Universidade Estadual de Goiás
Unidade Universitária de Itaberaí

Adão Aparecido de Oliveira
Marcos Suel Mendes Soares

Desenvolvimento de um sistema para gerência de matrículas dos alunos da Rede municipal de Ensino de Itaberaí - GO

Itaberaí
2011

Adão Aparecido de Oliveira
Marcos Suel Mendes Soares

Desenvolvimento de um sistema para gerência de matrículas dos alunos da Rede municipal de Ensino de Itaberaí - GO

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Sistemas de Informação da Universidade Estadual de Goiás - Unidade Universitária de Itaberaí, como requisito parcial, para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Profº Ms. Rogério Sousa e Silva

Itaberaí
2011

Adão Aparecido de Oliveira

Marcos Suel Mendes Soares

Desenvolvimento de um sistema para gerência de matrículas dos alunos da Rede municipal de Ensino de Itaberaí - GO

Aprovado em ___/___/____

Banca examinadora:

_____
Profª MS. Luciana Nishi

_____
Profº MS. Justino Porto

_____
Orientador: Profº MS. Rogério Sousa e Silva

RESUMO

A Secretaria Municipal de Educação (SME) de Itaberaí é um órgão do governo municipal que cuida da educação básica das escolas do município. Tem como objetivo elaborar a política educacional do município no que se refere à educação básica e coordenar sua implantação e obter resultados. E uma de suas funções é cuidar para que as escolas possam matricular os alunos de forma satisfatória. Ou seja, de forma rápida, sem erros e seguro para a população e escola. Um dos grandes problemas que as escolas enfrentam é ter que fazer matrícula de forma manual através de planilhas eletrônicas. O que gera muitos erros de cadastro e lentidão referente à "varredura" de arquivos manuais, prejudicando o atendimento da população da cidade, causando muitas perdas de informações de matrículas. Nesse sentido o desejo da SME é ter um software que possa atender esses problemas das escolas. E a solução é um sistema em que a população possa fazer solicitações de matrícula, e a partir daí as escolas possam matricular os alunos que estão nessas solicitações de matrícula. Avaliando essa situação o objetivo desse trabalho de conclusão de curso é fazer a análise desse software. Entretanto, não é o objetivo desse trabalho desenvolver o código fonte do software. A análise do software será feita com o uso de metodologia ágil juntamente com o Framework Scrum (Ver capitulo 1 no que se refere a modelo de processo de software).

Palavras-chave: Desenvolvimento ágil, Framework Scrum, Secretária Municipal de Educação, Matrículas on-line.

ABSTRACT

The Municipal Secretariat education (MSE) of Itaberaí is a local government agency that takes care of basic education schools in the municipality. It aims to develop the educational policy of the municipality with regard to basic education and coordinate its implementation and results. And one of its functions is to ensure that schools can enroll students in a satisfactory manner. That is, quickly, without errors and safe for the population and school.

One of the major problems that schools face is having to do registration manually using spreadsheets. What makes many registry errors and slowness in addressing the city's population and slow work on the "scan" the file manually. Causing much loss of information and enrollment slowdown in demand for them. In this sense the desire of the EMS is to have a software that can meet these problems in schools. And the solution is a system where people can make requests for registration, and thereafter for schools to enroll students who are these registration requests. Assessing this situation the objective of this work for conclusion of the course is to review this software, and this analysis can - is optionally build the product. But is not the goal of this work to develop the software source code. Software development will - with the use of Agile Scrum Framework together with the (See Chapter 1 in relation to software process model).

Keywords: Agile Development, Framework Scrum, Municipal Secretary of Education, Online enrollment.

SUMÁRIO

LISTA DE FIGURAS

INTRODUÇÃO

O propósito desse trabalho é fazer a análise de um software de gerenciamento de matrícula. Esse software tem o objetivo de atender Secretaria Municipal de Educação referente às escolas municipais do município de itaberaí – Goiás junto à população.

A análise desse software poderá proporcionar um produto em que a população do município poderá solicitar matrículas em uma escola. A partir dessas solicitações a escola poderá fazer a matrícula do futuro aluno que está em responsabilidade do solicitante.

O sistema proposto pela análise dará um software em que as escolas terão "energicamente" reduzidas a redundância, a lentidão de busca de matrículas e perda de informações.

Com objetivo de atender de forma rápida o órgão municipal, será usada a metodologia ágil juntamente com o framework de gerenciamento de software. Dessa forma tanto a análise quanto um possível desenvolvimento de código não serão prejudicados pelo excesso de artefatos que raramente ou nunca serão usados.

No primeiro capítulo serão abordados os conceitos da Engenharia de software para auxiliar no aspecto técnico e teórico na construção de software.

No segundo capítulo será visto o modelo de processo ágil bem como os passos para a análise do software para a possível construção do produto.

No terceiro capítulo serão vistos os conceitos de linguagem de modelagem UML, banco de dados e análise orientada a objetos. Esses assuntos proporcionarão base para construção dos diagramas de modelagem e diagrama entidade relacionamento (DER).

No quarto e último capítulo será trabalhado a forma como a análise irá propor que o software seja construído, entretanto, não é objetivo deste trabalho desenvolver o software.

1 Engenharia de Software

Segundo Sommerville (2007), construir software não é uma tarefa simples e corriqueira, por isso, é necessário desenvolver uma forma de trabalho organizada que mostre eficácia. Então cabe à engenharia de software mostrar boas formas de trabalho com os aspectos de processo de desenvolvimento de software que funcionem, aplicando conhecimento teórico, métodos e ferramentas da melhor forma possível, sempre buscando soluções para o possível problema. Levando em conta restrições organizacionais e financeiras.

Sommerville (2007, p. 5), "A engenharia de Software é uma disciplina de engenharia relacionada com todos os aspectos da produção de software, desde os estágios iniciais de especificação do sistema até sua manutenção, depois que entrar em operação".

Segundo Sommerville (2007), engenharia de software além de auxiliar nós aspectos técnicos do processo de desenvolvimento de software também contribuem para o gerenciamento de projetos e o desenvolvimento de ferramentas que contribui para o processo de software. A engenharia de software tem uma organização em seu trabalho, que foca na produção de software de alta qualidade e desempenho, procurando assim o método mais apropriado para cada caso.

Em seguida veremos alguns temas da engenharia de software como: Arquitetura cliente Servidor, Arquitetura MVC, Desenvolvimento Evolucionário, Engenharia de Software Baseado em Componentes, Processo de software. O processo de software que será o "ponta-pé" inicial para explicar o modelo de processo ágil e implementado pelo framework de gerenciamento de projetos Scrum utilizado nesse trabalho.

1.1 **Arquitetura Cliente Servidor**

A arquitetura cliente servidor é um conjunto de operação composto por aplicações de serviços fornecidos pelos servidores acessados por clientes através de terminais.

> O modelo de arquitetura cliente servidor é um modelo em que o sistema é organizado como um conjunto de serviços e servidores e clientes associados que acessam e usam os serviços. Um conjunto de servidores que oferecem serviços para outros subsistemas. Exemplos disso são servidores de impressoras que oferecem serviços de impressão, servidores de arquivos que

> oferecem serviços de gerenciamento de arquivos e um serviço de compiladores que oferece os serviços de compilação de linguagens de programação. Um conjunto de clientes que solicita os serviços oferecidos pelos servidores. Esses normalmente são subsistemas independentes. Pode haver várias instâncias de um programa cliente sendo executadas simultaneamente. Uma rede que permite aos clientes acessarem esses serviços. Isso não é estritamente necessário quando ambos, clientes e servidores, podem ser executados em uma única máquina. Na prática, contudo, a maioria dos sistemas cliente servidor é implementada como sistemas distribuídos. Sommerville (2007, p. 166).

A figura abaixo mostra a arquitetura de um sistema de acervo de filmes e fotografias:

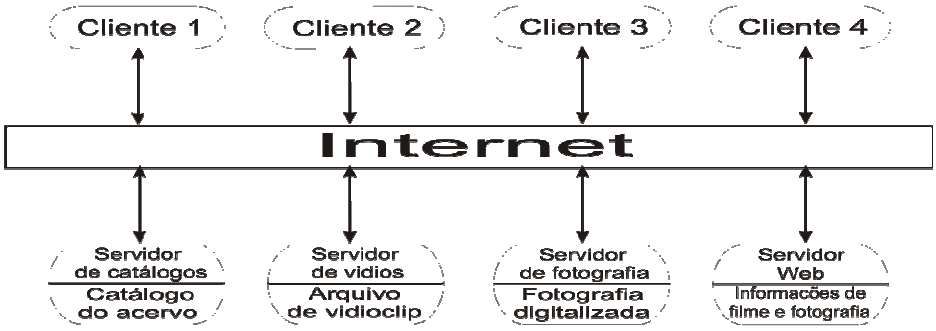


Figura 1: Arquitetura Cliente Servidor
Fonte: Sommerville (2007)

1.2 **Arquitetura MVC – Model, View, Controller**

Segundo Gamma, Erich et al (2000), a arquitetura MVC surgiu na década de 70. E após esses anos de sua criação ainda é um *pattern* (padrão) aplicável nas mais variadas aplicações, principalmente em aplicações web. Com o intuito de ser utilizado em projetos de interfaces visual, auxilia na tarefa de separar as responsabilidades promovendo um baixo acoplamento e alta coesão.

> MVC é um conceito (paradigma) de desenvolvimento e design que tenta separar uma aplicação em três partes distintas. Uma parte, a *Model*, esta relacionada ao trabalho atual que a aplicação administra outra parte a *View* esta relacionada a exibir os dados ou informações dessa uma aplicação e a terceira parte, *Controller*, em coordenar os dois anteriores exibindo a interface correta ou executando algum trabalho que a aplicação precisa completar. Edson Gonçalves (2007, p. 141).

> *Model*: O modelo (modelo) é objetivo que representa os dados do programa. Maneja esses dados e controlam todas suas transformações. Esse modelo não tem conhecimento específico dos controladores (*controller*) e das apresentações (*views*), nem sequer contém referências a eles, portanto, o *model* são as classes que trabalham no armazenamento e busca de dados. Por exemplo, um cliente pode ser modelado em uma aplicação, e pode haver vários modos de criar novos clientes ou mudar informações de um relativo cliente.
>
> *View*: A *view* (Apresentação) é o que maneja a apresentação visual dos dados representados pelo *Model*. Em resumo, é a responsável por apresentar os dados resultantes do *Model* ao usuário. Por exemplo, uma apresentação poderá ser um local administrativo onde os administradores se "*logam"* em uma aplicação. Cada administrador poderá visualizar uma parte do sistema que outro não vê.
>
> *Controller*: O *Controller* (Controlador) é o objetivo que responde as ordens executadas pelo usuário, a atuando sobre os dados apresentados pelo modelo, decidindo como o Modelo devera ser alterado ou devera ser revisto e qual Apresentação devera ser exibir uma lista de clientes interagindo com o Modelo e entregando uma Apresentação onde esta lista poderá ser exibida. Edson Gonçalves (2007, p. 386).

A arquitetura MVC oferece aos projetistas e desenvolvedores uma clara separação entre os processos, com a possibilidade de separar as funcionalidades do sistema em camadas possibilitando uma fácil manutenção do sistema. Dividindo as responsabilidades da equipe desenvolvedora.

1.3 **Processo de desenvolvimento iterativo**

> Processos de desenvolvimento rápido de software são projetados para criar software útil rapidamente. Geralmente, eles são processos interativos nos quais especificação, o projeto, o desenvolvimento e o teste são intercalados. O software não e desenvolvido e disponibilizado integralmente, mas em uma série de incrementos, e cada incremento inclui uma nova funcionalidade do sistema. O desenvolvimento incremental envolve a produção e a entrega de

software em incrementos, em vez de em um único pacote. Cada interação do processo produz um novo incremento do software. Entrega acelerada dos serviços ao cliente. Os incrementos iniciais do sistema podem fornecer uma funcionalidade de alta prioridade, de forma que os clientes logo poderão obter valores do sistema durante seu desenvolvimento. Os clientes podem ver seus requisitos e especificar mudanças para serem incorporadas nos *releases* posteriores do sistema. Sommerville (2007, p. 260).

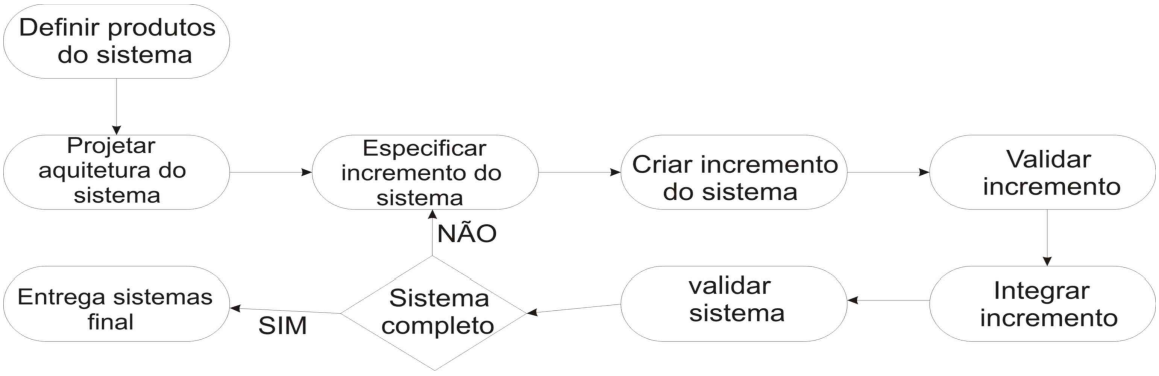


Figura 2: Processo de Desenvolvimento Iterativo
Fonte: Sommerville (2007, p. 160)

1.4 **Desenvolvimento Evolucionário**

O desenvolvimento evolucionário baseia-se na idéia de desenvolvimento de uma implementação inicial, expondo o resultado aos comentários do usuário e refinando esses resultados por meio de varias versões até que seja desenvolvido um sistema adequado. As atividades de especificação, desenvolvimento e validação são intercaladas, em vez de serem separadas, com *feedback* rápido que permeia as atividades. Uma abordagem evolucionária para desenvolvimento de software é frequentemente mais eficaz do que a abordagem em cascata na produção de sistemas que atendem às necessidades imediatas dos clientes. A vantagem de um processo de software baseado na abordagem evolucionária é que especificação pode ser desenvolvida de forma incremental. Á medida que o usuário compreende melhor seu problema, isso pode ser refletido no sistema de software. Sommerville (2007, p. 45).

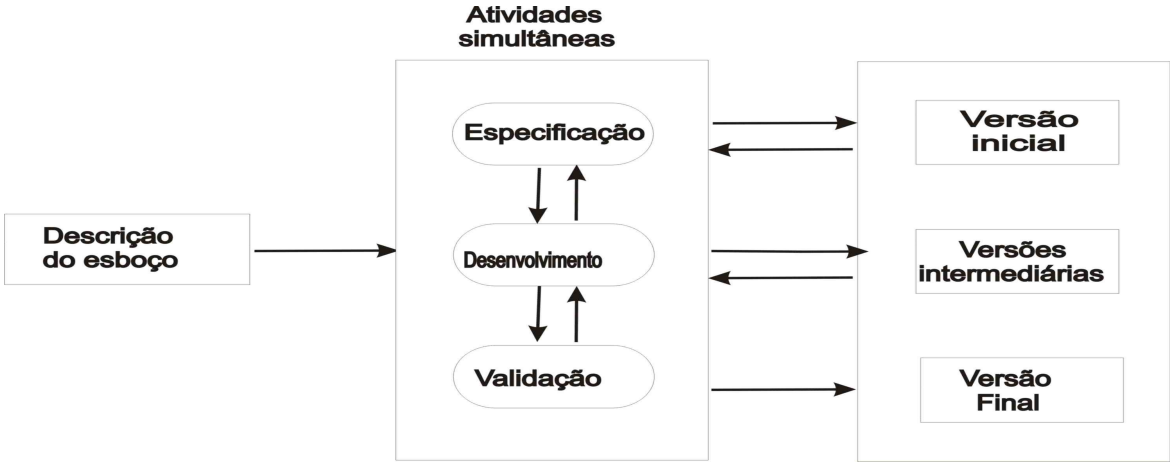


Figura 3: Desenvolvimento Evolucionário
Fonte: Sommerville (2007, p. 45)

1.5 **Engenharia de Software Baseado em Componentes**

Segundo Sommerville (2007, P. 46), vivemos em um mundo onde tudo ocorre muito rápido, e se tratando de software não é diferente, a cada dia o mercado necessita de sistemas que adeque as necessidades do mercado. Desta forma a engenharia de software baseada em reuso está sendo utilizada a cada dia em sistemas corporativos e comercias, a fim de proporcionar o desenvolvimento rápido do sistema. Isso geralmente ocorre de maneira informal, quando a equipe de desenvolvimento conhece códigos ou projetos similares aos necessários, eles modificam e os incorporam ao sistema. Na abordagem evolucionaria o reuso e essencialmente importante para o desenvolvimento rápido. O reuso informal ocorre independentemente do processo de desenvolvimento usado.

Sommerville (2007. P. 291), "A engenharia de software baseado em componentes CBSE (component based software enginnering), surgiu no final da década de 1990 com abordagem baseados no reuso para desenvolvimento de sistemas de software".

> A abordagem orientada a reuso depende de uma grande base de componentes de software reusáveis e algumas frameworks de interação desses componentes. Algumas vezes, esses componentes são sistemas comerciais independentes (COTS ou Commercial Off-The-Shelf System) que podendo fornecer funcionalidades especificas como a formatação de texto ou um calculo numérico. Sommerville (2007. P. 46).

Segundo Sommerville (2007. P. 292), prática do reuso tem se tornado cada vez mais difundido. Os clientes exigem sistemas confiáveis e rápidos, desta forma engenharia de software baseado em componentes tem a vantagem obvia de reduzir a quantidade de software a ser desenvolvido, reduzindo assim os custos. Mas é preciso cautela, compromissos com os requisitos são inevitáveis e é preciso combinar com um processo de software adequado a cada caso de desenvolvimento. Processo que será visto em seguida. A figura 1.5 mostra o lugar em que fica a Análise de componentes.

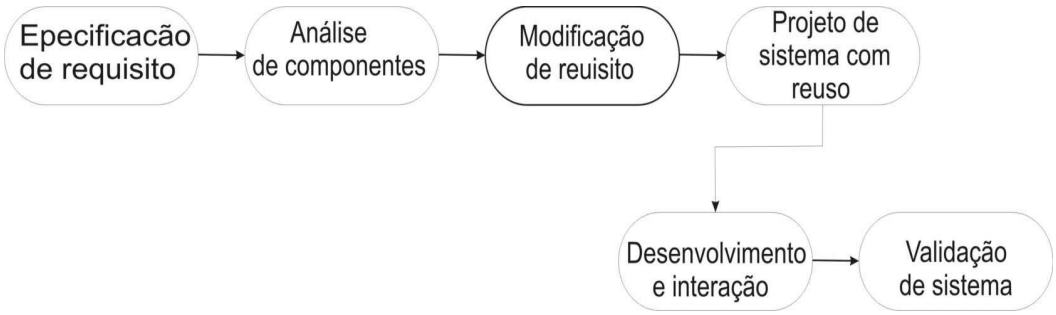


Figura 4: Engenharia de Software Baseado em Componentes
Fonte: Sommerville (2007, p. 45)

1.6 **Processo de software**

Segundo Sommerville (2007, p.43), o processo de software é um conjunto de elementos resultantes de um produto final ou uma representação abstrata de um processo que damos o nome de software. Modelos de processo de softwares genérico (paradigma de processo).

> Esses modelos genéricos não são descrições definitivas do processo de software. Ao contrário, são abstrações do processo que podem ser usada para explicar diferentes abordagens para desenvolvimento o desenvolvimento de software. Eles Podem ser considerados como frameworks de processo que podem ser ampliadas e adaptadas para criar processos mais específicos de engenharia de software. Sommerville (2007, p.43).

Segundo Sommerville (2007, p.43) o processo de software é complexo cabe à equipe de desenvolvimento escolher qual o melhor para cada caso. Cabe lembrar que não existe um processo ideal, varias organizações desenvolveram abordagem diferente para seu

desenvolvimento de software. Com o decorrer do tempo os processos evoluíram para explorar a capacidade dos envolvidos e das características especificas do sistema.

Temos diversos processos de softwares, mas algumas atividades são comuns entrem elas segundo sommerville:

> Especificação de software: A funcionalidade do software e as restrições devem ser definidas.
> Projeto e implementação de software: Deve ser produzido o software que atenda as especificações.
> Validação de software: o software dever ser validado junto ao cliente para ter certeza que é aquilo que o cliente necessita.
> Evolução do software: o software é flexível para atender as mudanças dos requisitos e exigências do mercado. Sommerville (2007, p.6).

Sommerville (2007 p. 6), "Diferentes tipos de sistemas necessitam de diferentes tipos de desenvolvimento. Alguns processos devem ser especificados antes do inicio de seu desenvolvimento. O uso do processo de software inadequado pode reduzir a qualidade".

Um modelo de processo de desenvolvimento de software é uma visão simplificada das atividades do processo. Segundo Sommerville:

> Modelo de *Workflow*: mostra a seqüência de passos ao decorrer do processo, entrada, saída e seu grau de dependência, essas necessidades retratam as ações humanas. Modelo de fluxo de dados ou modelo de atividades: cada atividade compõe passos do processo e transformações de dados. Retrata como a entrada do processo é transformada em uma saída, como um projeto.
> Modelo de papel/ação: mostra os papéis das pessoas envolvidas no processo em desenvolvimento. Sommerville **(**2007**)**

1.7 Modelo de Processo Ágil

1.8 **Manifesto Ágil**

Construir software não é uma tarefa trivial, exige organização para que se possam atingir os objetivos de forma satisfatória. E uma forma boa de organização é seguir uma gestão de projeto em que todos os envolvidos possam ter bons resultados. E visando alcançar satisfação e bons resultados, optou – se nesse projeto pelo uso das práticas e princípios ágeis.

Em fevereiro de 2001 dezessete conhecidos engenheiros de softwares entre eles Kent Beck, se reuniram em Utah EUA, nesse encontro discutiram como encontrar melhores maneiras para desenvolver software e então foi definido o manifesto ágil, através desse manifesto visaram alguns valores:

1 **Indivíduos e interação entre eles** mais que processos e ferramentas

2 **Software em funcionamento** mais que documentação abrangente

3 **Colaboração com o cliente** mais que negociação de contratos

4 **Responder a mudanças** mais que seguir um plano

> O movimento Agile não é anti-metodológico, de fato, muitos de nós queremos restaurar a credibilidade da palavra metodologia. Queremos restabelecer o equilíbrio. Nós abraçamos a modelagem, mas não de forma a apresentar alguns diagramas em um repositório corporativo empoeirado. Nós abraçamos a documentação, mas não centenas de páginas mantidas e raramente usadas ou nunca de fato. Nós planejamos, mas reconhecemos os limites do planejamento em um ambiente turbulento. Jim Highsmith (2001)

1.9 **Princípios Seguidos**

De acordo com Jim Highsmith (2001), no manifesto Ágil feito em 1990 alguns princípios mostram o que pretendem:

"Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado." Entregar software de forma continuada e com valor agregado é excelente, pois, diminuir muito os riscos de dar errado o projeto. A cada entrega o

cliente pode avaliar o que está a usar. Dessa avaliação de uso do sistema pode verificar pela equipe de desenvolvimento qualquer inconsistência com o solicitado pelo cliente.

"Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento." São as mudanças nos requisitos que moldam o desenvolvimento do software. Essas mudanças que darão à direção correta a seguir pela equipe de desenvolvimento, afetando positivamente na qualidade do produto.

"Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente." O desenvolvimento de software tem que estar além de produzir somente o esperado pelo cliente. Ele deve ir além mudar o sistema de uma forma que dê uma vantagem competitiva sobre os concorrentes. Melhorando dessa forma a visão estratégica da organização.

"Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo." De nada adiantaria fazer entregas se essas não funcionassem no ambiente do cliente. Um produto que funcione pode ser analisado por quem usa. Dessa forma erros podem ser encontrados e corrigido por quem o constrói.

"Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto." A idéia de que os desenvolvedores devem estar separados do resto da equipe já está ultrapassada. A equipe é formada por varias facetas diferentes com cada um tendo conhecimento do que deve ser feito para o projeto. Podendo ambos se auxiliarem.

"Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho." É importante não só investir em tecnologia, mas, em especial nas pessoas. São delas que irá sair às idéias necessárias para o desenvolvimento do produto. É então fundamental para elas terem um ambiente agradável e motivador para trabalhar. Evitando que a falta de motivação acarrete negativamente na qualidade do que se desenvolve.

"O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face." É imprescindível para um bom trabalho a comunicação. É através dela que a equipe saberá o que tem ser feito. E a conversa face a face faz com que a informação seja passada de acordo com o que o emissor quis passar para o receptor.

"Software funcionando é a medida primária de progresso." Software funcionando é onde se começa a medir se houve progresso no que está desenvolvendo. É a partir daí que se pode verificar se o que o cliente pediu está consistente com seus desejos de negocio.

Royce (1970) descreve, “Algumas metodologias ditas tradicionais surgiram em um contexto muito diferente dos dias atuais, pois o que existia no passado eram mainframes e terminais burros”. Essas metodologias têm um grande foco no processo, na documentação gerada e na antecipação dos problemas. O que pode tornar – se desgastante e oneroso, pois, alguma mudança pode desencadear alterações em todos os documentos, e levando em consideração que ocorrerá muitas mudanças pode piorar muito mais a situação. Problemas como estes certamente afetaram o escopo do projeto, o valor, a motivação da equipe e a satisfação do cliente que é o maior interessado no produto. E para piorar a situação essas metodologias como o RUP (Rational Unified Process) da IBM, são consideradas por serem extensas e meticulosas difíceis e demoradas para que se possa aprender.

Nesse contexto ser ágil é entregar um produto com qualidade e que funcione, de forma rápida em comparação a filosofias de desenvolvimento seculares. Em seguida será visto uso da metodologia ágil através do framework de gestão de projetos Scrum.

1.10 **Framework Scrum**

Optou-se nesse projeto pelo o uso do SCRUM para a gestão e desenvolvimento. Alguns dos motivos pela escolha foram:

Evita criar documentos desnecessários que nunca ou raramente serão usados;

Usa somente como artefato documentos que realmente são necessários;

Documentação abrangente;

Permite ser combinado com outros *frameworks* ou metodologias;

Equipe pequena;

Requisitos instáveis por se tratar de um órgão municipal que está diretamente subjugado as leis vigentes que vigoram sobre a educação;

Iteração curta, o que gera maior visibilidade do problema, maior feedback do cliente e maior satisfação para o cliente;

Time auto gerenciável.

Criado em 1993 por Jeff Sutherland em extensão a um trabalho realizado por Takeuchi e Nonaka em 1986 publicada na Harvard Business Review.

Scrum não é uma metodologia de desenvolvimento, mas sim um framework que tanto serve para gestão de projetos de software quanto a outros produtos. Esse documento mostrará como se dará no contexto de desenvolvimento do software.

Abaixo será mostrado como o SCRUM encara a gestão e o desenvolvimento de projeto:

> Ambos *Scrum* e *Extreme Programming* (XP) prega que a equipe complete alguma parte palpável de trabalho que seja entregável ao fim década iteração. Essas iterações são pensadas para serem curtas e com espaço de tempo definido. Este foco em entregas de código funcionando em um curto espaço de tempo significa que equipes *Scrum* e XP não têm tempo para teorias. Eles não perseguem o desenho do diagrama UML perfeito em ferramentas case, a escrita do documento de requisitos perfeito, ou a escrita do código que poderá acomodar todas as mudanças futuras imagináveis. Ao invés disso, equipes *Scrum* e XP focam em ter as coisas prontas. Essas equipes aceitam que cometem erros ao longo do caminho, mas também compreendem que o melhor modo de identificar esses erros é parar de pensar sobre o software em um nível teórico de análise e design e mergulhar fundo, sujar as mãos e começar a construir o produto. Henrik Kniberg (2007)

Esse framework não visa ensinar como fazer as coisas, mas mostra o que tem que ser feito para que se possam alcançar os resultados pretendidos de forma satisfatória tanto para quem faz o produto quanto para os clientes.

O SCRUM "diz" que iterações (são chamadas de *Sprint* no SCRUM) precisam ser de duas semanas a um mês, e que a cada fim de iteração possa ser entregue incrementos de produto funcional. Para satisfazer essas entregas o Scrum necessita de equipes que sejam auto-organizadas, ou seja, elas que irão decidir o que darão conta de entregar na data que foi decidida. Também depende do cliente que terá a função de transmitir ao "dono do produto" ou *Product Owner* a lista de funcionalidades que o produto terá e as prioridades.

O ciclo de vida do SCRUM acontece da seguinte maneira: o *Product Backlog* é obtido pelo *Product Owner* (dono do produto), logo após o time seleciona as funcionalidades que dão origem ao *Sprint Backlog* em que conseguirão desenvolver em duas a quatro semanas e no final desse ciclo é obtido um incremento de produto potencialmente utilizável.

A imagem abaixo mostra como funciona o ciclo de vida do SCRUM:

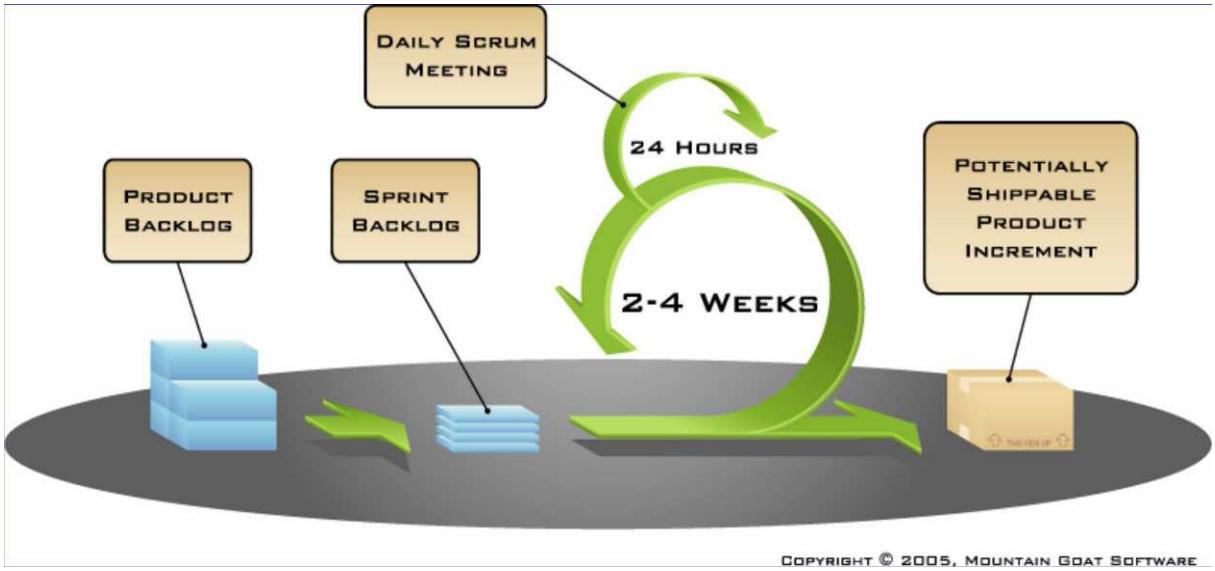


Figura 5: Ciclo de vida do SCRUM
Fonte: Mike Cohn (2011)

1.10.1 **Papéis do Scrum**

Os papéis representam quem de fato executa o SCRUM. Os papéis do SCRUM são: Product Owner, Scrum Master e Scrum Team. Esses que vão construir o projeto de acordo com as regras definidas pelo framework.

1.10.2 **Product Owner**

É o responsável pelas funcionalidades do produto. É ele que intermediará o cliente e as pessoas que desenvolverão o produto. Deve estar sempre disponível para a equipe, pois, a qualquer momento dúvidas podem surgir. Suas principais funções serão:

- Coleta dos requisitos através de histórias escritas pelos clientes.
- Determinar as prioridades sobre as histórias
- Determinar as datas de release
- Rentabilidade (ROI)

- Aceitação ou rejeição dos resultados de cada trabalho

> O Product Owner é uma pessoa, e não um comitê. Podem existir comitês que aconselhem ou influenciem essa pessoa, mas quem quiser mudar a prioridade de um item, terá que convencer o Product Owner. Empresas que adotam Scrum podem perceber que isso influencia seus métodos para definir prioridades e requisitos ao longo do tempo. Ken Schwaber (2009)

O Product Owner se parece com o gerente de projetos das metodologias ditas tradicionais. O fato determinante nessa comparação é que ele não irá definir para a equipe quais funcionalidades elas devem fazer, fica a cargo da equipe definir o quanto elas conseguem terminar em cada iteração.

> Para que o *Product Owner* obtenha sucesso, todos na organização precisam respeitar suas decisões. Ninguém tem a permissão de dizer ao Time para trabalhar em outro conjunto de prioridades, e os Times não podem dar ouvidos a ninguém que diga o contrário. As decisões do Product Owner são visíveis no conteúdo e na priorização do Backlog do Produto. Essa visibilidade requer que o Product Owner faça seu melhor, o que faz o papel de Product Owner exigente e recompensador ao mesmo tempo. Ken Schwaber (2009)

Nesse projeto quem fará o papel de Product Owner será o Gerente de TI da Secretaria Municipal de Educação. Essa escolha é viável, pois, é quem conhece em muito o funcionamento da intuição. Sendo assim será de grande intermediação entre a equipe de análise e a SME.

1.10.3 **Scrum Team**

É a equipe de desenvolvimento que irá construir o produto. Necessitam muito do Product Owner, pois, de acordo com as prioridades das histórias, irão calcular quais dessas tarefas poderão realizar em uma iteração. Por essa característica de poderem determinar o que irão realizar diz se que o SCRUM TEAM é auto-organizável. Isso é uma excelente

característica, pois, não se sobrecarregarão. Em metodologias seculares as ágeis, a equipe de desenvolvimento tem o gerente que diz o que eles têm que fazer e em quanto tempo.

> Membros do Time frequentemente possuem conhecimentos especializados, como programação, controle de qualidade, análise de negócios, arquitetura, projeto de interface de usuário ou projeto de banco de dados. No entanto, os conhecimentos que os membros do Time devem compartilhar - isto é, a habilidade de pegar um requisito e transformá-lo em um produto utilizável - tende a ser mais importantes do que aqueles que eles não dividem. As pessoas que se recusam a programar porque são arquitetas ou designers não se adaptam bem a Times. Todos contribuem, mesmo que isso exija aprender novas habilidades ou lembrar-se de antigas. Não há títulos em Times, e não há exceções a essa regra. Os Times também não contém subtimes dedicados a áreas particulares como testes ou análise de negócios. Ken Schwaber (2009)

1.10.4 **Scrum Master**

É responsável por proteger o time contra adversidades como, por exemplo: não permitir que o time se comprometa com tarefas que não conseguirão entregar. Também visa garantir que as praticas do Scrum sejam cumpridas. Deve – se tomar cuidado para não pensar que o Scrum Master é um gerente de projetos, pois, ele não delega tarefas, apenas monitora para que se possam alcançar os resultados esperados da sprint, ficando a cardo do time as atribuições de tarefas.

As responsabilidades do Scrum Master acontecem em 3 reuniões: Scrum Daily Meeting, Sprint Review e Retrospective. Essas reuniões serão explicadas posteriormente.

1.10.5 **Atividades do Scrum**

Para a gerencia do Scrum são usadas as seguintes atividades: Sprint, Reunião de Planejamento da Sprint, Reunião de Planejamento da Sprint, Revisão da Sprint, Retrospectiva da Sprint, Priorizando o Backlog do Produto, Estimando o Backlog do Produto e Reunião Diária.

1.10.6 **Sprint**

Desenvolver projeto torna complexo demais se ele for construído de uma vez e como um todo. Por isso nesse projeto dividiremos o projeto em Sprint's. Uma Sprint é uma iteração de duas semanas a um mês. Ela terá as funcionalidades que o time decidir que consegue terminar.

1.10.7 **Revisão da Sprint**

Ao final de cada Sprint o Scrum espera que tenha um produto potencialmente executável. E para que ele seja avaliado é realizada essa última reunião ao final de cada iteração. Nessa reunião será apresentado o que foi feito e algumas novas funcionalidades. O produto será validado de acordo com o que foi planejado no início.

Para o bom funcionamento da reunião, existem algumas regras que não permitem apresentações em Slides, o tempo de preparação para reunião não pode exceder 2 horas. Dessa forma o time não perderá o foco no trabalho e não servirá como um desvio para o time se desprender do trabalho.

1.10.8 **Retrospectiva da Sprint**

O desenvolvimento de software não é uma tarefa fácil, é complexo, por isso, é preciso analisar o que está sendo feito e o que será feito posteriormente. Essa reunião acontece ao final de cada Sprint. Tem por objetivo identificar o que cada membro do time fez, se gostou ou não gostou e o que será feito de diferente para a próxima Sprint.

Consegue ao final uma visibilidade do que precisa ser feito para melhorar o trabalho das Sprint's futuras.

1.10.9 **Reunião Diária**

É uma reunião que deve ser feita todos os dias, durante 15 minutos, não mais do que isto, especialmente de manha. De manha porque é nesse tempo que começa surgir os problemas a serem resolvidos. Essa reunião visa mostrar o andamento do projeto. Deve – se responder a perguntas como: "O que você fez ontem?", "O que você vai fazer hoje?", "Existe

algum impedimento?”. Com essas perguntas todos ficam cientes do que foi feito e do que será feito e o que se espera que termine.

Nessa reunião outros envolvidos no projeto podem ter participação, entretanto, somente o time pode se manifestar.

1.10.10 Estimando o Backlog do Produto

Com o passar do tempo o time ficará mais experiente e então poderão ajudar o Product Owner a tomar decisões sobre as funcionalidades. O time dará pontos às funcionalidades e através disso o Product Owner pode entender e encontrar as prioridades para novas funcionalidades.

1.10.11 Priorizando o Backlog do Produto

Através dos novos incrementos solicitados pelo cliente, o Product Owner irá balancear os custos e através disso delimitará as prioridades para cada funcionalidade que será desenvolvida pelo time.

Nesse projeto função do produto terá uma prioridade determinada pelas letras (A até E) sendo “A” o maior valor. Serão definidas pelo Product Owner quais serão as prioridades. Os itens de maior valor de prioridade serão os que terão maior urgência na sua realização.

1.10.12 Artefatos do Scrum

Os artefatos são os documentos obtidos durante reuniões com o cliente ou outros envolvidos com o projeto. Ele guiará durante todo o projeto o desenvolvimento do produto. Esses artefatos são: Product Backlog, Sprint Backlog, Incremento de Produto Potencialmente Utilizável e Quadro de Tarefas.

1.10.12.1 Product Backlog

Nesse documento serão colocadas pelo Product Owner as funcionalidades que os clientes necessitam. São listadas através de historias de usuários, ou seja, os desejos do cliente na linguagem dos usuários.

O Scrum assume o fato de que erros acontecem e que as necessidades para o produto são conhecidas superficialmente. Isso quer dizer que para esse framework as mudanças são bem vindas, pois, são elas que irão moldar o software de acordo com o desejo do cliente. Com esse conhecimento superficial o Product Owner poderá dar as prioridades para essas funcionalidades. Abaixo é mostrado um exemplo do product backlog:

PRODUCT BACKLOG (exemplo)					
ID	**Nome**	**Imp**	**Est**	**Como demonstrar**	**Notas**
1	Depósito	30	5	Logar-se, abrir a página de depósito, depositar R$ 10,00, ir para a página do meu saldo e verificar que este aumentou em R$ 10,00.	Precisa de uma diagrama UML de sequência. Não é necessário se preocupar com criptografia por enquanto.
2	Verificar seu próprio histórico de transações	10	8	Logar-se, clicar em "transações". Fazer um depósito. Voltar para transações, verificar se o novo depósito é listado.	Usar paginação para evitar consultas muito grandes ao banco de dados. Projetar de forma similar à página de visualização de usuários.

Figura 6: Product Backlog
Fonte: Henrik Kniberg (2007, p. 20)

Nesse projeto como mostrado na figura 1.7 o Product Backlog terá em sua composição, a identificação única (ID) da história de usuário, a prioridade, a descrição da historia, a pontuação data pelo time e notas.

1.10.12.2 **Sprint Backlog**

O Sprint Backlog terá as funcionalidades que o time definirá no Product Backlog de acordo com as prioridades e com a estimativa que eles acham que poderão entregar na próxima Sprint. Pode ser feito em planilhas ou em softwares especializados em *agile*.

1.10.13 **Incremento de Produto Potencialmente Utilizável**

Em cada Sprint é necessário que seja entregue um incremento que possa ser utilizável pelo cliente. Este produto precisa ser testado e documentado, com um manual de ajuda ao usuário.

Pode surgir a pergunta sobre o que é um produto utilizável. Essa questão deve ser debatida e respondida pelo time.

No próximo capítulo será visto o paradigma de análise, linguagem de Modelagem e Banco de dados.

2 Ferramentas essenciais para a análise

Na construção de qualquer produto é imprescindível que seja visualizado sobre diferentes perspectivas e complexidade. Na análise de software também não é nada diferente. Para ter essas "visualizações" algumas coisas são essenciais como: Uma linguagem de Modelagem, um paradigma de análise e a forma de armazenamento. A seguir tais conceitos de: Orientação a objetos, UML e banco de dados.

2.1 AOO – Análise Orientada a Objeto

O desenvolvimento orientado a objeto tem o objetivo de aproximar o mundo computacional do real. Baseia-se em conceitos que começamos a aprender no jardim-de-infância: objetos e atributos, classes e membros, o todo e suas partes. Objeto: seria a representação computacional dos objetos no mundo real. Um objeto seria constituído de: um nome, seus atributos e suas operações, tudo que e manipulável que possa ser perceptível. Classe: simplificando seria um conjunto de características comuns a vários objetos de onde estes mesmos objetos são derivados.

Segundo Bezerra (2007, p.4), o conceito de orientação a objetos surgiu com o intuito de minimizar os problemas encontrados até então na criação de softwares complexos. Com o foco na análise OO é no mapeamento de uma solução sistêmica para algum processo de negócio. Efetuar uma boa análise OO na criação de um sistema é fundamental para o seu sucesso. Por incrível que pareça, muita gente ainda enxerga a análise OO apenas como uma forma de documentar um sistema. Sim, esse é um dos ganhos que se tem com ela. Afinal, com seu uso, o conhecimento sobre o sistema não fica preso apenas na cabeça de quem o construiu, como também não fica submerso em meio a milhares de linhas de código fonte.

> O paradigma da orientação a objeto visualiza um sistema de software como uma coleção de agentes interconectados chamados objetos. Cada objeto é responsável por realizar tarefas especificas. É pela interação entre objetos que uma tarefa computacional é realizada. Bezerra (2007, p. 6).

Bezerra (2007, p. 7), classe “é uma descrição dos atributos e serviços comuns a um grupo de objetos, pode se entender uma classe como um molde a partir do qual objetos são construídos”. Representa um conjunto de objetos com características Similares ou comuns.

Figura 7: Classe com Métodos
Fonte: Bezerra (2007, p. 7)

> **Objeto** - o mundo real é formado de coisas. Na terminologia de orientação a objetos, essas coisas do mundo real são denominadas objetos. Uma unidade autônoma que contém seus próprios dados que são manipulados pelos processos definidos para o objeto e que interage com outros objetos para alcançar um objetivo. Bezerra (2007, p. 6, 7).
>
> **Encapsulamento** - Objeto possuiu comportamentos. O termo comportamento diz respeito a operações realizadas por um objeto, conforme este objeto recebe mensagens. O mecanismo de encapsulamento é uma forma de restringir o acesso ao comportamento interno de um objeto Bezerra (2007, p.9).

Bezerra (2007, p 11), **Generalização/Herança** “as características e o comportamento comuns a um conjunto de objetos podem ser abstraídos em uma classe”. Herda de alguém alguma coisa ou características.

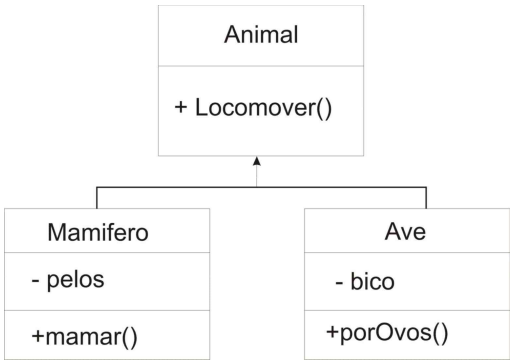


Figura 8: Exemplo Herança
Fonte: Bezerra (2007, p.11)

Bezerra (2007, p. 11) **Polimorfismo**, **“**indica capacidade de abstrair várias implementações diferentes em uma única interface”.

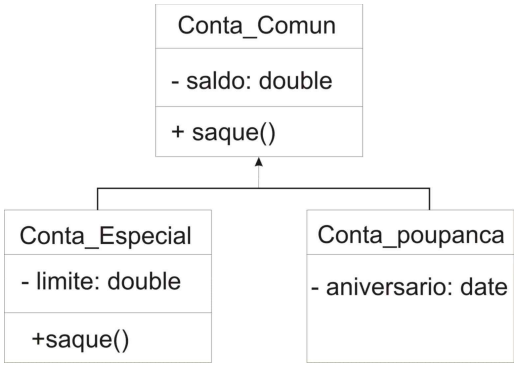


Figura 9: Polimorfismo
Fonte: Bezerra (2007, p.11)

2.2 UML

A UML - Linguagem Unificada de Modelagem é uma linguagem para especificar, visualizar, construir e documentar os artefatos de sistemas de software, bem como para modelar negócios e outros sistemas que não sejam de software. A primeira linguagem orientada a objeto é geralmente reconhecida com sendo a Simula-67, desenvolvida por Dahl e Nygaard, na Noruega, em 1967. Apesar dessa linguagem não ter tido muitos adeptos, seus conceitos foram muito importante para a consolidação e o amadurecimento de idéias de possíveis linguagens. No decorrer da década de 1990 um conjunto de idéias começou a se formar, quando Grand Booch (Rational Software Corporation), Ivan Jacobson (Objectory) e James Rumbaugh (General Electrics) começaram estabelecer *metadados* de idéias que vinha sendo reconhecidos mundialmente como os principais métodos orientados a objetos. Booch, Rumbaugh e Jacobson (2005).

2.2.1 Diagrama de Caso de Uso

Segundo Bezerra (2007, p. 53), um caso de uso é uma forma de descrição da linguagem natural entre o ator e o sistema que facilita o entendimento externo e o refinamento dos requisitos funcionais do sistema e tem o objetivo de ilustrar em alto nível de abstração os elementos que interagem com a funcionalidade do sistema. Mostra um relato de certas funcionalidades sem revelar a estrutura e o comportamento de seu funcionamento interno. Ao

observar o caso de uso é possível identificar as funcionalidades do sistema e seus resultados externos produzidos por ele.

> O diagrama de casos de uso é o diagrama mais geral e informal da UML, utilizado normalmente nas fazes de levantamento de requisitos do sistema, embora venha a ser consultado durante todo o processo de modelagem e possa servir de base para outros diagramas. Apresenta uma linguagem simples e de fácil compreensão para que os usuários possam ter uma idéia geral de como o sistema irá se comportar. Procura identificar os atores (usuários, outros sistemas ou até mesmo algum hardware especial) que utilizarão de alguma forma o software, bem como os serviços, ou seja, as funcionalidades que o sistema disponibilizará aos atores, conhecidas nesse diagrama como caso de uso. Gilleanes T. A. Guedes (2009, p. 31).

Dispõe de uma notação gráfica simples e de fácil entendimento entre clientes e especialistas. E composto por diversos componentes: caso de uso, atores e relacionamentos. A quantidade de caso de uso e relativo com a complexidade do negócio. A ilustração de um caso de uso e composto por ator que nem sempre representa um ser humano, que é composto por um boneco. Caso de uso que e representado por uma elipse cujo nome e posicionado abaixo ou dentro. Relacionamento de comunicação é representado por uma reta ligando ator e caso de uso. Esses três elementos são os mais utilizados.

