REVISTA COLETA
Científica

Year 1, Vol. I, n.1, Jan.- Jul . , 20 17

(CC BY 4.0) | ISSN 2763-6496

DOI: https://doi.org/10.5281/zenodo.7416079

ARK: https://n2t.net/ark:/24285/RCC.v1i1.4

SGHC Fees accounting

SGHC Accounting Fees

Received: 02/16/2017 | Accepted: 06/11/2017 | Published: 06/20/2017

Maydson Filipe Ramos
https://orcid.org/0000-0001-7798-2952
http://lattes.cnpq.br/83932303824345346
CNEC College, Unai, MG, Brazil
Email: maydsonunai@hotmail.com

Abstract
This article proposes to make the process of making accounting fees more agile and practical through electronic means, from launch to closing. Based on this, our objective was to develop a software that allows the user to generate the monthly fee for each client with all the services performed for them, as well as to generate periodic reports with the average of the accounting fees paid. Among the main features of the software developed, discussed in this article, one can mention the option that is available to the user to generate graphical reports with the average monthly accounting fees that a given customer paid in a certain period of time. Finally, it is concluded that the objective proposed in this research was achieved, since the software can facilitate the work of the accounting professional, since it will no longer add the fees manually, saving useful time.

Keywords: Accounting Fees. Software. Graphics . SGHC

Abstract
This article aims to make the process of making accounting fees more agile and practical through electronic means, from launch to closing. Based on this, we aimed to create software that allows the user to generate the monthly fee for each client with all services performed for them, as well as generate periodic reports with the average accounting fees paid. Among the main characteristics of the developed software, covered in this article, it is possible to mention the option that is available to the user to generate graphical reports with the average monthly accounting fees that a client paid in a certain period of time. Finally, it is concluded that the objective proposed in this research was achieved, since the software can facilitate the work of the professional in the accounting area, since it will no longer add the fees manually, saving useful time.

Keywords: *Accounting fees. Software. Graphics. SGHC*

Introduction

This article was prepared through field research in a company that provides accounting services, whose main objective was to make the process of making accounting fees more agile and practical through electronic means, from launching to closing, as well as streamline the process of making the fee; offer transparency to the client; generate periodic reports to administrators; store all customer fees.

Glimpsing the need for modernization and speed for the development of the company and customer satisfaction with regard to the control of accounting fees, a system was thought of that could fulfill such needs and, at the same time, contribute to the technology that modernity demands in any work environment to better carry out the company's proposals.

The company for which the software will be developed does not currently have a system to manage its fees and services provided by it. Considering that its clientele grows in large proportions, the control of these fees becomes increasingly difficult, thus, the administrator is unable to measure the average expenses of each client and neither his monthly or annual billing. The objective of the project is to develop software that allows the user to generate the monthly fee for each client with all the services performed for them, as well as to generate periodic reports with the average of the accounting fees paid.

This referential present aims to demonstrate the main activities, common to software development processes, used in organizations that seek a quality standard in the development of their applications, taking into account the opinion and work of some professionals in the area. A software development process is a set of activities and associated results that produce a software product (Somerville , 2007)

problem formulation

Among the problems faced in the preparation of fees we can mention:

- There is no breakdown of the services provided to the customer during the month;
- The monthly fees generated are not stored for consultation which, if necessary, must be done manually;
- Waste of useful time in the sum of the services provided, which is still done manually;
- The company's need for integration with new technologies that can strengthen the quality of services provided, as well as its credibility.

Main goal

Make the process of making accounting fees more agile and practical through electronic means, from launch to closing.

Specific objectives

The specific objectives of the system are:

- Streamline the payroll process.
- Offer customer transparency.
- Generate periodic reports to administrators.
- Store all customer fees.

Software development process concepts

Common activities of software development processes

Definition of Requirements/Implementation

It is the initial part of the development where the investigation tasks are found, it is in this phase of elaboration of the systems that the first meetings with the clients are held defining their needs for gathering the requirements.

