6
Linguagem de Modelagem UTP-C

Neste capítulo apresentamos a linguagem de modelagem UTP-C (UML Testing Profile for Coordination) criada a partir do modelo conceitual apresentado no capítulo 5 e que foca em modelar informações úteis para a coordenação dos testes de software. A UTP-C é uma extensão da UML Testing Profile (UTP), perfil padrão de teste da OMG para a UML, que foca no diagrama de classes estático e no diagrama de atividades de interação. Esses dois diagramas foram escolhidos porque permitem a incorporação dos aspectos estruturais e dinâmicos das informações úteis para a coordenação dos testes apresentadas no modelo conceitual proposto.

Este capítulo está organizado da seguinte maneira. Na Seção 6.1 são apresentados os mecanismos adotados para realizar a extensão na UML e na UTP original. Na Seção 6.2 é apresentado em detalhe o metamodelo da UTP-C acompanhado com simples exemplos. Na Seção 6.3 é apresentado como foi feito o mapeamento das informações consideradas no modelo conceitual proposto em relação à UTP-C. Por fim, na Seção 6.4 são apresentadas duas avaliações empíricas da linguagem UTP-C envolvendo pessoas com diferentes níveis de conhecimento na área de modelos e testes de software.

6.1.
Os Mecanismos de Extensão da UML e UTP

Visando modelar as informações apresentadas no modelo conceitual do capitulo 5, o metamodelo UML foi estendido. Como nosso objetivo é produzir uma extensão conservativa da UML (Turski et al., 1987), as metaclasses definidas em UML não foram modificadas durante a extensão. O mecanismo de extensão adotado foi à criação de um perfil que foca na definição de novas restrições e estereótipos na UML. Como na literatura há o perfil UML Testing Profile, que já oferece a modelagem de informações úteis sobre testes de software, decidimos estendê-lo.

Um estereótipo é o elemento de um modelo que define valores adicionais, outras restrições e, opcionalmente, uma nova representação gráfica. Todos os

elementos marcados (classificados) por um ou mais estereótipos recebem os valores e as restrições definidas pelo estereótipo, além dos atributos, associações e superclasses que o elemento tem em UML padrão. Os estereótipos oferecem uma forma de definir subclasses virtuais de metaclasses de UML com novos metaatributos e outra semântica. Portanto, novas restrições, estereótipos e propriedades foram adicionados na UTP original para formar o perfil UTP-C.

6.2. Metamodelo da UML Testing Profile for Coordination (UTP-C)

Na Figura 12 são ilustrados, exclusivamente, os estereótipos definidos pela UTP-C para os elementos da UML. Alguns estereótipos apresentados são totalmente novos, enquanto que outros já são oferecidos pela UTP original. No entanto, esses estereótipos tiveram a adição de restrições e/ou propriedades para representar informações providas no modelo conceitual proposto no capítulo 5.

A metaclasse *Element* é a superclasse da metaclasse *Classifier* (ver Figura 12), que é a superclasse, por exemplo, da metaclasse *Class* (usado em diagramas de classe) e *Activity* (usado em diagramas de atividades). Assim, os estereótipos *TestContext*, *OrderedSuite* e *TestCriterion* podem ser usados em qualquer submetaclasse de *Classifier*, enquanto que o estereótipo *TestCase* está relacionado à metaclasse *Behaviour* para permitir a modelagem em entidades comportamentais, como, por exemplo, *Activity*, assim como apresentado pela UTP (UTP, 2012).

Consideramos que o estereótipo *TestContext* representa a mesma ideia da entidade *Test* apresentado no modelo conceitual do capítulo 5. Adotamos essa sintaxe para manter conformidade à linguagem de modelagem UTP. O restante dos estereótipos é equivalente aos nomes usados pelas entidades propostas pelo modelo conceitual.

A Figura 12 também ilustra que a metaclasse *Classifier* está relacionada às metaclasses *StructuralFeature* e *BehavioralFeature*. Uma característica estrutural é uma característica de um classificador que especifica a estrutura de instâncias do mesmo. Uma característica comportamental é uma característica de um classificador que especifica um aspecto de comportamento de suas instâncias. Assim, a metaclasse *StructuralFeatures* é uma generalização da metaclasse *Property* (atributos de uma classe são representados como

instâncias de *Property*), e a metaclasse *BehavioralFeatures* é uma generalização das metaclasses *Operation*, de acordo com a definição de UML (UML, 2012). A UTP original considera que um caso de teste também pode ser expresso como uma operação. Dessa forma, o estereótipo *TestCase* é aplicável à metaclasse *Operation*.

A metaclasse *Comment* é subclasse da metaclasse *Element*, podendo receber o estereótipo *ArtifactUnderTest* ou FreeFormatComment. O primeiro estereótipo permite expressar informações referentes a algum artefato em teste do SUT a partir de uma estrutura pré-definida pela UTP-C. Já o segundo estereótipo permite que qualquer comentário livre possa ser criado sem que haja a necessidade de seguir algum formato. Dessa forma, podemos relacionar qualquer entidade modelada da UML com algum desses comentários.

Já os estereótipos *TestClassification* e *Development* são usados em pacotes para permitir o grupamento de test contexts e suites a partir de alguma classificação de teste (ver subseção 6.2.5) e por algum pacote de desenvolvimento (ver subseção 6.2.6), respectivamente.

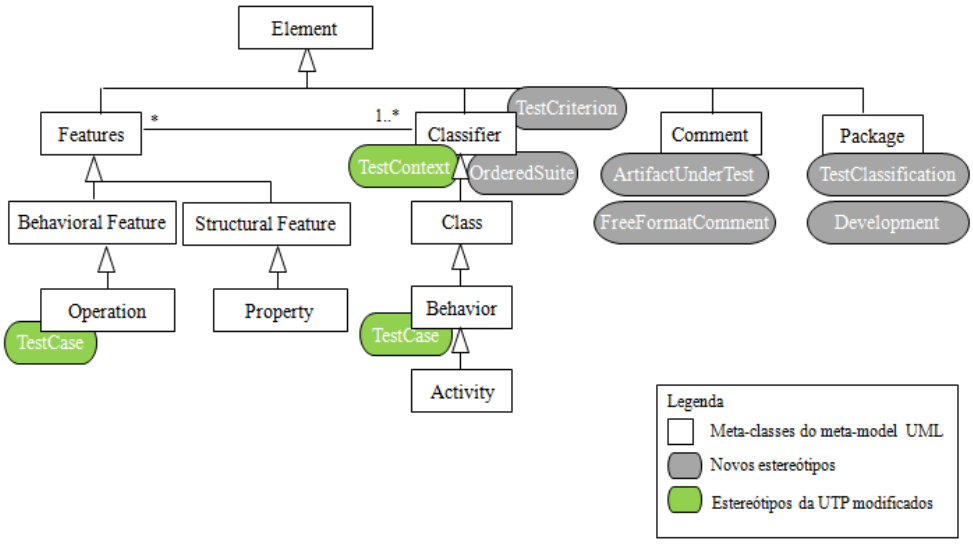

Figura 12. Metamodelo da UML com estereótipos da UTP-C.

Outra informação de extrema importância para a gerência dos testes é entender a dependência que existe entre eles. Para expressar a semântica das dependências, na Figura 13 são apresentados estereótipos que podem ser usados em relacionamentos de dependência na UML. Tais estereótipos são

baseados nos tipos de dependência apresentados no modelo conceitual proposto no capítulo 5.

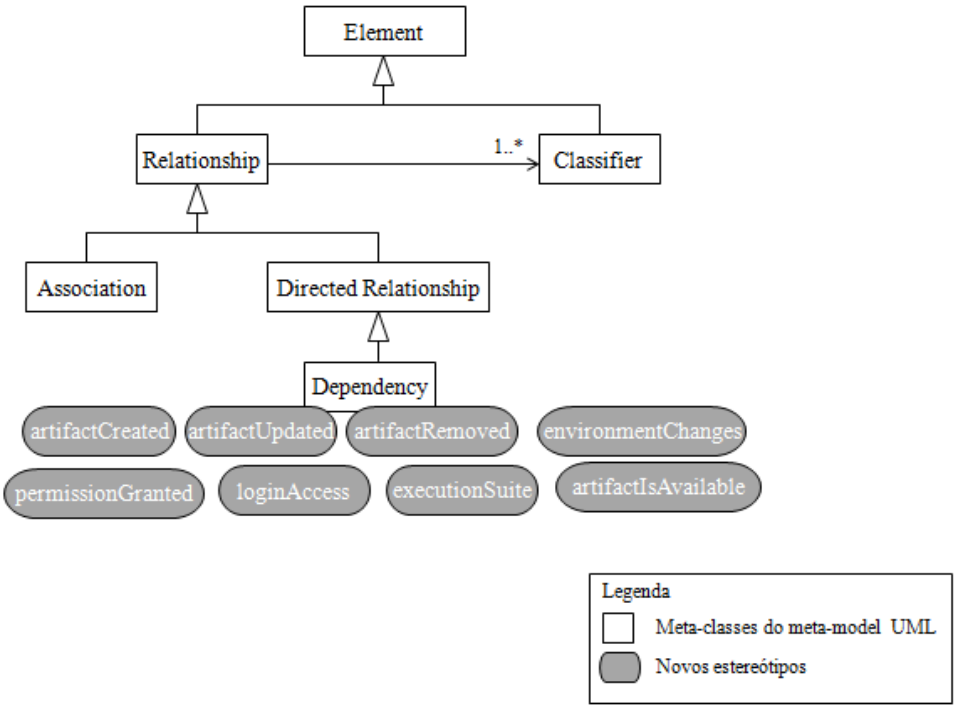

Figura 13. Estereótipos que expressam semântica de dependências.

