

Faculdade de Tecnologia de Sorocaba

Tecnologia em Análise e Desenvolvimento de Sistemas

Metodologia customizada

Documentação

Prof. Sérgio Bernardo

Disciplina: Engenharia de Software III

Grupo AViS

Pedro Bernardo de Sousa 0030481711006

Vítor Andrade Marques da Silva 0030481511040

Weuller Júnior Souza Bessa 0030481621040

Sorocaba

1° Semestre de 2019

**RESUMO**

Neste trabalho, nós definimos uma metodologia customizada baseada em AUP, UML e o manifesto ágil. Também levamos em consideração nossa breve experiência como desenvolvedores de software.

**Palavras-chave**

\*\*\* Inserir texto correspondente!!!

\*\*\* Inserir texto correspondente!!!

\*\*\* Inserir texto correspondente!!!

**Lista de Figuras**

**Figura 1**

Símbolos

**Figura 2**

Ciclo de Vida

**Figura 3**

Processos

**Lista de Tabelas**

**Tabela 1**

Padrão

**Tabela 2**

Ciclos

**Tabela 3**

Grupos

Sumário

[INTRODUÇÃO 6](#_Toc12616205)

[Etapa: Planejamento (13 horas) 6](#_Toc12616206)

[Metodologias: 6](#_Toc12616207)

[Etapa: Construção (28 horas) 6](#_Toc12616208)

[Processos: 6](#_Toc12616209)

[Etapa: Transição (3 horas) 6](#_Toc12616210)

[Processos: 6](#_Toc12616211)

[QUESTÃO DE PESQUISA: 8](#_Toc12616212)

[OBJETIVOS DA PESQUISA 11](#_Toc12616213)

[Objetivo Geral 11](#_Toc12616214)

[Objetivos Específicos 11](#_Toc12616215)

[FUNDAMENTAÇÃO TEÓRICA 12](#_Toc12616216)

[PROCESSOS ORGANIZACIONAIS 12](#_Toc12616217)

[PROCESSO UNIFICADO RATIONAL (RUP) 12](#_Toc12616218)

[PROCESSO UNIFICADO ÁGIL (AUP) 13](#_Toc12616219)

[DIAGRAMAS UML 13](#_Toc12616220)

[Diagramas estruturais 13](#_Toc12616221)

[Diagramas de classe 14](#_Toc12616222)

[Diagramas de objetos 14](#_Toc12616223)

[Diagramas de componentes 15](#_Toc12616224)

[Diagramas de implementação ou instalação 15](#_Toc12616225)

[Diagramas de pacotes 15](#_Toc12616226)

# INTRODUÇÃO

Em seu desenvolvimento, as etapas a serem percorridas até a entrega da primeira versão da plataforma do projeto AViS:

## Etapa: Planejamento (13 horas)

### Metodologias:

Definição de objetivos do ciclo atual e a respectiva divisão de tarefas (GitHub) duração 2 horas;

Cronograma: (Trello/EleGantt) duração 1 hora;

Modelagem: (UML/BPMN) duração 10 horas;

## Etapa: Construção (28 horas)

### Processos:

Programar: (linguagem XXX) duração 18 horas;

Testes;

Exploratórios: (testes manuais) duração 3 horas;

Automatizados: (codificar bateria de testes completo) duração 5 horas;

Correções de bugs dos ciclos anteriores (linguagem XXX) duração 2 horas;

## Etapa: Transição (3 horas)

### Processos:

Reuniões com o orientador (Professor XXX) duração 1 hora;

Avaliação dos Deadlines (reunião dos membros) duração 2 horas;

# QUESTÃO DE PESQUISA:

A maneira como se desenvolvia software 20 anos atrás se aplica ao contexto tecnológico de hoje? Se não, em quê? O que se pode aproveitar de metodologias de desenvolvimento de software que herdamos de gerações precedentes e como integrá-las ao modo de funcionamento de uma equipe moderna, de forma que as lições duramente aprendidas no passado não precisem ser aprendidas de novo. A metodologia base será a AUP. Com isso, pretende-se manter e aplicar a recomendação de basear o desenvolvimento em ciclos curtos, com resultados concretos e tangíveis. Pretende-se também manter em mente os **12 Princípios do Manifesto Ágil**.

1. Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado.
2. Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente.
3. Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.
4. Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.
5. Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho.
6. O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face.
7. Software funcionando é a medida primária de progresso.
8. Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
9. Contínua atenção à excelência técnica e bom design aumenta a agilidade.
10. Simplicidade: a arte de maximizar a quantidade de trabalho não realizado é essencial.
11. As melhores arquiteturas, requisitos e designs emergem de equipes auto organizáveis.
12. Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

Tendo esses 12 princípios em vista e utilizando como base a metodologia AUP, nós decidimos que o nosso ciclo de desenvolvimento terá 2 semanas.

Nós pretendemos também incorporar na nossa metodologia alguns aspectos da TDD, Test Driven Development. Em TDD, o teste é frequentemente escrito antes daquilo que ele testa. Ao escrever o teste, a equipe frequentemente atravessa perguntas sobre o comportamento da unidade. É útil que essas perguntas sejam respondidas antes de implementar a unidade. O teste é equivalente a redundância desejada, por exemplo, em engenharia mecânica: se uma bomba de fluido do sistema hidráulico é muito importante e a segurança do operador seria ameaçada em caso de falha daquela peça, acrescenta-se uma segunda bomba. Em engenharia de software, esse tipo de redundância (duplicar o código) é absurda, visto que bugs no código de origem estarão presentes na cópia. Um teste, por sua vez, contempla apenas a interface da unidade lógica (função, classe, etc.), mas mantém total independência dos algoritmos usados dentro dela. Assim, avaliamos automaticamente o comportamento da unidade.

Assim, vamos progressivamente expandir o projeto, em ciclos de 2 semanas, sempre dando ênfase aos 3 pilares do projeto: a modelagem, o teste e o código em si.

Isso significa que, ao final de cada ciclo, nós pretendemos poder apresentar novos diagramas, testes e unidades lógicas, assim como refinamentos nos diagramas, testes e unidades lógicas existentes.

O cronograma deverá ser elaborado contemplando as 4 fases AUP (concepção, elaboração, construção e transição) 3 vezes. A primeira iteração do conjunto de fases será realizada ao longo do semestre corrente, e deverá ser concluída em meados de junho de 2019. A segunda iteração das 4 fases AUP dirá respeito à disciplina de Laboratório de Engenharia de Software, e deverá ser concluída em meados de dezembro de 2019. Naquele período do projeto, uma ênfase especial será dada à fase de construção, dada a natureza da disciplina. Finalmente, ao longo do último semestre do curso, nós entraremos em um período final do projeto, ao final do qual estima-se que haverá um produto devidamente apresentável ao mercado, que poderá ser colocado à disposição dos usuários.

Neste primeiro período, que já começou, nós pretendemos enfatizar a arquitetura do sistema, a integração com o backend existente, através de pequenas unidades de código capazes de demonstrar o uso das tecnologias que deverão ser usadas.

Em particular, deverá ser implementado até junho de 2019:

Um programa capaz de trocar mensagens remotamente, através do protocolo UDP, usando endereços IPv6;

Um programa capaz de renderizar um ambiente em 3D:

1. Um processo capaz de gerar código a partir de diagramas UML e vice-versa;
2. Um programa capaz de efetuar o login na RESTful API da Alloy;
3. Um programa capaz de baixar conteúdo disponibilizado pela RESTful API da Alloy;
4. Um programa capaz de ingerir o fluxo de áudio de um microfone;
5. Um programa capaz de ingerir o fluxo de vídeo de um webcam;
6. Um programa capaz de identificar as coordenadas de um rosto em um fluxo de imagens;
7. Um programa capaz de processar um fluxo de imagens (recortar, transformar em monocromático, etc.);

Obviamente, esses programas deverão ser acompanhados de uma breve documentação que registra o porquê de cada demonstrativo, no contexto do cliente AViS, isto é, deverá ser devidamente explorado, conceitualmente, porque o cliente AViS só será possível se essas tecnologias forem demonstradas.