> System requirement definitions specify what the system must do (its functions) and its essential and desirable properties. As with software requirements analysis, creating system requirements definitions involves consultation with customers and end users of the system.
> An important part of the requirements definition phase is establishing a set of objectives that the system must meet. This set should not necessarily be expressed in terms of system functionality, but should define why the system is being purchased for a given environment . The software implementation stage is the process of converting a system specification into an executable system. It always involves the software design and programming processes, but if an evolutionary approach is used, it may also involve the refinement of the software specification (Somerville , 2007).

Software Testing

According to Sommerville (2009, p. 7), the software must be validated to ensure that it does what the customer wants.

Like most engineering activities, building software depends primarily on the skill, interpretation and execution, of the people who build it; therefore, errors end up appearing, even with the use of software engineering methods and tools.

> So that such errors do not last, there are a series of activities, collectively called "validation, verification and testing", with the purpose of guaranteeing that both the way in which the software is being built and the product itself are in accordance with the specified . (Delamaro , Maldonado & Jino , 2009).

Software development methodologies

According to Sommerville (2007, p. 43) Software processes are an abstract representation of a software process. Each process model represents a process from a certain perspective and thus provides only partial information about that process. These generic models are not definitive definitions of a software process. Rather, they are process abstractions that can be used to explain different approaches to software development.
The process models are:

- *Waterfall model* : Considers the fundamental process activities comprising specification, development, validation and evolution, which represents them as separate process phases, such as requirements specification, software process, implementation and testing.
- *Evolutionary development* : merges specification, development and validation activities. In the beginning system is rapidly developed based on abstract specifications. This system is then refined with customer input to produce a system that satisfies the customer's needs.
- *Component-based software engineering* : This approach relies on the existence of a significant number of reusable components. The system development process focuses on integrating these components rather than developing them from scratch.

Iterative and Incremental Model

Incremental Development is a staged planning strategy in which various parts of the system are developed in parallel, and integrated when complete. It does not imply, require or presuppose iterative or waterfall development – both are rework strategies . The alternative to incremental development is to develop the entire system with a single integration.

Iterative Development is a rework planning strategy in which the time to review and improve parts of the system is predefined. This does not assume incremental development, but it works very well with it. A typical difference is that the output of an increment is not necessarily the subject of further refinement, and its testing or user feedback is not used as input to revision plans or specifications for successive increments. In contrast, the output of one iteration is examined for modification, and especially for revising the goals of successive iterations.

The basic idea behind the iterative approach is to develop an incremental software system, allowing the developer to take advantage of what was learned during the initial development phase of a system release. Learning occurs simultaneously for both the developer and the user of the system.
[SOURCE: http://protocoloti.blogspot.com.br/2012/03/os-modelos-de-desenvolvimento-de.html]

Requirements Engineering

Classification of requirements

The set of requirements is what determines the scope of the project, it is everything that the software must implement and will be the reference for the entire development team. According to Sommerville (2007 p. 83, 84, 85) the requirements are classified as:

Functional Requirements : They are statements of functions that the system must provide, how the system must react to specific inputs and how it must behave in certain situations, they can also explicitly state what the system must not do.

Non-Functional Requirements: These are restrictions on the services or functions offered by the system. Among them are time constraints, on the development process or standards.

Domain Requirements: These are requirements that originate from the system's application domain and reflect characteristics of that domain. Can be functional or non-functional requirements

Elicitation Techniques

Requirements Gathering Techniques

The requirements survey techniques aim to overcome the difficulties related to this phase. All techniques have their own concept and their respective advantages and disadvantages, which can be used together by the analyst. Some requirements gathering techniques will be briefly presented in this article.

Viewpoint-oriented survey

Viewpoint-oriented approaches to requirements engineering recognize different points of view and use them to structure and organize the elicitation process and the requirements themselves. An important capability of viewpoint-oriented analysis is that it recognizes the existence of multiple perspectives and provides a *framework* to discover conflicts in requirements proposed by different *stakeholders.*

The VORD method (*viewpoint-oriented requirements definition* – view-oriented requirements definition) was designed as a *framework* service-oriented for requirements gathering and analysis.

