Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Testes em softwares educacionais para avaliação de aspectos de qualidade

Victor Hugo Bednarczuk

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Orientador
Prof. Dr. Wilson Henrique Veneziano

Brasília
2019

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Testes em softwares educacionais para avaliação de aspectos de qualidade

Victor Hugo Bednarczuk

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Prof. Dr. Wilson Henrique Veneziano (Orientador)
CIC/UnB

Prof. Dr. Jorge Henrique Cabral Fernandes
CIC/UnB

Prof.ª Dr.ª Germana Menezes da Nóbrega
CIC/UnB

Prof. Dr. Wilson Henrique Veneziano
Coordenador do Curso de Computação — Licenciatura

Brasília, 1 de fevereiro de 2019

Dedicatória

Dedico esse trabalho aos meus pais que sempre estão comigo durante todos os momentos da vida, aos familiares, amigos e à todas as outras pessoas que torcem pelo meu sucesso e que, mesmo indiretamente, contribuíram para a realização desse trabalho e finalização dessa etapa.

Agradecimentos

Agradeço ao professor Wilson Henrique Veneziano que forneceu o equipamento e contribuiu com seu grande conhecimento necessário para a elaboração desse trabalho, e também por seu profissionalismo, paciência, competência e grande disponibilidade para ajudar.

Resumo

Esse trabalho descreve uma avaliação de softwares educacionais com base em normas internacionalmente validadas e aceitas. As informações contidas aqui são úteis para avaliadores e desenvolvedores que buscam conhecer um processo avaliativo que abrange aspectos de qualidade importantes dentro do contexto educacional. O alvo dos testes foram aspectos de qualidade essencialmente técnicos e computacionais dos softwares, sem abordar aspectos pedagógicos. No final são feitas comparações, discussões e sugestões para a melhoria no desenvolvimento de softwares futuros e/ou manutenção dos softwares avaliados.

Palavras-chave: qualidade de software, software educacional, avaliação de software

Abstract

This work describes a educational software evaluation process based on internationally recognized and accepted norms. Information contained here is relevant for evaluators and developers who are looking for an evaluation process that approaches quality characteristics inside the educational software context. The process defined here aims to evaluate quality from an essentially technic point of view, not approaching pedagogical issues. In the end, comparisons, discussions and suggestion are made for the evaluated softwares,with the intent to raise the quality of the future development process and/or their maintenance.

Keywords: software quality, educational software, software evaluation

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Vivemos em uma época onde softwares e os sistemas por eles formados possuem uma função básica na sociedade, seja na vida cotidiana, no trabalho, nos momentos de lazer, eles estão sempre presentes. Esse papel fundamental é percebido nas mais diversas áreas: medicina, comércio, educação, aviação, entretenimento. Os computadores foram pensados e desenvolvidos de forma a sistematizar e reproduzir o raciocínio humano, portanto sua função de suporte à sociedade torna-se irrestrita e toda área de conhecimento pode beneficiar-se do seu uso.

Os softwares são objetos complexos, e compostos por diversas partes. Software não é apenas o programa, mas também todos os dados de documentação e configuração associados necessários para seu correto funcionamento [26]. Todo esse material é resultado de um trabalho humano, logo é evidente que softwares também podem ser vistos como produtos e podem ser analisado sob o ponto de vista de sua qualidade.

Existem algumas diferenças óbvias ao compararmos softwares à produtos comuns, o software é intangível e tem uma natureza abstrata, pode ser desde um simples programa que soma dois números inteiros quaisquer, até um complexo sistema formado por softwares menores que controlam uma usina nuclear.

Da necessidade de sistematizar o conhecimento e obter processos e técnicas de produção adequados, surgiu a disciplina chamada Engenharia de Software.

Uma vez que a presença e utilização de softwares é constante e seu papel fundamental na sociedade, é natural que surja uma preocupação com sua qualidade. A disciplina Engenharia de Software utiliza-se de várias métodos, técnicas e processos conjuntamente, objetivando um produto final que atenda requisitos mínimos de qualidade para seu público.

1.1 Problema

Na área educacional, os softwares podem ser usados como ferramentas de auxílio ao ensino, assim é necessário que atinjam níveis mínimos de qualidade e que atendam às suas especificações de maneira efetiva. Para garantir, ou medir essa qualidade, os softwares devem ser submetidos à um conjunto de testes e análises que revelem informações sobre seu funcionamento. Com base nisso, surge a pergunta: Como utilizar o conhecimento adquirido através de testes e avaliações para sugerir melhorias em determinados softwares?

1.2 Justificativa

O processo que envolve análise e testes de software contribui diretamente para a qualidade final do produto, é muito frequente que softwares sejam testados de forma incompleta e/ou ineficiente durante suas fases iniciais de desenvolvimento. Um processo de testes mais controlado e rigoroso, amparado por boas e recomendadas práticas, ainda que após o lançamento do produto, é útil para um controle de qualidade mais eficiente.

1.3 Objetivo Geral

Analisar softwares sob os mais variados aspectos que sejam relevantes ao seu contexto de uso, tendo como objetivo identificar oportunidades de melhoria em termos de qualidade de software. Então, a partir dessa análise, obter resultados significativos que possam auxiliar trabalhos futuros dos desenvolvedores ou outros interessados.

1.4 Objetivos Específicos

Para ser alcançado o objetivo geral de analisar softwares educacionais foram estabelecidos objetivos específicos:

- Obter e apresentar uma série de relatórios organizados sobre cada tipo diferente de análise feita para cada um dos diferentes softwares;
- Discutir os resultados encontrados e seu significado;
- Auxiliar futuros trabalhos e atividades de testes feitos de forma independente (fora de um contexto organizacional/empresarial).

Capítulo 2

Referencial Teórico

Esse capítulo apresenta o referencial teórico utilizado no desenvolvimento deste trabalho e essencial para sua compreensão.

2.1 Engenharia de Software

Segundo o dicionário [2] de termos da área, Engenharia de Software é a aplicação sistemática, disciplinada e quantificável de uma abordagem de desenvolvimento, operação e manutenção de software, vale ressaltar que essa área de conhecimento desenvolveu-se muito integrada ao contexto empresarial, logo conceitos como tempo e custo estão amplamente presentes também. Num contexto mais organizacional e empresarial, a Engenharia de Software é vista como uma área que foca no desenvolvimento de softwares de alta qualidade com um custo adequado e no prazo aceitável.

Para atingir este objetivo são utilizados métodos e técnicas que definem um complexo processo, onde o produto final é o software. Portanto, o processo de software é definido como um *framework* que inclui pessoas, métodos e técnicas de engenharia e que agindo conjuntamente resultam num produto final. É enfatizado na literatura da área o caráter não-determinístico desses processos e interações, isso torna a tarefa de determinar o melhor caminho para a elaboração de um software bastante difícil e cabe aos profissionais Engenheiros de Software a tomada dessas decisões. Essa natureza adaptativa de técnicas, métodos e processos na Engenharia de Software é uma característica marcante na área.

Uma outra definição particularmente interessante para este trabalho, é dada por *Pressman* [26]. Segundo o autor, a pedra fundamental que sustenta a engenharia de software é o foco na qualidade, sua definição por camadas pode ser observada na figura a seguir.

Figura 2.1: Foco na qualidade (Fonte: [26]).

2.1.1 Qualidade de Software

Segundo *Pfleeger*, a qualidade de um software pode ser compreendida e abordada a partir de 5 perspectivas diferentes [24].

- **Visão transcendental:** pela qual a qualidade é algo que se reconhece mas não se define;
- **Visão do usuário**: pela qual qualidade significa atingir seus objetivos;
- **Visão da manufatura:** pela qual qualidade está relacionada a conformidade com especificações;
- **Visão do produto:** pela qual qualidade está relacionada a características intrínsecas do produto;
- **Visão com base no valor:** pela qual a qualidade depende do valor que o cliente está disposto a pagar.

