CURSO DE CIÊNCIA DA COMPUTAÇÃO – TCC	
( ) PRÉ-PROJETO (X) 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 é resultado da experiência deste autor ao longo da sua trajetória 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. Ou seja, o software legado está muito associado a problemas de engenharia, falta da utilização de boas práticas que revertam as características de software legado. 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).

Sistemas legados trazem uma série de riscos às empresas, tanto para as empresas que mantém o sistema como para as empresas que o utilizam Warren (1999). Para as empresas destacamos os riscos da falta de estabilidade e o custo de manutenção, para as empresas que usam destacamos o risco de o software deixar de funcionar, causando prejuízos para o negócio da organização. Qualquer mudança na estrutura ou no comportamento do software legado pode trazer efeitos colaterais, como regressão de funcionalidades em razão do software não ser projetado para manutenção e pela ausência de cobertura do código por testes que trazem segurança para as mudanças.

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 para os profissionais envolvidos e para a organização se manter no mercado evoluindo seu negócio.

Nesse contexto o trabalho consiste em oferecer ao profissional que está dando manutenção ~~para~~ em um software com código legado ~~alguns~~ recursos para realizar testes. A ferramenta irá permitir que, para o cenário de teste, seja informado os elementos de entrada, a ação/funcionalidade que se quer testar e que elementos de saída devem ser produzidos pela funcionalidade sob teste. 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 disponibilizar uma ferramenta para a construção de testes automatizados para software legado em Java.

Os objetivos específicos são:

a) ~~implementar~~ disponibilizar um módulo que permita ao usuário informar as pré-condições, a operação no software que se quer testar e os elementos que devem ser produzidos ao final da operação;

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

c) oferecer para o projeto uma bateria de testes que pode ser utilizada constantemente, conforme o software é modificado.

Comentado [AS1]: Fortemente?

Comentado [AS2]: Esta frase falta uma conclusão. Ficou confuso.

Comentado [AS3]: Aonde?

Comentado [AS4]: Não está de acordo com a norma.

Comentado [AS5]: Texto deve ser escrito no impessoal.

Comentado [AS6]: Confuso. Rever redação.

Comentado [AS7]: Fonte?

Comentado [AS8]: Fonte?

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. Na seção 2.1 será apresentado o trabalho 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 (FANTINATO et al, ANO).

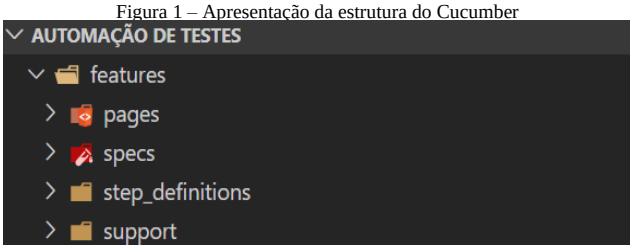
2.1 AUTOMAÇÃO DE TESTES DE SOFTWARE: ESTUDO DE CASO DA EMPRESA SOFTPLAN

O trabalho de Fernandes e 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. Ainda os autores estudaram sobre como a automação de testes pode auxiliar à equipe quanto a redução de tempo e cobertura de testes. Além disso realizaram um 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. Dentre as análises realizadas pelos autores, 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 **Erro! Fonte de referência não encontrada.** é apresentada a estrutura em que os arquivos são organizados, separando em camada de teste, em camada visual, em camada de especificação e de camada de suporte.

Figura 1 – Apresentação da estrutura do Cucumber



Fonte: Cucumber (2020).

Comentado [AS9]: São 3 trabalhos correlatos no total.

Comentado [AS10]: et al. (em itálico e com “.”). Rever todos no texto.

Comentado [AS11]: Aqui você deve colocar o ano da referência.

Comentado [AS12]: Tem ponto no final na fonte. Rever em todas as figuras.

A **Erro! Fonte de referência não encontrada.** demonstra a estrutura de um arquivo page. Neste arquivo é implementado os métodos de testes e são elencados todos os elementos das telas que serão utilizados na programação da automação.

