FATEC
Faculdade de Tecnologia do Estado de São Paulo

Laboratório de Engenharia de Software

Título do projeto

LEIA COM ATENÇÃO

Este é apenas um modelo de projeto. Você deve substituir os títulos e o conteúdo com dados do seu próprio projeto.

IMPORTANTE

Com o tempo, novas partes serão anexadas a este mesmo documento, fazendo-o crescer.
Quando enviar um documento na data pedida pelo professor, este deverá conter todas as partes (corrigidas) até aquela data.
Salvar o documento em PDF com o nome: *NOME_EQUIPE*-LABENGSW-MA5-PARTE_*X*.pdf ou *NOME_EQUIPE*-LABENGSW-MA5-PARTE_*X*.pdf, onde *NOME_EQUIPE* é o nome da equipe que foi definido cadastrado e *X* é o número do documento que foi pedido.

APAGUE ESTE AVISO ANTES DE ENVIAR SEU DOCUMENTO!

Nome	**RA**
Nome 1	RA1
Nome 2	RA2
Nome 3	RA3
Nome4	RA4

Relatório de andamento do projeto

Aluno: nome do aluno 1 **RA:** RA do aluno 1

Data inicial	Data final	Horas	Descrição sucinta das atividades

Relatório de andamento do projeto

Aluno: nome do aluno 2 **RA:** RA do aluno 2

Data inicial	Data final	Horas	Descrição sucinta das atividades

Relatório de andamento do projeto

Aluno: nome do aluno 3 **RA:** RA do aluno 3

Data inicial	Data final	Horas	Descrição sucinta das atividades

Relatório de andamento do projeto

Aluno: nome do aluno 4 **RA:** RA do aluno 4

Data inicial	Data final	Horas	Descrição sucinta das atividades

Sumário

1. Planejamento do sistema

1.1. Descrição do projeto

Descrever o projeto em uma linguagem não técnica voltada a algum interessado no mesmo (diretor, acionista, usuário e outros).

1.2. Cronograma

Elaborar uma primeira proposta de cronograma de trabalho, baseado no programa apresentado das aulas da disciplina, prevendo possíveis atrasos. Utilizar alguma ferramenta de controle de projeto. Toda nova versão submetida ao professor deverá apresentar o cronograma atualizado. O cronograma pode ser uma carta de Gantt ou uma tabela de tarefas/datas.

1.3. Recursos

1.3.1. Recursos humanos

Descrever os integrantes da equipe do projeto e o número de horas de trabalho disponíveis por semana (dentro e fora da sala de aula);

1.3.2. Recursos físicos

Descrever os recursos materiais, tais como hardware e software para o desenvolvimento.

2. Especificação dos Requisitos do Sistema

2.1. O produto

2.1.1. Propósito

Justificar a razão para construir o produto e as vantagens (de negócios) que se obterão.

2.1.2. Clientes

Descrever as pessoas (e/ou entidades) interessadas no produto;.

2.1.3. Usuários

Descrever os usuários finais e como eles afetarão a usabilidade do produto.

2.1.4. Convenções de nomes e definições

Definir aqui (se necessário) o vocabulário (glossário) de termos necessário para entender o produto. Exemplo:

Termo	**Sigla**	**Definição**
Conta-corrente	CC	Conta bancária comum, que é possuída por um cliente do banco.

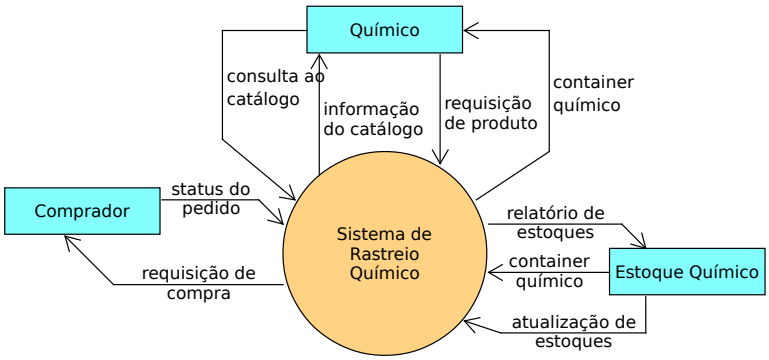
2.2. Restrições do produto

Descrever requisitos globais que se aplicam ao produto como um todo, definido antes dos requisitos funcionais e não funcionais. Eles acabam por “guiar” a definição dos outros requisitos.