# OBJETIVOS DA PESQUISA

## Objetivo Geral

Conhecer e aplicar padrões ao processo de software. Mapear modelos de representação. Integrar abordagens organizacionais e tecnológicas.

## Objetivos Específicos

Não temos aqui a pretensão de responder definitivamente as perguntas acima, mas temos antes o objetivo de estudar essas questões para trazer elementos de resposta, afim de nos prepararmos para o nosso futuro profissional, em um mercado borbulhante de inovações e, ao mesmo tempo dependente de disciplinas relativamente jovens, como Ciência da Computação e a Engenharia de Software. O mercado é desafiador e é de importância capital equipar-se o melhor possível com metodologias efetivas e inteligentes. Metodologias que não descartam lições aprendidas no passado nem ignoram a conjuntura tecnológica do presente.

Abordagem de Requisitos Organizacionais com a opção de Análise de Processos Gerenciais (Gestão por Processos e Gestão de Processos).

Considerar opções para projetos de software no contexto dos processos, que podem ser: Software Autônomo, Orientado por Processo e Automatizado por Processo.

No projeto do Software, utilizar Diagramas UML aprendidos em ES1 e ES2, completando e ampliando essas representações e aplicando o Processo Unificado (RUP) com abordagem Ágil (Agile Unified Process).

# FUNDAMENTAÇÃO TEÓRICA

É o que já existe, deveremos estudar e descrever os modelos atuais, antes de apresentar a nossa solução, quanto ao processo de desenvolvimento.

# PROCESSOS ORGANIZACIONAIS

\*\*\* Inserir texto correspondente!!!

\*\*\* Inserir texto correspondente!!!

\*\*\* Inserir texto correspondente!!!

# PROCESSO UNIFICADO RATIONAL (RUP)

RUP (Rational Unified Process) é um framework iterativo para desenvolvimento de software criado pela Rational Software, que é uma divisão da IBM. Não se trata de um método prescritivo concreto, mas um conjunto adaptável de conceitos, que podem ser seletivamente adotados por uma equipe de desenvolvimento, conforme suas necessidades. RUP é uma implementação de marca registrada do Processo Unificado proposto, que é mais genérico. UP, *Unified Process*, pode, ocasionalmente, ser usado no lugar de RUP para de evitar a infração de direitos autorais da IBM.

Ao longo dos anos 90, a Rational Software incorporou diversos elementos ao RUP, a medida que foi adquirindo outras empresas. Entre 2000 e 2003, mudanças foram introduzidas para permitir a adaptação do método com base em necessidades práticas, encontradas pelas equipes durante o uso da metodologia, por exemplo, foram introduzidos conceitos e técnicas de metodologias como XP (Extreme Programming), entre outras, que coletivamente seriam chamadas de metodologias Ágil. Entre as novas ideias que foram acrescentadas ao RUP, figura o chamado TDD, ou Test Driven Development, que dá mais destaque ao teste, não apenas para verificar a implementação correta de uma unidade de código, mas também para ajudar a equipe a definir com precisão determinística o que se espera de cada unidade.

