

# Módulo 4

Engenharia de Software



## Lição 3

Engenharia de Requisitos

*Versão 1.0 - Jul/2007*

**Autor**

Ma. Rowena C. Solamo

**Equipe**

Jaqueline Antonio  
 Naveen Asrani  
 Doris Chen  
 Oliver de Guzman  
 Rommel Feria  
 John Paul Petines  
 Sang Shin  
 Raghavan Srinivas  
 Matthew Thompson  
 Daniel Villafuerte

**Necessidades para os Exercícios****Sistemas Operacionais Suportados****NetBeans IDE 5.5** para os seguintes sistemas operacionais:

- Microsoft Windows XP Profissional SP2 ou superior
- Mac OS X 10.4.5 ou superior
- Red Hat Fedora Core 3
- Solaris™ 10 Operating System (SPARC® e x86/x64 Platform Edition)

**NetBeans Enterprise Pack**, poderá ser executado nas seguintes plataformas:

- Microsoft Windows 2000 Profissional SP4
- Solaris™ 8 OS (SPARC e x86/x64 Platform Edition) e Solaris 9 OS (SPARC e x86/x64 Platform Edition)
- Várias outras distribuições Linux

**Configuração Mínima de Hardware****Nota:** IDE NetBeans com resolução de tela em 1024x768 pixel

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	500 MHz Intel Pentium III workstation ou equivalente	512 MB	850 MB
Linux	500 MHz Intel Pentium III workstation ou equivalente	512 MB	450 MB
Solaris OS (SPARC)	UltraSPARC II 450 MHz	512 MB	450 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Série 1.8 GHz	512 MB	450 MB
Mac OS X	PowerPC G4	512 MB	450 MB

**Configuração Recomendada de Hardware**

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	1 GB
Linux	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	850 MB
Solaris OS (SPARC)	UltraSPARC IIIi 1 GHz	1 GB	850 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Series 1.8 GHz	1 GB	850 MB
Mac OS X	PowerPC G5	1 GB	850 MB

**Requerimentos de Software**

NetBeans Enterprise Pack 5.5 executando sobre Java 2 Platform Standard Edition Development Kit 5.0 ou superior (JDK 5.0, versão 1.5.0\_01 ou superior), contemplando a Java Runtime Environment, ferramentas de desenvolvimento para compilar, depurar, e executar aplicações escritas em linguagem Java. Sun Java System Application Server Platform Edition 9.

- Para **Solaris, Windows, e Linux**, os arquivos da JDK podem ser obtidos para sua plataforma em <http://java.sun.com/j2se/1.5.0/download.html>
- Para **Mac OS X**, Java 2 Platform Standard Edition (J2SE) 5.0 Release 4, pode ser obtida diretamente da Apple's Developer Connection, no endereço: <http://developer.apple.com/java> (é necessário registrar o download da JDK).

Para mais informações: <http://www.netbeans.org/community/releases/55/relnotes.html>

### ***Colaboradores que auxiliaram no processo de tradução e revisão***

Aécio Júnior  
Alexandre Mori  
Alexis da Rocha Silva  
Allan Souza Nunes  
Allan Wojcik da Silva  
Anderson Moreira Paiva  
Anna Carolina Ferreira da Rocha  
Antonio Jose R. Alves Ramos  
Aurélio Soares Neto  
Bruno da Silva Bonfim  
Carlos Fernando Gonçalves  
Daniel Noto Paiva  
Denis Mitsuo Nakasaki

Fábio Bombonato  
Fabrício Ribeiro Brigagão  
Francisco das Chagas  
Frederico Dubiel  
Jacqueline Susann Barbosa  
João Vianney Barrozo Costa  
Kleberth Bezerra G. dos Santos  
Kefreen Ryenz Batista Lacerda  
Leonardo Ribas Segala  
Lucas Vinícius Bibiano Thomé  
Luciana Rocha de Oliveira  
Luiz Fernandes de Oliveira Junior  
Marco Aurélio Martins Bessa

Maria Carolina Ferreira da Silva  
Massimiliano Girolodi  
Mauro Cardoso Mortoni  
Mauro Regis de Sousa Lima  
Paulo Afonso Corrêa  
Paulo Oliveira Sampaio Reis  
Ronie Dotzlaw  
Seire Pareja  
Sergio Terzella  
Thiago Magela Rodrigues Dias  
Vanessa dos Santos Almeida  
Wagner Eliezer Rancoletta

### ***Auxiliadores especiais***

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach
- **Guiné Bissau** – Alfredo Cá, Bunene Sisse e Buon Olossato Quebi – ONG Asas de Socorro

### ***Coordenação do DFJUG***

- **Daniel deOliveira** – JUGLeader responsável pelos acordos de parcerias
- **Luci Campos** - Idealizadora do DFJUG responsável pelo apoio social
- **Fernando Anselmo** - Coordenador responsável pelo processo de tradução e revisão, disponibilização dos materiais e inserção de novos módulos
- **Rodrigo Nunes** - Coordenador responsável pela parte multimídia
- **Sérgio Gomes Veloso** - Coordenador responsável pelo ambiente JEDI™ (Moodle)

### ***Agradecimento Especial***

**John Paul Petines** – Criador da Iniciativa JEDI™  
**Rommel Faria** – Criador da Iniciativa JEDI™

# 1. Objetivos

Projetar e construir um sistema de computador é desafiante, criativo e simplesmente divertido e prazeroso. Entretanto, desenvolver um bom programa que resolva o problema errado, não serve a ninguém. É importante compreender as necessidades e requisitos do usuário, de modo que resolvamos o problema correto e construamos o sistema correto.

Neste capítulo, discutiremos os conceitos e as dinâmicas da engenharia de requisitos. É uma fase de desenvolvimento de programa que consiste em sete tarefas ou atividades distintas que tem a meta de compreender e documentar os requisitos da parte interessada. Dois modelos importantes serão construídos: **modelo de requisitos**, que é o modelo de domínio de sistema ou problema e o **modelo de análise**, que serve como o modelo de base do programa (modelo de solução). A Matriz de Requisitos Rastreáveis (RTM, sigla em inglês para *Requirement Traceability Matrix*) será introduzida para ajudar os engenheiros de programa a administrar os requisitos e, também, as métricas de requisitos e seus significados serão discutidos.

Ao final desta lição, o estudante será capaz de:

- Aprender os conceitos e dinâmicas da engenharia de requisitos
- Aprender como construir o modelo de requisitos
- Aprender como construir o modelo de análise
- Aprender como manter a trilha dos requisitos usando a Matriz de Rastreabilidade de Requisitos (MRR)
- Aprender as métricas de requisitos e seus significados

## 2. Conceitos de Engenharia de Requisitos

**Engenharia de Requisitos** permite que os desenvolvedores de programas compreendam o problema que estão resolvendo. Abrange um conjunto de tarefas que conduzem a uma compreensão do impacto do negócio que o programa deverá ter, o que o cliente deseja e como o usuário final interagirá com o programa. Provê um mecanismo apropriado para as necessidades de compreensão da parte interessada, analisando requisitos, determinando viabilidade, negociando uma solução razoável, especificando as soluções claramente, validando especificação e administrando requisitos como eles são transformados em um sistema.

### **Significado para o Cliente, Usuários finais, Time de Desenvolvimento de Programa e às outras Partes Interessadas**

Engenharia de Requisitos fornece o contrato básico entre usuários finais e desenvolvedores, sobre o que o programa deve fazer. É uma instrução clara quanto ao escopo e aos limites do sistema a ser analisado e estudado. Fornece às partes interessadas uma oportunidade de definir seus requisitos compreensíveis à equipe de desenvolvimento. Isto pode ser obtido por diferentes documentos, artefatos ou produtos de trabalho, tais como: modelos de caso de uso, modelos de análise, características e lista de funções, cenários de usuário etc.

Projetando e construindo um programa de computador elegante que resolve o problema errado é um desperdício. Esta é a razão porque é importante compreender o que o cliente quer, antes de começar a projetar e construir um sistema baseado em computador. A engenharia de requisitos constrói uma ponte entre o projeto e a construção. Permite ao time de desenvolvimento de programa examinar:

- o contexto de trabalho do programa a ser executado
- as necessidades específicas que o projeto e a construção devem suprir
- as prioridades que guiam a ordem na qual o trabalho será completado
- os dados, funções e comportamentos que terão um impacto profundo no projeto resultante

Engenharia de requisitos é uma outra parte da engenharia de software, deve ser adaptada de acordo com as necessidades do processo, projeto, produto e pessoas que estão trabalhando. Esta é uma atividade que é iniciada a partir da concepção do modelo de software e que pode ser utilizada na fase de desenho e construção.

## 3. Tarefa de Engenharia de Requisitos

São sete tarefas distintas para a Engenharia de Requisitos a saber: **concepção<sup>1</sup>, elucidação<sup>2</sup>, elaboração, negociação, especificação, validação e negociação**. É importante manter em mente que estas tarefas podem ocorrer em paralelo e todas são adaptadas às necessidades do projeto. Todas objetivam definir o que o cliente deseja e servem para estabelecer uma fundamentação sólida para o design e a construção das necessidades do cliente.

### 3.1. Concepção

Em geral, a maioria dos projetos de software iniciam quando há um problema a ser resolvido ou uma oportunidade identificada. Como exemplo, considere que um negócio descubra uma necessidade ou um novo mercado ou um potencial serviço. Na concepção, o escopo do problema e sua natureza são definidos. **Engenharia de Software** faz um conjunto de perguntas livres de contexto com a intenção de estabelecer uma compreensão básica do problema, o que as pessoas querem de solução, a natureza da solução e a eficiência da compreensão preliminar e a colaboração entre o usuário final e o desenvolvedor.

#### Iniciando a Engenharia de Requisitos

Desde a investigação preliminar do problema, uma aproximação ou entrevista com P&R (perguntas e respostas) é uma técnica apropriada para entender o problema e sua natureza. Enumeramos a abaixo alguns passos recomendados para iniciar a fase de requisitos.

##### Passo 1: Identificar as partes interessadas

Uma parte interessada é qualquer um que se beneficie direta ou indiretamente do sistema que está sendo desenvolvido. O gerente de operações de negócio, o gerente de produtos, as pessoas de marketing, clientes internos e externos, usuários finais e outras pessoas comuns para entrevistar. É importante, nesta etapa, criar uma lista de pessoas que contribuem com a entrada de requisitos. A lista dos usuários crescerá com mais ou menos pessoas que começam a se envolver na elucidação.

##### Passo 2: Reconhecimento de múltiplos pontos de vista

É importante recordar que diferentes partes interessadas tem uma visão diferente do sistema. Cada um obtém diferentes benefícios quando o sistema é um sucesso; cada um corre diferentes riscos se o sistema falhar. Nesta etapa, deve-se categorizar todas as informações e requisitos da partes interessadas. Deve-se, também, identificar os requisitos que são inconsistentes e conflitantes com outros. Deve ser organizado de tal maneira que as partes interessadas possam decidir por um conjunto de requisitos exigidos para o sistema.

##### Passo 3: Trabalho para a colaboração

O sucesso da maioria dos projetos depende da colaboração. Para conseguir isto, é necessário procurar áreas dentro das exigências que são comuns às partes interessadas. Entretanto, o desafio está na resolução de inconsistências e conflitos. A colaboração não significa que um comitê decida as exigências do sistema. Em muitos casos, para resolver conflitos um líder de equipe, normalmente um gerente de negócio ou um desenvolvedor sênior, decide que exigências são incluídas quando o software é desenvolvido.

##### Passo 4: Fazendo as primeiras perguntas

Para definir o espaço e a natureza do problema, as perguntas são feitas aos clientes e às partes interessadas. Essas perguntas podem ser categorizadas. Como por exemplo:

Motivação da parte interessada ou do cliente:

1. Quem está solicitando este trabalho?
2. Por que estão solicitando tal trabalho?

---

<sup>1</sup> Ato ou efeito de conceber

<sup>2</sup> Ato ou efeito de esclarecer ou explicar

3. Quem são os usuários finais do sistema?
4. Quais são os benefícios quando o sistema for desenvolvido com sucesso?
5. Existe alguma outra maneira para fornecer a solução do problema? Quais são as alternativas?

Percepção do cliente e da parte interessada:

1. Como se pode caracterizar um bom produto de software?
2. Quais são os problemas que o software deverá solucionar?
3. Qual é o ambiente de negócio do cliente e/ou da parte interessada?
4. Há alguma restrição ou exigência que afete o modo que a solução é alcançada?

Eficácia da comunicação:

1. Estamos perguntando para as pessoas corretas as questões corretas?
2. As respostas estão sendo "oficialmente" formalizadas?
3. As perguntas são relevantes ao problema?
4. Sou eu quem faz perguntas demais?
5. Qualquer um pode fornecer informação adicional?
6. Há qualquer outra coisa que eu necessito saber?

### Inicializando o trabalho

O produto principal do trabalho é uma ou duas páginas do produto solicitado que é um sumário contendo a descrição do problema e a sua natureza.

## 3.2. Elucidação

Após a concepção, o próximo passo é a **elucidação**. Ajuda o cliente a definir o que foi requerido. Entretanto, esta não é uma tarefa fácil. Entre os problemas encontrados na elucidação são citados abaixo:

1. **Problemas de escopo.** É importante que os limites do sistema estejam definidos clara e corretamente. É importante evitar o uso demasiado de detalhes técnicos porque é possível confundir mais do que esclarecer os objetivos do sistema.
2. **Problemas de entendimento.** Em alguns momentos é difícil aos clientes e usuários possuir uma completa definição do que necessitam. Algumas vezes eles tem fraco entendimento da capacidade e limitações de seus ambientes computacionais, ou não possuem um entendimento completo do domínio do problema. Por vezes podem até omitir informações que acreditam ser óbvias.
3. **Problemas de volatilidade.** É inevitável que os requisitos mudem com o tempo.

Para ajudar a superar estes problemas, engenheiros de software devem abordar a pesquisa de requisitos de uma forma organizada e sistemática.

### Recolhendo requisitos colaborativos

Ao contrário da concepção onde abordamos o uso de perguntas e respostas, a elucidação combina os elementos de solução de problemas, elaboração, negociação e especificação dando um formato à elucidação de requisitos. Requer a cooperação do grupo de usuários-finais e desenvolvedores para esclarecer os requisitos. Trabalham juntos para:

- Identificar o problema
- Propor elementos de solução
- Negociar diferentes abordagens
- Especificar um conjunto de requisitos de soluções

### Desenvolvimento de Aplicações em Conjunto

*Joint Application Development* – JAD é uma técnica de recolhimento colaborativo popular para esclarecer os requisitos.

As tarefas envolvidas na elucidação podem ser categorizadas em três grupos, a saber, preparação

para sessão JAD, sessão JAD e pós-sessão JAD.

### **Preparação para sessão JAD**

1. Se não existe uma requisição, peça a um participante para escrever uma.
2. Marque o lugar, a data e hora da reunião.
3. Selecione um facilitador.
4. Convide aos participantes que serão parte das equipes de desenvolvimento, os clientes e outros participantes.
5. Distribua a requisição para todos os participantes antes da reunião.
6. A cada participante é pedido que faça o seguinte:
  - uma lista de objetos que são parte do ambiente que cerca o sistema;
  - uma lista de outros objetos que são produzidos pelo sistema;
  - uma lista dos objetos usados pelo sistema para realizar as suas funções;
  - uma lista de serviços (processos ou funções) que manipulam ou interagem com os objetos;
  - uma lista de restrições, tais como custo, tamanho e regras de negócio; e
  - uma lista de critérios de performance, tais como velocidade ou precisão.

Observe que não se espera que as listas sejam exaustivas mas espera-se que reflitam a percepção que a pessoa tem do sistema.

### **Tarefas na Reunião**

1. O primeiro tópico que precisa ser resolvido é a necessidade e a justificativa do novo produto. Todo mundo deve concordar que o produto é justificável.
2. Cada participante apresenta sua lista ao grupo.
3. Após a apresentação de todos os participantes, uma lista consolidada é criada. Ela elimina questões redundantes, adiciona novas idéias que porventura apareçam durante a discussão, mas não exclui nada.
4. A lista consensual em cada tópico (objetos, serviços, restrições e performance) é definida. A lista consolidada da tarefa anterior é diminuída, aumentada, ou reescrita para refletir o produto ou sistema a ser desenvolvido.
5. Um vez que a lista consensual está definida, a equipe é dividida em sub-equipes. Cada uma irá trabalhar para desenvolver as **mini-especificações** para uma ou mais entradas da lista consensual. A mini-especificação é simplesmente um detalhamento do item da lista utilizando palavras e frases.
6. Cada sub-equipe, então, apresenta a sua mini-especificação para todos os presentes. Inclusões, exclusões e maiores detalhamentos podem ser feitos. Em alguns casos, serão descobertos novos objetos, serviços, restrições ou requisitos de performance que serão adicionados às listas originais.
7. Em alguns casos, surgem questões que não poderão ser resolvidas durante a reunião. Uma lista de pendências é mantida e estas serão trabalhadas mais tarde.
8. Após se completar cada mini-especificação, cada presente faz uma lista de critérios de validação do produto ou sistema e apresenta a lista à equipe. Uma lista consensual de critérios de validação é criada.
9. A um ou mais participantes é associada a tarefa de escrever um esboço da Ata de Reunião com todas as questões levantadas na reunião.

### **Tarefas Pós-Reunião**

1. Compilar a Ata de Reunião com todos os itens discutidos na reunião.
2. Priorizar os requisitos. Pode-se usar a Técnica Qualidade da Função ou a Técnica de MoSCoW.



## Desenvolvimento com Qualidade da Função

É uma técnica que enfatiza o entendimento do que é importante para o usuário. Então, estes valores são desenvolvidos através do processo de Engenharia. Identifica três tipos de requisitos:

### Implantando as Funções de Qualidade

É uma técnica que enfatiza o entendimento do que é importante para o cliente. Então, utilizar estes valores por toda parte nos processos da engenharia de software. Para tal, são identificados três tipos de requisitos:

1. **Requisitos Normais** – Estes requisitos refletem diretamente nos objetivos expressados para um produto ou sistema durante as reuniões com os clientes. Isso significa que se os requisitos estão presentes, o cliente está satisfeito.
2. **Requisitos Esperados** – Estes requisitos são implícitos para o produto ou sistema e podem ser tão fundamentais que os clientes não os expressam explicitamente. A ausência destas necessidades podem causar uma significativa insatisfação. Exemplo dos requisitos esperados seriam as facilidades de interação entre homem e máquina, a maior parte das operações realizadas corretamente e confiáveis, além da facilidade de instalação das aplicações.
3. **Requisitos de Excitação** – Estes requisitos refletem as características que vão além das expectativas do cliente e provam ser muito satisfatórios quando presentes.

