Documento do Projeto
Java Teste de Unidade Funcional

Versão 1.0

São Carlos
2012

Histórico das Revisões

VERSÃO	DATA	AUTOR	DESCRIÇÃO
Versão 0.2	11/05	Alinne Livia	Versão Inicial da documentação
Versão 0.4	25/05	Alinne Faimison Livia Rafael Venilton	Definição dos Requisitos
Versão 0.6	08/06	Alinne Carlos José Dário Vinicius	Elaboração dos Diagramas
Versão 0.8	22/06	Alinne Livia	Finalização da documentação
Versão 1.0	02/06	Alinne Livia	Revisão geral da documentação

Índice

1. Introdução

1.1. Propósito

Este documento especifica os requisitos do framework Java Teste de Unidade Funcional – JTUF, fornecendo aos desenvolvedores e/ou testadores as informações necessárias para o projeto e implementação, assim como para a realização dos testes e homologação do sistema.

1.2. Público Alvo

Este documento se destina aos engenheiros de software, desenvolvedores e testadores.

1.3. Visão Geral do Framework

O Java Teste de Unidade Funcional - JTUF é um framework *open-source*, com suporte à execução de testes funcionais na linguagem de programação Java. Esse framework auxilia na execução de testes de unidade funcional sobre classes Java. Com JTUF pode ser verificado se cada método de uma classe funciona de acordo com a especificação, exibindo os possíveis erros que estão ocorrendo no método a partir da geração de casos de teste.

1.4. Visão Geral do Documento

- **Na seção** Erro! Fonte de referência não encontrada. são enumerados todos os requisitos funcionais e não-funcionais do framework.
- **Na seção** Erro! Fonte de referência não encontrada. são apresentados os diagramas do framework.

2. Requisitos

Os requisitos funcionais e não funcionais do framework JTUF serão detalhados a seguir.

2.1. Requisitos Funcionais:

ID	REQUISITO	DESCRIÇÃO	CASO DE USO	PRIORIDADE
[RF01]	Definir as pré-condições e pós-condições referente ao método a ser testado	Permitir a inclusão das pré-condições (correspondem às classes de equivalência) e pós-condições para cada método a ser testado por meio do COFOJA.	Definir pré-condição e pós-condição	Essencial
[RF02]	Definir invariantes de uma classe	Permitir a inclusão de invariantes referente a objetos de uma determinada classe.	Definir invariantes	Essencial
[RF03]	Definir as *annotations*	Permitir a inclusão de *annotations* para cada variável recebida como parâmetro de uma função.	Definir *annotations*	Essencial
[RF04]	Selecionar método a ser testado	Selecionar o método a ser testado de uma respectiva classe.	Selecionar método	Importante
[RF05]	Selecionar os nós da árvore	Selecionar um nó da árvore[1], sendo ele primitivo poderá ser adicionado um valor ou deixa-lo *default* (*null*).	Especificar nós da árvore	Importante
[RF06]	Gerar casos de teste	Gerar e executar casos de teste, onde primeiramente devem ser gerados os valores dos parâmetros pertencentes ao método em teste. Estes valores devem ser gerados manualmente ou automaticamente (de forma aleatória).	Gerar e executar casos de teste	Essencial
[RF07]	Detalhar os casos de teste	Permitir o detalhamento dos valores dos parâmetros.	Detalhar os casos de teste	Importante
[RF08]	Gerar relatório dos casos de teste	Gerar relatório para todos os casos de teste executados, onde será exibida uma lista detalhada com todas as execuções dos casos de teste, apresentando os casos de sucesso e os casos de falha.	Gerar relatório dos casos de teste	Essencial
[RF09]	Gerar tabela de decisão	Gerar a tabela de decisão a partir das relações entre causas e efeitos especificadas em um arquivo.	Gerar tabela de decisão	Essencial
[RF10]	Gerar dados de teste	Gerar dados de teste a partir das relações entre causa e efeitos definidas na tabela decisão.	Gerar dados de teste	Essencial
[RF11]	Gerar relatório dos dados de teste	Gerar relatório para todos os relacionamentos entre causas e efeitos da tabela de decisão.	Gerar relatório da tabela de decisão	Essencial

[1] A raiz de uma árvore é o método a ser testado e os nós da raiz são os respectivos parâmetros desse método. Já para um parâmetro que não é primitivo (um objeto), ao ser clicado, serão construídos nós filhos desse parâmetro na árvore, que correspondem aos respectivos parâmetros declarados no construtor desse objeto.

