5
Modelo Conceitual de Teste

Visando ilustrar a relação das informações de teste mencionadas no capitulo 3 e assim ajudar na atividade de gerência dos testes e na geração dos arquivos XML usados pelo framework JAAF+T, o modelo conceitual ilustrado na Figura 11 é apresentado. Esse modelo também usou como base o estudo apresentado em (Costa et al., 2010b), responsável por identificar relevantes informações de teste não suportadas por conhecidas linguagens de modelagem propostas na literatura e que poderiam ser modelados. Tal estudo usou como base a experiência prática de diferentes equipes de teste em coordenar e executar testes em sistemas industriais, contando assim com a participação de profissionais com diferentes conhecimentos na área de teste de software e modelos para identificar essas informações.

UML Testing Profile original (Baker et al., 2007), AML (Trost e Cavarra, 2012) e UTML (UTML, 2012), são exemplos de conhecidas linguagens de modelagem de teste que não representam diversas informações úteis para a coordenação dos testes. Seguem alguns exemplos de informações não manipuladas por essas abordagens: (i) qual versão do SUT cada teste é capaz de testar, (ii) quais testes são obrigatórios, (iii) quais tipos de teste foram criados (ex: funcional, performance, etc.), (iv) quais níveis de teste estão sendo contemplados (unitário, integração, sistema e aceitação) (v) quais tipos de dependências existem entre os testes modelados (ex: criação de algum artefato, tempo de execução, etc.), (vi) quais testes são automatizados e manuais, (vii) quais critérios de seleção de testes algum SUT possui, (viii) quais artefatos do SUT devem ser validados por quais testes, (viii) quais ferramentas de teste são usadas, etc.

As principais entidades definidas no modelo conceitual proposto foram as seguintes: caso de teste, teste, sistema em teste, suíte, artefato em teste, classificação de teste, pacote de desenvolvimento e critério de seleção. Consideramos que cada entidade possui propriedades e relacionamentos com outras entidades (Booch et al., 2005). A fim de esclarecer quais informações compõem cada uma delas, essas entidades são apresentadas em detalhe neste capítulo.

Usamos a expressão modelo conceitual de forma intercambiável com o termo metamodelo usado pela OMG (OMG, 2012). O modelo proposto visa incorporar aspectos estruturais e dinâmicos de testes de software na UML para ajudar na coordenação dos testes. Além disso, a semântica do metamodelo é apresentada usando a linguagem natural no estilo do metamodelo da UML (UML, 2012).

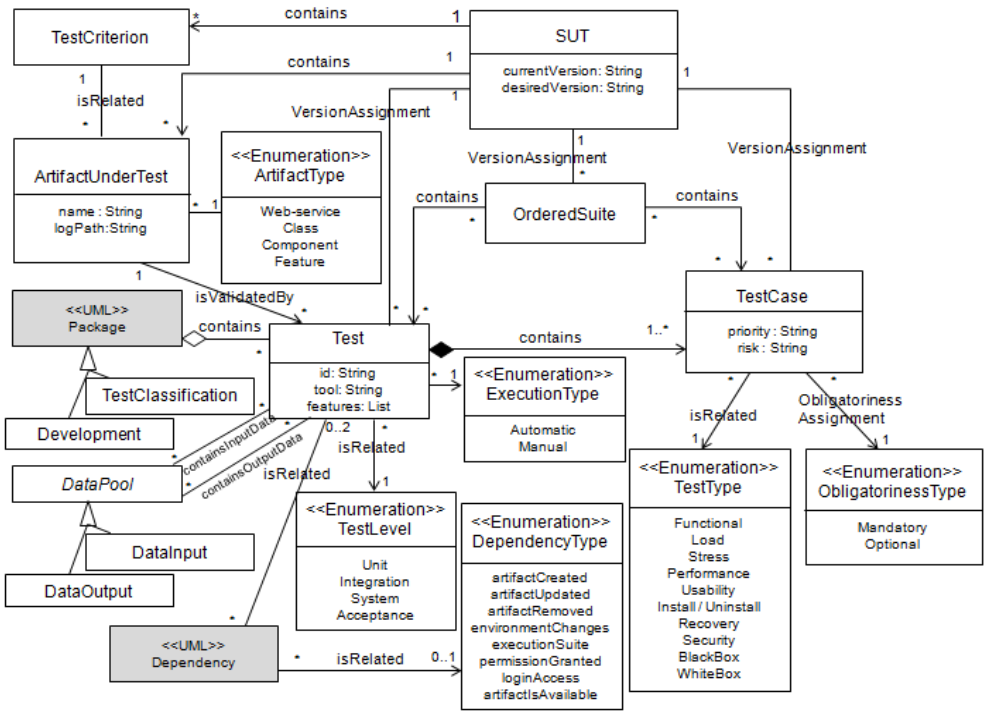

Figura 11. Modelo detalhando relacionamentos entre informações de teste.

