CURSO DE CIÊNCIA DA COMPUTAÇÃO – TCC	
(X) PRÉ-PROJETO ( ) PROJETO	ANO/SEMESTRE: 2020/2

FERRAMENTA PARA TESTES AUTOMATIZADOS EM SOFTWARE LEGADO EM JAVA

Jonathan Luiz de Lara

Prof. Everaldo Artur Grahl - Orientador

1 INTRODUÇÃO

Um dos grandes problemas em software legado é a dificuldade de compreensão das regras de negócio implementadas. Esta constatação pode ser vista pela experiência deste autor ao longo da experiência com desenvolvimento e testes de software. Outros problemas que dificultam a realização de testes em sistemas legados incluem desconhecimento das razões que levaram a determinadas decisões, problemas na estruturação do código, miscelânea de estruturas e estilos de programação e ausência de modularidade Código ~~legado~~ Legado e?~~é~~ um código sem testes automatizados (FEATHERS, 2004). Ou seja, a produção de software legado está muito associada ~~a~~a um problema de engenharia, falta da utilização de boas práticas que revertem tais características. Como bem visto nas características, esse tipo de software não foi construído para atender requisitos de automação de testes, visto a falta de separação em seus artefatos. O custo de manutenção de um sistema legado tem crescido de 40% nos anos 70 para o patamar de 90% atualmente (PIGOSKI, 1997).

Softwares legados trazem uma série de riscos à empresas que o mantém, incluindo grandes níveis de falta de estabilidade, visto que qualquer mudança em sua estrutura ou em seu comportamento podem trazer efeitos colaterais, como regressão de funcionalidades em razão do software não ser projetado para manutenção e por consequência impedindo de ser testado por outro software e conseguir de forma ágil descobrir o que falhou na mudança, impedindo que o erro seja entregue juntamente com uma nova funcionalidade.

Esse cenário traz uma oportunidade de melhoria, sabendo-se que a entrega de software com características de legado não está associada somente a tecnologia, mas sim a falta do emprego de conhecimento de engenharia, seja na definição estrutural e/ou na implementação de regras e controles. Essas definições acabam sendo feitas sem qualidade e impedindo que a evolução do software ocorra de forma sustentável, trazendo um ambiente cada vez mais complexo~~s~~ para os profissionais envolvidos e para a organização se manter no mercado evoluindo seu negócio.

O trabalho consiste em oferecer ao profissional que está desenvolvendo o software alguns recursos para realizar testes, informando por exemplo dados de entrada e de saída que devem ser produzidos pela funcionalidade (testes funcionais). Com estas definições de análise feitas pelo profissional, o software será capaz de aferir se para aquele conjunto de valores de entrada a funcionalidade está tendo o resultado que se espera.

1.1 OBJETIVOS

O objetivo principal deste trabalho é desenvolver uma ferramenta para a construção de testes automatizados para software legado em JAVA.

Os objetivos específicos são:

a) desenvolver uma ferramenta para facilitar a realização de testes em funcionalidades (módulos, classes, métodos);

b) apresentar ao desenvolvedor os resultados dos testes, sinalizando se a funcionalidade produziu o resultado esperado ou não;

c) permitir que funcionalidades que não foram projetadas para serem testadas por um outro software possam ser usadas neste trabalho;

d) oferecer para o projeto uma bateria de testes que pode ser utilizada constantemente, conforme o software vai sendo modificado.

2 TRABALHOS CORRELATOS

A seguir são apresentados dois trabalhos com características semelhantes aos principais objetivos do estudo aqui proposto. Em ambos, se nota a complexidade envolvida assim como a importância do conhecimento de engenharia de software aliada à experiência profissional para realizar com maestria o trabalho de cobertura de testes, além das incontáveis vantagens técnicas e organizacionais que esse trabalho entrega. Na seção 2.1 será

Comentado [AS1]: Redundante.

