Software Reuse

Marcelo Feliz nº50356

Department of Informatics, University of Évora, Portugal

Abstract. This article briefly summarizes software reuse, including how it is done and disadvantages and advantages, paired with some additional information about that subject. Development of service based software, also known as Service-oriented Architecture, how it should function and what benefits in using this approach. Software testing, we will summarise the most important types of tests and what is their objective. Testing and approaches to software and service-based systems as well as methods to ensure the consistency of their services through testing. We will also address methods to ensure the consistency of services throughout their lifecycle, not directly but instead throughout the article, as this is mainly done by continuous testing and ensuring that the software keeps working as intended. In the end there is a small information related to business that work with service-based software and a general idea of how they should control their services.

Keywords: software, software reuse, tests, software testing, services

1 Introduction

Software reuse or code reuse is the act of using already existing software to create new software. A reusable resource concerns to any information (physical or electronic) that a developer may need in the process of creating software. On the other hand, reusability is a measure that can be reused in another situation. Consumption of reusable resources means "development with reuse", and production of reusable resources means "development for reuse". [6]; [9]; [3]; [21]

Software reuse has been used since the beginning of programming, however only in 1968 it was declared as a renown study area of software engineering. There are several types of reuse, in regards to the motivations, the reuse can be opportunistic or planned, it can also be an internal reuse or an external reuse, lastly depending on the structure of the software, one can be referenced or bifurcated [3]; [21].

Software reuse's aim is to improve software quality and productivity, as well as reduce the development cost. Software reuse is of great interest because companies and developers want to build systems that are bigger, more complex, more reliable and less expensive. There has been a lot of research among software reuse, the active areas of research include reuse libraries, domain engineering methods and tools, reuse design, domain specific software architecture, business, finance, among others. [6]; [9];

A fundamental Service-Oriented Architecture (SOA) provide the basis for building software systems with services, it has the capability to support flexible composition of independent services. In an SOA, software resources are organized as services, which are well defined and capable to provide standard business functionality. Service-Oriented Architecture has several specificity's, the service maintains its own state, they are platform independent and the SOA assumes that various services can be dynamically located, invoked and re-combined. An SOA provides a flexible architecture that unifies business processes by modularizing large applications into services. [14]; [16].

The rest of this article is structured as follows. In Sect. 2, we specify the concept, objectives and specificity's of software reuse. In Sect. 3, we address the theme of development of service based software. In Sect. 4, we present the theme of software testing and it´s importance in the process of developing software. In Sect. 5, we present several approaches to software testing. Lastly, in sec. 6, will be discussed the consistency of services throughout their lifecycle.

2 Software Reuse

Code reuse can be made in different ways,the simplest one being by copying parts of code, complete functions or libraries, and a more complex one being the reuse of frameworks[3]; [21]. Reusing code normally implies using dependencies, which difficult the maintenance of the code [23], meaning problems like the following will be harder to correct or prevent:

- Correcting defeats or their cause
- Repair or replace faulty or worn-out components without having to replace still working parts
- Preventing unexpected working conditions
- Maximizing a product's useful life
- Maximize efficiency, reliability, and safety
- Meet new requirements
- Make future maintenance easier
- Cope with a changing environment

The main objective of code reuse is to save time and resources. The take from reuse is that parts of a software already written can or should be used to create other programs later. Libraries are a good example of code reuse. Libraries are code specifically made to be reused. Programmers can decide to create this custom libraries to their own use or internal abstractions on part of the code. Modularity is one of the characteristics that makes software reuse much easier, also loose coupling, high cohesion, information hiding and separation of concerns are some additional examples of other important characteristics [3]; [21].

The types of reuse we are going to considerate are relative to the knowledge forehand if the programmers know that they are going to use software reuse or not. This means that reuse can be Opportunistic or planned, the first one meaning that the programmers realised they could use reuse only during the project or at the start, either code or components but with no prior plan. In the other hand, planned reuse means that the programmers know before starting working that they can use already made code and have a plan for that, as well as design their components so that it can be reuse in future projects [20]; [21]; [19]. In regards to the usage of our own code from a different project and the usage of code made from other team, it can be classified as internal reuse and external reuse. Internal reuse meaning we are reusing code made by our team in a earlier project. External reuse means the team decided to use code from a third-party [15]; [5]. Lastly, in regards to the structure, the code can be Referenced, that means the code has a reference to the reused code, and thus they can have distinct versions. The code can also be forked, this way the used code is contained private or locally, and thus they share a single version [22]

Software reuse is a strategy made to increase productivity and to improve the quality of the software. Although in theory it is simple, well made software reuse implementation is difficult. There are some problems related to software reuse that need to be addressed such as not having a clear and well-defined product vision, there is a need for continuous management support and leadership to ensure success and to support SPL(software product line) engineering, an appropriate organization structure is needed.