> Inclusão é utilizada quando dois ou mais casos de uso incluem uma seqüência comum de interações, essa seqüência comum pode ser descrita em outro caso de uso. A partir daí, vários casos de uso do sistema podem incluir o comportamento desse caso de uso comum. Bezerra (2007, p. 62).

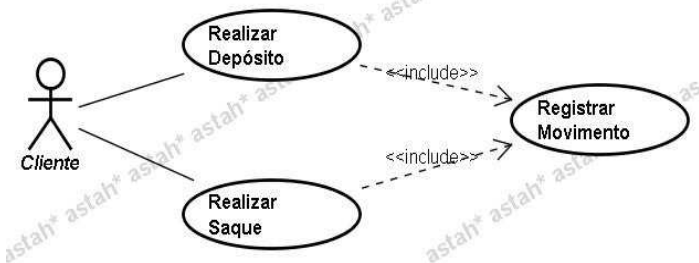


Figura 10: Include
Fonte: Bezerra (2007, P. 62)

> Extensão é utilizada para modelar situações em que diferentes seqüências de interações podem ser inseridas em um mesmo caso de uso. Cada uma dessas diferentes seqüências representa um comportamento eventual, um comportamento que só ocorre sob certas condições. Bezerra (2007, p. 63).

Figura 11: Extensão
Fonte: Bezerra (2007, P. 63)

2.2.2 Diagrama de Classe

O Diagrama de classe está entre os principais diagramas da UML. É utilizado na construção do modelo de classe, e tem por finalidade auxiliar no entendimento da estrutura do sistema. Mostra o relacionamento entre as classes, organiza um conjunto de classes em um modelo para atender a funcionalidades do sistema.

> O diagrama de classe é provavelmente o mais utilizado e é um dos mais importantes da UML. Serve de apoio para a maioria dos demais diagramas. Como o próprio nome diz, define a estrutura das classes utilizadas pelo sistema, determinando, os atributos e métodos que cada classe tem, além de estabelecer como as classes se relacionam e trocam informações entre si. Gilleanes T. A. Guedes (2009, p. 38).

Associações é existência de um relacionamento entre objetos possibilita a troca de informações entre si, produzindo desta forma funcionalidades para o sistema. Para possibilitar o relacionamento entre as classes usa-se a associação que é representada por uma linha reta ligando as classes.

Multiplicidades as associações possibilita o relacionamento entres as classes e a quantidade de objetos aos quais outros objetos podem estar associados. Esses limites são representados por multiplicidades na terminologia da UML. Em cada extremidade da associação possui uma representação que segue na figura abaixo:

Nome	Simbologia
Apenas um	1
Zero ou Muitos	0..*
Um ou Muitos	1..*
Zero ou Um	0..1
Intervalo Específico	1..1

Figura 12: Cardinalidade
Fonte: Bezerra (2007)

Associação de Agregação é usada para indicar a colaboração de um objeto com outro, apesar de sua extremidade não ser obrigatória. Na associação um objeto é parte do outro, de tal forma que a parte pode existir sem o todo. Um objeto possui referencia para outro.

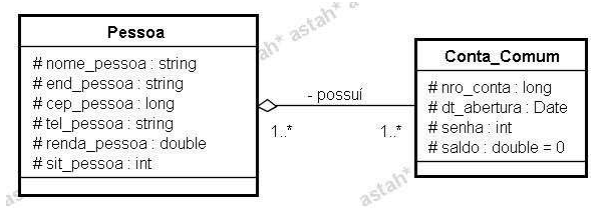


Figura 13: Agregação
Fonte: Bezerra (2007)

A Associação de Composição é usada para fazer parte de um todo. Os objetos são inseparáveis, uma classe depende de outra classe para ser utilizada e quando elimina essa classe a outra na faz sentido.

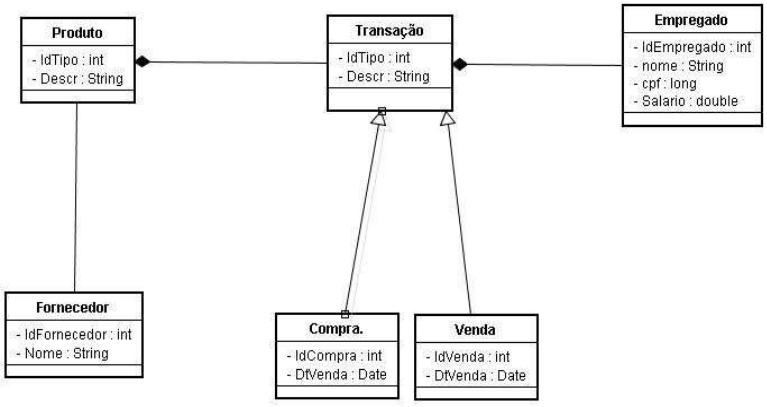


Figura 14: Diagrama de Classe com composição
Fonte: Bezerra (2007)

2.2.3 Diagrama de Atividades

O diagrama de atividades tem por objetivo representar a seqüência de passos de uma atividade num sistema. Seus elementos podem ser divididos em dois grupos: os que são utilizados na composição do fluxo de controle seqüenciais e os que são utilizados para representar o fluxo de controle paralelo. Deve ter um estado inicial e pode ter vários estados finais, também pode não ter estado final, o que significa que o processo é cíclico.

O diagrama de atividades preocupa-se em descrever os passos a serem percorridos para a conclusão de uma atividade especifica, podendo esta ser representada por um método com certo grau de complexidade ou mesmo por um processo completo. O diagrama de atividades concentra-se na representação do fluxo de controle de uma atividade. Gilleanes T. A. Guedes (2009, p. 38).

Há um ponto de ramificação onde possui um único ponto de entrada e vários pontos de saída. Onde para cada ponto de saída há uma condição a ser atendida, desta forma cada condição segue caminhos distintos. Podem ter dois ou mais fluxo de controle paralelo simultaneamente chamado de barra de bifurcação (FORK) que podem receber uma transição de entrada e cria dois ou mais fluxo de saída, já a barra de junção (JOIN) recebe dois ou vários fluxo de controle e une em um único.

Raias de natação dividem o diagrama de atividades em compartimentos ou raias que refere aos vários agentes que participa da atividade. Geralmente ocorre no negocio que varais pessoas ou departamentos interagem numa mesma atividade.

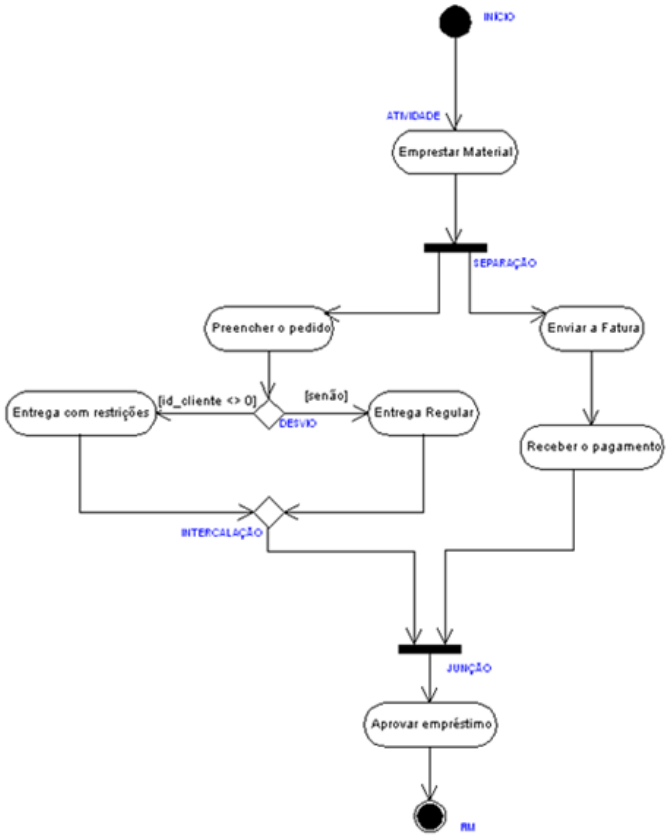


Figura 15: Diagrama de Atividades
Fonte: Booch, Rumbaugh e Jacobson (2005, p. 190)

2.2.4 Diagrama de objetos

Segundo Booch, Rumbaugh e Jacobson (2005, p. 190), "Diagrama de objetos é bastante dependente do diagrama de classes é um diagrama que mostra um conjunto de objetos e seus relacionamentos em um ponto no tempo. Graficamente, o diagrama de objetos é uma coleção de vértices e de arcos".

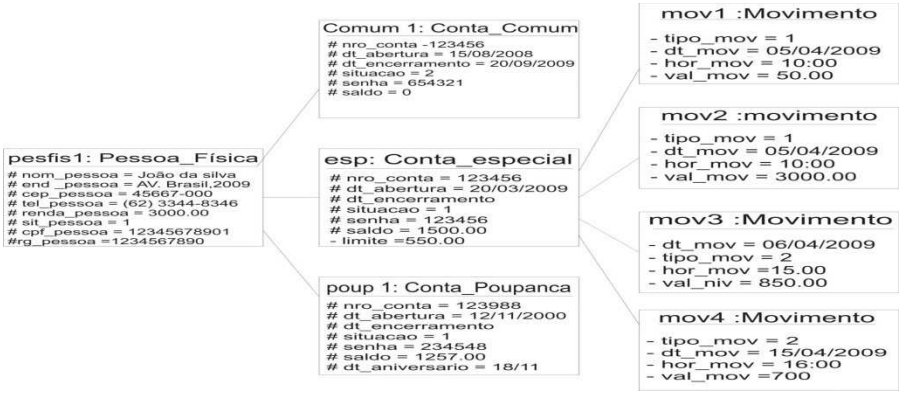


Figura 16: Diagrama de Objetos
Fonte: Booch, Rumbaugh e Jacobson (2005)

2.2.5 Diagrama de estrutura composta

Segundo Gilleanes T. A. Guedes (2009, p.30), "Diagrama de estrutura de componentes a estrutura interna de um classificador, como uma classe ou componentes, detalhando as partes internas que o compõem, como estas se comunicam e colaboram entre si".

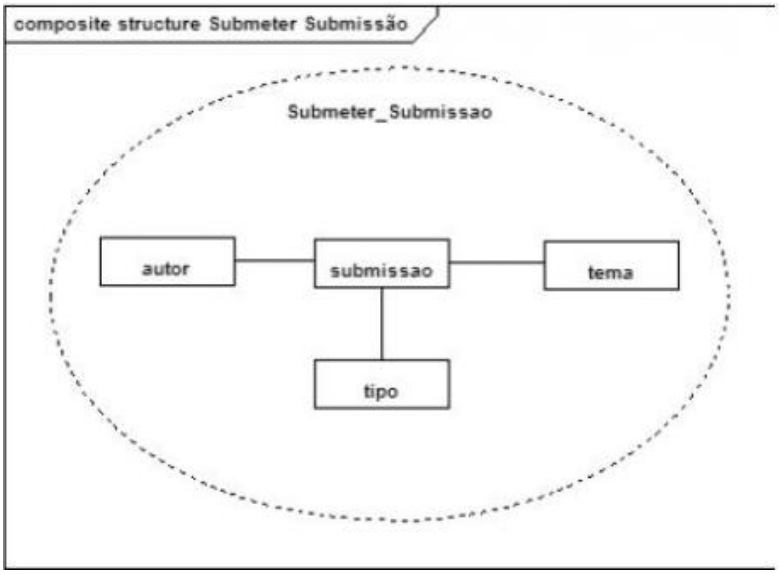


Figura 17: Diagrama de Estrutura Composta
Fonte: Booch, Rumbaugh e Jacobson (2005)

2.2.6 Diagrama de seqüência

Bezerra (2007, p.193) ''A finalidade do diagrama de seqüência é apresentar as interações entre objetos na ordem temporal em que elas acontecem. Possui um conjunto de elementos gráficos''.

> Este e um diagrama comportamental que procura determinar a seqüência de eventos que ocorrem em um determinado processo, identificando quais mensagens devem ser disparadas entre os elementos envolvidos e em que ordem. Assim, determinar a ordem em que os eventos ocorrem, as mensagens que são enviadas, os métodos que são chamados e como os objetos interagem dentro de um determinado processo é o objetivo principal desse diagrama. Gilleanes T. A. Guedes (2009, p. 200).

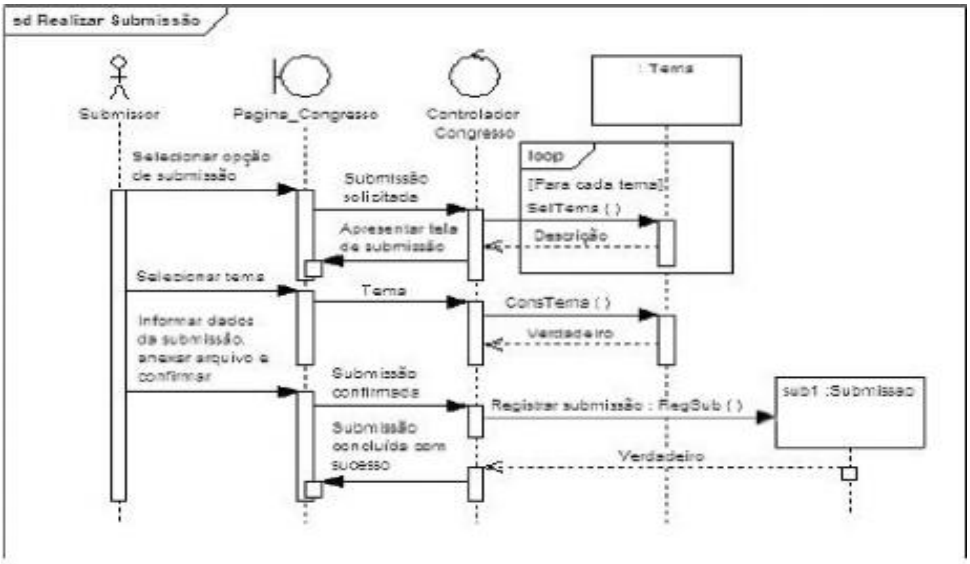


Figura 18: Diagrama de Seqüência
Fonte: Booch, Rumbaugh e Jacobson (2005)

2.2.7 Diagrama de colaboração (comunicação)

As informações mostradas no diagrama de comunicação são, com freqüência, praticamente as mesmas apresentadas no diagrama de seqüência, porem com enfoque diferente, visto que esse diagrama não se preocupa com a temporalidade do processo, concentrando-se em como os elementos do diagrama estão vinculados e quais mensagens trocam entre si durante um processo. Gilleanes T. A. Guedes (2009, p. 239).

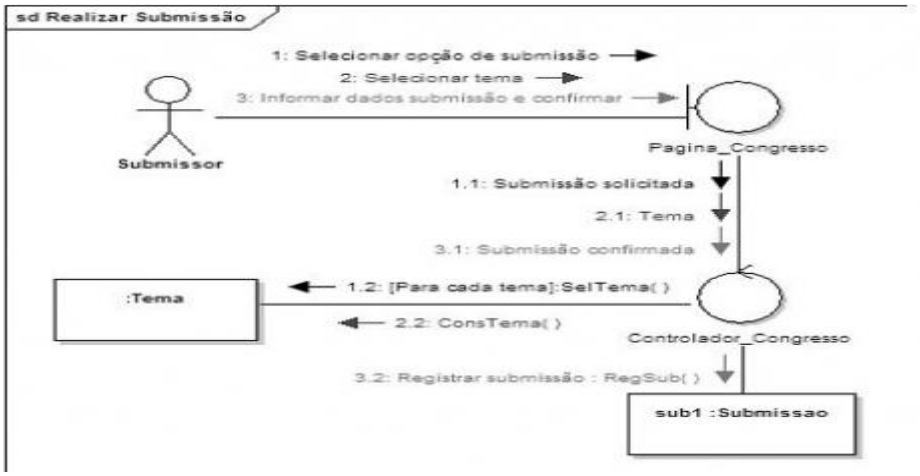


Figura 19: Diagrama de Colaboração/Comunicação
Fonte: Booch, Rumbaugh e Jacobson (2005)

2.2.8 Diagrama de gráfico de estado (máquina de estado)

> O diagrama de máquina de estado demonstra o comportamento de um elemento por meio de um conjunto finito de transições de estado, ou seja, uma máquina de estados. Além de poder ser utilizado para expressar o protocolo de uso de parte de um sistema, quando identifica uma máquina de estado de protocolo. Pode ser usada para especificar o comportamento de vários elementos do modelo. O elemento modelado muitas vezes é uma instância de uma classe. Gilleanes T. A. Guedes (2009, p. 250).

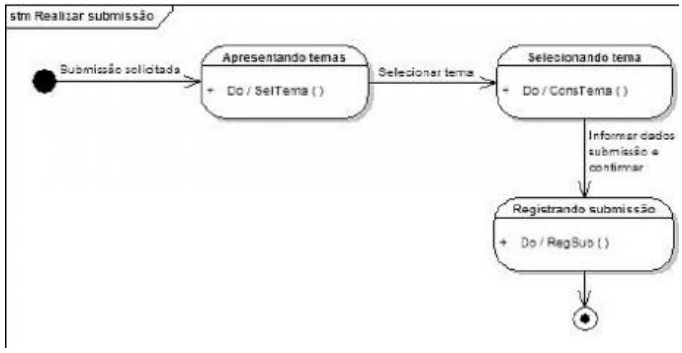


Figura 20: Imagem de estado (máquina de estado)
Fonte: Booch, Rumbaugh e Jacobson (2005)

2.2.9 Diagramas de componentes

Segundo Gilleanes T. A. Guedes (2009, p.33), "Diagrama de componentes representa os componentes do sistema quando o mesmo for ser implementado em termos de módulos de código fonte, bibliotecas, formulários, arquivos de ajuda, módulos executáveis etc".

> O diagrama de componentes pode ser utilizado como uma forma de documentar como estão estruturados os arquivos físicos de um sistema, permitindo assim uma melhor compreensão do mesmo, além de facilitar a reutilização de código. Esse diagrama também pode identificar os componentes utilizados no desenvolvimento de sistemas baseado em componentes. Gilleanes T. A. Guedes (2009, p. 329).

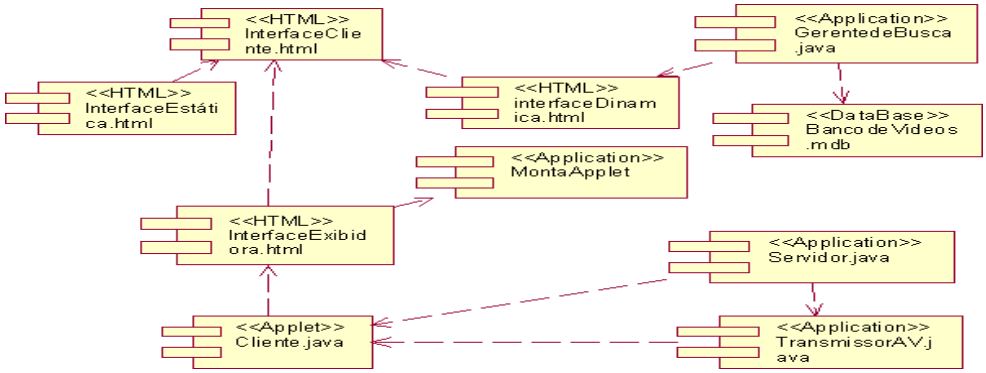


Figura 21: Exemplo de Diagrama de Componente
Fonte: Booch, Rumbaugh e Jacobson (2005)

2.2.10 Diagrama de implantação

Segundo Gilleanes T. A. Guedes (2009, p.34), "Diagrama de implantação determina a quantidade de hardware do sistema, as características físicas como servidores, estações, topologias, e protocolos de comunicação, ou seja, todo o aparato físico".

> Um diagrama de implantação obviamente só é útil quando o sistema modelado for executado sobre múltiplas maquinas, as quais executem determinados módulos do software ou armazenem arquivos necessários ao mesmo. Se o software for projetado para ser executado sobre uma única máquina individual que não se comunique com outro hardware, não é necessário modelar um diagrama de implantação. Gilleanes T. A. Guedes (2009, p. 343).

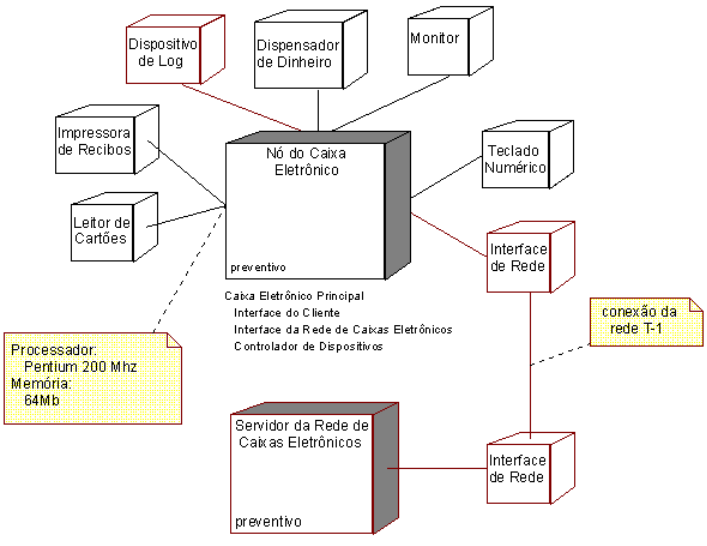


Figura 22: Diagrama de implantação
Fonte: Booch, Rumbaugh e Jacobson (2005)

2.2.11 Diagramas de pacotes

Segundo Gilleanes T. A. Guedes, (2009, p.35), “Seu objetivo é representa os subsistemas englobados por um sistema de forma a determinar as partes que o compõem”.

> O diagrama de pacotes descreve como os elementos do modelo estão organizados em pacotes e demonstra as dependências entre eles. Esse diagrama é muito útil para a modelagem de subsistemas e para a modelagem de subdivisões da arquitetura de uma linguagem. Pode ser utilizado também para representar um conjunto de sistemas integrados, representados por pacotes, ou ainda os submódulos englobados por um sistema. O diagrama pode ser utilizado ainda para demonstrar a arquitetura de uma linguagem, como ocorre com a própria UML, ou arquitetura de um processo de desenvolvimento. Gilleanes T. A. Guedes (2009, p. 195).

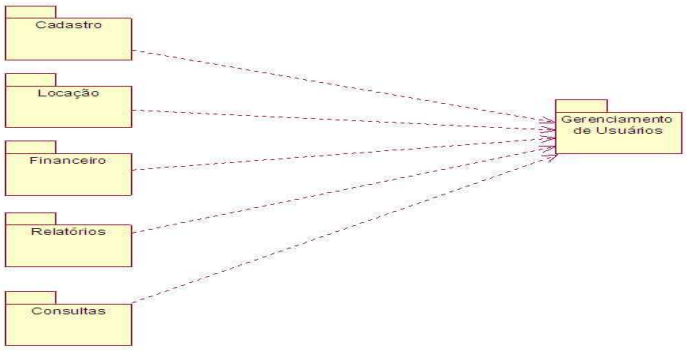


Figura 23: Diagrama de pacotes
Fonte: Booch, Rumbaugh e Jacobson (2005)

2.2.12 Diagrama de interação geral

Segundo Gilleanes T. A. Guedes (2009, p.35), “É uma variação do diagrama de atividades que fornece uma visão geral dentro de um sistema ou processo de negocio”.

> Seu objetivo é fornecer uma visão geral do controle de fluxo. O diagrama utiliza quadros no lugar dos nós de ação, embora símbolos como o nó de decisão e o nó inicial sejam também utilizados. Existem basicamente dois tipos de quadros: quadros de interação, que contêm qualquer tipo de diagrama

de interação de UML, e quadro de ocorrência de interação, que normalmente fazem uma referência a um diagrama de interação, mas não apresentam seu detalhamento. Gilleanes T. A. Guedes (2009, p. 195).