Ethnography
Ethnography is an observation technique that can be used to understand social and organizational requirements, that is, to understand organizational politics as well as work culture in order to become familiar with the system and its history. Social scientists and anthropologists use observational techniques to develop a complete and detailed understanding of particular cultures.

Workshops
elicitation technique used in a structured meeting. A team of analysts and a selection of *stakeholders* that best represent the organization and context in which the system will be used, thus obtaining a well-defined set of requirements.

prototyping
Prototype aims to explore critical aspects of a product's requirements, quickly implementing a small subset of this product's features. The prototype is indicated to study user interface alternatives; communication problems with other products; and the feasibility of meeting the performance requirements. The techniques used in the elaboration of the prototype are several: user interface, textual reports, graphical reports, among others.

interviews
The interview is one of the simplest traditional techniques to use and that produces good results in the initial phase of data collection. The interviewer should give the interviewee room to express his ideas . It is necessary to have an interview plan so that there is no dispersion of the main subject and the interview becomes long, leaving the interviewee tired and not producing good results.

Questionnaires
The use of a questionnaire is indicated, for example, when there are several groups of users that may be in different parts of the country. In this case, specific follow-up surveys are designed with selected users, where the potential contribution seems more important, as it would not be practical to interview all people in all locations.

brainstorming
brainstorming it is a technique for generating ideas . It consists of one or several meetings that allow people to suggest and explore ideas .

JAD
JAD (*Joint Application Design*) is a technique for promoting cooperation, understanding and teamwork among user developers. JAD makes it easy to create a shared vision of what the software product should be. Through their use, developers help users to formulate problems and explore solutions. In this way, users gain a sense of involvement, ownership, and responsibility for the product's success.

Requirements Management

According to Filho (2003, p.5) a common problem in software development is the **instability of requirements,** which occurs when customers and users bring new requirements, or change requirements, when development is already in an advanced stage. Instability usually has a high cost, which means losing work already done and patching others. Good requirements engineering reduces requirements instability by helping to get the requirements right at an earlier stage of development. However changes in requirements are sometimes unavoidable. Requirements engineering is subject to human limitations. Requirements management is the software engineering discipline that seeks to keep the set of requirements of a product under control even in the face of inevitable changes.

Software Analysis and Design

Analysis and Structured Design

According to Jones (1980, p. 3) Structured design is a disciplined approach to computerized system design, an activity that in the past has been notoriously uneven and full of problems. Its characteristics are systems that are easy to understand, reliable, capable of "long life ", easily developed, efficient and that WORK.

Object Oriented Analysis and Design

Yourdon , Argila (1999, p. 7) defined that an AOO model portrays objects that represent a specific application domain, along with several structural and communication relationships. The AOO model serves two purposes. First, to formalize the "view" of the real world within which the software system will be built. It establishes the objects that will serve as the main organized structures of the software system, as well as the main organizational structures that the real world imposes on any software system built in that application domain.

Aspect Oriented Software Design

Aspect-oriented programming proposes using aspects [Kiczales ET AL 1997] to separate the code that implements the non-functional purposes of an application, called orthogonal concerns or cross-sutting , from the code that implements the application's functionality. Aspects are modules that implement code regarding these concerns.

Data Storage Mechanisms

Relational Database

According to Nassur , Setzer (1999 p. 5 and 35) in In the relational model, data are stored in two-dimensional screens, called relations, each row of the table represents an element of the data set and each column of the table contains values from a defined set, called domain. A row in the relation is called a tuple .

Object Oriented Database

An object-oriented database is basically a system in which the storage unit is the object, with the same concept as object-oriented programming languages. The fundamental difference is the persistence of the objects, that is, the objects continue to exist even after the program is closed. We can classify systems that offer persistence to objects in four categories according to their architecture, which are, Extended Database System, Database Programming Language, Object Managers, and Database System Generators.