2.1 Motivations to use Software Reuse

There are several motivations to use reuse, today's organizations, more than ever, face competitive pressures in shortening the time required to bring a product to the market, reducing software development and maintenance costs, and increasing the quality of their software. Because of that, some organizations have increasingly turned to a reuse strategy in order to meet their business goals. Implemented appropriately, reuse can aid in achieving many goals.

Increase productivity, after an initial investment, reuse of existing assets will enable projects to decrease the cost of developing and maintaining software. Hewlett-Packard (HP) has an extensive history with software reuse, their reuse projects have reported productivity increases from 6 to 40% [4]. Shorten time-to-market, by reusing software to shorten the critical path in delivering a product, organizations within HP have reported a reduction in time-to-market from 12 to 42%.Improve

software quality, software that has been used multiple times will possess fewer defects than freshly coded components. One HP organization had mature reused code that had a defect density which was ten times better than new code. Quality improvements in HP products from reuse have ranged from 24 to 76% [4].

Software reuse brings several advantages, such as: provide consistency and interoperability across products, standard interfaces and common use of components across products facilitates ease of use and interoperability. For example, when several software products reuse the same user interface scheme, terms and conventions are used consistently. Reuse also contributes to interoperability. For example, when a component that contains an error-message handling routine is reused by several systems, these systems can expect consistent behavior.

Ensure product/system conformance with user requirements through prototyping, because the availability of reusable components facilitates prototyping, user requirements may be more easily validated. This prototyping will also enable detection and resolution of defects earlier in the software life cycle - avoiding more costly fixes later in the life cycle.

The risk in creating new software is reduced when available reusable components already encompass the desired functionality and have standard interfaces to facilitate integration.

Leverage technical skills and knowledge, reuse enables specialists to optimize software architectures and assets which may then be reused by others who are focusing on meeting product features and market needs.

Improve functionality and/or performance, reuse allows for the investment of time to improve functionality and/or performance. Because this time can be amortized over multiple uses of the assets, this investment for such improvements is more economically justified than the case where they would only be for single product [2]; [22].

3 Development of service based software

Development of service based software or Service-oriented architecture (SOA) is a type of software design that allows the components to be reused using interfaces of services with a common communication language in a network. A service is a unity or a independent group of software functionalities, develop to solve a specific situation, like recovering some information or executing an operation. This services can be accesses remotely and it is totally possible to interact with them and update them in an independent way. This means, SOA integrates software components that were implanted and are kept separated, allowing communication and work between them to form applications that work in different systems [13]; [14];[8].

Before SOA, in the final decade of 1990, the connection between services in different systems was a complex process that involved the integration of point-to-point(P2P), sometimes including routing, data transformation and others. Not only that but in every new project, the developers needed to recreate that connection. That model was known as monolithic. If even only one aspect of the application didn't work the intended way, it was needed to put everything offline temporarily to make corrections and then run the application again as a new version [10].

The objective of SOA is to allow users the grouping of functionalities in a way to obtain dedicated applications that will be built almost entirely from software services already made. The more the grouping, the lower the interface quantity needed to implement any set of functionalities, whoever, big groupings are harder to reuse. Each interface has a quantity of processing, because of that we have to consider the performance and detail of the services. With the perfect SOA, coordinate the services would be enough to create a new application.

Whoever, in order for SOA to work, no interaction should exist between the functionalities or inside them. Services should be implements with more functionalities than functions or classes, this way the designer of the application does not get overloaded with the complexity of the quantity of objects with bigger detail.