Com diversas reuniões com a equipe, **análises de valor** são conduzidas para determinar a prioridade relativa dos requisitos baseados em três implantações, definidas como: **implantação de funções, implantação de informação e implantação de tarefas**. Implantação de funções são usadas para determinar o valor de cada função que é necessária para o sistema. Implantação de informação identifica cada dado e evento que o sistema deve consumir e produzir. Implantação de tarefas examina o comportamento do produto ou sistema com o contexto do ambiente. Para cada análise de valor, cada requisito é categorizado em três tipos de requisitos.

### Técnica de MoSCoW

Cada requisito pode lidar classes de prioridades opostas, como definidas na Tabela 1. Durante o processo de engenharia de software, uma pequena reunião pode ser conduzida para rever as probabilidades e redefinir prioridades.

<b>Classificação</b>	<b>Significado</b>
Deve Ter	Este requisito será incluído no produto entregue.
Pode Ter	O plano do projeto atual indica que este requisito será incluído. Se as circunstâncias mudarem, este pode ser deixado de fora.
Talvez Tenha	O plano do projeto atual não indica que este requisito será incluído. Se as circunstâncias mudarem, este pode ser incluído.
Não Pode Ter	Este requisito não será incluído no produto entregue.

*Tabela 1: Classificação de Prioridades*

### Produto do trabalho de Esclarecimento

A saída da atividade de esclarecimento pode variar dependendo do tamanho do sistema ou do produto a ser construído. Para a maioria dos sistemas, os produtos da saída ou do trabalho incluem:

- ✓ Uma indicação da necessidade e da praticabilidade
- ✓ Uma indicação limitada do espaço para o sistema ou o produto
- ✓ Uma lista do cliente, dos usuários, e das outras partes interessadas que participaram do esclarecimento das necessidades.
- ✓ Uma descrição do ambiente técnico do sistema
- ✓ Uma lista de prioridade dos requisitos, preferencialmente, nos termos das funções, objetos e confinamentos do domínio que se aplicam a cada um.

### **3.3. Elaboração**

A informação obtida pela equipe durante o levantamento e esclarecimento é expandida e refinada durante a elaboração. Esta tarefa de exigência de engenharia tem o foco em definir, redefinir e refinar os modelos, a saber, os modelos de requisitos (domínio do sistema ou problema) e modelo de análise (domínio da solução). Tenta modelar "O QUE" ao invés de "COMO".

Modelo de Requisitos é criado usando metodologias que centralizam no usuário os cenários que definem a maneira como o sistema é usado. Descreve como os usuários finais e os atores interagem com o sistema.

Modelo da Análise é derivado do Modelo de Requisitos onde cada cenário é analisado para obter as classes de análise, isto é, o domínio das entidades de negócio que são visíveis ao usuário final. Os atributos de cada classe da análise são definidos e as responsabilidades que são exigidas por cada classe são identificadas. Os relacionamentos e a colaboração entre as classes são identificadas e uma variedade de diagramas suplementares de UML é produzida. O resultado final da elaboração é um modelo de análise que define o domínio informativo, funcional e o comportamento do problema. O desenvolvimento destes modelos será discutido no Modelo de Requisitos e Análise e na seção Especificação de Requisitos nesta lição.

### **Produto do trabalho de Elaboração**

O Modelo de Requisitos e o Modelo da Análise são os produtos principais de trabalho desta tarefa.

### **3.4. Negociação**

Na negociação, os clientes, as partes interessadas e a equipe do desenvolvimento do software negociam os conflitos. Os conflitos aumentam quando os clientes estão pedindo mais do que o desenvolvimento de software pode conseguir com os limitados recursos de sistema liberados. Para resolver estes conflitos, requisitos são classificados, os riscos associados com cada exigência são identificados e analisados, as estimativas de esforço do desenvolvimento e os custos são feitas, e prazo de entrega é definido.

O objetivo da negociação é desenvolver um plano de projeto que se adeque às exigências do usuário ao refletir acontecimentos do mundo real tais como o tempo, pessoas e o orçamento. Ele significa que o cliente obtém o sistema ou o produto que satisfaz à maioria das necessidades, e a equipe do software é capaz de trabalhar realisticamente para alcançar os prazos e orçamentos definidos nas reuniões.

### **A Arte de Negociação**

Negociação é um meio de estabelecer colaboração. Para o sucesso de um projeto de desenvolvimento de software, colaboração é a chave. Abaixo estão algumas diretrizes da negociação com as partes interessadas.

1. Lembre que a negociação não é nenhuma competição. Todo o mundo deve chegar a um acordo. Em algum nível, todos devem sentir que as preocupações foram enviadas, ou que alcançaram algo.
2. Tenha uma estratégia. Escute o que as partes querem alcançar. Decidam como juntos farão tudo acontecer.
3. Escute efetivamente. Mostre que está escutando e que está preocupado. Tente não formular a resposta ou uma reação enquanto o outro está falando. Pode-se adquirir algo que pode ajudar uma negociação futura.
4. Focalize no interesse da outra parte. Não leve posições duras para evitar conflito.
5. Não leve para o lado pessoal. Focalize no problema que precisa ser resolvido.
6. Seja criativo. Não tenha nenhum medo de "sair da caixa".
7. Esteja pronto para assinar. Uma vez que foi estabelecido um acordo, este deve ser assinado e passado para outros assuntos.

## Especificação

Uma especificação é o artefato final ou produto do trabalho produzido pelo engenheiro de software durante engenharia de requisitos. Serve como uma base para software subsequente da engenharia de atividades, particularmente, o desenho e a construção do software. Mostra os aspectos informais, funcionais e de comportamento do sistema. Pode ser escrito como um documento, um conjunto de modelos gráficos, um modelo matemático formal, um protótipo ou qualquer combinação destes. Os modelos servem como suas especificações.

## Validação

São avaliados os produtos de trabalho produzidos como consequência dos requisitos, acessados para a qualidade durante o processo de validação. Examina a especificação para assegurar que todas as exigências de software foram claramente declaradas e que foram descobertas inconsistências, omissões, e erros foram corrigidos. Confere a conformidade e trabalha os padrões de produtos estabelecidos no projeto de software.

A equipe de revisão que valida as exigências consiste em engenheiros de software, clientes, usuários, e outros participantes do projeto. Procuram por erros em conteúdo ou interpretação, áreas onde uma maior clareza é requerida, informação perdida, inconsistências, conflitos e exigências irreais.

## Lista de Verificação de Validação de Requisitos

A medida que os modelos são construídos, são examinados em termos de consistência, omissão, e ambigüidade. As exigências são priorizadas e se agrupam dentro de pacotes que serão implementados como incrementos de software e serão entregues ao cliente. Perguntas como as sugeridas por *Pressman* são listados abaixo para servir como uma diretriz para validar os produtos de trabalho da engenharia de requisitos.

1. Cada exigência é consistente com o objetivo geral para o sistema ou o produto?
2. Todas as exigências são especificadas no nível apropriado de abstração? Isto é, algumas exigências fornecem um nível de detalhe técnico que não é apropriado no estágio?
3. A exigência é realmente necessária ou representa uma característica que possa não ser essencial ao objetivo do sistema?
4. Cada exigência é limitada e clara?
5. Cada exigência tem a atribuição? Isto é, uma fonte (geralmente, um indivíduo específico) é anotada para cada exigência?
6. Algumas exigências entram em conflito com outras exigências?
7. Cada exigência é realizável no ambiente técnico que abrigará o sistema ou o produto?
8. Cada exigência é testável, uma vez implementada?
9. O modelo da exigência reflete corretamente a informação, a função e o comportamento do sistema a ser construído?
10. O modelo das exigências foi "dividido" de uma maneira que exponha uma informação progressivamente mais detalhada sobre o sistema?
11. O padrão das exigências é utilizado para simplificar o modelo das exigências? Todos os padrões foram validados corretamente? Todos os padrões são consistentes com as exigências do cliente?

Estas e outras perguntas devem ser feitas e respondidas para assegurar-se de que todos os produtos do trabalho reflitam as necessidades do cliente, de modo que forneça uma fundação sólida para o projeto e a construção.

## 3.5. Gerência

É o conjunto das atividades que ajudam a equipe de projeto a identificar, controlar e traçar as

exigências e suas mudanças em qualquer fase, enquanto o projeto progride. O gerenciamento das exigências começa uma vez que estas são identificadas. A cada exigência é atribuído um identificador único.

Uma vez que os requisitos tenham sido identificados, tabelas da **Matriz de Rastreabilidade** são desenvolvidas.

### **Matriz de Rastreabilidade de Requisitos**

*Requirements Traceability Matrix – RTM* será discutida na seção *Matriz de Rastreabilidade de Requisitos* nesta lição e ajudará aos engenheiros de software a gerenciar os requisitos durante o progresso do processo de desenvolvimento.

## 4. Modelo e Análise de Requisitos

Durante a elaboração, a informação obtida durante a concepção e a elucidação é ampliada e refinada para produzir dois importantes modelos: de requisitos e de análise. O modelo de requisitos determina o modelo de sistema ou o domínio do problema. Nesta seção, iremos discutir sobre o modelo de requisitos e como este pode ser construído.

### 4.1. O Modelo de Requisitos

O *Rational Rose* define o Modelo de Requisitos como ilustrado na Figura 1<sup>3</sup>. Este modelo consiste em três elementos, especificamente, **Modelo de Caso de Uso**, **Especificações Suplementares**, e **Glossário**.

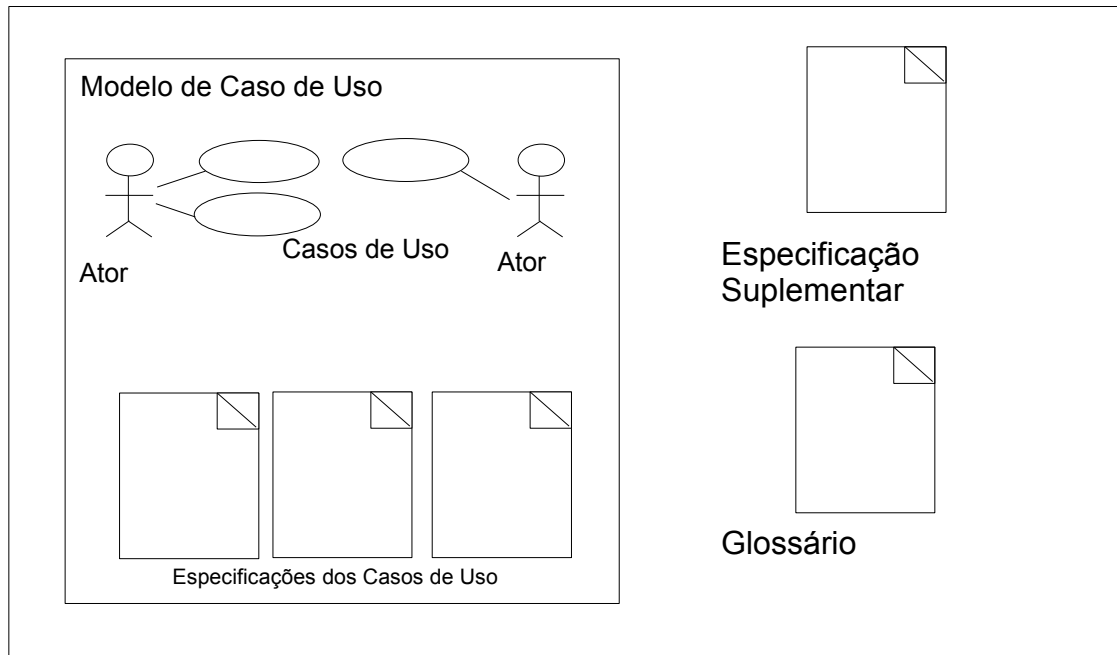


Figura 1: Modelo de Requisitos

#### Modelo de Caso de Uso

Este modelo é usado para descrever o que o sistema deverá fazer. Ele serve como um contrato entre clientes, usuários finais e desenvolvedores de sistema. Para clientes e usuários finais, este modelo é usado para validar o que o sistema deverá ter para atender às suas expectativas. Para os desenvolvedores, usado para garantir que o que estão construindo o esperado. O Modelo de Caso de Uso consiste em duas partes, a saber, o diagrama de caso de uso e a especificação de casos de uso.

1. **Diagrama de caso de uso consiste de atores e casos de uso.** Ele apresenta a funcionalidade que o sistema proverá aos usuários para se comunicarem com o sistema. Cada caso de uso no modelo é descrito em detalhes usando as especificações de caso de uso.
2. **Especificações de caso de uso.** São documentos textuais que especificam as propriedades dos casos de uso tais como fluxo de eventos, pré-condições, pós-condições etc. O diagrama da ferramenta UML usado para definir o Modelo de Caso de Uso é o Diagrama de Caso de Uso.

O instrumento de diagramação UML utilizado para definir o Modelo de Caso de Uso é chamado de **Diagrama de Caso de Uso**.

#### Especificações Complementares

Contém aqueles requisitos que não mapeiam um caso de uso específico. Podem ser **requisitos não funcionais** tais como: manutenção de códigos fonte, usabilidade, confiabilidade,

<sup>3</sup> Object-oriented Analysis and Design Using the UML, Rational Rose Corporation, 2000, páginas 3-5

performance, e sustentabilidade, ou **restrições de sistema** que restringem nossas escolhas na construção da solução para o problema, tais como sistemas que devem ser desenvolvidos em Solaris e Java. É um importante complemento para o Modelo de Caso de Uso pois possibilita uma especificação completa dos requisitos do sistema a ser desenvolvido.

## Glossário

Define uma terminologia comum para todos os modelos. É usado pelos desenvolvedores para estabelecer um dialeto comum com os clientes e usuários finais. Deve haver um único glossário para todo o sistema.

## 4.2. Modelagem de Cenário

O Modelo de Caso de Uso é um mecanismo para captura de comportamentos desejados do sistema sem especificar como o comportamento será implementado. O ponto de vista dos atores interagindo com o sistema será usado na descrição dos cenários. **Cenários** são instâncias de funcionalidade que o sistema fornece. Ele captura as interações específicas que ocorrem entre os produtores e consumidores de dados, e o próprio sistema. É importante observar que a construção do modelo é um processo iterativo de refinamento.

## Diagrama de Caso de Uso de UML

Como mencionado na seção anterior, o Diagrama de Caso de Uso é utilizado como instrumento de modelagem para o Modelo de Caso de Uso. A Tabela 2 mostra a notação básica.


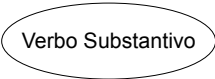
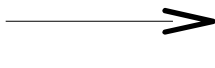
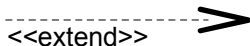
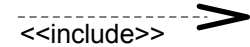
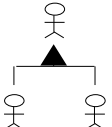
Notação Básica	Nome	Significado
 Ator-nome	<b>Ator</b>	Representar um conjunto coerente de papéis que usuários interpretam quando interagem com os casos de uso. Ele efetua chamadas ao sistema que por sua vez entrega um de seus serviços. Pode ser uma pessoa, um dispositivo, ou um outro sistema. Atores são nomeados com a utilização de substantivos.
 Verbo Substantivo	<b>Caso de Uso</b>	Descrever as funções que o sistema executa quando interage com atores. É uma seqüência de ações que proporcionam um resultado observável para os atores. É descrito utilizando-se frases verbo-substantivo.
ou 	<b>Associação</b>	Mostrar a relação ou associação entre um ator e o caso de uso, ou entre os casos de uso.

Tabela 2: Notação Básica do Diagrama de Caso de Uso

O Modelo de Caso de Uso pode ser refinado com a utilização de **estereótipos em associações e generalização de atores e casos de uso**. Na UML, estereótipos são elementos de modelagem especiais que representam um elemento de forma particular. São representados por uma palavra-chave envolta em chaves angulares (<< e >>) tais como <<extend>> e <<include>>. Generalização ou especialização seguem os mesmos conceitos que foram discutidos na seção Conceitos de orientação à objeto. A Tabela 3 mostra a notação estendida do Diagrama de Caso de Uso em UML.

Notação Básica	Nome	Significado
 <<extend>>	<b>Extend</b>	Mostrar que um caso de uso provê uma funcionalidade adicional que pode ser requerida por outro caso de uso.
 <<include>>	<b>Include</b>	Mostrar que quando um caso de uso é utilizado, outros (incluídos) também serão utilizados.
	<b>Generalização do Ator</b>	Pode haver casos no qual seria melhor mostrar um super-ator que comunica com um caso de uso em lugar de todos os atores que comunicam com o mesmo caso de uso, particularmente, quando todos eles interagirem com um sistema de mesma funcionalidade.

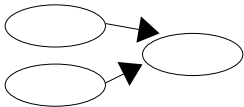
<b>Notação Básica</b>	<b>Nome</b>	<b>Significado</b>
	<b>Generalização do Caso de Uso</b>	Pode haver casos de uso semelhantes onde a funcionalidade comum é representada melhor generalizando toda a funcionalidade em um super caso de uso.

Tabela 3: Notação Estendida do Diagrama de Caso de Uso

## Desenvolvendo o Modelo de Caso de Uso

### PASSO 1: Identifique os atores

Identifique os atores externos que irão interagir com o sistema. Pode ser uma pessoa, um dispositivo ou outro sistema. Como exemplo, a Figura 2 identifica os atores para a Manutenção de Associação ao Clube do estudo de caso.

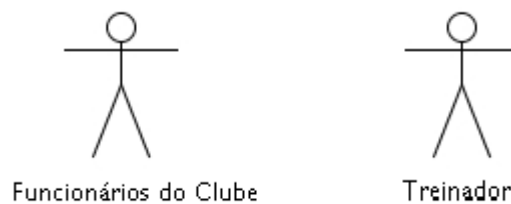


Figura 2: Atores de Manutenção de Associação ao Clube

Dois atores foram identificados, a saber, funcionários do clube e treinador.

### PASSO 2: Identifique os casos de uso

Identifique casos de uso que o sistema precisa executar. Lembre-se que casos de uso são seqüências de ações que dão um resultado observável aos atores. Como exemplo, a Figura 3 identifica os casos de uso iniciais.