Test Software

Test Process Activities

According to Rios, Moreira (2006, p. 37) the testing activities are as follows:

- ✓ Prepare the risk analysis of the test project.
- ✓ Analyze the possibilities of minimizing business risks through testing.
- ✓ Evaluate system requirements and system planning.
- ✓ Evaluate the preliminary data model and other models and diagrams of the system.
- ✓ Identify quality and performance indicators
- ✓ Establish which products will be checked.
- ✓ Develop the Project Test Strategy document.
- ✓ Establish the types of tests to be performed in development and deployment.
- ✓ Define test priorities for modules or subsystems, aiming at integration tests .
- ✓ Identify alpha and beta testing needs.
- ✓ Set priorities for tests
- ✓ Define Infrastructure (hardware/software/tools/personnel) to be used.
- ✓ Define the operational environments for carrying out the tests.
- ✓ Identify the need for static testing
- ✓ Identify subdivisions (subsystems, modules) required for the system, aiming at better management of the testing process through its modularization.
- ✓ Validate the test strategies with developed areas.
- ✓ Conduct a technical review of the products in this stage.
- ✓ Communicate the testing strategy to the testing team.
- ✓ Identify the techniques to be adopted for the tests.

According to Fernando Rodrigues, the tests must be carried out at several levels:

✓ **Unit testing or unit tests.**

✓ This test level aims to test the smallest software unit, trying to cause business rule failures. This test is done by small snippets of code in isolation.

✓ **Integration test**

✓ Checks for gaps between modules or interfaces when these are integrated when trying to make a whole work.

✓ **system test**

Evaluate the system as a whole, as if you were an end user, entering real data and analyzing whether your answers meet the requirements.

✓ **acceptance test**

✓ At the beginning of development, the functionalities that will be tested are declared so that the software is declared as delivered, this level of testing is usually carried out by a group of users who verify some functionalities so that the software is considered as accepted.

white box test

White box testing consists of testing line by line of code, flows, conditions, loops, for those who are familiar with Visual Studio it is the Unit Test with the Code option Coverage enabled.
This type of test aims to achieve the maximum possible coverage.

black box test

The black box test consists of testing the functionality as a whole, not caring about its flow but the expected result.

This test analyzes the handling of errors (exceptions) the validations, functions, for those who are familiar with Visual Studio, the unit tests without Code Coverage are considered as black box testing.

Development Paradigms

Structured Programming vs OO Programming

Flávia Neves explains it this way:
an object-oriented system is divided into components and no longer into processes. Imagine a financial system, where we would do all the administration of a company. We would have the following differences:

- **POO** : We would have a vendors object for example where all the roles would be grouped in the object and nowhere else.
- **Structured** : Supplier routines and functions would be spread throughout the system, such as accounts payable, accounts receivable, registration, etc.

Now imagine the supplier register, with all its routines and functions:

- **Structured** : If you need to change some data, function or property in the future, what in your program will be affected? What will have to be restructured? Imagine going back to the testing phase and analyzing your entire system until you're sure the change you made didn't trigger a finite list of changes you'll have to make across the entire system.

- **POO** : the properties, functions and routines of the supplier object are all in a single object, encapsulated, facilitating this future need for changes and updates.

Aspect Oriented Programming

The purpose of Aspect-Oriented Programming (AOP), which emerged as a complement to Object-Oriented Programming (OOP), is to make it easier to modularize *crosscutting* concerns, which, in a computer program, are aspects that affect others. Often these aspects cannot be clearly separated from the rest of the system, either in design or implementation, resulting in program scattering or clutter .

Aspect-oriented technology has many potential benefits. It provides a method of specifying and encapsulating cross-cutting characteristics in a given system. This power, in turn, allows us to do a better job of maintaining systems as they evolve (and they tend to). APOA would allow us to add new functionalities, in the form of features, to a system in an organized way. Thus, improvements in the structure would allow us to keep systems running for much longer and improve them without having to perform a complete rewrite of the code.