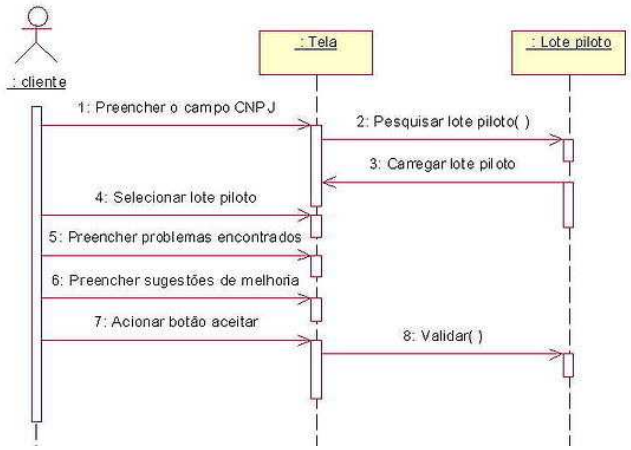


Figura 24: Diagrama de interação geral
Fonte: Booch, Rumbaugh e Jacobson (2005)

2.2.13 Diagrama de tempo

Segundo Gilleanes T. A. Guedes (2009, p.36), "O diagrama de tempo descreve as mudanças no estado ou condição de uma instância de uma classe ou seu papel durante um tempo. Tipicamente utilizada para demonstrar a mudança no estado de um objeto no tempo em resposta a eventos externos".

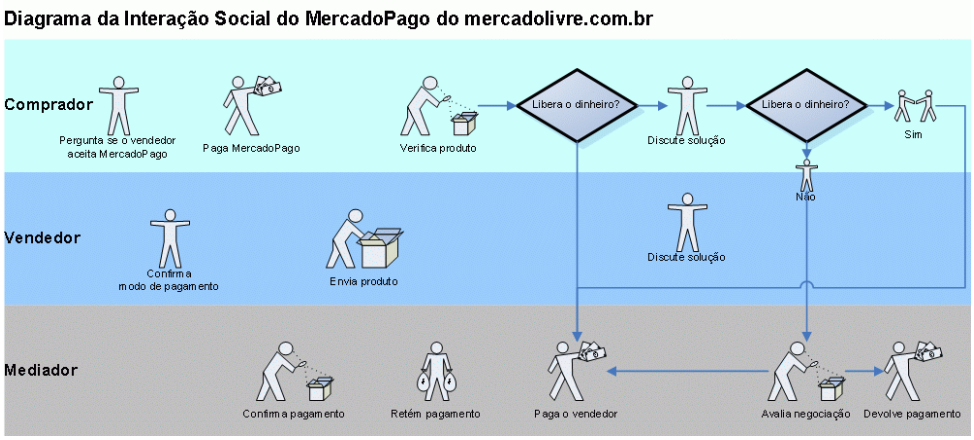


Figura 25: Diagrama de tempo
Fonte: Booch, Rumbaugh e Jacobson (2005)

2.3 Banco de Dados

2.3.1 Sistema de gerenciamento de bancos de dados (SGBD)

Segundo C. J Date. (2004), um sistema de gerenciamento de banco de dados é um sistema computadorizado cuja finalidade geral é armazenar informações e permitir que os usuários busquem e atualizem essas informações quando as solicitar. Banco de dados é uma coleção de dados que possui um relacionamento. Um repositório dos dados que elimina termos redundantes e a consistência.

Toby Teorey, Sam Lightstone, Tom Nadeau (2007, P. 2), "Um Sistema de Gerenciador de Banco de Dados (SGBD) é um sistema de software genérico para manipular bancos de dados".

2.3.2 Ciclo de vida do Banco de dados

As fazes de um projeto de banco de dados tem o objetivo de criar um banco de dados que possa armazenar informações sem redundância e possibilitando um fácil acesso as informações.

Segundo Toby Teorey, Sam Lightstone, Tom Nadeau (2007, P.3), "O ciclo de vida do banco de dados incorpora os passos básicos envolvidos no projeto de um esquema global do banco de dados lógico, a alocação dos dados por uma rede de computadores e a definição de esquema locais específicos do SGBD".

Para Toby Teorey, Sam Lightstone, Tom Nadeau (2007, p 3), "Análise de Requisitos, os requisitos do banco de dados são determinados entrevistando-se os produtores e os usuários dos dados".

Na visão de Toby Teorey, Sam Lightstone, Tom Nadeau, (2007, P.3), "**Projeto Lógico**, o esquema global, um diagrama de modelo de dados conceitual que mostra todos os dados e seus relacionamentos, é desenvolvido usando técnicas como ER".

No entendimento de Toby Teorey, Sam Lightstone, Tom Nadeau, (2007, P. 7) "**Projeto Físico -** envolve a seleção de índices (métodos de acesso), particionamento e clustering de dados".

2.3.3 Modelo de Dados Conceitual

> O modelo conceitual é o principal componente do projeto lógico do banco de dados. Os diagramas de esquema foram formalizados na década de 1960 por Charles Bachman. Ele usou retângulos para indicar os tipos de registros e setas direcionadas de um tipo de registro para outro a fim de indicar um relacionamento um para muitos entre as instancias de registros desses dois tipos. O modelo ER básico consiste em três classes de objetos: entidades, relacionamentos, e atributos. Toby Teorey, Sam Lightstone, Tom Nadeau, (2007, p. 9).

2.3.4 Normalização

Segundo Toby Teorey, Sam Lightstone, Tom Nadeau, (2007, p. 109), "As tabelas de banco de dados relacional, sejam elas derivadas dos modelos ER ou UML, às vezes sofrem com alguns problemas bastante sérios em termos de desempenho, integridade e facilidade de manutenção".

2.3.4.1 Primeira Forma Normal

> Uma tabela esta na primeira forma normal (1FN) se, e somente se, todas as colunas tiverem apenas valores atômicos, ou seja, se cada coluna só puder ter um valor para cada linha na tabela. Toby Teorey, Sam Lightstone, Tom Nadeau, (2007, p. 111)

2.3.4.2 Segunda Forma Normal

> Uma tabela esta na segunda forma normal (2FN) se, ela estiver na (1FN) e os atributos não chave forem totalmente dependentes da chave primaria. Um atributo será totalmente dependente da chave primária se estiver no lado direito de uma DF que tem no lado esquerdo a própria chave primária ou algo que possa ser derivado da chave primária usando a transitividade. Toby Teorey, Sam Lightstone, Tom Nadeau, (2007, p. 113)

2.3.4.3 Terceira Forma Normal

> Uma tabela está na terceira forma normal (3FN) se, e somente se, para cada dependência funcional não trivial X->A, onde X e A são atributos simples ou compostos, uma das duas condições precisam ser mantidas: ou a atributo X é uma superchave, ou atributo A é membro de uma chave candidata. Toby Teorey, Sam Lightstone, Tom Nadeau, (2007, p. 116)

3 Desenvolvimento do Projeto

Essa parte do trabalho tem o objetivo de mostrar os artefatos que foram produzidos. Esses que servem para o time desenvolverem o produto.

Foram feitos os seguintes artefatos: Documento Visão, artefatos do SCRUM, Modelagem de caso de uso, Diagramas de Atividades, Diagrama de Classe, Diagramas de Seqüência, descrição de caso de uso, Regras de Negocio, Glossário de mensagem, Arquitetura do sistema e Diagrama Entidade Relacionamento.

Os artefatos iniciais, como, documento visão e *product backlog* foram obtidos de reuniões com os usuários das escolas municipais. Esses usuários são: Auxiliar Escolar e Secretaria Escolar. O *product Owner* apresentava os artefatos na medida em que eram construídos. Após essa reunião os interessados decidiam junto com o órgão de tecnologia da informação da Secretaria Municipal de Itaberaí se os artefatos apresentados poderiam fazer parte do projeto.

Uma das grandes dificuldades enfrentadas ao levantar os requisitos ou validar algum artefato foi pelo fator cultural dos funcionários das escolas. Os colaboradores das instituições de ensino já estavam acostumados com a forma de trabalhar e na "visão" de alguns não havia necessidade de mudança.

3.1 Documento visão

A proposta deste documento é coletar, analisar e definir as necessidades e as funcionalidades gerais do sistema SGM – Sistema de Gestão de Matrícula.

Seu foco está nas necessidades dos usuários e no motivo da existência destas necessidades.

3.1.1 Introdução

A proposta deste documento é coletar, analisar e definir as necessidades e as funcionalidades gerais do sistema SGM – Sistema de Gestão de Matrícula.

Seu foco está nas necessidades dos usuários e no motivo da existência destas necessidades.

3.1.2 Problema

Atualmente a Secretaria Municipal de Educação (SME) juntamente com escolas fazem o controle das matrículas de alunos manualmente. E o fazem através de planilhas eletrônicas, editores de texto e pastas.

3.1.3 Resumo do Negócio

A SME é uma instituição educacional do município de itaberaí trabalha no controle de escolas.

E esse controle é feito na: Gerencia de matriculas que são feitas nas instituições de ensino, na contabilização da quantidade de alunos, no tamanho de todas as salas e na quantidade de alunos suportadas.

Quando é feita uma matrícula ou uma saída de aluno numa instituição de ensino cabe a essa manter a SME informada através de planilhas ou editores de texto. Através dessas informações a Secretaria pode autorizar que novas matrículas sejam feitas, verifica a quantidade de alunos, verifica a comportabilidade das salas de aula.

3.1.4 Problemas

O problema de	Gestão manual das matriculas, através de formulários.
Afeta	As escolas, e as famílias. Em que seus dados são cadastrados e mantidos através de formulários manuais e planilhas eletrônicas. Afeta também a própria SME que pelo modo “arcaico” de gestão tem a trabalho comprometido pela lentidão, erros, redundância e etc.
Cujo impacto é	O planejamento estratégico fica comprometido, não podendo ter melhoras eficientes para os anos posteriores.
Benefícios de uma solução seriam	Maior eficiencia no atendimento aos alunos, familias e colaboradores educionais. Maior agilidade na visualizaçao de como está a educaçao municipal. Referente no que se diz a respeito do que acontece nessas instituiçoes. Melhor gestão escolar o que irá gerar satisfaçao pela populaçao em relação ao ensino que recebe e o conforto nesses ambientes

	educacionais.

3.1.5 Usuários

3.1.5.1 Resumo dos Usuários

Nome	Descrição	Responsabilidades
SME	Secretaria Municipal de educação	Entidade Governamental que tem por responsabilidade de: o Fornecer os requisitos necessários o Tirar duvida em relação à Educação municipal o Cumprir as reuniões marcadas o Validar os produtos gerados pelo projeto o Homologar o projeto
Instituições de Ensino	Escolas	Entidade responsável por atuar junto com SME para fornecer as informações necessárias em relação ao seu ambiente funcional de trabalho.

3.1.5.2 Ambiente do Usuário

O SGM será disponibilizado na internet através de servidor *web*. A aplicação SGM deverá ser compatível com os principais e atuais navegadores de mercado.

Para que o ambiente do usuário esteja de acordo com as necessidades físicas e lógicas do sistema, itens como: compatibilidade de hardware e software, acesso à rede, servidores de arquivos e acesso à base de dados devem estar aptos ao acesso e utilização.

3.1.6 Necessidades dos Interessados

	Necessidade	Prioridade	Preocupações	Solução Atual	Soluções Propostas
1	Disponibilizar formulários para matrículas. Cadastro de familias, alunos e escolas forma *on-line* em uma aplicação *web*.	Crítico		Os formulários não estão sistematizados.	Disponibilizar uma interface para gestão da matricula de forma online *web*.
2	Possibilitar a atualização dos dados pertinentes às instituições de	Crítico		Os formulários não estão sistematizados.	Informações totalmente disponíveis para

	forma *on-line.*				atualização na *web*.
3	Gerar consultas	Importante	Disponibilizar dados operacionais para melhor acompanhamento e apoio a decisão. E planejamento estratégico.	Não existem consultas ou automatizados sobre o programa.	Consultas disponíveis para SME.

3.1.7 Requisitos Funcionais

- UC_01 - Manter Solicitações de Matrícula
- UC_02 - Gerar Vaga de Matrícula
- UC_03 - Manter Candidato a aluno
- UC_04 - Manter Matrícula
- UC_05 – Manter Escola
- UC_06 – Gerar Evasão
- UC_07 - Manter Disciplina
- UC_08 – Manter Aluno
- UC_09 – Manter Responsável
- UC_10 – Manter Série

3.1.8 Requisitos Não Funcionais

A Secretaria Municipal de Educação não especificou os requisitos funcionais.

3.2 Artefatos do Scrum

3.2.1 Product Backlog

O *Product Backlog* mostra as funcionalidades desejadas pelos interessados. Obtidas pelo *P*roduct *Owner*. Esse componente da equipe coloca as prioridades das funcionalidades.

ID	Nome	Importância	Estimativa	Como Demonstrar	Notas
UC_01	Manter Solicitações de Matrícula	**A**	**10**	Auxiliar Escolar Deseja Manter Solicitações de Matrícula	Diagramas da UML
UC_02	Gerar Vaga de Matrícula	**A**	**10**	Auxiliar Escolar Deseja Manter Vagas de Matrículas	Diagramas da UML
UC_03	Manter Candidato a Aluno	**A**	**10**	Auxiliar Escolar Deseja Manter Candidato a Aluno	Diagramas da UML
UC_04	Manter Matrícula	**A**	**10**	Auxiliar Escolar deseja manter matrícula	Diagramas da UML
UC_05	Manter Escola	**A**	**10**	Secretária Escolar Deseja Manter Escola	Diagramas da UML
UC_06	Gerar Evasão	**A**	**9**	Auxiliar Escolar Deseja Gerar Evasão	Diagramas da UML
UC_07	Manter Disciplina	**A**	**10**	Secretária Escolar Deseja Manter Disciplina	Diagramas da UML
UC_08	Manter Aluno	**A**	**10**	Auxiliar Escolar Deseja Manter Aluno	Diagramas da UML
UC_09	Manter Responsável	**A**	**10**	Auxiliar Escolar Deseja Manter Responsável	Diagramas da UML
UC_10	Manter Série	**A**	**10**	Secretária Escolar Deseja Manter Série	Diagramas da UML

3.2.2 Sprint Backlog

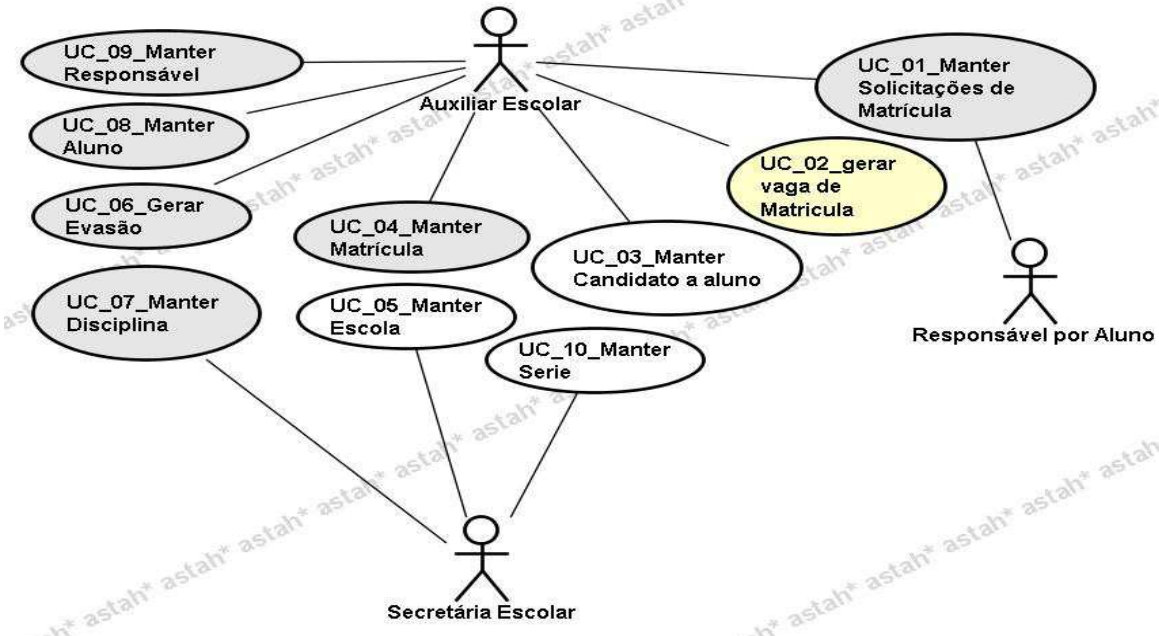
Após o *Product Owner* ter obtido o Product Backlog, o *time* coloca as notas nas funcionalidades de acordo com o que conseguirão desenvolver no tempo proposto. Essas funcionalidades com notas formam a Sprint Backlog, que são desenvolvidas em 2 semanas cada *Sprint.*

Sprint	Duração Semanas	ID	Data
1º	2	UC_01, UC_02, UC_03, UC_04, UC_05, UC_06	01/11/2011 a 18/11/2011
2º	2	UC_07, UC_08, UC_09, UC_10	21/11/2011 a 08/12/2011

3.3 Modelagem com Diagramas da UML

3.3.1 Modelagem de caso de uso

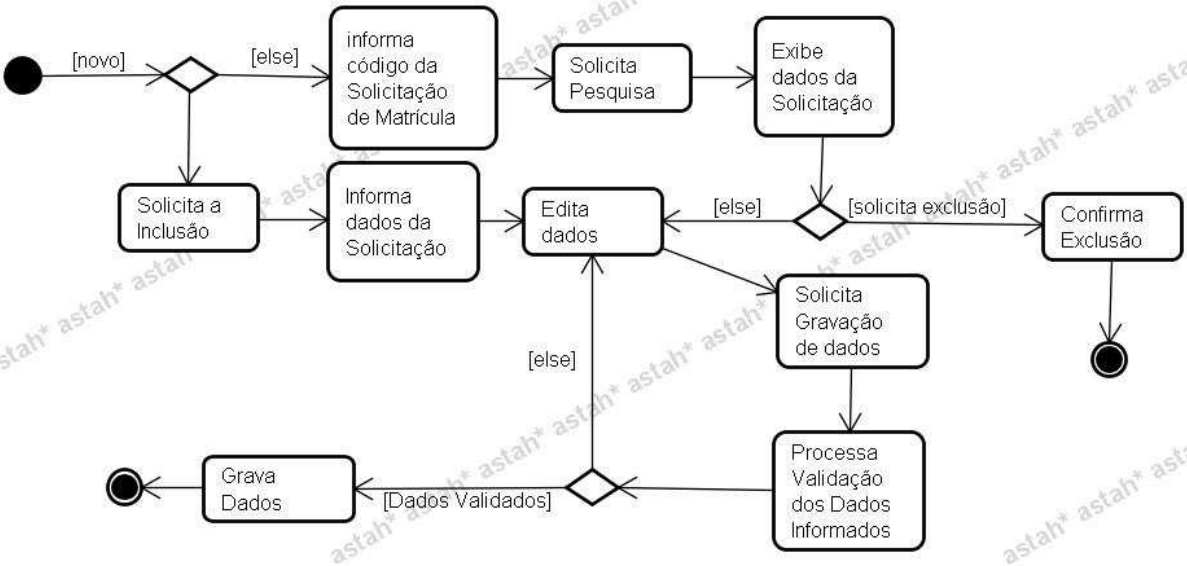
O diagrama de caso de uso irá mostrar as funcionalidades que os usuários irão utilizar no software.



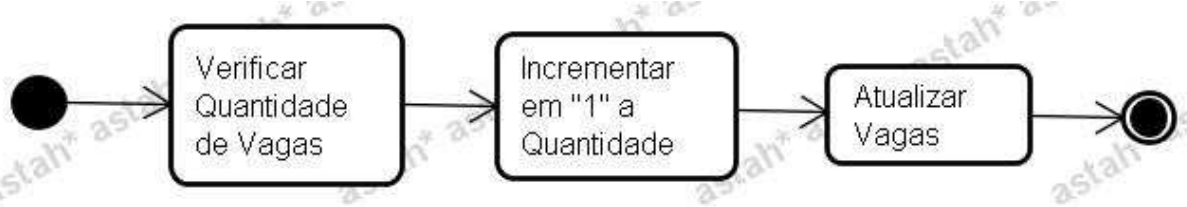
3.3.2 Diagramas de atividades

Os diagramas irão mostrar a seqüência das atividades por cada caso de uso.

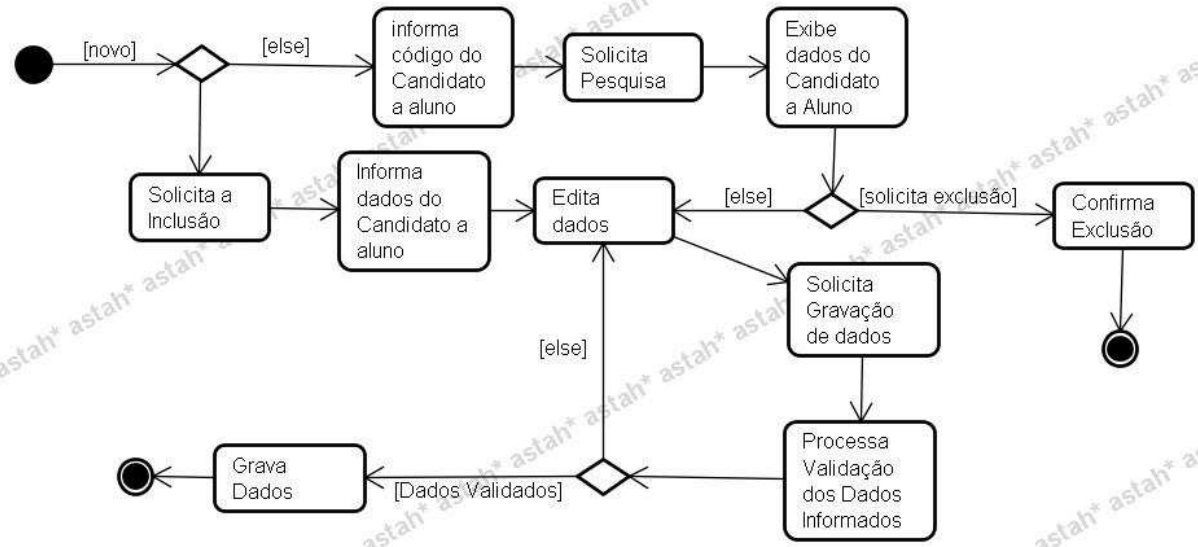
3.3.2.1 UC_01_Manter Solicitações Matrícula:



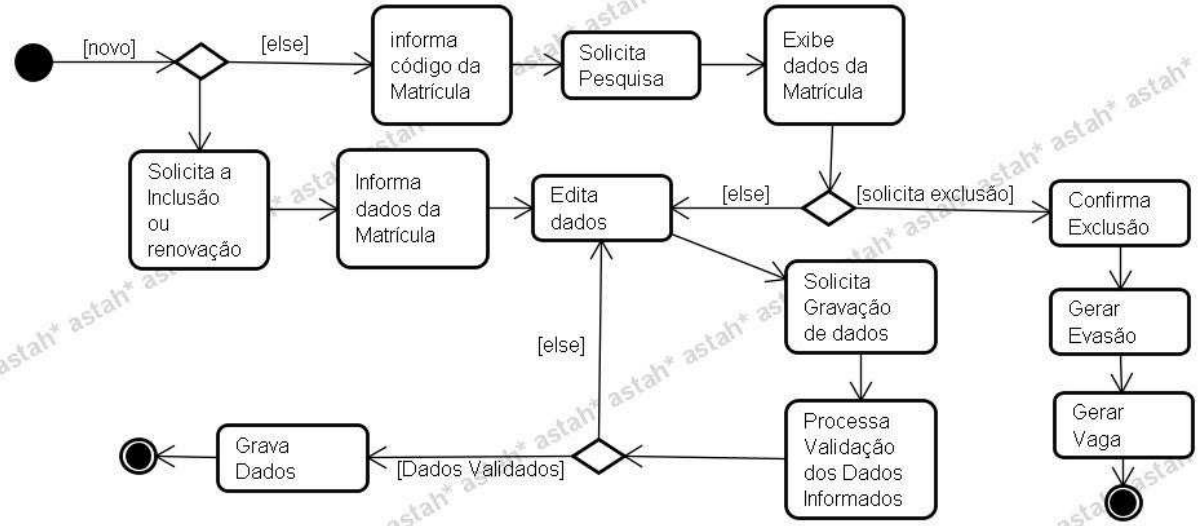
3.3.2.2 UC_02_Gerar Vaga Matrícula:



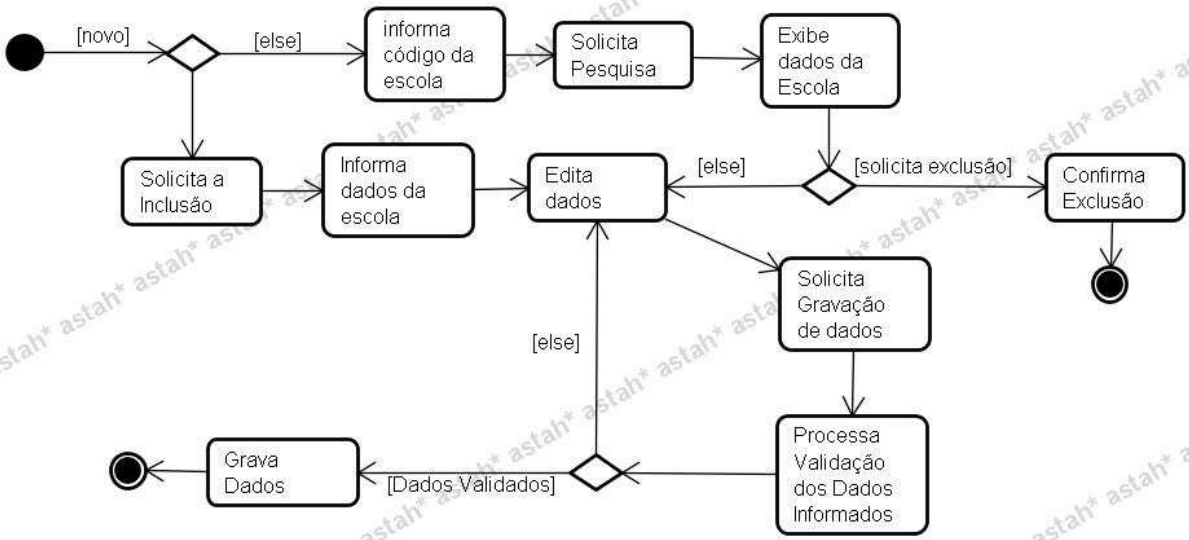
3.3.2.3 UC_03_Manter Candidato a Aluno:



3.3.2.4 UC_04_Manter Matrícula:

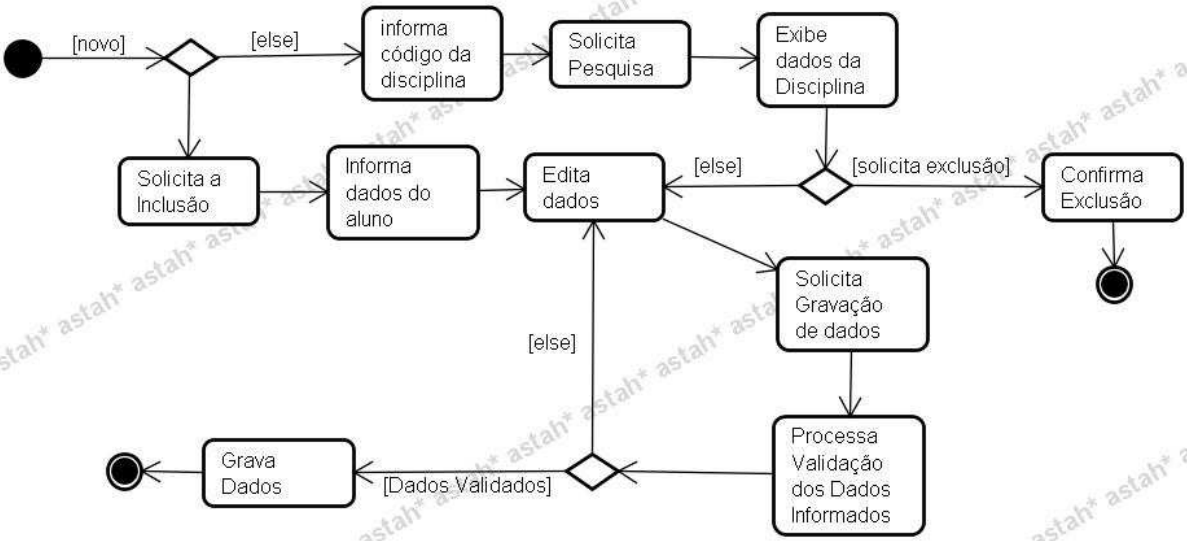


3.3.2.5 UC_05_Manter Escola:

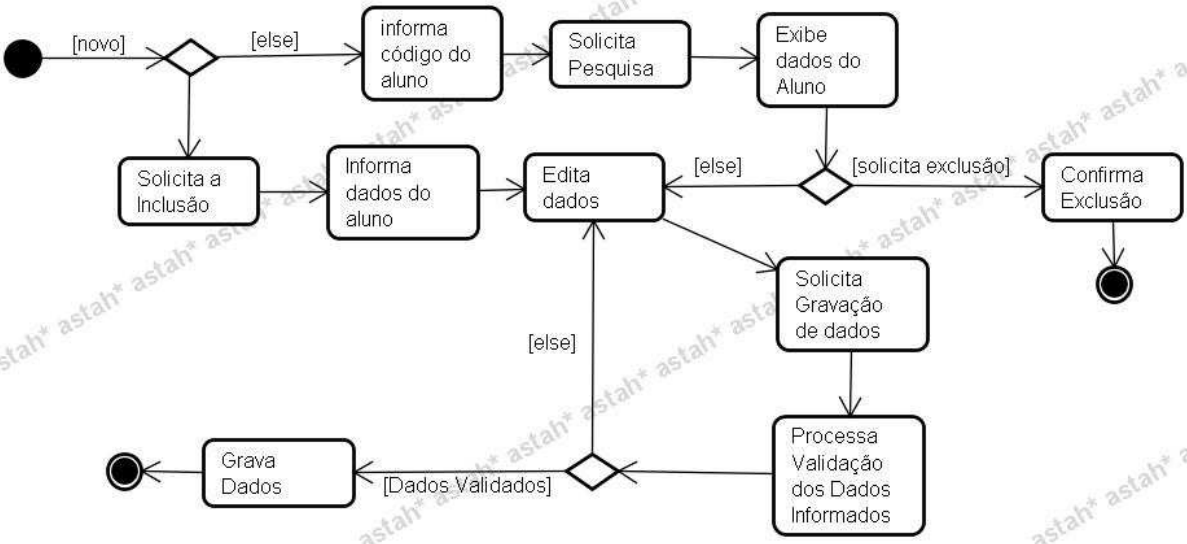


3.3.2.6 UC_06_Gerar Evasão:

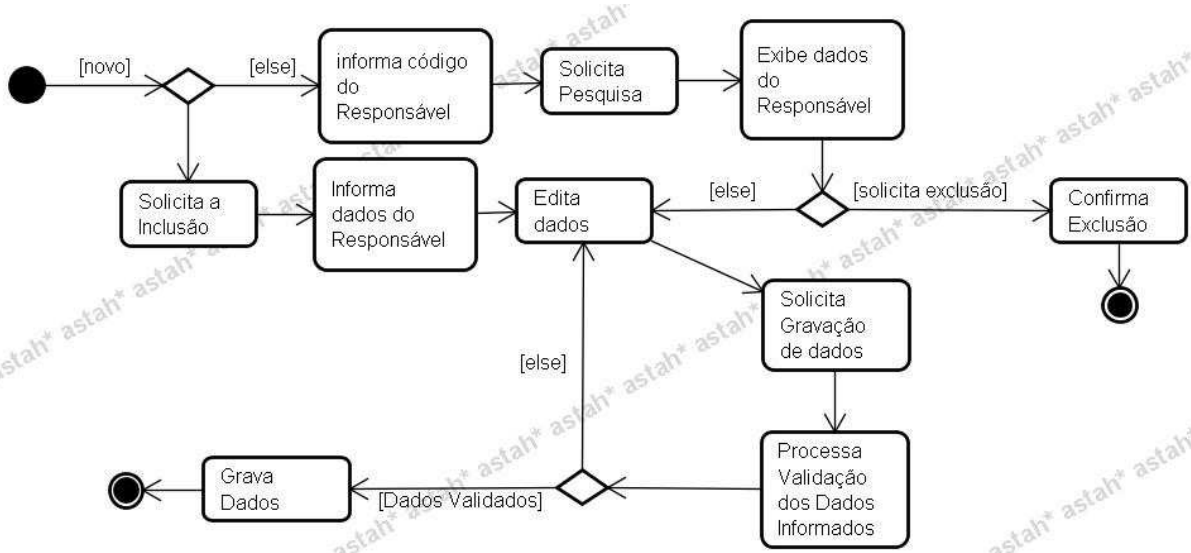
3.3.2.7 UC_07_Manter Disciplina:



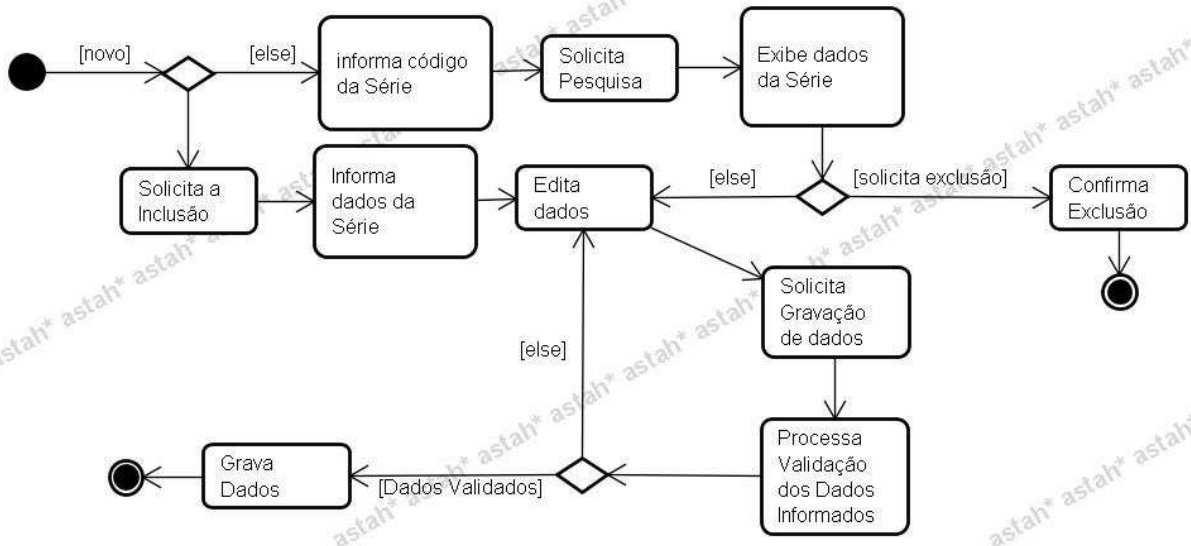
3.3.2.8 UC_08_Manter Aluno:



3.3.2.9 UC_09_Manter Responsável:

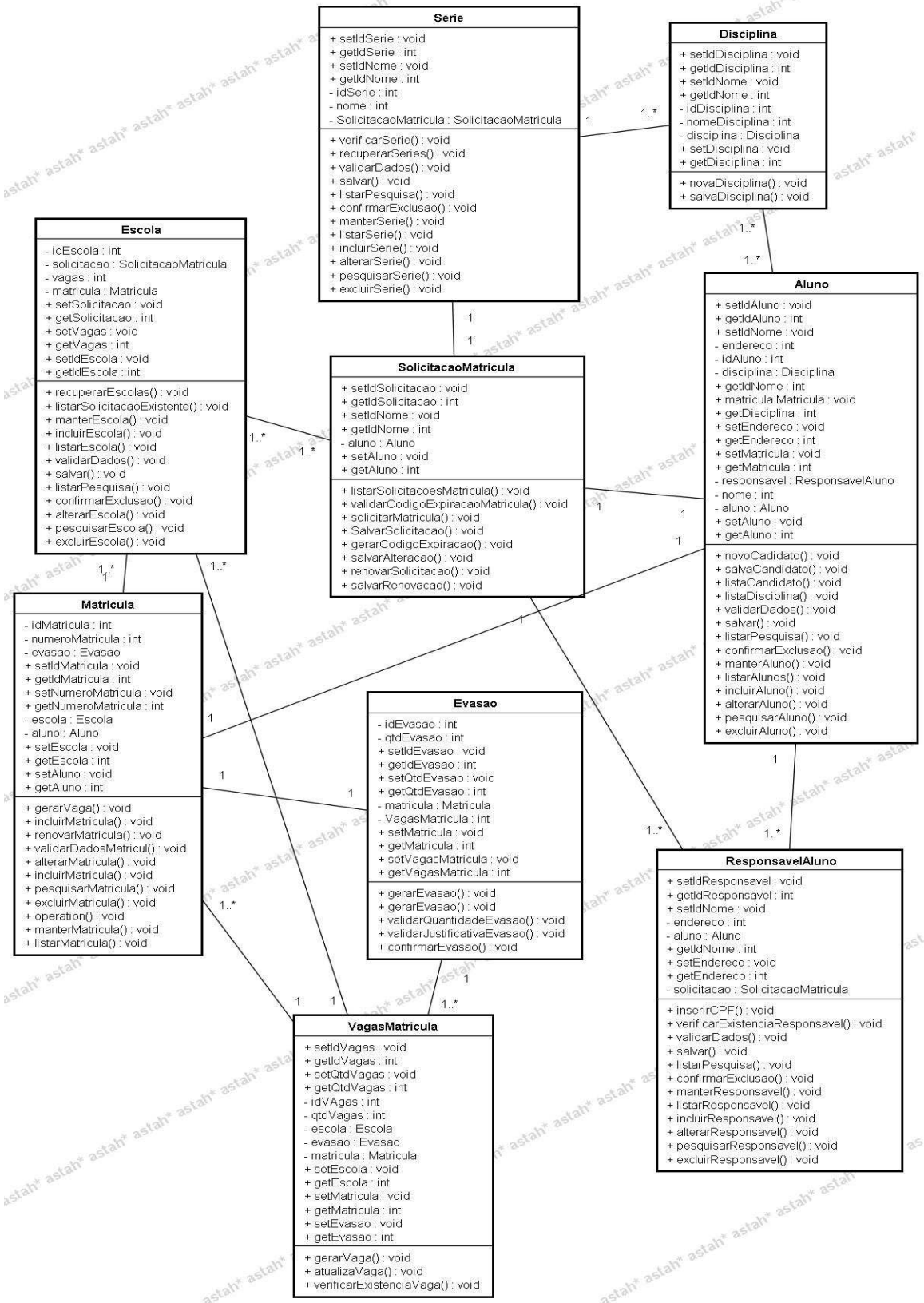


3.3.2.10 UC_10_Manter Série:



3.3.3 Diagrama de Classe

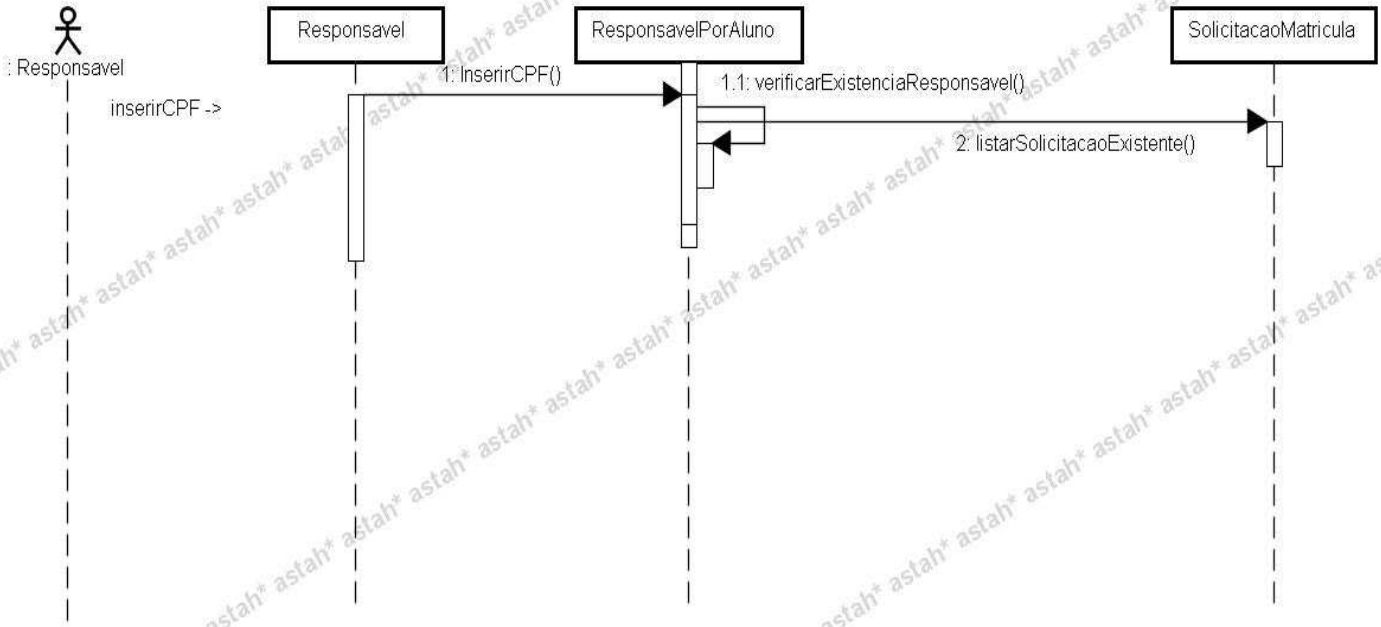
O diagrama de classe mostrará o relacionamento entre as classes e as características e comportamento que os *objetos* terão no software.



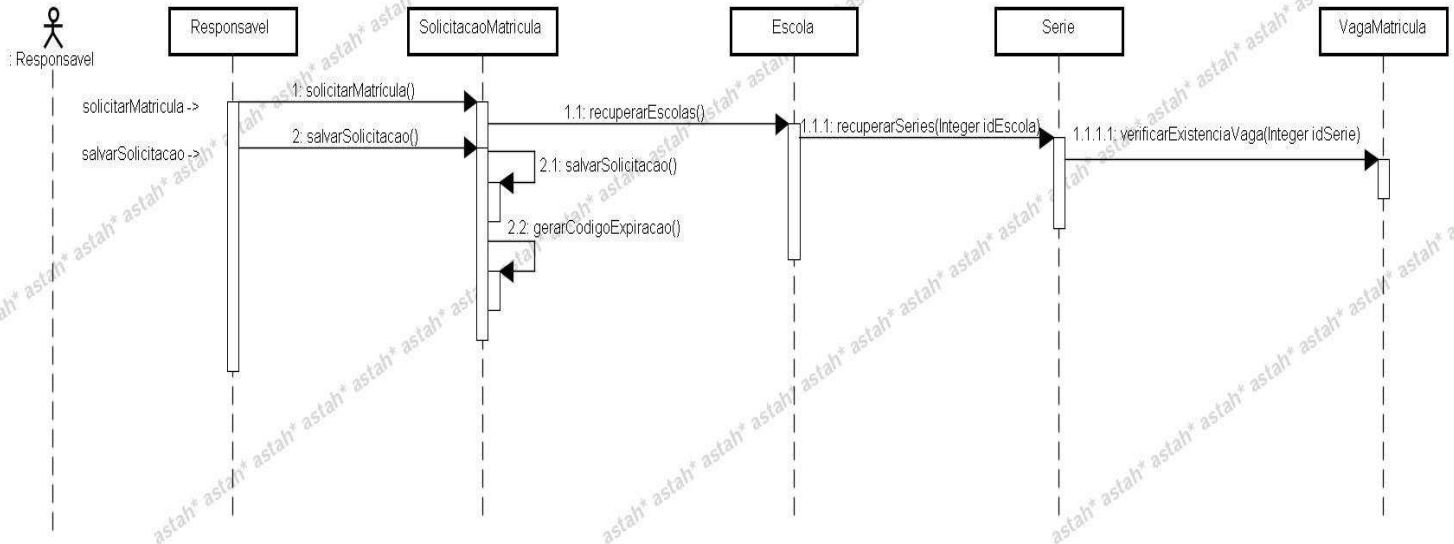
3.3.4 Diagramas de Seqüência

O diagrama de seqüência mostrará como os *objetos* do software irá se comunicar através dos *métodos*. Em Alguns diagramas um caso de uso foi dividido pelos fluxos, tendo vários diagramas. Já em outros teve um diagrama para um caso de uso.

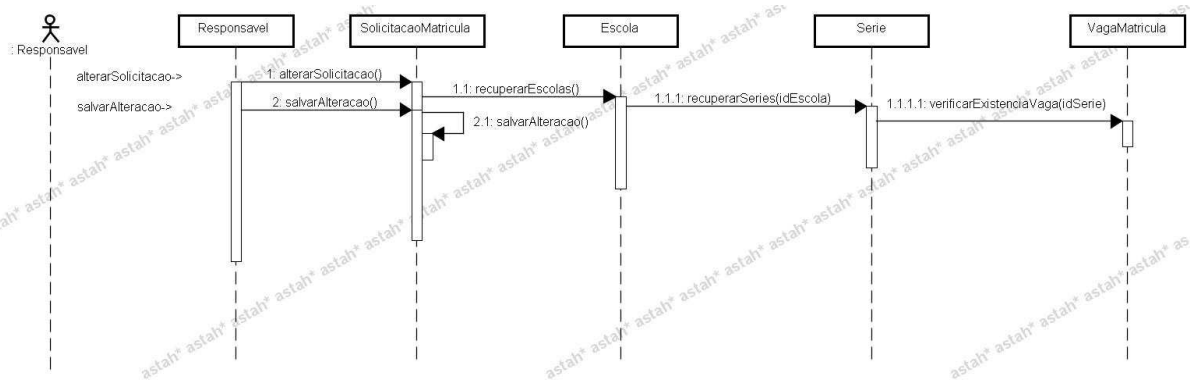
3.3.4.1 UC01 - Manter Solicitações de Matrícula – Fluxo Básico



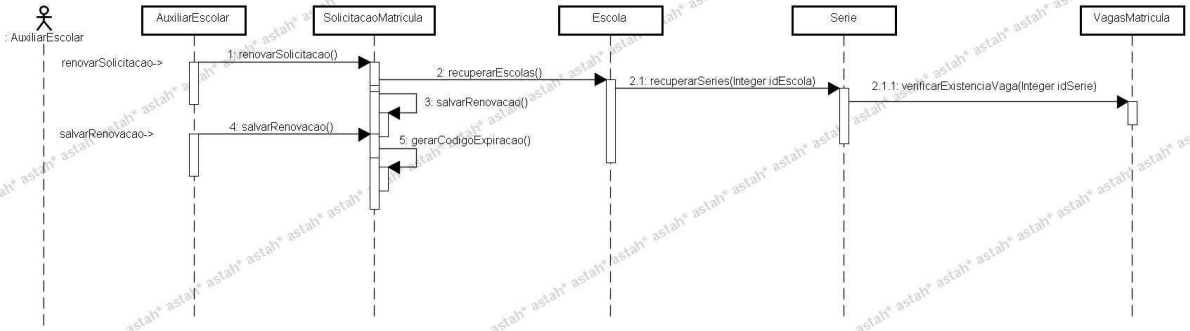
3.3.4.2 UC01 - Manter Solicitações de Matrícula - A1 - Solicitar Matrícula



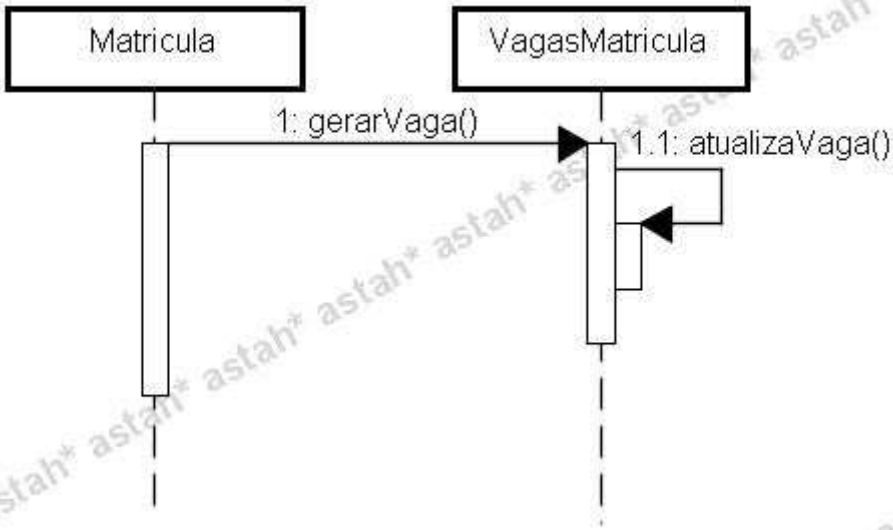
3.3.4.3 UC01 - Manter Solicitações de Matrícula - A2 - Alterar Solicitação de Matrícula