Comentado [AS13]: Não se faz parágrafo com uma única frase.

Comentado [AS14]: Não tem recuo.

Figura 2. Implementação das pages em Ruby

Comentado [AS15]: Explique o código-fonte.

arquivar_page.rb

features > pages > arquivar_page.rb

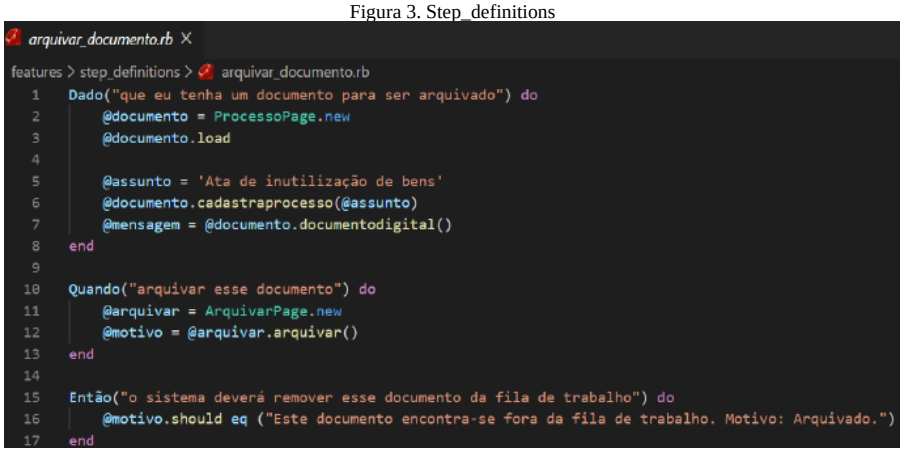
```
1   class ArquivarPage < SitePrism::Page
2       set_url 'portal/'
3       element :iframe_mural, 'iframe[id="idFrameMural"]'
4       element :anexos, 'input[value="Anexos"]'
5       element :iframe_documento,'iframe[id="iFrameDocumentos"]'
6       element :frame_main, 'frame[id="mainFrame"]'
7       element :aba_dados, 'li[id="aba"]'
8       element :outras_acoes, 'button[id="btn1"]'
9       element :menu_arquivar, 'span[id="btn2"]'
10      element :arquivar_item, 'input[name="btn3"]'
11      element :menu_desarquivar, 'span[id="botaoAcao"]'
12      element :desarquivar_item, 'input[value="acao"]'
13      element :messagem_operacao_sucesso, 'table[class="tabela"] td[class="msg"] b'
14
15      def arquivar()
16          within_frame(iframe_mural) do
17              anexos.click
18              within_frame(iframe_documento) do
19                  within_frame(frame_main)do
20                      aba_dados.click
21                  end
22              end
23              outras_acoes.click
24              menu_arquivar.click
25              arquivar_item.click
26              sleep 1
27              wait_until_messagem_operacao_sucesso_visible
28              motivo = find('div[id="processo"] p[class="sds-p"]', match: :first).text
29              puts motivo
30              motivo
31          end
32      end
```

Fonte: Ruby (2020).

A **Erro! Fonte de referência não encontrada.** demonstra a estrutura do arquivo Step_definitions. Nesta pasta, os cenários que foram definidos são transformados e interpretados em linguagem de programação. Os cenários são definidos em uma linguagem ubíqua e são executados pelo Cucumber para aferir o resultado de cada um.

Comentado [AS16]: Não deve ter recuo.

Figura 3. Step_definitions



Fonte: Ruby (2020).

A **Erro! Fonte de referência não encontrada.** demonstra o relatório de status do Cucumber. O relatório apresenta os cenários elaborados, exibindo falhas e aprovações.

Comentado [AS17]: O que exatamente a figura está apresentando.

Comentado [AS18]: Sem recuo.