Dos conceitos apresentados, três são particularmente interessantes para a Engenharia de Software e estão diretamente ligados ao processo de avaliação de um software a partir de um modelo de qualidade, que será apresentado posteriormente.

A visão do usuário é determinante para o sucesso do resultado final de um processo de desenvolvimento, pois é a eles que o produto se destina. Se os usuários não conseguem operar o software de forma a atingir seus objetivos e considerando um cenário atual de forte concorrência, o software não alcançará seu objetivo e o projeto fracassa. Esse aspecto da qualidade está relacionado ao conceito de *qualidade em uso* que é definida como a qualidade percebida por um usuário ao utilizar o software para alcançar seus próprios objetivos de acordo com suas necessidades em um determinado cenário.

A visão de manufatura é aquela utilizada pela equipe de desenvolvimento para verificar seu próprio trabalho. Durante um processo de desenvolvimento devem ser feitas análises para adequar e guiar o trabalho na direção correta, essa visão está relacionada ao conceito de *qualidade interna* que é definida como a totalidade das características do produto de software do ponto de vista interno [15]. Isso é, seu código, suas especificações e artefatos relacionados.

A visão de produto é aquela que considera as características intrínsecas do produto em sua totalidade, está relacionada com a *qualidade externa* que é definida como a qualidade quando o software é executado, o qual é tipicamente medido e avaliado enquanto está sendo testado num ambiente simulado. Também está relacionada à *qualidade interna* que já foi definida previamente. A figura a seguir demonstra a conexão que relaciona esses três conceitos de qualidade

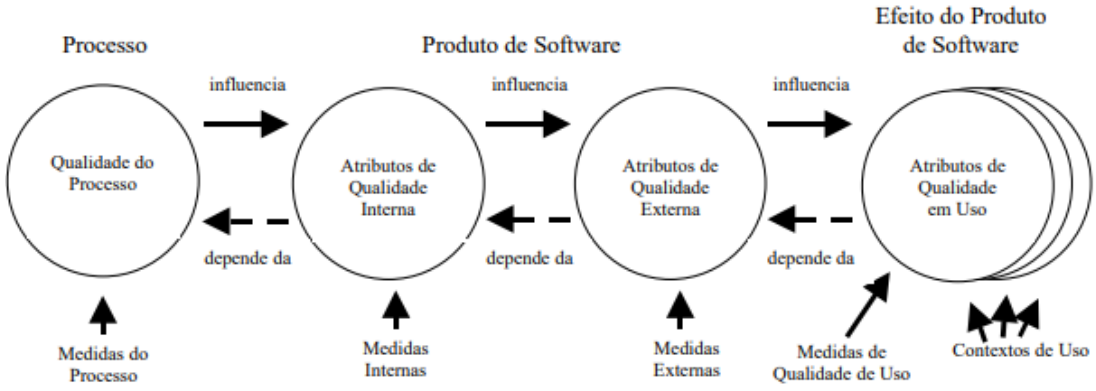


Figura 2.2: Qualidade no ciclo de vida (Fonte: [15]).

Esse diagrama demonstra a importância da busca pela qualidade durante todo o processo de software, pois os atributos estão encadeados e são dependentes uns dos outros. Os desenvolvedores criam código (qualidade interna) que influencia diretamente os atributos externos (qualidade externa) que por sua vez influencia a qualidade para fins de usos em cenários específicos de necessidades de um usuário (qualidade em uso).

A conexão entre atendimento aos requisitos e qualidade é evidente, uma vez que as funcionalidades especificadas nas fases de desenvolvimento são atendidas no produto de software, esse produto será então visto pelo usuário como algo de qualidade. Além disso, também impactam na percepção de qualidade os requisitos não-funcionais, que são as características não relacionadas às funcionalidades específicas de um sistema. Segundo *Sommerville*, eles estão relacionados às propriedades emergentes do sistema em uso, como confiabilidade, tempo de resposta e espaço de armazenamento [28].

O conjunto de funcionalidades e características explícitas e implícitas de um software são definidos como requisitos de software.

2.1.2 Medidas e Métricas

Durante o processo de desenvolvimento de software é necessário obter medidas que auxiliem nos processos de tomada de decisão dos gerentes. A informação qualitativa é útil, mas a obtenção de dados em forma numérica é mais objetiva e permite comparações.

Uma medida fornece uma indicação quantitativa da extensão, quantidade, capacidade ou tamanho de algum atributo de um produto ou processo. Medição é o ato de determinar uma medida [26]. Uma métrica de software relaciona medidas individuais de alguma maneira para derivar uma informação mais objetiva, essa relação pode ser o tempo médio para que um usuário alcance um grau de conhecimento em um software, por exemplo [24].

No contexto de avaliação de software, as medidas internas são aquelas relacionadas diretamente ao código do software, a obtenção da medida nesse caso é um processo objetivo, pois seu alvo está bem definido e não depende de um processo subjetivo de interpretação. Um exemplo é o número de linhas de código de um software.

As medidas externas estão relacionadas à visão externa do software, a obtenção não pode ser feita diretamente e de forma objetiva, os modelos de qualidade devem ser decompostos a partir das características mais gerais até um atributo que possa ser medido de forma mais precisa.

As medidas de qualidade em uso são aquelas derivadas a partir de casos de uso específicos de um usuário. Um exemplo é o tempo de resposta do software para a execução de determinada funcionalidade sob um cenário definido. Todas essas medidas são utilizadas na prática para processos de melhoria da qualidade do software, cabe aos profissionais

engenheiros da área a escolha das medidas e métricas mais adequadas que produzam um resultado desejado.

2.1.3 Processo de Software

Um processo de software é um conjunto de atividades e resultados associados que produz um software. Existem quatro atividades fundamentais de processo, que são comuns a todos os processos de software. São elas: especificação de requisitos, desenvolvimento, validação e evolução/manutenção [28].

- **Especificação de requisitos:** Clientes e engenheiros definem o software a ser produzido e suas restrições;
- **Desenvolvimento:** O software é projetado e programado;
- **Validação:** O software é verificado para garantir que está de acordo com o desejo do cliente;
- **Evolução:** O software é modificado para se adaptar às mudanças dos requisitos do cliente e do mercado.

A preocupação com a qualidade está presente desde a especificação do software, e existem diversos tipos de atividades a serem desenvolvidas para sua garantia.

2.1.4 Atividades de Validação, Verificação e Teste

Essas atividades têm como finalidade garantir que tanto o modo pelo qual o software está sendo construído quanto o produto em si estejam em conformidade com o especificado, além disso são divididas em análises estáticas e dinâmicas.

As estáticas não requerem a existência, nem execução de um programa ou modelo, já as dinâmicas são baseadas em execução de um programa ou modelo [7].

Qualquer artefato gerado desde as fases iniciais de desenvolvimento pode ser alvo de algum tipo de validação, uma inspeção na documentação de requisitos é um exemplo de atividade de análise estática, de forma geral esse tipo de atividades está mais presente nas fases de iniciais de desenvolvimento do software, porém não é regra. A inspeção e análise de documentação final destinada ao usuário é um exemplo de atividade estática que pode acontecer após o lançamento do produto.

A análise dinâmica também tem por objetivo detectar defeitos ou erros no software e envolve a execução do produto, seja o código propriamente dito ou mesmo uma especificação executável [5]. Ambos os tipos de atividades são importantes para a qualidade do software e são atividades compreendidas como complementares.

O teste de software é uma atividade dinâmica que envolve execução do programa, protótipo, ou modelo funcional. O objetivo dos testes é pôr sob prova requisitos do software e assim extrair alguma informação relevante desejada que possa contribuir para o seu desenvolvimento ou mesmo para uma análise que busque compreender o funcionamento e qualidade de um software.