Comentado [AS2]: Frase longa. Rever.
Não se faz parágrafo com uma única frase.

Comentado [AS3]: Desenvolver é metodologia e não objetivo.

apresentado o trabalho sobre um estudo de caso sobre automatização de testes de software na empresa de desenvolvimento Softpan. (FERNANDES, 2019). Na seção 2.2 será apresentado o trabalho de um estudo de caso sobre automação de software utilizando um framework da IBM.

Comentado [AS4]: Referência?

- Falta 1 correlato.

2.1 AUTOMAÇÃO DE TESTES DE SOFTWARE: ESTUDO DE CASO DA EMPRESA SOFTPLAN

O trabalho de Matheus Fernandes e Samuel Fonseca. (2019) tem como objetivo geral apresentar um estudo de caso sobre automatização de testes de software na empresa de desenvolvimento Softplan. Este trabalho foi realizado avaliando pesquisas bibliográficas sobre qualidade, testes de software e automatização deste processo, estudo sobre como a automação de testes pode auxiliar à equipe quanto a redução de tempo e cobertura de testes, realizado comparativo entre os testes manuais e automatizados apresentando suas vantagens e desvantagens.

Durante o trabalho foram analisados alguns frameworks para a realização da automação dos cenários de testes, como o framework Robot e a ferramenta comercial Ranorex, porém o software que demonstrou maior aderência ao propósito do trabalho foi a ferramenta Cucumber. Essa escolha ocorreu com base em fatores técnicos e sua aderência ao propósito do trabalho.

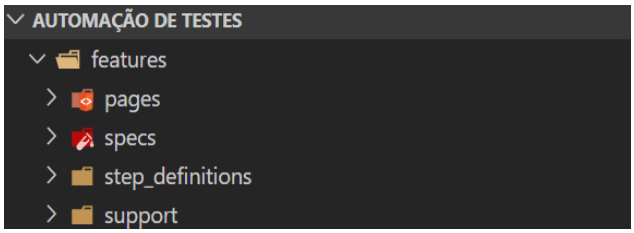
Na figura 1 é presentada a estrutura em que os arquivos da linguagem Ruby são organizados, separando as camadas de testes da camada visual, da camada de especificação e de suporte.

Comentado [AS5]: Coloque o recurso de referência cruzada para a figura. Faça isso em todo o texto.

Comentado [AS6]: Não se faz parágrafo com uma única frase.

Figura 1 – Apresentação da estrutura do Cucumber

Comentado [AS7]: - Tem estilo para legenda e fonte.
- figura, legenda e fonte devem estar centralizadas.
Verifique todos do seu texto. Estão errados, sem estilo.



Fonte: Cucumber (2020)

A figura 2 demonstra como é feita a implementação dos elementos visuais que serão utilizados na implementação da automação de teste.

Comentado [AS8]: Não se faz parágrafo com uma única frase.

Figura 2. Implementação das pages em Ruby

```ruby
class ArquivarPage < SitePrism::Page
    set_url 'portal/'
    element :iframe_mural, 'iframe[id="idFrameMural"]'
    element :anexos, 'input[value="Anexos"]'
    element :iframe_documento,'iframe[id="iFrameDocumentos"]'
    element :frame_main, 'frame[id="mainFrame"]'
    element :aba_dados, 'li[id="aba"]'
    element :outras_acoes, 'button[id="btn1"]'
    element :menu_arquivar, 'span[id="btn2"]'
    element :arquivar_item, 'input[name="btn3"]'
    element :menu_desarquivar, 'span[id="botaoAcao"]'
    element :desarquivar_item, 'input[value="acao"]'
    element :messagem_operacao_sucesso, 'table[class="tabela"] td[class="msg"] b'

    def arquivar()
        within_frame(iframe_mural) do
            anexos.click
            within_frame(iframe_documento) do
                within_frame(frame_main)do
                    aba_dados.click
                end
            end
            outras_acoes.click
            menu_arquivar.click
            arquivar_item.click
            sleep 1
            wait_until_messagem_operacao_sucesso_visible
            motivo = find('div[id="processo"] p[class="sds-p"]', match: :first).text
            puts motivo
            motivo
        end
    end
```

Fonte: Ruby (2020)

Na figura 3, é apresentadae a estrutura dos passos definidos para a execução dos cenários, estruturado no padrão BDD e pronto para execução pelo Cucumber para aferir o resultado de cada um.