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 Fantinato *et al*. (2019) tem o propósito de apresentar o *framework* AutoTest, um *framework* reutilizável para a automação da execução de teste funcional, amplamente aplicável em diferentes projetos de desenvolvimento de software. O trabalho foi realizado para comprovar que o investimento na realização da atividade de automação é plenamente vantajoso a médio e a longo prazo, visto que o maior trabalho é para realizar a análise, identificando os cenários, as pré-condições, o conjunto de valores dos elementos de entrada e o conjunto de valores dos elementos de saída.

Formatado: Fonte: Itálico

Fantinato *et al*. (2019) ~~O trabalho~~ avalia o esforço das atividades na realização de testes no modelo manual e faz um comparativo com as atividades para a realização no modelo de testes automatizado. Para ter o comparativo, todas as atividades dos dois modelos foram aferidas, com o objetivo de dar transparência sobre o resultado. O trabalho apresenta ~~os~~ como? resultados, ~~d~~a clareza e segurança a respeito do investimento e apresenta detalhes sobre seus benefícios e resultados.

Comentado [AS19]: De que forma? Explique melhor.

Na **Erro! Fonte de referência não encontrada.** é apresentado um resumo das principais métricas coletadas durante as execuções de testes, tanto manual quanto automatizada.

Comentado [AS20]: Explique a tabela.

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 e *frameworks* para automação de testes de software, alguns destes focando em testes de unidade, outros se concentram em apoiar os testes em nível de integração ou em nível de testes de aceitação. A proposta deste trabalho é atuar no nível de teste de integração, facilitando o desenvolvimento desses testes, removendo a necessidade de o desenvolvedor implementar toda a preparação do cenário, que também é conhecida como camada de preparo ou de pré-condições. Em software legado, a implementação dos testes de integração acaba sendo muito trabalhosa e por consequência muito cara, isso ocorre pelo fato do software ter muitos artefatos dependentes, trazendo um acoplamento muito alto e com baixa coesão. Neste contexto é necessário que os profissionais analisem toda a dependência envolvida na operação do software que deseja testar e, a partir desta análise, implementar o cenário respeitando todas as dependências mapeadas. A ferramenta proposta ~~vai~~ visa eliminar a etapa de implementação do cenário, deixando o profissional analisar o que é necessário para a montagem do cenário, informando na ferramenta os objetos que precisam ser criados que estão acoplados na operação que se deseja testar, substituindo a implementação do cenário por uma configuração na ferramenta.

Comentado [AS21]: Você não sabe as vai pois não está pronta

Nesta seção será apresentada a justificativa do trabalho proposto, bem como os Requisitos Funcionais (RF) e os Requisitos Não Funcionais (RNF). ~~terminando com~~Por fim, é apresentada a ~~apresentação da~~ metodologia e o cronograma ~~planejado~~ para o desenvolvimento deste trabalho.

3.1 JUSTIFICATIVA

O~~No~~ **Erro! Fonte de referência não encontrada.** ~~é possível fazer~~apresenta um comparativo entre os trabalhos correlatos~~, apresentando os trabalhos e~~ com as características propostas em cada trabalho.

Quadro 1 – Comparativo entre os trabalhos correlatos

Características	Cucumber	AutoTest (IBM)
Bateria de testes por funcionalidade	Sim	Sim
Criação de testes para uma classe	Sim	Sim
Scripts para testes	Sim	Sim
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.

Como pode ser visto no **Erro! Fonte de referência não encontrada.**, os trabalhos apresentam algumas diferenças na qual atendem as características elencadas.~~,~~ Fantinato (2019) utilizou um *framework* comercial e que atende todas as características citadas, já Fonseca (2019) utilizou o *framework* Cucumber que atende parcialmente o mesmo conjunto de características. No trabalho de Fantinato (2019) a criação dos cenários de teste ocorre por meio de um arquivo externo~~,~~. Para isso, é necessário criar este arquivo com os dados de teste que será consumido por um script de testes quando o teste for executado. No trabalho de Fonseca (2019) a criação dos cenários de testes ocorre através da implementação de scripts, utilizando a linguagem do *framework* e estruturando essa montagem por um formato pré-definido do *framework*.