# PROCESSO UNIFICADO ÁGIL (AUP)

AUP (Agile Unified Process) é uma versão simplificada do RUP desenvolvida por Scott Ambler, considerando técnicas e conceitos ágeis utilizando técnicas de desenvolvimento orientado a testes (TDD) onde o ciclo de desenvolvimento é mais curto e os requisitos são transformados em casos de teste, sendo assim, em vez de entregar o software de uma vez, são liberadas pequenas partes, o AUP utiliza também de modelagem ágil (AM) já que o ciclo é mais curto, a modelagem ágil permite um ambiente com possíveis rápidas mudanças. No AUP as etapas de (implementação, teste e implantação) ocorrem partindo da segunda etapa, já na AUP estas são as três últimas etapas, sendo assim, RUP é uma metodologia mais estática (sequencial), já que a documentação precede o sistema, não havendo muitas alterações nas etapas finais.

# DIAGRAMAS UML

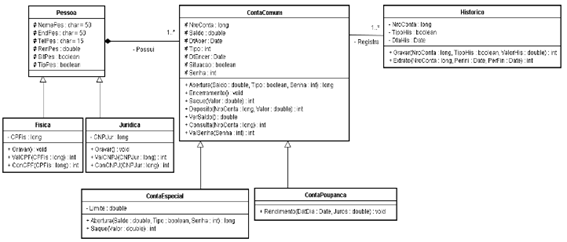
A sigla UML significa Linguagem de Modelagem Unificada utilizada para representar um sistema de forma padronizada. Não sendo uma metodologia, a UML tem como objetivos a especificação, documentação e estruturação por meio de diagramas representando o sistema como um todo.

Estes diagramas são classificados como, estruturais e comportamentais

## Diagramas estruturais

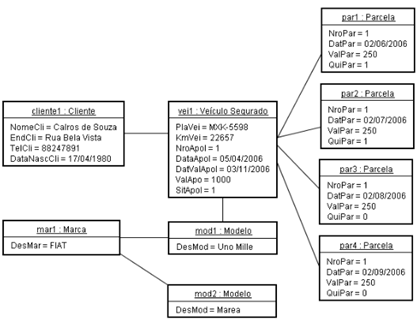
Tratam do ponto de vista do sistema quanto às classes, representando as estruturas estáveis.

### Diagramas de classe



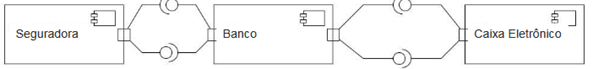
O Diagrama de Classes é utilizado para fazer a representação de estruturas de classes de negócio, interfaces e outros sistemas e classes de controle.

### Diagramas de objetos



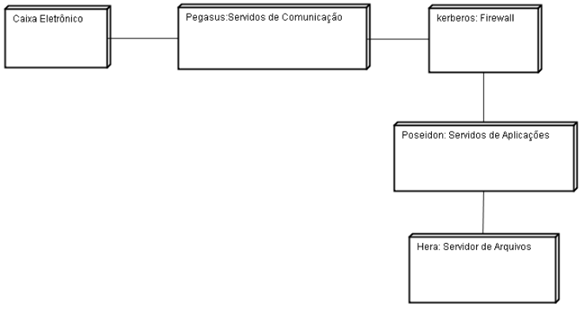
O diagrama de objetos representa os objetos de um diagrama de classes em um determinado instante de tempo, representando suas instâncias e seus relacionamentos, conforme definidos no diagrama de classes.

### Diagramas de componentes



Este diagrama mostra os artefatos de que os componentes são feitos, como arquivos de código fonte, bibliotecas de programação ou tabelas de bancos de dados.

### Diagramas de implementação ou instalação



O diagrama de implantação, consiste na organização do conjunto de elementos de um sistema para a sua execução.