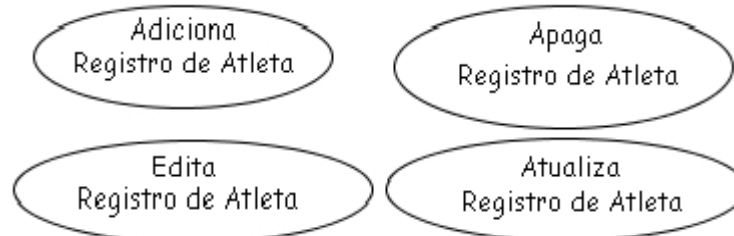


Figura 3: Casos de Uso de Manutenção de Associação ao Clube

Os seguintes casos de uso foram identificados.

- Adiciona Registro de Atleta
- Edita Registro de Atleta
- Apaga Registro de Atleta
- Atualiza Status de Atleta

### PASSO 3: Associe casos de uso com atores

A Figura 4 mostra a primeira iteração do modelo de caso de uso. Os casos de uso identificados são atribuídos aos dois atores.

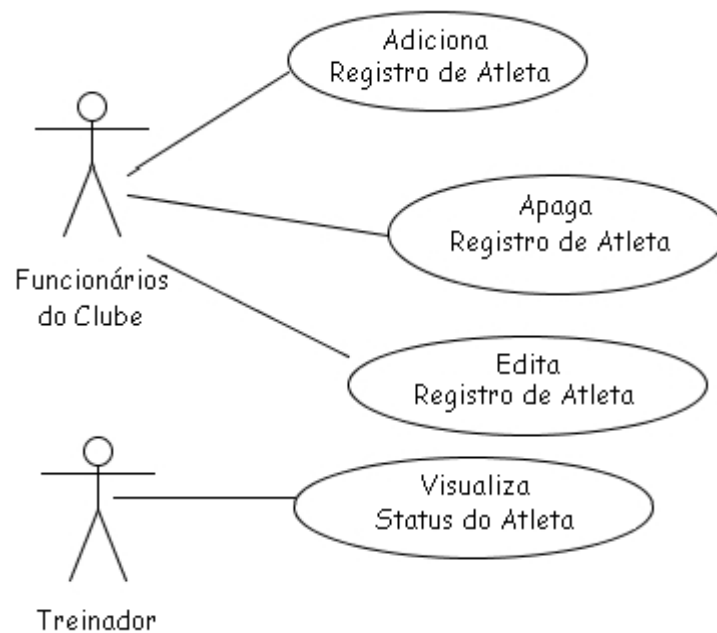


Figura 4: Primeira Iteração do Modelo de Caso de Uso de Associação ao Clube

#### PASSO 4: Refine e redefina o modelo

Este caso de uso pode ser refinado usando a notação melhorada. A segunda iteração do modelo de caso de uso é mostrada na Figura 5. Opcionalmente, numere os casos de uso.

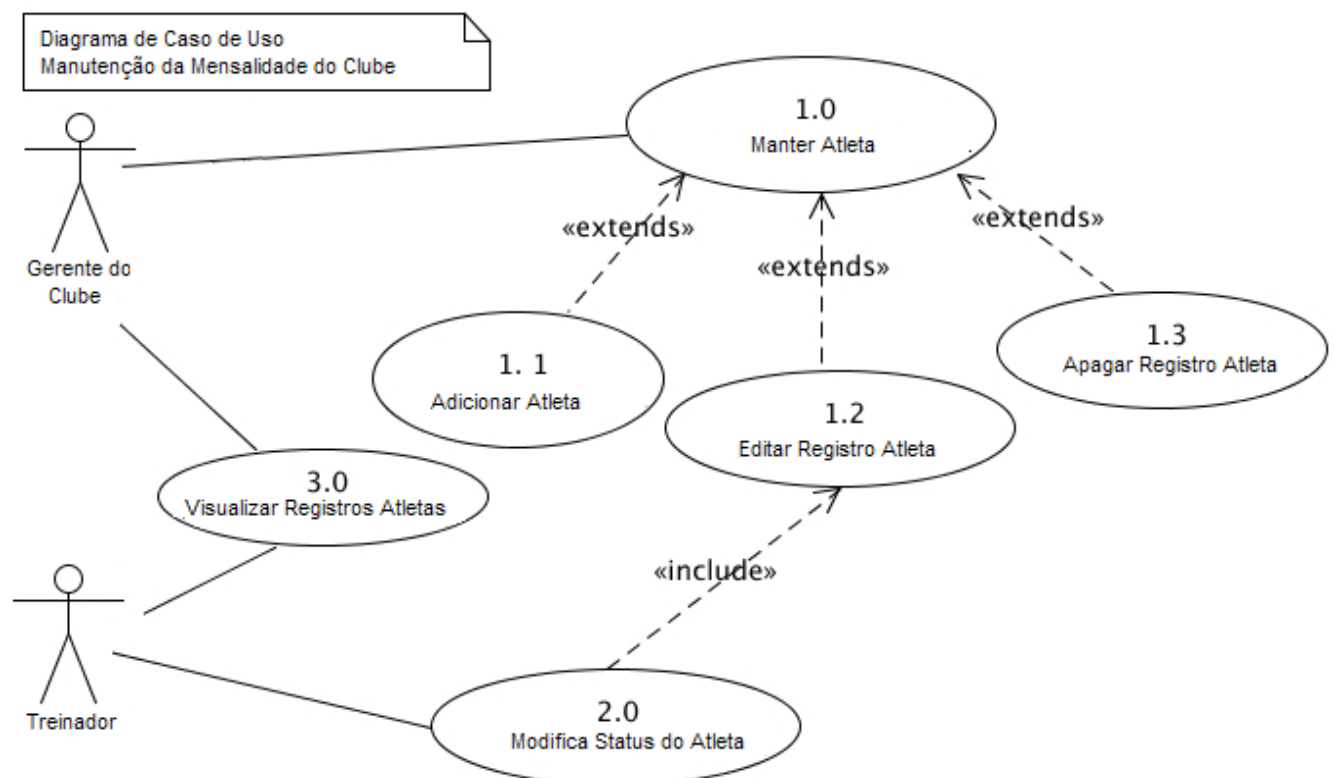


Figura 5: Segunda Iteração do Modelo de Caso de Uso de Associação ao Clube

Na segunda iteração, a elaboração é feita pela introdução do caso de uso **Manutenção de Registro de Atleta**. Isto é gerenciado pelos funcionários do clube. Estereótipos sobre associados são usados para incluir ou estender este caso de uso.

O caso de uso **Manutenção de Registro de Atleta** é estendido para ter as seguintes opções:

- 1.1 Adicionar Registro de Atleta



- 1.2 Editar Registro de Atleta
- 1.3 Remover Registro de Atleta

Toda vez que o caso de uso **Atualizar Registro de Atleta** é executado, um **Editar Registro de Atleta** é também executado.

Além disso, um outro caso de uso foi adicionado para especificar que determinados atores podem visualizar o registro do atleta (Caso de Uso **Visualizar Registro de Atleta**).

A modelagem é um processo iterativo. A decisão de quando parar a iteração é subjetiva e é tomada por quem modela. Para ajudar nesta decisão, responda às perguntas encontradas no *Checklist de Requisitos para a Validação de Modelos*.

### **PASSO 5: Para cada caso de uso, escreva a especificação do caso de uso**

Para cada caso de uso encontrado no Modelo de Casos de Uso, a especificação de caso de uso deve ser definida. A **especificação de caso de uso** é um documento onde os detalhes internos do caso de uso são especificados. Ela pode conter quaisquer das seções listadas abaixo. Contudo, as cinco primeiras são as seções recomendadas.

1. **Nome.** Este é o nome do caso de uso. Deve ser o mesmo que é encontrado no Modelo de Caso de Uso.
2. **Breve Descrição.** Descreve o papel e o propósito do caso de uso em poucas linhas.
3. **Pré-condições.** Nele são especificadas as condições necessárias antes da execução do caso de uso.
4. **Linha de Eventos.** Esta é a parte mais importante para a análise dos requisitos. São os eventos que descrevem o que o caso de uso irá fazer. Neste ponto, os cenários são identificados.
5. **Pós Condições.** Nesse item são especificadas as condições existentes após a execução do caso de uso.
6. **Relacionamentos.** Nesta seção, outros casos de usos são associados à corrente de acontecimentos usados na especificação deste caso de uso. Normalmente, eles são casos de uso entendidos <<extends>> ou incluídos<<include>>.
7. **Requisitos Especiais.** Eles contêm outros requisitos que não estão especificados nos diagramas que são semelhantes a requisitos não funcionais.
8. **Outros Diagramas.** Diagramas adicionais podem ajudar a esclarecer os requisitos. São utilizados como projetos protótipos, utilizados para esboçar o diálogo dos usuários e o sistema, etc.

A *linha de Eventos* possui as informações mais importantes para o funcionamento da modelagem dos casos de usos. Descrevem o que o caso de uso irá fazer; NÃO como o projeto sistema irá executar. Fornece uma descrição da sequência de ações necessária para o sistema dispor o serviço esperado pelo ator;

Um cenário é uma instância da funcionalidade do sistema. Ele também é descrito em paralelo a um Caso de Uso. A linha de Eventos de um caso de uso consiste em uma *linha básica* e várias *linhas alternativas*. Uma linha básica, normalmente, a execução mais simples do caso de uso, temos uma linha básica e constantemente modifica os endereços de linhas alternativas, onde os casos excepcionais serão tratados no caso de situações de erros.

Existem duas maneiras de demonstrar uma linha de evento – *textual* e *gráfica*. Para demonstrar graficamente uma linha de evento, é usado um Diagrama de Atividade da UML. Ele é usado similarmente a um fluxograma. Porém, são utilizados para modelar cenários de casos de uso. Utiliza-se a seguinte notação indicada na Tabela 4.

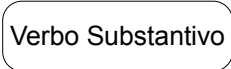

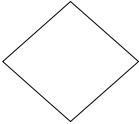

<b>Notação Básica</b>	<b>Nome</b>	<b>Significado</b>
	<b>Estado de Atividade</b>	Ela representa a execução de uma atividade ou uma execução interna de uma linha de trabalho. Atividades pode ser representados utilizando frases com verbos-substantivos .
	<b>Transição</b>	Mostra qual atividade será executada após a outra. Uma seta aponta a direção da próxima execução.
	<b>Decisão</b>	É utilizado para avaliar condições. O símbolo é semelhante ao utilizado em um fluxograma. Ele é usado para guardar as condições que determinem quais fluxos alternativos serão executados.
	<b>Barras de Sincronização</b>	São usados para mostrar sub-fluxos. É utilizado para ilustrar execuções concorrentes em uma linha de trabalho.

Tabela 4: Símbolos do Diagrama de Atividades

A Figura 6 é um exemplo de diagrama de atividade. Ilustra como atletas são inicialmente escalados para uma equipe. A Diretoria recebe os formulários preenchidos pelos atletas e prepara o teste simulado. Durante os testes, o comitê de seleção avalia o desempenho dos atletas. Se o atleta joga bem, é imediatamente colocado na Equipe de Competição. Caso contrário, é colocado na Equipe de Treinamento. A Diretoria manterá os registros dos atleta em arquivos apropriados.

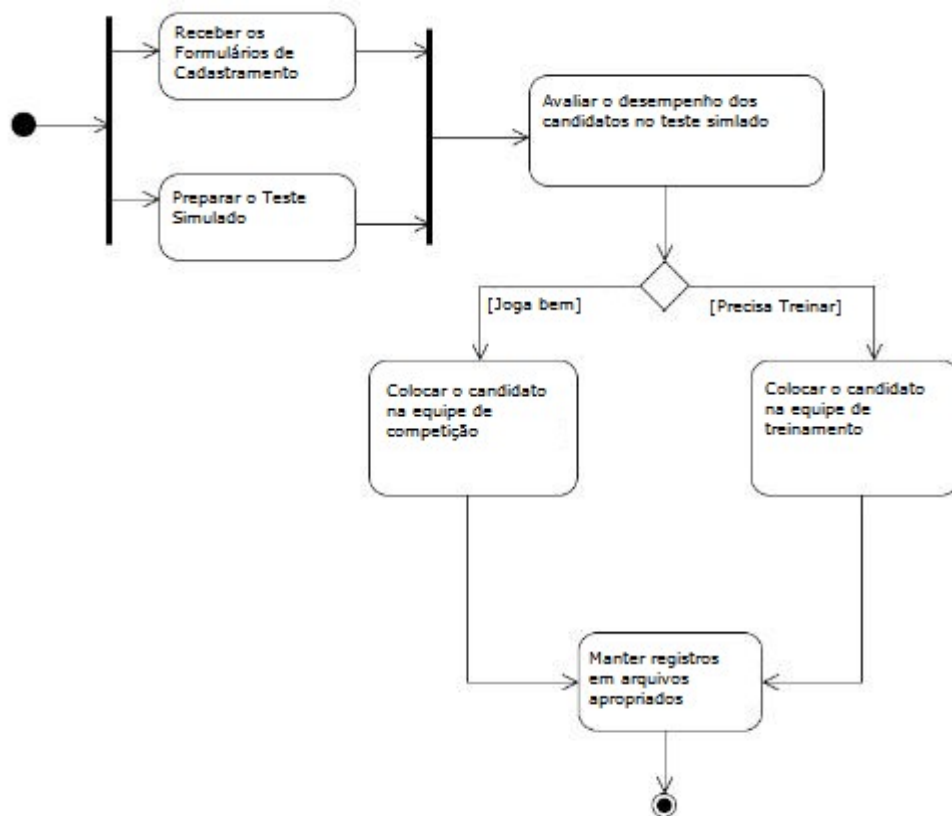


Figura 6: Recrutamento de Atletas

O Diagrama de Atividades pode ser modificado para o Diagrama **Swimlane**. Neste diagrama, atividades são alinhadas com base na responsabilidade dos atores para aquela atividade. A Figura 7 mostra o diagrama de atividades modificado, onde os atores responsáveis pela atividade são grafados acima da atividade.

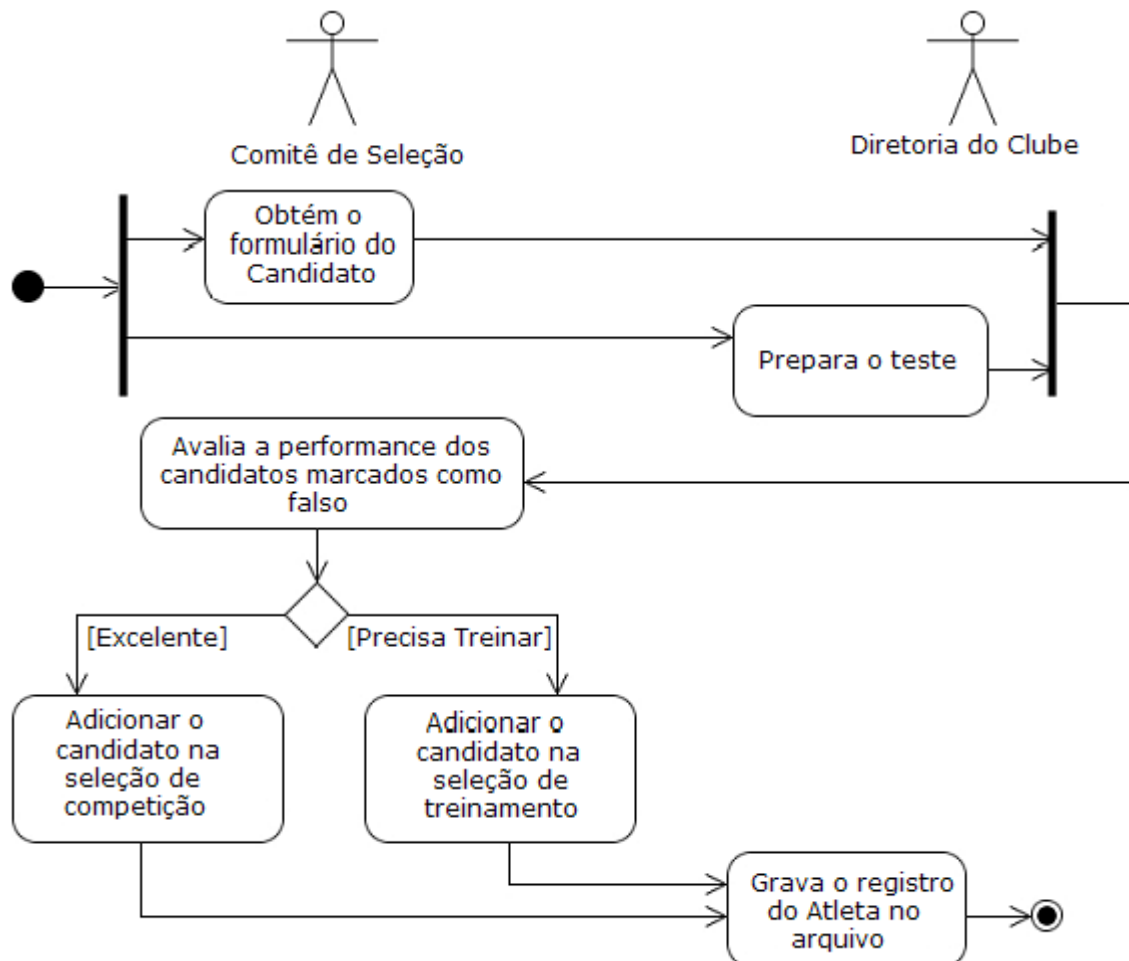


Figura 7: Atribuição inicial utilizando o Diagrama de Swimlane

Para auxiliar os engenheiros a definir o fluxo eventos, a lista de dicas é apresentada abaixo.

1. Para reforçar a responsabilidade do ator, inicie a descrição com "Quando o ator..."
2. Descrever a troca de dados entre o ator e o caso de uso.
3. Tentar não descrever os detalhes das interfaces, a menos que seja necessário.
4. Responda a todas as perguntas "porque". Os *designers* utilizarão essas respostas para identificar os Casos de Testes.
5. Evite terminologias tais como "Por exemplo,...", "processo" e "informação".
6. Descreva quando o caso de uso inicia e termina.

#### Passo 6: Refinar e Redefinir as especificações dos Casos de Uso

Similar ao desenvolvimento do Diagrama de Caso de Uso, pode-se refinar e redefinir suas especificações. Isto também é feito iterativamente. O momento de interromper a iteração depende da modelagem. O analista pode responder às perguntas presentes no Modelo de Validação de Requisitos para determinar quando encerrar as atividades de refinar e redefinir as especificações do caso de uso.