5.1. Caso de Teste

Cada caso de teste é desenvolvido para um objetivo particular ou condição de teste, como, por exemplo, exercitar um caminho particular do SUT ou verificar o comportamento de algum requisito específico (TMMi, 2012). Assim, dependendo da versão do sistema em teste, cada caso pode ser considerado obrigatório ou opcional para execução (uso do tipo enumerado *ObligatorinessType*). Dessa forma, essa identificação ajudará a planejar e orientar os testadores sobre quais casos deverão ser executados em um ciclo de teste.

Mesmo identificando os casos obrigatórios e opcionais, há situações que tornam necessário detalhar a prioridade da execução dos casos (uso do atributo *priority* na entidade *TestCase*). Essa informação é particularmente útil quando o tempo disponível para testar algum projeto é crítico e quando o conjunto de casos de teste obrigatórios é grande para um curto período de tempo. Assim, torna-se necessário definir quais casos obrigatórios possuem maior prioridade e devem ser executados inicialmente. Tal ideia ajuda na organização interna das equipes de teste, assim como mencionado pelo TMMi (TMMi, 2012).

Outra informação que ajuda na gerência dos testes é conhecer o risco do produto relacionado a algum caso de teste, caso alguma falha seja identificada a partir de sua execução (ISTQB, 2012; TMMi, 2012). Para representar tal informação definimos um atributo *risk* na entidade *TestCase*.

Como diferentes casos de teste podem ser criados, identificar seu tipo ajuda a entender sua finalidade, assim como contribui na delegação de tarefas para membros de alguma equipe, já que tais membros podem ter diferentes níveis de conhecimento nos diferentes tipos de teste (TMMi, 2012; ISTQB, 2012). Para representar essa informação, o tipo enumerado *TestType* foi definido, listando conhecidos tipos propostos na literatura, como, por exemplo, casos de teste funcionais, desempenho, usabilidade, segurança, etc.

Todas as informações apresentadas acima são informações estáticas que trazem informações adicionais dos casos a serem executados em algum SUT. Portanto, o uso de um diagrama estático, como, por exemplo, diagrama de classes da UML pode ser usado para representar tais informações.

5.2. Teste

Testes contêm um ou mais casos de teste (829-2008, 2012) a fim de organizá-los e gerenciá-los de forma conjunta, já que possuem alguma relação de responsabilidade em algum projeto. Assim, testes podem ter responsabilidades em comum caracterizando a relação com algum nível de teste (ISTQB, 2012; TMMi, 2012). Para expressar qual nível está relacionado a cada teste, foi definido o tipo enumerado *TestLevel*, responsável por listar conhecidos níveis adotados na literatura: unitário, integração, sistema e aceitação. Caso um teste esteja relacionado ao nível de sistema ou aceitação, torna-se importante informar qual(is) feature(s) do SUT está(ão) sendo testada(s) (uso do atributo

features na entidade *Test*). Além disso, cada teste deve ter um id único (uso do atributo *id* na entidade *Test*) para que não haja dificuldade na sua identificação.

Outro ponto importante na gerência dos testes é conhecer se cada um deles será executado de forma automática ou manual (uso do tipo enumerado *ExecutionType*). Quando há uma grande quantidade de testes, a identificação de quais são automáticos e manuais é uma atividade que pode gerar certo custo de tempo. No entanto, essa identificação é útil para guiar o testador a entender o esforço a ser concedido, principalmente quando a ferramenta a ser usada também é conhecida (representado pelo atributo *tool* na entidade *Test*).

Por fim, cada teste pode estar relacionado a diversos dados de entrada (representado pela entidade DataInput) e saída (uso da entidade DataOutput) para serem usados em sua(s) execução(ões). Esses dados caraterizam diferentes condições de teste.

