9
Trabalhos Relacionados

Neste capítulo são apresentados alguns trabalhos relacionados. Inicialmente são descritos trabalhos voltados a autoadaptação e autoteste em sistemas de software. Em seguida, são apresentados trabalhos referentes a linguagens e ferramentas de gerenciamento de testes de software.

9.1.
Trabalhos Voltados a Autoadaptação e Autoteste

O principal foco do trabalho apresentado em (Wen et al., 2005) é prover um framework de autoteste que deve ser usado ao testar os chips de microprocessadores. O framework consiste de duas etapas principais: (i) decidir quando e quais testes executar para avaliar o(s) núcleo(s) do processador de um chip, e (ii) testar e responder análises realizadas em outros componentes do chip, usando como base o(s) núcleo(s) do processador testado(s) como um gerador padrão e analisador de resposta. Já que os componentes do framework são amarrados aos componentes relacionados aos microprocessadores, não é possível usá-lo em diferentes domínios, assim como o JAAF+T.

Em (Denaro et al., 2007) os autores apresentam uma abordagem autoadaptativa que usa um control-loop estruturado da seguinte maneira: mecanismos de monitoramento, mecanismos de diagnóstico, e estratégias de adaptação. A abordagem autoadaptativa consiste de um pré-processamento e uma etapa de geração. O pré-processamento é composto da: identificação de possíveis problemas de integração, geração de casos de teste para revelar problemas de integração, e a definição de adequadas ações de recuperação. Mesmo considerando tal estrutura, a abordagem não considera importantes etapas definidas no JAAF+T, como, por exemplo, a possibilidade de definir dados de entrada e assertivas de saída a serem usados em diferentes testes, assim como definir a ordem de execução dos testes responsáveis para validar artefatos do sistema em teste e critérios de seleção para executar testes. Em adição, (Denaro et al., 2007) propõe um processo fixo de autoteste,

diferentemente do JAAF+T, já que diversos control loops compostos por diferentes atividades podem ser criados.

O trabalho (Stevens et al., 2007) propõe um framework para testar sistemas autoadaptativos. Tal trabalho introduz o conceito de um container autonômico considerado uma estrutura de dados que possui autogerenciamento de *capabilities*, e que também possui capacidade de realizar autoteste. Tal abordagem usa uma estratégia que copia testes de recursos gerenciados enquanto que o recurso correntemente gerenciado está sendo usado pelo sistema. Essa abordagem é uma técnica conhecida como replicação com validação. No entanto, esse trabalho não permite usar as atividades de autoteste em outras arquiteturas, já que o framework é amarrado à arquitetura de computação autonômica da IBM (IBM, 2003). Assim, não é possível usar as atividades de autoteste em diferentes processos de autoadaptação. Além disso, o framework não ajuda na criação de casos de teste que possam utilizar dados de diferentes bases de dados. Outra característica apresentada pelo JAAF+T, mas não incluída nos trabalhos mencionados, é a integração com conhecidas APIs de teste, como, JAT, DBUnit e JUnit.

Além desses trabalhos mencionados, diversas outras abordagens conhecidas e relacionadas à autoadaptação e que não realizam autoteste, foram analisadas, como, (Dobson et al., 2006), Rainbow (Garlan et al., 2004), Kinesthetics extreme (Kaiser et al., 2003) e The Agent Building and Learning Environment (ABLE) (Bigus et al., 2001). Percebeu-se que esses trabalhos definiram um processo fixo de autoadaptação, impedindo que outros control-loops pudessem ser aplicados assim como outras atividades. JAAF+T possui a ideia de permitir a flexibilidade na criação desses processos de autoadaptações. Assim, o framework pode ser usado para definir control-loops sem a ideia de autoteste. No entanto, o ponto principal do framework é a possibilidade de criar agentes de software capazes de realizar autotestes.

9.2. Linguagens e Ferramentas de Gerenciamento de Teste

A linguagem de teste, AGEDIS modeling language - AML (Trost e Cavarra, 2012), baseia-se no metamodelo da UML 1.4 e permite a especificação de testes para os aspectos estruturais (estático) e comportamentais (dinâmica) dos modelos computacionais UML. AML surge como parte da metodologia AGEDIS (AGEDIS, 2012) e foi concebido com dois objetivos principais em mente: criar

uma abstração de teste adequado ao SUT que será analisado pelas ferramentas AGEDIS, o que permite gerar automaticamente suítes de teste, além de definir um conjunto de diretrizes significantes para o processo de teste. Apesar da abordagem propor a modelagem de interessantes conceitos de teste, ela não provê a modelagem de importantes informações úteis para a gerência dos testes, como, por exemplo, a identificação de (i) qual versão do sistema em teste 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, e (vii) quais critérios de seleção de testes algum SUT possui.

Já o Testing and Test Control Notation (TTCN-3) é uma linguagem modular parecida com uma linguagem típica de programação. Essa linguagem é considerada um padrão para o desenvolvimento de testes na área de telecomunicações e comunicação de dados. A principal razão disso é que ela compreende conceitos adequados a diversos tipos de testes distribuídos em sistemas, como, por exemplo, importantes features necessárias para especificar testes funcionais, conformidade, interoperabilidade, carga e escalabilidade. Além disso, essa linguagem possui mecanismos que permitem comparar resultados esperados com respostas providas pelo SUT, controlar comunicação síncrona e assíncrona, assim como definir temporizadores e monitoramento de dados. Todos as informações relacionadas a testes de software não considerados pela AML também não são contemplados nesse trabalho.