Frameworks and Architectures for WEB Development

Model - view - controller (MVC)

Civil Engineer Christopher Alexander created what is considered the first design pattern in the mid-1970s. A design pattern is considered to be a tested and documented solution that can solve a specific problem in different projects. Through Alexander's work, software development professionals used these concepts to initiate the first documentation of design patterns, making them accessible to the entire development area.

The then employee of the Xerox PARC corporation, Trygve Reenskaug , started in 1979 what would become the birth of the MVC design pattern. THE original implementation was described in the article “Applications Programming in Smalltalk-80: How to use Model-View-Controller”. Reenskaug 's idea generated an application architecture pattern whose objective is to separate the project into three independent layers, which are the model, the view and the controller. This separation of layers helps reduce coupling and promotes increased cohesion in design classes. Thus, when the MVC model is used, it can greatly facilitate the maintenance of the code and its reuse in other projects.

Client/Server

For Sommerville , (2007, p.166) the client-server architecture model is a model in which the system is organized with a set of services and associated servers and clients that access and use the services. The main components of this model are:

1. A set of servers that provide services to other subsystems. Examples of these are printer servers that provide printing services, file servers that provide file management services, and a compiler server that provides programming language compilation services.
2. A set of clients requesting services offered by servers. These are typically independent subsystems. There can be multiple instances of a client program running concurrently.
3. A network that allows clients to access these services. This is not strictly necessary when both clients and servers can run on a single machine. In practice, however, most client-server systems are implemented as distributed systems.

According to Sommerville (2007 p. 16) the most important advantage of a client-server model is that it is a distributed architecture. Effective use of networked systems can be made with many distributed processors. It's easy to add a new server and integrate it with the rest of the system or transparently upgrade servers without affecting other parts of the system.

Life Cycle of the Used Software Process Model

Scrum Lifecycle

The main idea of SCRUM is that software development involves many technical and environmental variables, such as requirements, resources and technology, which can change during the process. This makes the development process unpredictable and complex, requiring flexibility to keep up with changes. The result of the process should be software that is really useful for the customer.

The SCRUM lifecycle is based on three main phases, divided into sub-phases :

1) Pre - planning (Pre-game phase): The requirements are described in a document called a backlog. Subsequently, they are prioritized and effort estimates are made for the development of each requirement. The planning also includes, among other activities, the definition of the development team, the tools to be used, the possible risks of the project and the training needs. Finally, a development architecture is proposed. Eventual changes in the requirements described in the backlog are identified, as well as their possible risks.

2) Development (game phase): the many previously identified technical and environmental variables are observed and controlled during development. Instead of considering these variables only at the beginning of the project, as in the case of traditional methodologies, in SCRUM control is carried out continuously, which increases flexibility to keep up with changes. In this phase, the software is developed in cycles (sprints) in which new functionalities are added. Each of these cycles is developed in the traditional way, that is, the analysis is carried out first, then the design, implementation and testing. Each of these cycles is designed to last from a week to a month.
3) Post-game phase : after the development phase, meetings are held to analyze the progress of the project and demonstrate the current software to customers. In this phase, the integration steps, final tests and documentation are carried out.

Requirements Gathering Methods and Techniques Used

Ethnography
Ethnography is an observation technique that can be used to understand social and organizational requirements, that is, to understand organizational politics as well as work culture in order to become familiar with the system and its history. Social scientists and anthropologists use observational techniques to develop a complete and detailed understanding of particular cultures.

Notations Used for Software Analysis and Design

Use Case Diagrams
For Leandro Ribeiro, and this diagram documents *what the system does from the user's point of view* . In other words, it describes the main functionalities of the system and the interaction of these functionalities with the users of the same system. In this diagram we do not delve into technical details that say *as the system does* .
This artifact is commonly derived from the requirements specification, which in turn is not part of the UML. It can also be used to create the requirements document.