5.3. Artefato em Teste

Diversos artefatos (representado pela entidade *ArtifactUnderTest*) de um SUT podem ser validados por diferentes testes de software. Assim, entender a rastreabilidade de quais testes devem ser executados para validar algum artefato, ajuda a gerenciar os testes.

Diferentes tipos de artefatos podem ser testados, como, por exemplo, agentes de software, serviços web, classes, componentes, etc. Visando representar esses diferentes tipos, o tipo enumerado *ArtifactType* foi definido. Além disso, cada artefato deve ter um log e um nome. O log (representado pelo atributo *logPath*) é responsável por armazenar os resultados dos testes executados. A partir dele, pode-se analisar se algum artefato está ok para uso. Já o nome permite que um artefato possa ser identificado de forma mais fácil (uso do atributo *name* na entidade *ArtifactUnderTest*).

5.4. Suite de Teste

A entidade *OrderedSuite* representa um suíte de teste, que nada mais é que um componente de código responsável por executar um conjunto de testes automaticamente em uma ordem específica. Na literatura há diferentes ferramentas e APIs que podem ser usadas para criar suítes e executá-las, como, por exemplo, Rational Quality Manager (RQM, 2012), DBUnit (DBUnit, 2012), e

Rational TestManager (RTM e RMT, 2012). Vale ressaltar a diferença entre testes e suites, onde testes são compostos por casos de teste, enquanto que suites são exclusivamente responsáveis por executar testes ou casos de teste implementados em alguma classe.

As suites podem ser criadas e atualizadas para alguma versão específica de um SUT. Por isso que no modelo conceitual é apresentada uma associação entre as entidades *OrderedSuite* e SUT. Portanto, modelar suítes ajuda a entender o fluxo de execução dos testes a fim de validar artefatos em teste. Para representar esses fluxos de execução, o ideal é que algum diagrama dinâmico da UML, como, por exemplo, um diagrama de atividade. Se realmente um diagrama de atividade for usado, cada atividade modelada pode ser considerada um teste de software a fim de esclarecer a ordem de execução de um suíte. Adicionalmente, um estereótipo pode ser adicionado em cada atividade para expressar a ideia que elas representam testes.

5.5. Classificação de Teste

Uma informação considerada útil para a coordenação dos testes é o uso de classificações. Classificação de teste ajuda a agregar de forma visual testes a partir de qualquer informação que o designer considere relevante. Consequentemente, ela ajuda no planejamento dos testes, pois permite identificar de forma mais fácil outras relações conceituais entre testes. Diferentes classificações podem ser usadas, como, por exemplo, testes novos e atualizados para algum SUT. Testes novos são aqueles criados exclusivamente para validar os novos requisitos definidos no sistema em teste, enquanto que os testes atualizados são aqueles impactados, devido esses requisitos.

Para representar essas classificações, a entidade *TestClassification*, que estende da metaclasse *Package* da UML, foi representada. Por ser um pacote, torna-se possível agregar um ou mais testes a partir de alguma classificação. Essa representação permite que visões de grupamento similares às oferecidas pelas ferramentas Rational TestManager (RTM e RMT, 2012) e Rational Quality Manager (RQM, 2012) sejam disponibilizadas.

5.6. Pacote de Desenvolvimento

O pacote de desenvolvimento (extensão da metaclasse *Package*) é onde estão realmente armazenadas entidades criadas para algum projeto, como, por exemplo, testes ou suítes. Perceba que o objetivo central das entidades *TestClassification* e *Development* (ver Figura 11) é apresentar uma semântica adicional de grupamentos de entidades. A fim de permitir essas representações, estereótipos podem ser usados em pacotes modelados na UML. Assim, como a UML permite a representação de diferentes níveis de abstrações, o estereótipo referente ao pacote de desenvolvimento garante que as entidades contidas em tais pacotes estão realmente armazenadas no caminho informado nos diagrama correspondente.

5.7. Sistema em Teste

Uma informação importante na coordenação dos testes é conhecer a relação de cada teste, caso de teste e suite com a versão do sistema em teste. Essa informação é crucial, pois dependendo da versão considerada, um específico caso de teste pode ser ou não executado. Assim, a entidade SUT foi definida no modelo conceitual para representar a versão corrente de um sistema em teste (representado pelo atributo *currentVersion*) e uma nova versão que se deseja testar (representado pelo atributo *desiredVersion*). Tais informações devem estar associadas às entidades *Test*, *TestCase* e *OrderedSuite*. Perceba que um teste pode referenciar uma versão antiga do SUT, assim como uma nova versão.