Nas próximas subseções são apresentadas em detalhe as restrições e propriedades definidas em cada estereótipo usado pela UTP-C. Para os estereótipos já propostos pela UTP original (*TestContext* e *TestCase*) são apresentados, exclusivamente, as informações incluídas nas entidades modeladas que possuem tais estereótipos. Para ajudar no entendimento da UTP-C, a seguinte estrutura é adotada para explicação: (i) descrição do estereótipo, (ii) notação a ser usada, (iii) restrições, e (iv) um simples exemplo ilustrando como representá-lo em um modelo. Tal estrutura é baseada em documentos oferecidos pela OMG que apresentam perfis oferecidos para a UML, como, por exemplo, (UTP, 2012).

6.2.1.
Suite de Teste

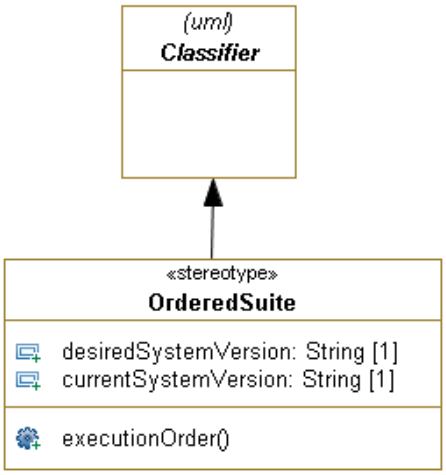

Figura 14. OrderedSuite na UTP-C.

Descrição: OrderedSuite é um classificador (UML, 2012) responsável por executar um conjunto de casos de testes, desenvolvidos em test contexts em uma ordem específica (ver Figura 14).

Notação: A notação para OrderedSuite é representada pelo uso do estereótipo <<OrderedSuite>>.

Restrição:

1. Um OrderedSuíte tem que ter exatamente um atributo que descreva qual versão do SUT ele deve estar atualizado (representado pelo atributo *desiredSystemVersion*).
2. Um OrderedSuite tem que ter exatamente um atributo que descreva qual versão corrente do SUT ele está atualizado (representado pelo atributo *currentSystemVersion*).
3. Um *OrderedSuíte* tem que ter uma ordem de execução dos testes definida no método "*executionOrder()*".

Exemplo: Na Figura 15 é ilustrado um exemplo de OrderedSuite, que deve estar atualizado para a versão "3.0" do SUT, mas está correntemente atualizado para a versão "2.0".

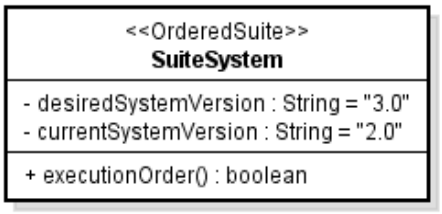

Figura 15. Exemplo de modelagem usando <<OrderedSuite>>.

6.2.2. Contexto de Teste (TestContext)

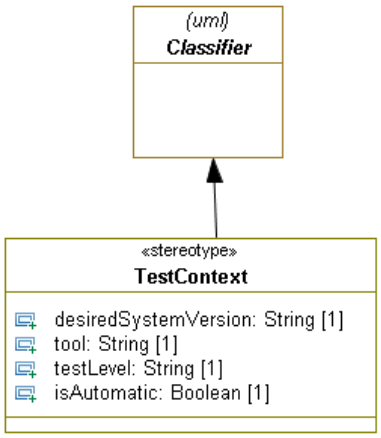

Figura 16. TestContext na UTP-C.

Descrição: Test context é um classificador (ver Figura 16) responsável por agrupar um conjunto de casos de teste. Um test context possui as seguintes informações: a versão do SUT que ele deve estar atualizado, a ferramenta de teste que deve ser usada para executá-lo, seu nível de teste, e se ele é executado de forma automática ou manual. Caso algum dos casos de teste de um text context precise recuperar informações de um banco de dados (como, por exemplo, casos de teste criados em DBUnit) (DBUnit, 2012), algum arquivo de propriedades descrevendo o login, senha, driver de comunicação e URL deve ser provido para que a conexão com o banco seja realizada. Caso esse arquivo de propriedades não seja utilizado, tais informações devem estar presentes em algum ponto do código do projeto de teste.

Notação: A notação para test context é o uso do estereótipo <<TestContext>>.

Restrição:

1. Um test context tem que ter exatamente um atributo que descreva qual a versão do sistema (system under test - SUT) que ele deve estar atualizado (representado pelo atributo *desiredSystemVersion*).
2. Um test context tem que ter exatamente um atributo que descreva a ferramenta usada para executá-lo (uso do atributo *tool*).
3. Um test context tem que ter exatamente um atributo que descreva seu nível de teste (uso do atributo testLevel).
4. Um test context tem que ter exatamente um atributo que descreva se ele é executado de forma automática ou manual (uso do atributo isAutomatic).
5. Um test context deve ser composto por um ou mais casos de teste.

Exemplos:

O estereótipo test context pode ser usado tanto em classes (diagramas de classe estáticos) como em atividades (diagramas de atividade dinâmicos). Para ilustrar essas duas situações, dois exemplos são apresentados a seguir.

Exemplo1*:* Na Figura 17, é apresentado um test context CreateUser com dois casos de teste chamados de create1 e create2. Além disso, é informado no modelo que o test context deve estar atualizado para a versão “3.0” do SUT, a ferramenta a ser usada para executá-lo é o JUnit, e ele é um teste unitário e automatizado.

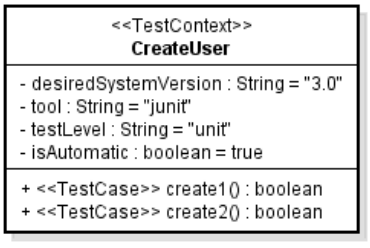

Figura 17. Exemplo de Test Context.

Exemplo 2: Já na Figura 18 é apresentado um diagrama de atividades que informa a ordem de execução de dois test contexts definidos em um suíte chamado SuiteTestAccessSystem.

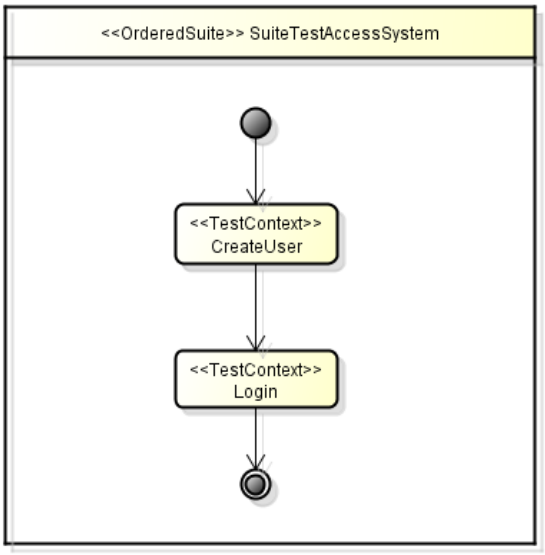

Figura 18. Exemplo de diagrama de atividades com test contexts.

6.2.3. Caso de Teste

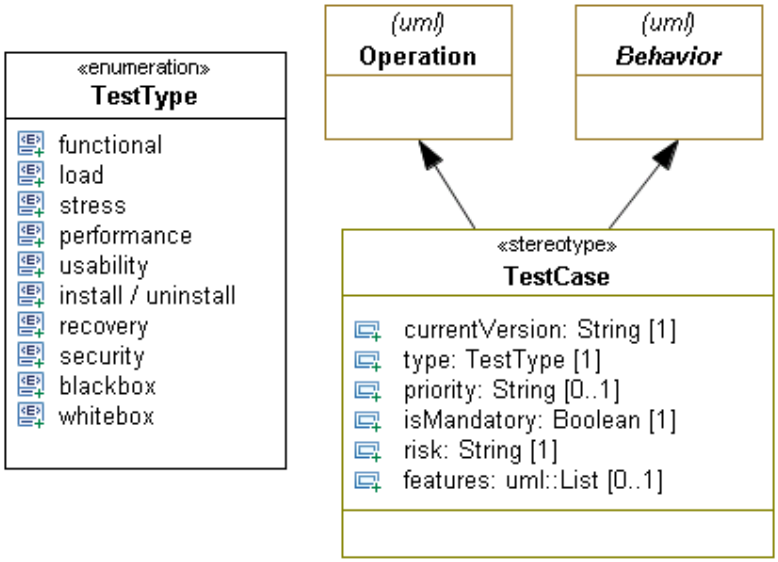

Figura 19. Caso de Teste na UTP-C.

Descrição: O estereótipo TestCase é responsável por representar um caso de teste que valida algum comportamento do SUT. Assim, um caso de teste é uma *feature* comportamental ou é responsável por especificar algum teste de um dado comportamento, assim como mencionado pela UTP original (UTP, 2012). Um caso de teste é composto pelas seguintes informações: versão corrente do SUT, tipo de teste, prioridade de execução, se é obrigatório ou opcional, e o risco do produto que o teste está relacionado caso alguma falha seja identificada a partir da sua execução. Vale ressaltar que um teste somente estará atualizado para uma versão desejada do SUT (ver subseção 6.2.2), quando todos os seus casos de teste estiverem com suas versões correntes iguais à versão desejada pelo teste.

Notação: A notação para modelar um caso de teste é a partir do estereótipo <<TestCase>>.