2.3. Escopo do projeto

Explicar o que o produto a ser desenvolvido contemplará, de uma forma resumida, de modo que se tenha uma ideia clara de suas capacidades.

Sugere-se a utilização de um diagrama de contexto – ferramenta que explica as entradas e saídas dos sistemas e seus interagentes externos. Depois, em texto, explicar seus componentes. Exemplo:



Alternativamente pode ser utilizado um diagrama de casos de uso, resumindo cada caso (sem entrar em detalhes).

2.4. Requisitos funcionais

Descrever por meio de tabelas os requisitos funcionais do sistema. Por exemplo:

ID do Requisito	**Descrição**	**Caso de Uso Relacionado** (preencher apenas quando se for modelar os casos de uso!)
R001	O sistema de caixa-eletrônico deverá verificar a validade do cartão inserido.	UC0001
R002	O sistema de caixa-eletrônico deverá verificar se o código do usuário corresponde aquele cadastrado	UC0003

2.5. Requisitos não funcionais

Descrever os requisitos não funcionais a seguir apenas para aqueles que são relevantes ao seu projeto. Aqueles que não se aplicam, não é necessário incluir. A seguir, tem-se uma classificação comum para os requisitos não funcionais (**NOTA**: apague os itens 2.5.1 a 2.5.8 na sua documentação – eles são apenas um guia para pensar em requisitos não funcionais – basta

preencher corretamente a tabela de requisitos não funcionais).

2.5.1. Requisitos de aparência

A aparência do produto – discutir requisitos de usabilidade.

2.5.2. Requisitos de usabilidade

São aqueles requisitos que dependem do usuário, isto é, como o sistema será utilizado; facilidade de uso.

2.5.3. Requisitos de eficiência

Velocidade, tempo de resposta, acuidade, segurança, confiabilidade etc.

2.5.4. Requisitos operacionais

O ambiente operacional do produto. Onde o produto vai executar (exemplo: sistema operacional, sistema embarcado) e como isso é importante para o sucesso do produto.

2.5.5. Requisitos de manutenibilidade e portabilidade

Como o produto poderá ser modificado e ampliado.

2.5.6. Requisitos de segurança

Tópicos de segurança, confidencialidade e integridade do produto.

2.5.7. Requisitos culturais e políticos

Se necessário, descrever possíveis fatores culturais e políticos que devem ser levado em consideração.

2.5.8. Requisitos legais

Se necessário, explicar a conformidade às leis que são aplicáveis ao produto.

Organizar os requisitos não funcionais também em uma tabela. Exemplo:

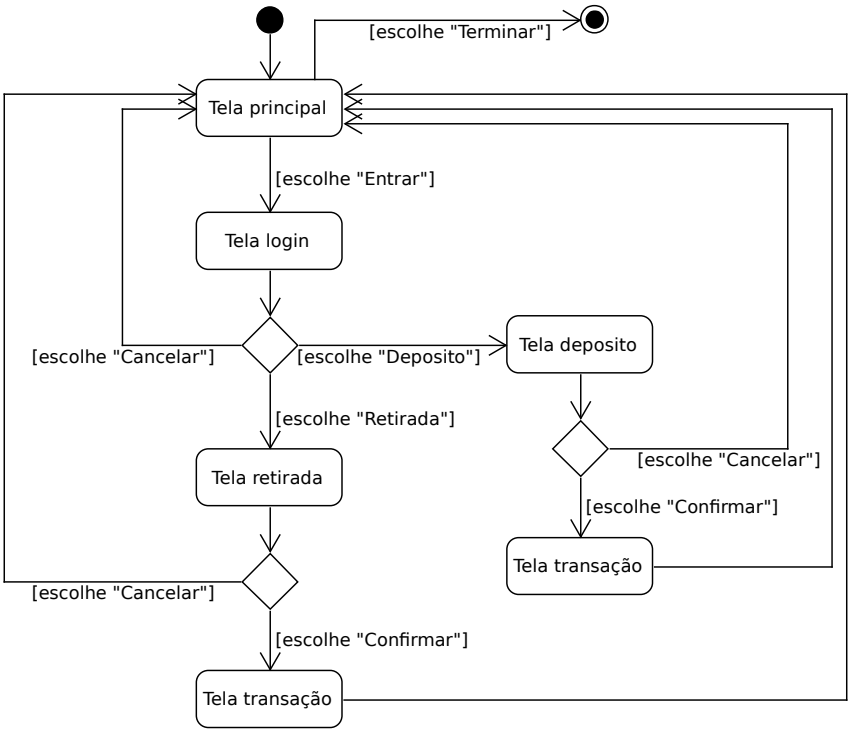
ID do Requisito	**Descrição**	**Caso de Uso Relacionado** (preencher apenas quando se for modelar os casos de uso!)
RN01	O sistema deverá completar qualquer transação até 30 segundo	UC0007
RN02	A interface de operação será baseada em ícones apresentados em uma tela “touch screen”	UC0010, UC0011

2.6. Protótipos do produto

Para entender melhor o que se deseja construir, deve ser elaborado um conjunto de telas e o modelo de navegação do software de acordo com seus requisitos funcionais.

