UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Diogo Canut Freitas Peixoto

Geração Automática de Relatórios de Confiabilidade de Software: Projeto e Implementação de Engine para a Plataforma X-RAT

Uberlândia, Brasil

2023

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Diogo Canut Freitas Peixoto

Geração Automática de Relatórios de Confiabilidade de Software: Projeto e Implementação de Engine para a Plataforma X-RAT

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Orientador: Rivalino Matias Jr.

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2023

Diogo Canut Freitas Peixoto

Geração Automática de Relatórios de Confiabilidade de Software: Projeto e Implementação de Engine para a Plataforma X-RAT

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Trabalho aprovado. Uberlândia, Brasil, 23 de janeiro de 2023:

Rivalino Matias Jr.
Orientador

Celso Maciel da Costa

Caio Augusto Rodrigues dos Santos

Uberlândia, Brasil

2023

Agradecimentos

Ao Prof. Dr. Rivalino Matias Jr, pela orientação e incentivo, sem os quais este trabalho não seria possível.

Aos professores da Universidade Federal de Uberlândia (UFU), em especial da Faculdade de Computação (FACOM), por contribuírem na minha formação e no meu crescimento como indivíduo.

Aos alunos do laboratório .SDLAB, pela amizade e colaboração.

Aos meus pais, por todo apoio e incentivo ao estudo.

"Uma das regras básicas do universo é que nada é perfeito. A perfeição simplesmente não existe. Sem imperfeição, nem você nem eu existiríamos." Stephen Hawking

Resumo

Em meio a dependência de sistemas computacionais presente na sociedade moderna, torna-se um importante objeto de estudo a confiabilidade destes sistemas. Para determinados softwares, que executam funções críticas, operar livre de falhas é crucial, visto que uma falha nestes sistemas pode ocasionar acidentes fatais.

Neste contexto, este trabalho faz parte do desenvolvimento de funcionalidades para a plataforma X-RAT, uma ferramenta para análise de confiabilidade de software, que visa a coleta, *parsing* e análise de *logs* de falhas, a fim de ao final do processo produzir um relatório de confiabilidade para o usuário.

Assim, este trabalho engloba o desenvolvimento dos módulos de *parsing*, análise de *logs* de falhas e produção do relatório de confiabilidade.

Palavras-chave: Confiabilidade de software, relatório técnico, serviço online.

Lista de ilustrações

Figura 1 – Arquitetura da Engine de análise 22
Figura 2 – Mensagem de falha 24
Figura 3 – Como o Event Viewer mostra uma mensagem de falha 25
Figura 4 – Estrutura de um evento do Win10 no arquivo de log 26
Figura 5 – Etapas do Analisador 28
Figura 6 – Interface para coleta de logs de falha 30
Figura 7 – Questionário sobre o perfil do usuário 31
Figura 8 – Relatório de confiabilidade 32
Figura 9 – Características das falhas 33
Figura 10 – Falhas mais recorrentes 33
Figura 11 – Ranking de distribuições 34
Figura 12 – Métricas de confiabilidade 34

Lista de abreviaturas e siglas

.SDLAB	Software Dependability Laboratory
X-RAT	X-Reliability Analysis Tool
Win10	Microsoft Windows 10
Win11	Microsoft Windows 11
SO	Sistema Operacional
SRE	Software Reliability Engineering
TBF	Time Between Failures
MTTF	Mean Time To Failure
MTTR	Mean Time To Repair
MTBF	Mean Time Between Failure
USER_{APP}	Aplicação de Usuário
SO_{APP}	Aplicação de Sistema Operacional
SO_{SVC}	Serviço de Sistema Operacional
SO_{KNL}	Kernel
UTC	Universal Time Coordinated

Sumário

1 INTRODUÇÃO 9
1.1 Justificativa 10
1.2 Objetivos 11
1.2.1 Geral 11
1.2.2 Específicos 11
1.3 Metodologia 11
1.4 Demais capítulos 12

2 REVISÃO DA LITERATURA 13
2.1 Referencial teórico 13
2.1.1 Engenharia de Confiabilidade de Software 13
2.1.2 Conceitos básicos 14
2.1.3 Métricas de confiabilidade 14
2.2 Trabalhos relacionados 16
2.3 Trabalhos correlatos 18

3 PLATAFORMA X-RAT 21
3.1 Visão geral 21
3.2 Relatórios de Confiabilidade 21
3.3 Projeto Desenvolvido 22
3.3.1 Banco de dados de mensagens de falha 23
3.3.2 Parser 25
3.3.3 Analisador 27

4 RESULTADOS 30
4.1 Caso de uso 30
4.2 Resultados alcançados 33
4.3 Trabalhos futuros 35

5 CONCLUSÃO 36
5.1 Técnica 36
5.2 Aprendizado 36
5.3 Dificuldades encontradas 37

REFERÊNCIAS 38

1 Introdução

Vive-se atualmente na era da informação, em que a utilização de sistemas computacionais se tornou intrinsecamente parte da rotina humana e está presente nos mais diversos setores, desde aplicações médicas, empresariais, educacionais e até para o entretenimento.

É fato que a sociedade moderna vive em dependência de sistemas computacionais. Em consequência dessa ampla adoção, falhas nesses sistemas podem gerar desde simples inconvenientes até acidentes fatais. Alguns exemplos de falhas de alto impacto são:

- Therac-25: Problemas no desenvolvimento do software responsável por controlar o feixe de energia de uma máquina de radioterapia, chamada de Therac-25, causaram uma série de acidentes entre 1985 e 1987, nos quais pacientes receberam overdoses de radiação, resultando em mortes e lesões graves (LEVESON; TURNER, 1993).

- Ariane 5, voo 501: Em junho de 1996, o foguete não tripulado Ariane 5 explodiu quarenta segundos após seu lançamento. O custo do projeto foi de aproximadamente 7 bilhões de dólares. A causa do problema foi um defeito de conversão numérica no software de controle inercial do foguete (DOWSON, 1997).

- Boeing 737 Max: Mais de 300 pessoas morreram após duas colisões entre outubro de 2018 e março de 2019 de aeronaves modelo 737 Max. Esses acidentes foram causados por falhas em um sistema de software de prevenção de estol, que empurrou o nariz dos aviões para baixo. A família de aeronaves 737 Max está impedida de voar desde março de 2019, custando bilhões de dólares para a Boeing e para diversas companhias aéreas (WALL; SHERMAN, 2019).

Todos os desastres mencionados acima foram causados por falhas de software. De fato, dentre todas as ameaças ao correto funcionamento de sistemas computacionais, é bem conhecido que as falhas de software são dominantes. Neste contexto, garantir a qualidade do software tornou-se um importante requisito. Pode-se medir a qualidade do software usando atributos como funcionalidade, usabilidade, capacidade, manutenção, confiabilidade, segurança e desempenho. Entretanto, a confiabilidade é considerada o principal fator de qualidade entre esses atributos, uma vez que leva em consideração falhas de software, as quais são responsáveis pela maior parte da inatividade não planejada dos sistemas computacionais (LYU, 2007). Formalmente, a confiabilidade de um software consiste na probabilidade de um software desempenhar com êxito seu propósito especificado por um

determinado período de tempo (ANSI/IEEE, 1991).

A estimativa de métricas de confiabilidade envolve a análise de dados de falhas dos sistemas, os quais normalmente estão disponíveis em *logs*. Tendo em vista automatizar esse processo, está sendo desenvolvido no grupo de pesquisas Software Dependability Laboratory (.SDLAB[1]), a plataforma de software X-RAT (*X-Reliability Analysis Tool*), que visa a coleta, *parsing* e análise de *logs* de falhas produzindo um relatório de confiabilidade para o usuário.

1.1 Justificativa

Atualmente, tornou-se indispensável garantir a confiabilidade de sistemas de software, dado que muitos desses sistemas são empregados em diversas áreas da sociedade. No entanto, a preocupação com o correto funcionamento não deve ser exclusiva de sistemas críticos, especializados ou customizados, uma vez que esses sistemas muitas vezes utilizam partes de software de uso geral, especialmente sistemas operacionais. Como por exemplo, o amplo uso de sistemas operacionais Linux em diferentes sistemas embarcados críticos e não críticos (ABBOTT, 2012). Desta forma, mesmo que seja altamente confiável, uma aplicação crítica pode ser prejudicada caso ocorram falhas no sistema operacional subjacente.

Este trabalho foi desenvolvido no âmbito do projeto X-RAT, que consiste em uma ferramenta capaz de coletar e analisar *logs* de falhas de diferentes aplicações de software. A ferramenta é dividida em duas partes: (1) um sistema coletor de dados, onde voluntários informam o perfil de uso de seus computadores e fazem *upload* de seus arquivos de *log* de falhas "manualmente"(por meio de um *website*) ou "automaticamente"(via uma aplicação de software), e (2) uma *engine*, responsável por analisar e caracterizar as falhas armazenadas nos *logs*, além de empregar diferentes técnicas estatísticas com a finalidade de estimar a confiabilidade das aplicações analisadas. Por fim, será gerado um relatório de confiabilidade para o voluntário sobre o seu sistema computacional. O foco deste trabalho encontra-se no desenvolvimento de novas funcionalidades no software X-RAT, em especial no escopo da *engine* de análise.

A importância desse trabalho se encontra no tocante à necessidade de automatizar o processo de coleta de dados e análise de confiabilidade a partir de métricas amplamente utilizadas na área de engenharia de confiabilidade. Além disso, busca-se compreender como diferentes sistemas de software falham, ou seja, busca-se entender a dinâmica que rege os diferentes tipos de manifestação de falhas de software.

[1] <https://sdlab.facom.ufu.br/>

1.2 Objetivos

1.2.1 Geral

Projetar e programar algoritmos para estimar automaticamente métricas de confiabilidade de software com base em dados de *logs*.

1.2.2 Específicos

- Automatizar o processo de estimativa de métricas de confiabilidade a partir de leitura e *parsing* de dados de *logs*.

- Gerar um relatório de confiabilidade com base nas métricas de confiabilidade estimadas.

1.3 Metodologia

Conforme mencionado na Seção 1.1, este trabalho faz parte do desenvolvimento da X-RAT, em especial no desenvolvimento de funcionalidades para a análise de arquivos de *log* de falhas. Inicialmente será utilizado um conjunto de dados composto por falhas de software que ocorreram em computadores executando o Microsoft Windows 10 (Win10) e 11 (Win11). Ambos foram escolhidos por serem os sistemas operacionais (SOs) *desktop* mais utilizados atualmente (DESKTOP. . . , 2022). Além disso, esses SOs possuem um serviço responsável pelo registro detalhado de eventos chamado *Windows Event Log*.