### 4.3. Modelo de Validação dos Requisitos

Assim como qualquer produto, em qualquer fase é preciso fazer uma validação dos requisitos. Abaixo, está uma lista das questões que servem de base para se criar o modelo de validação de requisitos. É importante notar que essa lista serve como um guia. O engenheiro pode adicionar ou retirar questões dependendo das circunstâncias e necessidades do desenvolvimento do projeto.

**Modelo de Validação para o Caso de Uso**

1. É possível entender o Diagrama de Caso de Uso?
2. É possível ter uma idéia clara de todas as funcionalidades do sistema?
3. É possível ver os relacionamentos entre as funcionalidades que o sistema precisa executar?
4. Todas as funcionalidades foram cumpridas?
5. O Diagrama de Caso de Uso contém algum comportamento inconsistente?
6. O Diagrama de Caso de Uso pode ser dividido em pacotes? Estão divididos apropriadamente?

**Lista de Validação para o Ator**

1. Todos os possíveis atores foram identificados?
2. Todos os atores identificados estão associados a pelo menos um Caso de Uso?
3. Um ator especifica um grupo? É necessário criar o grupo ou separar os atores?
4. Os atores possuem nomes intuitivos e descritivos?
5. Usuários e Cliente entendem os nomes dos atores?

**Validação para o Caso de Uso**

1. Todos os casos de uso estão associados a atores ou outros casos de uso?
2. Os casos de uso são independentes?
3. Existe algum caso de uso que apresenta um comportamento similar ou fluxo de eventos?
4. Foi dado a todos os casos de uso um nome único, intuitivo e explicativo?
5. Clientes e Usuários estão prontos para entender os nomes e as descrições dos casos de uso?

**Checklist de Validação da Especificação de Caso de Uso**

1. Podemos ver claramente o que o caso de uso deseja executar?
2. O objetivo do caso de uso está claro?
3. A descrição do caso de uso está clara e bem definida? Podemos entender? Incorpora o que o caso de uso está pretendendo fazer?
4. Está claro quando o fluxo de eventos inicia? Quando termina?
5. Está claro como o fluxo de eventos se inicia? Quando termina?
6. Podemos entender claramente as interações do ator e a troca de informação?
7. Há algum caso de uso complexo?

**Verificação de Validação do Glossário**

1. O termo está claro e conciso?
2. Os termos usados na especificação de caso de uso?
3. Os termos usados de maneira consistente no sistema? Há sinônimos?

## 5. Especificação de Requisitos

Na seção anterior, o modelo de requisitos foi introduzido e discutido. Nessa seção, veremos o modelo de análise que fornece um modelo base para as funções do software, o que irá ou não fornecer. Esse é o ponto principal que será desenvolvido na fase de engenharia de requisitos, pois serve de base para o projeto e construção do software. Aprenderemos como o modelo de análise pode ser derivado do modelo de requisitos.

### 5.1. O Modelo de Análise

O modelo de análise está ilustrado na Figura 8. Consiste de dois elementos, particularmente, **Modelo de Análise** e **Modelo Comportamental**.

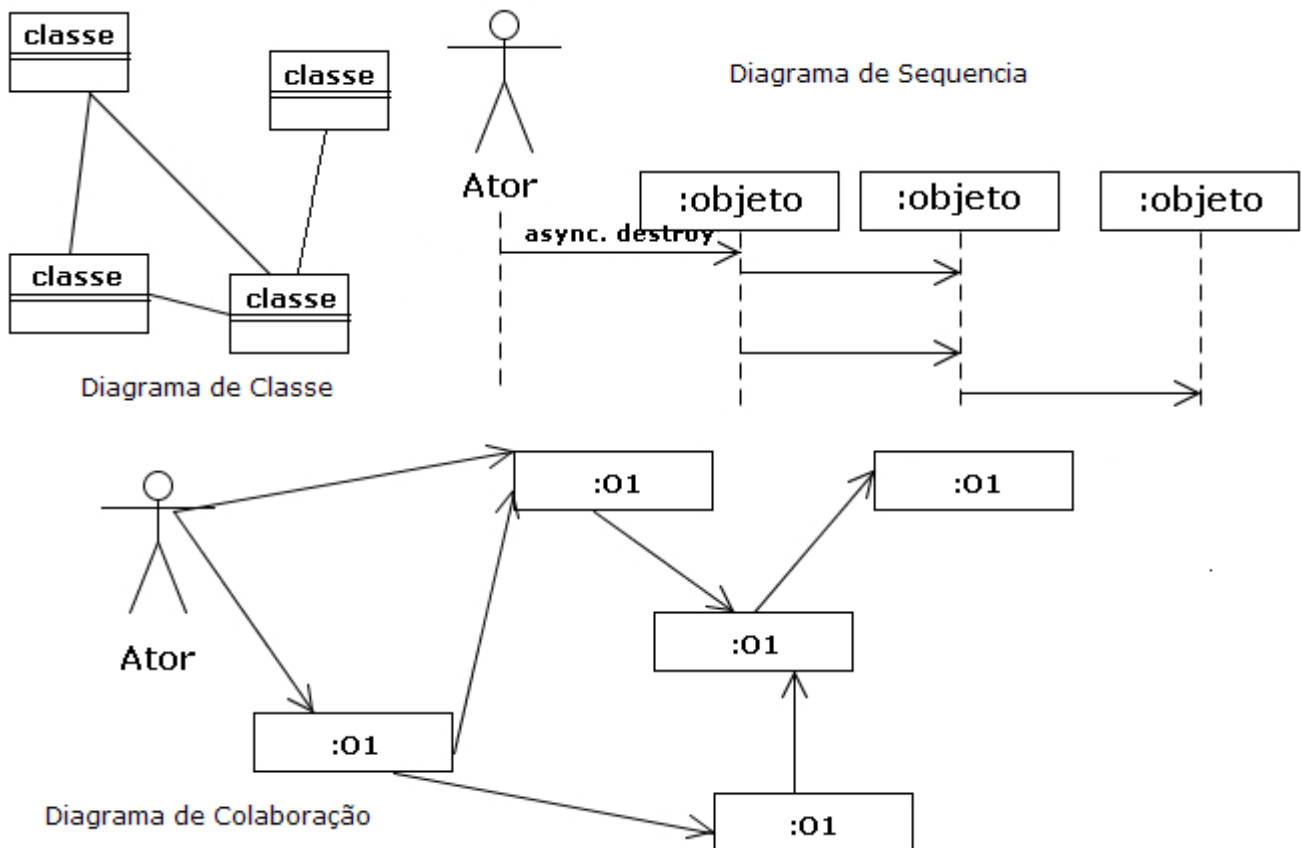


Figura 8: Modelo de Análise – Diagrama de Classe, de Seqüência e de Colaboração

Para criar o modelo de análise, os seguintes documentos de entrada são necessários.

- Modelo de Requisitos
- Documento de Arquitetura de Software (Opcional)

#### Modelo de Análise

É criado utilizando-se o Diagrama de Classe da UML. Este é o diagrama mais importante a ser desenvolvido, pois serve como entrada principal para a fase de projeto da engenharia. Consiste de **Classes de Análise** que representam um primeiro modelo conceitual para os aspectos do sistema que tenham responsabilidades e comportamentos. É usado para obter o primeiro esboço dos objetos que queremos que o sistema suporte sem decidir quantos deles serão suportados pelo hardware e quantos pelo software. Raramente sobrevive inalterado a fase de Engenharia do Projeto.

#### Modelo comportamental

Utiliza a criação de eventos ou o Diagramas de Interação da UML, consiste nos diagramas de Seqüência e de Colaboração. Representa os aspectos dinâmicos do sistema. Isto mostra a

interação e a colaboração entre classes de análise.

## 5.2. Técnica de Análise do Caso de Uso

É uma técnica derivada do Modelo de Análise e do Modelo de Requisitos. Aqui, o Modelo de Caso de Uso é analisado gramaticalmente para extrair classes de análise. São analisados a estrutura e comportamento das classes de análise. E então são documentados. Deve capturar o aspecto informador, funcional e de comportamento do sistema.

### Desenvolvendo o Modelo de Análise

#### Passo 1: Valide o Modelo do Caso de Uso.

O modelo de Caso de Uso é validado para conferir as exigências. Por vezes captura uma informação adicional que pode ser necessária para o entendimento do comportamento interno do sistema. Também é possível encontrar algumas exigências incorretas ou mal compreendidas. Em tal caso, deve ser atualizado o fluxo original de eventos, isto é, é necessário retornar à fase de análise do modelo de requisitos.

Pode-se utilizar uma aproximação de "Caixa Branca" descrevendo e entendendo o comportamento interno do sistema que precisa ser executado. Como um exemplo, considere as frases seguintes. Qual está clara?

1. O comitê de seleção obtém os formulários de aplicação.
2. O comitê de seleção recorre ao formulário de aplicação dos candidatos que são marcados como falso e armazenados na gaveta do gabinete de arquivamento com o título "Candidatos para aplicar o teste".

#### Passo 2: Para cada caso de uso, localizar as classes de análise.

Uma vez que o Modelo de Caso de Uso é validado, as classes de análise de candidatas são identificadas e são descritas em alguns frases.

Uma classe é uma descrição de conjuntos de objetos que compartilham os mesmos atributos, operações, relações e semântica. É uma abstração que enfatiza características pertinentes e suprime outras características. Dizemos que um objeto é um exemplo de uma classe. Uma classe é representada por uma caixa retangular com três compartimentos. O primeiro contém o nome da classe. O segundo contém uma lista de atributos. O último contém uma lista de operações (métodos). Veja na Figura 9 um exemplo de uma classe. Neste exemplo, o nome da classe é *Athlet*. Os atributos são mostrados no segundo compartimento que incluem *id*, *lastname*, *firstname*, *mi*, *address*, *postalCode*, *telephone*, *bday*, *gender* e *status*. Os métodos são listados no terceiro compartimento que inclui *addAthlete()*, *editAthlete()*, *deleteAthlete()* e *updateStatus()*.

«entity» Athlete
private int id private String lastname private String firstname private String mi private String address private String postalCode private String telephone private Date bday private char gender {M or F} private String status
public void addAthlete(Athlete a) public void editAthlete(Athlete a) public void deleteAthlete(Athlete a) public void updateStatus(Athlete a, String status)

Figura 9: Representação de uma classe

Três perspectivas são utilizadas para identificar as classes de análise. São o limite entre o sistema

e seu ator, a informação e os usos de sistemas, e a lógica de controle do sistema. Oferecem um modelo mais robusto porque isolam coisas que provavelmente iriam mudar em um sistema. Em UML, eles são determinados estereótipos e são descritos na Tabela 5.

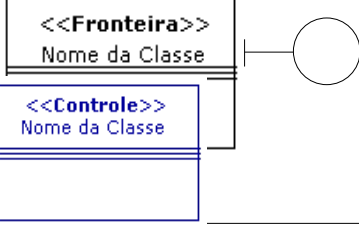
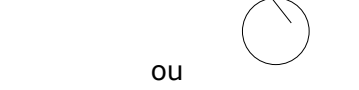

<b>Esteriótipo</b>	<b>Representação</b>	<b>Significado</b>
Fronteira		É usado como modelo de interação entre um ambiente de sistemas e trabalhos internos. Esclarece sistemas de fronteiras e auxilia o projetista ao permitir um bom ponto de partida para identificar serviços relacionados. Isola forças externas de mecanismos internos e vice-versa. Faz o intermédio entre a interface e algo fora do sistema. Existem três tipos: <b>interface-usuário</b> , <b>interface-sistema</b> e <b>interface-dispositivo</b> .
Controle		<p>A maior parte da funcionalidade do sistema é executada por essa classe. Proporciona comportamento que:</p> <ul style="list-style-type: none"> <li>• É independente do ambiente.</li> <li>• Define controle lógico e transações dentro de um caso de uso.</li> <li>• Requer pequenas mudanças se a estrutura interna ou o comportamento das classes de entidade mudar.</li> <li>• Obtém usuários ou um conjunto com o conteúdo de diversas classes de entidade, e portanto necessita coordenar o comportamento dessas classes de entidade.</li> <li>• Não é realizado da mesma forma toda vez que o mesmo é ativado.</li> </ul> <p>Proporciona coordenar os comportamentos do sistema. Separar as classes fronteira com as de entidade.</p>
Entidade		Representa depósitos de informações no sistema. É usado para guardar e atualizar informações sobre algum fenômeno, tal como um evento, uma pessoa ou algum objeto da vida real. As suas principais responsabilidades são armazenar e gerenciar informações do sistema. Representa os conceitos chaves do sistema sendo desenvolvido.

Tabela 5: Três estereótipos em UML

### Identificando Classes de Fronteira

Usando o Diagrama de Caso de Uso, uma classe fronteira existe para todo conjunto ator/caso de uso. Um exemplo é mostrado na Figura 10. A *RegistroAtletaUI* é considerada a classe de fronteira entre o ator *Pessoa Clube* e o caso de uso *Manter Registro Atleta*.

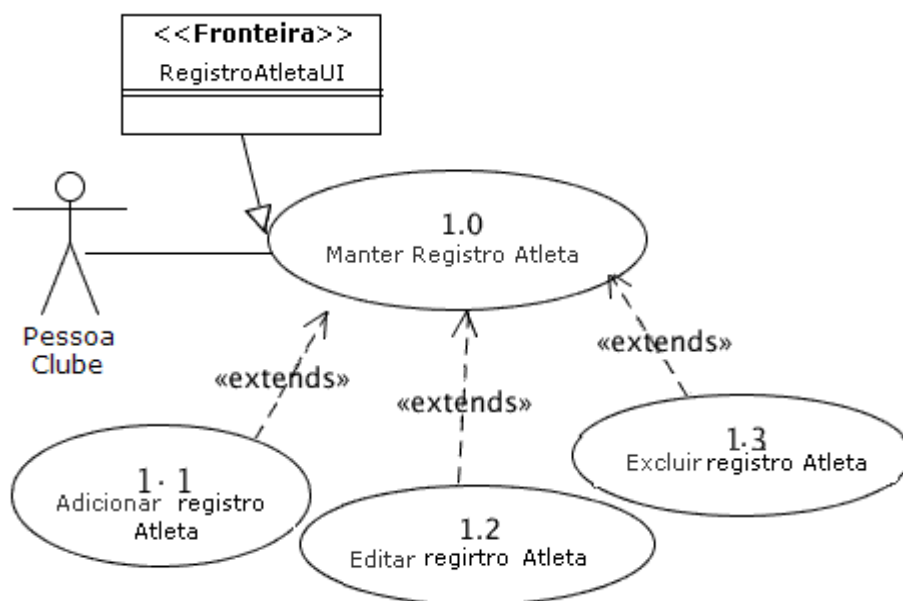


Figura 10: Sócio de clube - Exemplo de Classes Fronteira

## Identificando Classe de Controle

Usando o Diagrama de Caso de Uso, existe uma classe de controle para todo caso de uso. Um exemplo é mostrado na Figura 11. Quatro classes de controle foram identificadas, nomeadas, *ManutencaoRegistroAtleta*, *AdicionaAtleta*, *EditaAtleta* e *ExcluiAtleta*.

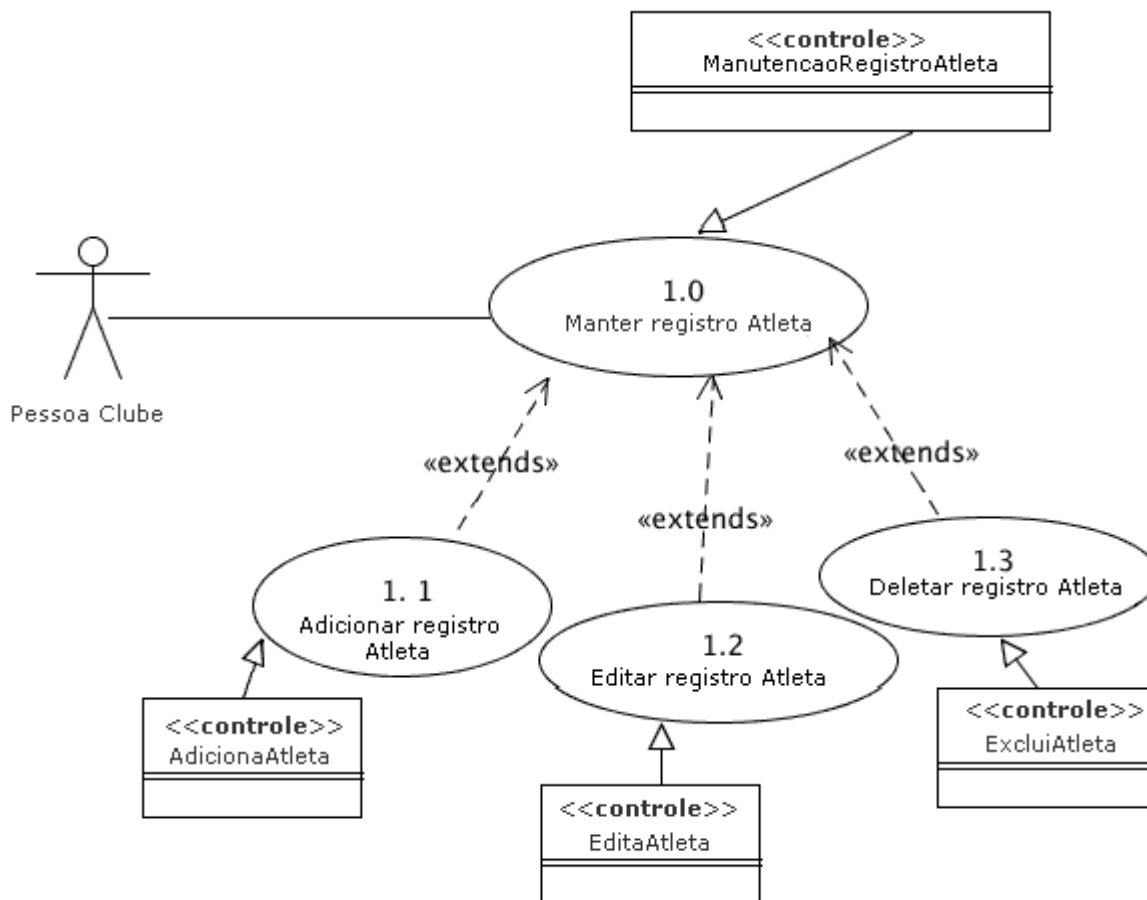


Figura 11: Sócio do Clube - Exemplo de Classes de Controle

## Identificando Entidades de Classes

Utilizando a especificação do caso de uso, entidades de classes podem ser encontradas:

- usando o fluxo de eventos do caso de uso como entrada
- pegando abstrações chaves dos casos de uso
- filtrando substantivos

Alguém pode empregar técnicas utilizadas para descobrir entidades em diagramas de Entidade-Relacionamento (DER). Um exemplo de entidades de classes derivadas dos casos de uso *Adicionar Registro de Atleta* são mostradas na Figura 12. São entidade de classes *Atleta*, *Segurança* e *StatusAtleta*.