The SOA architecture has the service orientation as is principal fundamental of the design. This means that if a service presents a simple interface that abstracts the complexity involved in it, users can use that services without the need to fully understand the implementation [26]

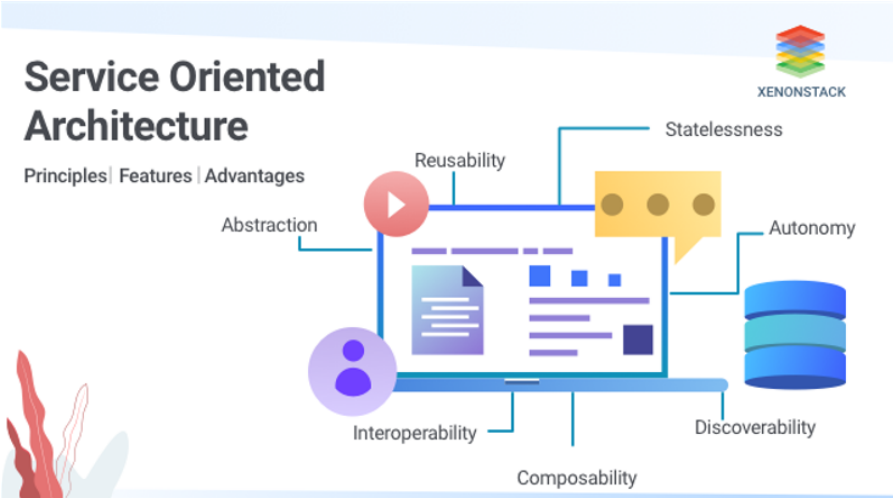

Fig. 1. SOA[25]

There are several advantages in the development of service based software:

- Accelerated and more flexible time to market, that is, reusing the services makes it easier and faster to build applications. The developers don't always need to start from scratch.

- Market focus: With SOA, it is easier to developers to escalate and enlarge the use of one functionality to platforms or new environments.

- Reduced costs obtained by a bigger agility and efficiency in the development.

- Easy maintenance, because all services are self-sufficient and independent, it is possible to change and update them based on need, without affecting other services.

- Scalability: SOA allow the execution of services using various services, platforms and programming languages, because of that, there is a big upgrade in scalability. Not only that, but SOA uses a patronizes communication protocol, allowing enterprises to reduce the interaction between clients and services. With this reduction to the interaction level, it is possible to escalate applications with less urgency and problems.

- Bigger reliability, SOA allows the developers to create reliable applications, because it is easier to debug smaller services than a big code.

Availability, in the sense that SOA's resources are available to everyone [16]; [24]

4 Software testing

Testing represents an extremely important step in the process of developing software, this is true because tests have the objective of validate if an application is working correctly and if all specific requirements are met, not only that but testing can also reduce the risk of failure in operation. Testing is not only about executing tests and seeing the results, executing tests is only one of the activities. The complete process of testing requires planing, scanning, modeling, and implementation of tests [12]; [18]; [7].

Testing is important in 3 main ways. To reduce risk of failure during operation. Contribute with quality of components or systems and to meet the contractual requirements or specifics from the sector they are aimed for. There are various types of tests, but first lets address the difference between test and debug. Test consists on executing tests that can show flaws caused by defects in the software. On the other hand, debug is the development activity that locates, scans and corrects that defects [12]; [18]; [7].

4.1 Different types of software testing

-Unit tests, this tests have the objective of validate small parts of the software, this is made with the possible inputs and outputs. The unities used in this type of testing are the smallest possible

to test in a system, normally functions that receive arguments and return a value. Unit testing are very used in TDD(test-driven development), that is a development methodology that lay in testing to implement functionality instead of the apposite [12]; [18]; [7]; [11].

-Functional tests, this type of tests instead of evaluate the code of the program, analyze his external behavior. Normally this type of tests is based on system requirements and consists on doing real actions on the system, inputting values and evaluating the output [12].

-Regression tests. This is a test technique applicable to a new version of the software. Consists in applying all tests that were already applied at each new version of the software or at each cycle. To increase the productivity and the viability of the tests, it is recommended the usage of a test automation tool, so that all the tests can be executed again easily [12].

-Integration tests, have the objective of unite the different modules of the system and test them as a set. They are done after other tests that assure the individual functionality of the parts [18]; [7]; [1].

-Acceptance test form one of the last stages in development of a system, this is true because they validate if the software is ready to go into production. To evaluate this, normally a client or tester, with good knowledge of the requirements, evaluate if the application is working as intended and can be considered ready [12].

-Performance tests, also called load tests, have the objective of submit the software to intense situations of use and evaluate their performance. This type of testing is very common in web applications, they are submitted to a big number of requests to verify if the server can support that much load. In this stage, normally specific tools are used to simulate the use of the applications in a bigger scale [12].

-Security tests aim to ensure the right application of security defined to the software, reaching this way an environment that can be safer [12].