No Windows, um evento é qualquer acontecimento potencialmente importante para o usuário, para um administrador de sistema ou de rede, para o sistema operacional ou para uma aplicação. Dentre os eventos registrados pelo *Windows Event Log*, apenas os eventos de falha serão filtrados e analisados neste trabalho. Como a intenção é ter uma visão geral da confiabilidade de um sistema de software, este trabalho engloba tanto as falhas de sistemas operacionais, quanto de aplicações do usuário.

Nos sistemas analisados, o serviço *Windows Event Log* armazena dados sobre eventos do sistema em arquivos de extensão *evtx* (BOTT; STINSON, 2016). Portanto, a primeira etapa deste trabalho consiste no desenvolvimento de funcionalidades para leitura desse tipo de arquivo. Em seguida, como o foco é em avaliar a confiabilidade dos sistemas, foi necessário o desenvolvimento de um *parser* capaz de diferenciar eventos de falha e outros tipos de eventos armazenados no arquivo de *log*. Ao final, os eventos de falha são filtrados e armazenados em um banco de dados a fim de serem processados à *posteriori*.

A etapa seguinte consiste em realizar análises estatísticas sobre o conjunto de

falhas de cada computador investigado. Consequentemente, foi desenvolvida uma funcionalidade capaz de automatizar o cálculo de estatísticas descritivas e estimar métricas de confiabilidade a partir dos dados de falha armazenados no banco de dados.

Por fim, os principais resultados obtidos por meio das análises descritas na etapa anterior, são sumarizados em um relatório sobre a confiabilidade de software do computador analisado. Desta forma, foi desenvolvido uma funcionalidade de geração automática do relatório para o usuário no formato *pdf*. Os principais elementos do relatório são: número de identificação do computador, sistema operacional do usuário, tipos de falhas, período do dia em que as falhas ocorreram, falhas mais recorrentes, caraterística das falhas, métricas de confiabilidade, entre outros.

1.4 Demais capítulos

No capítulo 2 é feita a revisão da literatura relacionada ao tema deste trabalho, com intuito de preparar o leitor e prover o embasamento teórico necessário para os capítulos seguintes.

No capítulo 3 é apresentada a visão geral sobre o que é o software X-RAT, a definição de um relatório de confiabilidade e qual a contribuição deste trabalho ao X-RAT. É descrito passo a passo o funcionamento do software desenvolvido e as funcionalidades agregadas ao projeto, são elas: banco de dados de mensagens de falha, *parser* e Analisador.

No capítulo 4 são apresentados os resultados obtidos neste trabalho, incluindo o funcionamento do ponto de vista do usuário, e o que pode ser melhorado e incrementado em trabalhos futuros.

O capítulo 5 contém as conclusões técnicas sobre o trabalho, o que foi aprendido durante sua execução e as principais dificuldades encontradas.

Ao final do documento são apresentadas as referências bibliográficas.

2 Revisão da Literatura

Neste capítulo, serão apresentados conceitos básicos, referencial teórico, trabalhos relacionados e o estado da arte para uma melhor compreensão do trabalho.

2.1 Referencial teórico

O subsídio teórico utilizado para o planejamento do desenvolvimento deste trabalho se encontra na literatura da área de Engenharia de Confiabilidade de Software.

2.1.1 Engenharia de Confiabilidade de Software

Com ênfase na confiança do funcionamento de sistemas, a Engenharia de Confiabilidade de Software (SRE) é definida como o estudo quantitativo do comportamento operacional de sistemas baseados em software com respeito aos requisitos do usuário em relação à confiabilidade (IEEE, 1995), sendo esta uma área multidisciplinar, englobando diversas outras áreas como estatística, probabilidade e computação.

Os objetivos da SRE, em ordem de prioridade, são (O'CONNOR, 2011):

1. Aplicar conhecimento em engenharia e técnicas especializadas para prevenir ou reduzir a probabilidade ou frequência de falhas.

2. Identificar e corrigir as causas de falhas que ocorrem apesar dos esforços para preveni-las.

3. Determinar formas de lidar com falhas caso as suas causas não tenham sido corrigidas.

4. Aplicar métodos para estimar a provável confiabilidade de sistemas e analisar dados de confiabilidade.

Para melhor entendimento dos objetivos apresentados é preciso primeiramente definir o significado de termos como erros, falhas, faltas e tempo. Na seção seguinte serão definidos alguns dos principais conceitos da SRE.

2.1.2 Conceitos básicos

Os conceitos básicos adotados neste trabalho seguem a taxonomia descrita em (AVIZIENIS et al., 2004).

Serviço. Comportamento esperado de um software na perspectiva do usuário.

Falha. Uma falha é um evento que ocorre quando o serviço entregue não é o serviço correto. Um serviço falha quando não cumpre com sua especificação ou sua especificação não descreve corretamente sua função.

Erro. É o desvio do funcionamento correto. Erros podem ser transformados em outros erros (propagação de erro). A propagação de erro leva um sistema a falhar se um erro é propagado até a interface de serviço do sistema, ou seja, provocando o desvio do serviço em relação a sua especificação.

Falta. A causa de um erro é a ativação de uma falta (defeito ou *bug*). Uma falta é inicialmente considerada inativa, e sua ativação causa um erro que pode atingir ou não a interface do serviço.

Tempo. Quantidades de confiabilidade são, em sua maioria, definidas em respeito ao tempo. Sendo de maior interesse os três tipos definidos a seguir: (1) O tempo de execução de um software, que é o tempo que a CPU gasta executando o software. (2) O tempo do calendário, que é o tempo percebido pelo usuário em termos de anos, meses, semanas, dias, etc. (3) O tempo de relógio do início até o final da execução de um software.

Tempo entre falha TBF (*time between failure*) dado uma falha que ocorreu no tempo t1 e outra falha que ocorreu no tempo t2, o TBF é definido como a diferença de tempo entre t1 e t2.

2.1.3 Métricas de confiabilidade

Falhas de software formam um processo estocástico e distribuições de probabilidades podem ser utilizadas a fim de descrever o comportamento probabilístico de tais processos. Neste trabalho será utilizado o método Curve Fitting sobre o conjunto dos TBFs do *log* de falhas, com intuito de se descobrir qual distribuição teórica melhor se adere aos dados observados. Para tal, será feito um ranking contendo os valores dos testes de aderência Kolmogorov-Smirnov (CULLEN; FREY, 1999), Anderson-Darling (CULLEN; FREY, 1999) e Akaike information criterion (AKAIKE, 1974), e a distribuição que apresentar melhores resultados será escolhida.

Nesta seção serão apresentadas as definições das principais métricas de confiabili-

dade utilizadas neste trabalho, com base nos estudos (LYU, 1996) e (PHAM, 2007).

Função densidade de probabilidade *f(t)* (*probability density function*) define a função de probabilidade que representa a densidade de uma variável aleatória contínua situada entre um intervalo específico de tempo.

$$P(a < T < b) = \int_a^b f(t)\,dt$$

Confiabilidade *R(t)* (*reliability*) é a probabilidade de que o sistema irá operar sem falha no intervalo de zero até o tempo t, sob determinadas condições operacionais.

$$R(t) = P(T > t), t \geq 0$$

onde T denota o tempo até a falha.

Função de risco *h(t)* (*hazard rate*) é a probabilidade da primeira e única falha em determinado tempo t.

$$h(t) = \frac{f(t)}{R(t)}$$

Falta de confiabilidade *U(t)* (*unreliability*) é probabilidade de falha, ou seja, que o sistema irá falhar de zero até o tempo t, sob determinadas condições operacionais.

$$U(t) = P(T \leq t), t \geq 0$$

Tempo médio para falha *MTTF* (*mean time to failure*) é o valor esperado de t, *E(t)*, ou seja o tempo médio até a falha.

$$MTTF = \int_0^\infty t f(t)\,dt$$

Taxa de falha Definido como a probabilidade de uma falha por unidade de tempo, ocorrer no intervalo [t1, t2], dado que não tenha ocorrido falhas antes de t1.

$$Taxa\ de\ falha = \frac{R(t1) - R(t2)}{(t2 - t1)R(t1)}$$

Tempo médio para reparo *MTTR* (*mean time to repair*) é o valor esperado do tempo de reparo t, *E(t)*.

$$MTTR = \int_0^\infty tg(t)\,dt$$

Tempo médio entre falhas *MTBF* (*mean time between failure*) implica que o sistema falhou e foi reparado, é uma combinação entre as métricas *MTTR* e *MTTF*.

$$MTBF = MTTR + MTTF$$

Disponibilidade *A(t)* (*availability*) é a operação sem falha por um tempo determinado, onde não existe falha e reparo.

$$A(t) = \frac{MTTF}{MTTF + MTTR}$$

Warranty time é definido pelo maior TBF observado no conjunto de dados.

2.2 Trabalhos relacionados

Diversos trabalhos relacionados à confiabilidade de software foram desenvolvidos pelo grupo .SDLAB nos últimos anos, principalmente trabalhos focados em confiabilidade de sistemas operacionais. Ressalta-se que na maioria dos sistemas computacionais, o nível de software é dividido entre aplicações do usuário e o sistema operacional. Dada a natureza hierárquica dessas duas camadas de software, as aplicações do usuário dependem totalmente do sistema operacional para serem executadas. Devido a essa dependência técnica, falhas do SO podem prejudicar até mesmo a aplicação mais confiável, o que justifica a importância de tais trabalhos.

Em 2013, um dos primeiros artigos produzidos pelo .SDLAB (MATIAS; OLIVEIRA; ARAUJO, 2013) apresenta uma nova perspectiva para a análise da confiabilidade de sistemas operacionais. Diferente de outros trabalhos nesta área (GANAPATHI; GANAPATHI; PATTERSON, 2006), (SWIFT; BERSHAD; LEVY, 2003), (LI et al., 2008), (XU; IYER, 1999) que consideram como falhas de SO apenas eventos de falhas provenientes do núcleo do SO (espaço do *kernel*), o artigo propõe uma abordagem mais realística de análise de confiabilidade, considerando falhas de componentes do sistema operacional tanto no espaço do usuário quanto no espaço do *kernel*. Um sistema operacional moderno não é um programa monolítico único, mas é composto de muitas partes em execução.