Figura 12: Exemplo de entidades de classes Sócios do Clube



### Passo 3: Modelar o comportamento das classes de análise.

Para modelar o comportamento das classes de análise, modelamos a natureza dinâmica dentre as classes de análise. Utiliza-se os digramas de eventos ou interação da UML, especificamente, Seqüência e Colaboração. Quando modelamos o comportamento do sistema, nível-objeto é usado ao invés de nível-classe para permitir cenários e eventos que utilizam mais de uma instância de uma classe. Dois elementos básicos de modelagem são usados: **objetos** e **mensagens**.

**Objetos** são instâncias de uma classe. Eles podem ser **Objetos Nomeados** ou **Objetos Anônimos**. Figura 13 mostra exemplo de objetos.

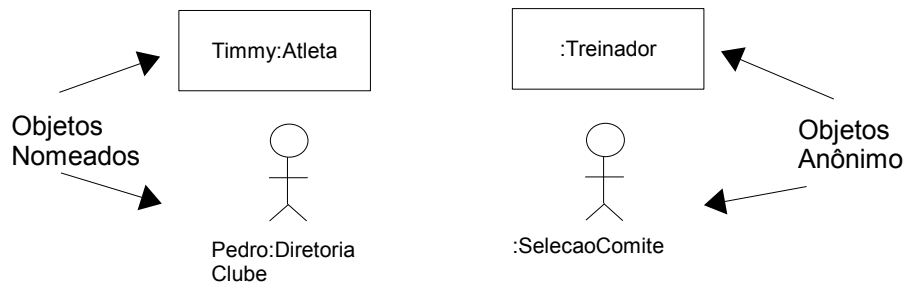


Figura 13: Exemplo de Objetos

Objetos Nomeados são identificados colocando seus nomes antes do nome da classe; são separados por dois-pontos (:). no diagrama, **Timmy:Atleta** é considerado um objeto nomeado. Objetos anônimos não possuem nomes; para representá-los, colocamos dois-pontos antes do nome da classe. **:Treinador** é um exemplo de objeto anônimo.

**Mensagens** são uma forma de comunicação entre objetos. Possuem o seguinte formato.

**\* [condição] : ação( lista de parâmetros ) : tipoValorRetorno**

Tabela 6 mostra uma descrição de cada componente da mensagem. Somente o componente **ação** é obrigatório.

Componentes da Mensagem	Significado
<b>*</b>	Representar qualquer número de repetições da ação.
<b>[condição]</b>	Representar uma condição que deve ser verdadeira antes que uma ação possa ser realizada.
<b>ação(lista de parâmetros)</b>	Representar uma ação que deve ser realizada por um objeto. A ação, opcionalmente, pode conter uma lista de parâmetros.
<b>tipoValorRetorno</b>	Caso uma ação retornar um valor, precisamos especificar o tipo desse valor.

Tabela 6: Componentes da Mensagem

Exemplos de mensagens são mostrados a seguir.

1. `submitApplication()`

Essa mensagem significa que uma aplicação foi submetida.

2. `[athleteIsHere = yes] playBasketball(athlete) : void`

Essa mensagem significa que, se atleta está aqui no teste, ele joga basquetebol.

3. `*[for each athlete] isAthleteHere(athlete) : boolean`

Essa mensagem verifica, para cada atleta, se o atleta está aqui. Retorna SIM se ele estiver aqui; caso contrário, retorna NÃO.

### Descrevendo os Diagramas de Seqüência

O **Diagrama de Seqüência** é usado para modelar interações entre objetos, o qual mapeia interações seqüenciais para interações de objeto. Ele mostra explicitamente a seqüência de

mensagens sendo passadas de um objeto para outro. É usado para especificar interação de tempo real e cenários complexos. Similar aos diagramas de caso de uso, há uma notação básica e uma notação aprimorada. Figura 14 Mostra a notação básica do diagrama de seqüência.

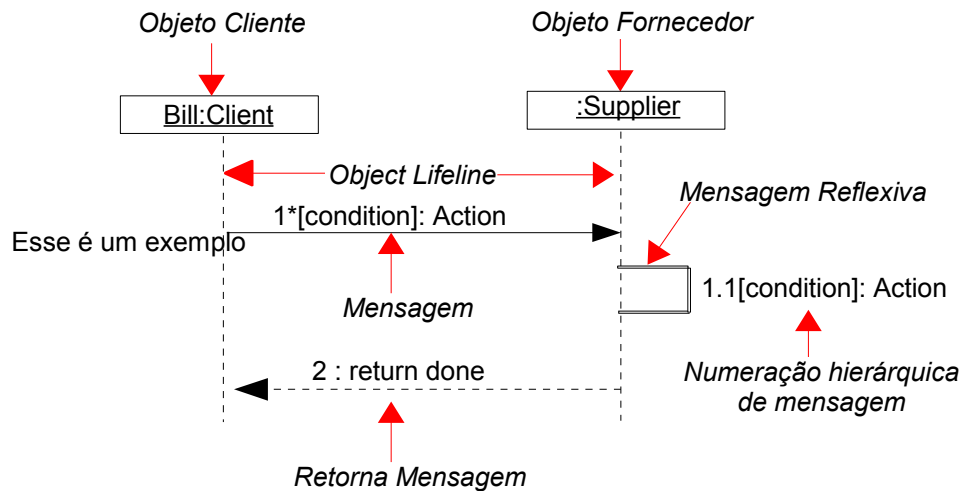


Figura 14: Notação Básica do Diagrama de Seqüência

Objetos Cliente são diferenciados dos Objetos Fornecedor. **Objetos Cliente** são os objetos enviando mensagens. São os que pedem um serviço para executar a ação especificada na mensagem. **Objetos Fornecedor** são os receptores das mensagens; eles são requisitados para executar a ação especificada pela mensagem. Objetos são dispostos horizontalmente. Cada objeto deve ter uma **linha de vida**. Ela representa a existência do objeto em um instante em particular. Uma **mensagem** é mostrada como uma linha horizontal com uma seta, a partir da linha de vida de um objeto para a linha de vida de outro objeto. Uma **mensagem reflexiva** mostra uma seta que começa e termina na mesma linha de vida. Cada seta é rotulada com um número seqüencial e a mensagem. Uma **mensagem de retorno** é mostrada como uma linha horizontal pontilhada, com a mensagem *return* opcionalmente seguida pelo valor de retorno.

Figura 15 mostra a notação aprimorada do Diagrama de Seqüência. Ela usa os conceitos de criação e término de objeto, ativação e desativação de objeto, e mensagens customizadas.

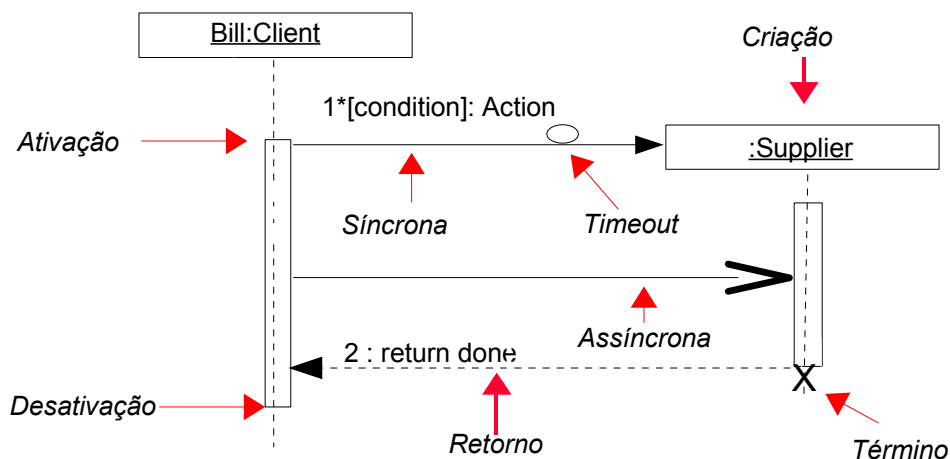


Figura 15: Notação Aprimorada do Diagrama de Seqüência

Colocando um objeto no topo do diagrama implica que o objeto existe antes do cenário iniciar. Se um objeto é criado durante a execução do cenário, posicione o objeto no ponto em que ele foi criado. Para mostrar que um objeto terminou, posicione um **X** no ponto onde o término ocorreu.

Para mostrar a ativação e a desativação de um objeto, usamos o **foco de controle**. Ele representa o tempo relativo em que o fluxo de controle é focado no objeto, assim representar o tempo de um objeto é direcionar mensagens. Isso é mostrado como um grande retângulo sobre a linha de vida do objeto. O topo do retângulo indica quando um objeto se torna ativo; a parte de

baixo do retângulo indica quando um objeto se tornou inativo.

Mensagens podem ser customizadas para serem mensagens síncronas ou assíncronas. **Mensagens síncronas** são usadas quando o objeto cliente espera pela resposta do objeto Fornecedor antes de retomar a execução. **Mensagens Assíncronas** são usadas quando o objeto cliente não precisa esperar pela resposta do objeto Fornecedor. Para diferenciar entre mensagens síncronas e assíncronas, olhe para a ponta da seta. Se a ponta da seta é preenchida, então é síncrona. Se a ponta da seta é uma linha, então é assíncrona.

**Timeout** é usado quando o cliente abandona a mensagem se o objeto Fornecedor não responder em um dado período de tempo.

### Desenvolvendo o Diagrama de Seqüência

Usamos o fluxo de eventos definidos em uma especificação de caso de uso, para ajudar-nos a definir os **diagramas de seqüência**. Como foi mencionado, a modelagem do comportamento é feita em nível de objeto em lugar de nível de classe. Isto significa que, para todo cenário definido no fluxo de eventos devemos ter um diagrama de seqüência. No exemplo seguinte, a adição bem sucedida do registro do atleta do Caso de Uso Adicionar Registro do Atleta, deverá ser usada. Neste caso, o fluxo básico é considerado.

1. Enfileirar os objetos participantes da análise no topo do diagrama. Como regra, sempre colocar o objeto de controle entre os objetos de limite e objetos de entidade. Lembrar que o objeto de controle serve como um intermediário entre estes tipos de objetos. De preferência, todos os atores são colocados ou no princípio ou fim da linha. Um exemplo é mostrado na Figura 16. Neste exemplo, foram identificados três objetos de análise e um ator. Especificamente, eles são *AthleteRecordUI*, *AddAthlete*, *Athlete* e *Club Staff*. Observado que a classe de controle *AddAthlete* está colocada entre os objetos *AthleteRecordUI* e *Athlete*. Serve como o objeto intermediário entre o limite e a entidade. O objeto *Club Staff* é o ator principal deste cenário.

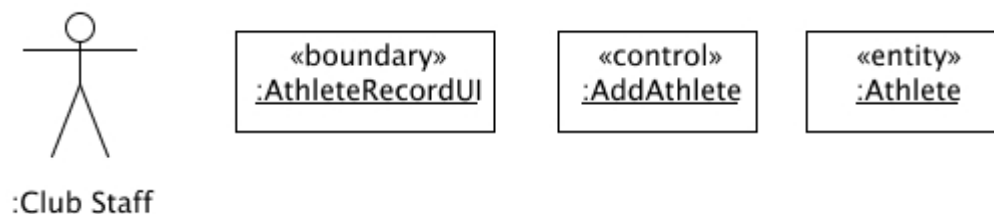


Figura 16: Formação de Classes e Atores

2. Desenhar uma linha tracejada vertical abaixo de cada objeto para representar a linha de vida do objeto. A Figura 17 ilustra um exemplo;

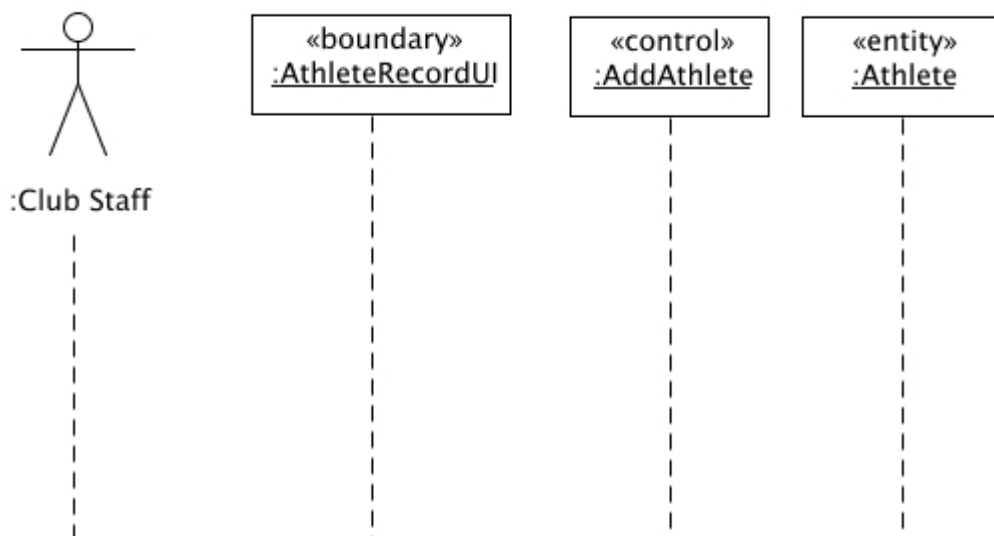


Figura 17: Com Linha de Vida do Objeto

3. Desenhar as mensagens síncrona que ocorrem de um objeto para outro. Como regra, nenhuma mensagem deve ser enviada entre o objeto fronteira e o objeto entidade. Devem se comunicar indiretamente através do objeto de controle. Figura 18 mostra um exemplo.

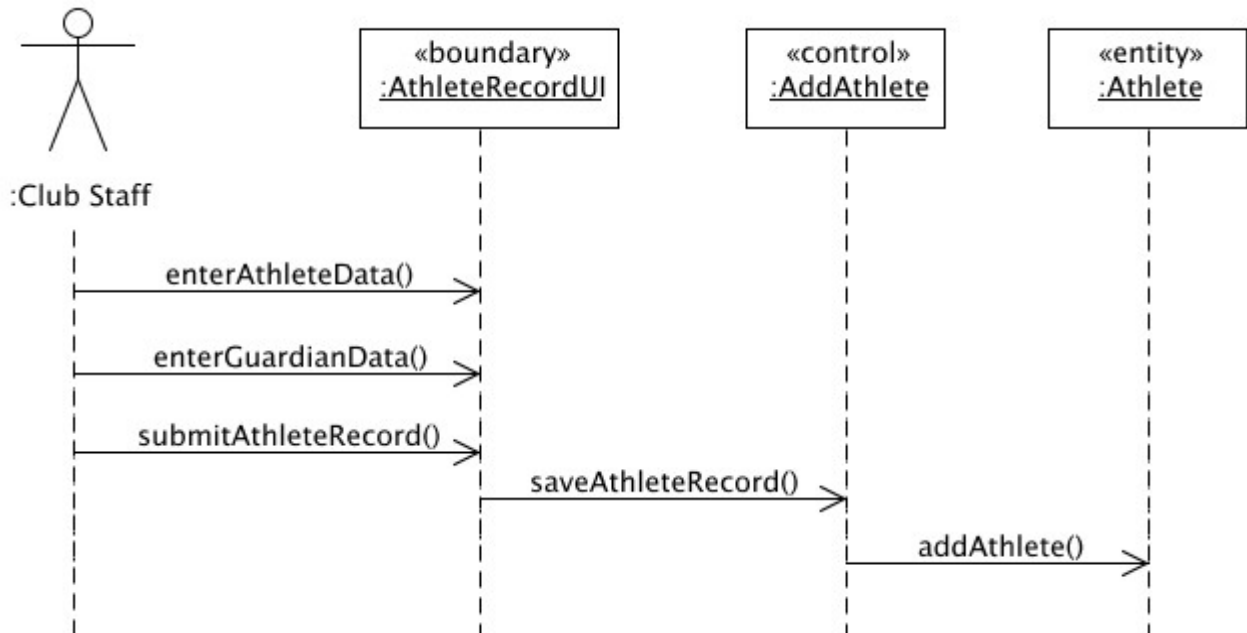


Figura 18: Com Mensagens enviadas para os Objetos

Aqui, o objeto *Club Staff* entra com a informação do atleta e do guarda através do objeto de fronteira *AthleteRecordUI* enviando as mensagens *enterAthleteData()* e *enterGuardianData()*. Para dizer que um registro do atleta está salvo, o assistente do clube envia uma mensagem *submitAthleteRecord()* para o objeto de controle *AddAthlete*. Essa classe, para seguir o fluxo, envia uma mensagem, *addAthlete()*, para o objeto de entidade *Athlete* para que possa adicionar o registro.

4. Desenhar as mensagens de retorno. Um exemplo é exibido na Figura 19. O objeto de entidade *Athlete* retorna o estado de feito para o objeto de controle *AddAthlete*. Semelhantemente, o objeto de controle *AddAthlete* retorna o estado de feito para o objeto de fronteira *AthleteRecordUI*.

