# 1
# Introdução

A criação e execução de testes de software é uma atividade extremamente crítica no desenvolvimento de sistemas. Dependendo do tamanho e complexidade do sistema avaliado (*System Under Test - SUT*), uma grande quantidade de teste deve ser criada e mantida (Burnstein et al., 2012; Burnstein e Miller, 2012; TMMi, 2012). O Instituto Nacional de Padrões e Tecnologias dos EUA (*The National Institute of Standards and Technology*) afirma que sistemas sem testes de software adequados geram despesas de até $59,5 bilhões de dólares a cada ano. Esse valor equivale a quase 1% do produto interno bruto dos EUA (NIST, 2002).

Para que haja o acompanhamento apropriado dos testes, torna-se necessária a aplicação de um processo de gerência (Black, 2009) que permita executar tais testes de maneira controlada com o objetivo de avaliar se o mesmo se comporta conforme o especificado. Mesmo parecendo algo simples, há muitas dificuldades neste processo, como alto custo para empresa pela necessidade de profissionais especializados, ou mesmo o desconhecimento tanto para realizar os procedimentos, técnicos ou mesmo de planejamento.

Atualmente diversas ferramentas oferecidas ajudam na coordenação da execução dos testes, como, por exemplo, ferramentas de integração contínua (Duvall et al., 2007) e de gerenciamento de testes (Black, 2009). No entanto, para executar tais testes há a necessidade que uma pessoa solicite de forma explícita a execução deles, ou ao menos informe previamente um horário que tal execução deve ser iniciada, como, por exemplo, todos os dias às 23h00min, quando houver mudanças no repositório do projeto correspondente, etc. No entanto, quando sistemas/componentes precisam ter autonomia para definir quando e quais testes devem ser executados para ajudar na tomada de decisões em tempo de execução, o uso do paradigma de agentes de software (Wooldridge e Jennings, 1998) torna-se apropriado. A principal razão para seu uso são as propriedades de autonomia e pró-atividade que compõem os agentes. Autonomia caracteriza que um agente deve ser capaz de operar sem intervenção externa (de um usuário), e ter algum tipo de controle sobre suas ações e seu estado interno. Já pró-atividade representa que o agente não

responde apenas aos estímulos de outros agentes e do seu ambiente, devendo tomar iniciativa, quando necessário, para que seus objetivos sejam alcançados.

Esse contexto está fortemente relacionado à linha de pesquisa em que agentes procuram realizar autoadaptações confiáveis e seguras para que seus objetivos possam ser alcançados. Uma das soluções adotadas por diversos trabalhos (Wen et al., 2005; Denaro et al., 2007; Stevens et al., 2007; Martins et al., 2001) é a aplicação da técnica de autoteste, que permite aos agentes testarem a si mesmos em tempo de execução.

Para ajudar no entendimento da execução, assim como na criação e manutenção dos testes, as informações usadas pelos agentes devem ser documentadas. Documentar testes de software é uma das atividades essenciais no processo de gerência teste, assim como mencionado pelos modelos de maturidade Test Maturity Model - TMM (Burnstein et al., 2012; Burnstein e Miller, 2012) e Test Maturity Model Integration - TMMi (TMMi, 2012).

Uma das abordagens que tem ganhado importância para documentar e ajudar nas atividades de criação, execução e manutenção dos testes é o uso de linguagens de modelagem, já que oferecem uma visão gráfica que facilita a abstração de conceitos e a comunicação com os envolvidos do projeto em questão. Na literatura há diversas abordagens relacionadas à modelagem de teste, como, por exemplo, UML Testing Profile (UTP, 2012), Agedis Modeling Language (AGEDIS, 2012) e Unified Testing Modeling Language (UTML, 2012).



## 1.1. Motivação

Há seis anos o Laboratório de Engenharia de Software (LES) na PUC-Rio começou a trabalhar de forma intensiva na criação e gerência de testes para sistemas de diferentes domínios, como, por exemplo, sistemas relacionados ao domínio de deslizamento de terra, controle de estoque e suprimento de petróleo e derivados do petróleo (ex: gasolina, querosene, etc) no Brasil, etc.

Como alguns desses sistemas eram compostos por agentes autoadaptáveis (Costa et al., 2010a; Costa et al., 2011a), capazes de adaptar seus comportamentos em tempo de execução a fim de atender solicitações de usuários, analisamos como essas adaptações poderiam ser realizadas de forma mais confiável a partir de trabalhos oferecidos na literatura que aplicam a técnica de autoteste (Denaro et al., 2007; Stevens et al., 2007; Wen et al., 2005). A partir dessa análise, percebemos que tais trabalhos foram desenvolvidos de forma *ad*

*hoc*, impedindo seu reuso em diferentes domínios de aplicação. Devido à ausência de uma abordagem genérica, decidimos investigar quais informações poderiam ser usadas pelos agentes para coordenar a execução dos seus testes em diferentes domínios, já que autoadaptações precisam ser validadas antes que sejam efetivamente executadas e configuradas no agente correspondente. A partir dessa investigação, uma abordagem genérica poderia ser oferecida para ajudar na criação de agentes autoadaptativos capazes de realizar autotestes.