O sistema operacional possui um *kernel* (geralmente um *microkernel*) que engloba os subsistemas mais críticos executados em modo privilegiado e vários outros programas integrados executados em modo não privilegiado (espaço do usuário). As partes do sistema operacional em execução no espaço do usuário executam suas funções em primeiro plano (por exemplo, aplicações do sistema operacional como gerenciador de janelas) ou em segundo plano (por exemplo, serviços do sistema operacional como *spooler* de impressão). Desta forma, as falhas analisadas foram classificadas em três categorias: Aplicação de SO ($\mathrm{SO_{APP}}$), Serviço de SO ($\mathrm{SO_{SVC}}$) e Kernel ($\mathrm{SO_{KNL}}$). Por fim, os autores observaram que, em média, falhas de $\mathrm{SO_{SVC}}$ (Grupo1: 78,28% e Grupo2: 77,72%) foram predominantes na amostra, enquanto que falhas $\mathrm{SO_{KNL}}$ (Grupo1: 6,85% e Grupo2: 0,15%) foram as menos observadas.

Posteriormente em (SANTOS et al., 2018), os autores mostram quantitativamente que analisar apenas falhas no espaço do *kernel* não é suficiente para avaliar com precisão toda a confiabilidade do SO, $R_{OS}(\mathrm{t})$. Eles compararam as estimativas de $R_{KNL}(\mathrm{t})$ e $R_{OS}(\mathrm{t})$, em que a primeira é a confiabilidade do sistema operacional calculada com base em apenas falhas do espaço do *kernel*, já a segunda é a confiabilidade do sistema operacional calculada com base nas falhas do *kernel* e do espaço do usuário, $R_{OS}(\mathrm{t}) = R_{KNL}(\mathrm{t}) \times R_{SVC}(\mathrm{t}) \times R_{APP}(\mathrm{t})$. Para todos os intervalos de tempo avaliados, t = 4 h, 8 h, 12 h, 24 h, 48 h, 168 h, 720 h, o $R_{OS}(\mathrm{t})$ foi significativamente menor que o $R_{KNL}(\mathrm{t})$, demonstrando a importância de não considerar apenas as falhas do *kernel* ao estimar a confiabilidade do sistema operacional.

Em (MATIAS et al., 2014), foi desenvolvida uma análise estatística dos tempos de falha do SO. Foram adotados quatro testes de aderência amplamente utilizados: Anderson Darling, Kolmogorov-Smirnov test, Chi-Squared test, e Likelihood ratio test. Os autores descobriram que as funções de densidade que melhor se ajustavam aos tempos de falha do SO eram Gamma e Weibull. Para os autores, este conhecimento pode ser utilizado em diferentes estudos na área de modelagem e simulação de confiabilidade de sistemas de computação.

Recentemente, foram desenvolvidas diferentes abordagens de detectar e caracterizar padrões de falhas de SO e de aplicações do usuário (SANTOS; MATIAS, 2018), (SANTOS; MATIAS; TRIVEDI, 2019). Padrões são associações de eventos de falha que se repetem, sistematicamente, na amostra de trabalho. Em primeiro lugar, buscou-se responder a seguinte pergunta de pesquisa: Falhas de software seguem padrões? Foram encontradas fortes evidências da existência de padrões nas amostras de falha analisadas, de modo que o trabalho avançou abordando demais questões de interesse, tais como: “Quais componentes de software mais contribuíram para a composição desses padrões?” “Quais falhas foram recorrentes nos padrões identificados?” “Quais tipos de relação entre falhas

estão presentes nos padrões identificados?" As causas de falha também foram consideradas, o que possibilitou responder outra importante pergunta de pesquisa: As causas de falha também seguem padrões? Fortes evidências indicam padrões consistentes também entre as causas de falha.

Por fim, foi desenvolvido um método para estimar as probabilidades de falha com base nos padrões de associação de falhas encontrados nos trabalhos anteriores (SANTOS; MATIAS; TRIVEDI, 2020). Essa abordagem é genérica o suficiente para ser aplicada a diferentes sistemas de software com pequenas alterações. Como em qualquer abordagem baseada em dados, ela deve conter o importante requisito de processar grandes conjuntos de dados com eficiência. Outro requisito importante para essa abordagem é a capacidade de realizar a descoberta de padrões e os cálculos de probabilidade de falha automaticamente, uma vez que meu objetivo final é usar essa abordagem para implementar previsão de falha *online*. Considerando a natureza dos registros de falha encontrados em *logs* de computador reais, nos quais na maioria das vezes os tipos de falha devem ser tratados como variáveis categóricas, três métodos são promissores para resolver este problema de pesquisa; Multinomial Logistic Regression (com e sem Ridge regularization), Decision Tree, e Random Forest. No entanto, planeja-se também incluir outros métodos (por exemplo, Neural Networks).

Todos os artigos citados nesta seção influenciaram o desenvolvimento deste trabalho.

2.3 Trabalhos correlatos

Dada a importância da confiabilidade como um fator crítico na qualidade de um software, pesquisas na área de Confiabilidade de Software deram origem a diversas ferramentas de análise de confiabilidade. No estudo "An overview and comparison of Software Reliability tools"(SASANKAR, 2016), é realizado o levantamento de diversas destas ferramentas. Um fato constatado foi que a maioria das ferramentas disponíveis possui apenas interface por linha de comando e não utilizam dos benefícios de uma interface gráfica. Nestas ferramentas, tornam-se útil exibições de informações como: número cumulativo de erros, curva de crescimento de confiabilidade e os resultados dos métodos estatísticos empregados a fim de determinar se o modelo é apropriado para o projeto a ser analisado.

Tarefas de uma ferramentas de confiabilidade software incluem (LYU, 1996):

- Coleta de falhas e tempo de teste.
- Cálculo de estimativas com modelos parâmetros utilizando as informações dis-

poníveis.

- Teste para ajustar modelos às informações coletadas.
- Selecionar um modelo a fim de fazer predições de faltas, tempo de teste, etc.
- Aplicar modelos.

De modo geral, estas ferramentas proporcionam análises de confiabilidade em detalhes, de forma que podem ser utilizadas posteriormente para melhorar a confiabilidade para aplicações em tempo real (SINGH, 2016).

Dentre as ferramentas conhecidas que estão disponíveis, têm-se (SASANKAR, 2016):

CASRE. Ferramenta que utiliza um conjunto de dados de falha e executa um modelo. A intensidade das falhas estimadas ou tempo entre falhas são desenhadas em um display gráfico, sendo permitido ao usuário controlar os displays de diversas formas (LYU; NIKORA, 1992).

SMERFS. Aplicação de software amplamente conhecida e aceita para avaliação de dados de teste para taxas de falha e predição de taxa de descoberta de defeitos. Utilizada um total de 16 diferentes modelos de crescimento de confiabilidade (FARR; SMITH, 1988).

SoftRel. Captura os efeitos das inter-relações entre atividades e caracteriza todos os eventos como processos Markovianos com funções de taxa de eventos definidas explicitamente (TAUSWORTHE, 1991).

SRMP. É uma ferramenta de linha de comando que contém 9 modelos e utiliza estimativa por máxima verossimilhança para calcular os parâmetros dos modelos. Provê métricas ao usuário como: funções de confiabilidade, taxa de falha, MTTF e os parâmetros de cada modelo (RELIABILITY; CONSULTANTS, 1988).

MEADEP. Ferramenta de avaliação de confiabilidade para análise de medição de sistemas críticos. O uso da ferramenta em dados de falha permite avaliações quantitativas da confiabilidade do sistema analisado sem a necessidade de grandes habilidades em análise de dados e modelagem de sistemas do usuário (TANG et al., 1998).

Sorel. É uma ferramenta para análise e predição de confiabilidade, implementa um método global para avaliação a partir modelos de crescimento de confiabilidade e testes de crescimento de confiabilidade. Suas principais métricas são: MTTF, função de

intensidade, número cumulativo de falhas, taxa residual de falha (KANOUN et al., 1993).

SREPT. Implementa diversas técnicas que permite que a ferramenta seja usada em todos os estágios de vida de um software, desde o desenvolvimento até o estágio operacional. Permite monitorar a qualidade do desenvolvimento do software a partir de um *framework* para predição e estimativa de confiabilidade de software, combinando a habilidade de diversas ferramentas em um *framework* unificado (TRIVEDI, 2002).

SRTpro. Foi desenvolvido como uma ferramenta de medição de confiabilidade de software focada para a fase de desenvolvimento do software. Possui modelos de predição e de estimativa a fim de lidar com a confiabilidade nas fases iniciais do ciclo de vida do software, como levantamento de requisitos, design, codificação e fases de teste (KANG; GU; BAIK, 2009).

A abordagem deste trabalho se diferencia das ferramentas citadas no fato de que, como dito anteriormente, cálculos de probabilidade de falha e descoberta de padrões serão implementados para serem realizados de forma automática e não interativa, em forma de um serviço *online*, sendo inicialmente aplicado para *logs* de falha dos sistemas operacionais Windows 10 e Windows 11 e disponibilizando os resultados em formato *pdf*.

3 Plataforma X-RAT

Nesta seção, são apresentados os aspectos gerais presentes na plataforma X-RAT, desde a funcionalidade da plataforma, incluindo o desenvolvimento e a arquitetura do projeto, até o relatório de confiabilidade obtido ao final do processo.

3.1 Visão geral

A plataforma X-RAT, desenvolvida em conjunto pelo grupo .SDLAB, é parte de uma pesquisa maior na área de confiabilidade de software, e é uma solução projetada com o intuito de avaliar a confiabilidade de sistemas de forma automatizada, a partir de uma interface simplificada. Ela pode ser utilizada por empresas, provedores de serviços, organizações ou pessoas que desejam medir a confiabilidade de seus sistemas.

Como apresentado na Seção 1.1, a plataforma consiste em uma ferramenta capaz de analisar *logs* de falhas de diferentes aplicações de software, que por meio de *upload*, serão carregados remotamente por voluntários em um sistema coletor de dados, e estes dados serão analisados pela *engine*, a fim de ao final do processo gerar um relatório de confiabilidade sobre a aplicação de software avaliada.

3.2 Relatórios de Confiabilidade

Os relatórios de confiabilidades são documentos que têm como objetivo fornecer informações ao usuário a respeito da confiabilidade do software analisado. Estes relatórios são importantes pois permitem aos desenvolvedores ou provedores de serviço entender como o software está funcionando em condições reais de uso e identificar quaisquer problemas de confiabilidade que precisem ser resolvidos, além de avaliar o desempenho final do software e facilitar a tomada de decisões.