Softwares de áreas distintas possuem prioridades distintas relacionadas à sua área de especialização, elas devem ser levadas em consideração durante as atividades de testes. Um software bancário deve ser confiável, um editor de texto de uso geral deve ser facilmente operável, etc.

Para uma medição de qualidade efetiva, é necessária uma base ou modelo de referência teórica que norteie as avaliações, isso se faz pela necessidade de serem encontrados aspectos e características que possam ser objetivamente julgados e analisados e que estejam dentro do contexto da categoria do software. De acordo com *Rocha*, cada uma das características pode ser detalhada em vários níveis de subcaracterísticas, chegando-se a um amplo conjunto de atributos que descrevem a qualidade de um produto de software [5].

2.1.5 Categorias de Software

Um processo de categorização de software é importante. Ao se conhecer uma sistematização básica de categorias, o processo de encontrar um software que se adeque a necessidades específicas torna-se muito mais simples. Atualmente, sete grandes categorias de software apresentam desafios contínuos para engenheiros de software [26].

1. **Software de sistema**: Conjunto de programas feito para atender a outros programas. Certos softwares de sistema processam estruturas de informação complexas; porém, determinadas. Outras aplicações de sistema processam dados amplamente indeterminados;

2. **Software de aplicação**: Programas independentes que solucionam uma necessidade especifica de negócio. Aplicações nessa área processam dados comerciais ou técnicos de uma forma que facilite operações comerciais ou tomadas de decisão administrativas/técnicas;

3. **Software de engenharia/científico**: Uma ampla variedade de programas de "cálculo em massa" que abrangem astronomia, vulcanologia, análise de estresse automotivo, dinâmica orbital, projeto auxiliado por computador, biologia molecular, análise genética e meteorologia, entre outros;

4. **Software embarcado:** Residente num produto ou sistema e utilizado para implementar e controlar características e funções para o usuário e para o próprio sistema. Executa funções limitadas e específicas ou fornece função significativa e capacidade de controle;

5. **Software para linha de produtos:** Projetado para prover capacidade específica de utilização por muitos clientes diferentes. Software para linha de produtos pode se concentrar em um mercado hermético e limitado, ou lidar com consumidor de massa;

6. **Aplicações Web/aplicativos móveis:** Esta categoria de software voltada às redes abrange uma ampla variedade de aplicações, contemplando aplicativos voltados para navegadores e software residente em dispositivos móveis;

7. **Software de inteligência artificial:** Faz uso de algoritmos não numéricos para solucionar problemas complexos que não são passíveis de computação ou de análise direta. Aplicações nessa área incluem: robótica, sistemas especialistas, reconhecimento de padrões, redes neurais artificiais, prova de teoremas e jogos.

Dentro do contexto do processo de avaliação deste trabalho, a categorização é particularmente útil pois auxilia na definição de requisitos e características de qualidade adequados para os softwares que serão objeto da avaliação.

Também são importantes as categorizações definidas em normas internacionalmente aceitas, um órgão que definiu uma dessa normas é o IEEE (*Institute of Electrical and Electronics Engineers*.

A norma IEEE 1062 (*Recommended Practice for Software Acquisition*) define uma categorização baseada no processo de aquisição e não num modelo conceitual como o anterior. Neste documento são definidos três tipos de softwares sob ponto de vista de sua aquisição [17].

1. **Software desenvolvido sob medida:** Esse processo de aquisição envolve um contrato. No desenvolvimento desse tipo de software, o processo está fortemente conectado aos interessados em todas suas fases desde a concepção até o release e manutenção. A saída é um software altamente personalizado e que satisfaz as necessidades específicas do contratante;

2. **Software de prateleira:** Neste processo o software adquirido é uma solução já pronta fabricada por alguma organização interessada no campo de atuação do software. A organização ou indivíduo interessado no software fabricado deve estar de acordo com os termos de uso de execução do software e geralmente não tem controle algum relacionado à suas especificações e funcionalidades, nem acesso ao código-fonte. O software geralmente é lançado com o acordo de licença as is, isso significa que o usuário aceita utilizá-lo na forma em que foi disponibilizado. Essa categoria inclui softwares comerciais de prateleira (COTS), software livre e software open source (FOSS) adquiridos em condição as is;

3. **Software como serviço:** Software como serviço é uma solução onde o adquirente entra em acordo com o fornecedor para utilização do software por um determinado intervalo de tempo de forma remota através de alguma conexão. As entradas de dados do software são transmitidas para o fornecedor que por sua vez, retorna uma resposta com os dados processados.

Os softwares educacionais abordados neste trabalho possuem a característica marcante de estarem disponíveis na internet sob uma licença específica *as is* para o público interessado em seu conteúdo. Isso significa que os produtos são entregados sem garantias. Portanto, o público em geral não tem nenhum vínculo com os desenvolvedores do projeto e não tem nenhum controle sob suas especificações. Sendo assim, os softwares se encaixam na categoria software de prateleira, ou COTS(*Comercial off the shelf*). Vale ressaltar que o uso da palavra *Comercial* nesse contexto particular, não está relacionado à venda e preço do software, mas sim em relação à sua forma de distribuição pública.

Nessa categoria, a documentação destinada aos aquisidores é particularmente importante para a qualidade do software, pois as especificações e funcionalidade são formuladas de forma a abranger seu público-alvo de uma forma mais generalizada. Assim documentos de descrição do produto e outros tipos de documentos auxiliares de alta qualidade destinados ao usuário final são necessários.

Na literatura da área, essa categoria também é chamada pacote de software, assim eram chamados os softwares na época de surgimento da primeira tentativa normalização que tinha como objetivo avaliação de softwares adquiridos com licenças do tipo *as is*.

2.2 Modelo de Qualidade e Processo de Avaliação

Os modelos de qualidade surgem pela necessidade de analisar de forma sistemática e objetiva a qualidade dos softwares. Além disso, esses modelos também permitem entender como diferentes facetas contribuem para a qualidade do produto como um todo [24].

A terminologia utilizada para características da qualidade de software difere entre cada tipo de modelo. Cada modelo, muitas vezes, possui diferentes quantidades de níveis hierárquicos e também diferentes quantidades de características no total. Muitos autores produziram modelos de qualidade de software com características, ou atributos, que são úteis para discutir, planejar e avaliar a qualidade de produtos de software [4].

Um trabalho pioneiro, amplamente citado e reconhecido nessa área é o de *McCall*, [20] com sua proposta de categorização de fatores que afetam a qualidade de software. Os 11 fatores sugeridos de qualidade divididos em três grupos diferentes são apresentados na figura a seguir.

Figura 2.3: Fatores de qualidade de McCall (Fonte: [26]).

Os fatores de qualidade de McCall são divididos em:

- **Operação:** São os fatores emergentes da utilização do produto propriamente dita;
- **Revisão:** São os fatores relacionados à todo tipo de atividade de manutenção do produto;
- **Transição:** São os fatores relacionados à capacidade adaptativa do software à outros ambientes e sua interação com outros softwares.

É difícil e em alguns casos impossível, desenvolver medidas diretas desses fatores de qualidade. Na realidade, muitas das métricas definidas nesse modelo podem ser medidas apenas indiretamente. Entretanto, avaliar a qualidade de uma aplicação usando esses fatores possibilitará uma sólida indicação da qualidade de um software [26]

A utilização de um modelo de qualidade de software genérico como padrão por toda indústria de software compõe um cenário ideal, pois seria útil no sentido de trazer uma certa uniformização na visão de produto, tanto para clientes, quanto para fabricantes. Uma vez que todo software pode ser decomposto e analisado sob características universais, o processo de análises e comparação seria facilitado, beneficiando desenvolvedores e usuários.

Porém, vale ressaltar novamente que é difícil classificar os softwares com base em suas características devido às suas diferentes aplicações, domínios de uso e características abstratas, um modelo de qualidade universal e imutável é algo fora da realidade do mundo da Engenharia de Software.