Restrição:

1. Um caso de teste tem que ter exatamente um atributo que descreva qual versão do SUT ele está correntemente atualizado (uso do atributo *currentVersion*).
2. Um caso de teste tem que ter exatamente um atributo que descreva seu tipo de teste (uso do atributo *type*).
3. Um caso de teste tem que ter exatamente um atributo que descreva sua prioridade de execução em detalhe (uso do atributo *priority*).
4. Um caso de teste tem que ter exatamente um atributo que descreva se ele é considerado obrigatório (uso do atributo *isMandatory*) para execução.
5. Um caso de teste tem que ter exatamente um atributo que descreva o risco do produto relacionado (uso do atributo *risk*).
6. Se um caso de teste estiver relacionado a um test context que esteja no nível de teste de sistema ou aceitação, uma ou mais features, a serem testadas, devem ser informadas a partir do atributo *features*.

Exemplos:

Assim como o test context, casos de teste podem ser modelados em diagramas de classes e diagramas de atividades. Para ilustrar essas situações, dois exemplos são apresentados a seguir.

Exemplo 1: Na Figura 20 é ilustrada uma classe test context modelada com um caso de teste chamado “communication”, atualizado para a versão “3.0” do SUT (uso do atributo *currentVersion)*. Além disso, o exemplo mostra que o caso de teste é unitário e obrigatório, além de possuir prioridade alta e risco de produto médio.

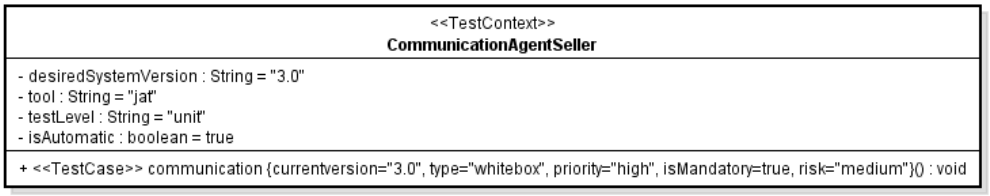

Figura 20. Exemplo de um caso de teste de unidade modelado.

Exemplo 2: Na Figura 21 é ilustrada uma classe test context modelada com o nome “CreatingUser”, responsável por testar o cadastro de usuários em um sistema e que deve estar atualizado para a versão “4.0” do SUT (uso do atributo *desiredSystemVersion*), já que seus casos de teste (registerOpt1 e registerOpt2) estão atualizados para a versão “3.0” (uso do atributo *currentVersion*). Adicionalmente, o exemplo mostra que os casos de teste são manuais e obrigatórios, estão relalionados ao nível de teste de aceitação, possui prioridade e risco de produto médio, e estão relacionados à feature “Register Users” (uso do atributo features).

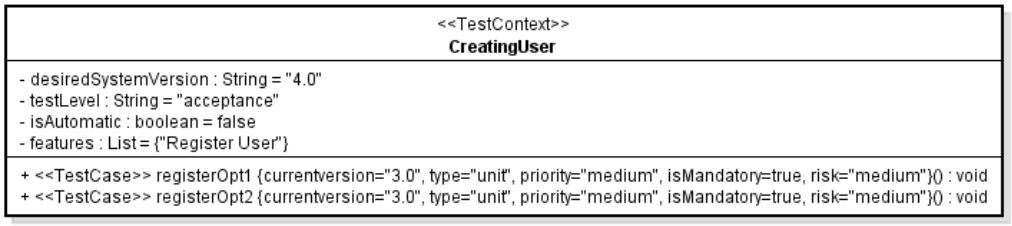

Figura 21. Exemplo de casos de teste de aceitação modelados.

Exemplo 3: Na Figura 22 é apresentado um diagrama de atividade que descreve o fluxo de execução de dois casos de teste (communication e verifyingBelief) desenvolvidos em diferentes test contexts (CommunicationAgentSeller e TestBelief). Geralmente essa modelagem é usada quando há a necessidade de definir a ordem de execução de casos de teste especiais, que sejam responsáveis por validar algum artefato específico. Perceba que para realizar essa modelagem, cada atividade possui o estereótipo <<TestCase>> e seu nome adota a estrutura a seguir.

"<Nome_Test_Context>.<Nome_Caso_de_Teste>".

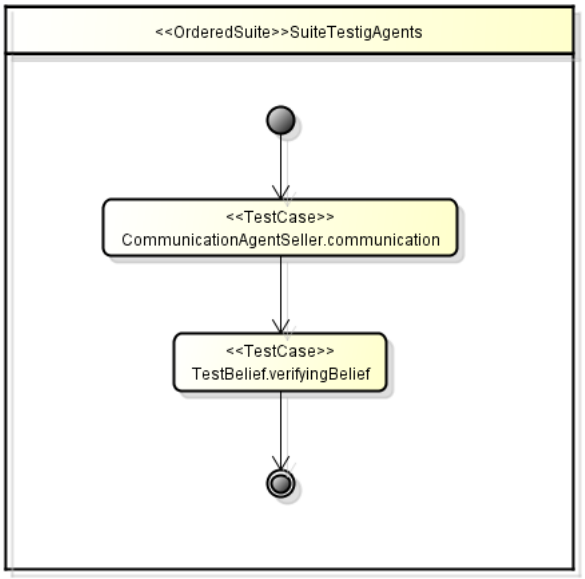

Figura 22. Exemplo de diagrama de atividades com casos de teste.

6.2.4. Critério de Seleção de Teste

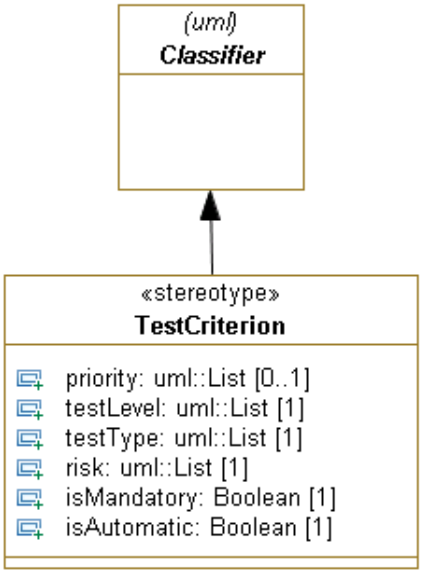

Figura 23. Critério de seleção na UTP-C.

Descrição: TestCriterion é um classificador responsável por representar critérios de seleção para a execução de testes, assim como ilustrado na Figura 23. Para permitir a modelagem de diferentes critérios, a UTP-C define um conjunto de informações para o TestCriterion (ver Figura 23): (i) prioridade de execução, (ii) nível de teste, (iii) tipo de teste, (iv) risco do produto, (v) tipo de obrigatoriedade de execução (obrigatório ou opcional), e (vi) tipo de automação dos testes (automático ou manual). Como todas essas informações estão relacionadas aos testes modelados a partir da UTP-C, torna-se imediata a identificação de quais testes atendem algum critério.

Notação: Uso do estereótipo <<TestCriterion>>.

Restrição:

1. Um critério tem que ter exatamente um atributo que descreva o(s) nível(is) de teste a serem executados (uso do atributo *testLevel*).
2. Um critério tem que ter exatamente um atributo que descreva o(s) tipo(s) de teste a serem executados (uso do atributo *testType*).

3. Um critério tem que ter um ou mais riscos de produto que descrevam os possíveis riscos relacionados aos testes (representado pelo atributo *risk*).
4. Um critério tem que ter exatamente um atributo que descreva se deverão ser selecionados testes considerados obrigatórios para execução (uso do atributo *isMandatory*).
5. Um critério tem que ter exatamente um atributo que descreva se deverão ser selecionados testes considerados automáticos ou manuais para execução (uso do atributo *isAutomatic*).

Exemplo: Na Figura 24 é ilustrado um critério de seleção que informa quais testes são válidos para execução. Nesse exemplo, os testes válidos são aqueles com prioridade alta, relacionado ao nível de teste unitário ou de sistema, ser um teste de regressão, esteja relacionado a qualquer risco de produto (devido o valor nulo informado), sejam obrigatórios e que sejam executados de forma automática ou manual (tanto faz um ou outro devido o valor nulo informado).

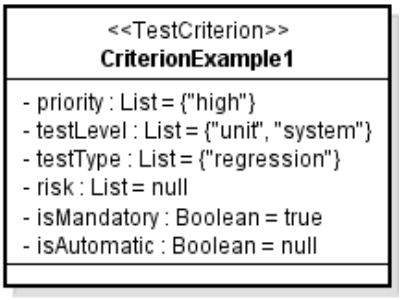

Figura 24. Exemplo de modelagem de um critério de seleção da UTP-C.

6.2.5. Classificação de Teste

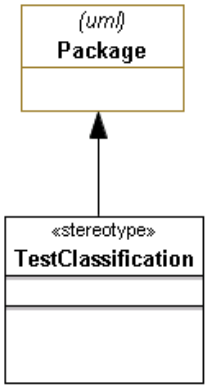

Figura 25. Classificação de Teste na UTP-C.

Descrição: O estereótipo TestClassification representa um pacote (ver Figura 25) que permite definir classificações de teste para que seja possível visualizar e agrupar test contexts ou suítes que estejam relacionados a uma mesma classificação. Exemplos de classificações podem ser baseados em tipos de teste, níveis de teste, assim como qualquer outra informação que permita agrupar conceitualmente testes ou suites modelados.

Notação: Representado pelo estereótipo <<TestClassification>> em algum pacote.