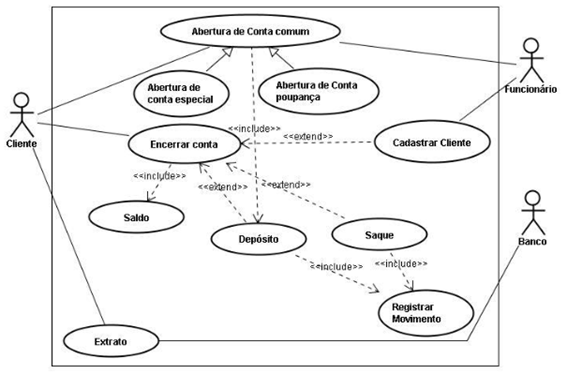
### Diagramas de pacotes

O diagrama de pacotes tem o objetivo de transformar as classes em pacotes. O critério para definir os pacotes é subjetivo e depende da visão e das necessidades do projetista. Este deve definir uma certa semântica e colocar os elementos similares e que tendem a serem modificados em conjunto num mesmo pacote. Como também, pode-se usar os pacotes para mostrar a arquitetura do sistema.

## Diagramas comportamentais

Utilizado para visualizar, especificar, construir e documentar aspectos dinâmicos, ou seja, que se alteram, como por exemplo, itens como o fluxo de mensagens.

### Diagrama de casos de uso



Um diagrama de casos de uso mostra um conjunto de casos de uso e atores, (um tipo especial de classe) e seus relacionamentos.

### Diagramas de sequência

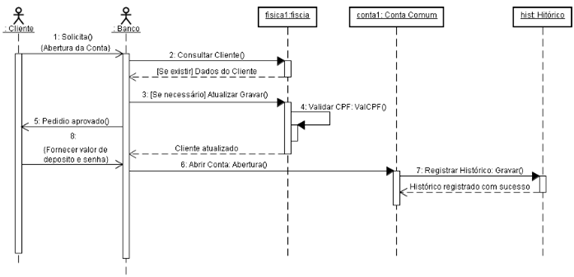


Diagrama de interação que dá ênfase à ordenação temporal de mensagens.

### Diagramas de colaboração

Um diagrama de comunicação é um diagrama de interação que dá ênfase a organização estrutural dos objetos que enviam e recebem mensagens.

### Diagramas de transição de estados

Mostra uma máquina de estados, que consiste de estados, transições, eventos e atividades. Usa-se o diagrama de estados para ilustrar a visão dinâmica de um sistema. Esses diagramas são importantes principalmente para fazer a modelagem do comportamento de uma interface, classe ou colaboração.

# ANÁLISE DE NEGÓCIOS (INÍCIO)

## SITUAÇÃO ATUAL

Um dos primeiros passos para se desenvolver uma solução em software é mapear a situação atual do negócio que se pretende melhorar. Isso deve ser feito através de entrevistas com stakeholders, especialmente pessoas que trabalham com proximidade das atividades relevantes. Anotações são tomadas ao longo dessa pesquisa inicial. Posteriormente, essas notas são usadas pela equipe para elaborar um diagrama de processos de negócio, em BPML. Ao longo dos esforços para determinar a Situação Atual, é preciso manter claro que se trata de mapeamento. Ideias sobre melhorias do sistema devem ser excluídas ativamente, visto que pensar a respeito delas é o impulso natural do desenvolvedor.

## PROBLEMAS

Com um mapeamento bem feito da situação atual, torna-se possível realizar a análise de eficiência e eficácia do processo de negócio. É importante identificar tudo aquilo que representa um espaço para melhoria ou um problema. Tudo que pode, potencialmente, ser feito mais rápido ou com menos recursos precisa ser apontado e catalogado. Inconveniências para pessoas que operam diretamente no processo (dificuldade de uso, etc.) devem ser apontadas também.

## SOLUÇÕES

Essa área do trabalho deve corresponder aos problemas levantados anteriormente. É preciso pensar em quais problemas podem ser endereçados mais facilmente para começar por eles. Ao propor soluções, frequentemente vamos incorporar o desenvolvimento e implantação de novos sistemas de informação, mas não necessariamente. Também é importante estimar, quantitativamente, o valor que será agregado ao negócio, quanto será economizado, quanto lucro haverá a mais, etc.

# SISTEMA DE INFORMAÇÕES (ELABORAÇÃO)

## CONTEXTO

\*\*\* Inserir texto correspondente!!!

