming flow

These activities are:

i. *Core system design*: it involves design of the system architecture that would support the core functionality of the system. During this phase, it is important to take into consideration the dependability and performance requirement.

ii. *Aspect identification and design*: once the aspects have been identified, they can be designed separately while considering the design of the core system features

iii. *Composition design*: In this phase, the core system and aspect design are analyzed to find out where the aspects should be composed and/ or weaved to the core system

iv. *Conflict analysis and resolution*: it involves identifying, analyzing and resolving conflicts that may arise when different aspects clash

v. *Name design*: the standards for naming entities in the program are set in this phase so as to avoid the problems of accidental point cuts when two or more aspects contain similar names

The advantages associated to using AOP include but not limited to :

i. Representing the cross-cutting concerns as aspects makes it easy to understand, reuse and independently modify the concerns without regard of where the code is used

ii. Better support of the software design process by isolating application business logic from support and secondary functions

iii. The AOP approach may be used with other agile processes and coding standards

iv. Modularization supports better software designs

v. Better handling of localization of cross cutting concerns as a result of encapsulating to different modules

USE OF RESTFUL WEB APIS

The representational state transfer (REST) is a set of rules that developers follow when creating an application programming interfaces (APIs) that consists of rules that allow programs communicate(Ong et al., 2015). One of the REST rules states that one should be able to get a piece of data (also known as a resource) when you link to a specific universal resource locater (URL). The URL contains the request sent by the client. The request is made of the endpoint, method, headers and data as described in the table 2(Paredes-Valverde, Alor-Hernández, Rodr'guez-González, & Hernández-Chan, 2012).

Table 2: Components of a Request

S. No	Component	Description
1	Endpoint	It consists of the URL that a client has requested for
2	Method	This shows the type of request a client has sent to the server. They include GET, POST, PUT, PATCH and DELETE methods. These methods are used to perform CREATE, READ, UPDATE and DELETE (CRUD) activities. i. The GET method is used to perform READ operations on the server based on the request sent by the client. It returns a response based on what the client had requested for. ii. The POST method is used to CREATE a new resource on the server and returns a response of whether the creation was successful or not successful. iii. The PUT and PATCH method are used to UPDATE a resource on the server. iv. The DELETE method is used to DELETE data on the server and returns a response of whether the deletion was successful or not.
3	Header	They are used to provide information to both the client and server. It is used for several functions that include authentication and providing information about the body content
4	Data	This is the resource the client requests for

For the web, REST commonly utilizes the hypertext transfer protocol (HTTP) thus enabling developers use it without having the need additional softwares and libraries. Additionally, it provides an interface between systems using HTTP protocol to obtain data and generate operations on those data in all possible formats such as Xtensible Markup Language (XML) and JavaScript Object Notation (JSON)(Gossett et al., 2018).

A key advantage of using REST is its flexibility, reliability and validity in that the data is not tied to resources or methods. Consequently a user can handle multiple types of calls, return different data formats and perform structural changes with the implementation of hypermedia. Similarly, the separation of the client and server machines improves on the portability of the interface on other platforms hence increasing the scalability of projects and allowing different components of the developments to be evolved independently.

There exists some constraints that one needs to put into consideration before selecting RESTFUL APIs. They are:

a) It needs to operate in a client-server environment where they both operate separately and evolve individually

b) RESTFUL APIs are stateless hence calls are made independently of another call and each call contains data which will be required to complete it successfully

c) A RESTFUL API should be designed to allow storage of cacheable data due to its capabilities of handling large loads of incoming and outbound calls due to its stateless state

d) Existence of a uniform interface that will allow independent evolution of the applications without having the application’s services or models and actions tightly coupled to the API layer

e) It is a layered system as it has different layers of its architecture working together to build a hierarchy that helps create a more scalable and modular application.