2.6.1. Modelo de navegação

Explica como o protótipo funcionará sob o ponto de vista de comandos, eventos, links clicados etc. Pode ser elaborado com um **diagrama UML de Atividade**. Exemplo:



2.6.2. Protótipos de tela

Desenhar e explicar cada tela planejada pelo sistema. **Não** é **necessário implementar** nenhuma operação – apenas **apresentar figuras representativas** das **interfaces** de usuário (**com** um **título cada uma**) seguida de uma **explicação** do que cada **elemento** da **interface faz**. **Relacionar** as **telas** desta **seção** com os **nomes** do **diagrama** de **navegação** do **item anterior**.
A **prototipação** pode se **executada** com:

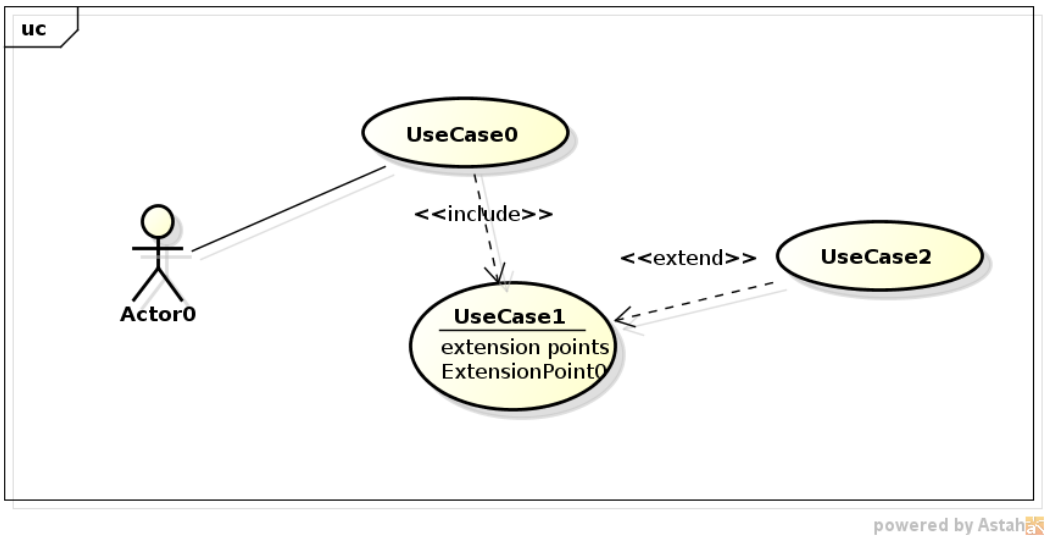
- **Programação** e algum ***framework*** de **HCI** (interface homem-máquina). Por exemplo, Java + Swing;
- **Programas** de ***mock-up***: sem programação, utiliza elementos de desenho para a criação das interfaces. Exemplos de programas que podem ser utilizados:
- **Visio** (pago);
- **Pencil** - http://pencil.evolus.vn/Downloads.html (gratuito).

Na Internet, procure por “mock-up free” e então escolha um programa para criar seu protótipo.

3. Modelo de Casos de Uso do Sistema

3.1. Diagrama de Casos de Uso do sistema

Inserir aqui o diagrama de casos de uso do sistema. Ele deve ser o mais completo possível, contemplando os relacionamentos de inclusão, extensão e generalização. Se o diagrama ficar muito grande, ele pode ser quebrado em mais de um diagrama. Exemplo:



Depois vem a parte da documentação, que deve seguir algum tipo de padrão, como indicado a seguir.

3.2. Documentação dos Atores

Modelo de documentação para cada ator do sistema.