-Automated tests. During the development of a software various tests can be made and normally each one of them will be executed various times. The automation of this process has the objective of reducing time wasted on that step as well as reducing the probability of human failure. Nonetheless, this is not a easy task, and because of that it is important to plan and use efficient tools [12].

Test reports, with the information of all the tests done during the testing, either them being successful or not, those values can help in the decision and plan of new tests and evaluation of the quality of the process.

The cost to find defects and remove them in the specification stage of the development is very low. In the design stage this costs get higher, and as the stages go on, so does the price, meaning that in the production stage the price of solving a bug can be very expensive. This happens because problems in the specification stage can be easily solved while when in the latter stages they usually require more tests, meaning more time, correction or discard of code and in the worth cases a throwback to the specification and design stage. When in the production stage, in addition to all the previous costs, there are expenses from attendance to the client, replication of defects in laboratory and even possible recalls. In the end the conclusion is simple, the sooner the problems are found the less expensive they will be to both the developers and the clients [17]; [18]; [7].

5 Testing and their approaches to software and service-based systems

Nowadays testing is very common, and there is not doubts in their importance on the quality of an application. Many types of tests and techniques are already used and known, whoever as the architecture changes and evolves, the techniques need to evolve too.

When we have a system based in sessions we can test our components individually, now it is very common to work with service based architecture, and generally we have a BFF layer (Back For Front), responsible for using several other services to applications, either mobile or web. Behind this layer we have different services that not only respond to the BFF layer but also to each other. The problem in this situation is, how do we test this? Many developers do what they already know and usually do in other situations, either for lack of time or knowledge, unit tests and integration of the service they are using and rarely test the system as one. However, the system should be tested as one because even thou we have tests and they pass, rules in the business or interface can change, not only that but other services possibly don't know about that changes, or themselves

could have been changed. This won't usually happen in a small team, but as a project gets bigger it will certainly possible to happen [12]; [18].

We already addressed the diverse types of tests and we know that they can be automatized, if w write those tests to everything their execution will take too much time and become impracticable. Every time a problem is found running all the tests, it is not possible to do. And because of that we need a good strategy.

One efficient strategy to automatized tests is to think of them like a pyramid, where in the base we have the faster tests and lightest and at the top the slower and heavier ones. This means that in the base we will have unit tests,because they are fast to execute, followed by integration and functionality tests, as this ones have more dependencies therefore bigger time. In the end we will have the manual and interface tests, the "bigger" ones [12].

6 Service management for business

As we know services have a lifecycle, what this means is that like many other things it is inserted in the concept of beginning, middle and end. This means that changing and transformation are occurring from one phase to another. Changing and evolution is needed to prevent a service becoming outdated.

If we take a look at the cycle of the services in the market we can observe that many services no longer exist and many tasks stopped being done. Whoever other services started being relevant, and that's what is important about the lifecycle of a service.

Because of that, it is very important to pay attention to create a table in witch all services offered are organized so that a easy overview of what the company has, can be possible. Adding to that information, the table can have things like, what abilities and skills are needed to upgrade and perfect the services already provided. This way the company can have a better view of what services are in decline and respond accordingly.

One of the methods to obtain consistency of services throughout their lifecycle is based on standards from ISSO like ISO 9001 defined by specialists and focused on management of a business's quality

There are several phases to the development of a service, such as:

Service Design: In this phase, the service is designed and molded with his values, objectives and divulging.

Service Transition: In this phase, it is when things start to be done in practice, this is the period when reunions are made with clients to test and evaluate the assertive and negative points of the new service.

Service Operation: In this phase the service is already operational with total efficiency. The service should be know by the public because of divulging already made.

Continuous Service Improvement: With the service in execution it has to be evaluated. It is needed to collect data and information, understand the time of each process, listen to the clients and save that information. This information will be helpful to create strategies to the service, either to differentiate from other or to try to have more quality.

Understanding the cycle is very important, as it makes it possible to know in which phase the service is. From that information, the team can organize to know what is the best for the service, upgrade it or focus in new services and innovations.

There are also some software that can help with the management of services lifetime by: Optimized spare parts planning Guaranteed access to up-to-date service information Connected Services Resources

7 Conclusion

Software reuse can be a very strong tool as we could see, however it needs to be done in a planed way, as it can turn in to a problematic situation in terms of maintenance. With a good plan and strategy reuse is without a doubt a good way of developing software, not only increasing the speed of the development, but probably also the quality by using good services implemented by other

experienced developers and tested by many other users. Not only that, but productivity will also increase while the costs will decrease. Testing has one of the most important roles in service-based systems. There are various types of tests and they should be used accordingly. At last, running all the tests in a scheduled way or by tasks is one of the most important ways of maintaining a software working as intended.