Exemplo: Na Figura 26 é ilustrado um exemplo com três classificações de teste: regressão, atualizados e novos. Testes de regressão são aqueles que devem ser sempre executados independentemente da versão do SUT. Testes atualizados são aqueles modificados a partir dos novos requisitos definidos no SUT. Já testes novos são aqueles criados para validar os novos requisitos definidos em uma release de um SUT. Tanto essas classificações como os testes relacionados a cada classificação podem ser definidos na etapa de planejamento dos testes. No entanto, como mudanças podem acontecer durante a criação dos testes, os diagramas UTP-C criados podem ser alterados. Apesar disso, o ideal é que poucas mudanças sejam feitas nas classificações de teste para que o planejamento realizado não seja muito impactado. Com as

classificações modeladas e os testes vinculados a alguma classificação, os envolvidos do projeto (testadores, gerentes, desenvolvedores, etc.) podem visualizar de forma imediata relações conceituais entre testes, como, por exemplo, quais deles são os testes de regressão.

Uma situação comum presente em projetos de software é quando duas releases de um mesmo SUT estão sendo testadas e modeladas em paralelo. Considerando esse cenário, alguém da equipe deve verificar de tempos em tempos se a modelagem de uma release impacta na outra. Testes classificados em uma release mais antiga, por exemplo, podem mudar de classificação em outra release, como, por exemplo, os testes novos. Segundo (SVN, 2012a), essa verificação de impacto entre releases deve ser feita em curtos intervalos de tempo para evitar grandes custos de trabalho no projeto.

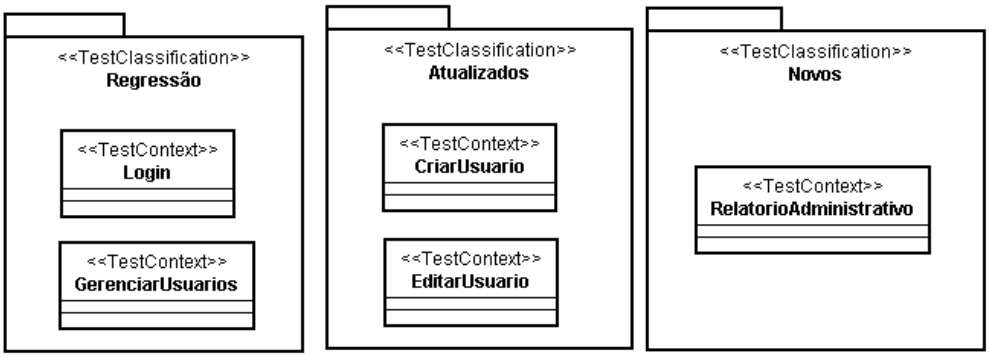

Figura 26. Modelagem de classificações de teste.

6.2.6. Pacote de Desenvolvimento

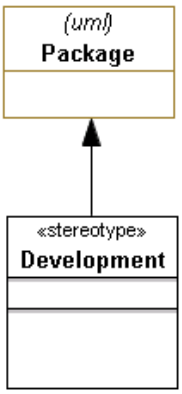

Figura 27. Pacote de Desenvolvimento na UTP-C.

Descrição: O estereótipo Development representa um pacote que armazena em algum projeto, os testes ou suites criados (ver Figura 27).

Notação: Representado pelo estereótipo <<Development>>.

Exemplo: Na Figura 28 são ilustrados os test contexts *Signout* e *GerenciarUsuarios*, que estão armazenados no pacote test.regression (\\test\regression) de um projeto de teste.

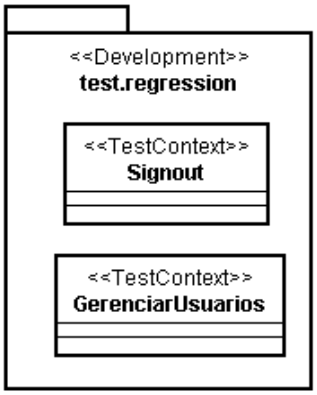

Figura 28. Modelagem usando o estereótipo <<Development>>.

6.2.7. Artefato em Teste

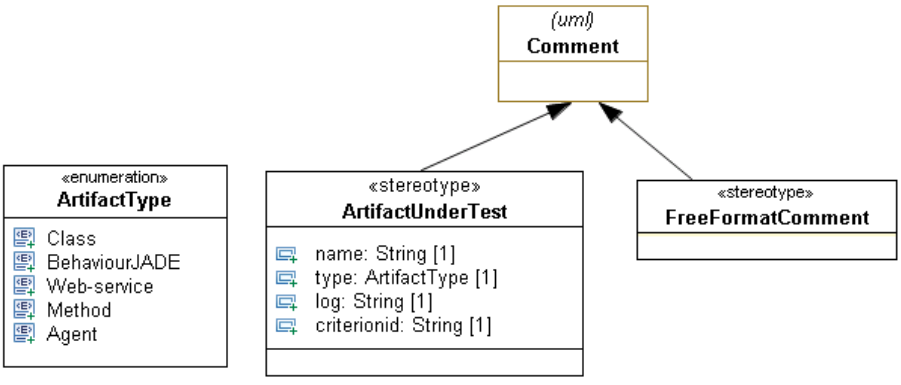

Figura 29. Estereótipo relacionado a artefatos de um sistema.

Descrição: O estereótipo ArtifactUnderTest, ilustrado na Figura 29, deve ser aplicado em entidades do tipo comentário para expressar que as informações contidas em algum comentário estão relacionadas a algum artefato em teste. Um artefato em teste é composto pelas seguintes informações: nome do artefato, tipo de artefato, log que armazena os resultados dos testes executados, e critério que descreve a política de seleção para executar testes responsáveis por validá-lo. Para permitir uma fácil recuperação dessas informações modeladas, o comentários com o estereótipo ArtifactUnderTest devem seguir uma estrutura proposta de preenchimento. No entanto, caso o designer deseje criar comentários livres, sem seguir qualquer formato, o estereótipo FreeFormatComment deve ser usado. Assim, torna-se possível diferenciar comentários livres dos comentários estruturados voltados para a descrição de artefatos em teste.

Notação: Representado pelo estereótipo <<ArtifactUnderTest>> em uma entidade comentário.

Restrição:

1. Um artefato em teste tem que ter um nome (representado pelo atributo name).
2. Um artefato em teste tem que ter um tipo (representado pelo atributo *type*). Na Figura 29 são oferecidos algumas opções a partir do tipo enumerado ArtifactType (ver Figura 29): (i) classe (Class), (ii) comportamento JADE (BehaviourJADE), (iii) serviço web (Web-service), (iv) método (Method), e (v) agente (Agent). Novas opções podem ser definidas dependendo da necessidade do designer.
3. Um artefato em teste tem que ter um log que armazene os resultados dos testes responsáveis por validá-lo (representado pelo atributo *log*).
4. Quando um artefato em teste tiver relação com um critério de seleção de teste, tal critério deve ser informado (uso do atributo *criterionid*).

Exemplo: Na Figura 30 é ilustrado um exemplo de comentário com o estereótipo <<ArtifactUnderTest>> e outro com o estereótipo FreeFormatComment, criados em um diagrama de atividade responsável

por descrever o fluxo de execução de testes responsáveis por validar tal artefato. Perceba que o artefato a ser validado pelo suíte modelado possui o nome SellerAgent, é do tipo “Agent”, os resultados dos testes são armazenados no log \\logs\logSellerAgent.txt, e seu critério de seleção é chamado de CriterionExample1. Já o comentário com o estereótipo FreeFormatComment informa que a suíte modelada descreve o fluxo de execução para validar um agente vendedor do sistema em teste.

Parte ou todas as informações contidas nas entidades que possuam o estereótipo <<ArtifactUnderTest>> podem estar espalhadas em outros diagramas modelados. Portanto, ferramentas que permitam manter a consistência dessas informações representadas em diferentes diagramas podem ser desenvolvidas e utilizadas.

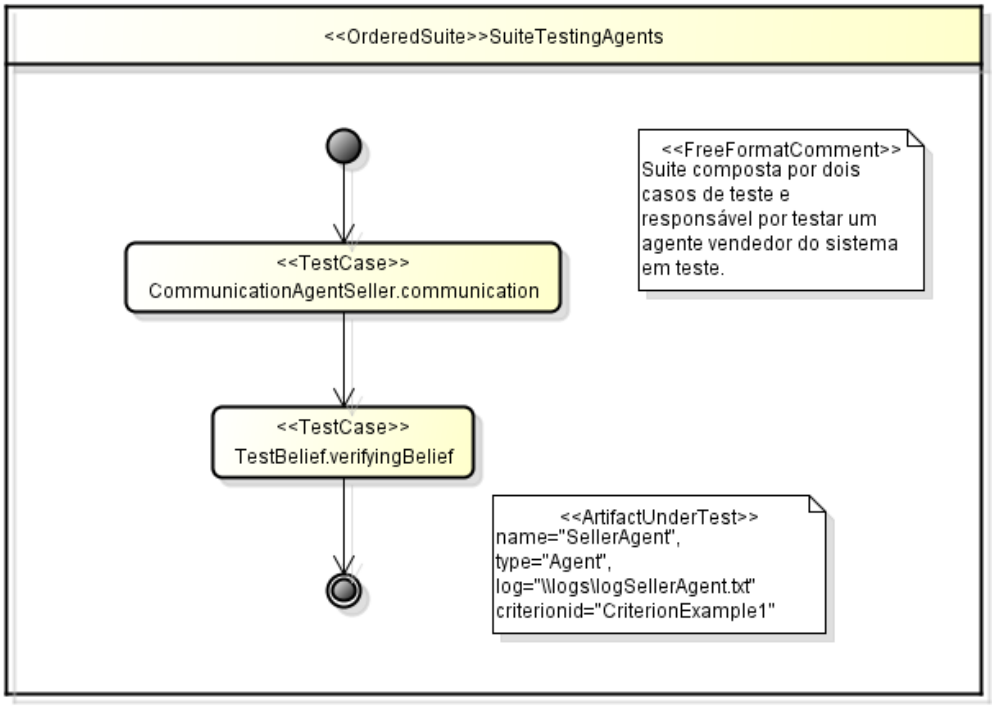

Figura 30. Modelando artefatos usados em sistemas.