Comentado [AS9]: Não se faz parágrafo com uma única frase.

Figura 3. Step_definitions

arquivar_documento.rb

features > step_definitions > arquivar_documento.rb

```ruby
Dado("que eu tenha um documento para ser arquivado") do
    @documento = ProcessoPage.new
    @documento.load

    @assunto = 'Ata de inutilização de bens'
    @documento.cadastraprocesso(@assunto)
    @mensagem = @documento.documentodigital()
end

Quando("arquivar esse documento") do
    @arquivar = ArquivarPage.new
    @motivo = @arquivar.arquivar()
end

Então("o sistema deverá remover esse documento da fila de trabalho") do
    @motivo.should eq ("Este documento encontra-se fora da fila de trabalho. Motivo: Arquivado.")
end
```

Fonte: Ruby (2020)

Na figura 4 é apresentado um relatório separado por feature, informando quantos cenários passaram e quantos falharam. Informando a duração da execução dos testes e sua respectiva situação.

Figura 4. Relatório de status dos testes

	Steps						Scenarios			Features	
Feature	Passed	Failed	Skipped	Pending	Undefined	Total	Passed	Failed	Total	Duration	Status
Arquivamento	24	0	0	0	0	24	6	0	6	3:16.290	Passed
Assinatura	32	0	0	0	0	32	8	0	8	4:32.534	Passed
Cadastros básicos	8	0	0	0	0	8	2	0	2	24.091	Passed

Fonte: Fernandes e Fonseca (2020)

2.2 AUTOTEST – UM FRAMEWORK REUTILIZÁVEL PARA A AUTOMAÇÃO DE TESTES FUNCIONAL DE SOFTWARE

O trabalho de Marcelo Fantinato, Adriano C. R. da Cunha, Sindo V. Dias, Sueli A. Mizuno e Cleida A. Q. Cunha (2019) tem como objetivo central apresentar um estudo de caso sobre automatização de testes de software utilizando o framework AutoTest em um software que gerencia e controla faturamentos.

Comentado [AS10]: Não se faz parágrafo com uma única frase.

Comentado [AS11]: Não está de acordo com a ABNT.

Este trabalho foi realizado com uma metodologia de coleta de métricas, para comprovar que o investimento na realização da atividade de automação é plenamente vantajoso a médio e longo prazo, visto que o maior trabalho é para realizar a análise, identificando os cenários, as pré-condições, o conjunto de elementos de entrada e a identificação dos conjunto de saída. Para isso, foi elaborado um plano para realizar este trabalho, que consiste em medir o esforço das atividades na realização da atividade de teste de forma manual e comparando com a forma de teste automatizada, para isso todas as atividades dos dois modelos foram aferidos, com o objetivo de dar transparência sobre este comparativo. O objetivo do trabalho foi apresentar os resultados, dar clareza e segurança a respeito do investimento e dar detalhes sobre seus ganhos e resultados.

Comentado [AS12]: Frase longa e confusa. Rever.

Na figura 5 é apresentado um resumo das principais métricas coletadas durante as execuções de testes, tanto manual quanto automatizada.