2.2.1 Modelo de Qualidade de Software ISO/IEC 25010

ISO (*International Standards Organization*) é uma organização não-governamental que reúne especialistas para desenvolver padrões internacionais baseados em consenso e relevância para o mercado [18]. Possui a missão de promover o desenvolvimento da normalização com os objetivos de facilitar as trocas internacionais de bens e serviços e de desenvolver a cooperação nos campos da atividade intelectual, científica, tecnológica e econômica.

IEC (*International Electrotechnical Commission*) é a principal organização global publicadora de padrões internacionais baseados em consenso e gerencia sistemas de avaliação de conformidade para produtos, sistemas e serviços elétricos e eletrônicos, coletivamente conhecidos como eletrotecnologia [13].

Essas duas organizações atuam conjuntamente em algumas áreas, criando comitês para discussão de padrões internacionais, o comitê responsável pela área de tecnologia da informação, incluindo Engenharia de Software é o ISO/IEC JTC 1.

A ABNT (Associação Brasileira de Normas Técnicas) é o Foro Nacional de Normalização por reconhecimento da sociedade brasileira desde a sua fundação, em 28 de setembro de 1940, e confirmado pelo governo federal por meio de diversos instrumentos legais. Entidade privada e sem fins lucrativos, a ABNT é membro fundador da ISO. Desde a sua fundação, é também membro da IEC [1].

As normas decretadas válidas em território nacional pela ABNT (Assoicação Brasileira de Normas Técnicas) nem sempre estão atualizadas em relação às normas criadas e publicadas pela ISO/IEC, isso acontece por diversas razões internas das organizações. Este trabalho considera as publicações mais recentes publicadas em conjunto por ISO/IEC como seu objeto principal de estudo e consulta.

A primeira tentativa de *normalização* relativa à modelos de qualidade de software, foi feita na família de normas ISO/IEC 9126. São apresentados dois modelos de qualidade, um análogo ao modelo McCall, chamado modelo de qualidade de produto que aborda qualidade interna e externa e um outro especificado e desenvolvido para abranger a qualidade em usos específicos, chamado modelo de qualidade em uso. Os dois modelos são definidos na ISO/IEC 9126-1, métricas de qualidade externas são apresentadas na ISO/IEC 9126-2, métricas de qualidade internas na ISO/IEC 9126-3 e métricas de qualidade em uso na ISO/IEC 9126-4.

A figura a seguir demonstra o modelo de qualidade definido na ISO/IEC 9126-1

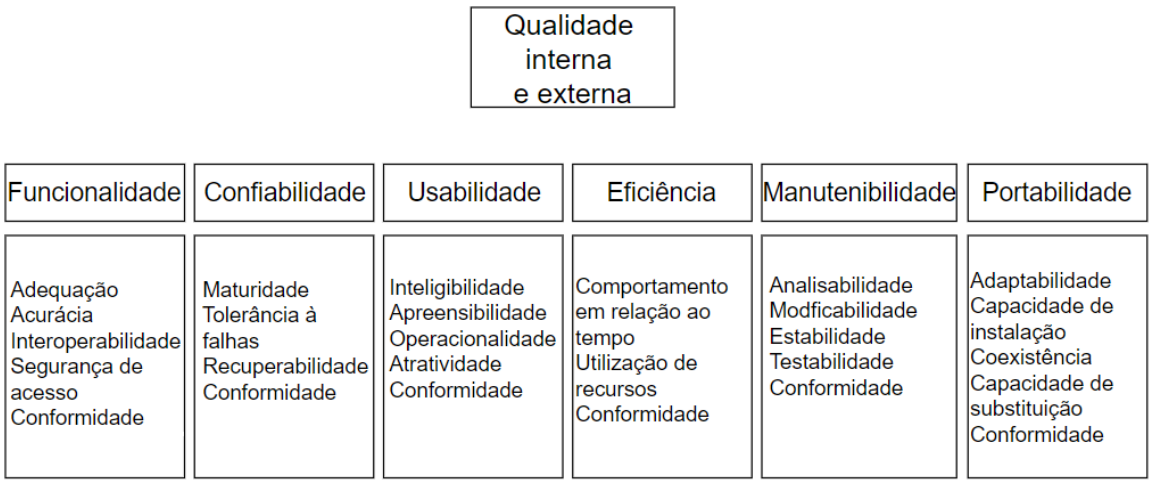


Figura 2.4: Modelo de Qualidade de Software ISO/IEC 9126-1 (Fonte: [15]).

Os conceitos apresentados nessas normas permanecem em sua maioria válidos, mas por questões de organização de informação foi criado o projeto SQuaRE (*Software Quality Requirements Evaluation*), que harmoniza essa e outras normas da área de forma mais coerente e atualizada, esse projeto está agrupado na família de normas 25000.

Uma abordagem mais recente iniciada pela ISO/IEC para modelo de qualidade de software foi definida na norma ISO/IEC 25010.

A norma tem importância significativa para Engenharia de Software. O modelo de qualidade de produto apresentado nela é composto por 8 características de qualidade, que são decompostas em níveis menores (sub-características), possui uma similaridade substancial com o modelo anterior apresentado, pois 4 dos seus fatores também estão inclusos no modelo [8].

A ISO/IEC 25010 define o seguinte [4]:

- **Um modelo para qualidade em uso:** Composto por cinco características, nas quais algumas são subdivididas em subcaracterísticas, elas estão relacionadas à visão de qualidade do usuário ao interagir com o software em um contexto de uso particular. O modelo é aplicável tanto a sistemas computacionais em uso, quanto para produtos de software;
- **Um modelo de qualidade de produto:** Composto por oito características, nas quais algumas são divididas em subcaracterísticas, elas estão relacionadas tanto às propriedades estáticas do software, quanto às dinâmicas. O modelo é aplicável tanto a sistemas computacionais em uso, quanto aos produtos de software.

As características definidas e apresentadas para o modelo de qualidade de produto são as seguintes:

- **Adequação Funcional:** A capacidade de um software prover funcionalidades que satisfaçam o usuário em suas necessidades declaradas e implícitas, dentro de um determinado contexto de uso;
- **Performance/Eficiência:** A capacidade do software de possuir tempo de execução e utilização de recursos compatíveis com o nível de desempenho do software;
- **Confiabilidade:** A capacidade do produto de software de manter um nível de desempenho especificado, quando usado em condições específicas;
- **Usabilidade:** A capacidade do produto de software de ser compreendido, aprendido, operado e atraente ao usuário, quando usado sob condições especificadas;
- **Manutenibilidade:** A capacidade (ou facilidade) do produto de software ser modificado, incluindo tanto as melhorias ou extensões de funcionalidade quanto às correções de defeitos, falhas ou erros;
- **Portabilidade:** A capacidade do sistema ser transferido de um ambiente de hardware/software para outro com especificações diferentes;
- **Compatibilidade:** A capacidade do sistema de trocar e compartilhar informações com outros produtos, sistemas ou componentes sob um mesmo ambiente de hardware/software;
- **Segurança:** A capacidade do sistema de proteger informações e dados de forma que pessoas, outros produtos ou sistemas tenham um nível de acesso adequado aos seus tipos e níveis de autorização.