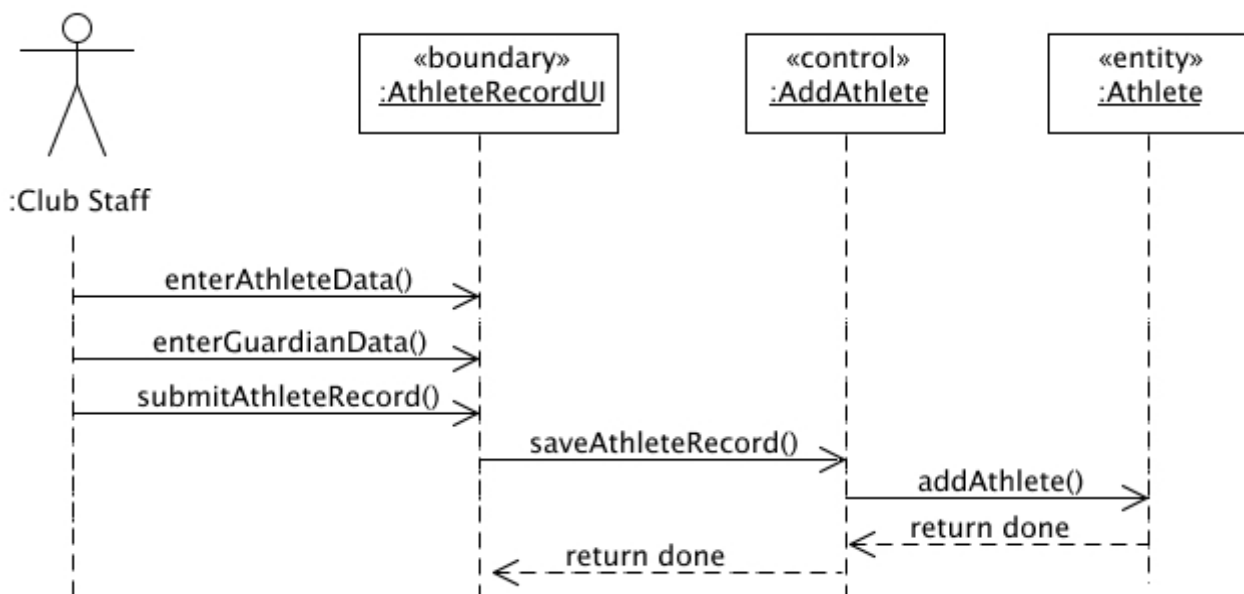


Figura 19: Com Mensagens de Retorno

5. Enumerar as mensagens de acordo com a ordem cronológica de sua invocação. Figura 20 mostra um exemplo.

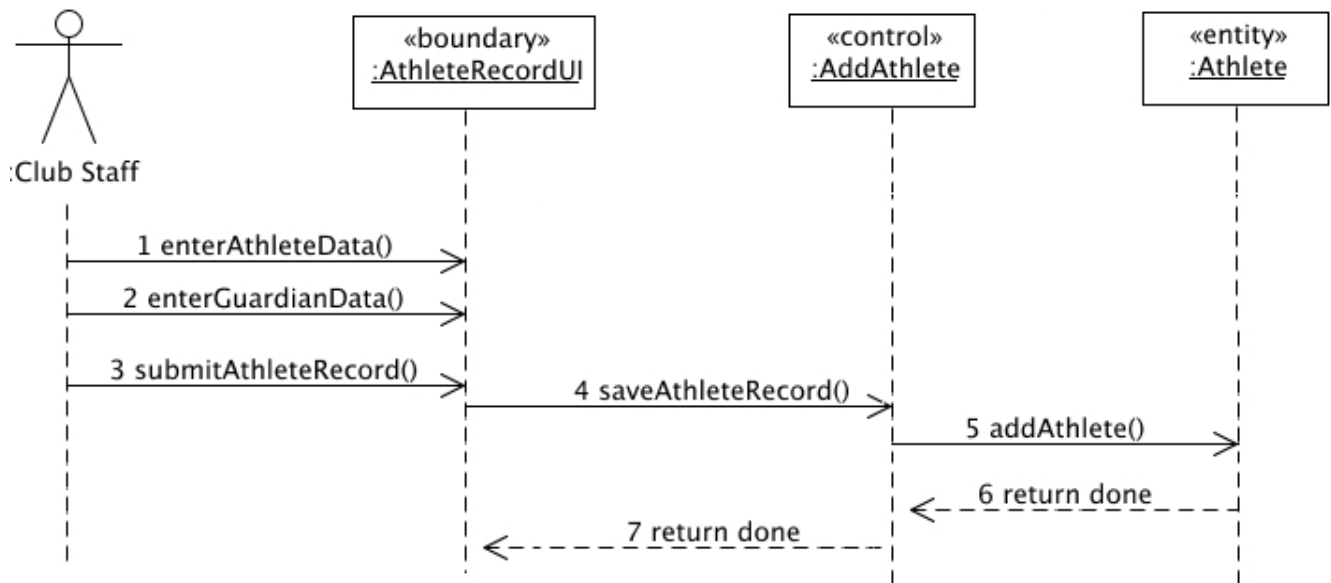


Figura 20: Com Números de Mensagem

A numeração das mensagens é como segue:

1. enterAthleteData()
2. enterGuardianData()
3. submitAthleteRecord()
4. saveAthleteRecord()
5. addAthlete()
6. return done
7. return done

6. Inserir nas mensagens as seguintes especificações: repetição, condições, lista dos parâmetros e tipo do retorno. Figura 21 mostra uns exemplos.

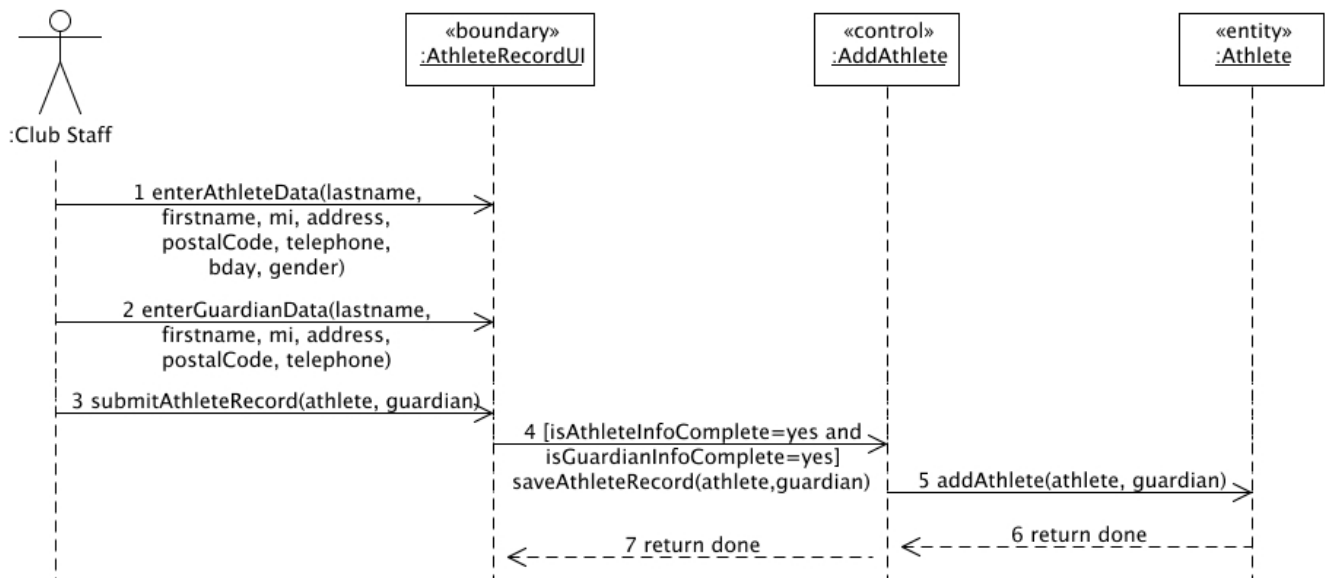


Figura 21: Com Mensagens Elaboradas

As mensagens elaboradas são como segue:

- a) **enterAthleteData()** a mensagem foi elaborada para incluir os seguintes parâmetros: **lastname, firstname, mi, address, postalCode, telephone, bday e gender**.
  - b) **enterGuardianData()** a mensagem foi elaborada para incluir os seguintes parâmetros: **lastname, firstname, mi, address, postalCode e telephone**.
  - c) **submitAthleteRecord()** a mensagem foi elaborada para incluir os seguintes parâmetros: **athlete** e **guardian**, o objeto que representa as informações do atleta e responsável, respectivamente.
  - d) **saveAthleteRecord()** a mensagem foi elaborada para incluir os seguintes parâmetros: **athlete** e **guardian**, o objeto que representa as informações do atleta e responsável, respectivamente. Também, foram identificadas duas condições que devem ser verdadeiras para que a mensagem seja executada. Elas são **[isAthleteInfoComplete = yes] e [isGuardianInfoComplete = yes]**, que significa que o registro do atleta é salvo somente se as informações do atleta e do responsável, estão completas.
  - e) **addAthlete()** a mensagem foi elaborada para incluir os seguintes parâmetros: **athlete** e **guardian**, o objeto que representa as informações do atleta e responsável, respectivamente.
7. Opcionalmente, podemos aprimorar o diagrama de seqüência utilizando a notação estendida. A Figura 22 mostra um exemplo.

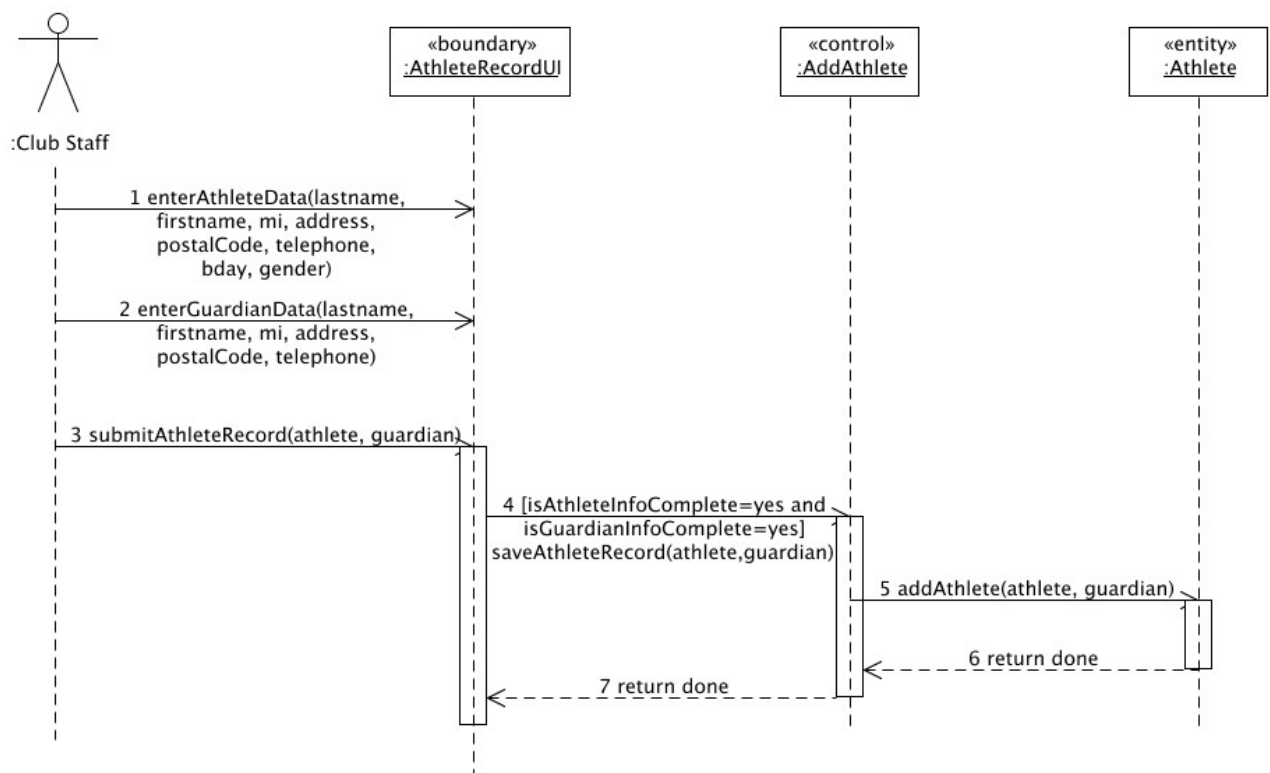


Figura 22: Diagrama de Seqüência Estendido para Add Athlete Record

Este pode não ser o único diagrama de seqüência para o caso de uso Add Athlete Record. Outros cenários para este Caso de Uso são: informações incompletas do atleta, informações incompletas do responsável, recorde do atleta já existe, etc. Para cada cenário identificado, um diagrama de seqüência correspondente é criado.

## Descrevendo o Diagrama de Colaboração

O **Diagrama de Colaboração** é utilizado para modelar um padrão de interação entre objetos. Ele modela a participação dos objetos nas interações pelas ligações entre cada um deles e pelas mensagens trocadas entre si. Ele valida o diagrama de classes demonstrando a necessidade para cada associação. Modela efetivamente a mensagem no objeto receptor, o que define uma interface

para aquele objeto. A Figura 23 mostra a notação do diagrama de colaboração.

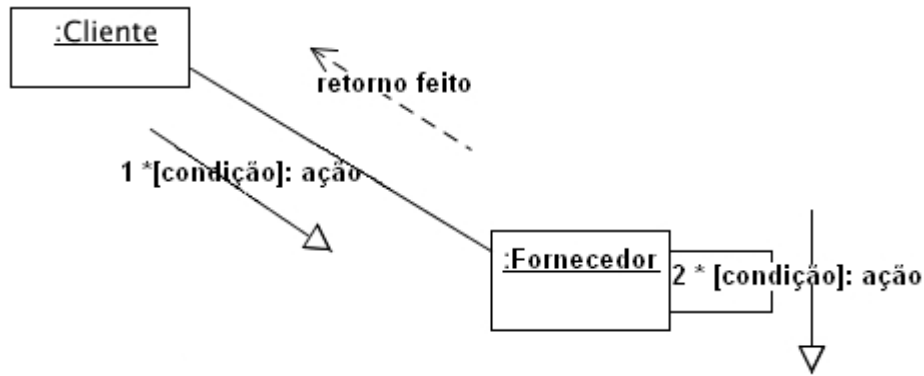


Figura 23: Diagrama de Colaboração

A notação utilizada no Diagrama de Colaboração é similar ao Diagrama de Seqüência exceto que não modelamos as mensagens em ordem cronológica. Somente mostramos as interações das mensagens.

### Desenvolvendo os Diagramas de Colaboração

Diagramas de Colaboração podem ser derivados dos Diagramas de Seqüência. Similarmente ao Diagrama de Seqüência, o nível do objeto é usado na modelagem. Conseqüentemente, para todo Diagrama de Seqüência, um Diagrama de Colaboração será criado. No exemplo a seguir, o cenário da adição bem sucedida de um registro de atleta do Caso de Uso Add Athlete (Adicionar Atleta) é usado.

1. Desenhar os atores e objetos participantes. Figura 24 mostra um exemplo.



Figura 24: Atores e Objetos

Novamente, os objetos da análise *AthleteRecordUI*, *AddAthlete* e *Athlete* são usados com o *Club Staff*.

2. Se os objetos trocam mensagens como mostrado nos Diagramas de Seqüência, desenhar uma linha conectando-os, o que significa uma associação. As linhas representam as ligações. Por exemplo, a Figura 25 demonstra um caso. O *Club Staff* troca mensagens com o *AthleteRecordUI*. O *AthleteRecordUI* troca mensagens com o *AddAthlete*. E para terminar, o *AddAthlete* troca mensagens com o *Athlete*.

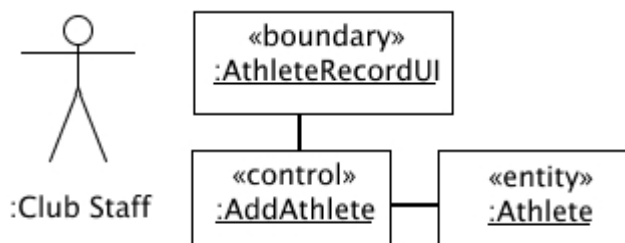


Figura 25: Atores e Objetos com as ligações

3. Colocar as mensagens nas linhas de associação. É mostrado um exemplo na Figura 26. As mensagens são representadas como setas com a mensagem escrita sobre elas. Como exemplo, a mensagem número 1 é a mensagem `enterAthleteData()` com seus parâmetros correspondentes. A definição completa da mensagem deve ser escrita.

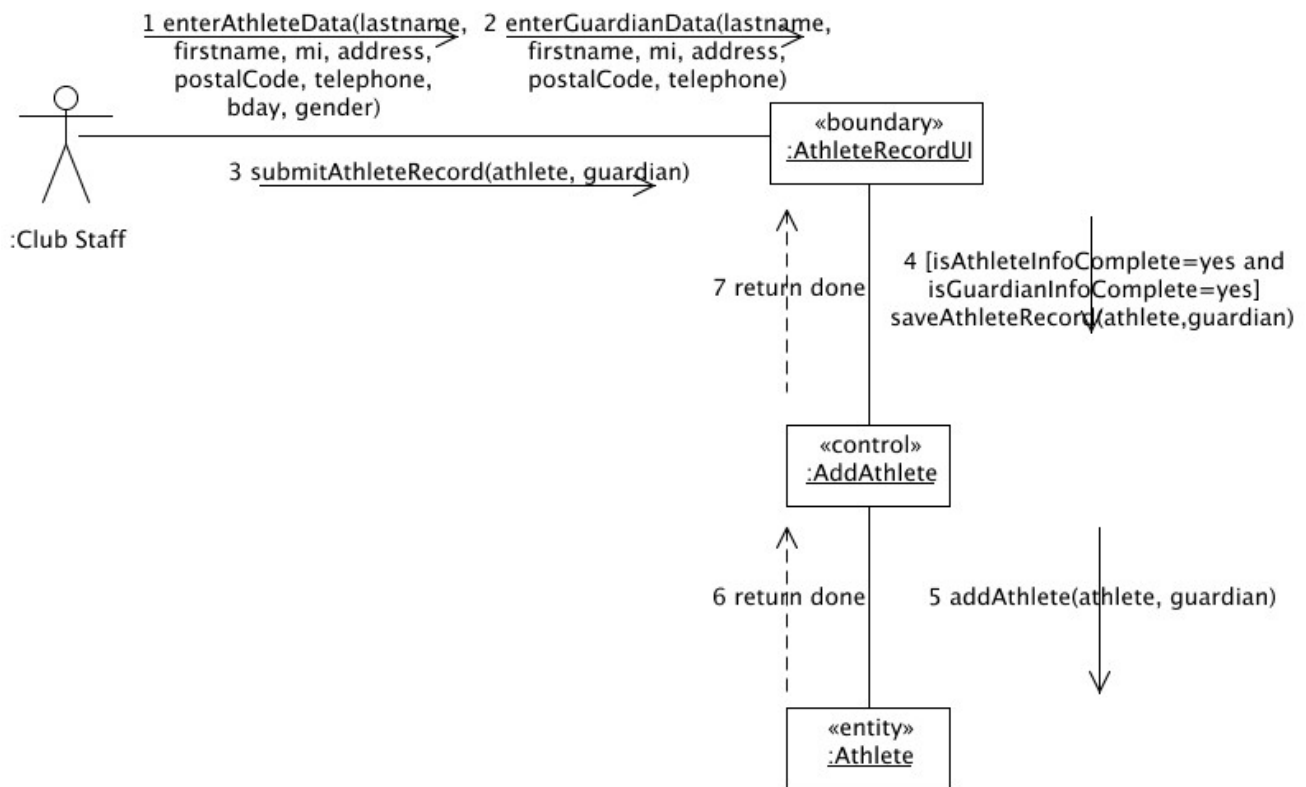


Figura 26: Diagrama de Colaboração para inclusão um registro Atleta

Este é o diagrama de colaboração do cenário realizado com sucesso para a inclusão de um Atleta no Caso de Uso **Incluir Atleta**. Similar ao Diagrama de Seqüência, o Diagrama de Colaboração é criado para cada seqüência que for identificada.