Figura 5 – Métricas coletadas

Tabela 1 – Principais Métricas Coletadas para o Módulo de Promoções

Tipo de Métrica	Teste Manual	Teste Automatizado
Número de Casos de Teste Executados	930	1644
Cobertura Funcional dos Casos de Teste[1]	65 %	88 %
Erros Detectados	174	+ 33
Tempo de Projeto de Teste	101 h	101 h
Tempo de uma Execução Completa dos Casos de Teste	123 h	14 h
Análise dos Resultados e Registro de Erros	34 h	28 h

Fonte: Fantinato, Cunha e Dias (2004)

3 PROPOSTA DA FERRAMENTA

Existem vários projetos, ferramentas, frameworks com suporte a automação de testes de software, porém cada ferramenta ~~desta~~ é limitada a um determinado escopo de aplicabilidade, as quais são implementadas para conhecer detalhadamente este escopo e serem efetivas dentro daquele contexto. A proposta deste trabalho é dar o primeiro passo para a construção de uma ferramenta que diminua o trabalho do desenvolvedor na criação de testes automatizados, oferecendo ao desenvolvedor recursos na camada de preparo (pré-condições) de tal forma que acelere a cobertura do cenário da funcionalidade do software, sendo essa unidade coesa, projetada para ser testada ou não. A ideia é oferecer na ferramenta a configuração das pré-condições, deixando a cargo da ferramenta as operações de manipulação (persistências) que a unidade sendo testada precisará para ser executada, toda a dependência que a unidade possui será possível configurar na ferramenta, sem necessidade de implementação por parte de quem está montando o cenário. Como resultado, espera-se maior agilidade na criação dos testes e por consequência um aumento na qualidade da análise realizada.

Nesta seção será apresentada a relevância deste trabalho quanto ao benefício do seu resultado, ~~quanto~~ ao conhecimento do que o software deve de fato garantir como resultado e como melhorar de forma gradativa as entregas do software. Serão descritos os ~~requisitos~~ Requisitos ~~funcionais~~ Funcionais (RF) e os ~~requisitos~~ Requisitos ~~não~~ Não ~~funcionais~~ Funcionais (RNF), terminando com a apresentação da metodologia e o cronograma planejado para o desenvolvimento deste trabalho.

3.1 JUSTIFICATIVA

A razão mais forte em investir em pesquisa neste trabalho é a falta de uma ferramenta de apoio na construção de testes para softwares que não foram projetados para testes, ou seja, que possuem uma estrutura de código com muita dependência, com nível de acoplamento alto, trazendo para os testes um trabalho muito grande para a criação da camada das pré-condições, obrigando o profissional a realizar a programação de todo essa fase de pré-condição. Outro ponto é a falta de qualidade de software legado em operação somado a dificuldade de investimento e pela alta complexidade em reconstruir o software e realizar as atividades de migração. Apenas resgatando que a definição empregada para software legado não está limitada à obsolescência da tecnologia e a falta de documentação, mas principalmente pela ausência de boas práticas de projetos e técnicas de engenharia que elevam a qualidade de manutenção do software.

Dito isto, esse trabalho é justificado pela ausência de manutenibilidade, visto que software legado em geral tem problemas sérios de arquitetura, comprometendo a manutenção visto que não se aplica o conceito de modularização, aumentando potencialmente os riscos de uma alteração refletir em algum ponto do software que não devia, pela falta de um projeto de software bem elaborado, respeitando técnicas e padrões mundialmente recomendados a área de testes acaba ficando sobrecarregada, visto que para qualquer alteração no software, precisa-se garantir que outra funcionalidade, requisito não tenha regredido, ou seja, deixado de funcionar.

No quadro 1 são apresentadas características verificadas nos dois trabalhos correlatos e que parcialmente auxiliaram a identificar requisitos para a ferramenta proposta.

Quadro 1 – Comparativo entre os trabalhos correlatos

Características	Cucumber	AutoTest (IBM)

Comentado [AS13]: O texto deve ser escrito com linguagem formal.

Comentado [AS14]: O texto deve ser escrito com linguagem formal.

Comentado [AS15]: O texto deve ser escrito com linguagem formal.
Frase longa. Rever.

