

**Gestão de Trajes**

**Engenharia de Software II**

**Engenharia Informática**

**Trabalho realizado por:**

André Pedro Nº1011418

Índice

[Descrição do Tema 5](#_Toc530318992)

[Diagrama de Contexto 6](#_Toc530318993)

[Resumo QuittingTime 7](#_Toc530318994)

[SpiralDevelopment 8](#_Toc530318995)

[TwoTierReview 9](#_Toc530318996)

[Tabela de Atores e respetivos casos de uso 10](#_Toc530318997)

[Casos de Uso 11](#_Toc530318998)

[Descrição dos Casos de Uso 12](#_Toc530318999)

[Diagramas de Sequência 12](#_Toc530319000)

[Diagrama de Estados 12](#_Toc530319001)

[Diagrama de Classes 13](#_Toc530319002)

[Atividades e Tempos Gastos 14](#_Toc530319003)

[Dicionário de dados 14](#_Toc530319004)

[Diagrama de Componentes 14](#_Toc530319005)

[Diagrama de Pacotes 14](#_Toc530319006)

[Diagrama de Atividade 14](#_Toc530319007)

[Diagrama de Instalação 14](#_Toc530319008)

[Protótipo 14](#_Toc530319009)

[Conclusão 14](#_Toc530319010)

Descrição do Tema

Irei criar uma aplicação “Gestão de Trajes” de gestão de artigos numa loja de trajes tendo como objectivo ajudar na organização de compras e arranjos.

Esta aplicação ira permitir a criar fichas de clientes alterar artigos, criar vendas, criar arranjos entre outras.

A aplicação será feita em c# numa Windows form.

Diagrama de Contexto

O **DFD** (diagrama de fluxo de dados) de mais alto nível que representa todo o sistema como um único processo é conhecido como **diagrama de contexto**, e é composto por fluxos de dados que mostram as interfaces entre o sistema e as entidades externas. O diagrama é uma forma de representar o objeto do estudo, o projeto, e sua relação ao ambiente.

Um diagrama de contexto permite identificar os limites dos processos, as áreas envolvidas com o processo e os relacionamentos com outros processos e elementos externos à empresa (ex.: clientes, fornecedores) e mostra as características do sistema como podemos ver na figura seguinte o diagrama de contexto relativo á aplicação que está a ser desenvolvido.

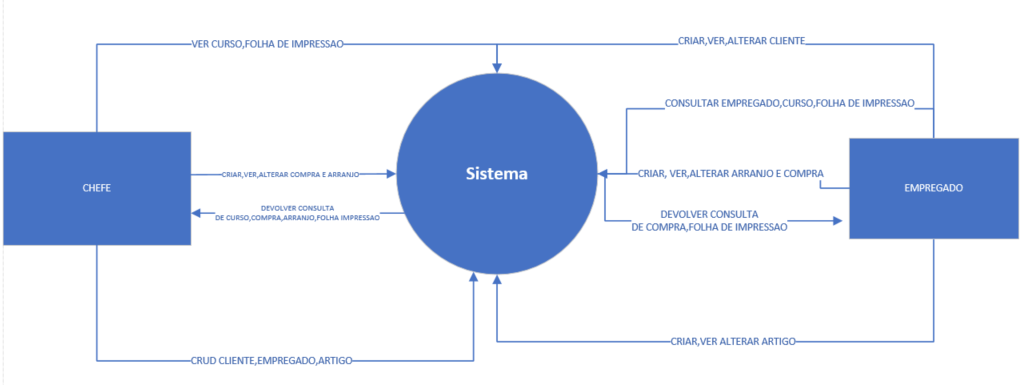


Figura – Diagrama de Contexto

Resumo QuittingTime

Escrever e criar casos de uso a mais do que o cliente aprova é um desperdício de tempo, recursos e ainda pode atrasar o projeto. Os requerimentos devem ser especializados para reduzir o risco de falhanço do projeto e devemos seguir esses procedimentos rigorosamente para correr tudo como o cliente quer e definiu. Contudo nem sempre este planeamento é tao linear como parece.