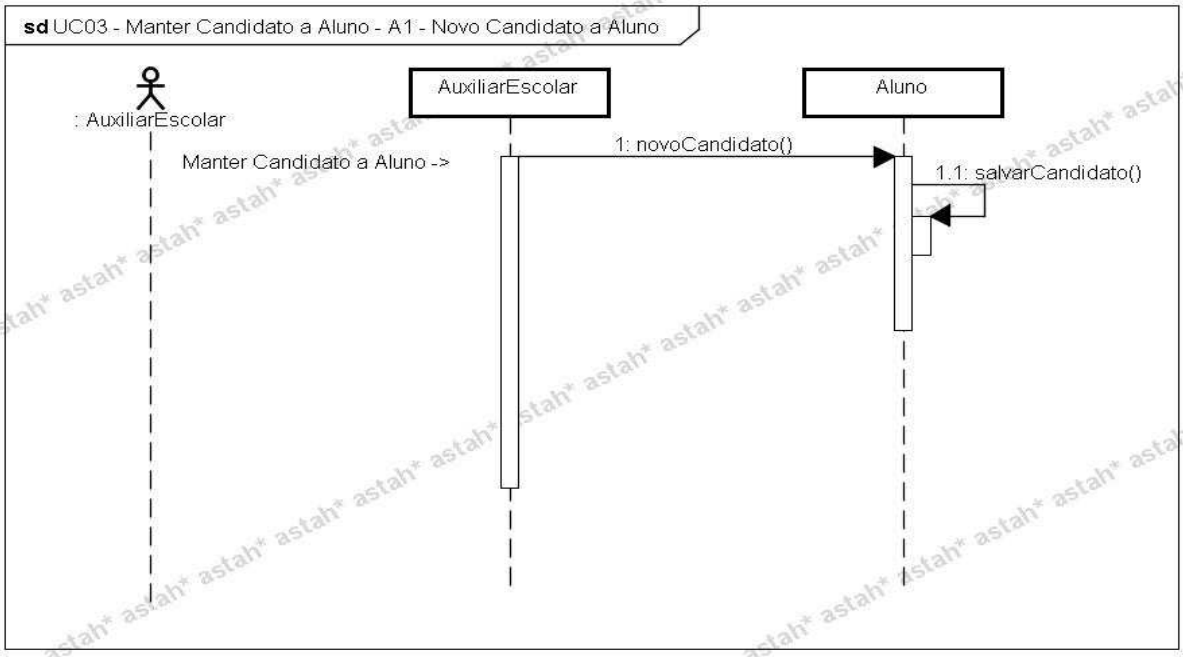
3.3.4.4 UC01 - Manter Solicitações de Matrícula - A5 - Renovar Solicitações de Matrícula



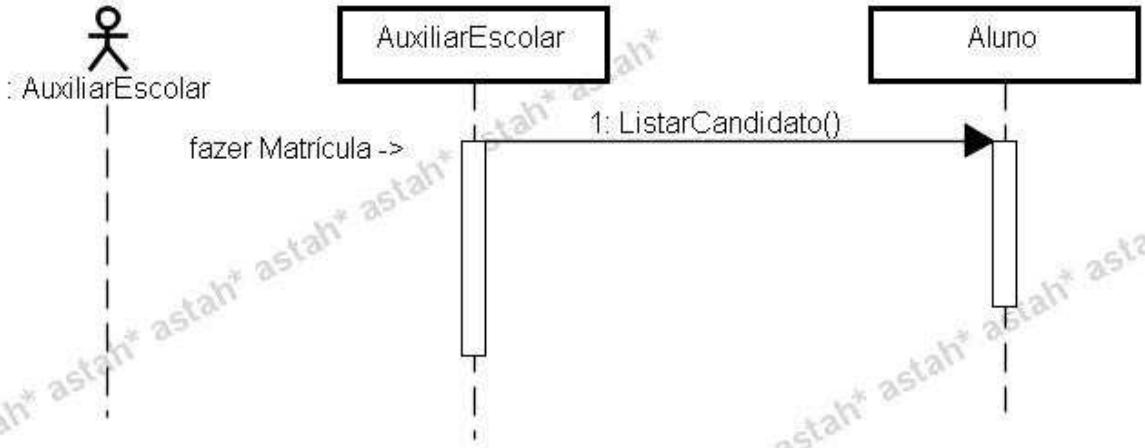
3.3.4.5 UC02 - Gerar Vaga Matrícula - Fluxo Básico



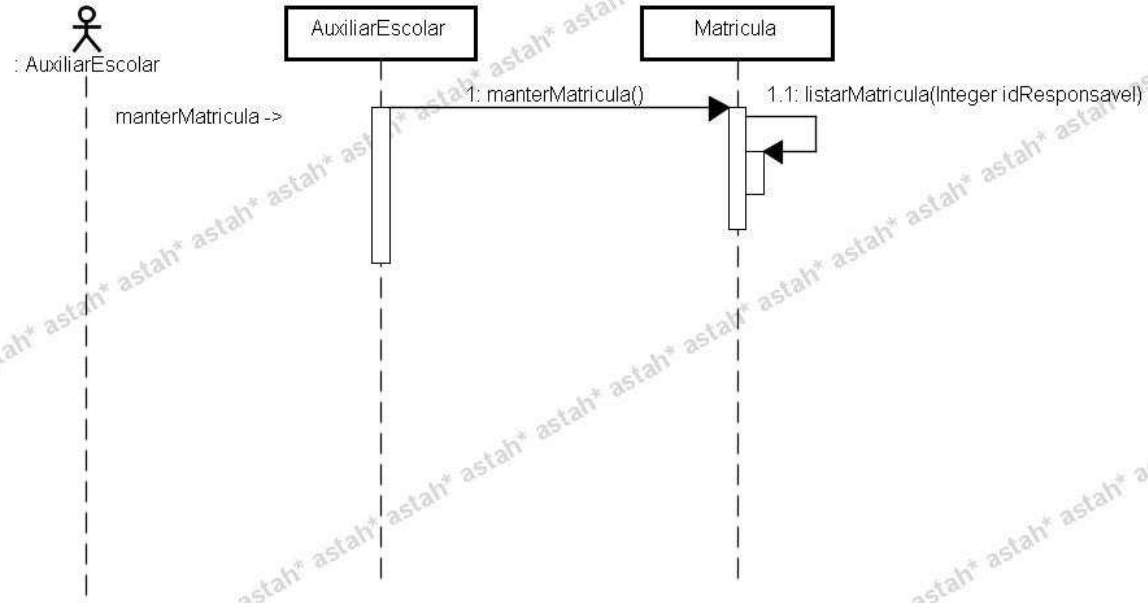
3.3.4.6 UC03 - Manter Candidato a Aluno - A1 - Novo Candidato a Aluno



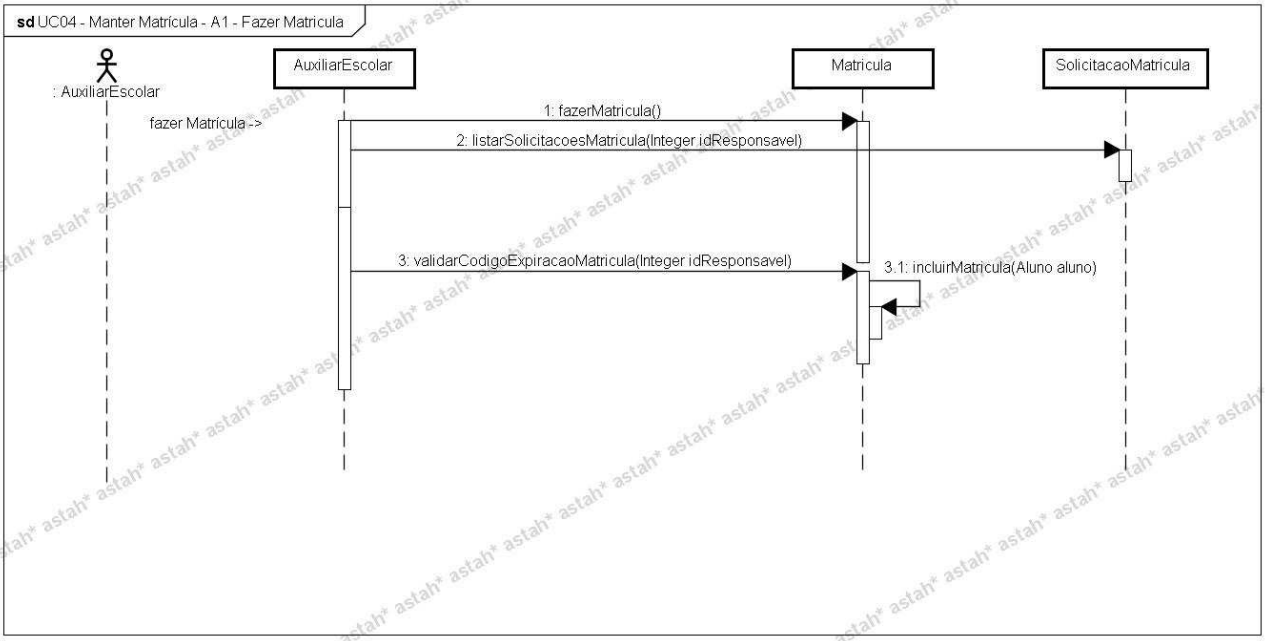
3.3.4.7 UC03 - Manter Candidato a Aluno - Fluxo Básico



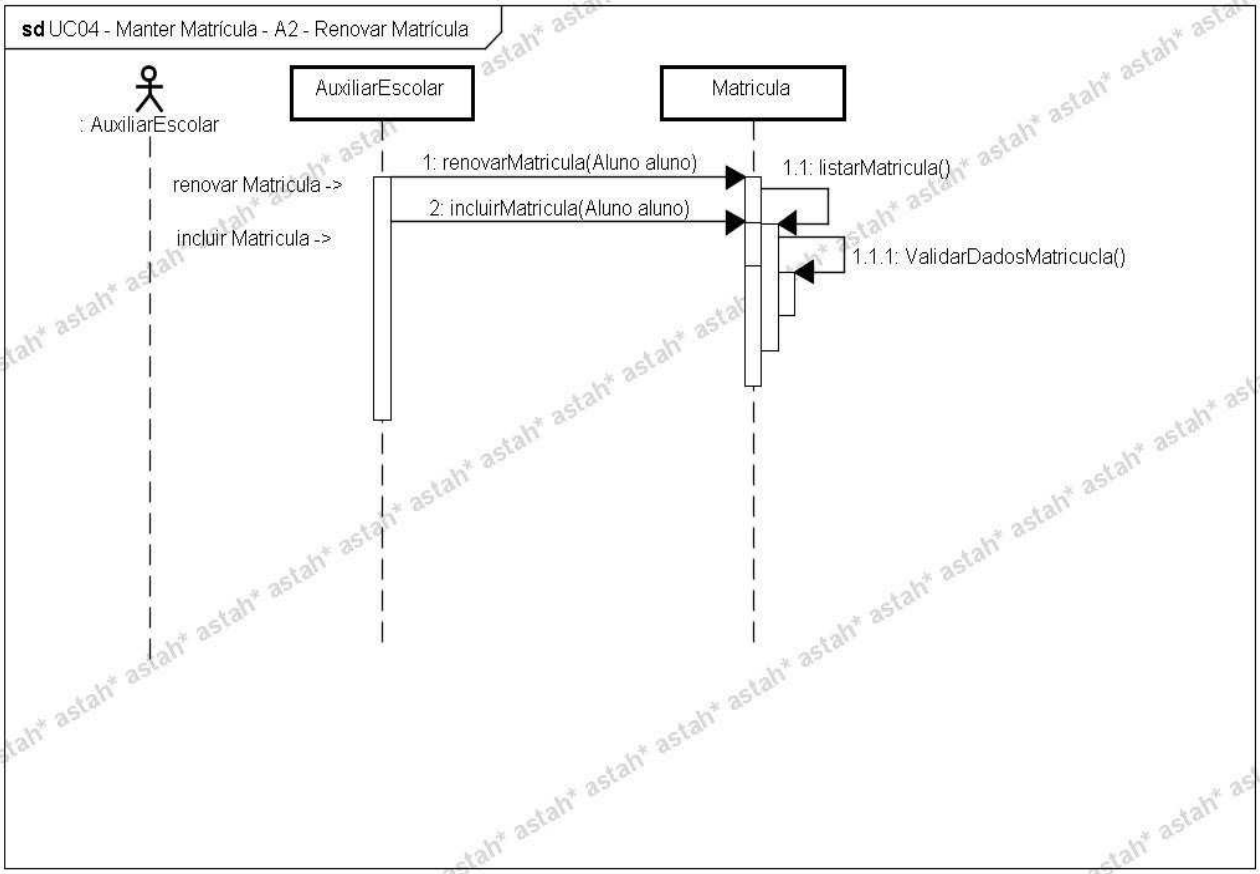
3.3.4.8 UC04 - Manter Matrícula - Fluxo Básico



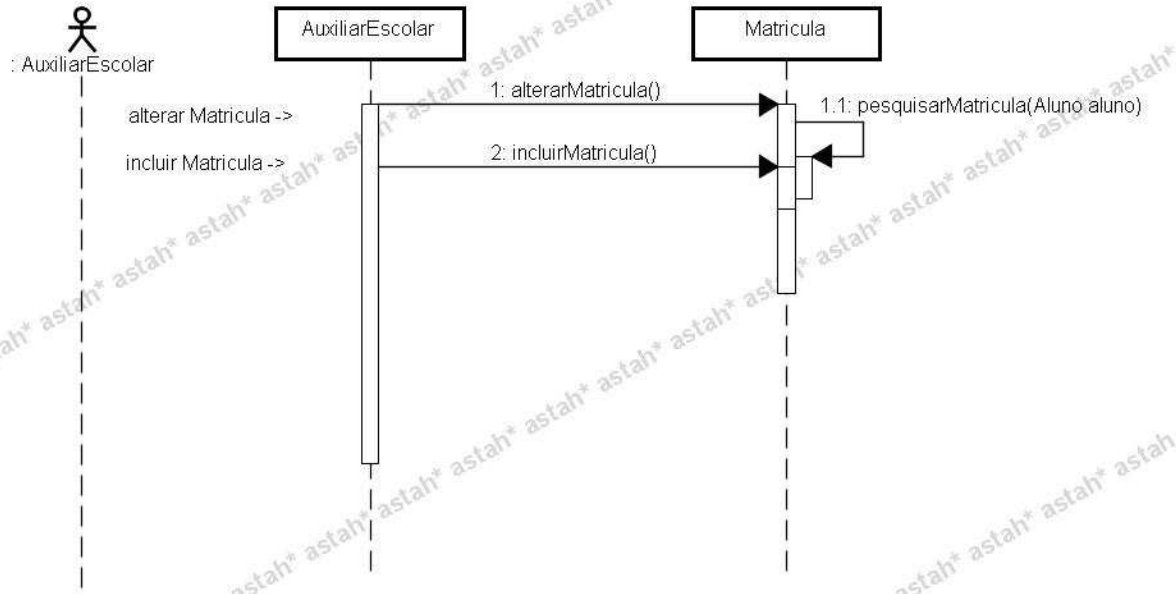
3.3.4.9 UC04 - Manter Matrícula - A1 - Fazer Matrícula



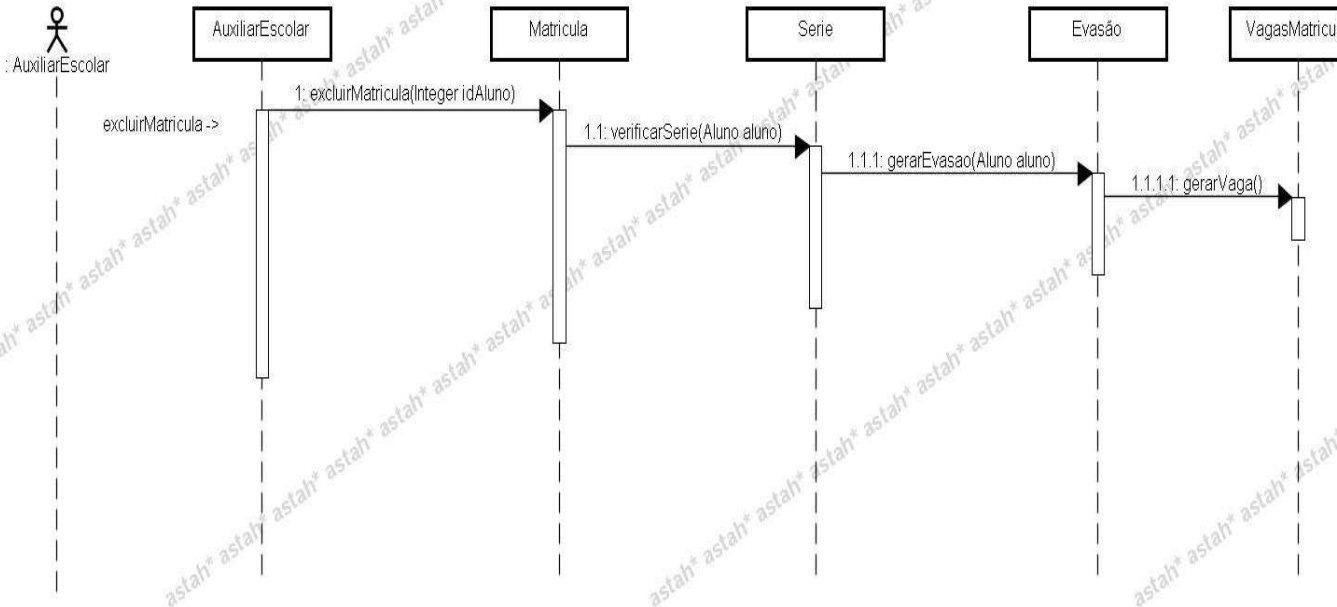
3.3.4.10 UC04 - Manter Matrícula - A2 - Renovar Matrícula



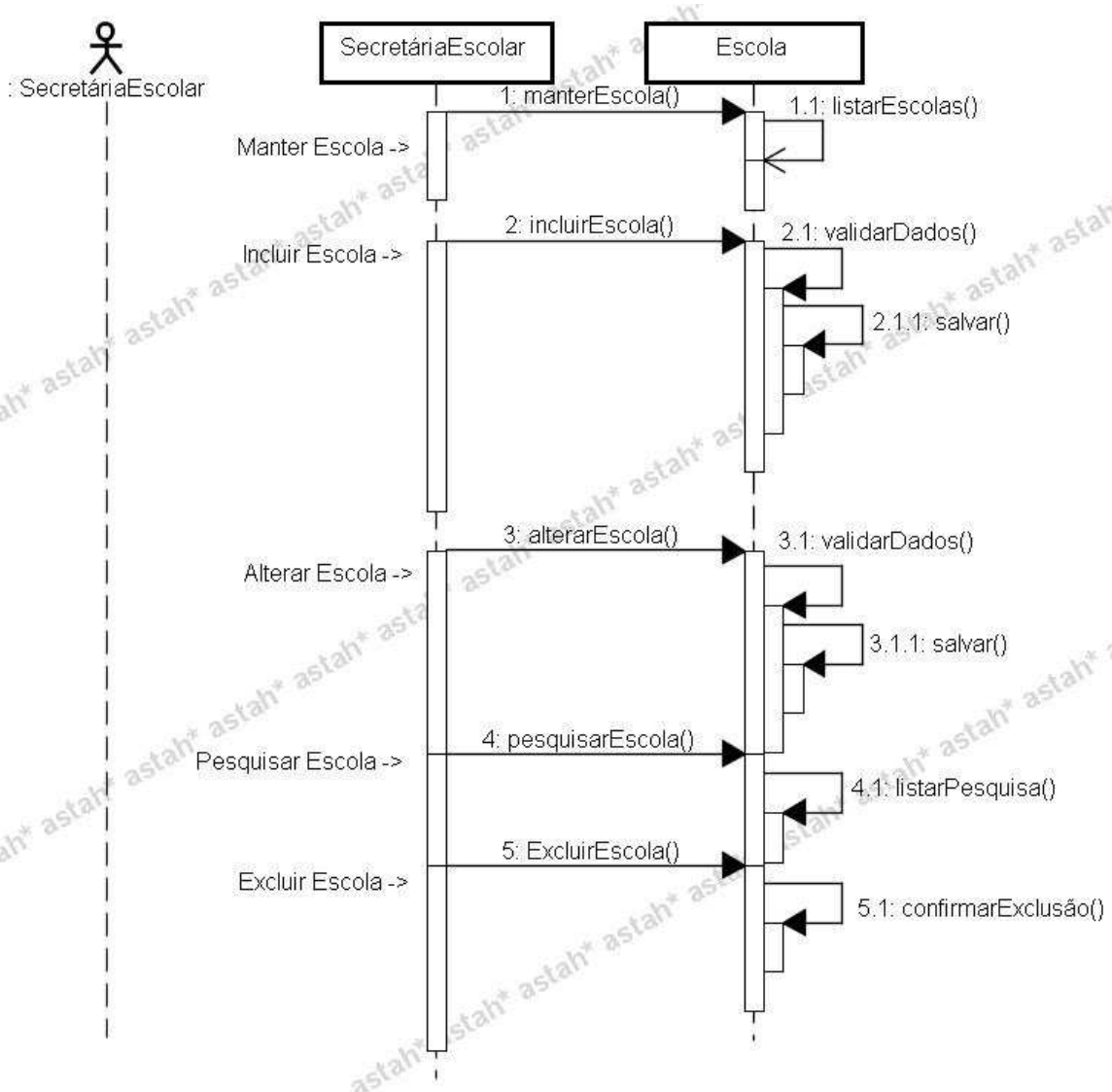
3.3.4.11 UC04 - Manter Matrícula - A4 - Alterar Matrícula



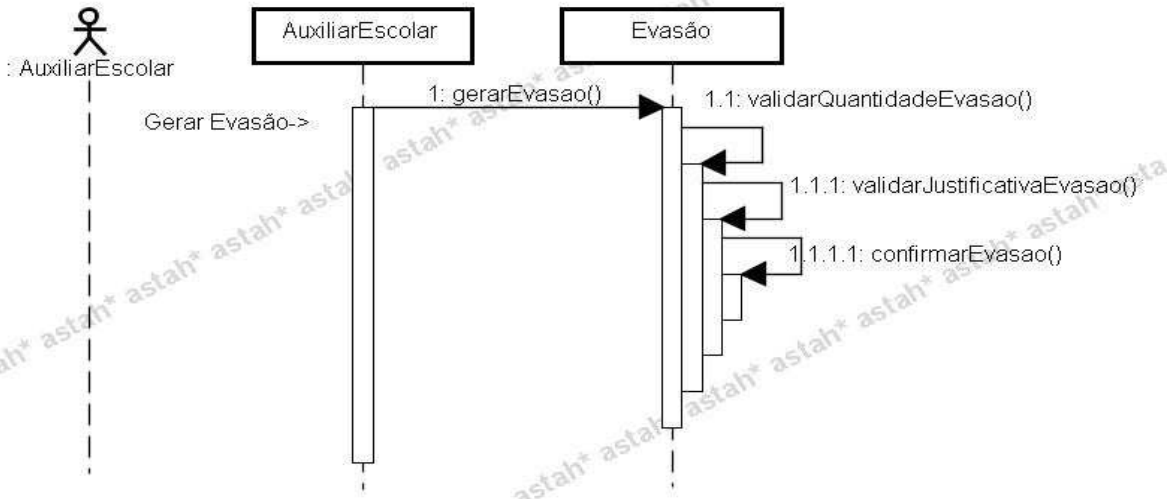
3.3.4.12 UC04 - Manter Matrícula - A5 - Excluir Matrícula