6.2.8. Estereótipos para Relacionamentos de Dependência

Na Figura 31 são ilustrados alguns estereótipos responsáveis por informar a semântica de dependências modeladas entre entidades de teste. Esses tipos

de dependência foram criados a partir de situações comuns presentes em projetos de teste.

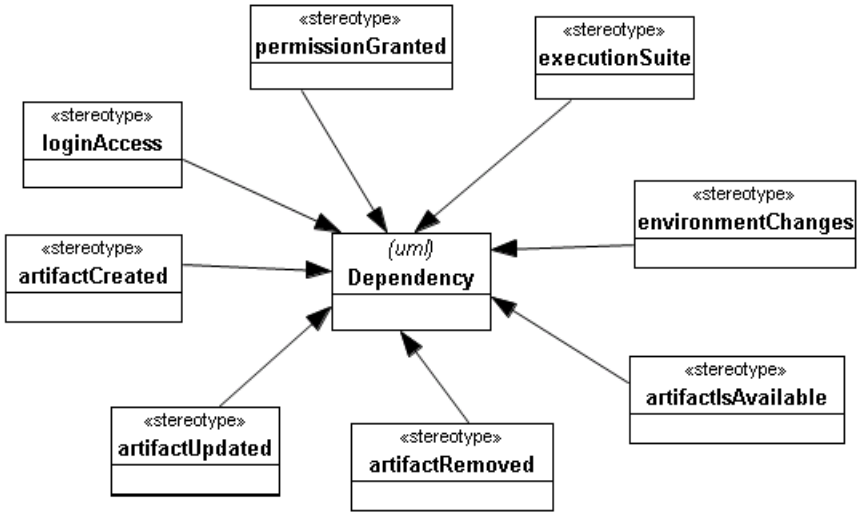

Figura 31. Semântica de dependências incluídas.

ArtifactCreated

Descrição: Estereótipo usado em um relacionamento de dependência quando um teste depende da criação de um ou mais artefatos (ex: arquivo, componente, entidade, etc.) realizadas por outro teste.

Notação: Representado pelo estereótipo <<artifactCreated>>. Caso deseje informar a quantidade de artefatos que devem ser criados, a notação <<artifactCreated [N]>> pode ser usada, onde 'N' representa a quantidade correspondente.

Restrição:

1. Deve ser aplicado em um relacionamento de dependência que envolva dois test contexts.

Exemplo: Na Figura 32 é apresentado um diagrama de classes em que o test context Login depende da criação de algum artefato realizada pelo test context CreatingUser. Caso deseje explicitar qual caso de teste o test context Login depende, no próprio relacionamento pode ser incluído

[<nome(s)_do(s)_caso(s)_de_teste>], assim como ilustrado na Figura 33. Nesse exemplo dois casos de teste são executados (create1 e create2) e o text context Login também depende da criação de dois artefatos para funcionar corretamente. Todos os estereótipos propostos na Figura 31 seguem a mesma ideia dos modelos apresentados nas Figuras 30 e 31.

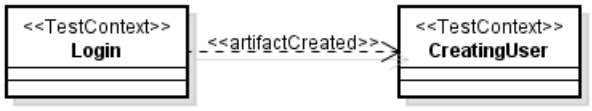

Figura 32. Dependência entre testes de criação de artefato.

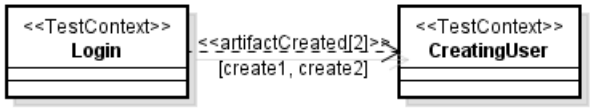

Figura 33. Dependência entre testes informando o caso de teste que depende.

ArtifactUpdated

Descrição: Estereótipo usado em um relacionamento de dependência quando um teste depende da atualização de algum artefato (ex: arquivo, componente, entidade, etc.) realizada por outro teste.

Notação: Representado pelo estereótipo <<artifactUpdated>>. Caso deseje informar a quantidade de artefatos que devem ser atualizados, a notação <<artifactUpdated [N]>> pode ser usada, onde 'N' representa a quantidade correspondente.

Restrição:

1. Deve ser aplicado em um relacionamento de dependência que envolva dois test contexts.

ArtifactRemoved

Descrição: Estereótipo responsável por indicar que algum teste depende da exclusão de algum artefato realizada por outro teste.

Notação: Representado pelo estereótipo <<artifactRemoved>>. Caso deseje informar a quantidade de artefatos que devem ser removidos, a notação <<artifactRemoved [N]>> pode ser usada, onde 'N' representa a quantidade correspondente.

Restrição:

1. Deve ser aplicado em um relacionamento de dependência que envolva dois test contexts.

EnvironmentChanges

Descrição: Estereótipo usado quando o teste depende de mudanças no ambiente realizadas por outro teste, como, por exemplo, mudanças em variáveis de ambiente, configurações no SUT, etc.

Notação: Representado pelo estereótipo <<environmentChanges>>.

Restrição:

1. Deve ser aplicado em um relacionamento de dependência que envolva dois test contexts.

PermissionGranted

Descrição: Estereótipo usado quando um teste depende de alguma permissão concedida no SUT por outro teste. Assim, um teste torna-se o responsável por conceder alguma permissão de acesso para que outro teste possa ser executado.

Notação: Representado pelo estereótipo <<permissionGranted>>.

Restrição:

1. Deve ser aplicado em um relacionamento de dependência que envolva dois test contexts.

LoginAccess

Descrição: Usado quando um teste depende do login realizado no sistema por outro teste.

Notação: Representado pelo estereótipo <<loginAccess>>.

Restrição:

1. Deve ser aplicado em um relacionamento de dependência que envolva dois test contexts.

ExecutionSuite

Descrição: Usado quando um suíte executa casos de teste de testes criados.

Notação: Representado pelo estereótipo <<executionSuite>>.

Restrição:

1. Deve ser aplicado em um relacionamento de dependência que envolva um suíte (OrderedSuite) e um test context.

ArtifactIsAvailable

Descrição: Usado quando um teste depende da disponibilidade de algum artefato para uso, como, por exemplo, agente de software, web-service, etc.

Notação: Representado pelo estereótipo <<artifactIsAvailable>>. Caso deseje informar a quantidade de artefatos que devem estar disponíveis, a notação <<artifactIsAvailable [N]>> pode ser usada, onde ‘N’ representa a quantidade correspondente.

Restrição:

1. Deve ser aplicado em um relacionamento de dependência que envolva dois test contexts.

6.3. Mapeamento do Modelo Conceitual para a Linguagem UTP-C

Visando ilustrar como as informações representadas pelo modelo conceitual apresentado no capitulo 5 foram manipuladas pela linguagem de modelagem UTP-C, a Tabela 2 ilustra como foi realizado esse mapeamento.

Informação no Modelo Conceitual	Informação na UTP-C
Entidade Test	Estereótipo <<TestContext>>
Atributo id da entidade Test	Representado pelo nome do TestContext, como, por exemplo, nome da classe ou atividade modelada.
Atributo tool da entidade Test	Atributo tool de TestContext
Tipo enumerado TestLevel	Atributo testLevel de TestContext
Tipo enumerado ExecutionType	Atributo isAutomatic de TestContext
Entidade TestCase	Estereótipo TestCase
Atributo priority da entidade TestCase	Atributo priority de TestCase
Atributo risk da entidade TestCase	Atributo risk de TestCase
Tipo enumerado TestType	Atributo type de TesCase
Tipo enumerado ObligatorinessType	Atributo isMandatory de um caso de teste.
Entidade OrderedSuite	Estereótipo OrderedSuite
Entidade SUT	Estereótipo SUT proposto pela UTP original sem realizar inclusões de propriedades e restrições.
Atributo currentVersion da entidade SUT	Atributo currentVersion no TestCase, e atributo currentSystemVersion no OrderedSuite.
Atributo desiredVersion da entidade SUT	Atributo desiredSystemVersion no TestContext e OrderedSuite.
Entidade ArtifactUnderTest	Estereótipo ArtifactUnderTest
Atributo id da entidade ArtifactUnderTest	Atributo name do comentário com estereótipo ArtifactUnderTest,
Atributo type da entidade ArtifactUnderTest.	Atributo type do comentário com estereótipo ArtifactUnderTest.
Atributo logPath da entidade ArtifactUnderTest.	Atributo log do comentário com estereótipo ArtifactUnderTest.
Entidade TestClassification	Estereótipo TestClassification
Entidade Development	Estereótipo Development
Entidade TestCriterion	Estereótipo TestCriterion

Tipo enumerado DependencyType	Estereótipos de dependência definidos na UTP-C, como, por exemplo, <<artifactCreated>>, <<executionSuite>>, <<loginAccess>>, etc.

Tabela 2. Mapeamento do modelo conceitual para UTP-C.