#### Passo 4: Para cada classe de análise resultante, identificar as responsabilidades.

Uma responsabilidade é algo que um objeto fornece a outros objetos ou atores. Pode ser uma ação que o objeto pode realizar ou um conhecimento que um objeto pode conter e fornecer a outros objetos. Pode ser derivado de mensagens obtidas de eventos ou diagramas de interação.

Nos diagramas de colaboração, as mensagens de entrada são a responsabilidade da classe. Para se obter as responsabilidades das classes de análise do Caso de Uso **Incluir Atleta**, precisamos utilizar os diagramas de colaboração. A Figura 27 mostra as responsabilidades das classes do Caso de Uso **Incluir Atleta** utilizando diagramas de colaboração. Neste exemplo, as responsabilidades são `enterAthleteData()`, `enterGuardianData()` e `submitAthleteRecord()`. A responsabilidade de `AddAthlete` é `saveAthleteRecord()`. Finalmente, as responsabilidades de `Athlete` são `addAthlete()`, `editAthlete()`, `deleteAthlete()` e `updateStatus()`.

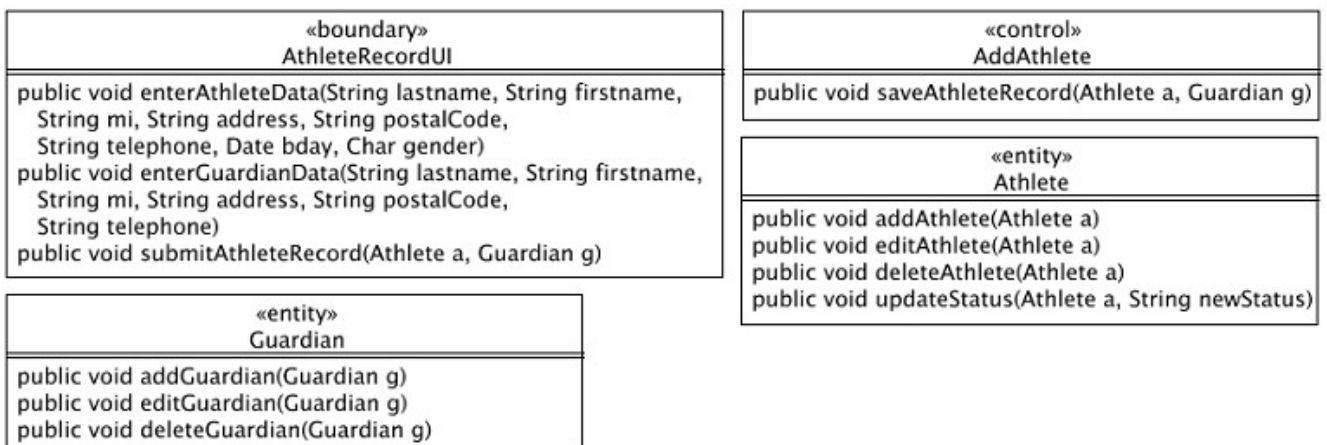


Figura 27: Identificação da Responsabilidade



Para ajudar a identificar a responsabilidade da classe, utilize os seguintes guias.

1. Verificar se as classes têm responsabilidades consistentes. Se parecer que as responsabilidades são disjuntas, divida o objeto em duas ou mais classes. Atualizar os Diagramas de Seqüência e Colaboração para refletir as mudanças.
2. Verificar se existem classes com responsabilidades similares. Se houver classes com as mesmas responsabilidades, combine-as. Atualizar os Diagramas de Seqüência e Colaboração para refletir as mudanças.
3. A melhor distribuição das responsabilidades pode ser evidente ao se trabalhar com outro diagrama. Em tal caso, é melhor e mais fácil não modificar e não atualizar diagramas precedentes. Ajuste os diagramas sem pressa e corretamente, mas não fique ansioso tentando otimizar as interações da classe.
4. Não há nenhum problema se uma classe tiver uma responsabilidade, pergunte-se se é uma classe realmente necessária. É necessária para justificar sua existência?

Compilar todas as responsabilidades vistas em todos os Diagramas de Colaboração em uma única definição da classe.

#### **Etapas 5: Para classes resultantes de cada análise, identificar os atributos.**

Um atributo é uma informação que um objeto possui ou conhece sobre si mesmo. É usado para armazenar informação. Identificar atributos especifica a informação que uma classe de análise é responsável por manter. São especificados usando o seguinte formato:

**atributo : tipo\_de\_dado = valor\_padrão {constraints}**

Tabela 7 dá a descrição de cada componente da especificação do atributo. O tipo de dados do atributo é requerido.

<b>Componente do atributo</b>	<b>Significado</b>
<b>atributo</b>	É o nome do atributo. Ao nomear um atributo certifique-se de que descreve o valor que contém.
<b>tipo_de_dado</b>	Especifica o tipo do atributo. Preferivelmente, use os tipos de dados definidos na Linguagem de Programação Java. Evite utilizar os tipos de dados definidos pelo usuário.
<b>valor_padrão</b>	É o valor padrão conferido ao atributo quando um objeto é instanciado.
<b>constraints</b>	Representa as restrições adicionais aplicadas aos valores possíveis do atributo tais como escalas numéricas.

*Tabela 7: Especificações do atributo*

Abaixo estão alguns exemplos de especificação de atributos.

1. NumeroConta: Numérico – um atributo nomeado NumeroConta contém um valor numérico.
2. Temperatura: Numérico {4 casas decimais} – um atributo nomeado Temperatura contém um valor numérico com um formato de quatro casas decimais.
3. NumeroSequencia: Inteiro = 0 {atribuído seqüencialmente pelo sistema} – um atributo nomeado NumeroSequencia contém um inteiro com um valor padrão igual a 0. Sucessivamente, lhe será conferido um número atribuído seqüencialmente pelo sistema.

Para ajudar a descobrir e especificar atributos, pode-se usar o roteiro abaixo.

1. Fontes de atributos possíveis são conhecimentos, exigências e glossário do domínio do sistema.
2. Atributos são usados no lugar de classes quando:
  - Somente o valor da informação, não sua posição, é importante.
  - A informação é possuída exclusivamente pelo objeto ao qual pertence; nenhum outro objeto referencia a informação.
  - A informação é alcançada somente pelas operações que recuperam, alteram ou executam transformações simples na informação; a mesma não tem nenhum comportamento real a

não ser o de fornecer seu valor.

3. Se a informação tiver comportamento complexo, ou for compartilhada por dois ou mais objetos, a mesma deve ser modelada como uma classe separada.
4. Atributos dependem do domínio.
5. Atributos devem suportar pelo menos um caso do uso.

No exemplo da Figura 28, são mostrados os seguintes atributos identificados. Semelhante ao passo anterior, todos os atributos vistos nos diagramas de colaboração devem ser reunidos em uma única classe de análise. Como um exemplo, a análise classe *Athlete* tem os seguintes atributos: *id*, *lastname*, *firstname*, *mi*, *address*, *postalCode*, *telephone*, *bday*, *gender*, e *status*.

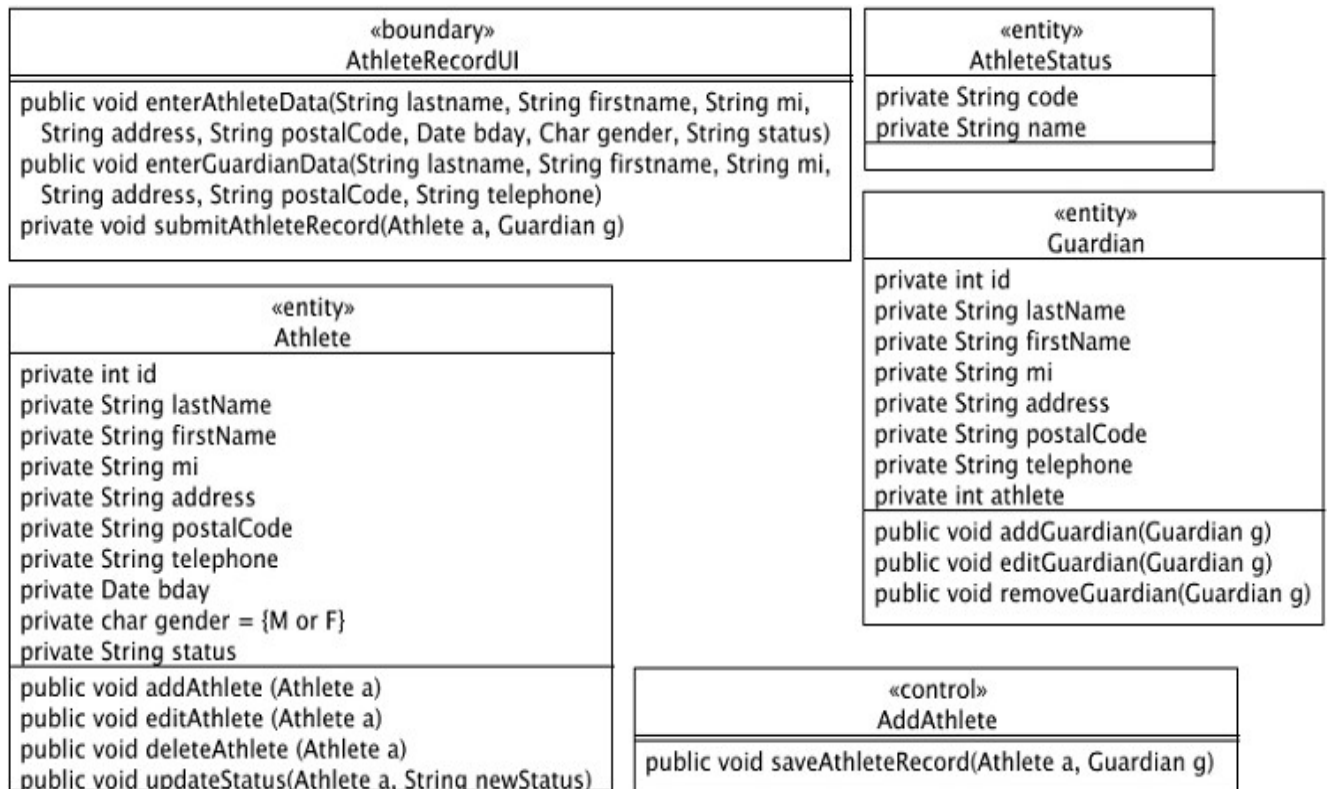


Figura 28: Identificação dos atributos

### Passo 6: Para cada classe de análise resultante, identifique as associações.

Uma associação representa as relações estruturais entre objetos de classes diferentes. Conecta instâncias de duas ou mais classes em um momento de duração. Pode ser:

1. **Associação Unária** – relação de objetos da mesma classe.
2. **Associação Binária** – relação de dois objetos de classes diferentes.
3. **Associação Ternária** – relação de três ou mais objetos de classes diferentes.

A Figura 29 mostra exemplos dos tipos de associações. O primeiro exemplo mostra uma associação unária onde um gerente administra zero ou muitos empregados e um empregado está sendo administrado por um gerente. O segundo exemplo mostra uma associação binária onde um atleta pertence a zero ou a um time enquanto um time tem muitos atletas.

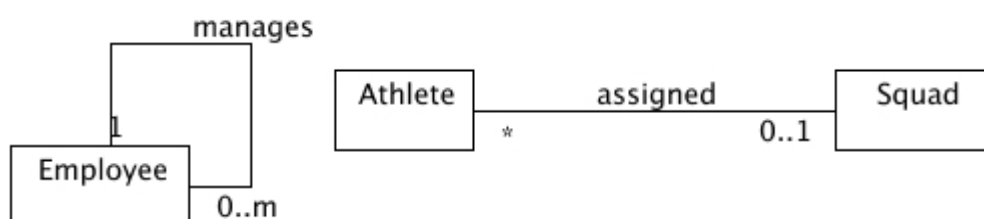


Figura 29: Tipos de Associações

Agregação é uma forma especial de associação que modela uma relação de parte-inteira entre um agregado e sua parte. Algumas situações onde uma agregação é apropriada são:

- Um objeto é fisicamente composto de outros objetos.
- Um objeto é uma coleção lógica de outros objetos.
- Um objeto contém outros objetos fisicamente.

Agregação é modelada na Figura 30 que mostra uma outra maneira de modelar o atleta e sua relação com o time. Um time é uma agregação de zero ou muitos atletas. O fim da ligação tem um símbolo na forma de diamante que representa “partes” ou “componentes” do agregado ou inteiro.

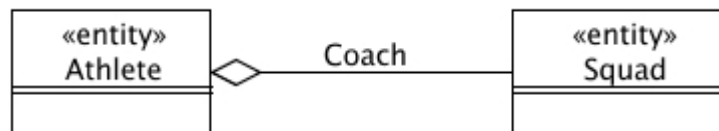


Figura 30: Agregação

Multiplicidade é o número de objetos de uma classe que podem ser associados com um objeto de outra classe. É indicado por uma expressão de texto na linha da associação. Responde às seguintes perguntas:

- A associação é opcional ou obrigatória?
- Qual é o número mínimo e máximo de instâncias que podem se conectar a uma instância de um objeto?

É possível que haja uma associação múltipla na mesma classe. Entretanto, devem representar o relacionamento entre objetos, distintos, da mesma classe. As especificações da multiplicidades são descritas na Tabela 8:

<b>Multiplicidade</b>	<b>Descrição</b>
_____	Não especificado. Utilizado quando não se conhece o número de exemplos associados há um objeto.
1	Somente um. Utilizado quando um único objeto está associado a outro.
* 0..*	Nenhum ou vários. Utilizado quando nenhum ou vários objetos estão associados com o outro. É chamada de associação opcional.
1..*	Um ou vários. Utilizado quando um ou vários objetos estão associado com outro.
0..1	Nenhum ou um. Utilizado quando nenhum ou um objeto está associado com outro. É conhecida, também, como associação imperativa.
2-4	Escala Especificada. Utilizado quando o número dos objetos tiver um tamanho específico.
2,4..6	Múltiplo, Deslocado.

Tabela 8: Especificações de Multiplicidade

Para ajudar na identificação das associações, as seguintes instruções podem ser seguidas:

1. Começar estudando as ligações nos diagramas de colaboração. Eles indicam que classes necessitam comunicar-se;
2. Ligações reflexivas não necessariamente são instâncias de associações unárias. Um objeto pode emitir uma mensagem para ele mesmo. O relacionamento unário é necessário quando dois objetos da mesma classe necessitam comunicar-se;
3. Focar somente nas associações necessárias para compreender o caso do uso;
4. Dê nome às associações, porém não use verbos ou nomes de funções;
5. Faça uma breve descrição da associação para indicar como esta é usada, ou que relacionamentos representa.

Em nosso exemplo, as associações identificadas são mostradas na Figura 31.

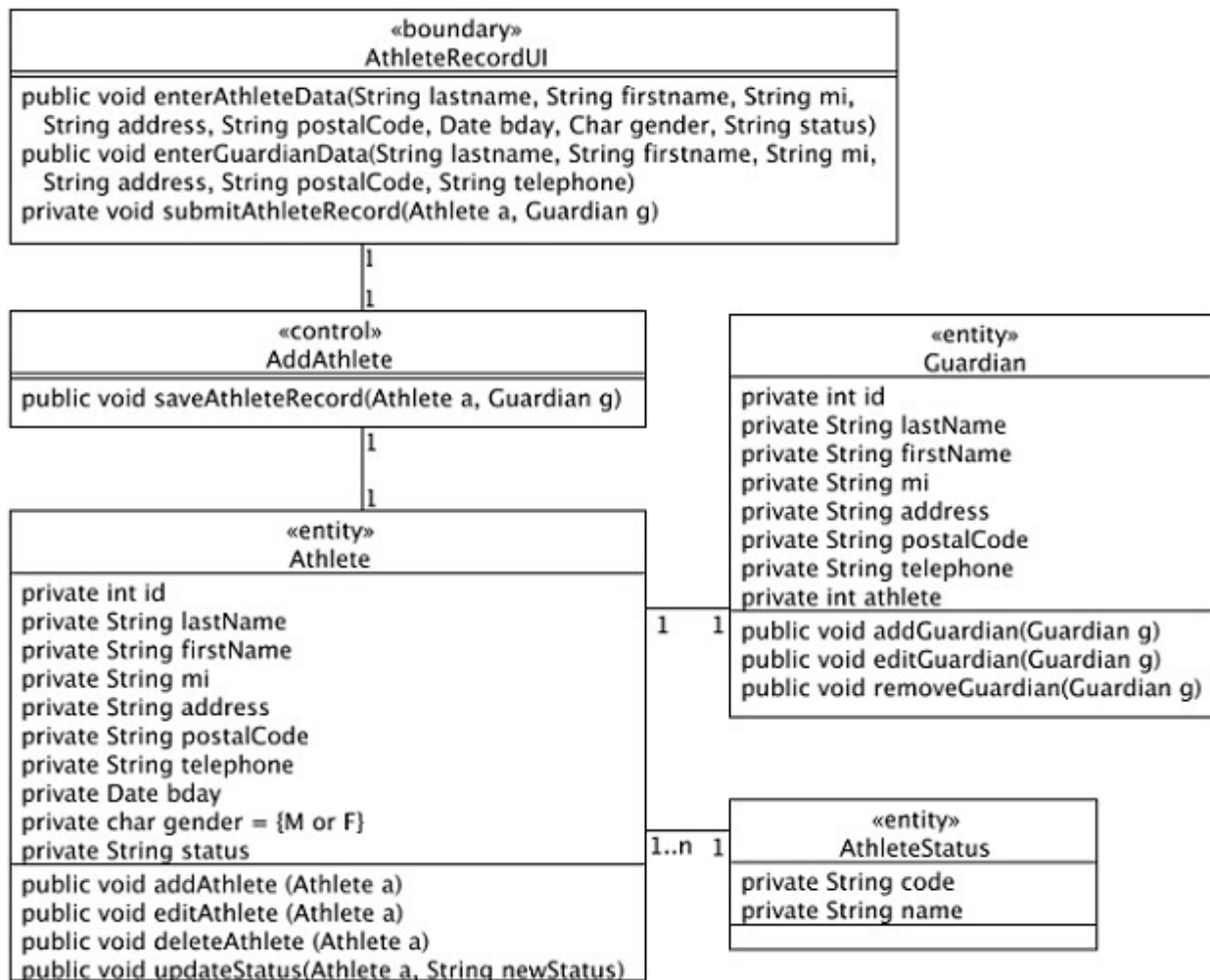


Figura 31: Identificação de Associação

Uma classe de fronteira **AthleteRecordUI** interage com apenas uma classe de controle **AddAthlete**. Uma classe de controle **AddAthlete** interage com apenas uma classe de entidade **Athlete**. Um **Athlete** tem apenas um **Guardian** e um **AthleteStatus**. Um **AthleteStatus** pode ser designado para zero ou mais **Athlete**s.