Comentado [AS16]: Frase longa. Rever.
Não se faz parágrafo com uma única frase.

Comentado [AS17]: Retirar espaço

Comentado [AS18]: Faltou discutir o quadro apresentado.

Comentado [AS19]: Deixe o quadro na mesma página.

Bateria de testes por funcionalidade	Sim	Sim
Criação de testes para uma classe	Sim	Sim
Implementação dos cenários	Sim	Não
Registro das funcionalidades	Não	Sim
Registro dos casos de testes	Não	Sim
Consultas sobre os resultados dos testes	Sim	Sim
Relatórios sobre os resultados dos testes	Sim	Sim
Gráficos sobre os resultados dos testes	Sim	Sim

Fonte: elaborado pelo autor

3.2 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os requisitos funcionais deste trabalho são:

a) permitir o registro da funcionalidade e dos casos de testes abrangidos por ela;

b) permitir a configuração das pré-condições para cada cenário de teste;

c) permitir consultar as funcionalidades e seus casos de testes que estão sob cobertura de testes pela ferramenta;

d) permitir a criação de cenários previstos para a operação de cada classe/método;

e) permitir a execução de uma funcionalidade específica assim como também de um cenário específico;

f) apresentar o resultado da execução do cenário, podendo ser “OK” ou “NOK”.

Os requisitos não funcionais deste trabalho são:

a) ser construído na plataforma de desenvolvimento Java versão 11;

b) a IDE de construção deverá ser o IntelliJ IDEA Community;

c) ser utilizado a prática de desenvolvimento guiado por testes (TDD), utilizando o framework Junit.

3.3 METODOLOGIA

A construção do trabalho respeitará a seguintes fases:

a) pesquisa bibliográfica: pesquisar características de código legado, código monolítico, avaliação e testes em estruturas de códigos que não são modulares;

b) elicitação dos requisitos: revisar os requisitos previamente identificados;

c) projeto do software: construir o modelo estrutural e comportamental da ferramenta com o uso do diagrama de casos de uso, de classes e de atividades da UML, dando uma visão técnica clara da sua estrutura e do seu comportamento, facilitando estudos, evoluções ou até mesmo correções;

d) prova de conceito: realizar uma prova de conceito, visto a complexidade do tema serão necessários realizar alguns ensaios técnicos para aferir se o que se pensa é de fato implementável;

e) construção: fase que será implementada a ferramenta, para todas as camadas, sendo camada de apresentação, aplicação, domínio e infraestrutura. A construção será dirigida por testes, usando a prática de análise de código TDD;

f) testes: fase que será realizada junto a alguns usuários para validar as funcionalidades da ferramenta.

As fases serão realizadas nos períodos relacionados no Quadro 2.

Quadro 2 - Cronograma

	mar.		abr.		jun.		jul.	
etapas / quinzenas	1	2	1	2	1	2	1	2
pesquisa bibliográfica	X							
elicitação dos requisitos		X						
projeto de software			X					
prova de conceito				X	X			
construção						X	X	
testes								X

Fonte: elaborado pelo autor.

Comentado [AS20]: - Falta uma parte do quadro
- se inicia na 2ª quinzena de fevereiro

4 REVISÃO BIBLIOGRÁFICA

A definição de código legado é o código que recebemos de alguém. ~~,~~ ~~a~~A empresa pode tê-lo adquirido de outra ou os membros da equipe original podem ter passado para outros projetos. Código legado, portanto, é um código de outra pessoa e que você terá que alterar, mas que não compreende realmente. Na indústria, com frequência o termo código legado é usado com a gíria para um código difícil de alterar e que não entendemos. Porém após anos trabalhando com diversas equipes, ajudando-as a superar problemas graves de código, cheguei a uma definição diferente. Código legado é simplesmente código sem testes. (FEATHERS, 2013).

Comentado [AS21]: Fonte?