3.2.1. Ator *<nome do ator>*

Ator: *<nome do ator>*
ID: *<número do ator>*
Resumo: *<Resumir o papel representado pelo ator>*
Responsabilidades: 1. *<Responsabilidade 1>* 2. *<Responsabilidade 2>* 3. ...
Ambiente Físico *<Ambiente onde o ator está localizado>*
Número e Tipo *<Descrição da quantidade e tipo de atores necessários>*
Frequência com que usa o sistema *<Descrição do grau de utilização do ator no sistema>*

3.3. Documentação dos Casos de Uso

Modelo de documentação para cada caso de uso do sistema (**NOTA**: se no fluxo de eventos houver referência a um ponto de extensão ou inclusão, isto deve ser feito no texto e depois detalhado em casos de uso separados).

3.3.1. Caso de uso *<nome do caso de uso>*

Caso de Uso: *<nome do caso de uso>*
ID: *<número do caso de uso>*
Resumo: *<resumir aqui o que o caso de uso representa>*
Atores primários: *<nomes dos atores que iniciam o caso>*
Atores secundários: *<nomes dos atores que interagem com o caso de uso após ele ter sido iniciado>*
Pré-condições: 1. *<pré-condição 1>* 2. *<pré-condição 2>* 3. ...
Fluxo de eventos principal: 1. *<evento 1>* 2. *<evento 2>* 3. ...
Pós-condições: 1. *<pós-condição 1>* 2. *<pós-condição 2>* 3. ...
Fluxo de eventos alternativo: *<nome do fluxo alternativo 1>* *<nome do fluxo alternativo 2>* ...

IMPORTANTE: durante a confecção dos casos de uso, preencher a matriz de requisitos, indicando qual(ais) caso(s) de uso contemplam os requisitos em questão.

Caso de Uso: <*nome do caso de uso*>
ID: <*número do caso de uso*>
Resumo: <*resumir aqui o que o caso de uso representa*>
Atores primários: <*nomes dos atores que iniciam o caso*>
Atores secundários: <*nomes dos atores que interagem com o caso de uso após ele ter sido iniciado*>
Pré-condições: 1. <*pré-condição 1*> 2. <*pré-condição 2*> 3. ...
Fluxo de eventos principal: 1. <*evento 1*> 2. <*evento 2*> 3. ...
Pós-condições: 1. <*pós-condição 1*> 2. <*pós-condição 2*> 3. ...
Fluxo de eventos alternativo: <*nome do fluxo alternativo 1*> <*nome do fluxo alternativo 2*> ...

4. Modelos de Análise do Sistema

Nesta seção, aplica-se a análise orientada a objetos, que aqui consistirá em:

(a) **Diagramas de pacotes e de classe**: explica o sistema do ponto de vista estrutural (estático), com suas classes e relacionamentos entre as classes. Nesta etapa, procura-se descobrir principalmente as classes relacionadas ao negócio (classes do domínio da aplicação). Para se chegar às classes, basta realizar sessões de CRC com o grupo. **As fichas CRC não precisarão constar da documentação entregue – use-a apenas como técnica de apoio (se precisar) para descobrir classes do sistema**. Especificar as classes como no modelo exemplificado. Nas operações utilizar ou pseudocódigo ou diagrama de atividade para explicar seu funcionamento.

(b) **Realizações de casos de uso**: utilizar diagramas de sequência para representar os cenários de utilização mais importantes do sistema.

4.1. Especificação das classes de análise do sistema

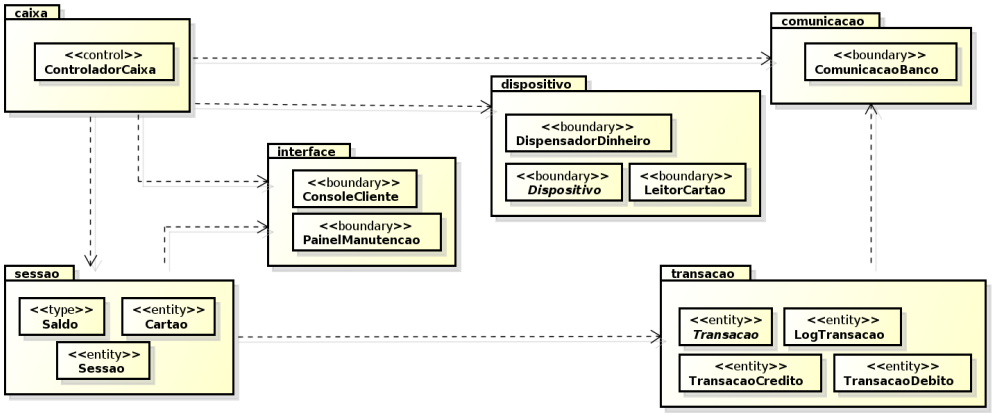
Inserir aqui os diagramas de classes do sistema organizadas, de preferência, em pacotes lógicos, como estudado no curso. Sugestão de organização:

- Apresentar e explicar os pacotes do sistema;
- Para cada pacote, especificar cada classe.