Um exemplo de um caso de uso de um relatório de confiabilidade no desenvolvimento de um produto, seria na verificação da seguinte hipótese: Dado uma determinada aplicação de software, esta aplicação é confiável o suficiente para que possa ser utilizada em produção sem que seu funcionamento seja afetado por falhas críticas de software?

3.3 Projeto Desenvolvido

O foco deste trabalho concentra-se no escopo da *engine* de análise, sendo contempladas as falhas do Microsoft Windows 10 (Win10) e 11 (Win11). Após a coleta dos dados, as falhas são filtradas, classificadas e analisadas, empregando diferentes técnicas estatísticas, a fim de estimar métricas de confiabilidade e gerar um relatório de confiabilidade sobre a aplicação de software avaliada. Detalhes do projeto serão descritos nas etapas a seguir.

A arquitetura do projeto da *engine* de análise é dividida em 2 módulos principais, separados por suas funcionalidades. O primeiro módulo o *parser*, responsável por efetuar a leitura dos eventos de falha presentes no arquivo em formato *evtx*, padronizar os dados, encontrar a mensagem de falha referente ao evento, classificar o tipo de falha e armazenar os eventos em estrutura previamente definida no banco de dados, que será discutida na Seção 3.3.2.

O segundo módulo é o Analisador, que é responsável por calcular métricas baseadas na data e horário de falha dos eventos, calcular o TBF das falhas, e posteriormente realizar testes de aderência sobre uma lista de distribuições de probabilidade candidatas, a fim de determinar qual distribuição melhor se adere ao conjunto de dados de falha. A distribuição de probabilidade escolhida é utilizada para o cálculo das métricas de confiabilidade que serão discutidas na Seção 3.3.3.

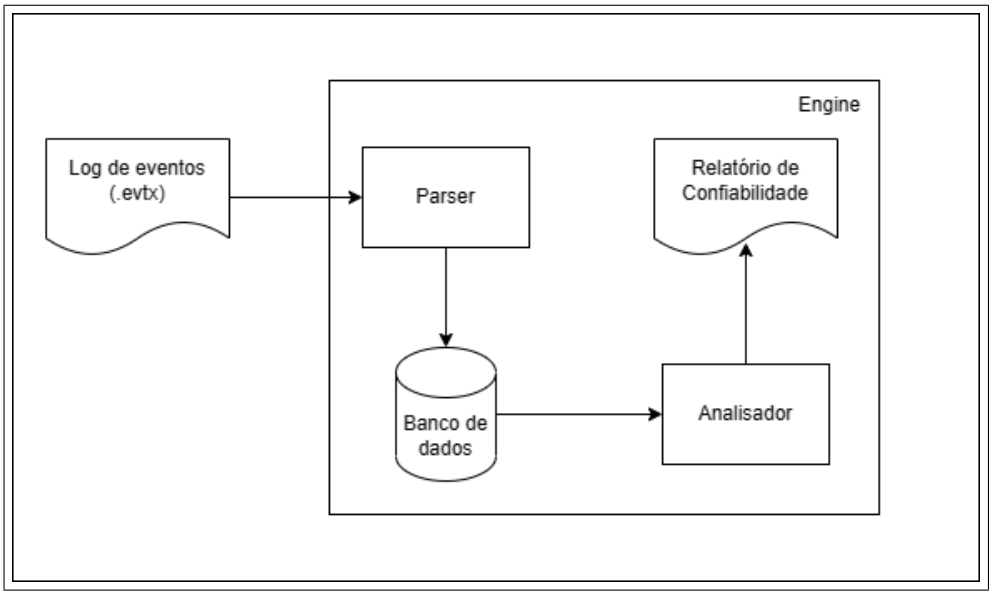


Figura 1 – Arquitetura da Engine de análise.

A última etapa, também realizada no Analisador, é a geração do relatório de confiabilidade, em formato *pdf*, contendo informações sobre o conjunto de dados analisado,

as métricas estimadas, falhas mais recorrentes, o ranking de distribuições, um gráfico de densidade das falhas por minutos e o plot de gráficos.

3.3.1 Banco de dados de mensagens de falha

Um evento de um *log* de falhas do Microsoft Windows 10 e 11 contém os seguintes campos (MICROSOFT, 2022):

- **Provider:** Nome da aplicação que publicou o evento.
- **EventID:** Código atribuído a cada tipo de atividade auditada.
- **Version:** Versão da falha.
- **Level:** Nível de severidade do evento em questão.
- **Task:** Usado para representar subcomponente ou categoria sobre a aplicação que publicou o evento.
- **Opcode:** Atribuído pelo componente que gerou o evento.
- **Keywords:** Conjunto de categorias que pode ser usado para filtrar ou procurar por eventos, por exemplo: "Network", "Security"ou "Resource not found".
- **TimeCreated:** *Timestamp* contendo o tempo em que o evento foi criado.
- **EventRecordID:** Número de identificação do evento em relação ao log.
- **Correlation:** Identificador utilizado para especificar correlações entre eventos.
- **ProcessId:** Número de identificação do processo que gerou o evento.
- **ThreadId:** Número de identificação da thread que gerou o evento.
- **Channel:** Canal em que o evento foi publicado.
- **Computer:** Nome do computador em que ocorreu o evento.
- **UserId:** Número de identificação do usuário.
- **EventData:** Contém informações especificadas pela aplicação ao qual o evento foi publicado.

- **Message:** Mensagem descritiva gerada pela aplicação que publicou o evento. Possui variáveis que são preenchidas com dados do EventData.

No software Event Viewer do Microsoft Windows, a mensagem de falha (campo Message) é atribuída ao evento a partir de uma consulta a aplicação ao qual a falha se originou. Caso o usuário procure por uma falha na qual a aplicação não está mais disponível na máquina em uso ou esteja visualizando a falha de outro computador, a mensagem não será exibida. A Figura 2 contém um exemplo de uma mensagem de falha, na área circulada em vermelho.

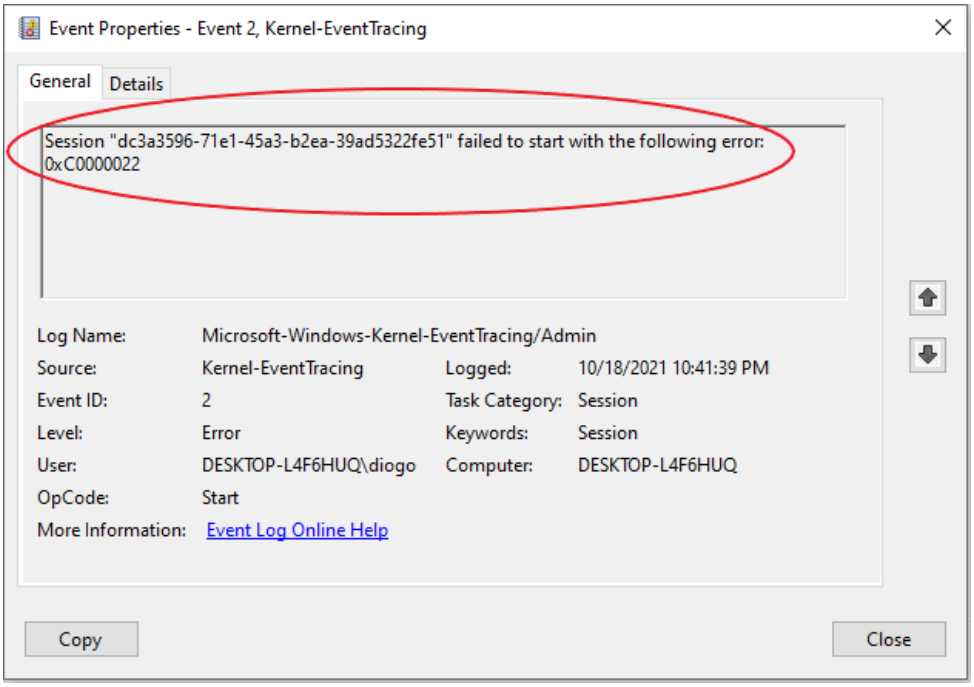


Figura 2 – Mensagem de falha.

A primeira etapa deste projeto se deu pela construção de uma base de dados de mensagens de falha, visto que estas não são armazenadas nos *logs* de falhas, mas sim obtidas por meio de consulta em tempo real ao registro da aplicação que publicou o evento de falha. As mensagens de falha são utilizadas durante a etapa de classificação das falhas e cada evento de falha é atribuído a sua respectiva mensagem pelos campos EventID e Provider.

A Figura 3 descreve visualmente o processo de obtenção da mensagem de falha pelo software Event Viewer, que utiliza dos campos Provider e EventID como identificadores do evento e realiza a consulta no registro da aplicação pela mensagem referente ao evento desejado. O campo EventData contém informações específicas do evento de falha que são utilizadas para preencher campos variáveis da mensagem.

Para a construção de uma base de dados de mensagens de falhas, foi necessária a

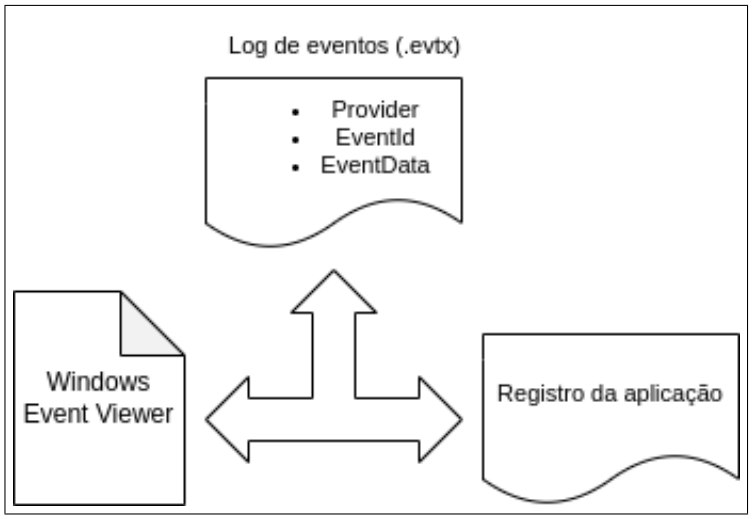


Figura 3 – Como o Event Viewer mostra uma mensagem de falha.

utilização de um software chamado Event Sentry (NETIKUS, 2022), que mapeia todas as aplicações disponíveis no computador local e gera um arquivo no formato *txt* contendo mensagens de falha presentes em cada aplicação individual.