6.4. Avaliações Empíricas para a UTP-C

Visando analisar a linguagem UTP-C, duas avaliações empíricas (Kithenham, 2002; Wohlin et al., 1999) foram realizadas. O foco dessas avaliações foi investigar as seguintes hipóteses:

Hipóteses relacionadas à manutenção de modelos:

H1: UTP-C reduz o esforço em manter modelos de teste comparado a modelos baseados na UTP original com o uso livre de técnicas UML (ex: criação de novos estereótipos e comentários).

H2: Quando um designer usa UTP-C, a quantidade de erros na manutenção em modelos de teste é reduzida em relação ao uso da UTP original com técnicas UML.

Hipóteses relacionadas à criação de modelos:

H3: UTP-C reduzem o esforço na criação de modelos de teste comparados ao uso da UTP original com técnicas UML.

H4: Quando um designer usa UTP-C, a quantidade de erros na criação de modelos de teste é reduzida em relação ao uso da UTP original com técnicas UML.

Visando analisar as hipóteses apresentadas acima, as variáveis independentes consideradas foram as seguintes: (i) participantes envolvidos, (ii) treinamentos aplicados para modelar com UTP, UTP-C, e UML, e (iii) atividades solicitadas aos participantes para a criação e manutenção de modelos. Já as variáveis dependentes foram: (i) o tempo (esforço) gasto, e (ii) a quantidade de erros modelados pelos participantes (qualidade dos modelos) a partir das atividades solicitadas. Essas informações são detalhadas na descrição de cada avaliação aplicada.

Este capítulo inicialmente descreve como foi aplicada a avaliação empírica voltada para manutenção de modelos (hipóteses H1 e H2), seguido pela avaliação voltada à criação de modelos (hipóteses H3 e H4). Por fim são apresentados os seus resultados.

Apesar da avaliação de manutenção ser apresentada inicialmente neste capítulo, a avaliação de criação de modelos foi aplicada anteriormente. Decidimos seguir tal ordem, pois consideramos mais fácil a explicação da

avaliação de manutenção, pois envolveu menos participantes do que a avaliação de criação.

6.4.1. Avaliação para Manutenção de Modelos

Para avaliar as hipóteses H1 e H2, dezesseis pessoas (ver Tabela 3) com diferentes habilidades e experiência na área de teste e modelos responderam dois tipos de questionários aplicados. Metade desses participantes teve formação na Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), enquanto que a outra metade teve formação em outras universidades: Universidade Federal Fluminense (2 pessoas), Universidade Federal de Viçosa (3 pessoas), Universidade Cândido Mendes (1 pessoa) e Universidade Estácio de Sá (2 pessoas).

Pessoas e seus Papéis	Anos de Experiência com Testes	Descrição
(4 pessoas) 1 gerente 1 analista de requisito 2 desenvolvedores sênior	> 3 anos	Alto conhecimento e experiência na área de teste, modelagem UML e conceitos de desenvolvimento.
(10 pessoas) 5 desenvolvedores sênior 5 testadores	1 a 3 anos	Já trabalhou extensivamente com testes funcionais manuais e automatizados. Bom conhecimento na modelagem UML.
(2 pessoas) 2 desenvolvedores junior	< 1 ano	Pouca experiência com testes e modelagem UML. No entanto, bom conhecimento sobre os principais conceitos de teste.

Tabela 3. Perfil dos participantes da avaliação de manutenção.

Um dos questionários aplicados (questionário A) solicitou a manutenção de modelos baseados na UML Testing Profile com o uso de técnicas UML (ex: uso de estereótipos e comentários), enquanto que o outro questionário (questionário B) solicitou a manutenção de modelos baseados na UTP-C. Para cada

questionário duas versões foram criadas a fim de mapear situações de dois sistemas. Os sistemas considerados foram referentes ao domínio de: (i) suprimento e estoque de petróleo e derivados do petróleo e (ii) alocação de lotes de petróleo no Brasil.

Alguns dos principais objetivos do sistema relacionado ao estoque e suprimento de petróleo são: (i) registrar rotas (isto é, caminhos) baseados em dutos e navios que podem ser usados para transportar os produtos derivados; (ii) planejar quando esses produtos deverão chegar em terminais ou refinarias localizadas em diferentes lugares do Brasil; (iii) planejar as melhores rotas para transportar algum produto; (iv) apresentar dados realizados responsáveis por informar quando e quais produtos chegaram em terminais e refinarias; (v) apresentar um conjunto de relatórios e gráficos envolvendo dados planejados e realizados; e (vi) apresentar um mapa 2D do Brasil ilustrando rotas, terminais e refinarias cadastradas no sistema. Na Tabela 4 são apresentadas algumas características desse sistema em mais detalhe.

Tempo de projeto	8 anos
Pessoas envolvidas	30 pessoas
Time de teste	7 pessoas
Status	Em andamento
Tamanho do modelo	46 casos de uso 580 classes
Banco de dados	283 tabelas
Casos de teste	46 testes unitários de banco 15 testes de desempenho 100 testes funcionais automatizados 558 testes funcionais manuais

Tabela 4. Informação sobre o sistema de estoque e suprimento de petróleo.

Já o sistema referente à alocação de lotes de petróleo tem como principais funcionalidades: (i) permitir a criação de lotes de petróleo para serem alocados em alguma refinaria; (ii) apresentar quanto está sendo produzido de petróleo em cada refinaria; (iii) gerenciar vendas de lotes; (iv) visualizar quais petróleos foram importados a partir de outro país e aqueles produzidos no Brasil; e (v) gerar relatórios e gráficos que detalhem informações dos produtos alocados em refinarias no Brasil. Na Tabela 5 são apresentados mais detalhes desse sistema.

Tempo de projeto	7 anos
Pessoas envolvidas	13 pessoas
Time de teste	4 pessoas
Status	Em andamento
Tamanho do modelo	65 casos de uso 380 classes
Banco de dados	176 tabelas
Casos de teste	46 testes unitários de banco 21 testes funcionais automatizados 81 testes funcionais manuais

Tabela 5. Informação sobre o sistema alocação de petróleo.

O nível de complexidade dos questionários aplicados foi similar para não influenciar os resultados da avaliação. Esses questionários foram avaliados por três especialistas em modelagem e engenharia de software experimental. Além disso, 50% dos participantes responderam questionários baseados em situações do sistema voltado para o suprimento e estoque de petróleo, enquanto que os outros 50% responderam questionários baseados em situações do sistema de alocação de lotes de petróleo. No Apêndice A do documento é apresentado um exemplo de questionário aplicado a fim de exemplificar o nível de complexidade dos modelos envolvidos em todos os questionários aplicados. Perceba que a primeira parte do questionário do Apêndice A solicita informações sobre o participante, enquanto que a segunda parte solicita mudanças em diagramas UTP-C.

Na Tabela 6 está detalhado o objetivo e quais conceitos de teste estavam envolvidos em cada questão dos questionários aplicados. Esses questionários foram divididos em três exercícios e cada um desses exercícios apresentou um diagrama de classes que deveria ser atualizado por algum participante a partir dos requisitos definidos no enunciado. Os diagramas não baseados na UTP-C foram gerados e providos a partir das respostas dos participantes envolvidos na avaliação voltada a criação de modelos (explicado na próxima subseção 6.4.2).

Questões	Principais Informações de Teste Envolvidas	Objetivo Principal
Questão 1	• Caso de teste • Test context • Tipo de teste (ex: functional, stress, etc.) • Nível de teste (ex: unitário, sistema, etc.) • Obrigatoriedade de cada caso de teste • Prioridade de cada caso de teste • Versão do sistema em teste que algum teste está correntemente atualizado. • Versão do sistema em teste que desejasse que algum teste esteja atualizado. • Pacote de desenvolvimento	Manter um diagrama de classe que modela um conjunto de testes com informações úteis para gerenciá-los.
Questão 2	• Caso de Teste • Test context • Ordered suites • Dependências entre test contexts • Dependências entre suites e test contexts • Semântica das dependências	Manter um diagrama de classe que foca na representação de dependências que envolvem testes e suites.
Questão 3	• Test context • Classificação de teste	Manter um diagrama de classe que representa classificações de teste.

Tabela 6. Detalhamento dos questionários aplicados.

Essa avaliação foi aplicada em seis etapas, assim como ilustrado na Figura 34. A primeira etapa realizou o treinamento dos participantes para explicar como eles poderiam manter diagramas UML já criados baseados na UTP e com o uso livre de técnicas UML. Assim, o foco do treinamento foi apresentar como as pessoas poderiam modelar os conceitos de teste considerados no modelo conceitual apresentado no capítulo 5. Como tínhamos dezesseis participantes, eles foram divididos em quatro grupos. As principais razões para realizar essa divisão foram às seguintes: (i) dificuldade em marcar um horário único para que todos os envolvidos participassem do treinamento, (ii) maior facilidade em responder dúvidas levantadas pelos grupos, e (iii) procurar dividir os participantes em grupos com níveis similares de conhecimento. Cada treinamento durou em torno de quarenta a noventa minutos. Esse tempo variou principalmente devido o nível de conhecimento das pessoas.

A segunda etapa aplicou individualmente aos participantes o questionário A, que solicitou a manutenção de modelos baseados na UTP com técnicas UML. As respostas dos participantes foram controladas e as questões foram baseadas em situações reais de projetos de teste, assim como já mencionado. Para realizar esse controle de tempo em cada questão, o participante teve que informar o horário (formato hh:mm:ss) que iniciou cada questão, assim como o horário que a terminou. Decidimos disponibilizar uma pessoa para verificar se esse procedimento estava sendo feito de forma adequada e para tirar possíveis dúvidas identificadas pelo participante.