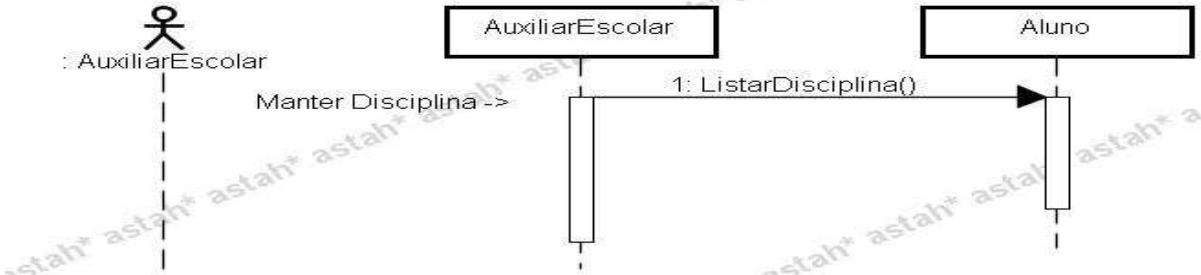
3.3.4.13 UC05 – Manter Escola



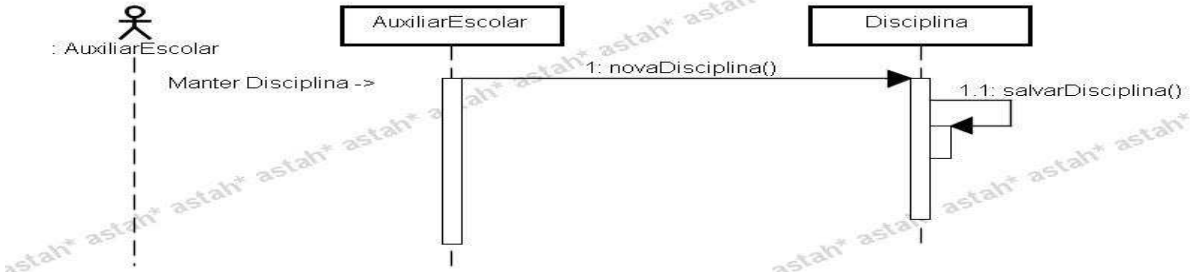
3.3.4.14 UC06 - Gerar Evasão



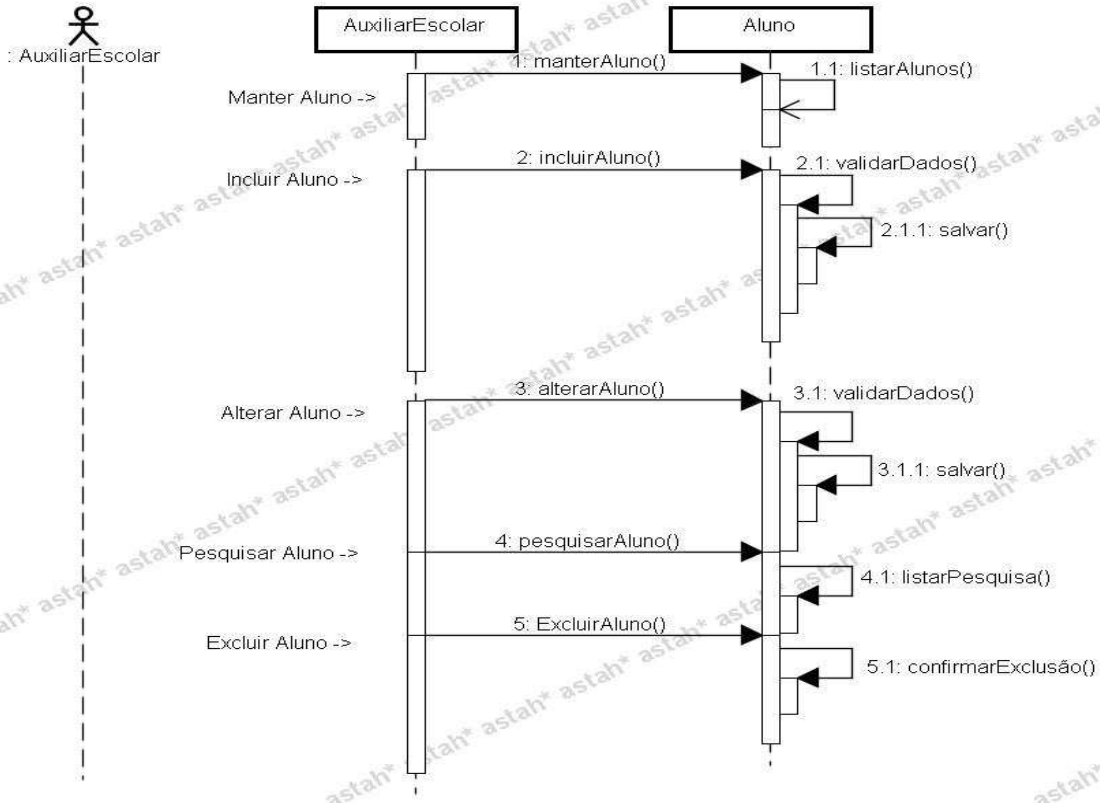
3.3.4.15 UC07 - Manter Disciplina - Fluxo Básico



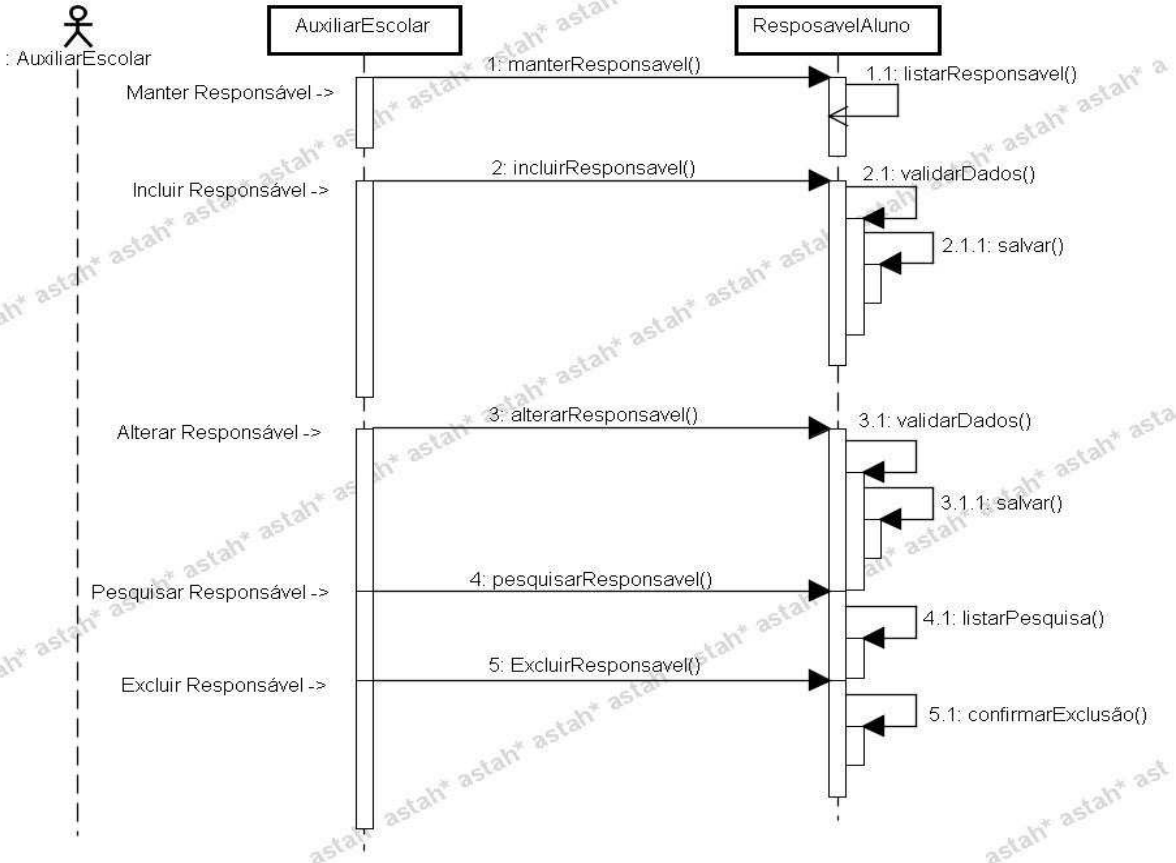
3.3.4.16 UC07 - Manter Disciplina - A1 - Nova Disciplina



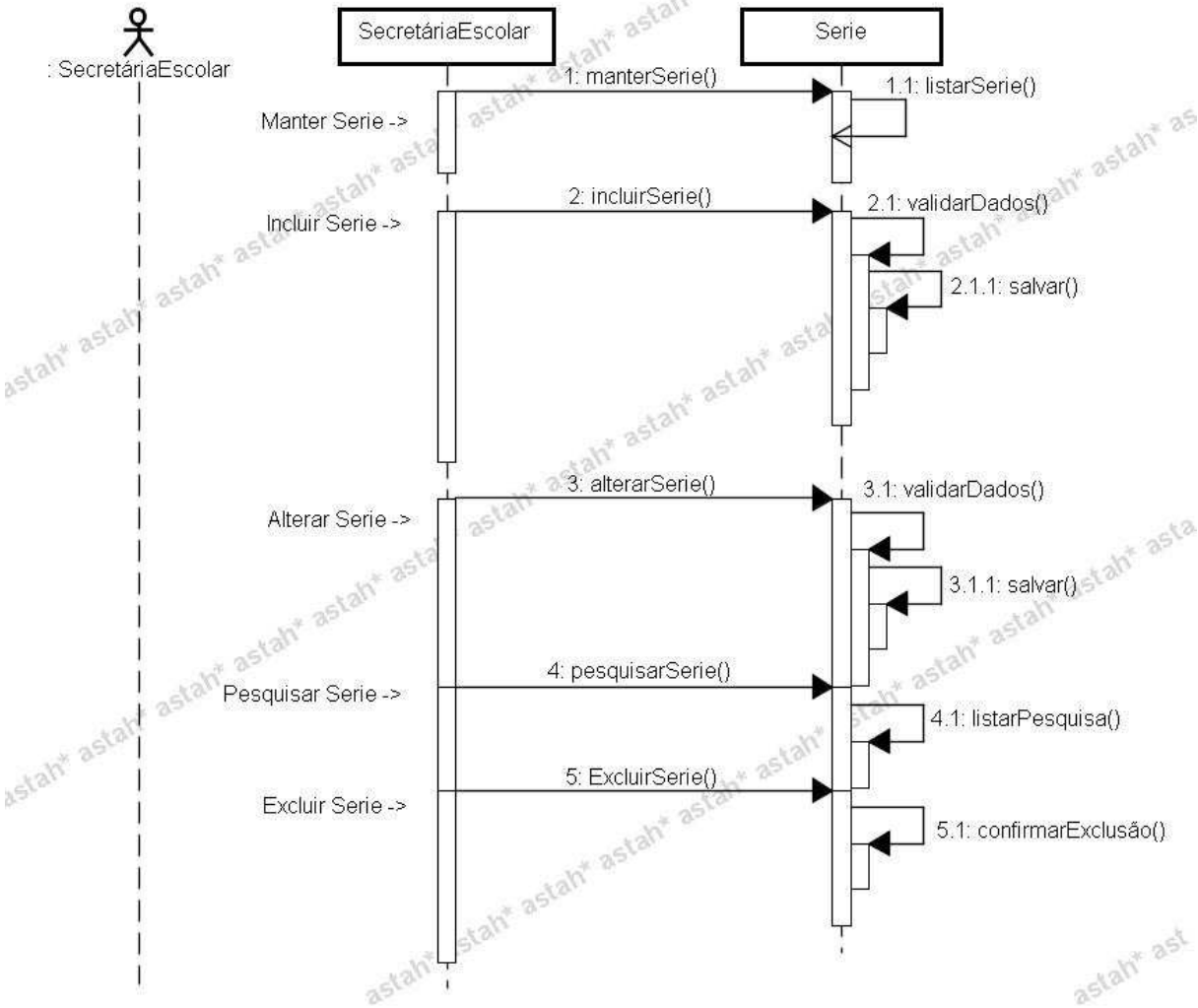
3.3.4.17 UC08 – Manter Aluno



3.3.4.18 UC09 – Manter Responsável



3.3.4.19 UC10 – Manter Série



3.4 Descrição de caso de uso

Essa descrição mostrará como o ator relacionara com o sistema.

Casos de uso:

3.4.1 UC_01_Manter Solicitações de Matrícula

Descrição

Este caso de uso visa incluir solicitações de matrícula, bem como pesquisar, alterar e excluir.

Atores

Responsável por Aluno;

Auxiliar Escolar;

Pré-condições

O ator deve estar “logado” no sistema.

Pós-condições

Não se aplica.

Fluxos de Eventos

Fluxo Básico

1 O caso de uso tem início quando o cadastrante seleciona a opção de Manter Solicitações de Matrícula.

2 O sistema solicita a identificação do responsável.

3 Caso exista segue para o próximo passo.

4 Caso não exista o sistema informa ao usuário que o responsável precisa ser previamente cadastrado. **[MSG01] [MSG08]**

5 O cadastrante informa a identificação do responsável. **[MSG01] [MSG02]**

6 O sistema exibe solicitação(s) de matricula caso exista(m). **[A1] [A2] [A3]**

7 Caso o código de expiração do responsável por aluno tenha vencido, são desabilitados os fluxos A2 e A3 e habilitado o fluxo A5. **[RN1.3] [MSG09]**

8 O fluxo se encerra.

Fluxos Alternativos

A1 – Solicitar Matrícula

1 Este fluxo alternativo é iniciado quando o cadastrante seleciona a opção Solicitar Matrícula.

2 O sistema recupera as escolas para seleção. **[MSG02] [A5]**

3 O cadastrante seleciona uma escola. **[MSG02]**

4 O sistema recupera as séries da escola. **[MSG02]**

5 O cadastrante seleciona uma série.

6 O sistema verifica se existe(m) vaga(s) de solicitações. **[RN1.1]**

7 Caso exista o sistema solicita que seja cadastrado o candidato a aluno. Segue para o próximo passo deste fluxo alternativo. **[MSG03]**

8 O cadastrante cadastra o candidato a aluno.

9 O cadastrante solicita que seja incluída a solicitação de matrícula. **[RN1.2] [MSG02]**

10 O sistema exibe a mensagem de sucesso. [MSG04]

11 O sistema gera o código de expiração de matrícula.

12 O fluxo se encerra.

A2 – Alterar Solicitação de Matrícula

1 Este fluxo alternativo é iniciado quando o cadastrante seleciona a opção de alterar uma solicitação de matrícula.

2 O sistema abre os dados da solicitação em modo de edição. **[A5]**

3 O cadastrante altera as informações.

4 O cadastrante confirma a alteração. **[RN1.2] [MSG02] [MSG05]**

5 O sistema exibe a mensagem de sucesso. **[MSG06]**

6 O fluxo se encerra.

A3 – Excluir Solicitação de Matricula

1 Este fluxo alternativo é iniciado quando o cadastrante seleciona a opção para Excluir uma solicitação de Matrícula.

2 O sistema solicita a confirmação da intenção de excluir.

3 Caso confirme segue para o próximo passo. **[MSG02]**

4 Caso não confirme segue para o passo 4 do fluxo básico.

5 O sistema exibe a mensagem de sucesso. **[MSG07]**

6 O fluxo se encerra.

A4 - Cancelar Edição

1 O cadastrante seleciona a opção de cancelar.

2 O sistema desconsidera alguma alteração que tenha ocorrido com as informações exibidas.

3 O sistema retorna para o seguinte passo 4 do fluxo básico.

4 O fluxo se encerra.

A5 – Renovar Solicitação de matricula

1 O cadastrante seleciona a opção de Renovar Solicitação de Matrícula. **[MSG02]**

2 O sistema segue para o passo 2 do A1.

3 O fluxo se encerra.

Relacionamento com Outros Casos de Usos

Casos de Uso de Inclusão:

Não se aplica

Casos de Uso de Extensão

Não se aplica.

3.4.2 UC_02_Gerar vaga de Matrícula

Descrição

Este caso de uso gera vaga de matrícula.

Atores

Responsável por Aluno;

Auxiliar Escolar;

Pré-condições

O ator deve estar “logado” no sistema.

Pós-condições

Não se aplica.

Fluxos de Eventos

Fluxo Básico

1 O caso de uso tem início quando é solicitada a geração de uma vaga.

2 O sistema atualiza aumentando a quantidade de vagas.

3 O sistema mostra a mensagem. **[MSG07]**

4 O caso de uso se encerra.

Fluxos Alternativos

Não se aplica.

Relacionamento com Outros Casos de Usos

Casos de Uso de Inclusão

Não se aplica.

Casos de Uso de Extensão

Não se aplica

3.4.3 UC_03_Manter Candidato a Aluno

Descrição

Este caso de uso visa cadastrar Candidato a aluno, bem como pesquisar, alterar e excluir.

Atores

Auxiliar Escolar;

Pré-condições

O ator deve estar “logado” no sistema.

Pós-condições

Não se aplica.

Fluxos de Eventos

Fluxo Básico

1 O caso de uso tem início quando o ator seleciona a opção de Manter Candidato.

2 O sistema solicita o preenchimento do CPF responsável. **[MSG01]**

3 O sistema lista o candidato a aluno. **[A1] [A2] [A3] [A4]**

4 O fluxo se encerra.

Fluxos Alternativos

A1 – Novo Candidato a Aluno

1 Este fluxo alternativo é iniciado quando o ator seleciona a opção cadastrar novo aluno.

2 O sistema habilita os campos para preenchimento de informações.

3 O ator preenche as informações supracitadas.

4 O ator solicita que seja salva as informações.

5 As informações são incluídas.

6 O sistema mostra mensagem de sucesso. **[MSG11]**

7 O fluxo se encerra.

A2 – Alterar Candidato a Aluno

1 Este fluxo alternativo é iniciado quando o cadastrante seleciona a opção de Alterar Aluno.

2 O sistema abre os dados em edição para serem alterados.

3 O ator preenche as informações supracitadas.

4 O ator confirma a alteração. **[MSG02]**

5 O sistema mostra a mensagem de sucesso. **[MSG11]**

6 O fluxo se encerra.

A3 – Pesquisar Candidato a aluno

1 Este fluxo alternativo é iniciado quando o cadastrante preenche os filtros e seleciona a opção Pesquisar Candidato a aluno.

2 O sistema retorna a pesquisa solicitada de acordo com os filtros preenchidos. **[MSG01] [MSG02]**

3 O fluxo se encerra.

A4 – Excluir Candidato a aluno

1 O cadastrante seleciona a opção de Excluir Candidato a aluno.

2 O sistema consiste a intenção do cadastrante em excluir.

2.1 Caso seja sim segue para o próximo passo

2.2 Caso seja não volta ao passo 3 do fluxo básico.

3 O sistema mostra mensagem de sucesso. **[MSG07]**

4 O fluxo se encerra.

Relacionamento com Outros Casos de Usos

Casos de Uso de Inclusão

Não se aplica

Casos de Uso de Extensão

Não se aplica.

3.4.4 UC_04_Manter Matrícula

Descrição

Este caso de uso visa fazer matrícula, bem como pesquisar, alterar e excluir.

Atores

Auxiliar Escolar;

Pré-condições

O ator deve estar “logado” no sistema.

Pós-condições

Não se aplica.

Fluxos de Eventos

Fluxo Básico

1 O caso de uso tem início quando o ator seleciona a opção de Manter Matricula.

2 O sistema solicita o preenchimento do CPF responsável. **[MSG01]**

3 O sistema lista matricula(s) caso existam. **[A2] [A4] [A5]**

4 O sistema habilita as opções de matrícula. **[A1] [A3]**

5 O fluxo se encerra.

Fluxos Alternativos

A1 – Fazer Matrícula

1 Este fluxo alternativo é iniciado quando o ator seleciona a opção Fazer Matrícula.

2 O ator preenche a informação supracitada.

3 O sistema lista a(s) solicitação(s) de matrícula para o responsável. **[MSG01] [MSG02]**

4 O ator seleciona uma solicitação de matrícula.

5 O sistema solicita o preenchimento do código de expiração de solicitação de matricula.

6 O sistema verifica se o código expirou o seu tempo de validade.

7 Caso tenha expirado mostra mensagem e volta para o passo 1 do fluxo básico. **[MSG09]**

8 Caso não tenha expirado o sistema solicita o preenchimento das informações do aluno e segue para o próximo passo.

9 As informações são incluídas. **[RN5.1] [MSG02] [MSG10]**

10 O fluxo se encerra.

A2 – Renovar Matrícula

1 Este fluxo alternativo é iniciado quando o cadastrante seleciona a opção de Renovar Matrícula.

2 O sistema habilita os campos de dados para atualizar as informações.

3 O ator confirma a inclusão. **[MSG02]**

4 O sistema mostra a mensagem de sucesso. **[MSG11]**

5 O fluxo se encerra.

A3 – Pesquisar Matrícula

1 Este fluxo alternativo é iniciado quando o cadastrante preenche os filtros e seleciona a opção Pesquisar Matrícula.

2 O sistema retorna a pesquisa solicitada de acordo com os filtros preenchidos. **[MSG01] [MSG02]**

3 O fluxo se encerra.

A4 - Alterar Matrícula

1 Este fluxo alternativo é iniciado quando o cadastrante seleciona a opção de alterar matrícula na lista de matrícula(s). **[MSG02]**

2 O sistema abre os dados em edição para serem alterados.

3 O ator preenche as informações supracitadas. **[RN5.1] [MSG10]**

4 O ator confirma a alteração. **[MSG02]**

5 O sistema mostra a mensagem de sucesso. **[MSG11]**

6 O fluxo se encerra.

A5 – Excluir Matrícula

1 O cadastrante seleciona a opção de Excluir Matrícula.

2 O sistema verifica a série do aluno.

3 Caso a série seja: “9° ano "Reprovado" ou outras séries”, o sistema solicita ao ator que gere uma evasão. Segue para o próximo passo deste fluxo alternativo. **[RN5.2]**

4 Caso a série seja: “9° ano Aprovado” segue para o passo [3] deste fluxo alternativo.

5 O sistema solicita a confirmação. **[MSG14]**

6 Caso seja confirmada a exclusão segue para o próximo passo deste fluxo.

7 Caso não confirme segue para o passo 3 do fluxo básico.

8 O sistema gera uma vaga de matrícula.

9 O sistema mostra mensagem de sucesso. **[MSG07]**

10 O fluxo se encerra.

Relacionamento com Outros Casos de Usos

Casos de Uso de Inclusão

Não se aplica.

Casos de Uso de Extensão

Não se aplica.

3.4.5 UC_05_Manter Escola

Descrição

Este caso de uso visa incluir escola, bem como pesquisar, alterar e excluir.

Atores

Secretária Escolar;

Pré-condições

O ator deve estar “logado” no sistema.

Pós-condições

Não se aplica.

Fluxos de Eventos

Fluxo Básico

1 O caso de uso tem início quando o ator seleciona a opção de Manter Escola.

2 O sistema lista as escolas(s). **[MSG01] [MSG02] [A1] [A2] [A3] [A4]**

3 O fluxo se encerra.

Fluxos Alternativos

A1 – Incluir Escola

1 Este fluxo alternativo é iniciado quando o ator seleciona a opção Incluir Escola.

2 O sistema solicita o preenchimento das informações da escola.

3 O ator preenche a informações supracitadas.

4 O ator solicita que as informações sejam incluídas.

5 As informações são incluídas. **[MSG02] [MSG10]**

6 O sistema mostra a mensagem de sucesso. **[MSG04]**

7 O fluxo se encerra.

A2 - Alterar Escola

1 Este fluxo alternativo é iniciado quando o cadastrante seleciona a opção de Alterar Escola. **[MSG02]**

2 O sistema abre os dados em edição para serem alterados. **[MSG01]**

3 O ator altera as informações supracitadas.

4 O ator confirma a alteração. **[MSG02]**

5 O sistema mostra a mensagem de sucesso. **[MSG06]**

6 O fluxo se encerra.

A3 – Pesquisar Escola

1 Este fluxo alternativo é iniciado quando o cadastrante preenche os filtros e seleciona a opção Pesquisar Escola.

2 O sistema retorna a pesquisa solicitada de acordo com os filtros preenchidos. **[MSG01] [MSG02]**

3 O fluxo se encerra.

A4 – Excluir Escola

1 O cadastrante seleciona a opção de Excluir Escola. **[MSG02]**

2 O sistema solicita a confirmação. **[MSG14]**

3 O ator confirma a exclusão.

4 O sistema mostra mensagem de sucesso. **[MSG07]**

5 O fluxo se encerra.

Relacionamento com Outros Casos de Usos

Casos de Uso de Inclusão

Não se aplica.