REFERENCES

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., … Szyperski, C. (2015). What characterizes a (software) component? In *Software Concepts and Tools Concepts and Tools* (Vol. 19, pp. 49–56). https://doi.org/10.1007/s003780050007

Burridge, R., Rizzi, A., & Koditschek, D. (2016). Sequential Composition of Dynamically Dexterous Robot Behaviors. *I. J. Robotic Res.*, *18*, 534–555. https://doi.org/10.1177/02783649922066385

Councill, B., & Heineman, G. T. (2001). Component-Based Software Engineering: Putting the Pieces Together. In *Open Distributed Systems* (p. 880). https://doi.org/http://dx.doi.org/10.1016/j.foodchem.2014.01.059

Gharehchopogh, F. S., Amini, E., & Zebardast, B. (2013). Software Development based solution for intervention concerns problems. *International Journal of Computer Applications*, *63*(4), 16–25.

Gossett, E., Toher, C., Oses, C., Isayev, O., Legrain, F., Rose, F., … Curtarolo, S. (2018). AFLOW-ML: A RESTful API for machine-learning predictions of materials properties. *Computational Materials Science*, *152*, 134–145. https://doi.org/https://doi.org/10.1016/j.commatsci.2018.03.075

Gulia, P., & Palak, ,. (2017). Component Based Software Development Life Cycle Models: A Comparative Review. *Oriental Journal of Computer Science and Technology*, *10*, 467–473. https://doi.org/10.13005/ojcst/10.02.30

Hull, M. E. C., Taylor, P. S., Hanna, J. R. P., & Millar, R. J. (2017). Software development processes — an assessment. *Information and Software Technology*, *44*(1), 1–12. https://doi.org/https://doi.org/10.1016/S0950-5849(01)00158-6

Mohammad, N., Munassar, A., & Govardhan, A. (2010). A Comparison Between Five ModMohammad, N., Munassar, A., & Govardhan, A. (2010). A Comparison Between Five Models Of Software Engineering. International Journal of Computer Science Issues, 7(5), 94–101. https://doi.org/10.1.1.402.9250els Of Software Engin. *International Journal of Computer Science Issues*, *7*(5), 94–101. https://doi.org/10.1.1.402.9250

Nautiyal, L. (2013). Elicit – A New Component based Software Development Model. *International Journal of Computer Applications*, *63*(21), 53–57.

Ong, S. P., Cholia, S., Jain, A., Brafman, M., Gunter, D., Ceder, G., & Persson, K. A. (2015). The Materials Application Programming Interface (API): A simple, flexible and efficient API for materials data based on REpresentational State Transfer (REST) principles. *Computational Materials Science*, *97*, 209–215. https://doi.org/10.1016/j.commatsci.2014.10.037

Ortin, F., Vinuesa, L., & Felix, J. (2011). *The Dsaw Aspect-Oriented Software Development Platform. International Journal of Software Engineering and Knowledge Engineering* (Vol. 21). https://doi.org/10.1142/S0218194011005554

Paredes-Valverde, M. A., Alor-Hernández, G., Rodr'guez-González, A., & Hernández-Chan, G. (2012). Developing Social Networks Mashups: An Overview of REST-Based APIs. *Procedia Technology*, *3*, 205–213. https://doi.org/https://doi.org/10.1016/j.protcy.2012.03.022

Shivani Bahuguna,Sushil Chandra Dimri,Umesh Kumar Tiwari, L. N. (2016). Elite : A New Component-Based Software Development Model. *International Journal for Computer Techology & Application*, *3*(1), 119–124.

Sommerville, I. (2005). Component-Based Software Engineering. In *Software Engineering* (9th ed., pp. 70–95). Newyork: Pearson Education. https://doi.org/10.1007/11560647_5

Sutton Jr, S. (2005). Aspect-Oriented Software Development. In *Unifying the Software Process Spectrum, International Software Process Workshop,* (Vol. 3840, pp. 177–191). Beijing,China. https://doi.org/10.1007/11608035_17

Szyperski, C. (2002). *Component Software: Beyond Object-Oriented Programming* (2nd ed.). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Young, D. C. (2013). Software Development Methodologies. *White Paper,* 10.

Zhang, X., Hu, T., Dai, H., & Li, X. (2019). Software Development Methodologies, Trends, and Implications. In *Software Development Methodologies, Trends, and Implications* (p. 6). Software Development Methodologies, Trends, and Implications.