O foco da terceira etapa foi preparar os participantes para manter modelos baseados na UTP-C. Para realizar tal treinamento, a mesma ideia de dividir as pessoas em grupos, aplicada na etapa 1, foi adotada. O treinamento durou de trinta a sessenta minutos, dependendo do nível de conhecimento dos participantes.

A quarta etapa envolveu o design de outro questionário (questionário B), que solicitou a manutenção de modelos baseados na UTP-C aos envolvidos na avaliação. Esse questionário seguiu a mesma estrutura apresentada na Tabela 6 e manteve a mesma complexidade das questões providas no questionário da segunda etapa. Visando avaliar essas preocupações, três especialistas em modelagem e engenharia de software experimental validaram tal questionário.

Tanto os questionários da segunda e quarta etapas, como os respectivos treinamentos foram aplicados em diferentes ordens. Consequentemente, parte dos participantes (50% das pessoas) respondeu o questionário A seguido pelo B, enquanto que a outra parte respondeu inicialmente o questionário B seguido pelo A. A principal razão dessa ideia foi evitar que ações pudessem influenciar no resultado da avaliação, como, por exemplo, ocorrer algum tipo de aprendizado.

Em seguida, na quinta etapa, entrevistas foram realizadas com cada participante. A principal ideia dessa etapa foi entender os problemas que cada pessoa identificou quando realizou a manutenção dos diagramas nos questionários A e B.

Finalmente, a sexta e última etapa recolheu e analisou em detalhe todas as respostas dos questionários providas pelos participantes. Esses resultados são mencionados em detalhe na subseção 6.4.3.

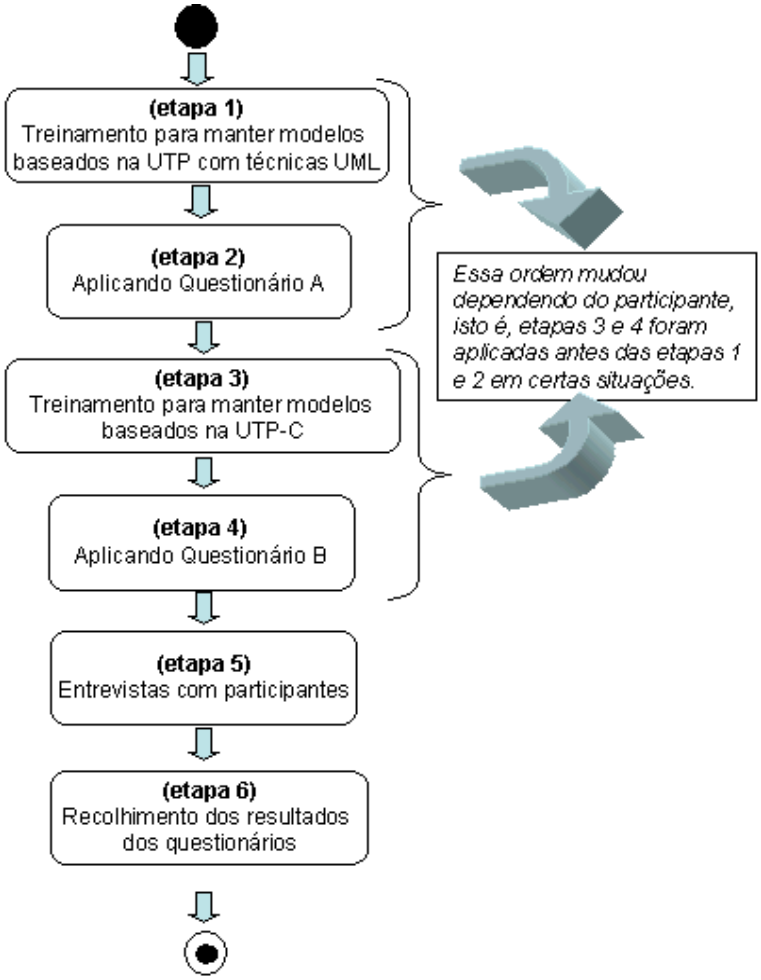

Figura 34. Procedimentos empíricos adotados na avaliação de manutenção.

6.4.2. Avaliação para Criação de Modelos

Para analisar as hipóteses H3 e H4, outra avaliação empírica foi aplicada com quarenta e um participantes com diferentes conhecimentos e experiências na área de teste e modelos. Os participantes envolvidos tiveram formações em diferentes universidades, assim como apresentado a seguir: PUC-Rio (19 pessoas), Universidade Federal Rural do Rio de Janeiro (14 pessoas), Universidade Federal Fluminense (2 pessoas), Universidade Federal de Viçosa (3 pessoas), Universidade Cândido Mendes (1 pessoa) e Estácio de Sá (2 pessoas). Detalhes adicionais do perfil desses participantes são apresentados na Tabela 7.

Pessoas e seus Papéis	Anos de Experiência	Descrição
(9 pessoas) 1 gerente 1 analista de requisito 5 desenvolvedores sênior 2 analistas de teste	> 3 anos	Alto conhecimento e experiência na área de teste, modelagem UML e conceitos de desenvolvimento.
(11 pessoas) 1 analista de requisitos 5 desenvolvedores sênior 5 testadores	1 a 3 anos	Já trabalhou extensivamente com testes funcionais manuais e automatizados. Bom conhecimento na modelagem UML.
(21 pessoas) 21 desenvolvedores junior	< 1 ano	Pouca experiência com testes e modelagem UML. No entanto, bom conhecimento sobre os principais conceitos de teste.

Tabela 7. Perfil das pessoas que participaram da avaliação de criação de modelos.

Similar à avaliação de manutenção, a avaliação de criação de modelos também adotou seis etapas (ver Figura 34). A seguir, são apresentadas as principais diferenças entre a avaliação de manutenção e a de criação de modelos.

A primeira etapa realizou o treinamento dos participantes para que pudessem criar modelos de teste a partir do uso da UTP original e de técnicas livres da UML (ex: uso de estereótipos e comentários) para complementar a modelagem da UTP. Como tivemos quarenta e um participantes, dividimos essas pessoas em seis grupos, sendo que o treinamento de cada grupo durou de quarenta a noventa minutos. As razões para essa divisão foram às mesmas da avaliação de manutenção. Além disso, o tempo de treinamento variou devido o nível de conhecimento dos participantes nas áreas de teste e modelos.

Na segunda etapa foi aplicado um questionário solicitando a criação de diagramas de classe baseados na UTP e o uso de técnicas UML. A estrutura do questionário foi similar à estrutura apresentada na Tabela 6, assim como a complexidade aplicada.

Já a terceira etapa focou no treinamento dos participantes para criar modelos baseados na UTP-C. Para realizar tal treinamento, os envolvidos também foram divididos em seis grupos, assim como na etapa 1. O treinamento durou de 30 a 60 minutos.

A quarta etapa aplicou outro questionário aos participantes solicitando a criação de diagramas de classe baseados na UTP-C. Tanto a estrutura como a complexidade desse questionário foi similar ao questionário aplicado na segunda etapa. Para evitar qualquer influência nas questões dos questionários três especialistas em modelagem e engenharia de software experimental realizaram a validação deles.

Assim como a avaliação de manutenção, a avaliação de criação de modelos foi aplicada em diferentes ordens, isto é, parte dos participantes realizou as etapas 1 e 2 seguidas pelas etapas 3 e 4. No entanto, a outra parte fez o inverso, isto é, inicialmente as etapas 3 e 4, seguidas pelas etapas 1 e 2.

Já a quinta etapa realizou entrevistas com cada participante para identificar quais facilidades e dificuldades eles encontraram enquanto criavam os diagramas solicitados. Por último, na sexta etapa, os dados providos pelos participantes foram recuperados e analisados. Detalhes da quinta e sexta etapa são apresentados na próxima subseção.

6.4.3. Resultados das Avaliações Aplicadas

Na avaliação de manutenção de modelos todos os participantes informaram que não tiveram dificuldades em manter os diagramas baseados na UTP-C. No entanto, alguns deles mencionaram que dependendo da quantidade de classes modeladas, a manutenção pode ser muito custosa e trabalhosa. A seguir, há frases mencionadas pelos participantes:

> "*Manter diagramas baseados na UTP-C é uma atividade fácil de ser realizada dependendo da quantidade de classes modeladas.*"
>
> "*UTP-C é uma boa abordagem para modelar informações de teste. No entanto, uma avaliação envolvendo uma grande quantidade de classes deveria ser feita."*

O problema referente à manutenção de um grande volume de classes e diagramas é conhecido pela comunidade de modelagem. Consequentemente, para identificarmos o impacto dessas novas abordagens em tal situação,

pretendemos aplicar a UTP-C em grandes projetos reais de teste para que uma melhor análise na manutenção seja realizada.

Na Figura 35 são ilustrados os resultados obtidos a partir da avaliação referente à manutenção de modelos. A Figura 35a mostra que 93.75% dos participantes responderam o questionário B (etapa 4 da subseção 6.4.1 que usa a UTP-C) em menos de 40 minutos, enquanto que para o questionário A (etapa 2 da subseção 6.4.1) foram 87.50%. Outro dado apresentado na Figura 35a é que 12.50% dos participantes levaram mais do que 40 minutos para responder o questionário A, enquanto que para o questionário B somente 6.25% dos envolvidos.