A figura a seguir demonstra as características decompostas em sub-características do modelo de qualidade de produto definido na ISO/IEC 25010

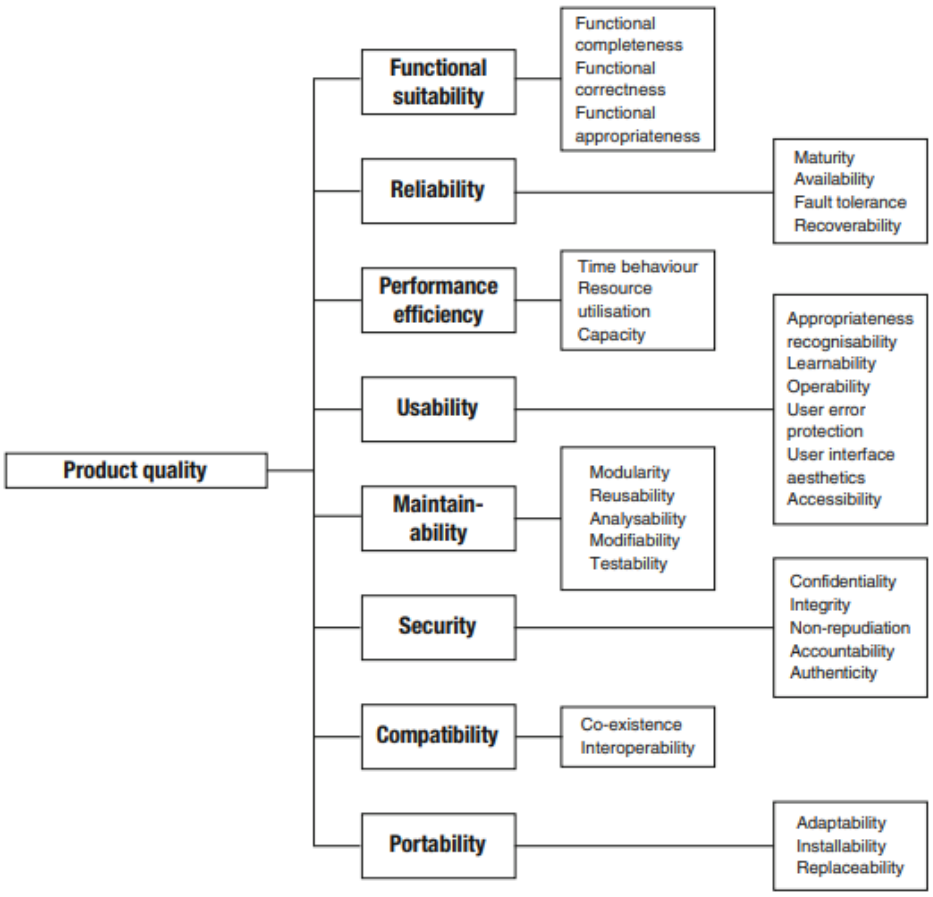


Figura 2.5: Modelo de Qualidade ISO/IEC 25010 (Fonte: [29]).

Esse conjunto de características e subcaracterísticas são derivados do modelo de qualidade da família da norma ISO/IEC 9126-1, existem algumas mudanças pontuais na terminologia e adição de duas características novas, segurança e compatibilidade.

A ISO/IEC 9126-1 define um conjunto de característica de qualidade e suas subcaracterísticas correspondentes. Estas subcaracterísticas são manifestadas externamente, quando o software é utilizado como uma parte de um sistema computacional e são resultantes de atributos estáticos do produto de software [15].

Definido o modelo de qualidade a ser utilizado para avaliação de qualidade, deve ser definido um processo de avaliação eficiente com fases bem-definidas e estruturadas e que proporcionem o alcance dos objetivos propostos inicialmente.

Usabilidade do Software

Nos meados de 1960, a maioria dos usuários de software eram profissionais da área de computação que desenvolviam e efetuavam manutenção do software em seus locais de trabalho. Com o surgimento dos computadores pessoais, tornou-se comum que usuários instalem e configurem seus próprios softwares. [10]

A visão técnica dos primeiros utilizadores não poderia ser um fator limitador para a utilização de softwares, pois isso restringiria sua utilização. Dessa forma, a preocupação por parte dos desenvolvedores com a facilidade de uso das interfaces dos softwares tornou-se um conceito básico da Engenharia de Software e que está diretamente relacionado à qualidade do software.

Para que um produto de software seja bem sucedido, deve apresentar boa usabilidade, uma medida qualitativa da facilidade e eficiência com a qual um ser humano consegue empregar as funções e os recursos oferecidos pelo produto de alta tecnologia [26].

A norma ISO 9241-11 que trata da ergonomia na relação homem-computador, define usabilidade como a capacidade de um produto ser usado por usuários específicos para atingir objetivos específicos com eficácia, eficiência e satisfação em um contexto específico de uso [19].

Para atingir os objetivos citados, o desenvolvedor, ou o responsável pela criação da interface do usuário, deve conhecer princípios fundamentais do Design e conhecer métodos de aplicação voltados aos software e suas interfaces. Dentro da área de tecnologia, a disciplina *Interação Humano-Computador* aborda fundamentos de Design e de muitas outras áreas de conhecimento para compreender os fenômenos de interação e também desenvolver métodos eficientes para interfaces.

Interação Humano-Computador é um campo de estudo interdisciplinar que tem como objetivo geral entender como e porque as pessoas utilizam, ou não utilizam, a tecnologia da informação [6].

Avaliação da Usabilidade

Os métodos de avaliação de usabilidade podem envolver usuários finais, nesse tipo de avaliação são elaboradas atividades baseadas em observação comportamental, checklists, questionários, entrevistas, dentre outros. Esse tipo de avaliação têm restrições relativas à recursos humanos, é necessário encontrar pessoas dispostas a efetuar os testes do software, além disso devem representar uma parcela significativa dos usuários finais ou potenciais de um software [27].

Todos esses elementos somados ao tempo gasto e recursos financeiros associados podem ser fatores proibitivos para uma avaliação desse tipo.

Existem maneiras de avaliar uma interface sem a presença de usuários finais, isso pode ser feito por uma equipe, ou indivíduo que possua conhecimento em princípios de design, e em algum método de avaliação.

A *avaliação heurística* constitui-se em em uma técnica de inspeção de usabilidade em que especialistas orientados por um conjunto de princípios de usabilidade conhecidos como heurísticas, avaliam se os elementos da interface com o usuário - caixas de diálogo, menus, estruturas de navegação, ajuda on-line, etc - estão de acordo com os princípios [25].

As 10 heurísticas desenvolvidas por *Nielsen* são as seguintes [23]:

1. **Visibilidade do status do sistema:** O software deve fornecer feedback constante de seu estado atual ao usuário;
2. **Compatibilidade do sistema com o mundo real:** Deve ser utilizada uma linguagem natural ao usuário, com frases, símbolos e conceitos familiares. O conteúdo deve fluir de forma organizada e natural;
3. **Controle de usuário e liberdade:** O usuário deve conseguir utilizar as funcionalidades de forma exploratória, sem que isso prejudique o andamento das atividades, deve ser possível reverter ações com facilidade;
4. **Consistência e padrões:** A forma com que a informação é representada, e as formas de interação devem ser consistentes por todo o software;
5. **Mensagens de erro úteis:** Devem ser expressadas em linguagem compreensível e adequada ao usuário, devem indicar o problema precisamente e sugerir uma solução de forma construtiva;
6. **Reconhecer, ao invés de relembrar:** Os objetos, ações e estados do sistema devem estar sempre visíveis;
7. **Prevenção de erros:** A interface deve prevenir, através de algum mecanismo, que o usuário cometa erros; Reconhecer, ao invés de relembrar: Os objetos, ações e estados do sistema devem estar sempre visíveis;
8. **Flexibilidade e eficiência de uso:** Devem ser oferecidos aceleradores, ou atalhos, para que usuários atinjam seus objetivos mais rapidamente;
9. **Design minimalista:** Não devem existir elementos inadequados ou irrelevantes que desviem a atenção do usuário das informações relevantes;
10. **Ajuda e documentação:** Deve haver ajuda ao usuário, fácil e simples de usar e tem de ser aplicável ao contexto de uso.

Confiabilidade de Software

Além de cumprir as funções previamente designadas e planejadas pelos envolvidos no projeto, o software também deve ser suficientemente robusto para que não ocorram falhas que prejudiquem sua utilização. Esse requisito de qualidade está fortemente presente e relacionado à projetos de sistemas críticos, onde um mau-funcionamento pode ter graves consequências para seus usuários.