\*\*\* Inserir texto correspondente!!!

\*\*\* Inserir texto correspondente!!!

## REQUISITOS NÃO FUNCIONAIS

Definem características que limitam o sistema, como por exemplo, tempo, hardware, sistema operacional, etc.

## REQUISITOS FUNCIONAIS

Define funções necessárias para que um sistema ou seu componente cumpra com suas tarefas de acordo com o objetivo da existência do software.

## DIAGRAMA DE CASO DE USO

Descreve a interação entre um humano e um sistema de forma que componentes específicos indiquem os processos e a quais atores eles estão ligados, sendo ator, um dos componentes do diagrama de caso de uso, fazendo referência ao usuário e suas possíveis ações, onde essas, no diagrama são os casos de uso representados por elipses.



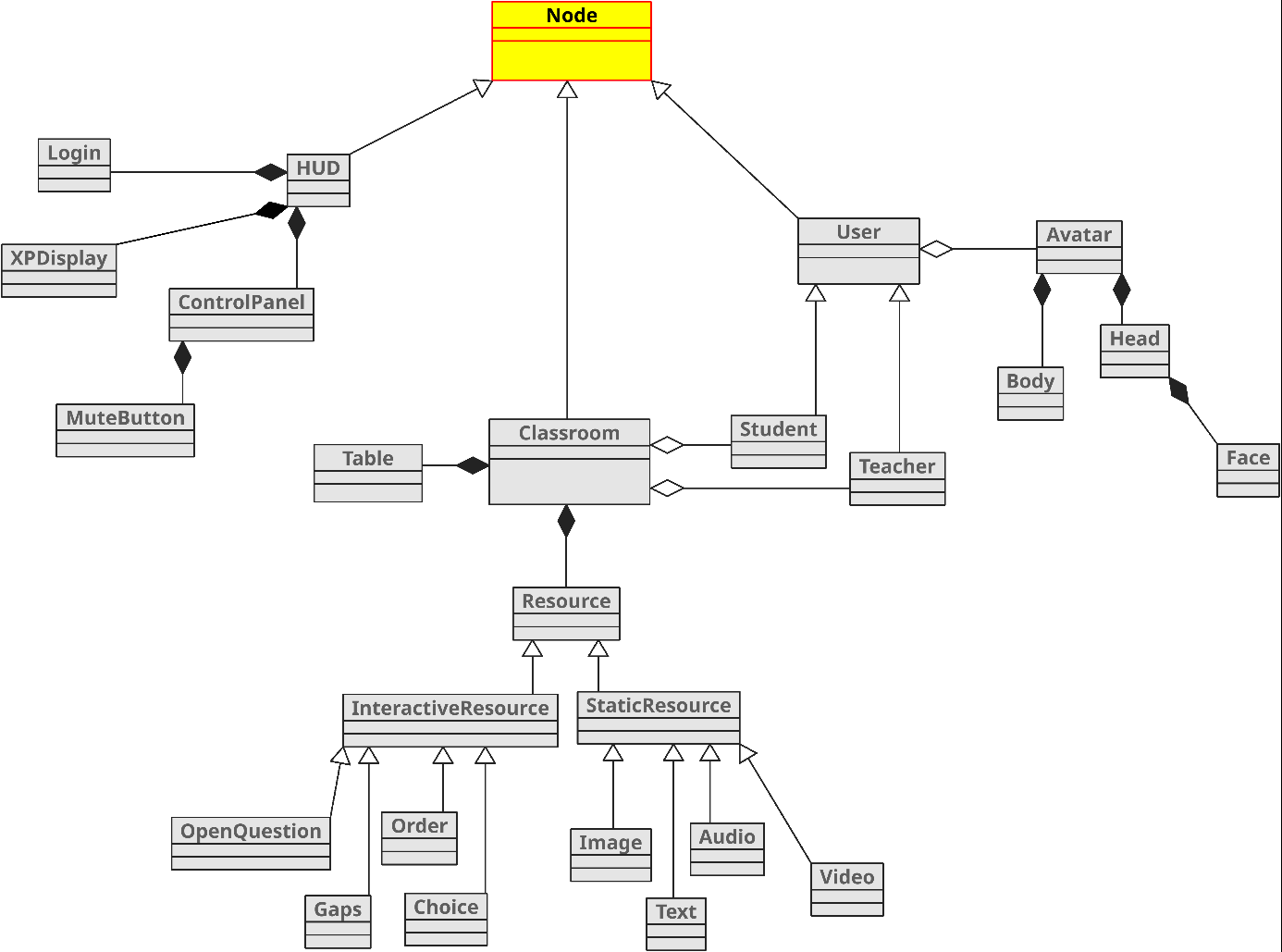
## CASOS DE USO DETALHADOS

UC001 – Nome do caso de uso
Objetivo:
Requisitos:
Atores:
Prioridade:
Pré-condições:
Frequência de uso:
Pós-condições:
Cam...

Fazem uma descrição detalhada de cada caso de uso dando assim um apoio para os desenvolvedores sendo o fluxo principal um guia de como se desenvolve a operação, pois fazem menção a prioridades e pré-condições, além de ter informações sobre objetivo, requisitos, atores, frequência de uso, pós-condições, fluxo principal / alternativo / exceções, validações e protótipos de telas.

## OUTROS DIAGRAMAS

### Diagrama de Classes



Mapeiam de forma clara a estrutura do sistema com o objetivo de modelar as classes utilizadas para o desenvolvimento, com este diagrama é possível ter informações técnicas que posteriormente se tornarão código.