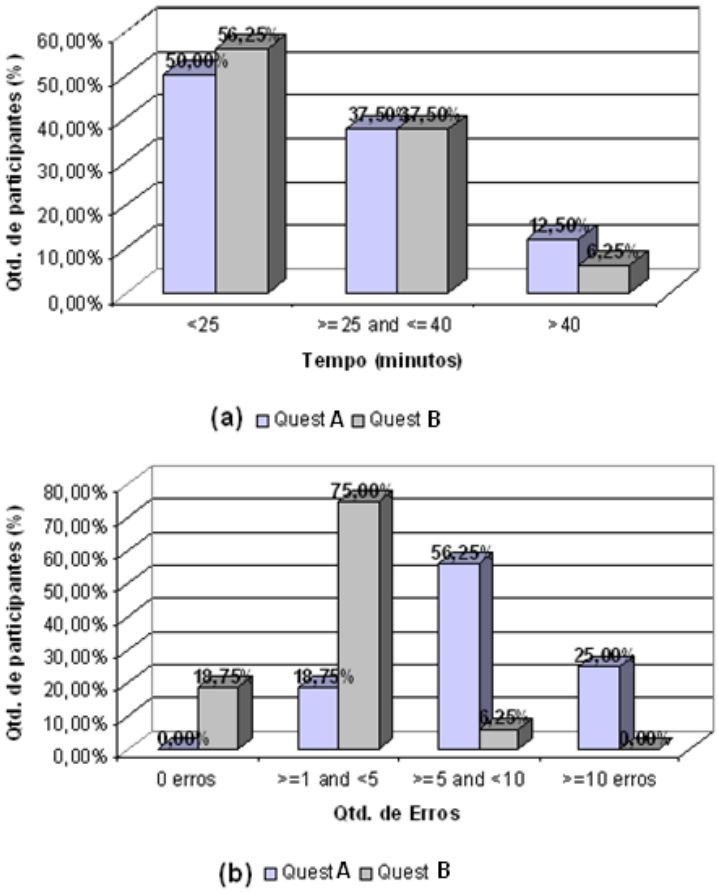

Figura 35. Resultados da avaliação de manutenção de modelos.

Já a Figura 35b informa a quantidade de erros modelados pelos participantes em cada questionário aplicado. No questionário B, 18.75% dos participantes modelaram corretamente 100% dos exercícios solicitados. Por outro lado, todos os participantes tiveram algum problema no questionário A. Além disso, 25% dos envolvidos tiveram mais do que dez erros na modelagem do questionário A. Enquanto que no questionário B nenhum participante teve

mais do que dez erros. Os tipos de erros identificados nas respostas providas pelos envolvidos são apresentados na Tabela 8. Perceba que o fato de haver um padrão de modelagem para modelar informações de teste a partir da UTP-C ajudou a diminuir a quantidade de erros na manutenção dos modelos.

Na avaliação referente à criação de modelos, os participantes consideraram mais fácil o uso da UTP-C do que a UTP original com o uso de técnicas UML. Uma das principais razões foi que a UTP não manipula diversas informações de teste solicitadas para modelar. Além disso, todos os participantes comentaram que a UTP-C é uma abordagem fácil e intuitiva de trabalhar. A seguir, há alguns exemplos de frases mencionadas pelos participantes.

"*UTP-C é fácil e intuitiva para realizar a criação de diagramas de classe compostos por informações de teste.*"

"*Não tive nenhum problema para criar modelos baseados na UTP-C.*"

Questionário A (Uso da UTP e técnicas UML)	**Questionário B (Uso da UTP-C)**
1) Problema no uso da sintaxe UML.	1) Problema no uso da sintaxe UML.
2) Não modelou algum ponto solicitado.	2) Não modelou algum ponto solicitado.
3) Uso de comentários que não são claros para prover algum tipo de informação nos diagramas.	3) Erro no uso da sintaxe da UTP-C.
4) Não seguiu um padrão de modelagem, gerando diferentes respostas para situações similares.	4) Modelou de forma errada alguma informação, como, por exemplo, nome do test context ou caso de teste errado.
5) Erro no uso da sintaxe da UTP.	-
6) Modelou de forma errada alguma informação, como, por exemplo, nome do test context ou caso de teste errado.	-

Tabela 8. Tipos de erros identificados nas respostas informadas pelos participantes nas avaliações aplicadas.

Na Figura 36 são ilustrados os resultados obtidos a partir da avaliação voltada à criação de modelos. A Figura 36a mostra que 95.63% dos participantes responderam o questionário B (etapa 4 da subseção 6.4.2) em menos de 50 minutos, enquanto que somente 43.91% dos participantes gastaram o mesmo período de tempo para responder o questionário A (etapa 2 da subseção 6.4.2).

Outro dado apresentado na Figura 36a é que 39.02% dos participantes levaram mais de 60 minutos para responder o questionário A, enquanto que no questionário B nenhum participante gastou tal tempo.

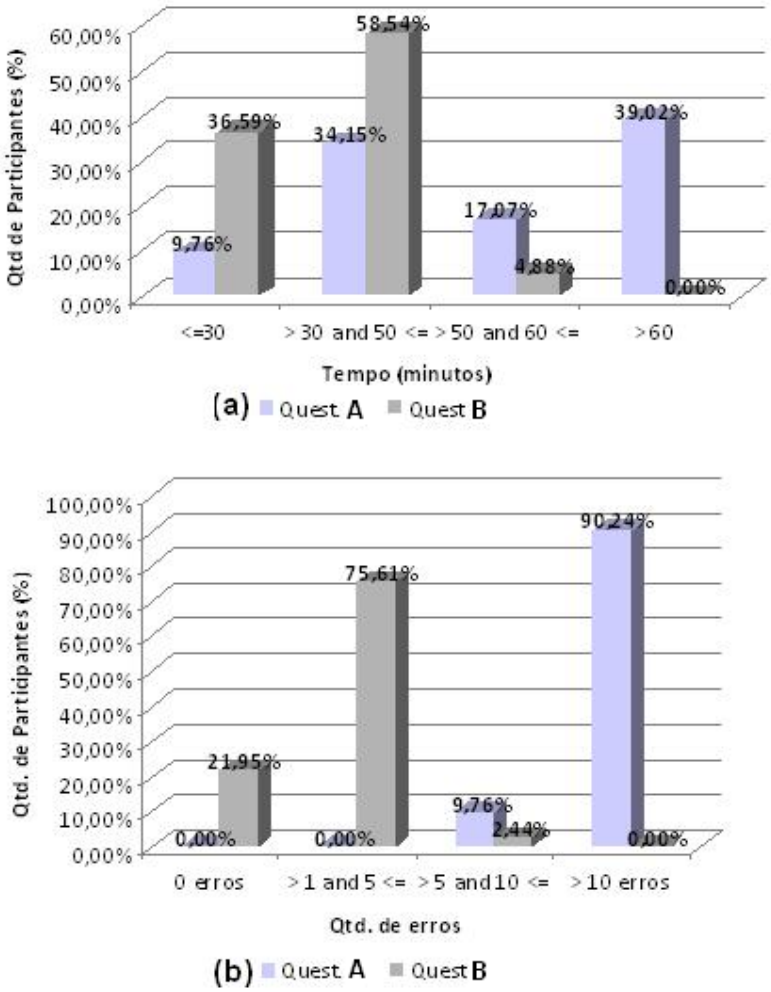

Figura 36. Resultados da avaliação de criação de modelos.

Já a Figura 36b informa a quantidade de erros modelados pelos participantes em cada questionário aplicado. No questionário B 21.95% dos participantes modelaram corretamente 100% dos exercícios solicitados. Por outro lado, no questionário A todos os participantes tiveram algum problema. Além disso, 90.24% dos envolvidos tiveram mais de dez erros na modelagem do questionário A. Por outro lado, no questionário B nenhum participante teve mais de dez erros. A natureza dos erros identificados a partir das respostas providas pelos participantes foram os mesmos da avaliação de manutenção (ver Tabela 8).

A partir da análise desses dados pudemos concluir que as hipóteses H1, H2 e H3 são verdadeiras, pois o tempo e a quantidade de erros identificados foram substancialmente menor no questionário B do que o questionário A. No entanto, a avaliação não pôde concluir a hipótese H4, já que os dados recolhidos não apresentaram diferenças relevantes quando comparado os questionários aplicados. Assim, uma análise mais detalhada envolvendo mais participantes será realizada para concluirmos a análise da hipótese H4.

Além disso, algumas informações representadas pela UTP-C não foram consideradas nas avaliações apresentadas, como, por exemplo, risco de produto, ferramenta a ser usada para executar algum teste e critério de seleção dos testes. A principal razão, é que essas informações foram incluídas na UTP-C após a aplicação das avaliações. No entanto, pretendemos aplicar alguma avaliação complementar para analisá-las também em relação à criação e manutenção de modelos.

6.5. Considerações Finais

Neste capítulo foi apresentada a linguagem UML Testing Profile for Coordination (UTP-C) voltada em modelar informações úteis para a coordenação dos testes de software. Para explicá-la foi apresentado: (i) os mecanismos adotados para realizar a extensão na UML e na UTP original, já que a UTP-C estende tais linguagens (Seção 6.1); (ii) seu meta-modelo acompanhado com simples exemplos (Seção 6.2); (iii) o mapeamento das informações consideradas no modelo conceitual, proposto no capítulo 5, em relação à UTP-C (Seção 6.3); e (iv) avaliações da linguagem UTP-C envolvendo pessoas com diferentes níveis de conhecimento na área de modelos e testes de software.

Visando exemplificar como a linguagem UTP-C pode ser usada em conjunto com o framework JAAF+T, o próximo capítulo apresenta um estudo de caso relacionado ao domínio de compras e vendas de produtos.