Em ambos os trabalhos é claraõ a necessidade de um esforço em realizar o trabalho de montagem dos cenários. No trabalho do Fantinato (2019) é necessário conhecer a estrutura do arquivo que mantêm os dados de testes, ou seja, os dados que serão imputados para a operação que se deseja testar. No trabalho do Fonseca (2019) é necessário conhecer a linguagem implementada no *~~F~~ramework* para realizar toda a implementação do cenário. Ambos os trabalhos não possuem uma mecanização para a etapa de implementação dos cenários, e ~~ambos os trabalhos~~ exigem ~~a~~ implementação.

A partir do que foi apresentado ~~acima~~, este trabalho se torna relevante pelo fato de entregar uma ferramenta que elimina a necessidade de o desenvolvedor implementar qualquer cenário de teste. O desenvolvedor ~~irá~~ poderá analisar o que precisa ser montado de cenário, abrir a ferramenta e configurar o cenário, não sendo necessário conhecer a ~~nenhuma~~ estrutura de arquivo adicional e ~~nenhuma~~ linguagem de programação ou ambiente de desenvolvimento. Esta configuração poderá ser realizada por um profissional que não conhece desenvolvimento, como por exemplo um profissional que atua somente na área de testes.

Comentado [AS22]: Confuso. Rever.

Comentado [AS23]: Cuidado com a repetição de palavras.

Formatado: Realce

Formatado: Realce

Formatado: Realce

Formatado: Realce

Comentado [AS24]: De que?

Comentado [AS25]: Não deixou claro as contribuições teóricas, práticas ou sociais que justificam a proposta.

3.2 REQUISITOS PRINCIPAIS

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 desenvolvido na plataforma de desenvolvimento Java versão 11;

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

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

3.3 METODOLOGIA

O desenvolvimento 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 realizando algumas POCs para definir a melhor abordagem;

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

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

Fonte: elaborado pelo autor.

Comentado [AS26]: Só isso?

Comentado [AS27]: Aqui você já deve ter apresentado o seu TCC.

Comentado [AS28]: Não deve deixar este espaço.

4 REVISÃO BIBLIOGRÁFICA

Este capítulo descreve os assuntos que darão fundamentação ao trabalho a ser realizado: código legado, e testes de integração, que será utilizado para realizar testes de maneira mais rápida e fácil para códigos de software legado.

Segundo Warren *et al.* (1999), o termo sistema legado descreve um sistema antigo que permanece em operação em uma organização. Geralmente utilizam banco de dados obsoletos. Normalmente são aplicações complexas e de difícil manutenção por falta de documentação e com saída dos profissionais técnicos que participaram em algum momento do projeto. Sistemas legados geralmente apresentam problemas para compreender a regra de negócio neles implementados, problemas de estruturação do código, vários estilos de programação, alto acoplamento e baixa coesão.

Para Ning *et al.* (1994) sistemas legados inibem o crescimento de uma organização e sua capacidade de mudança, Bennett *et al.* (1995) diz que são sistemas que realizam tarefas úteis para a organização, mas foram desenvolvidos com tecnologias e métodos considerados obsoletos. Segundo Umas *et al.* (1997), sistemas legados são sistemas com valor crítico para o negócio das empresas desenvolvidos há cindo ou mais anos. Por outro lado, Juric *et al.* (2000) define que é qualquer sistema que, independentemente da idade ou arquitetura, ainda é útil e está em uso pela organização.

A qualidade de um software depende de todas as disciplinas da engenharia de software, desde ~~da~~ a sua concepção até sua implantação. Porém a garantia da qualidade do software está relacionada, de acordo com Bastos *et al.* (2007), com o aprimoramento da atividade de testes. Quando o processo de teste é devidamente executado há uma redução notória dos custos de manutenção, que por consequência traz credibilidade junto ao cliente. Conforme diz Bastos *et al.* (2007), os testes eram efetuados pelos próprios desenvolvedores de software, cobrindo aquilo que hoje chamamos de testes unitários e testes de integração.