2.2. Requisitos Não Funcionais:

Os requisitos não funcionais estão divididos em Requisitos de Performance, Portabilidade, Confiabilidade e Usabilidade.

2.2.1. Performance:

ID	DESCRIÇÃO
[RNF01]	O tempo para a geração do relatório deve ser no máximo de 5 segundos.
[RNF02]	O tempo para a geração da tabela de decisão deve ser no máximo de 5 segundos.
[RFN03]	O tempo para a geração de dados de testes deve ser no máximo de 5 segundos.

2.2.2. Portabilidade:

ID	DESCRIÇÃO
[RNF04]	O desenvolvimento do framework deve ser realizado utilizando a linguagem de programação Java.
[RNF05]	A plataforma para desenvolvimento do framework que deve ser utilizada é o Eclipse versão Indigo 3.7.2 ou superior.
[RNF06]	O framework deve funcionar em qualquer sistema operacional.
[RNF07]	O framework deve funcionar em qualquer IDE (*Integrated Development Environment*).
[RFN08]	Para auxiliar o desenvolvimento do framework deve ser utilizado *Reflection*, COFOJA, Contratos e *Annotations*.

2.2.3. Confiabilidade:

ID	DESCRIÇÃO
[RNF09]	O framework deve ter alta disponibilidade, ou seja, deve estar disponível 99% do tempo.
[RNF10]	Caso o framework apresente algum tipo de falha relacionada às suas funcionalidades, as mesmas deverão ser corrigidas em até 24 horas.
[RNF11]	A execução dos casos de teste deve ser feita por meio do framework para obter as saídas necessárias e validá-las com as pós-condições.

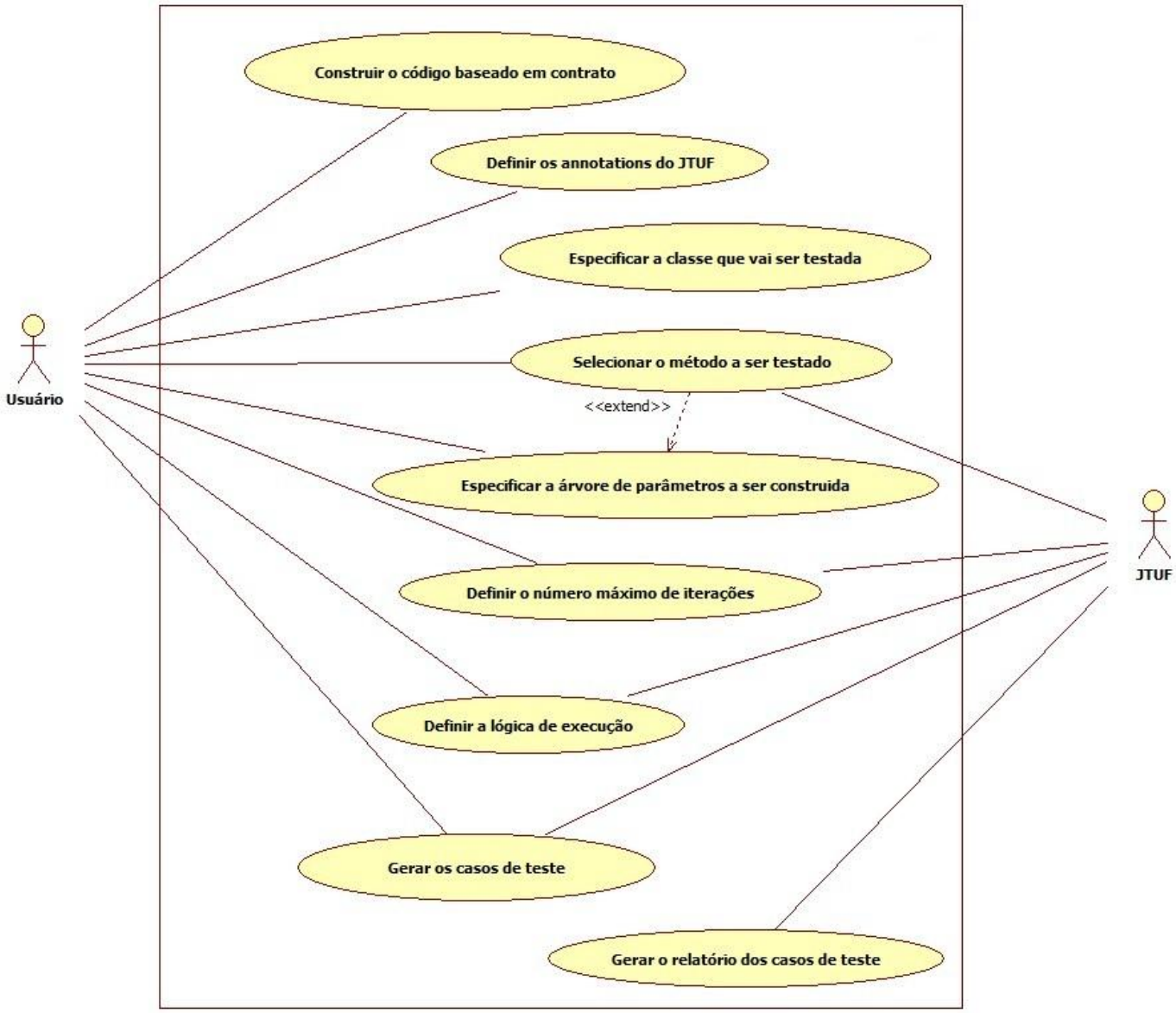
2.2.4. Usabilidade:

ID	DESCRIÇÃO
[RNF12]	O framework deve permitir a visualização de todos os métodos a serem testados pertencentes a uma determinada classe, com os seus respectivos atributos e parâmetros.
[RNF13]	O framework deve permitir a seleção de um único método por vez para ser testado.
[RNF14]	O framework deve permitir ao usuário visualizar e alterar os parâmetros desejáveis dos seus respectivos métodos, os quais devem estar representados no formato de uma árvore.
[RNF15]	O framework deve permitir a geração dos casos de teste por meio de iterações ou tempo (segundos e/ou minutos).
[RFN16]	O framework deve permitir a seleção de um único relacionamento entre causas e efeitos por vez para ser gerado os dados de teste.
[RFN17]	O framework deve permitir a exclusão de uma única linha por vez da tabela de decisão.

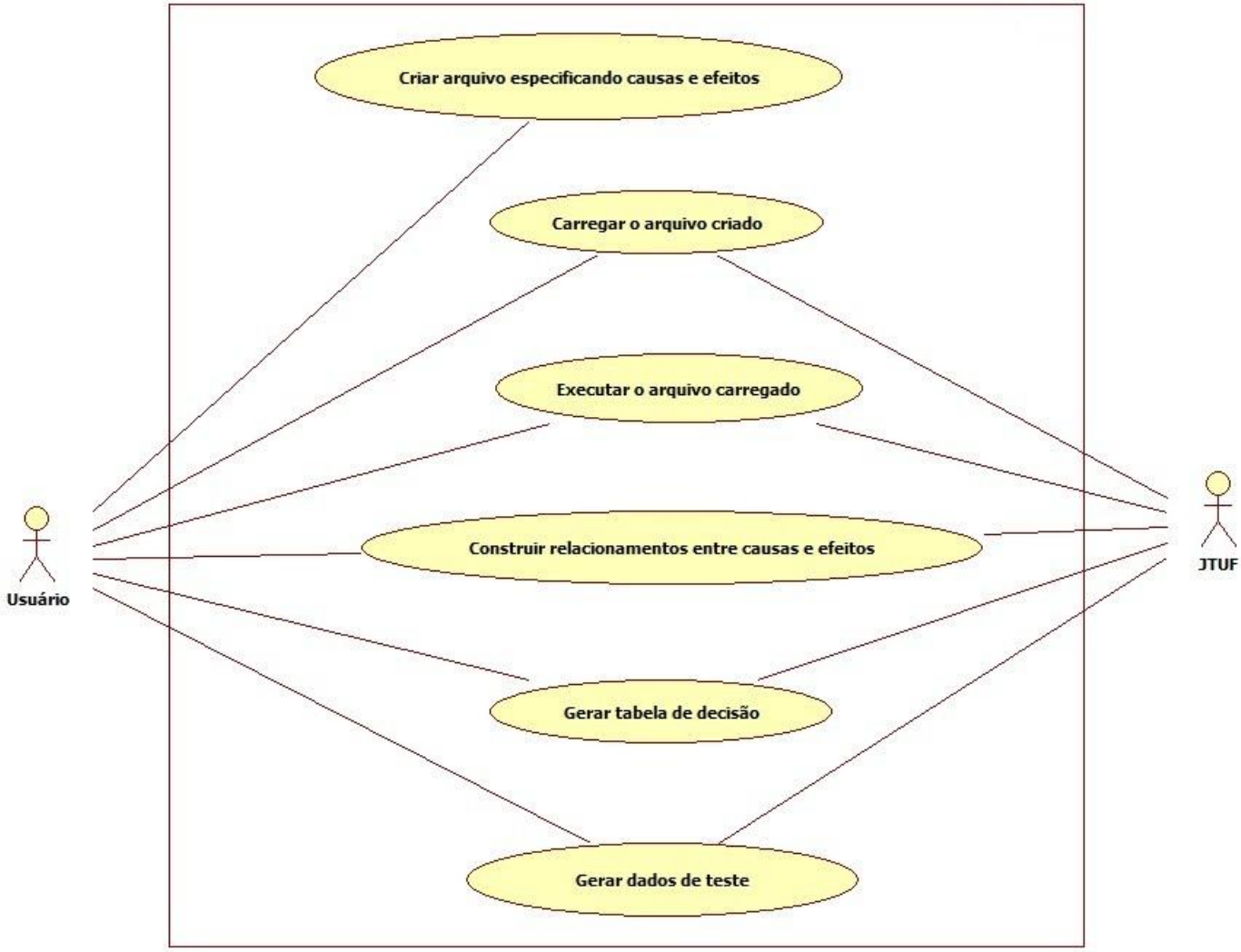
3. Diagramas

3.1. Caso de Uso

3.1.1. Geração de Classe de Equivalência e Valor limite

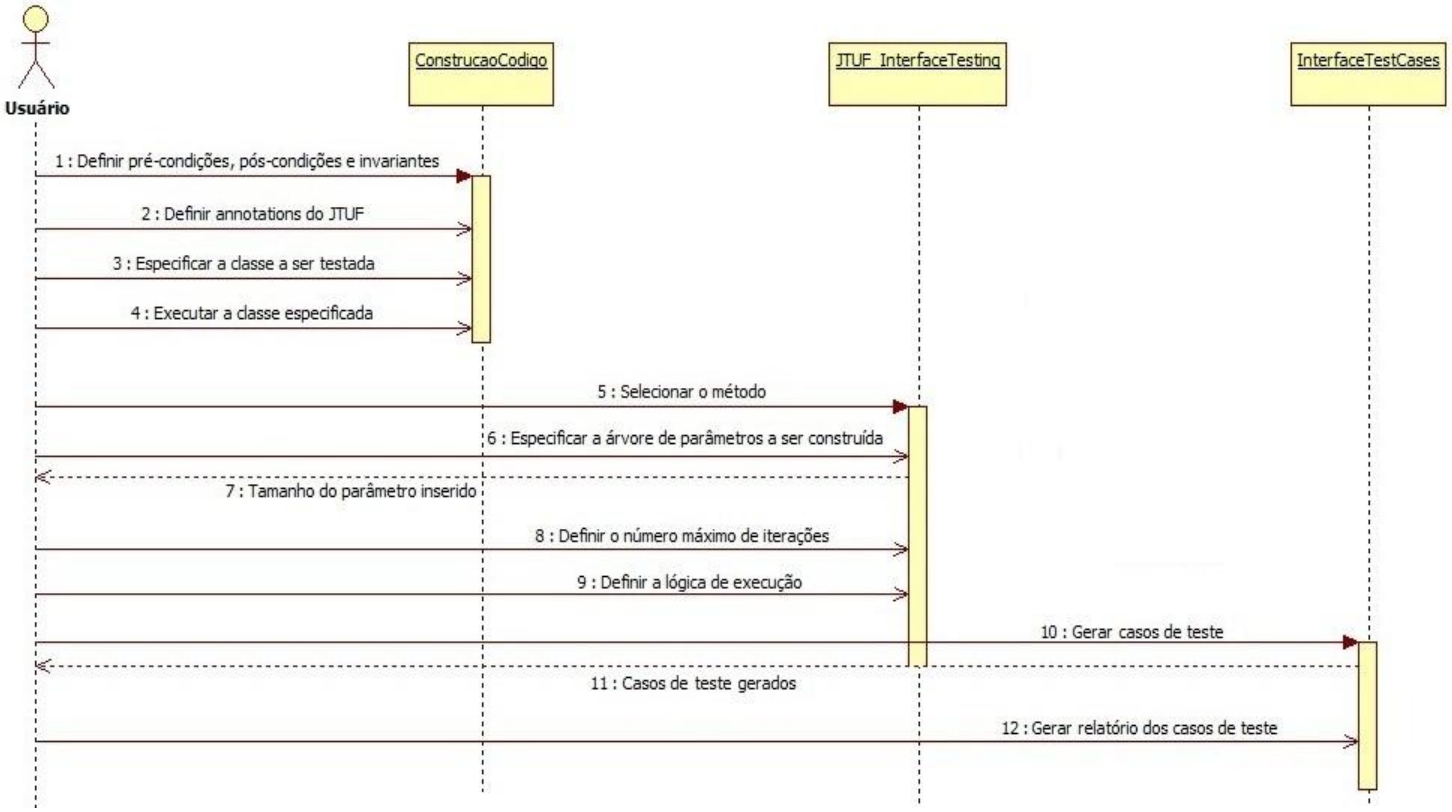


3.1.2. Geração dos Dados de Teste