As pessoas, como são inexperientes, não sabem ao certo tudo o que querem. Ou seja, novas ideias estão sempre a vir ao de cima. Os casos de uso para serem bons tem de ser equilibrados e sobretudo, devem ter o essencial para o projeto funcionar, ou seja, poucos casos de uso é mau e ter muitos também é, dai ter de existir um meio-termo. Por vezes é complicado deixar de escrever casos de uso porque temos de alternar e não dificultar o processo porque por vezes temos de escolher entre uns casos e outros para não acrescentar demasiada informação, muita dela inútil. Existem casos em que um cliente, após a realização dos casos de uso pretendem mudar as coisas e adicionar ou retirar casos de uso. Isto é bastante complicado para o programador porque um projeto destes deve ser bem lineado e pensado cautelosamente. A palavra Formalidade é bastante importante na realização dos projetos para reduzir a ambiguidade de algumas situações. Estas situações ambíguas podem destruir um projeto visto que, como podem ter vários significados o cliente pode explicar e querer algo que o programador pode interpretar de maneira diferente e depois existe uma insatisfação das 2 partes. Parar de escrever casos de uso demasiado cedo é a maneira mais fácil e ingénua de adicionar ambiguidades a um projeto.

Nas empresas existem sempre pessoas com uma vasta experiencia o que ajuda a que não aconteçam este tipo de erros e ambiguidades. Requerimentos específicos e detalhados podem comprometer os requerimentos pré-propostos. No projeto um erro ao princípio pode ser muito melhor porque são mais fáceis de reparar do que for um mais tarde porque pode acontecer que tenhamos de começar tudo de novo o que é bastante mau para um projeto de um cliente e para o próprio programador. Não devemos adicionar casos de uso a mais do que aqueles que já foram aprovados pelo cliente. Estas são as 3 perguntas para que um cliente saiba se os casos de uso estão completos ou não.

* Foram identificados e documentados todos os objetivos e todos os atores?
* O cliente ou alguém que o represente reconheceu e aprovou todos os casos de uso antes o desenvolvimento do projeto?
* O designer pode implementar os casos de uso?

Se a resposta a estas perguntas for não ainda há trabalho para fazer. Caso seja sim então deve se avançar com o projeto.

Exemplo: Wings Over The World.

Ahmed defende que que os casos de uso não devem ser completos ate estar tudo bem detalhado e discutido. Ralph defende que não se deve perder tempo e avançar com o projeto sem quaisquer preocupações. Qual deles terá razão?

“Quando é que se sabe quando os casos de uso estão completos?” Esta é uma pergunta bastante difícil nesta área.

O QuittingTime equilibra o risco do atraso do projeto com o risco da falta de requerimentos.

Estes fatores são bastante comuns em qualquer empresa. Deve se ter sempre cuidado no que se faz e implementa. A comunicação entre cliente e programador deve ser bastante cautelosa para que não haja erros e para que não haja falsas interpretações.

SpiralDevelopment

Desenvolver casos de uso num único passo é difícil e pode tornar difícil a tarefa de incorporar novas informações neles, pode até atrasar a descoberta de fatores de risco.

Escrever casos de uso interactivamente fornece uma perspetiva de vista em que facilmente podemos recuar e rescrever ou riscar algo se descobrirmos que esta errado ou que não esta a funcionar corretamente. Podemos perder algum trabalho, mas será muito menor do que teríamos perdido se tivéssemos escrito os casos de uso todos de uma vez. Mais importante podemos identificar e confrontar potenciais problemas mais cedo usando um método de aproximação interativo.

Pode levar um longo tempo para perceber o comportamento de um sistema, e quanto mais complicado for, mais tempo ira demorar para se compreender. Atrasos custam recursos, e os requerimentos iniciais são fundamentais para o sucesso do produto, mas isto e só uma parte do projeto. Muitas pessoas podem ter deadlines que só são possíveis de cumprir se os casos de uso fornecidos forem bons.

Os requerimentos provavelmente irão mudar durante a sua análise, visto que os requerimentos são voláteis, e facilmente alterados quando sabemos mais deles. Um requerimento que parece concreto no início e necessário pode ficar inútil e desnecessário numa análise mais profunda. Todo o processo acaba por estabilizar, mas só depois de muita análise e reconstrução.

O custo de erros iniciais é alto, sendo que no inicio ate pode custar muito pouco reparar certos erros, mas a medida que o processo vai evoluindo o custo do erro e dificultando a sua deteção.

TwoTierReview

Muitas pessoas têm a necessidade de rever os casos de uso, mas isto e dispendioso, leva tempo.

As revisões são necessárias para validar, verificar e avaliar a sua plenitude na parte inicial do processo de desenvolvimento.