### Passo 7: Unificar classes de análise em um diagrama de classe.

Unificar classes de análise significa assegurar que cada classe de análise represente conceitos bem definidos, sem sobreposição de responsabilidades. É filtrar classes de análise para assegurar que um número mínimo de conceitos sejam criados.

Para ajudar na unificação de classes de análise em um único diagrama de classe, as seguintes linhas de direção podem ser usadas.

1. O nome da classe de análise deve capturar exatamente a função desempenhada pela classe no sistema. Deve ser único, e duas classes não devem ter o mesmo nome.
2. Uma classes que tenham um comportamento semelhante.
3. Una classes de entidade que tenham atributos similares, mesmo se seus comportamentos definidos sejam diferentes; agregue os comportamentos das classes unificadas.
4. Quando uma classe é atualizada, qualquer documentação suplementar deve ser atualizada, quando necessário. Algumas vezes, uma atualização nos requisitos originais pode ser requerida. No entanto, isso deve ser controlado, pois os requisitos são o contrato com o usuário/cliente, e quaisquer mudanças devem ser verificadas e controladas.
5. Avalie o resultado. Tenha certeza que:
  - As classes de análise atendem aos requisitos funcionais encontrados no sistema
  - As classes de análise e seus relacionamentos são consistentes com a colaboração que eles

oferecem.

Observando o diagrama de classe unificado, atributos e responsabilidades foram refinados para dar uma definição clara das classes. Lembre-se que classes de análise raramente sobrevivem depois da fase de projeto. O principal foco nesse ponto é que dados (como os representados pelos atributos) e as operações (como representado pelas responsabilidades) são representados e distribuídos pelas classes. Serão reestruturados e refinados durante a fase de projeto. Use a Lista de Validação que será discutido na seção seguinte para checar o Modelo de Análise. A Figura 32 mostra o diagrama de classe unificado de *Club Membership Maintenance*.

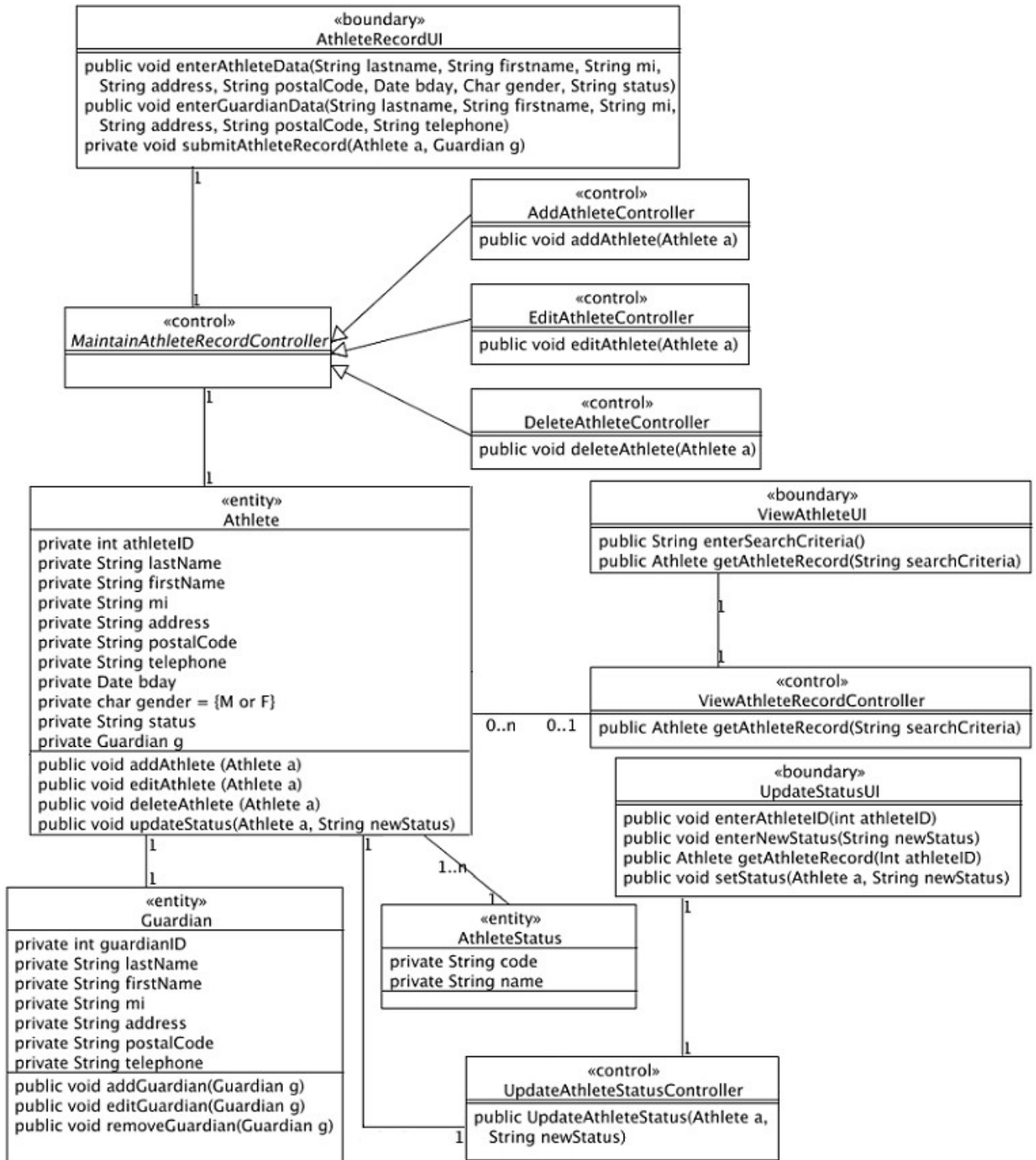


Figura 32: Diagrama de Classe do Club Membership Maintenance

### **5.3. Lista de validação do modelo de análise**

Similar ao Modelo de Requisitos, uma validação é requerida. As perguntas a seguir podem ser usadas como direcionamento.

#### **Modelo do objeto**

1. Todas as classes analisadas são classificadas, identificadas, definidas e documentadas?
2. O número de classes é razoável?
3. A classe tem um nome indicativo de seu papel?
4. As classes abstratas estão bem definidas?
5. Todos os atributos e responsabilidades estão bem definidos nas classes? São acoplados funcionalmente?
6. A classe pode ser requerida?

#### **Modelo de Comportamento**

1. Todos os cenários foram seguramente modelados? (Incluindo casos excepcionais)
2. A interação dos diagramas é clara?
3. Os relacionamentos são claros e consistentes?
4. As mensagens são bem definidas?
5. Todos os diagramas atendem às exigências?

## 6. Matriz de Rastreabilidade de Requisitos (MRR)

A **Matriz de Rastreabilidade de Requisitos (MRR)** é uma ferramenta para gerência de requisitos que pode ser usada não somente na engenharia de requisitos mas ao longo de todo processo de engenharia de software. Ela tenta tornar mais transparente ou “caixa-branca” a abordagem de desenvolvimento.

A **abordagem “caixa-branca”** se refere ao desenvolvimento dos requisitos do sistema sem considerar a implementação técnica daqueles requisitos. Da mesma forma como a UML, esta abordagem usa o conceito de perspectivas. Existem três, a saber, **conceitual, de especificação e de implementação**. Porém, em termos de requisitos, lidamos apenas com a conceitual e de especificação.

A Perspectiva Conceitual lida com os “conceitos de um domínio”. A Perspectiva de Especificação trata com as interfaces. Como um exemplo, conceitos dentro do domínio do estudo de caso são atletas, treinadores, equipes e times. Interfaces tratam com “como” os conceitos (elementos do sistema) interagem um com o outro tal como promovendo ou rebaixando um atleta.

### 6.1. Componentes da Matriz de Rastreabilidade de Requisitos

Existem muitas variações sobre os componentes que formam a MRR. Aqueles listados na Tabela 9 são os elementos iniciais recomendados da MRR.

Componente da MRR	Descrição
MRR ID	Um número identificador único para um requisito específico. Escolha um código que seja facilmente identificável.
Requisitos	Uma sentença estruturada descrevendo os requisitos no formato de “deve”, isto é, “o sistema deve...”, “o usuário deve...” ou “um item selecionado deve...”
Notas	Notas adicionais sobre o requisito.
Solicitante	Pessoa que solicitou o requisito.
Data	Data e hora que o requisito foi requisitado pelo requerente.
Prioridade	Prioridade dada a um requisito. Pode-se usar um sistema de prioridade numérico, a Quality Function Deployment, ou a técnica MoSCoW.

Tabela 9: Componentes Iniciais da MRR

Enquanto os elementos acima são os iniciais da MRR, pode-se customizar a matriz para ligar os requisitos com outros produtos do trabalho ou documentos. Esta é realmente onde a potência da MRR é refletida. Pode-se adicionar outros elementos. Um exemplo é mostrado na Tabela 10.

Componente Adicional da MRR	Descrição
MRR ID Relacionado	Especificar se o corrente MRR ID é direta ou indiretamente relacionado a outro requisito.
Especificação Trabalho (ET)	Descrever o que relaciona os requisitos diretamente a um parágrafo específico dentro da Especificação de Trabalho (ET). Uma ET é um documento que especifica a natureza do engajamento da equipe de desenvolvimento com seus clientes. Ela lista os produtos do trabalho e o critério que definirá a qualidade dos produtos do trabalho. Ela ajuda a guiar a equipe de desenvolvimento na sua abordagem para entrega dos requisitos.

Tabela 10: Componentes Adicionais da MRR

Os componentes e o uso da MRR devem ser adaptados para as necessidades do processo, do projeto e do produto. Eles crescem a medida que o projeto se desenvolve. Como engenheiros de

software, pode-se utilizar todos os componentes, remover componentes ou adicionar componentes quando houver necessidade.

Considere um exemplo da MRR para o desenvolvimento do sistema para a Liga *Ang Bulilit: Club Membership Maintenance* conforme mostrado na Tabela 11. A organização é centrada em caso de uso.

<b>MRR ID</b>	<b>Requisito</b>	<b>Notas</b>	<b>Solicitante</b>	<b>Data</b>	<b>Prioridade</b>
Caso de Uso 1.0	O sistema deve ser capaz de manter informação pessoal de um atleta.		RJCS	12/06/05	Deve ter
Caso de Uso 2.0	O sistema deve ser capaz de permitir ao treinador mudar o estado de um atleta.		RJCS	12/06/05	Deve ter
Caso de Uso 3.0	O sistema deve ser capaz de ver a informação pessoal do atleta.		RJCS	12/06/05	Deve ter
Caso de Uso 1.1	O sistema deve ser capaz de adicionar um registro do atleta.		RJCS	13/06/05	Deve ter
Caso de Uso 1.2	O sistema deve ser capaz de editar um registro do atleta.		RJCS	13/06/05	Deve ter
Caso de Uso 1.3	O sistema deve ser capaz de excluir um registro do atleta.		RJCS	13/06/05	Deve ter

*Tabela 11: Exemplo Inicial da MRR (RTM - sigla em inglês) para a Manutenção dos Sócios do Clube*



## 7. Métrica de Requisitos

Medir requisitos focaliza-se geralmente em três áreas: processo, produto e recursos. Os requisitos trabalham produtos que podem ser avaliados, olhando-se primeiro o número de requisitos. À medida que cresce o número, obtemos uma indicação de como o projeto de software é grande. O tamanho do requisito pode ser uma boa entrada para estimar o esforço de desenvolvimento de programa. Também, como os progressos de desenvolvimento de software, o tamanho de requisito pode ser capturado. Com o andamento do projeto e desenvolvimento, temos uma compreensão mais profunda do problema e a solução que poderia levar a descoberta dos requisitos que não são aparentes durante o processo de Engenharia de Requisitos.

Similarmente, pode-se medir o número de vezes que os requisitos mudaram. Um grande número de mudanças indica um pouco de instabilidade e incerteza em nossa compreensão do que o sistema deveria fazer ou como deveria se comportar. Em tal caso, os requisitos deveriam ser completamente compreendidos e, possivelmente, repetir a fase de engenharia de requisitos.

Os requisitos podem ser usados pelos projetistas e testadores. Podem ser usados para determinar se eles estão prontos para lhes ser entregues. O **Teste de Perfil de Requisitos** é uma técnica empregada para determinar a prontidão em entregar os requisitos aos projetistas ou testadores<sup>4</sup>.

Os projetistas são solicitados a avaliar cada requisito em uma escala de 1 a 5, baseado em um sistema como especificado em Tabela 12.

<b>Taxa do Projeto de Sistema</b>	<b>Descrição</b>
1	Significa que o projeto corresponde completamente aos requisitos, e requisitos similares foram solicitados em projetos anteriores. Logo, não terá problemas com este projeto.
2	Significa que existem aspectos novos de requisitos ao projeto; entretanto, não são radicalmente diferentes de requisitos de projetos concluídos com sucesso em projetos anteriores.
3	Significa que há aspectos de requisitos que são muito diferentes dos requisitos projetados anteriormente; entretanto, compreende-se e são confiáveis para desenvolver um bom projeto.
4	Significa que há partes de requisitos que não são compreendidas; não são confiáveis para o desenvolvimento de um bom projeto.
5	Significa que não é possível compreender estes requisitos e não se pode desenvolver um bom projeto com eles.

Tabela 12: Descrição da escala dos projetistas de sistema

Os testadores são solicitados a avaliar cada requisito em uma escala de 1 a 5 baseado em um sistema como especificado na Tabela 13.

<b>Taxa de teste do sistema</b>	<b>Descrição</b>
1	Significa que os requisitos foram compreendidos completamente, e que já foram testados requisitos similares em projetos anteriores; é provável que não existirão problemas em testar o código.
2	Significa que existem alguns aspectos dos requisitos são novos ao projeto; entretanto, não são radicalmente diferentes dos requisitos testados com sucesso em projetos anteriores.
3	Significa que alguns aspectos dos requisitos são muito diferentes dos requisitos testados anteriormente; entretanto, compreende-se e é confiável testá-los.

<sup>4</sup> Pfleeger, *Software Engineering Theory and Practice*, pp. 178-179

<b><i>Taxa de teste do sistema</i></b>	<b><i>Descrição</i></b>
4	Significa que existe alguns aspectos dos requisitos que não são compreensíveis; não é confiável que se planeje um teste para este requisito.
5	Significa que este requisito não é compreensível e não pode ser desenvolvido.

*Tabela 13: Descrição da escala de verificação do sistema*

Se o resultado do perfil de requisitos resultar em 1 ou 2, os requisitos podem ser passados aos projetistas e testadores. Se não, os requisitos necessitam ser reescritos. Será necessário retornar à fase de Engenharia de Requisitos.

A avaliação é subjetiva. Entretanto, as contagens podem oferecer informações úteis que irão incentivar a qualidade dos requisitos antes que o projeto prossiga.

## 8. Exercícios

### 8.1. Criar o Modelo de Exigências

1. Criar um Modelo de Exigências do Sistema de Informação Técnica
  - Desenvolver os Diagramas de Caso de Uso
  - Para cada Caso de Uso, desenvolver a Especificação do Caso de Uso
  - Criar o Glossário
2. Criar um Modelo de Exigências do Sistema de Manutenção *Squad & Team*
  - Desenvolver os Diagramas de Caso de Uso
  - Para cada caso de uso, desenvolver a Especificação do Caso de Uso
  - Criar o Glossário

### 8.2. Criar o Modelo de Análise

1. Criar um Modelo de Análise do Sistema de Informação Técnica
  - Desenvolver os Diagramas de Seqüência de todo o cenário definido nas especificações de Caso de Uso
  - Criar o Diagrama de Colaboração para cada Diagrama de Seqüência
  - Criar o Diagrama de Classes das classes analisadas
2. Criar o modelo de análise do Sistema de Manutenção *Squad & Team*
  - Desenvolver o Diagrama de Seqüência de todo o cenário definido na especificação de Caso de Uso
  - Criar os Diagramas de Colaboração para cada Diagrama de Seqüência
  - Criar o Diagrama de Classes para as classes analisadas

### 8.3. Atribuição do Projeto

O objetivo da atribuição do projeto é reforçar o conhecimento e habilidades adquiridas neste capítulo. Particularmente, são estas:

1. Desenvolver o modelo de exigências
2. Desenvolver o modelo de análise
3. Desenvolver a matriz de rastreabilidade de requisitos

### PRINCIPAIS TRABALHOS PRODUZIDOS:

1. O Modelo de Requisitos
  - Diagrama de Caso de Uso
  - Especificação de Caso de Uso
2. O Modelo de Análise
  - Diagrama de Classes
  - Diagrama de Seqüência
  - Diagrama de Colaboração
3. Lista de Ações
4. Matriz de Rastreabilidade de Requisitos

## Parceiros que tornaram JEDI™ possível



### **Instituto CTS**

Patrocinador do DFJUG.

### **Sun Microsystems**

Fornecimento de servidor de dados para o armazenamento dos vídeo-aulas.

### **Java Research and Development Center da Universidade das Filipinas**

Criador da Iniciativa JEDI™.

### **DFJUG**

Detentor dos direitos do JEDI™ nos países de língua portuguesa.

### **Banco do Brasil**

Disponibilização de seus *telecentros* para abrigar e difundir a Iniciativa JEDI™.

### **Politec**

Suporte e apoio financeiro e logístico a todo o processo.

### **Borland**

Apoio internacional para que possamos alcançar os outros países de língua portuguesa.

### **Instituto Gaudium/CNBB**

Fornecimento da sua infra-estrutura de hardware de seus servidores para que os milhares de alunos possam acessar o material do curso simultaneamente.