As falhas de software são relativamente comuns. Na maioria dos casos, essas falhas causam inconveniências, mas não danos sérios no longo prazo. No entanto, em alguns sistemas, as falhas podem resultar em perdas econômicas significativas, danos físicos ou ameaças à vida humana. Esses sistemas são chamados sistemas críticos. São sistemas técnicos ou sociotécnicos dos quais as pessoas ou negócios dependem. Se esses sistemas falharem ao desempenhar seus serviços conforme esperado, podem causar sérios problemas e prejuízos significativos [28].

A preocupação com a confiabilidade norteia os sistemas críticos, neles as atividades de validação e verificação consomem muito mais recursos do que o normal. No entanto, a confiabilidade deve estar presente em qualquer software, em maior ou menor grau.

Não há nenhuma dúvida de que a confiabilidade de um programa de computador é um elemento importante de sua qualidade global. Se um programa falhar frequentemente e repetidas vezes, pouco importa se outros fatores da qualidade de software sejam aceitáveis [26].

De acordo com o modelo de qualidade estabelecido pela norma ISO/IEC 9128, a confiabilidade é uma característica relativa à capacidade do produto de software de manter um nível de desempenho especificado, quando usado em condições especificadas [15].

Diferentemente de outros fatores de qualidade, a confiabilidade pode ser medida de forma mais direta e precisa utilizando técnicas de estatística. A confiabilidade de software é definida em termos estatísticos como a probabilidade de operação sem falhas de um programa de computador em dado ambiente por determinado intervalo de tempo [21].

Avaliação da Confiabilidade

Considerando um sistema computacional qualquer, uma medida de confiabilidade simples é o tempo médio entre falha MTBF(*mean time between failure*), que é dado em função da soma do tempo médio para falhar MTTF(*mean time to fail*) e o tempo médio para restauração MTTR(*mean time to restore*) [26].

$$mtbf = mttf + mttr \tag{2.1}$$

Na operação de software pelo usuário, algumas falhas apresentadas podem ser corrigidas ou recuperadas pelo próprio sistema, nesse caso MTTR está relacionada à capacidade do software de recuperar de seus erros, em casos de falhas irrecuperáveis e que geralmente terminam a execução, pode ser atribuído ao MTTR o tempo de execução necessário para que seja alcançado, ou restaurado, o estado de execução no qual a falha surgiu, isso inclui todo o processo desde a reexecução do software.

Na métrica utilizada para esse trabalho o MTTR utilizado para casos de falhas irrecuperáveis será de 40 segundos e será documentada no relatório como FI (Falha irrecuperável), e o utilizado para falhas recuperáveis será de 5 segundos e será documentada como FR (Falha recuperável).

2.2.2 Processo de avaliação ISO/IEC 25040

Esse conjunto de normas contém diretrizes de uso para avaliação de qualidade de software. Fornecem uma descrição de processo para avaliações e inclui os requisitos para a aplicação desse processo. Esse processo constitui uma base de avaliação e pode ser utilizado para diferentes propósitos e abordagens. Portanto, podem ser utilizadas para avaliação da qualidade externa e interna, além da qualidade em uso de softwares previamente desenvolvidos ou softwares customizados em fase de desenvolvimento. É destinada ao uso por responsáveis pela avaliação de software e é adequada para desenvolvedores, compradores e outros interessados em avaliações independentes [16].

Essas normas possuem origem na família de normas ISO/IEC 14598. Com o surgimento da série de normas SQuaRE (System and Software Quality Requirements and Evaluation), algumas dessas normas foram revisadas, substituídas e realocadas.

A norma guia ISO/IEC 25000 contém explicação detalhada do processo de transição das antigas normas ISO/IEC 9126 e ISO/IEC 14598 para SQuaRE [14]

O processo de avaliação deve ser utilizado junto de um modelo de qualidade que servirá como uma base para o estabelecimento de métodos que possam mensurar as características do software. Entretanto, o processo de mensurar diretamente essas características não é prático. É necessário o desdobramento de características em subníveis que possam ser mais facilmente medidas.

A norma apresenta alguns conceitos que devem nortear o processo de avaliação, a fim de minimizar sua natureza subjetiva. O processo deve possuir os seguintes atributos: [9]

- **Repetível:** Avaliações repetidas de um produto realizadas por um mesmo avaliador e com especificações de avaliação iguais devem produzir resultados aceitos como idênticos;
- **Reproduzível:** A avaliação do mesmo produto com uma mesma especificação de avaliação, realizada por um avaliador diferente, deve produzir resultados que podem ser aceitos como idênticos;
- **Imparcial:** A avaliação não deve ser influenciada frente a nenhum resultado particular;
- **Objetivo:** Os resultados da avaliação devem ser factuais, ou seja, não influenciados pelos sentimentos ou opiniões do avaliador.

O processo de avaliação descrito é composto por 4 fases: estabelecimento de requisitos de avaliação, especificação de avaliação, projeto de avaliação, execução de avaliação. Cada uma das fases possuem suas próprias atividades distintas, como mostrado a seguir.

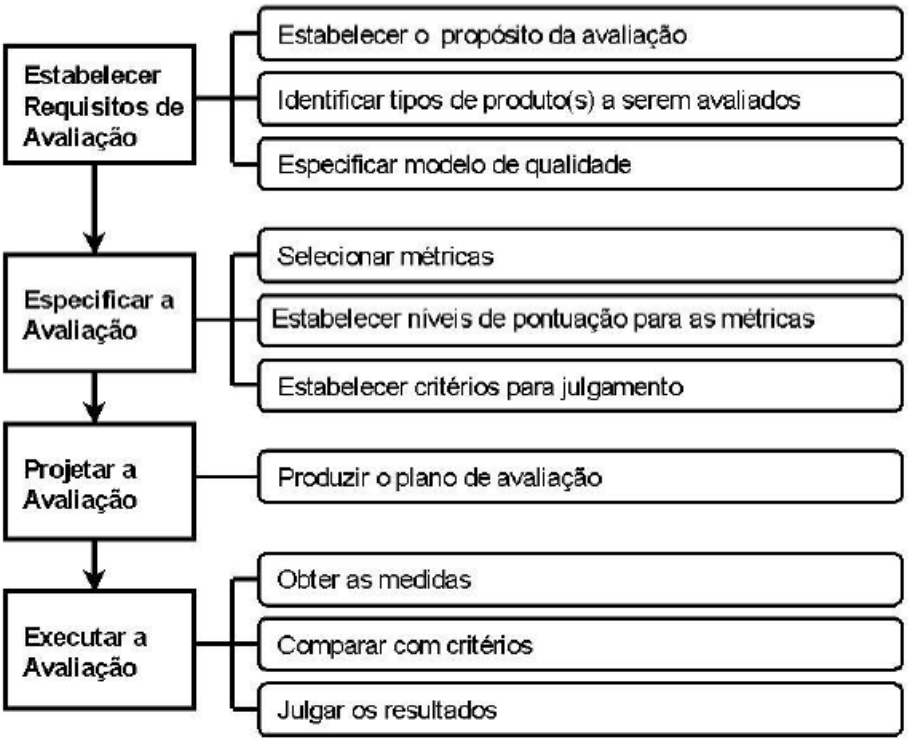


Figura 2.6: Processo de avaliação (Fonte: [29]).