Seguem exemplos.

4.1.1. Visão geral dos pacotes de análise

O diagrama de pacotes a seguir exibe as classes do sistema XXX:



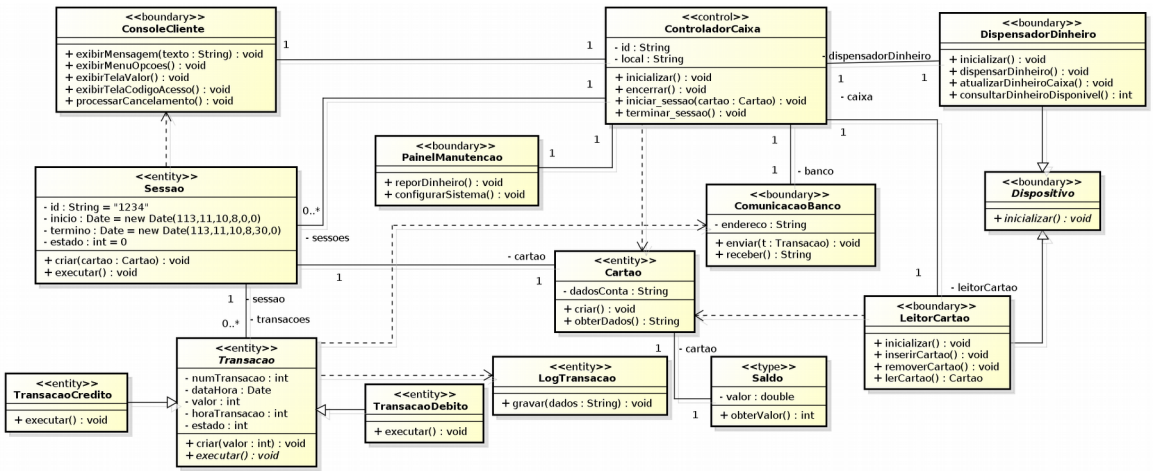
Descrição dos pacotes:

4.1.1.1. Pacote caixa

O pacote caixa serve para …

4.1.2. Diagramas e especificação das classes

Nesta seção, apresentar o(s) diagramas(s) de classes e especificar cada classe utilizando o padrão a seguir (exemplo).



4.1.2.1. Classe Conta

Descrição
A classe conta representa uma conta do banco ...

Superclasse(s)
Nenhuma

Subclasse(s)
ContaPoupanca, ContaInvestimento

Atributos

Nome:	**numero**
Descrição:	Representa o número único da conta
Tipo:	string, limitado a 5 dígitos
Domínio:	De “00000” a “99999”

Nome:	**saldo**
Descrição:	Representa o saldo da conta
Tipo:	double
Domínio:	Qualquer valor no intervalo numérico de double

Operações

Nome:	Transferir	
Descrição:	Debita o valor de uma conta (origem) e credita em outra (destino)	
Parâmetros		
Nome	**Tipo**	**Propósito**
origem	Conta	Representa a conta origem a ser debitada
destino	Conta	Representa a conta destino a ser creditada
valor	Double	Representa o valor a ser transferido entre contas
Pré-condições:		
1. A conta origem deve ter saldo suficiente para ser debitada.		
Especificação (diagrama de atividade ou pseudocódigo)		
1. Atualizar o valor do saldo da conta **origem** com o valor atual menos o **valor** a ser transferido; 2. Atualizar o valor do saldo da conta **destino** com o valor atual mais o **valor** a ser transferido;		
Pós-condições:		
1. A conta origem é debitada de um montante representado pelo parâmetro valor; 2. A conta destino é creditada de um montante representado pelo parâmetro valor;		