5.8. Critério de Seleção de Testes

Um sistema em teste pode ter diversos critérios que auxiliem a escolha de quais testes deverão ser executados para validar algum artefato específico. Diferentes informações podem ser usadas para definir um critério, como, por exemplo, nível de teste, tipo de teste, prioridade, risco, etc. Assim como mencionado no capítulo 3, exemplos de critérios são os seguintes: (i) executar somente testes de regressão, (ii) testes com prioridade alta e que sejam de regressão, (iii) somente aqueles com prioridade alta, unitários e que sejam testes

de segurança, etc. Para representar a ideia desses critérios, a entidade *TestCriterion* foi definida no modelo conceitual ilustrado na Figura 11.

5.9. Semântica de Dependências de Testes

Outra informação adicional apresentada no modelo conceitual e de grande importância para a coordenação dos testes é a dependência entre testes. Essa informação é considerada crucial para definir a correta ordem de execução dos testes. Para representar tal ideia, a metaclasse *Dependency* da UML 2.0 é reusada e passa a estar relacionada com o tipo enumerado *DependencyType* que descreve os tipos de dependências entre testes. A*rtifactCreated*, por exemplo, é um dos tipos apresentados no tipo enumerado *DependencyType*, e é usado quando um teste depende de algum artefato (ex: arquivo, componente, entidade, etc.) criado por outro teste. Já a*rtifactUpdated* indica que algum teste depende da atualização de algum artefato. Já as outras opções dizem respeito: (i) a exclusões de algum artefato (ex: artifactRemoved), (ii) envolvimento de um conjunto de artefatos (ex: setArtifactCreated, setArtifactRemoved e setArtifactUpdated), (iii) mudanças aplicadas no ambiente de execução por um teste, como, a definição de uma nova variável de ambiente que será usada por outro teste, (iv) login realizado em algum sistema para que outro teste possa ser executado (loginAccess), (v) atribuição de alguma permissão concedida para que um teste possa ter sucesso em sua execução (permissionGranted), (vi) comunicação bem sucedida com algum artefato (communicationIsOk), e (vi) disponibilidade de uso de algum artefato (artifactIsAvailable). Já a opção *executionSuite* representa as dependências de um *OrderedSuite* em relação aos testes que ele executa em alguma ordem específica.

5.10. Possíveis Perguntas para o Modelo Conceitual

A partir das propriedades e relacionamentos das entidades apresentadas, diversas perguntas podem ser realizadas ao modelo conceitual. A seguir, são apresentados alguns exemplos de possíveis perguntas.

1. Quantos casos de teste compõe algum teste?
2. Quais testes são executados por alguma suíte específica?
3. Quantos suítes executam testes funcionais?

4. Há testes atualizados para a versão “v7_00” do sistema em teste (SUT)?
5. Quantos testes são responsáveis por validar algum artefato do sistema em teste?
6. Quais testes fazem parte da classificação de “testes novos”?
7. Quais são os testes que ainda devem ser atualizados para a versão “v7_00” do sistema em teste e que sejam unitários e automáticos?
8. Quais são os artefatos que são validados por casos de teste funcionais?
9. Quais testes possuem casos de teste obrigatórios e de desempenho?
10. Quais testes possuem casos de teste de desempenho ou de usabilidade?
11. Quais critérios de seleção foram definidos no sistema em teste?
12. Quais dados de entrada e saída são usados por cada teste?
13. Quais features estão sendo testadas no SUT?

5.11. Considerações Finais

Neste capítulo foi apresentado um modelo conceitual responsável por ilustrar a relação das informações de teste mencionadas no capitulo 3 e assim ajudar na atividade de gerência dos testes e na geração dos arquivos XML usados pelo framework JAAF+T. A partir desse modelo, pôde-se perceber que perguntas úteis para a coordenação dos testes podem ser realizadas, como, por exemplo, quais features são testadas em um SUT, quais testes estão atualizados para alguma versão desejada, etc. Na Seção 5.10 são apresentados exemplos dessas possíveis perguntas.

No próximo capítulo é apresentada uma nova linguagem de modelagem que usa como base o modelo conceitual proposto. O foco central da linguagem é permitir a modelagem de informações que ajudem na coordenação de testes de software em diferentes domínios de aplicação.