Comentado [AS22]: Texto deve ser escrito no impessoal.

Comentado [AS23]: Texto deve ser escrito no impessoal.

Comentado [AS24]: Não se coloca "." antes da referência.

Teste consiste na execução de um programa com um conjunto finito de casos, com o objetivo de verificar se ele possui o comportamento esperado. Existem diversos tipos de testes, como testes de unidade (quando se testa uma pequena unidade do código, como uma classe), testes de integração (quando se testa uma unidade com maior granularidade, como um conjunto de classes), testes de performance (quando se submete o sistema a uma carga de processamento para verificar seu desempenho). (VALENTE, 2020)

A garantia da qualidade de software é atualmente reconhecida como um assunto que permeia todo o processo de desenvolvimento, os testes e a verificação dos problemas propriamente ditos são tópicos de grande importância no processo de qualidade. Já existe uma discussão técnica para verificar a correção de programas de forma matematicamente rigorosa, mas a conclusão é que a maioria dos sistemas é verificada por meio testes, infelizmente, tais testes são, na melhor das hipóteses, inexatos. Não é possível garantir que uma unidade de software esteja correta por meio de testes, a menos que fôssemos capazes de executar testes que exaurissem todos os cenários possíveis, sabemos que humanamente isto é muito difícil. Mesmo programas simples podem existir bilhões de caminhos diferentes a serem percorridos. Então testar todos os caminhos possíveis dentro de um programa complexo é uma tarefa impossível. (BROOKSHEAR, 2013).

REFERÊNCIAS

Comentado [AS25]: Rever todas as referências e ajustá-las de acordo com a norma NBR-6023-2018.

DINIZ, Samuel. **O problema do software legado.** Disponível em: < http://blogdosamueldiniz.blogspot.com/2010/02/o-problema-do-software-legado.html> Acesso em 09 set. 2020.

FANTINATO, Marcelo. AutoTest – **Um framework reutilizável para a automação de testes funcionais de software.** Disponível em: <https://www.researchgate.net/profile/Marcelo_Fantinato/publication/229004366_AutoTest-Um_Framework_Reutilizavel_para_a_Automacao_de_Teste_Funcional_de_Software/links/00b7d525a17636e087000000/AutoTest-Um-Framework-Reutilizavel-para-a-Automacao-de-Teste-Funcional-de-Software.pdf>. Acesso em 01 out. 2020.

FEATHERS, Michael. **Working effectively with legacy code**. 1º edição. Amazon: Pearson, 22 set. 2004.

FERNANDES, Matheus. **Automação de testes de software: Estudo de caso da empresa softplan.** Disponível em: <https://www.riuni.unisul.br/bitstream/handle/12345/10127/AUTOMA%c3%87%c3%83O%20DE%20TESTES%20DE%20SOFTWARE-ESTUDO%20DE%20CASO%20DA%20EMPRESA%20SOFTPLAN-TCC_MATHEUS%20FERNANDES-SAMUEL%20TOMKELSKI%20FONSECA.pdf?sequence=1&isAllowed=y> Acesso em 03 out. 2020.

Formatado: Inglês (Estados Unidos)

IEEE, **Raking das linguagens de programação mais utilizadas no mundo.**
Disponível em: <https://canaltech.com.br/software/java-lidera-ranking-das-linguagens-de-programacao-mais-utilizadas-no-mundo-24970/> Acesso em 05 out. 2020.

MARTINS, Daniele. **Identificação de caracteríscas de sistemas legados a partir de análise de conteúdo da literatura.**
Disponível em: <https://sol.sbc.org.br/index.php/eres/article/download/10084/9965/> Acesso em 11 set. 2020.

PIGOSKI, Thomas M. **Practical Software Maintenance: Best Practices for Managing your Software Investment**, 1997. O apoio e encarecimento de produtos de software a partir do projeto e desenvolvimento inicial através da sua vida útil;

SILVA, Luciana P. **Uma proposta de evolução em sistemas legados.**
Disponível em: <http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER04/Luciana_Paiva.pdf> Acesso em 13 set. 2020.