De acordo com (UTML, 2012) os benefícios da “engenharia dirigida a modelos” (MDE) para o desenvolvimento de produtos de software tem sido demonstrado em numerosas ocasiões. Consequentemente, tais benefícios podem também ser alcançados em MDE para o desenvolvimento de testes de software. Dessa forma, o paradigma dos testes baseados em modelos é chamado de “engenharia de testes dirigida a modelos” ou simplesmente “testes dirigidos a modelos” (MDT – Model Driven Test). No entanto, para otimizar a eficiência de MDT, boas práticas e específicos padrões para desenvolvimento de teste tem que ser considerados e adotados. Baseado nessa ideia, (Feudjio, 2012) propôs uma notação projetada para MDT orientada a padrões chamada de Unified Test Modeling Language (UTML). Ela provê os meios para projetar todos os aspectos de um sistema de teste em um alto nível de abstração e que sejam independentes de qualquer infraestrutura específica de nível mais baixo de teste.

Além disso, essa abordagem também provê uma orientação para seguir padrões de projeto de teste e evitar problemas usuais de MDT. Tal abordagem oferece uma ferramenta chamada de MDTester, que permite modelar os conceitos propostos pela UTML. No entanto, essa ferramenta e a linguagem considerada não representam as informações de teste apresentadas no capítulo 3.

Rational Test Manager - RTM (RTM e RMT, 2012) propõe uma ferramenta central para execução, gerenciamento de atividades e relatórios de testes. Ela fornece interessantes pontos de vista que não são fornecidos por outras abordagens, sendo assim escolhida para ser utilizada em diversos projetos de teste do LES. Um exemplo é a possibilidade de agrupar conceitualmente casos de teste, assim como visualizar quais suites estão disponíveis em um SUT, além de quais casos de teste cada suíte executa. No entanto, RTM não fornece informações importantes, tais como: (i) dependências entre testes, (ii) identificação de quais os testes são obrigatórios e opcionais, (iii), quais testes são automáticos e manuais, (iv) tipos de teste considerados, e (v), se os testes estão atualizados para alguma release do sistema em teste.

Rational Quality Manager - RQM (RQM, 2012), sucessor do RTM, também é uma ferramenta da IBM voltada para gerenciamento de atividades, execução e relatórios de teste. No entanto, RQM é uma aplicação web (diferentemente do RTM), que além dos recursos providos pela RTM, também permite a rastreabilidade de requisitos e bugs com os testes criados. Além disso, caso alguma informação não esteja presente para documentar os testes (ou tenha alguma informação em excesso), qualquer formulário da ferramenta pode ser configurado para atender a necessidade da equipe. Essa liberdade tem motivado equipes de teste a migrar para essa ferramenta e verificar como seria o trabalho com modelos criados no RSA, já que a modelagem UML traz uma visão diferente dos formulários providos pelo RQM.

Além dos trabalhos mencionados, outras abordagens responsáveis por modelar conceitos de teste a partir da UML e realizar a geração de informações relacionadas a teste (ex: casos de de teste, dados, suites, etc.) foram analisadas: (Offutt e Abdurazik, 2000), (Abdurazik e Offutt , 2012), (Hartmann et al., 2000), (Kim, 1999), (Toth et al, 2003), (Riebish et al, 1994), (Bertolino et al., 2005) e (Neto, Subramanyan et al., 2007). Pôde-se avaliar que os diagramas UML mais utilizados para realizar geração de testes são os seguintes: diagrama de classes, estados, colaboração e atividades.

A Tabela 11 apresenta parte das informações de teste consideradas pela UTP-C, proposta no capítulo 6, a fim de apresentar quais delas são manipuladas pelas abordagens mencionadas nesta seção.

Informações Identificadas	**UTP**	**TTCN-3**	**AML**	**UTML**	**RTM**	**RQM**
Ferramenta necessária para executar algum teste.	-	Repre-sentado	-	-	-	Repre-sentado
Casos de teste obrigatórios ou opcionais.	-	-	-	-	-	-
Testes automatizados ou manuais.	-	-	-	-	-	Repre-sentado
Dependências entre testes	Repre-senta, exceto a semânti-ca das depen-dências	Repre-senta, exceto a semânti-ca das depen-dências	Repre-senta, exceto a semânti-ca das depen-dências	-	-	-
Representar classificações de teste.	-	-	-	-	-	-
Suites criadas para um SUT	Repre-sentado	-	Repre-sentado	-	Repre-sentado	Repre-sentado
Níveis de teste	-	-	-	-	-	-
Tipos de teste	-	-	-	-	-	Repre-sentado
Risco de Produto	-	-	-	-	-	-
Critérios de Seleção para executar testes	-	-	-	-	-	-

Tabela 11. Análise de quais trabalhos relacionados estão considerando informações de teste manipuladas pela UTP-C.