Etapas da avaliação

A seguinte sequência de passos é descrita na norma:

1. Estabelecer requisitos de avaliação:

   (a) Estabelecer o propósito da avaliação: Toda avaliação deve ter um propósito, um software pode ser avaliado por diversos motivos. O avaliador pode estar interessado em compará-lo a outros softwares, adequá-lo a alguma norma estabelecida, identificar pontos à serem melhorados para versões futuras;

   (b) Identificar tipo(s) de produto(s): Conhecer o produto de forma de forma a compreender de que forma será usado, como será usado, suas características gerais, muitas vezes vários produtos podem ser analisados conjuntamente e de forma similar por pertencerem às mesmas categorias;

   (c) Especificar modelo de qualidade: É importante o avaliador compreender quais características do produto devem ser abordadas na avaliação, um modelo de qualidade genérico como o proposto na ISO/IEC 9126 pode ser utilizado e adequado em caso de necessidade;

2. Especificar a avaliação:

   (a) Selecionar métricas: É necessário selecionar métricas criteriosamente, elas são necessárias para que seja obtido um resultado numérico significativo que possa ser atribuído a determinada característica apresentada no modelo de qualidade, uma forma comum de utilizar métricas é elaborando um questionário com respostas objetivas, que estejam diretamente relacionadas às características e subcaracterísticas a serem avaliadas. Existem diversas métricas disponíveis para utilização de avaliadores, algumas delas são sugeridas nas normas ISO/IEC 25022 (métricas de qualidade interna) e 25023 (métricas externas e para qualidade em uso). Além disso, o avaliador poderá utilizar a norma ISO/IEC 25021 como um guia para elaborar novas métricas, ou adequar métricas já existentes;

   (b) Estabelecer níveis de pontuação para as medidas: As medidas obtidas devem ser mapeadas para números, geralmente essas medidas são valores nominais obtidos do questionário, como “Sim”, “Não”, “Algumas Vezes”, esses dados devem ser submetidos à algum processo que os transformem em números para uma avaliação efetiva. Uma forma usual de se fazer isso, é normalizar todas essas respostas para uma escala de 0 a 1. Existem casos onde a métrica não pode ser obtida para determinado software, nesses casos pode ser atribuído um valor específico como “N/A” (não se aplica) e deve-se ignorar o item para que não prejudique o resultado final;

(c) Estabelecer critérios para julgamento: Uma vez que são obtidos os dados numericamente, é ainda necessário estabelecer um processo de julgamento para que possa haver uma interpretação dos resultados. Podem ser aplicados diferentes pesos para características do modelo, de acordo com sua importância, por exemplo;

3. Projetar a avaliação:

   (a) Produzir o plano de avaliação: A escolha e definição de um método são muito importantes para o resultado final da avaliação, o processo básico é extrair uma resposta objetiva do avaliador. Esse processo deve ser pensado em termos de métodos que minimizem o erro associado presente nessa resposta. No caso de uma lista de verificação, devem ser levados em conta a ordem das questões, o limite de tempo para a entrega dos resultados, etc. Também faz parte dessa etapa a elaboração de um conjunto de instruções detalhado destinado ao avaliador;

4. Executar a avaliação:

   (a) Obter as medidas: As medidas selecionadas são aplicadas ao software pelo avaliador e então é aplicado o método de extração de resposta previamente definido, no caso da utilização de lista de verificação, as questões são respondidas de acordo com sua valoração pré-estabelecida;

   (b) Comparar com critérios: Os valores numéricos obtidos por si só não representam informação útil, nessa etapa pode ser utilizada uma escala para comparar. Uma vez que o valor esteja posicionado dentro da escala, a informação deve ser comparada e interpretada com os critérios utilizados;

   (c) Julgar os resultados: Os valores numéricos obtidos por si só não representam informação útil, nessa etapa pode ser utilizada uma escala para comparar. Uma vez que o valor esteja posicionado dentro da escala, a informação deve ser comparada e interpretada com os critérios utilizados.

2.3 Teste de Software

As atividades de teste de software são utilizadas para que o software final seja entregue com o menor número possível de erros. Seu intuito é executar um programa ou modelo utilizando algumas entradas em particular e verificar se seu comportamento está de acordo com o esperado [7]. Existem situações onde é necessária a comprovação da presença de determinadas funcionalidades no software, nesse caso particular o teste é chamado teste

de especificação [28]. Porém, na maior parte do ciclo de vida do software, a atividade de testes deve ser executada de maneira a encontrar erros. Existe, portanto, uma diferença antagônica entre desenvolvimento e teste, o primeiro processo é construtivo, o segundo destrutivo [22].

Na área de teste de software é necessário determinar com maior especificidade o significado de alguns termos. Erro, falha, defeito e engano são conceitos que devem ser definidos e compreendidos. De forma geral na literatura da área, defeito é uma definição incorreta que se faz presente em um trecho de código do software, engano é a ação humana que produz o defeito, um erro de digitação do programador por exemplo. Os conceitos de erro e falha estão relacionados à dinâmica do software, isso é à sua execução. O erro é um estado inesperado atingido pelo software, caso esse estado inesperado seja refletido em uma saída de dados incorreta para o usuário, é dito que o erro gerou uma falha no software para aquela situação de execução em particular.

Uma estratégia de teste de software deve acomodar testes de baixo nível, necessários para verificar se um pequeno segmento de código fonte foi implementado corretamente, bem como testes de alto nível, que validam as funções principais do sistema de acordo com os requisitos do cliente [26]. É interessante perceber que, segundo a definição feita dos termos, apenas os testes de baixo nível são capazes de reconhecer defeitos e enganos, pois eles envolvem a análise e exercitação diretamente do código-fonte do software. Os testes que envolvem execução dinâmica do software, quando utilizados isoladamente somente podem revelar falhas e os erros que as originam.

Dada a relação direta entre defeito, engano, erro e falha, fica evidente perceber que a abordagem mais eficiente do ponto de vista de testes é preventiva. Ao iniciar as atividades o quanto antes possível, um defeito causado por um engano corrigido em fase inicial de codificação não se tornará uma falha na utilização do sistema após o lançamento do software.

Bons testes são aqueles que exercitam situações particularmente problemáticas para o software, pois testes nos quais toda a sequência possível de ações é testada são impossíveis na prática [7]. Segundo *Djikstra*, testes somente sinalizam a presença de defeitos, nunca a sua ausência. Portanto, a abordagem para a seleção de itens a serem testados deve ser conduzida e baseada em riscos, assim, os cenários mais prováveis de uso de um software têm prioridade e devem ser determinados e testados o quanto antes.

Quando a atividade de teste é realizada de maneira criteriosa e é embasada tecnicamente, obtém-se uma confiança de que o software se comporta corretamente para grande parte do seu domínio de entrada [7].

2.3.1 Processo de Teste

O teste de software consiste em aplicar um conjunto de dados T, obtido de um conjunto do domínio de entrada D(P), em um software P.

A observação desse fenômeno irá determinar o sucesso ou falha. No caso demonstrado a seguir, o procedimento que determina o resultado final está definido em função dos requisitos do software e é executado pelo oráculo S(P). Se, para algum caso de teste, o resultado obtido difere do esperado, então um defeito foi revelado. A figura a seguir ilustra o processo.

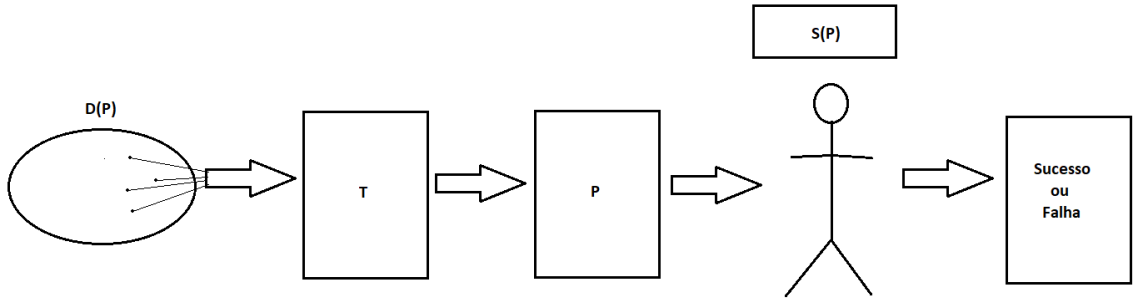


Figura 2.7: O processo de teste de software (Fonte: [7]).