Para ajudar no entendimento da coordenação, criação e manutenção dos testes, documentações tornam-se necessárias. Tendo essa preocupação em mente, somada à demanda pela documentação de testes a partir da criação de modelos (usando, por exemplo, UML) (UML, 2012), investigamos como documentar as informações usadas pelos agentes para ajudar na automatização do gerenciamento dos testes a partir do uso de uma nova linguagem de modelagem, ajustada a esta situação.

Visando exemplificar uma situação com autoteste, considere que um agente comprador precisa se autoadaptar para concluir uma compra de um produto com um agente vendedor. Cada comprador do sistema possui um vendedor preferencial para compras, e em um dado momento uma compra solicitada não pôde ser realizada com tal vendedor. Visando contornar essa situação, o agente se autoadapta para encontrar uma forma de concluir a compra. Para que isso seja possível, inicialmente há a necessidade de investigar a razão do insucesso, para que assim o comprador possa decidir se: (i) irá mudar seu critério de compra (ex: gastar no máximo R$100,00, aceitar itens usados para a compra, etc.) ou (ii) escolherá outro vendedor que consiga lhe vender o produto desejado. Caso o comprador decida solicitar a compra para outro agente, os vendedores disponíveis deverão ser analisados (uso de autoteste), já que o sistema em questão é composto por agentes heterogêneos. Diferentes testes podem ser aplicados para ajudar o comprador nessa decisão, como, por exemplo, analisar se o vendedor tem o produto solicitado e atende seu critério de compra, se o vendedor possui boa reputação de venda em relação à sociedade de agentes do sistema, etc. Ao escolher o vendedor, o agente comprador reconfigura seu comportamento a fim de concluir sua autoadaptação. Após tal configuração, o comprador efetiva a solicitação de compra, considerando boas chances de ser atendida, já que testes foram executados para garantir uma autoadaptação confiável e segura.

## 1.2. O Problema

Devido à carência de abordagens que permitam a criação de agentes autoadaptativos capazes de realizar autoteste em diferentes domínios, é necessária a descoberta de um mecanismo que permita a criação de diferentes agentes autoadaptáveis capazes de realizar autotestes a partir de informações geralmente consideradas úteis para a coordenação dos testes, assim como proposto em (Burnstein e Miller, 2012; TMMi, 2012; 829-2008, 2012; Graham et al., 2008; Black, 2008).

Para a identificação dessas informações, é necessária uma abordagem baseada em modelos voltada para modelagem de informações úteis para coordenação dos testes. Portanto, para usufruir os benefícios desta abordagem e ajudar na automatização do gerenciamento dos testes, são necessárias ferramentas que realizem a geração automática de artefatos a partir dos modelos, como, por exemplo, artefatos usados por agentes autoadaptáveis para controlar a execução dos seus testes.

## 1.3. Principais Contribuições

A seguir, são apresentadas as principais contribuições da tese.

- Framework JAAF+T (Java self-Adaptive Agent Framework for self-Test) voltado para a criação de agentes autoadaptativos capazes de realizar autotestes.
- Modelo conceitual apresentando o relacionamento entre as informações utilizadas pelos agentes JAAF+T para coordenar a execução dos testes.
- Nova linguagem de modelagem baseada no modelo conceitual proposto para ajudar na coordenação dos testes, chamada UTP-C (UML Testing Profile for Coordination).
- Estratégia proposta para realizar a geração automática de artefatos de teste a partir de modelos baseados na UTP-C.

## 1.4. Organização do Documento

A tese está estruturada da seguinte forma:

- No capítulo 2 é apresentada a fundamentação teórica da tese. Nela são mencionados os principais conceitos utilizados e mencionados no documento.
- No capítulo 3 são apresentadas informações que agentes de software devem utilizar para ajudar a automatizar a coordenação da execução dos testes.
- No capítulo 4 é apresentado o framework JAAF+T (Java self-Adaptive Agent Framework for self-Test).
- No capítulo 5 é apresentado o modelo conceitual de teste baseado nas informações identificadas no capítulo 3 e utilizadas pelo JAAF+T para a coordenação dos testes.
- No capítulo 6 é apresentada a nova abordagem de modelagem de teste chamada UTP-C (UML Testing Profile for Coordination), criada a partir do modelo conceitual apresentado no capítulo 5.
- No capítulo 7 é apresentado um estudo de caso que ilustra o uso da UTP-C com o JAAF+T.
- No capítulo 8 são apresentadas duas ferramentas desenvolvidas responsáveis por realizar transformações de diagramas UTP-C para artefatos úteis para equipes de teste, como, por exemplo, artefatos usados pelo JAAF+T.
- No capítulo 9 são apresentados os trabalhos relacionados.
- No capítulo 10 são apresentadas as conclusões e os trabalhos futuros.