Sequence Diagrams
According to Geraldo Magela, Sequence diagrams are very useful in providing real implementation support and in constituting rich high-level documentation. They represent the objects participating in a collaboration while issuing and receiving messages in order to realize a use case. The messages are presented in their temporal order, which makes it easier to understand the flow of

control of the use case. The greatest difficulty associated with their creation seems to be related to the degree of detail to be applied to these diagrams. But a practical perspective can be adopted for the creation of really useful documentation, and that does not become an even more arduous task than the coding of the system itself.

Class Diagrams

class diagram the representation of the structure and relationships of classes that serve as a model for objects.

a very useful modeling for the development of systems, because it defines all the classes that the system needs to have and it is the base for the construction of the diagrams of communication, sequence and states .

Mechanisms for Data Storage Used, Techniques and Levels of Applied Tests

According to Sommerville (2009, p. 7), the software must be validated to ensure that it does what the customer wants.

Like most engineering activities, building software depends primarily on the skill, interpretation and execution, of the people who build it; therefore, errors end up appearing, even with the use of software engineering methods and tools.

> So that such errors do not last, there are a series of activities, collectively called "validation, verification and testing", with the purpose of guaranteeing that both the way in which the software is being built and the product itself are in accordance with the specified . (Delamaro , Maldonado & Jino , 2009).

System Test: evaluates the software for flaws by using it as if it were an end user. In this way, the tests are executed in the same environments, with the same conditions and with the same input data that a user would use in his day-to-day manipulation of the software. Verifies that the product meets your requirements.

Unit Test: also known as unit tests. Its objective is to explore the smallest unit of the project, trying to cause failures caused by logic and implementation defects in each module, separately. The target universe of this type of test are object methods or even small code snippets.

Used programming paradigms

Object Oriented Programming

OOP was created to try to approximate the real world to the virtual world: the fundamental idea is to try to simulate the real world inside the computer. For this, nothing more natural than using Objects.

In OOP, the programmer is responsible for shaping the world of objects, and explaining to these objects how they should interact with each other. The objects "talk" to each other through the sending messages , and the main role of the programmer is to specify which messages each object can receive, and also what action that object must perform when receiving that specific message.

Frameworks and Architectures for WEB Development Used

Model - view - controller (MVC)

Civil Engineer Christopher Alexander created what is considered the first design pattern in the mid-1970s. A design pattern is considered to be a tested and documented solution that can solve a specific problem in different projects. Through Alexander's work, software development professionals used these concepts to initiate the first documentation of design patterns, making them accessible to the entire development area.

The then employee of the Xerox PARC corporation, Trygve Reenskaug , started in 1979 what would become the birth of the MVC design pattern. THE original implementation was described in the article “Applications Programming in Smalltalk-80: How to use Model-View-Controller”. Reenskaug 's idea generated an application architecture pattern whose objective is to separate the project into three independent layers, which are the model, the view and the controller. This separation of layers helps reduce coupling and promotes increased cohesion in design classes. Thus, when the MVC model is used, it can greatly facilitate the maintenance of the code and its reuse in other projects (Cui , Na & Zhang, 2021).

S ***truts***

One way to separate issues in a software application is to use a Model - View - Controller (MVC) architecture. O *model* represents the company or database code, the *Vision* represents the design code of the page, and the *controller* represents the navigation code. The Struts framework is designed to help developers create web applications that use the MVC architecture.

Hibernate

Hibernate is an ORM - ObjectRelationalMapping framework . It is a tool that helps us to persist Java objects in a relational database. The developer's job is to define how the objects are mapped in the database tables and Hibernate does all the access to the database, even generating the necessary SQL commands.

factory

is a design pattern that aims to encapsulate the creation of an object in a method. Factory is probably one of the most used patterns because it is so natural. It is often used without awareness that a pattern is being used.

The implementation using this design pattern directly affects the quality of the design, facilitating any future changes in the system. The company responsible for the business area is VLM Contabilidade, where an employee will be responsible for the specific information related to the areas in which the system will operate.

The purpose of this document is to collect, analyze and define the needs and functionalities of the software project that will be developed for VLM Contabilidade, focusing on the needs of those involved who will benefit from the software.