É difícil para os developers encontrarem os seus próprios erros pois fazem inferências que podem não estar lá e estão demasiados familiarizados com o trabalho. As pessoas que leem um documento pela primeira vez estão mais propícias a encontrarem erros porque estes não os deixam entender o propósito do documento.

Os stakeholders têm um interesse nos casos de uso, assim como cada grupo de dev's têm em determinados casos de uso pois dependem destes para os ajudar a fazer o trabalho deles. Por isso e no interesse de todos que estes sejam consultados o mais cedo possível e regularmente para que os casos de uso estejam o mais corretos possível e não sejam precisas alterações no futuro.

É demasiado caro, esgotante e lento envolver todos os elementos de um projeto nestas revisões e também é ineficiente pois elas vão acabar por "atropelar" umas as outras por isso a SmallWritingTeam é mais eficiente, mas como a equipa e pequena nem todos os interesses dos stakeholders são incorporados. É difícil se não impossível para uma equipa pequena representar a visão de todos os elementos do projeto. Este tipo de revisões deve ser usado com sensatez marcando-as estritamente quando necessário.

Assim sendo o twotierreview tem dois tipos de revisões o primeiro com uma equipa pequena interna e fá-las diversas vezes. A segunda pelo grupo completo pelo menos uma vez. Primeiro deve rever-se os casos de uso internamente para verifica se são fáceis de se entender, implementar e a sua precisão. Esta revisão pode ser informal, formal ou uma combinação das duas. Podem ser precisas varias revisões e vários para as diversas áreas assim cada stakeholder pode avaliar em profundidade os casos de uso com o seu ponto de vista. Por fim nas revisões internas as equipas assertaram que é QuittingTime e que esta tudo em ordem.

Tabela de Atores e respetivos casos de uso

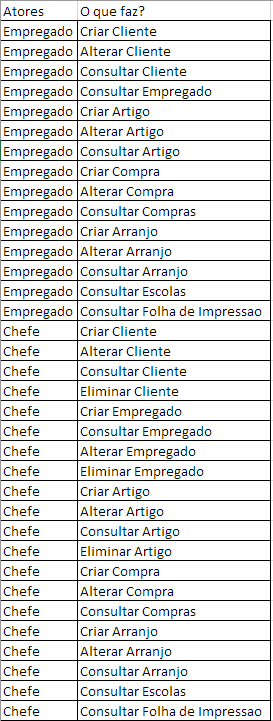


Figura - Tabela de Atores e Respetivos casos de uso

Casos de Uso

Figura - Diagrama de Casos de Uso

Descrição dos Casos de Uso

Diagramas de Sequência

Os diagramas de Sequência das ferramentas UML que representa as interações entre o utilizador e o sistema. Estes diagramas foram construídos a partir dos casos de uso:

Diagrama de Estados

Diagrama de Classes

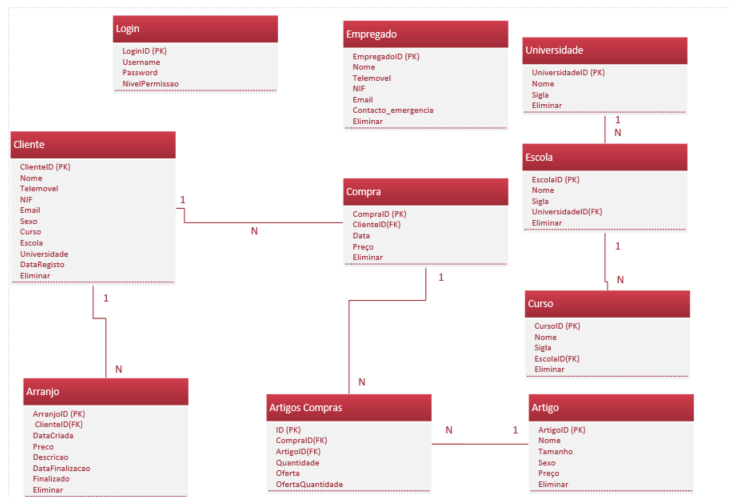
O diagrama de classes é composto pelas tabelas, os seus atributos e as ações que irem ser realizadas em cada tabela.  
 

Figura - Diagrama de Classes

Atividades e Tempos Gastos

Dicionário de dados

Diagrama de Componentes

Diagrama de Pacotes

Diagrama de Atividade

Diagrama de Instalação

Protótipo

Conclusão