VALENTE, Marco. **Engenharia de software moderna.**
Disponível em: <https://docplayer.com.br/178875236-Engenharia-de-software-moderna.html>
Acesso em 06 out. 2020.

Comentado [AS26]: Referência não encontrada no texto.

Comentado [AS27]: Referência não encontrada no texto.

Formatado: Inglês (Estados Unidos)

Comentado [AS28]: Referência não encontrada no texto.

ASSINATURAS

(Atenção: todas as folhas devem estar rubricadas)

Assinatura do(a) Aluno(a): _____

Assinatura do(a) Orientador(a): _____

Assinatura do(a) Coorientador(a) (se houver): _____

Observações do orientador em relação a itens não atendidos do pré-projeto (se houver):

FORMULÁRIO DE AVALIAÇÃO – PROFESSOR TCC I

Acadêmico(a): _____

Avaliador(a): _____

	ASPECTOS AVALIADOS[1]	atende	atende parcialmente	não atende
ASPECTOS TÉCNICOS	1. INTRODUÇÃO O tema de pesquisa está devidamente contextualizado/delimitado?	x		
	O problema está claramente formulado?	x		
	2. OBJETIVOS O objetivo principal está claramente definido e é passível de ser alcançado?		x	
	Os objetivos específicos são coerentes com o objetivo principal?	x		
	3. JUSTIFICATIVA São apresentados argumentos científicos, técnicos ou metodológicos que justificam a proposta?		x	
	São apresentadas as contribuições teóricas, práticas ou sociais que justificam a proposta?		x	
	4. METODOLOGIA Foram relacionadas todas as etapas necessárias para o desenvolvimento do TCC?	x		
	Os métodos, recursos e o cronograma estão devidamente apresentados?	x		
	5. REVISÃO BIBLIOGRÁFICA (atenção para a diferença de conteúdo entre projeto e pré-projeto) Os assuntos apresentados são suficientes e têm relação com o tema do TCC?		x	
ASPECTOS METODOLÓGICOS	6. LINGUAGEM USADA (redação) O texto completo é coerente e redigido corretamente em língua portuguesa, usando linguagem formal/científica?			x
	A exposição do assunto é ordenada (as ideias estão bem encadeadas e a linguagem utilizada é clara)?			x
	7. ORGANIZAÇÃO E APRESENTAÇÃO GRÁFICA DO TEXTO A organização e apresentação dos capítulos, seções, subseções e parágrafos estão de acordo com o modelo estabelecido?		x	
	8. ILUSTRAÇÕES (figuras, quadros, tabelas) As ilustrações são legíveis e obedecem às normas da ABNT?		x	
	9. REFERÊNCIAS E CITAÇÕES As referências obedecem às normas da ABNT?			x
	As citações obedecem às normas da ABNT?	x		
	Todos os documentos citados foram referenciados e vice-versa, isto é, as citações e referências são consistentes?		x	

PARECER – PROFESSOR DE TCC I OU COORDENADOR DE TCC
(PREENCHER APENAS NO PROJETO):

O projeto de TCC será reprovado se:

- qualquer um dos itens tiver resposta NÃO ATENDE;
- pelo menos **4 (quatro)** itens dos **ASPECTOS TÉCNICOS** tiverem resposta ATENDE PARCIALMENTE; ou
- pelo menos **4 (quatro)** itens dos **ASPECTOS METODOLÓGICOS** tiverem resposta ATENDE PARCIALMENTE.

PARECER: ( ) APROVADO ( ) REPROVADO

Assinatura: _____ Data: _____

[1] Quando o avaliador marcar algum item como atende parcialmente ou não atende, deve obrigatoriamente indicar os motivos no texto, para que o aluno saiba o porquê da avaliação.