The objective of this document is to show in a way what will be developed and who are involved in the development.

Its scope includes the handling of data provided by the company accompanied by the person in charge of the financial department.

Scenario

Currently, the Company does not use any specific system to detail the fees provided, but an online system is used to generate the total value of the service.

There are similar systems on the market that serve accounting firms and self-employed accountants in general, such as the one described below:

Strong Accounting Fees

The Fortes Accounting Fees that was developed by Grupo Fortes Informática Ltda. **(** http://www.honorarioscontabeis.com.br/ index.php ?/ content /id/2) offers the user services for calculating and controlling the costs of services provided by accounting, advisory, auditing, expertise and similar companies.

Business Opportunity

The software will enable the company to better control information on users, clients and specification of fee services, enabling the extraction of important information through reports, in addition to adding adequate control to access information.

Project description

The scope of the project is the digital control of services, accounting fees provided to customers, user management system, issuance of detailed reports of services provided and/or access to the system

Conclusion

From the completion of this article, it can be concluded that the developed software project, despite the difficulties encountered throughout its development, met its initial purpose, which is the management of accounting fees.

The software proved to be very efficient with regard to the control of accounting fees. The user has an easy-to-understand interface where he can register the fees, make entries throughout the month and generate the monthly fee, so that the system, by itself, automatically searches and calculates all service entries, simplifying the calculation process of accounting fees that was done manually.

Among the main features of the developed software, one can mention the option available to the user to generate graphical reports with the average monthly accounting fees that a given customer paid in a certain period of time.

Finally, it is concluded that the proposed objective was achieved, as the software can facilitate the work of the accounting professional, since he will no longer add the fees manually, saving useful time.

References

Delamaro, ME, Maldonado, JC, & Jino , M. (Orgs .). (2007). *Introduction to Software Testing* . Elsevier.

Dias, BF, Barboza, LGS, Bertolini, GRF, & Vesco , DGD (2016). Intervention Applied to a Beauty Salon for Adequacy of Financial Aspects. *International Journal of Professional Business Review* , *1* (2), 76–90. https://doi.org/10.26668/businessreview/2016.v1i2.21

Higor , M. (2013). *Factory Design Pattern Method in Java* , available at <http://www.devmedia.com.br/padrao-de-projeto-factory-method-em-java/26348> Accessed on 04/23/2013.

Moraes, R. (2013). Article Software Engineering 2 - *Requirements gathering techniques*
. Available at < http://www.devmedia.com.br/artigo-engenharia-de-software-2-tecnicas-para-levantamento-de-responsabilidades/9151> Accessed on 04/20/2013.

Neto, ACD (2013). Article Software Engineering – *Introduction to Software Testing* – available at <http://www.devmedia.com.br/artigo-engenharia-de-software-introducao-a-teste-de-software/8035> Accessed 02/ 05/2013.

Portela, Cristiano RR (2013). *Requirements Analysis Concepts* – available at < http://www.paiossin.com/wordpress/wp-content/uploads/2011/11/Anlise-de-Requisitos-Conceitos.pdf> Accessed on 28/04/2013.

Prado, Alexandre A. (2013). *Fundamentals of Java Struts* , available at < http://www.devmedia.com.br/fundamentos-do-java-struts/7238> Accessed on 04/24/2013.

Protocoli , (2013). *Software Development Models* . Available at <http://protocoloti.blogspot.com.br/2012/03/os-modelos-de-desenvolvimento-de.html> Accessed on 04/21/2013.

Sommerville , I. (2007). *Ian Engenharia de Software trad* . Selma Shin Shimizu Melnikoff , Reginaldo Arakaki, Edílson de Andrade Barbosa -8. Ed. - São Paulo: Pearson Addison Wesley, 2007.

Sommerville , I. (2003). *Software Engineering/ Ian Sommerville* ; translation by André Mauricio de Andrade Ribeiro; kechi technical review Hirama.São Paulo: Addison Wesley, 2003.