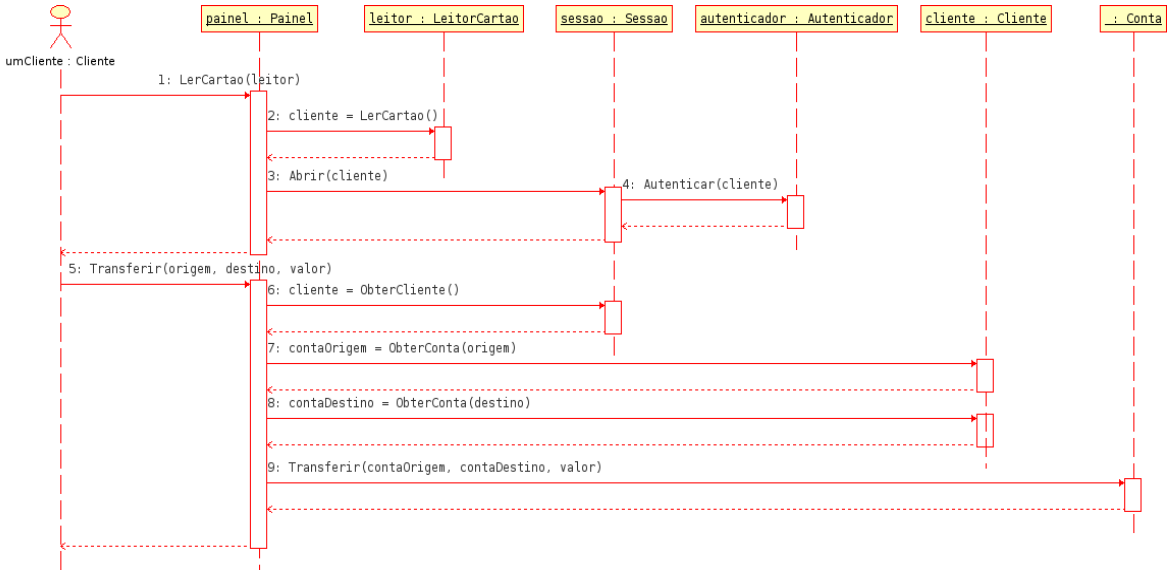
4.2. Realizações de Casos de Uso

O objetivo aqui é demonstrar com mais detalhes como os casos de uso podem ser simulados pelas

classes descobertas na Análise. Aqui, cada caso de uso é realizado com objetos das classes de análise determinadas anteriormente. Utilizar diagramas de sequência para representar os cenários de utilização mais importantes do sistema.

4.2.1. Cenário: Transferir Valor

Por exemplo, imagine que no banco Transferir Valor seja um dos casos de uso. Pode-se comprovar as classes de análise desenvolvidas até então com o seguinte diagrama de sequência:



Pode-se anotar textos explicativos logo após o diagrama, se necessário.

4.2.2. Cenário: Outra realização …

…

etc.

5. Modelos de Projeto do Sistema

Enquanto que na Análise Orientada a Objetos o foco está na criação de um modelo lógico do sistema de modo a representar toda a funcionalidade que o sistema deve prover para satisfazer os requisitos do usuário, o objetivo do Projeto (“design”) Orientado a Objetos do sistema está na especificação completa de como esta funcionalidade será implementada.

5.1. Classes de projeto

São elaboradas a partir das classes de Análise – **domínio do problema** – e de classes existentes em um **domínio de soluçã**o – classes, bibliotecas, componentes reutilizáveis que já estão prontas e que resolvem algum problema específico (por exemplo, classes Date, Time, persistência de objetos

com “frameworks” tais como Hibernate, classes de interface homem-máquina como Swing e assim por diante). Utilizar as observações a seguir como um “checklist”:

- Deve-se criar aqui diagramas UML de classes de projeto – baseadas no que foi dito no parágrafo anterior. Assim, não haverá em geral uma tradução 1:1 das classes de análise para as classes de projeto pois mais elementos poderão ser agregados.
- Explicar as classes do domínio de solução que foram utilizadas, indicando o endereço na WEB onde podem ser obtidas e a versão. Justificar seu uso no projeto, comparando com outras possíveis soluções que não foram adotadas.
- As classes de projeto deverão possuir as seguintes qualidades (quanto mais é melhor):
    - Serem completas e suficientes: ser completa significa possuir todas as funcionalidades esperadas pelos cientes da classe; ser suficiente significa que a classe possui apenas as operações que realizam o objetivo da classe. Essas duas condições significam: a classe deve oferecer apenas o que os clientes da classe esperam e nada mais;
    - As operações devem ser primitivas: evitar operações desnecessariamente complexas – a menos que hajam uma boa justificativa, elas deve focar apenas na solução de problemas básicos;
    - Alta coesão: as classes devem representar uma única abstração, utilizando um conjunto mínimo de características e evitar classes que resolvam mais de um problema;
    - Baixo acoplamento: se possível, a classe deve depender de um número mínimo de outras classes de modo a atender suas responsabilidades.
- Utilizar herança de modo efetivo:
    - Herança é o tipo mais forte de acoplamento que existe entre as classes, então seu uso deve ser revisto para evitar projetos com pouca flexibilidade. Por exemplo, a classe Empregado poderia ser superclasse das classes Programador e Gerente. Mas, se no projeto em questão Programador e Gerente forem cargos, então deve-se remover a relação de herança e substituí-la por uma associação ou agregação: Empregado possui Cargo e Cargo torna-se a superclasse de Programador e Gerente;
    - Deve-se utilizar corretamente herança e implementação de interfaces: na herança, uma classe-filha herda a interface (operações públicas) e a implementação (atributos, relacionamentos, operações protegidas e privadas) da sua classe-mãe, enquanto que na implementação de interface, a classe-filha obtém um conjunto de operações públicas, atributos e relacionamentos que não possuem implementação.