A partir da obtenção dos arquivos contendo o mapeamento das mensagens de falha por aplicação (Provider), foi desenvolvido um algoritmo em Python responsável pela leitura, *parsing* e armazenamento das mensagens no banco de dados. Posteriormente essas mensagens são utilizadas pelo módulo descrito na Seção 3.3.2.

3.3.2 Parser

A extensão de arquivos *evtx* é um formato desenvolvido pela Microsoft, de forma que os eventos de *log* são gravados no arquivo em formato *xml* e codificados em binário com intuito de serem posteriormente decodificados pela aplicação Event Viewer quando requisitada leitura. Por este motivo foi necessária a implementação de um módulo capaz de decodificar os arquivos *evtx*, realizar o *parsing* dos campos *xml* e aplicar algoritmos de tratamento de dados para que ao final do processo os eventos de *log* possam ser visualizados em formato fidedigno a representação apresentada no Event Viewer.

A primeira etapa do módulo *parser* é a decodificação do arquivo *evtx* gravado em formato binário para *xml*. Para tal, foi utilizada a biblioteca do Python pyevtx-rs (BENAMRAM, 2022), que realiza a decodificação do arquivo e retorna um vetor iterável onde cada elemento é um evento representado em formato *xml*, como descreve a Figura 4.

A etapa seguinte se deu pela implementação de um algoritmo capaz de extrair as informações de cada um dos campos do evento da estrutura *xml* correspondente. Nesta etapa foi a utilizada a biblioteca lxml (TEAM, 2022), que fornece uma interface para

```
- <Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
  - <System>
      <Provider Name="Microsoft-Windows-Kernel-EventTracing" Guid="{b675ec37-bdb6-4648-
        bc92-f3fdc74d3ca2}" />
      <EventID>2</EventID>
      <Version>0</Version>
      <Level>2</Level>
      <Task>2</Task>
      <Opcode>12</Opcode>
      <Keywords>0x8000000000000010</Keywords>
      <TimeCreated SystemTime="2021-10-19T01:41:39.3169064Z" />
      <EventRecordID>207</EventRecordID>
      <Correlation />
      <Execution ProcessID="10900" ThreadID="23452" />
      <Channel>Microsoft-Windows-Kernel-EventTracing/Admin</Channel>
      <Computer>DESKTOP-L4F6HUQ</Computer>
      <Security UserID="S-1-5-21-3596145874-1425323603-3254715764-1001" />
    </System>
  - <EventData>
      <Data Name="SessionName">dc3a3596-71e1-45a3-b2ea-39ad5322fe51</Data>
      <Data Name="FileName" />
      <Data Name="ErrorCode">3221225506</Data>
      <Data Name="LoggingMode">4194560</Data>
    </EventData>
  </Event>
```

Figura 4 – Estrutura de um evento do Win10 no arquivo de log.

leitura e extração de textos *xml*, sendo cada evento de *log* do vetor iterável processado individualmente em um *loop*. Para cada evento de *log*, informações adicionais que não estão presentes nos campos do evento, mas estão disponíveis no Event Viewer, são usadas. Entre elas estão: nome da categoria do evento, que se baseia nos campos Task e Provider, e nome do tipo do evento que se baseia no campo Level.

Na terceira etapa do módulo de *parsing*, optou-se pela escolha da biblioteca Pandas (TEAM, 2022) para armazenamento em memória dos eventos de falha extraídos na etapa anterior, em formato *dataframe*. O Pandas (TEAM, 2022) é uma biblioteca responsável por fornecer uma interface para operações em grandes conjuntos de dados com boa *performance*, sendo esta interface útil nas próximas etapas.

Um dos problemas encontrados ao longo do desenvolvimento do *parser* se deu pelo fato de que a Microsoft converte o horário do evento do *log* para o *timezone* UTC, perdendo-se o horário local em que ocorreu o evento. Porém é possível encontrar qual o *timezone* do computador do usuário por meio de um evento informativo que armazena esta informação, sendo este evento representado pelo campo EventID número 6013.

Na quarta etapa, é realizada a busca pelo evento com campo EventID número 6013, e é extraído do campo EventData qual o *timezone* do computador em que ocorrem as falhas. É criado então um novo campo Time, que consiste na data e horário disponíveis

no campo TimeCreated porém aplicando o *timezone* encontrado, para se obter o horário local.

Por último, as seguintes operações são realizadas nesta etapa de *parsing*:

- É realizada uma consulta no banco de dados pelas mensagens de falha, que posteriormente são mapeadas aos eventos que estão sendo processados em um novo campo Message.

- É aplicado um algoritmo que realiza a classificação das falhas entre as categorias: Aplicação de usuário (USER_{APP}), Aplicação de SO (SO_{APP}), Serviço de SO (SO_{SVC}) e Kernel (SO_{KNL}) (MATIAS; OLIVEIRA; ARAUJO, 2013). A primeira categoria contém falhas em aplicações de usuário, que são programas com os quais os usuários interagem diretamente (Exemplo: notepad.exe). A segunda categoria contém falhas causadas pelo mau funcionamento de aplicativos do sistema operacional, os quais são executados sob demanda do usuário (Exemplo: explorer.exe). A terceira categoria inclui falhas de serviços do sistema operacional que, geralmente, executam em segundo plano e com mínima ou nenhuma interação do usuário (Exemplo: MsInstaller). A quarta categoria possui falhas causadas pelos subsistemas do kernel do SO, as quais exigem a reinicialização do sistema na maioria dos casos (Exemplo: DRIVER_POWER_STATE_FAILURE, que indica que um driver está em um estado de energia inconsistente ou inválido).

- Os eventos de falha são armazenados no banco de dados.

3.3.3 Analisador

O Analisador representa a fase final da *engine* de análise, seu código fonte foi implementado na linguagem Python e utiliza da biblioteca rpy2 (GAUTIER, 2022), que fornece uma interface para executar algoritmos escritos na linguagem R embutidos no código fonte Python. Esta interoperabilidade entre as linguagens permite com que sejam explorados os pontos fortes da linguagem R no quesito de análise estatísticas de dados, cujo os algoritmos implementados serão descritos a seguir.

A primeira etapa realizada no Analisador é a leitura dos eventos do *log* no banco de dados, em seguida os eventos são armazenados em um *dataframe* do Pandas (TEAM, 2022), a fim de se utilizar a interface disponibilizada pela biblioteca para manipulação de dados. Na sequência são realizadas as primeiras operações sobre os dados, que consiste em recolher informações úteis sobre o *dataset* em questão, como: ID do computador em que ocorreram as falhas, primeira e última data de ocorrência de um evento, sistema operacional e tipos de eventos.

Na segunda etapa, os eventos de falha são identificados e separados dos eventos informacionais e de *warning*, com base no campo Level, que possui como valor um inteiro variando de 0 a 4. Os eventos informacionais são representados pelo valor 4, eventos de *warning* possuem valor 3, eventos de falha são representados pelos valores 1 e 2, e eventos que representam ações executadas com sucesso possuem valor 0. As falhas são posteriormente separadas em dois grupos, a partir dos tipos de falha obtidos durante a etapa de classificação (MATIAS; OLIVEIRA; ARAUJO, 2013). O primeiro grupo consiste nas falhas de Aplicação de usuário (USER_{APP}) e o segundo nas falhas de Aplicação de SO (SO_{APP}), Serviço de SO (SO_{SVC}) e Kernel (SO_{KNL}).

Em seguida são calculadas as seguintes métricas de confiabilidade: quantidade e porcentagem do número total de falhas por grupo, falhas por horas do dia de cada grupo e número de falhas por dia da semana de cada grupo. É também analisada qual a fonte das falhas (Provider) mais recorrente de cada grupo.

Na terceira etapa é realizado o cálculo dos TBFs do conjunto de falhas, que é utilizado como entrada para o algoritmo implementado em R, que aplica o método Curve Fitting para as distribuições de probabilidade candidatas Normal, Lognormal, Weibull, Exponencial e Gamma. Após a aplicação do Curve Fitting, são realizados os testes de aderência Kolmogorov-Smirnov, Anderson-Darling e Akaike information criterion sobre as distribuições obtidas.

Após a realização dos testes de aderência, os resultados são utilizados na construção de um ranking com os valores referentes a cada uma das distribuições candidatas, com o intuito de se escolher a distribuição que melhor se adequa ao conjunto de dados (TBFs). A distribuição escolhida é utilizada para o cálculo das métricas de confiabilidade, sendo neste trabalho calculadas as métricas *MTBF*, *Warranty time*, *R(t)*, *h(t)*, com t = 1, 2, 4, 8, 12, 24; os valores t representam o tempo de missão em horas. A figura 5 descreve as etapas do Analisador descritas anteriormente.

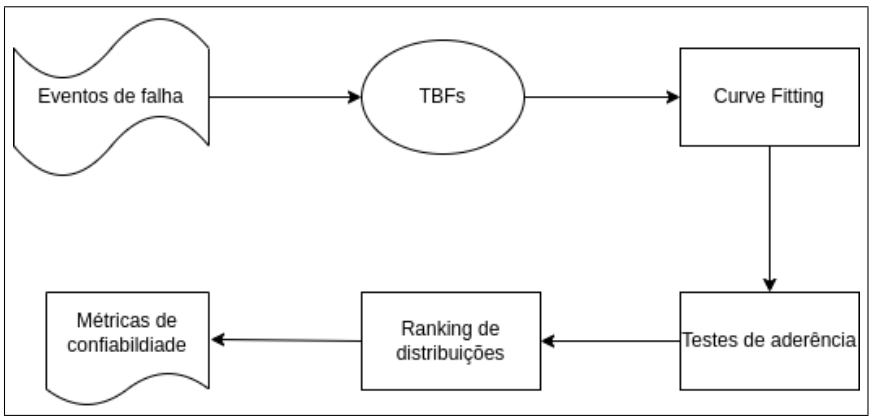


Figura 5 – Etapas do Analisador.

A última etapa realizada pelo módulo do Analisador, é a exportação do relatório de confiabilidade em formato *pdf*, contendo todas as informações obtidas nas etapas anteriores, junto com o histograma de densidade dos TBFs, e o gráficos referentes a distribuição escolhida: histograma de densidades empírica e teórica, Q-Q plot e P-P plot.

Para gerar o relatório de confiabilidade foi utilizada a biblioteca reportlab (ROBINSON, 2022), disponível para a linguagem Python, que permite desenhar formas, plotar gráficos e exportar arquivos em formato *pdf*.