References

[1] Hareton KN Leung and Lee White. "A study of integration testing and software regression at the integration level". In: *Proceedings. Conference on Software Maintenance 1990*. IEEE. 1990, pp. 290–301.

[2] James W Hooper and Rowena O Chester. *Software reuse: guidelines and methods*. Springer Science & Business Media, 1991.

[3] Charles W Krueger. "Software reuse". In: *ACM Computing Surveys (CSUR)* 24.2 (1992), pp. 131–183.

[4] Wayne C Lim. "Effects of reuse on quality, productivity, and economics". In: *IEEE software* 11.5 (1994), pp. 23–30.

[5] William Frakes and Carol Terry. "Software reuse: metrics and models". In: *ACM Computing Surveys (CSUR)* 28.2 (1996), pp. 415–435.

[6] Yongbeom Kim and Edward A Stohr. "Software reuse: Survey and research directions". In: *Journal of Management Information Systems* 14.4 (1998), pp. 113–147.

[7] Jiantao Pan. "Software testing". In: *Dependable Embedded Systems* 5 (1999), p. 2006.

[8] David Sprott and Lawrence Wilkes. "Understanding service-oriented architecture". In: *The Architecture Journal* 1.1 (2004), pp. 10–17.

[9] William B Frakes and Kyo Kang. "Software reuse research: Status and future". In: *IEEE transactions on Software Engineering* 31.7 (2005), pp. 529–536.

[10] Norbert Bieberstein et al. *Service-oriented architecture compass: business value, planning, and enterprise roadmap*. FT Press, 2006.

[11] Per Runeson. "A survey of unit testing practices". In: *IEEE software* 23.4 (2006), pp. 22–29.

[12] Raymond McLeod. "Software Testing: Testing Across the Entire Software Development Life Cycle". In: (2007).

[13] Liam O'Brien, Paulo Merson, and Len Bass. "Quality attributes for service-oriented architectures". In: *International Workshop on Systems Development in SOA Environments (SDSOA'07: ICSE Workshops 2007)*. IEEE. 2007, pp. 3–3.

[14] Mike P Papazoglou and Willem-Jan Van Den Heuvel. "Service oriented architectures: approaches, technologies and research issues". In: *The VLDB journal* 16.3 (2007), pp. 389–415.

[15] Veronika Bauer. "Facts and fallacies of reuse in practice". In: *2013 17th European Conference on Software Maintenance and Reengineering*. IEEE. 2013, pp. 431–434.

[16] Bernard P Zeigler and Hessam S Sarjoughian. "Service-based software systems". In: *Guide to Modeling and Simulation of Systems of Systems*. Springer, 2013, pp. 205–230.

[17] Vahid Garousi and Mika V Mäntylä. "When and what to automate in software testing? A multi-vocal literature review". In: *Information and Software Technology* 76 (2016), pp. 92–117.

[18] Muhammad Abid Jamil et al. "Software testing techniques: A literature review". In: *2016 6th international conference on information and communication technology for the Muslim world (ICT4M)*. IEEE. 2016, pp. 177–182.

[19] Tommi Mikkonen and Antero Taivalsaari. "Software reuse in the era of opportunistic design". In: *IEEE Software* 36.3 (2019), pp. 105–111.

[20] Juliane Fischer et al. "Reengineering workflow for planned reuse of IEC 61131-3 legacy software". In: *2020 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*. IEEE. 2020, pp. 1126–1130.

[21] Niko Mäkitalo et al. "On opportunistic software reuse". In: *Computing* 102.11 (2020), pp. 2385–2408.

[22] Wardynski DJ. *Brainspire*. https://www.brainspire.com/blog/maximize-your-ability-for-code-reusability-8-effective-ways. Accessed: 2022-5-05.

[23] Wolff E. *Code Redundancy or Reuse?* https://www.innoq.com/en/blog/code-reuse-or-redundancy/.

[24] IBM Cloud Education. *SOA (Service-Oriented Architecture)*. ://www.ibm.com/cloud/learn/soa.

[25] Gill S. *Xenonstack*. https://www.xenonstack.com/insights/service-oriented-architecture. Accessed: 2022-05-05.

[26] Nolle T. *TechTarget*. https://www.techtarget.com/searchapparchitecture/definition/service-oriented-architecture-SOA. Accessed: 2022-5-07.