Casos de Uso de Extensão

Não se aplica.

3.4.6 UC_06_Gerar Evasão

Descrição

Este caso de uso visa gerar uma evasão.

Atores

Auxiliar Escolar;

Pré-condições

O ator deve estar “logado” no sistema.

Pós-condições

Não se aplica.

Fluxos de Eventos

Fluxo Básico

1 O caso de uso tem início quando ator deseja Gerar uma evasão para um aluno excluído anteriormente.

2 O sistema habilita as opções de preenchimento.

3 O ator preenche as informações supracitadas. **[RN5.2]**

4 O ator confirma a evasão. **[MSG02]**

5 O sistema gera a evasão.

6 O sistema mostra mensagem. **[MSG13]**

7 O fluxo se encerra.

Fluxos Alternativos

Não se aplica.

Relacionamento com Outros Casos de Usos

Casos de Uso de Inclusão

Não se aplica.

Casos de Uso de Extensão

Não se aplica.

3.4.7 UC_07_Manter Disciplina

Descrição

Este caso de uso visa cadastrar disciplina, bem como pesquisar, alterar e excluir.

Atores

Secretaria Escolar;

Pré-condições

O ator deve estar “logado” no sistema.

Pós-condições

Não se aplica.

Fluxos de Eventos

Fluxo Básico

1 O caso de uso tem início quando o ator seleciona a opção de Manter disciplina.

2 O sistema lista a(s) disciplina(s). **[A1] [A2] [A3] [A4]**

3 O fluxo se encerra.

Fluxos Alternativos

A1 – Nova Disciplina

1 Este fluxo alternativo é iniciado quando o ator seleciona a opção cadastrar nova disciplina.

2 O sistema habilita os campos para preenchimento de informações.

3 O ator preenche as informações supracitadas.

4 O ator solicita que seja salva as informações.

5 As informações são incluídas.

6 O sistema mostra mensagem de sucesso. **[MSG11]**

7 O fluxo se encerra.

A2 – Alterar Disciplina

1 Este fluxo alternativo é iniciado quando o cadastrante seleciona a opção de Alterar Disciplina.

2 O sistema abre os dados em edição para serem alterados.

3 O ator preenche as informações supracitadas.

4 O ator confirma a alteração. **[MSG02]**

5 O sistema mostra a mensagem de sucesso. **[MSG11]**

6 O fluxo se encerra.

A3 – Pesquisar Disciplina

1 Este fluxo alternativo é iniciado quando o cadastrante preenche os filtros e seleciona a opção Pesquisar Disciplina.

2 O sistema retorna a pesquisa solicitada de acordo com os filtros preenchidos. **[MSG01] [MSG02]**

3 O fluxo se encerra.

A4 – Excluir Disciplina

1 O cadastrante seleciona a opção de Excluir Disciplina.

2 O sistema consiste a intenção do cadastrante em excluir.

2.1 Caso seja sim segue para o próximo passo

2.2 Caso seja não volta ao passo 2 do fluxo básico.

3 O sistema mostra mensagem de sucesso. **[MSG07]**

4 O fluxo se encerra.

Relacionamento com Outros Casos de Usos

Casos de Uso de Inclusão

Não se aplica

Casos de Uso de Extensão

Não se aplica.

3.4.8 UC_08_Manter Aluno

Descrição

Este caso de uso visa incluir aluno, bem como pesquisar, alterar e excluir.

Atores

Auxiliar Escolar;

Pré-condições

O ator deve estar “logado” no sistema.

Pós-condições

Não se aplica.

Fluxos de Eventos

Fluxo Básico

1 O caso de uso tem início quando o ator seleciona a opção de Manter Aluno.

2 O sistema lista o(s) aluno(s). **[MSG01] [MSG02] [A1] [A2] [A3] [A4]**

3 O fluxo se encerra.

Fluxos Alternativos

A1 – Incluir Aluno

1 Este fluxo alternativo é iniciado quando o ator seleciona a opção Incluir Aluno.

2 O sistema solicita o preenchimento das informações de aluno

3 O ator preenche as informações supracitadas.

4 O ator solicita que as informações sejam incluídas.

5 As informações são incluídas. **[MSG02]**

6 O sistema mostra mensagem de sucesso. **[MSG02] [MSG04]**

7 O fluxo se encerra.

A2 - Alterar Aluno

1 Este fluxo alternativo é iniciado quando o cadastrante seleciona a opção de alterar aluno.

2 O sistema abre os dados em edição para serem alterados. **[MSG01] [MSG02]**

3 O ator preenche as informações supracitadas.

4 O ator confirma a alteração. **[MSG02]**

5 O sistema mostra a mensagem de sucesso. **[MSG06]**

6 O fluxo se encerra.

A3 – Pesquisar Aluno

1 Este fluxo alternativo é iniciado quando o cadastrante preenche os filtros e seleciona a opção Pesquisar Aluno.

2 O sistema retorna a pesquisa solicitada de acordo com os filtros preenchidos. **[MSG01] [MSG02]**

3 O fluxo se encerra.

A4 – Excluir Aluno

1 O cadastrante seleciona a opção de Excluir Aluno.

2 O sistema solicita a confirmação. **[MSG14]**

3 O ator confirma a exclusão

4 O sistema mostra mensagem de sucesso. **[MSG07]**

5 O sistema retorna para o passo 2 do fluxo básico.

6 O fluxo se encerra.

Relacionamento com Outros Casos de Usos

Casos de Uso de Inclusão

Não se aplica.

Casos de Uso de Extensão

Não se aplica.

3.4.9 UC_09_Manter Responsável

Descrição

Este caso de uso visa incluir Responsável, bem como pesquisar, alterar e excluir.

Atores

Auxiliar Escolar;

Pré-condições

O ator deve estar "logado" no sistema.

Pós-condições

Não se aplica.

Fluxos de Eventos

Fluxo Básico

1 O caso de uso tem início quando o ator seleciona a opção de Manter Responsável.

2 O sistema lista o(s) responsável (eis). **[MSG01] [MSG02] [A1] [A2] [A3] [A4]**

3 O fluxo se encerra.

Fluxos Alternativos

A1 – Incluir Responsável

1 Este fluxo alternativo é iniciado quando o ator seleciona a opção Incluir Responsável.

2 O sistema solicita o preenchimento das informações do responsável.

3 O ator preenche as informações supracitadas.

4 O ator solicita que as informações sejam incluídas.

5 As informações são incluídas. **[MSG02]**

6 O sistema mostra mensagem de sucesso. **[MSG02] [MSG04]**

7 O fluxo se encerra.

A2 - Alterar Responsável

1 Este fluxo alternativo é iniciado quando o cadastrante seleciona a opção de alterar responsável.

2 O sistema abre os dados em edição para serem alterados. **[MSG01] [MSG02]**

3 O ator preenche as informações supracitadas.

4 O ator confirma a alteração. **[MSG02]**

5 O sistema mostra a mensagem de sucesso. **[MSG06]**

6 O fluxo se encerra.

A3 – Pesquisar Responsável

1 Este fluxo alternativo é iniciado quando o cadastrante preenche os filtros e seleciona a opção Pesquisar responsável.

2 O sistema retorna a pesquisa solicitada de acordo com os filtros preenchidos. **[MSG01] [MSG02]**

3 O fluxo se encerra.

A4 – Excluir Responsável

1 O cadastrante seleciona a opção de Excluir Responsável.

2 O sistema solicita a confirmação. **[MSG14]**

3 O ator confirma a exclusão

4 O sistema mostra mensagem de sucesso. **[MSG07]**

5 O sistema retorna para o passo 2 do fluxo básico.

6 O fluxo se encerra.

Relacionamento com Outros Casos de Usos

Casos de Uso de Inclusão

Não se aplica.

Casos de Uso de Extensão

Não se aplica.

3.4.10 UC_10_Manter Série

Descrição

Este caso de uso visa incluir série, bem como pesquisar, alterar e excluir.

Atores

Secretária Escolar;

Pré-condições

O ator deve estar “logado” no sistema.

Pós-condições

Não se aplica.

Fluxos de Eventos

Fluxo Básico

1 O caso de uso tem início quando o ator seleciona a opção de Manter Série.

2 O sistema lista a(s) série(s). **[MSG01] [MSG02] [A1] [A2] [A3] [A4]**

3 O fluxo se encerra.

Fluxos Alternativos

A1 – Incluir Série

1 Este fluxo alternativo é iniciado quando o ator seleciona a opção Incluir Série.

2 O sistema solicita o preenchimento das informações de Série

3 O ator preenche a informações supracitadas.

4 O ator solicita que as informações sejam incluídas.

5 As informações são incluídas. **[MSG02]**

6 O sistema mostra mensagem de sucesso. **[MSG02] [MSG04]**

7 O fluxo se encerra.

A2 - Alterar Série

1 Este fluxo alternativo é iniciado quando o cadastrante seleciona a opção de alterar Série.

2 O sistema abre os dados em edição para serem alterados. **[MSG01] [MSG02]**

3 O ator preenche as informações supracitadas.

4 O ator confirma a alteração. **[MSG02]**

5 O sistema mostra a mensagem de sucesso. **[MSG06]**

6 O fluxo se encerra.

A3 – Pesquisar Série

1 Este fluxo alternativo é iniciado quando o cadastrante preenche os filtros e seleciona a opção Pesquisar Série.

2 O sistema retorna a pesquisa solicitada de acordo com os filtros preenchidos. **[MSG01] [MSG02]**

3 O fluxo se encerra.

A4 – Excluir Série

1 O cadastrante seleciona a opção de Excluir Série.

2 O sistema solicita a confirmação. **[MSG14]**

3 O ator confirma a exclusão

4 O sistema mostra mensagem de sucesso. **[MSG07]**

5 O sistema retorna para o passo 2 do fluxo básico.

6 O fluxo se encerra.

Relacionamento com Outros Casos de Usos

Casos de Uso de Inclusão

Não se aplica.

Casos de Uso de Extensão

Não se aplica.

3.5 Regras de negócio

Essas regras serão colocadas por caso de uso, entretanto, outros casos de uso podem reaproveitar as mesmas de outro caso de uso.

Finalidade

As regras de negócio mostram as restrições e imposições do negócio do cliente.

Escopo

Foram feitas as regras de negocio dos principais casos de uso, podendo serem aproveitadas em outros casos de uso. Os casos de uso são: UC-01 – Manter Solicitações de Matrícula e UC_04 - Manter Matrícula.

3.5.1 UC_01 – Manter Solicitações de Matrícula

[RN1.1] – Vagas de Solicitações

Só será permitido realizar uma solicitação de matrícula caso a quantidade de solicitações anteriores não tiverem atingido o limite máximo de vagas predeterminadas.

[RN1.2] – Atomicidade de Solicitação de Matrícula

O cadastrante só poderá fazer uma solicitação para um aluno em apenas uma escola, não permitindo que um mesmo aluno possa pertencer a mais de uma solicitação de matrícula em outras escolas.

[RN1.3] – Código de expiração

Esse código de expiração serve para que o responsável por aluno tenha um prazo predeterminado na sua solicitação de matrícula para poder entrar com a intenção de matricular o aluno. Caso vença esse prazo a solicitação de matrícula será considerada nula, não podendo ser feita a matrícula para esta solicitação enquanto não for renovado esse código de expiração.

3.5.2 UC_04 - Manter Matrícula

[RN5.1] – Aluno já Matriculado

Não será permitido o cadastro do mesmo aluno mais de uma vez na mesma escola. E não será permitido o cadastro do mesmo aluno em mais de uma escola.

[RN5.2] - Evasão

Todo aluno que tiver uma matrícula excluída será gerado uma evasão mostrando o motivo pelo quão o aluno está sendo “desligado” da escola, exceto, aqueles que terminarem o ensino básico.

3.6 Glossário de mensagens

Nessa parte será mostrada a lista de mensagens usadas nos casos de uso.

Lista de mensagens:

[MSG01] – Não foi encontrado nenhum resultado!

[MSG02] – Erro ao acessar a base de dados!

[MSG03] – Não há vagas!

[MSG04] – Salvo com Sucesso!

[MSG05] – Já existe uma Solicitação de Matrícula para este aluno!

[MSG06] – Alterado Com Sucesso!

[MSG07] – Excluído com Sucesso!

[MSG08] – Antes precisa cadastrar o responsável!

[MSG09] – Código de expiração vencido! Renove Sua Solicitação de Matrícula!

[MSG10] – Aluno já se encontra cadastrado!

[MSG11] – Matrícula renovada com sucesso!

[MSG12] – Uma vaga foi gerada!

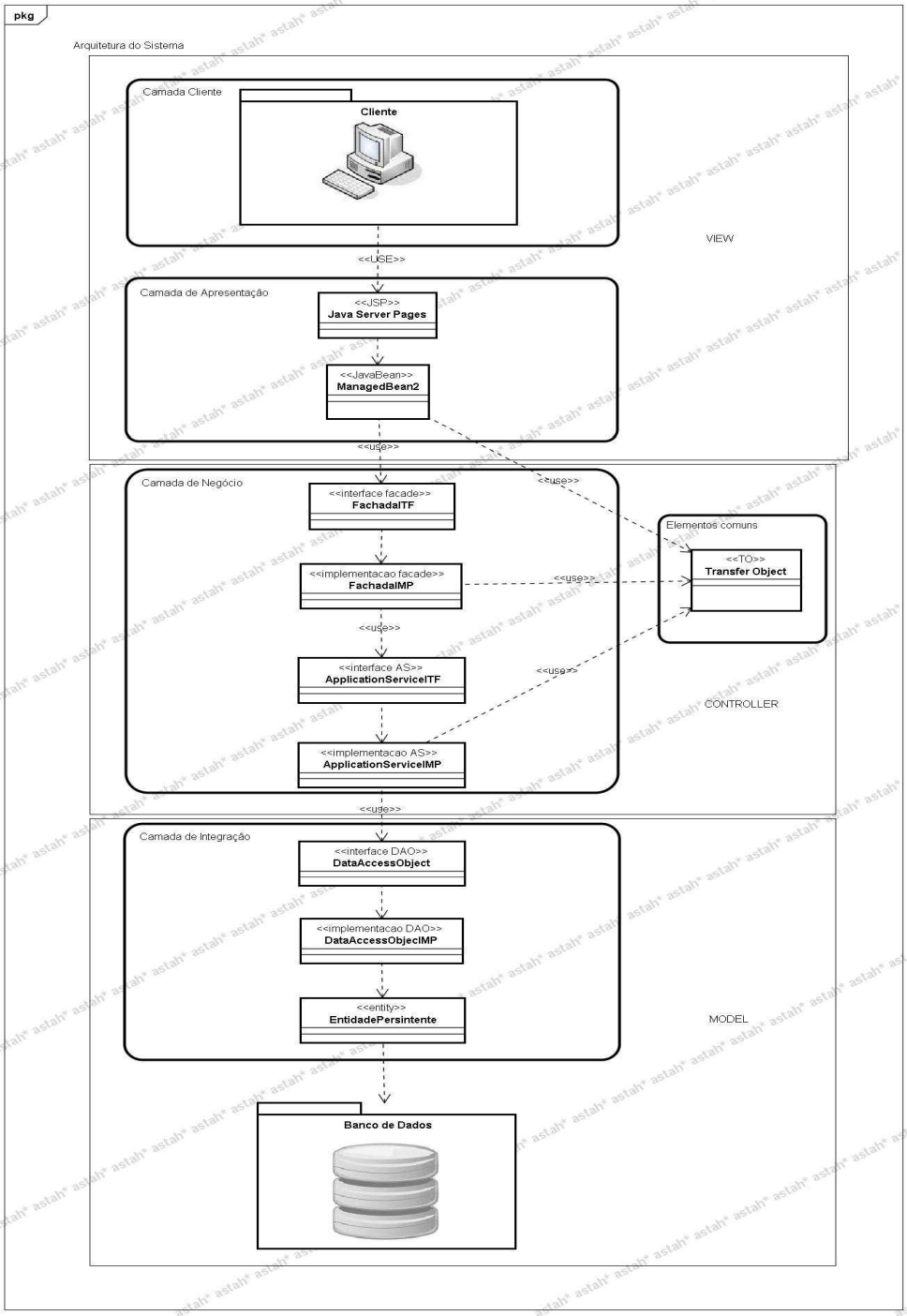
[MSG13] – Uma evasão foi gerada!

[MSG14] – Deseja realmente excluir!

[MSG15] – Candidato a aluno já se encontra cadastrado!

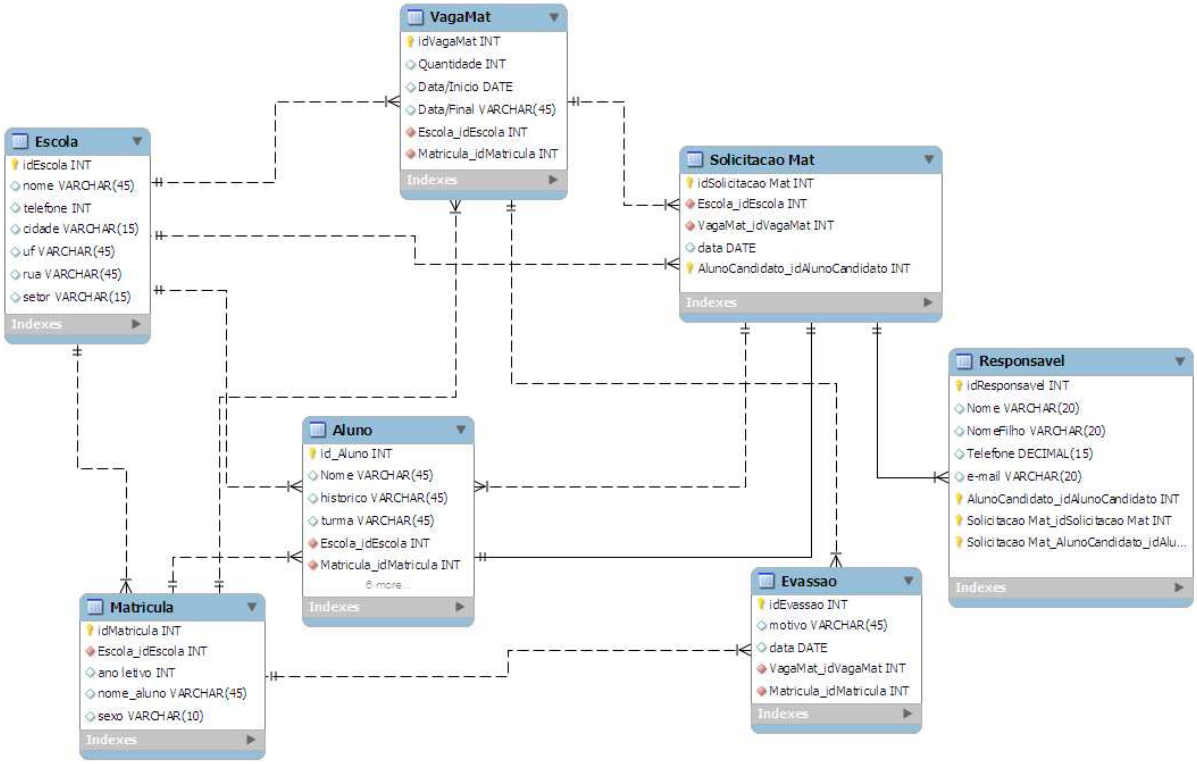
3.7 Arquitetura do Sistema.

Essa arquitetura visa mostrar a implementação do *MVC.* Nesse modelo pode ser visto como as camadas irão se comunicar, desde o navegador do cliente até o banco de dados.



3.8 Diagrama Entidade Relacionamento.

Nessa parte podem ser visualizadas as tabelas do banco de dados relacionadas.



Considerações Finais

Construir a análise não é uma tarefa corriqueira, pois, é preciso que técnicas, ferramentas, soluções e muito conhecimento intelectual para que possa ter um produto com qualidade, que atenda as necessidades dos usuários. Por essa verdade que se deu a elaboração desse trabalho.

Foram mostradas as definições de engenharia de software para basear para que a análise pudesse ser feita.

Juntamente com o framework Scrum a rapidez e a qualidade são percebidas para poder construir a análise, pois, não se presa à documentação "volumosa" ou que raramente será usada.

Com o Scrum foram obtidas as funcionalidades e priorizadas para que possam ser desenvolvidas. Em seguida artefatos foram construídos para poder ser compreendidas em um nível mais baixo de abstração.

Esse trabalho tem um interesse de deixar conhecimentos teóricos e práticos para a comunidade acadêmica de um projeto gerenciado pelo Scrum.

Referências Bibliográficas

ARAÚJO, Rodrigo. **Estudo sobre Engenharia de Software**. Disponível em: http://imasters.com.br/artigo/7909/linguagens/estudo-sobre-engenharia-de-software. Acesso em: 10 de dezembro 2011.

BEZERRA, Eduardo, **Princípios de Análise e Projeto de Sistemas com UML**. 2º Edição. Campus, Ano 2007.

BLAIS, Steve. **Agile Project Management**. Disponível em: http://www.4pm.pl/artykul/agile_project_management_fear_of_agile_by_steve_blais_pmp-45-1333.html. Acesso em: 10 de dezembro 2011.

BOOCH; RUMBAUGH e JACOBSON. **Guia do Usuário.** Tradução da 2ª Edição. Editora Campus, Ano 2005.

CARLOS, José Macoratti. **Padrões de Projeto: O modelo MVC - Model View Controller**. Acesso em: http://www.macoratti.net/vbn_mvc.htm.

COHN, Mike. **An Overview of Scrum for Agile Software Development**. Disponível em: http://www.mountaingoatsoftware.com/scrum/overview. Acesso em: 16 de dezembro de 2011.

DATE, C. J, **Introdução a sistemas de Banco de Dados**. (tradução da 8º edição Americana). Editora CAMPUS, Ano 2004.

FOWLER, Martin. **Continuous Integration**. Disponível em: http://martinfowler.com/articles/continuousIntegration.html. Acesso em: 10 de dezembro 2011.

FOWLER, Martin; HINGHSMITH, Jim. **The Agile Manifesto**. Disponível em: http://andrey.hristov.com/fht-stuttgart/The_Agile_Manifesto_SDMagazine.pdf. Acesso em: 10 de dezembro 2011.

GAMMA, Erich et al. **Padrões de Projeto:** Soluções reutilizáveis de software Orientado a Objetos. Porto Alegre: Bookman, Ano 2000.

GONÇALVES, Edson, **Desenvolvendo Aplicações Web com Jsp, Servlets, Java server Faces, Hibernate, Ejb 3 Persistence e Ajax.** Rio de Janeiro: Editora Ciência Moderna Ltda., Ano 2007.

GUEDES, Gilleanes T. A, **UML 2 UMA ABORDAGEM PRÁTICA.** 3º edição. São Paulo: Editora Novatec, Ano 2009.

HIGHSMITH, Jim. **History: The Agile Manifesto**. Disponível em: http://agilemanifesto.org/history.html. Aceso em: 10 Setembro de 2011.

JACOBSON, Ivar et al. 1992. **Object-Oriented Software Engineering - A Use Case-Driven Approach**, Wokingham, Inglaterra: Addison Wesley Longman.

KNIBERG, Henrik. **Scrum e XP direto das Trincheiras Como fazemos Scrum.** Free online edition, Disponível em: http:/infoq.com/br/minibooks/scrum-xp-from-the-trenches. Acesso em: 10 Setembro de 2011.

ROYCE, W. W. Managing the Developmente of Large Software Systems: Concepts and Techniques. Proc. **IEEE Westcon, Los Angeles, CA, Ano 1970.**

SCHWABER, Ken. **Guia do Scrum**. Disponível em: http://www.trainning.com.br/download/GUIA_DO_SCRUM.pdf. Acesso em: 10/12/2011.

SOMMERVILLE, **Engenharia de software.** 8º edição. São Paulo: Editora Pearson Education, Ano 2007.

TEOREY, Toby; LIGHTSTONE, Sam; NADEAU,Tom, **Projeto e Modelagem de Banco de Dados.** Tradução da Quarta Edição. Rio de Janeiro, Campus, Ano 2007.