4 Resultados

Neste capítulo é apresentado um exemplo completo do uso da plataforma X-RAT, as funcionalidades que este trabalho trouxe para a plataforma e itens que podem ser desenvolvidos em trabalhos futuros.

4.1 Caso de uso

Nesta seção é descrito um caso de uso completo da plataforma X-RAT. O primeiro passo é acessar o *website* da plataforma[2] e fazer o *upload* dos arquivos de *log*. Como citado na Seção 1.1, o *upload* é realizado de duas formas: "manualmente"por meio do website, ou "automaticamente"por meio do *download* de uma aplicação de software que automatiza o processo de coleta no computador do usuário.

Como o sistema coletor não faz parte do escopo do trabalho, foi optado apenas pelo caso de uso do *upload* manual, via *website* cujo interface é descrita na Figura 6.

Figura 6 – Interface para coleta de logs de falha.

Após o usuário selecionar os arquivos de *log* de sua máquina, a tela seguinte corresponde a uma pesquisa sobre o perfil do usuário que utiliza o computador, como descrito na Figura 7.

[2] <https://x-rat.facom.ufu.br/>

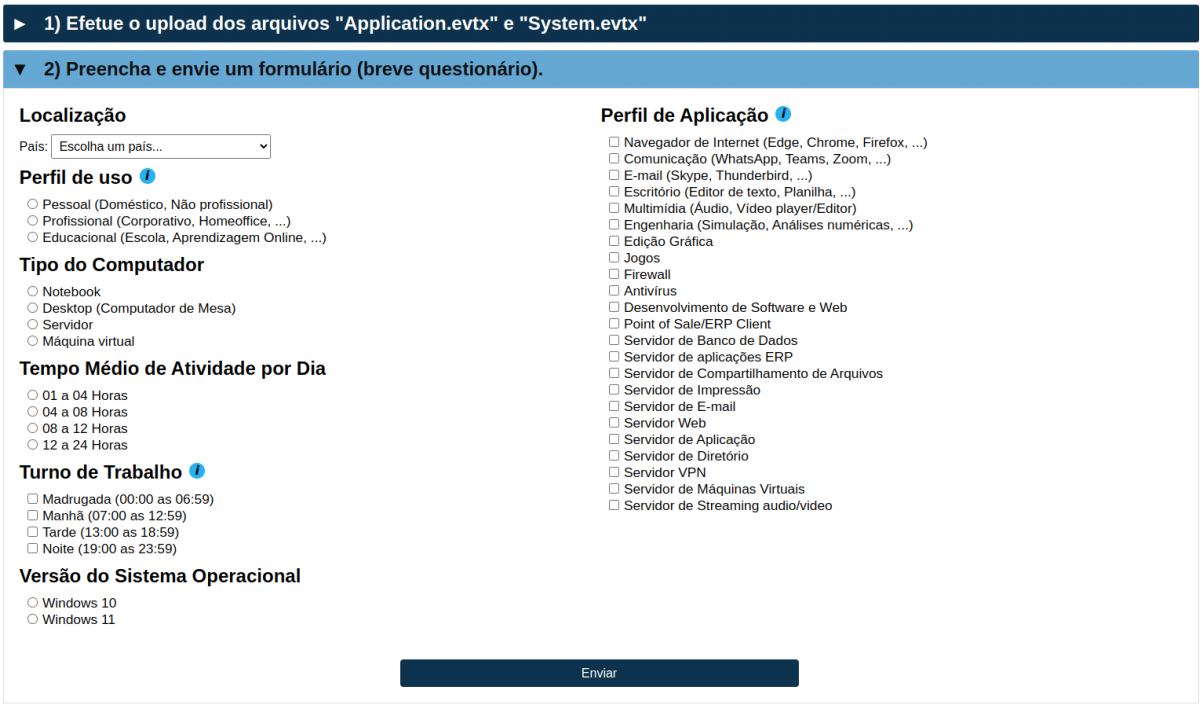


Figura 7 – Questionário sobre o perfil do usuário.

Após o envio dos dados para o servidor da aplicação, a *engine* de análise será acionada e realizará o processamento dos *logs*, a fim de ao final do processo, retornar para o usuário o relatório de confiabilidade em formato *pdf*, conforme mostra a Figura 8.

A primeira parte do relatório de confiabilidade tem como intuito exibir características do conjunto de falhas analisado. No primeiro quadro, exibido na Figura 9, estão disponíveis as informações sobre a quantidade de falhas de cada grupo (Aplicação e SO) e a porcentagem que estas quantidades representam, também está incluso as partes do dia e os dias da semana em que as falhas ocorreram. O segundo quadro contém o nome da aplicação responsável pelas falhas mais recorrentes em cada grupo, como mostra a Figura 10.

O terceiro quadro, mostrado na Figura 11, exibe o ranking de distribuições, juntamente com os parâmetros gerados pelo método Curve Fitting para cada distribuição, e os resultados dos testes de aderência. Também contém os gráficos: histograma do conjunto de TBFs, histograma de densidades empírica e teórica, Q-Q plot e P-P plot.

O quarto e último quadro, exibido na Figura 12, contém as métricas de confiabilidade: *MTBF*, *Warranty time*, *R(t)* e *h(t)*, com tempo de missão em horas.

Software Dependability Laboratory

(sdlab.facom.ufu.br)

Reliability Analysis Report

System identification:

Computer ID: desktop-j7gvk15
Operating System: Win10/...

Dataset:

Failure types: Operating system and Applications
Sample period: 07/26/2021 - 04/05/2022

No. of failures: OS 913 (96.1%) Applications 37 (3.9%)

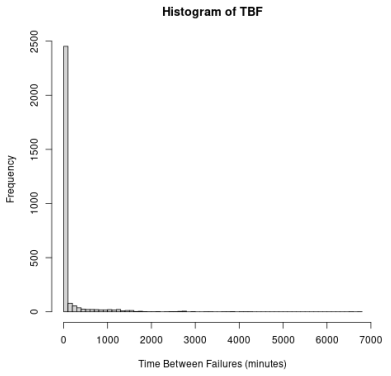
No. of failures per Parts of Day:
12 am to 06 am: OS (19.3%) App. (24.3%)
06 am to 12 pm: OS (7.1%) App. (2.7%)
12 pm to 06 pm: OS (40.5%) App. (27.0%)
06 pm to 12 am: OS (33.1%) App. (45.9%)

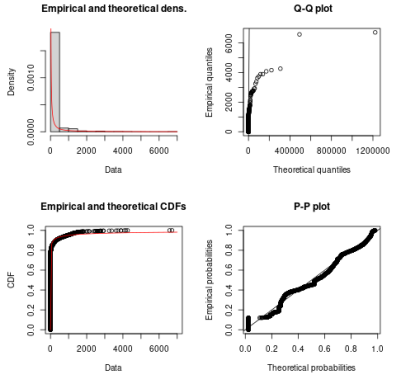
No. of failures per Days of Week:
Sun: OS (16.8%) App. (5.4%)
Mon: OS (7.7%) App. (0.0%)
Tue: OS (10.8%) App. (35.1%)
Wed: OS (13.1%) App. (10.8%)
Thu: OS (14.2%) App. (16.2%)
Fri: OS (15.7%) App. (5.4%)
Sat: OS (21.7%) App. (18.9%)

Most recurrent failures (OS vs APP):

OS	APP
service control manager	application error
microsoft-windows-ndis	application hang
vss	
volsnap	
volmgr	
microsoft-windows-windowsupdateclient	
application error	
schannel	
microsoft-windows-wer-systemerrorreporting	
microsoft-windows-driverframeworks-usermode	

	Distribution	Params	Values	AIC	KS	AD	AIC_rank	KS_rank	AD_rank
1	weibull	scale, shape	0.62897288372756, 0.149069255256104	-5996.761602	0.095960	36.892442	1	1	1
2	gamma	shape, rate	0.0823340929688964, 0.000650800548011519	-5980.155666	0.119655	48.113322	2	2	2
3	lnorm	meanlog, sdlog	-4.55600611669301, 9.24326301721107	-5252.221599	0.140261	115.926296	3	3	3
4	norm	mean, sd	126.512021571683, 440.901871993335	43218.671576	0.402211	nan	5	4	4
5	exp	rate	0.00790438718928634	33618.981705	0.726940	nan	4	5	5





MTBF: 126.5 minutes
Warranty time: 6713.0 minutes

	h(t)	R(t)
1 hours:	h(1) = 0.455833326	R(1) = 0.139065399
2 hours:	h(2) = 0.352464825	R(2) = 0.112190263
4 hours:	h(4) = 0.272537013	R(4) = 0.088417679
8 hours:	h(8) = 0.210734286	R(8) = 0.067899380
12 hours:	h(12) = 0.181301154	R(12) = 0.057422697
24 hours:	h(24) = 0.140187818	R(24) = 0.042073150

Figura 8 – Relatório de confiabilidade.

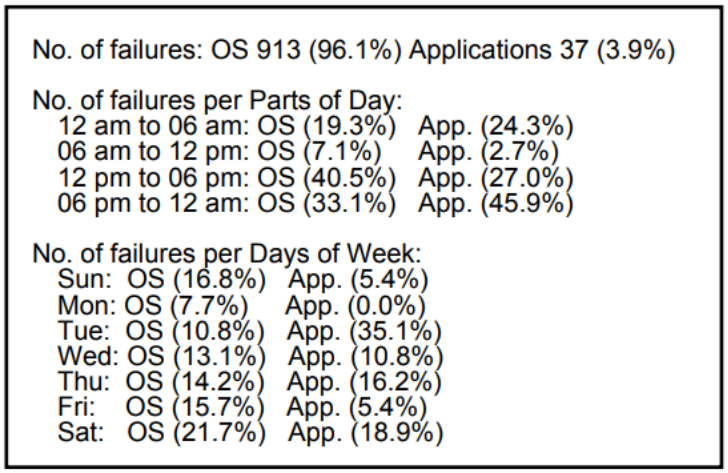


Figura 9 – Características das falhas.