Herança deve ser utilizada quando se deseja herdar detalhes de implementação e implementação de interfaces deve ser utilizada quando se deseja definir apenas um "contrato", sem a necessidade de detalhes de implementação;

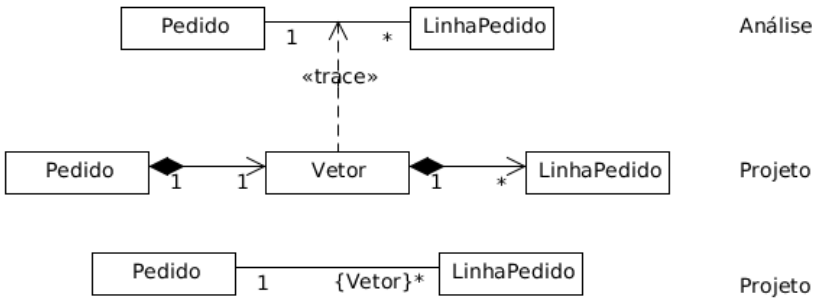
- Classes parametrizadas (ou "templates"): em linguagens tais como Java, C++ e C#, pode-se fazer uso de classes parametrizadas para se resolver um problema particular (exemplo: uma lista em Java implementada com auxílio de um ArrayList). Assim, deve-se indicar em UML qual foi a classe parametrizada empregada e que classes foram instanciadas a partir dela.

- Classes aninhadas: se houver algum caso de uma classe que foi definida dentro do escopo de outra (isto é possível em Java), deve-se representar em UML pela relação de "conter" (cuja simbologia é a mesma daquela utilizada entre pacotes).

5.2. Refinamento dos relacionamentos

Consiste em refinar os relacionamentos do modelo de análise de modo que sejam posteriormente implementados na linguagem de programação escolhida.

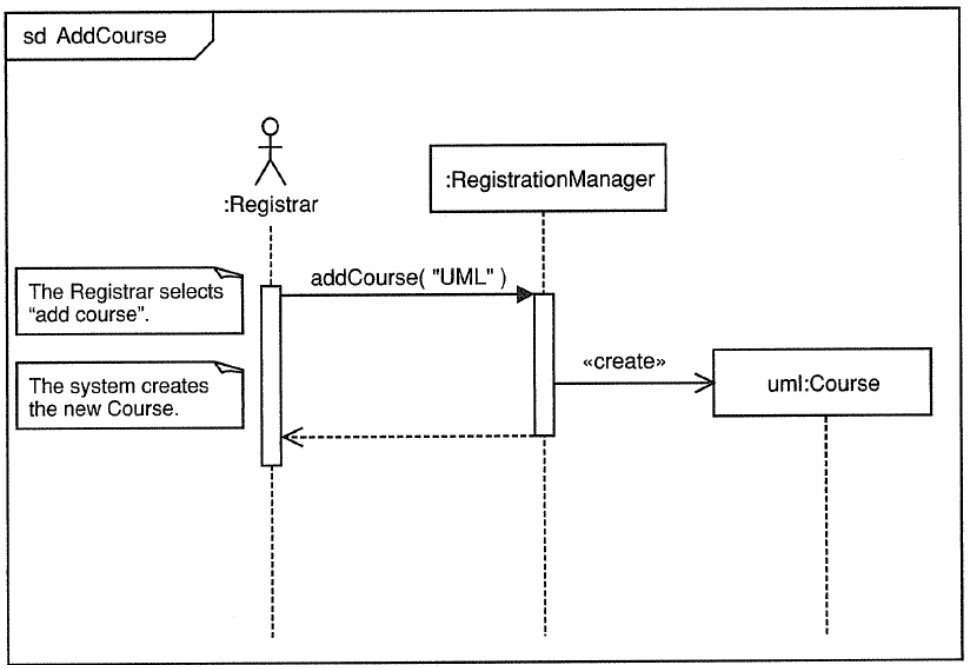
- Agregação e composição: alguns relacionamentos de associação poderão ser convertidos em relacionamentos de agregação (todo e parte fracamente relacionados) ou de composição (todo e parte fortemente relacionados). Assim que decididos, verificar:
  - Adicionar multiplicidades e nomes de papéis às associações que ainda não as possuem;
  - Decidir que lado faz o papel de "todo" e que lado faz o papel de "parte";
  - Quando se trata de uma composição, o lado "todo" possui multiplicidade 0..1 ou exatamente 1, senão, trata-se de uma agregação;
  - Adicionar navegabilidade (→) do "todo" para a "parte";
- Uma associação "um-para-um" pode ser representada como uma composição;
- Uma associação "vários-para-um" pode ser representada como uma agregação;
- Em associações "um-para-vários" existe uma coleção de objetos do lado das "partes". Na implementação deste tipo de relacionamento, deve ser utilizada algum suporte nativo para coleções (por exemplo Vector ou List em Java). Exemplo:



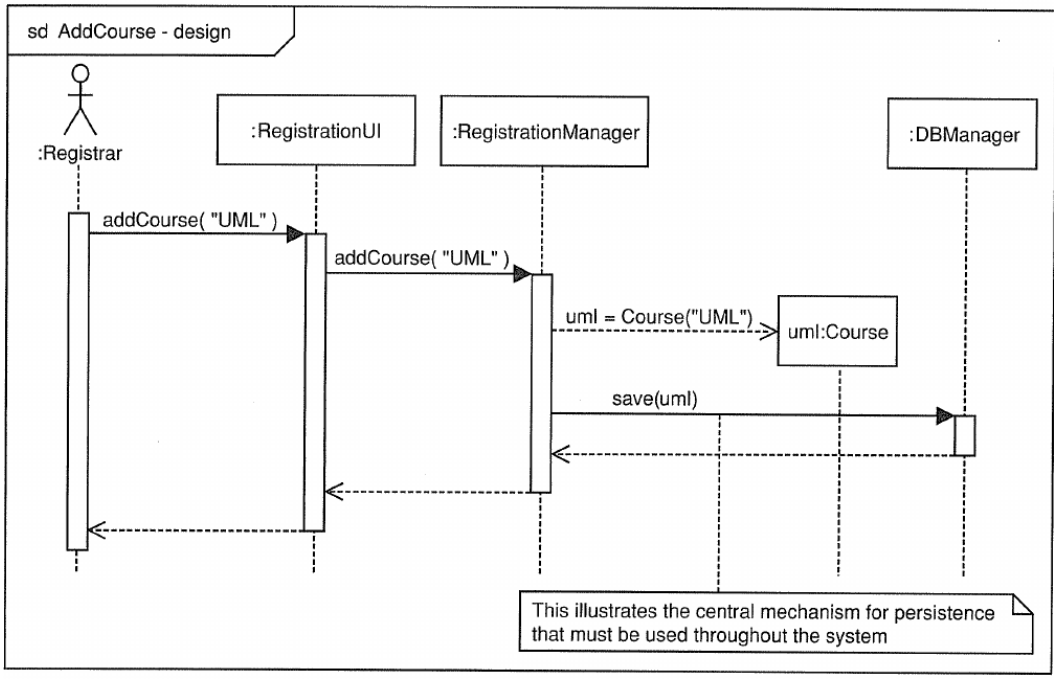
5.3. Realizações de casos de uso – Projeto

Aqui refinam-se os diagramas de interação da análise ou ainda acrescentam-se outros, para suportar os novos elementos adicionados em projeto, tais como persistência de objetos.

Por exemplo, um cenário de Análise como o apresentado a seguir:



Poderia ser assim detalhado em Projeto, adicionado detalhes de como um objeto será salvo em um banco de dados:


sd AddCourse - design
:Registrar
:RegistrationUI
:RegistrationManager
:DBManager
addCourse("UML")
addCourse("UML")
uml = Course("UML")
uml:Course
save(uml)
This illustrates the central mechanism for persistence
that must be used throughout the system