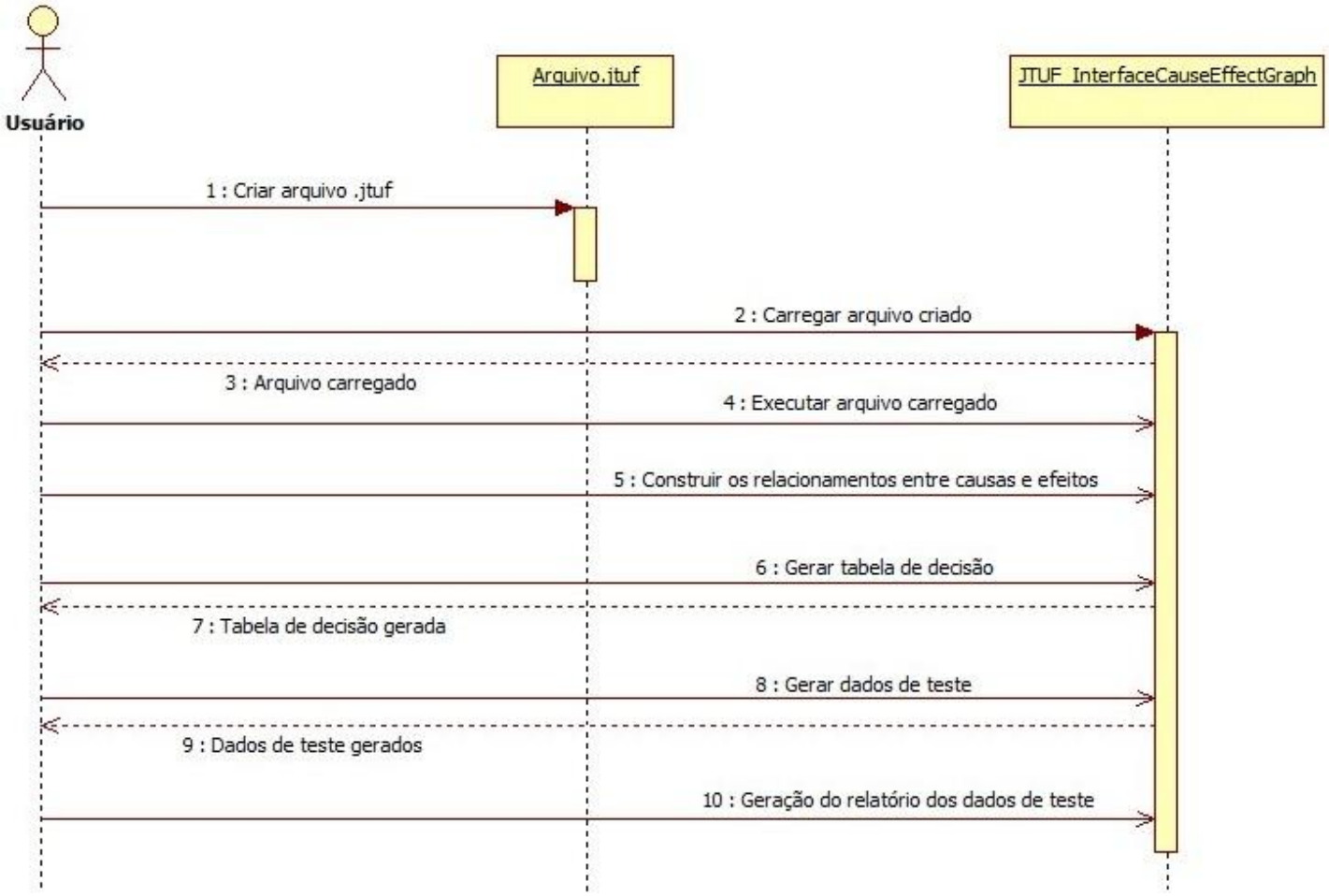


3.2. Diagrama de Sequência

3.2.1. Geração de Classe de Equivalência e Valor Limite

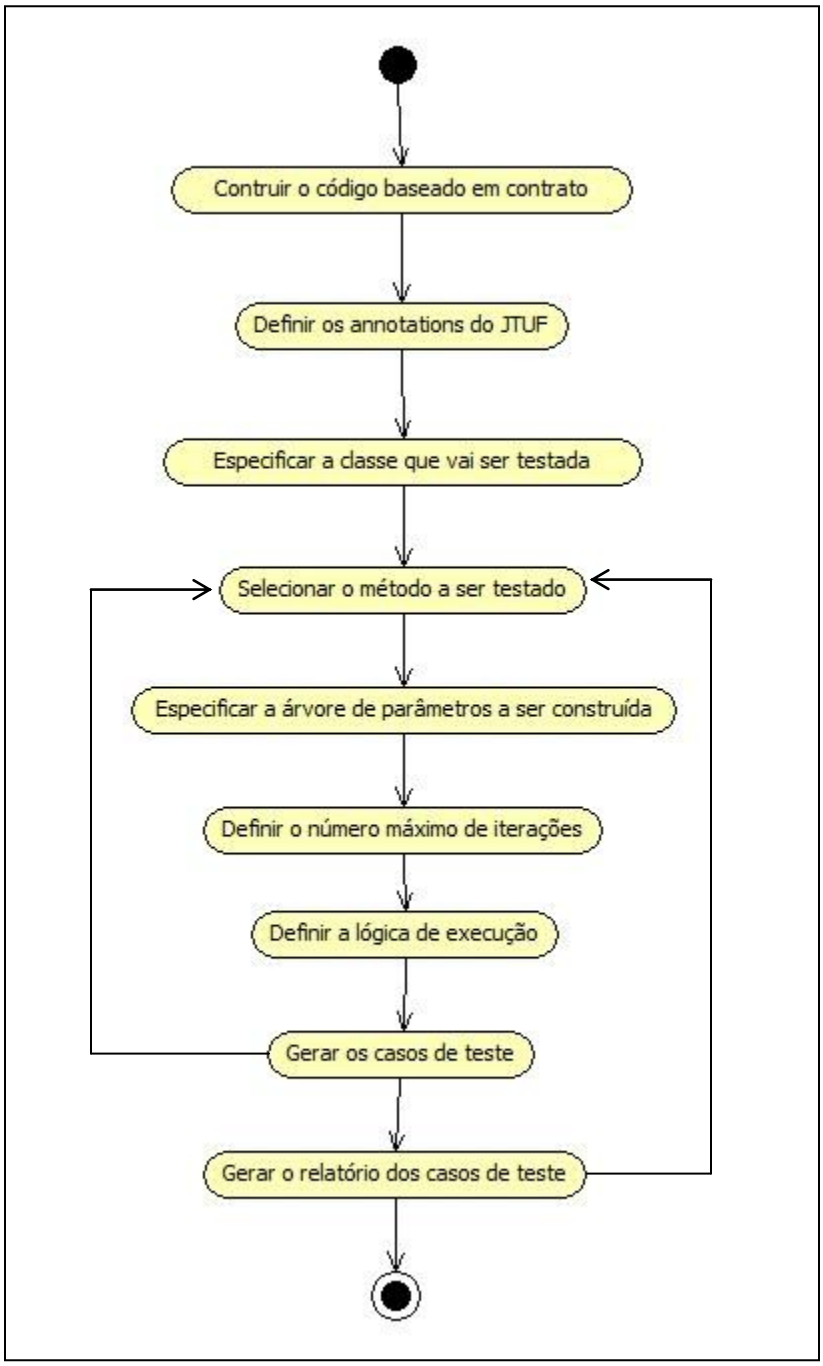


3.2.2. Geração dos Dados de Teste



3.3. Diagrama de Atividades

3.3.1. Geração de Classe de Equivalência e Valor Limite



3.3.2. Geração dos Dados de Teste