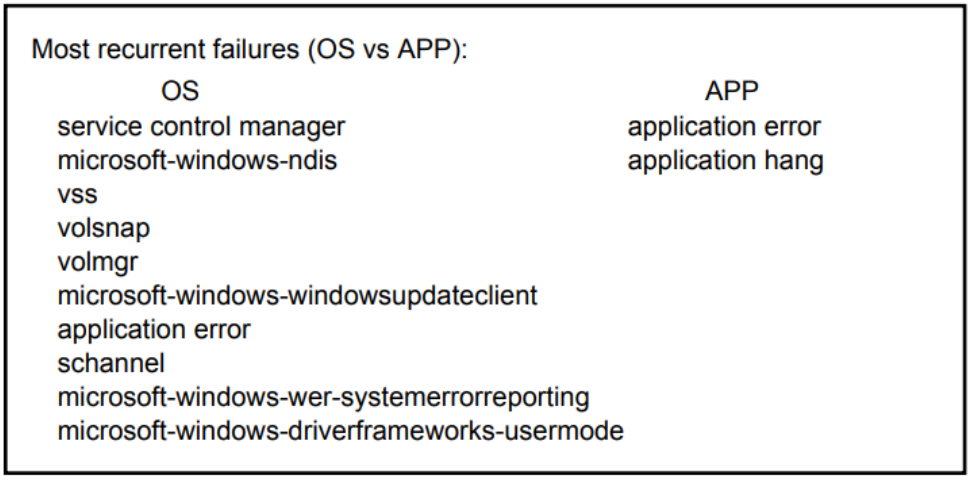


Figura 10 – Falhas mais recorrentes.

4.2 Resultados alcançados

A execução deste trabalho teve como resultado a adição de quatro novas funcionalidades para a *engine* de análise da plataforma X-RAT, divididas em módulos, que são descritas em detalhes no capítulo 3.

A primeira nova funcionalidade adicionada a plataforma foi um banco de dados de mensagens de falhas, construído com a finalidade de retirar a dependência do registro de mensagens da aplicação que publicou o evento, como faz o software Event Viewer. Desta forma é possível obter todos os campos do evento de falha que serão necessários para as próximas etapas.

A segunda funcionalidade adicionada foi o módulo *parser*, que realiza a leitura do *log* de falhas, faz *parsing* dos campos, classifica os eventos à partir das categorias obtidas no estudo (MATIAS; OLIVEIRA; ARAUJO, 2013), vincula o evento a sua respectiva

	Distribution	Params	Values	AIC	KS	AD	AIC_rank	KS_rank	AD_rank
1	weibull	scale, shape	0.62897288372756, 0.149069255256104	-5996.761602	0.095960	36.892442	1	1	1
2	gamma	shape, rate	0.0823340929688964, 0.000650800548011519	-5980.155666	0.119655	48.113322	2	2	2
3	lnorm	meanlog, sdlog	-4.55600611669301, 9.24326301721107	-5252.221599	0.140261	115.926296	3	3	3
4	norm	mean, sd	126.512021571683, 440.901871993335	43218.671576	0.402211	nan	5	4	4
5	exp	rate	0.00790438718928634	33618.981705	0.726940	nan	4	5	5

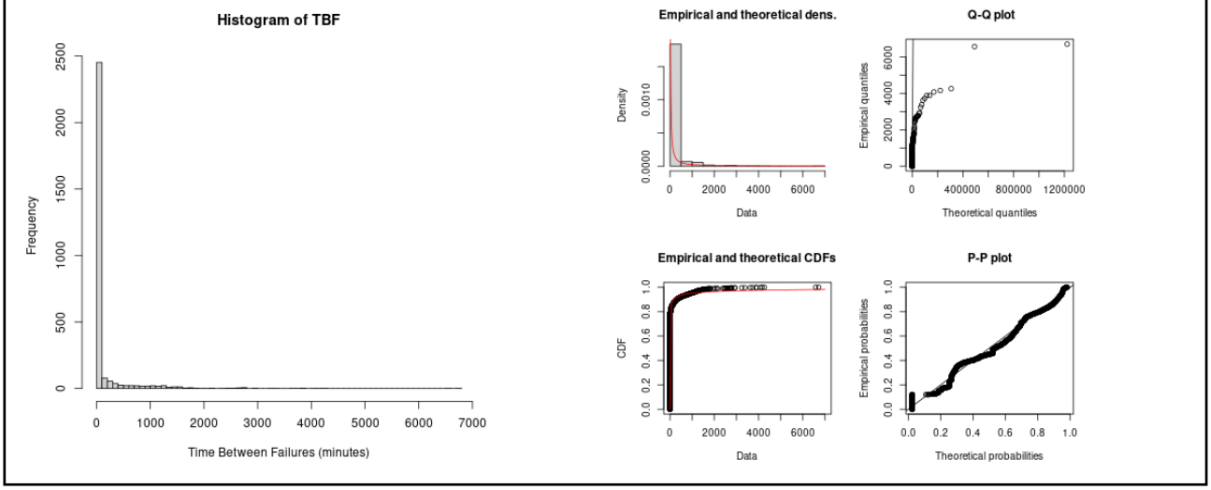


Figura 11 – Ranking de distribuições.

MTBF: 126.5 minutes
Warranty time: 6713.0 minutes

	h(t)	R(t)
1 hours:	h(1) = 0.455833326	R(1) = 0.139065399
2 hours:	h(2) = 0.352464825	R(2) = 0.112190263
4 hours:	h(4) = 0.272537013	R(4) = 0.088417679
8 hours:	h(8) = 0.210734286	R(8) = 0.067899380
12 hours:	h(12) = 0.181301154	R(12) = 0.057422697
24 hours:	h(24) = 0.140187818	R(24) = 0.042073150

Figura 12 – Métricas de confiabilidade.

mensagem, realiza um procedimento para descoberta do *timezone* do computador do usuário em que a falha ocorreu à fim de se obter o horário local da falha, dado que nos arquivos de falha *evtx* a hora é armazenada em horário UTC, e por fim armazena os eventos no banco de dados.

A terceira funcionalidade é o módulo Analisador, responsável por ler os eventos armazenados no banco de dados pelo *parser*, filtrar quais são os eventos de falhas e quais são eventos informativos ou de *warning*, separar as falhas em dois grupos (falhas de aplicação e de SO), gerar métricas simples sobre as características de cada grupo,

transformar as falhas em um conjunto de TBFs, aplicar o método Curve Fitting sobre os TBFs utilizando um conjunto de distribuições pré-definidas, realizar testes de aderência a fim de criar um ranking de distribuições para escolher do melhor *fit*, e então calcular métricas de confiabilidade de software.

A quarta funcionalidade é um relatório de confiabilidade em formato *pdf* que possui como intuito disponibilizar ao usuário diversas as informações obtidas nas etapas anteriores, como métricas de confiabilidade, características dos grupos de falha, ranking das distribuições candidatas e gráficos informativos.

4.3 Trabalhos futuros

Apesar dos resultados obtidos neste trabalho, a plataforma X-RAT ainda encontra-se em estado inicial de desenvolvimento, sendo possível a adição de diversas novas funcionalidades que possam ser úteis no contexto de uma ferramenta de análise de confiabilidade. Portanto, o objetivo para trabalhos futuros consiste na adição de novos recursos para a plataforma, tanto no escopo da *engine* de análise quanto no escopo do sistema coletor de dados.

Considerando o escopo da *engine*, existe a possibilidade de atuação no contexto das duas fases principais, o *parser* e o Analisador. Trabalhos futuros podem empregar técnicas estudadas nos trabalhos (SANTOS; MATIAS; TRIVEDI, 2017) (SANTOS; MATIAS; TRIVEDI, 2021) a fim de se obter também a causa da falha, que implementada na fase de *parsing*, permitirá o armazenamento das causas em um novo campo na tabela dos eventos no banco de dados, que posteriormente serão mostradas no relatório de confiabilidade.

Pode-se também considerar para trabalhos futuros expandir a quantidade de métricas de confiabilidade abordados no relatório, o que resulta no trabalho de implementar novos algoritmos para o cálculo de métricas e estatísticas que serão posteriormente adicionados ao código fonte do Analisador.

Também é possível para trabalhos futuros atuar no contexto do coletor de dados em conjunto com o módulo *parser*, a fim de empregar técnicas de *parsing* e algoritmos de coleta para adicionar suporte a novos tipos de *logs* de falha, seja de outros sistemas operacionais como Linux ou falhas de alguma aplicação específica, dado que inicialmente apenas são suportados *logs* de falha dos sistemas operacionais Win10 e Win11.

5 Conclusão

Neste capítulo são apresentados as conclusões técnicas sobre o trabalho, o aprendizado obtido e as principais dificuldades encontradas ao longo do processo.

5.1 Técnica

Durante a fase de implementação do *parser*, inicialmente foi utilizada a biblioteca python-evtx (BALLENTHIN, 2021) para decodificação e leitura dos arquivos de *log* em formato *evtx*. No entanto, foi observado que a performance do Python para este tipo de tarefa não era apropriada para este tipo de aplicação online, cujo objetivo é logo após receber os logs emitir o relatório, dado que um único arquivo de *log* demandava de 1 a 2 minutos de processamento. Note que um tempo mais longo nesse ponto do programa poderia motivar o usuário a se desconectar do sistema antes mesmo de receber o relatório.

Para mitigar os problemas de performance do Python, a biblioteca pyevtx-rs (BENAMRAM, 2022) fornece uma interface Python e realiza o processamento dos dados na linguagem Rust. Com o uso desta biblioteca, o ganho de performance foi significativo, sendo necessário apenas alguns segundos para a decodificação e leitura dos arquivos de *log*.

Optou-se também para realização das técnicas de Curve Fitting e testes de aderência sobre as distribuições candidatas o uso da linguagem R, por meio da biblioteca rpy2 (GAUTIER, 2022), que fornece uma interface Python para executar código R. Esta escolha se deu pela facilidade proveniente da linguagem R para realizar análise estatísticas de dados reais, enquanto no Python, as funcionalidades que seriam utilizadas estavam dispersas em diferentes bibliotecas, em que a maioria delas foram escritas visando casos de uso diferentes, além de nem sempre fornecerem a saída necessária.

5.2 Aprendizado

Dado que o estudo da confiabilidade de software não é comum nas matérias da graduação, elaborar este trabalho abriu as portas para uma série de conhecimentos que foram adquiridos em decorrência dos estudos realizados nesta área. Diversos novos conceitos foram apreendidos, incluindo a aplicação de distribuições de probabilidade no contexto de confiabilidade de software, e também o cálculo de métricas de confiabilidade.

Durante o desenvolvimento do projeto, foi possível aprender sobre a importância dos logs de falha em um sistema operacional, e também como este mecanismo funciona no Microsoft Windows. Foi necessário também lidar com o processamento e operações em grandes quantidades de dados, estruturação de banco de dados para armazenamento de informação, realizar a integração entre duas linguagens de programação e implementar algoritmos matemáticos responsáveis por gerar métricas de confiabilidade.