# CONSIDERAÇÕES FINAIS

\*\*\* Inserir texto correspondente!!!

\*\*\* Inserir texto correspondente!!!

\*\*\* Inserir texto correspondente!!!

# REFERÊNCIAS

BIZAGI. ***Bizagi Modeler***. Disponível em:

<http://www.bizagi.com/pt/produtos/bpm-suite/modeler>

Acesso em: 16 de julho de 2018

NONAKA, Ikujiro e TAKEUCHI, Hirotaka.

Criação do Conhecimento na Empresa:

como as empresas geram a dinâmica da inovação.

Rio de Janeiro: Campus,1997.

OMG. “Business Process Model and Notation: BPMN.”

Versão 2.0 2011

PMI. “Guia do Conhecimento em Gerenciamento de Projetos: Guia PMBOK”. / 5 ed.

São Paulo: Saraiva, 2013.

PROBST, G.; RAUB, S.; ROMHARDT, K.

“**Gestão do Conhecimento** - Os Elementos Construtivos do Sucesso”

Porto Alegre: Bookman, 2002

WANKES, Leandro**. Ciclo de Vida de um Projeto**. Disponível em:

<<http://www.wankesleandro.com>>

Acesso em: 16 de julho de 2018

Medeiros, Higor. **Modelos de processos ágeis – Conceitos e Princípios**. Disponível em:

<https://www.devmedia.com.br/modelos-de-processos-ageis-conceitos-e-principios/30059>

Acesso em: 28 de junho de 2019

**Definição de RUP**. Disponível em:

<https://en.wikipedia.org/wiki/Rational_Unified_Process>

Acesso em: 28 de junho de 2019

<http://www.ambysoft.com/unifiedprocess/agileUP.html>

Acesso em: 28 de junho de 2019

<https://www.infoescola.com/engenharia-de-software/rup/>

Acesso em: 28 de junho de 2019

<https://homepages.dcc.ufmg.br/~figueiredo/disciplinas/aulas/uml-diagramas_v01-1.pdf>

Acesso em: 28 de junho de 2019

<https://pt.wikipedia.org/wiki/UML>

Acesso em: 28 de junho de 2019