O objetivo dos testes de integração é verificar as estruturas de comunicação entre as unidades que compõe o sistema. Técnicas que exercitam caminhos específicos do programa são usadas para garantir a cobertura dos principais caminhos de controles (PRESSMAN, 2005). A automação desses testes consiste na utilização de uma ferramenta para controlar a execução dos testes, utilizando os casos de testes para validar o software. Segundo (Medium, 2019), são verificados, de forma automatizada se os requisitos funcionais ou não funcionais atendem ao padrão especificado pelo negócio, comparando os resultados com o que os requisitos produziram, com o objetivo de reduzir ao máximo o envolvimento de humanos nessas tarefas.

Segundo Bernado Kon (2008), os benefícios dos testes automatizados são estabilidade, confiabilidade e facilidade de manutenção. Dando grande ênfase em como a transformação dos testes executados de forma manual para a forma de testes automatizados pode gerar valor agregado para as organizações.

Comentado [AS29]: No projeto deve ser apresentado estudo inicial sobre o tema escolhido, detalhando cada parágrafo, na forma de seções, os assuntos relacionados no pré-projeto.

A revisão bibliográfica consiste na sistematização de ideias e fundamentos de autores que dão sustentação ao assunto estudado.

Deve ser melhorado.

Comentado [AS30]: Referências muito antigas para revisão bibliográfica.

Comentado [AS31]: Texto deve ser escrito no impessoal.

Comentado [AS32]: Deve-se evitar iniciar a frase com gerúndio. Gerúndio complementa alguma ideia.

REFERÊNCIAS

Comentado [AS33]: Algumas referências você coloca o nome completo, outras você coloca o nome do autor abreviado. Você deve seguir um padrão.

BASTOS, Aderson; RIOS Emerson; CRISTALLI Ricardo; MOREIRA Trayahú. **Base de conhecimento em teste de Software**. São Paulo: Martins, 2007.

Bennett, K. (1995). **Legacy systems**: Coping with success. IEEE Software, 12(1):19–23.

Comentado [AS34]: Não está de acordo com a norma.

BERNARDO, P. C.; KON, F. **A importância dos testes automatizados. Engenharia de Software Magazine**, p. 1Ű7, 2008. Citado na página 13.

Comentado [AS35]: Referência não encontrada no texto.

FANTINATO, Marcelo. AutoTest – **Um framework reutilizável para a automação de testes funcionais de software.** Trabalho de monografia – CPqD Telecom & Solutions / DSB – Diretoria de Soluções em Billing.

FERNANDES, Matheus. **Automação de testes de software: Estudo de caso da empresa softplan.** Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) – Universidade do Sul de Santa Catarina, Unisul.

Ian Warren (1999). **The Renaissance of Legacy Systems**. Method Support for Software System Evolution. London: Springer. ISBN 978-1-85233-060-6.

Comentado [AS36]: Não está de acordo com a norma.

Juric, M. B., Rozman, I., and Hericko, M. (2000**). Performance comparison of corba and RMI**. Information and Software Technology, 42(13):915–933.

Comentado [AS37]: Não está de acordo com a norma.

MEDIUM. **Automação de testes: o que é, quando e por que automatizar**. [S.l.], 2019.

Ning, J., Engberts, A. P., and Kozaczynski, W. V. (1994). **Automated support for legacy code understanding. Communications of the ACM**, 37(5):50–57.

Comentado [AS38]: Não está de acordo com a norma.

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.

PRESSMAN, R. S. **Software Engineering – A Practilioner's Approach.** 5ª. ed. [S.1]: McGraw-Hill, 2000.

Umar, A. (1997). **Application (re)engineering: Building Web-based Applications and Dealing with Legacies. Prentice Hall**, 1th edition.

Comentado [AS39]: Não está de acordo com a norma.

Comentado [AS40]: 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): Jonathan Luiz de Lara_____

Avaliador(a): Andreza Sartori_____

	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 (x) REPROVADO

Assinatura: _____ Data: 08/12/2020 _____

[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.