Diversas dificuldades foram encontradas e superadas durante o desenvolvimento do projeto, e estas serão descritas na seção a seguir.

5.3 Dificuldades encontradas

No início do projeto uma das dificuldades encontradas se deu pelo mapeamento dos eventos com sua respectiva mensagem de falha, dado que a mensagem de falha no Microsoft Windows é retirada da aplicação que publicou o evento, tornando-se necessário a criação de uma base de dados de mensagens de falha própria.

Outra dificuldade encontrada nas etapas iniciais do projeto foi a necessidade de otimização do código para que o tempo de execução passasse de minutos para segundos, de forma a não impactar a espera do usuário final pelo relatório de confiabilidade.

Na etapa de análise, uma das dificuldades se deu no estudo do referencial teórico para aplicação dos conceitos de confiabilidade necessários para realização deste trabalho, visto que diversos conceitos não foram antes introduzidos durante a graduação.

Outra dificuldade que ocorreu ao decorrer da implementação das fases do Analisador, se deu pela necessidade de aplicar técnicas de normalização dos dados para que fosse possível aderir a determinadas distribuições de probabilidade, o que fez com que tivessem de ser tomadas algumas decisões a respeito dos possíveis impactos destas técnicas ao resultado final.

Felizmente todas as dificuldades encontradas foram superadas e servirão de aprendizado tanto para a vida profissional quanto para trabalhos futuros no âmbito da ferramenta X-RAT.

Referências

ABBOTT, D. **Linux for Embedded and Real-time Applications**. [S.l.]: Newnes, 2012. v. 3. 296 p. Citado na página 10.

AKAIKE, H. A new look at the statistical model identification. **IEEE Transactions on Automatic Control**, v. 19, n. 6, p. 716–723, 1974. Citado na página 14.

ANSI/IEEE. Standard glossary of software engineering terminology. ANSI/IEEE, 1991. Citado na página 10.

AVIZIENIS, A.; LAPRIE, J.-C.; RANDELL, B.; LANDWEHR, C. Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing, v. 1, p. 11–33, 2004. Citado na página 14.

BALLENTHIN, W. **python-evtx**. 2021. Disponível em: <https://github.com/williballenthin/python-evtx>. Citado na página 36.

BEN-AMRAM, O. **pyevtx-rs**. 2022. Disponível em: <https://github.com/omerbenamram/pyevtx-rs>. Citado 2 vezes nas páginas 25 e 36.

BOTT, E.; STINSON, C. **Windows 10 Inside Out**. [S.l.]: Microsoft Press, 2016. Citado na página 11.

CULLEN, A. C.; FREY, H. C. **Probabilistic techniques in exposure assessment : a handbook for dealing with variability and uncertainty in models and inputs / Alison C. Cullen and H. Christopher Frey.** New York: Plenum Press, 1999. ISBN 0306459566. Citado na página 14.

DESKTOP Operating System Market Share Worldwide. 2022. Disponível em: <https://gs.statcounter.com/os-market-share/desktop/worldwide>. Citado na página 11.

DOWSON, M. The ariane 5 software failure. ACM SIGSOFT Software Engineering Notes, v. 22, n. 2, p. 84, 1997. Disponível em: <https://doi.org/10.1145/251880.251992>. Citado na página 9.

FARR, W. H.; SMITH, O. D. A tool for statistical modeling and estimation of reliability functions for software: Smerfs. **Journal of Systems and Software**, v. 8, n. 1, p. 47–55, 1988. ISSN 0164-1212. Software Engineering. Disponível em: <https://www.sciencedirect.com/science/article/pii/016412128890043X>. Citado na página 19.

GANAPATHI, A.; GANAPATHI, V.; PATTERSON, D. Windows xp kernel crash analysis. in Proc. of the Conference on Large Installation System Administration, p. 149–159, 2006. Citado na página 16.

GAUTIER, L. **rpy2**. 2022. Disponível em: <https://rpy2.github.io/>. Citado 2 vezes nas páginas 27 e 36.

IEEE. Charter and organization of the software reliability engineering committee. 1995. Citado na página 13.

KANG, M.; GU, T.; BAIK, J. Fast abstract: A user friendly software reliability analysis tool based on development process to iteratively manage software reliability. In: . [S.l.: s.n.], 2009. Citado na página 20.

KANOUN, K.; KAANICHE, M.; LAPRIE, J.-C.; METGE, S. Sorel: A tool for reliability growth analysis and prediction from statistical failure data. In: **FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing**. [S.l.: s.n.], 1993. p. 654–659. Citado na página 20.

LEVESON, N. G.; TURNER, C. S. An investigation of the therac-25 accidents. IEE Computer, v. 26, n. 7, p. 18–41, 1993. Disponível em: <https://doi.org/10.1109/MC.1993.274940>. Citado na página 9.

LI, P. L.; NI, M.; XUE, S.; MULLALLY, J. P.; GARZIA, M.; KHAMBATTI, M. Reliability assessment of mass-market software: insights from windows vista. in Proc. of the Int'l Symp. on Software Reliability Engineering, p. 265–270, 2008. Citado na página 16.

LYU, M.; NIKORA, A. Casre: a computer-aided software reliability estimation tool. In: **[1992] Proceedings of the Fifth International Workshop on Computer-Aided Software Engineering**. [S.l.: s.n.], 1992. p. 264–275. Citado na página 19.

LYU, R. **Handbook of Software Reliability Engineering**. New York, NY, USA: IEEE Computer Society Press, 1996. Citado 2 vezes nas páginas 15 e 18.

_____. **Software Reliability Engineering: A Roadmap**. [S.l.]: Proc. 29th Int. Conf. Softw. Eng., 2007. 153-170 p. Citado na página 9.

MATIAS, R.; OLIVEIRA, G.; ARAUJO, L. Operating system reliability from the quality of experience viewpoint: an exploratory study. in Proc. of the ACM Symp. on Applied Comp., p. 1644–1649, 2013. Citado 4 vezes nas páginas 16, 27, 28 e 33.

MATIAS, R.; PRINCE, M.; BORGES, L.; SOUSA, C.; ; HENRIQUE, L. An empirical exploratory study on operating system reliability. In Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC '14), ACM, New York, NY, USA, p. 1523–1528, 2014. Citado na página 17.

MICROSOFT. **Event Schema Elements**. 2022. Disponível em: <https://learn.microsoft.com/en-us/windows/win32/wes/eventschema-elements>. Citado na página 23.

NETIKUS. **Event Sentry**. 2022. Disponível em: <https://www.eventsentry.com/>. Citado na página 25.

O'CONNOR, P. **Practical Reliability Engineering**. [S.l.]: John Wiley Sons, 2011. Citado na página 13.

PHAM, H. **System Software Reliability**. [S.l.]: Springer, 2007. Citado na página 15.

RELIABILITY; CONSULTANTS, S. **SRMP**. 1988. Citado na página 19.

ROBINSON, R. B. A. **ReportLab**. 2022. Disponível em: <https://www.reportlab.com/>. Citado na página 29.

SANTOS, C. A. R. D.; ANTUNES, M.; MATIAS, R.; ASSUNçãO, L.; MACIEL, V. Reliability assessment of commercial off-the-shelf operating system software: An empirical study. in Proc. of the Brazilian Symp. on Computing Systems Engineering, 2018. Citado na página 17.

SANTOS, C. D.; MATIAS, R. Failure patterns in operating systems: An exploratory and observational study. Elsevier Journal of Systems and Software, p. 512–530, 2018. Citado na página 17.

SANTOS, C. D.; MATIAS, R.; TRIVEDI, K. S. An empirical study on patterns of failure causes in a mass market operating system. In: SEFFAH, A.; PENZENSTADLER, B.; ALVES, C.; PENG, X. (Ed.). **Proceedings of the Symposium on Applied Computing, SAC 2017, Marrakech, Morocco, April 3-7, 2017**. ACM, 2017. p. 1542–1547. Disponível em: <https://doi.org/10.1145/3019612.3019740>. Citado na página 35.

_____. An empirical exploratory analysis of failure sequences in a commodity operating system. in Proceedings of the Brazilian Symposium on Computing Systems Engineering, 2019. Citado na página 17.

_____. A statistical approach to predict operating system failures based on multiple failures association. in Proceedings of the Brazilian Symposium on Computing Systems Engineering, p. 1–8, 2020. Citado na página 18.

_____. A multisite characterization study on failure causes in system and applications software. In: **2021 XI Brazilian Symposium on Computing Systems Engineering (SBESC)**. [S.l.: s.n.], 2021. p. 1–8. Citado na página 35.

SASANKAR, D. S. D. A. An overview and comparison of software reliability tools. IOSR Journal of Computer Engineering (IOSR-JCE), Department, of computer Science, G.H. Raisoni Institute of Information Technology, Nagpur, India, 2016. Citado 2 vezes nas páginas 18 e 19.

SINGH, M. Software reliability testing tools: An overview and comparison. International Journal Of Engineering And Computer Science, 2016. Citado na página 19.

SWIFT, M. M.; BERSHAD, B. N.; LEVY, H. M. Improving the reliability of commodity operating systems. in Proc. of the ACM Symposium on Operating Systems Principles, p. 207–222, 2003. Citado na página 16.

TANG, D.; HECHT, M.; MILLER, J.; HANDAL, J. Meadep: a dependability evaluation tool for engineers. **IEEE Transactions on Reliability**, v. 47, n. 4, p. 443–450, 1998. Citado na página 19.

TAUSWORTHE, R. A general software reliability process simulation technique. 05 1991. Citado na página 19.

TEAM lxml dev. **lxml**. 2022. Disponível em: <https://lxml.de/>. Citado na página 25.

TEAM, T. P. D. **Pandas**. 2022. Disponível em: <https://pandas.pydata.org/>. Citado 2 vezes nas páginas 26 e 27.

TRIVEDI, K. Srept: a tool for software reliability estimation and prediction. In: **Proceedings International Conference on Dependable Systems and Networks**. [S.l.: s.n.], 2002. p. 546–. Citado na página 20.

WALL, R.; SHERMAN, M. The multiple problems, and potential fixes, with the boeing 737 max. 2019. Disponível em: <https://www.wsj.com/articles/fixing-the-problems-with-boeings-737-max-11566224866>. Citado na página 9.

XU, Z. K. J.; IYER, R. Networked windows nt system field failure data analysis. In Proc. of Pacific Rim Int'l Symp. on Dependable Computing, p. 178–185, 1999. Citado na página 16.