Na maioria dos casos, o papel de oráculo é feito pelo próprio testador, mas existem casos onde esse processo pode ser automatizado e sem interação humana.

O primeiro passo para a atividade de teste ilustrada é identificar o conjunto D(P) que é o domínio de entrada e possui todas as entradas possíveis para P. Em um caso onde P é um software matemático que realiza uma operação qualquer, D(P) seria o domínio de entrada de dados permitido pelo software, caso o software seja adequadamente implementado, somente serão permitidos entradas para as quais a operação está definida matematicamente.

No caso de um software interativo com interface gráfica, onde o mouse é o dispositivo de entrada utilizado, D(P) pode ser definido como o conjunto de ações que podem ser praticadas pelo usuário, isso inclui clicar, arrastar, mover o mouse para determinada posição, etc. Para determinar e identificar um subconjunto de dados de teste são estabelecidas certas regras para definir quando dados de teste devem estar no mesmo subdomínio, ou não. Em geral, são definidos requisitos de teste, como por exemplo, executar uma determinada estrutura do software. Os dados de teste que satisfazem esses requisitos pertencem ao mesmo subconjunto [7].

É evidente que o sucesso de um processo de testes está relacionado à habilidade do testador em encontrar o subconjunto de dados de teste ideal que irá revelar o maior número de defeitos possível dentro do software. É necessário também minimizar esse subconjunto, pois o tempo é uma variável que limitará em algum momento a atividade.

2.3.2 Teste de unidade e integração (caixa-branca)

A atividade de testes de forma geral possui diferentes fases, com diferentes objetivos.Nas fases iniciais de desenvolvimento, os testes de unidade são utilizados para verificar o comportamento das menores unidades do software, que podem ser funções, procedimentos, métodos ou classes [7].

O teste de unidade tem por objetivo explorar a menor unidade do projeto, procurando identificar erros de lógica e de implementação em cada módulo, separadamente [5]. Uma vez que é demonstrado o funcionamento esperado das unidades individualmente, é necessário testar se a interação entre as unidades está adequada.

O teste de integração tem ênfase na construção da estrutura do sistema. À medida que as diversas partes do software são colocadas para trabalhar juntas, é preciso verificar se essa interação funciona de maneira adequada e não introduz possíveis erros [7].

Testes de unidade e integração possuem uma característica em comum importante, são aplicados diretamente no código-fonte do software, por isso eles são agrupados na categoria de teste caixa-branca. São testes que dependem do conteúdo interno, ou seja, código do software.

Após essa fase de testes com código, o software passa por testes de ordem superior, em que seus procedimentos estão mais relacionados à requisitos do projeto e não ao código ou métodos de implementação, são chamados testes funcionais. A figura a seguir ilustra as diferentes fases do processo de testes.

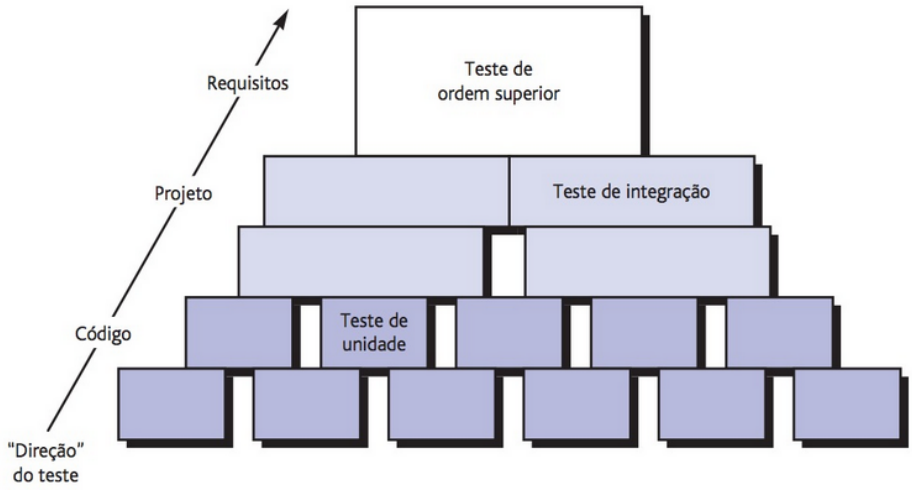


Figura 2.8: As fases dos testes de software (Fonte: [26]).

2.3.3 Teste funcional (caixa-preta)

O teste funcional, também chamado teste de sistema, acontece geralmente após toda a integração das diferentes partes e concentra-se em verificar se as funcionalidades requeridas para o software estão corretamente implementadas.

Aspectos de correção, completude e coerência devem ser explorados, bem como requisitos não funcionais como Segurança, Performance e Robustez. Esse é o tipo de teste utilizado num contexto de avaliação de software já lançado, pois o objeto de avaliação é o software e seu funcionamento e não seu código-fonte [7].

Por esses motivos, o teste funcional é conhecido também como teste caixa-preta. Nele, o software é enxergado como uma caixa em que apenas entrada e saída são importantes, sem considerações à sua estrutura interna de codificação. Essencialmente, esse é o procedimento que define a adequação de um software à seus requisitos, pois estruturas internas e código considerados isoladamente não determinam a presença de funcionalidades no produto final.

Os testes funcionais são diretamente dependentes dos requisitos funcionais do software, portanto a qualidade dessa atividade de testes está diretamente relacionada à qualidade da documentação de requisitos, ou do conhecimento do sistema por parte do responsável pelo teste.

As principais estratégias e técnicas utilizadas nos testes funcionais são definidas a seguir.

Partição em classes de equivalência

A técnica base de minimização é particionar o conjunto domínio de entrada em subconjuntos menores de forma que, para os elementos desses novos subconjuntos possa se assumir que o software tem o mesmo comportamento, à esse conjunto também se dá o nome de classe de equivalência. A figura a seguir ilustra a utilização da técnica numa situação abordada neste trabalho.

Figura 2.9: Exemplo de uso do Software D.

No caso de um teste funcional como esse, é importante que se conheça a especificação do software, pois essa será a base para encontrar subconjuntos de interesse no domínio de entrada. Na figura acima é demonstrada uma tela interativa onde o usuário deve digitar o nome do objeto mostrado no canto superior esquerdo da figura.

Nesse caso, o programa verifica, se o usuário digitou o conjunto de letras “faca”, ao digitar corretamente é exibido um vídeo na tela. A ordem das letras é importante, mas para cada uma das quatro posições o usuário pode cometer um número indeterminado de erros.

Portanto, “faca” é equivalente a “afbaecad” e também equivalente a “abcfacacde”, então podemos generalizar e concluir que para todo conjunto de caracteres contendo a palavra “faca”, o software terá comportamento igual, ou seja, para qualquer elemento escolhido da classe de equivalência, o software exibirá o vídeo de sucesso na tela.

Hipoteticamente, se fosse encontrado uma falha em que o software reconhecesse o símbolo ç como “c”, o que significa que a palavra faça, seria aceita como correta. Poderia-se , então, generalizar o comportamento e criar uma classe de equivalência de erros para todas as outras palavras semelhantes apresentadas pelo software.

Análise do valor-limite

A experiência mostra que dados de teste que exploram as condições limite de uma funcionalidade tem maior chance de revelar falhas [22]. Condições limite são aquelas situações onde uma determinada entrada está, diretamente acima ou abaixo dos limites especificados para esta entrada. Nesse caso devem ser determinados dados de teste, que exercitem os limites derivados da especificação.

Os dados de teste podem ser facilmente encontrados em muitas ocasiões, especialmente quando as especificações possuem palavras como: no máximo, entre, pelo menos. Também existem casos onde a experiência do testador está envolvida na derivação dos dados, a figura a seguir é um exemplo onde esse fator é explorado.

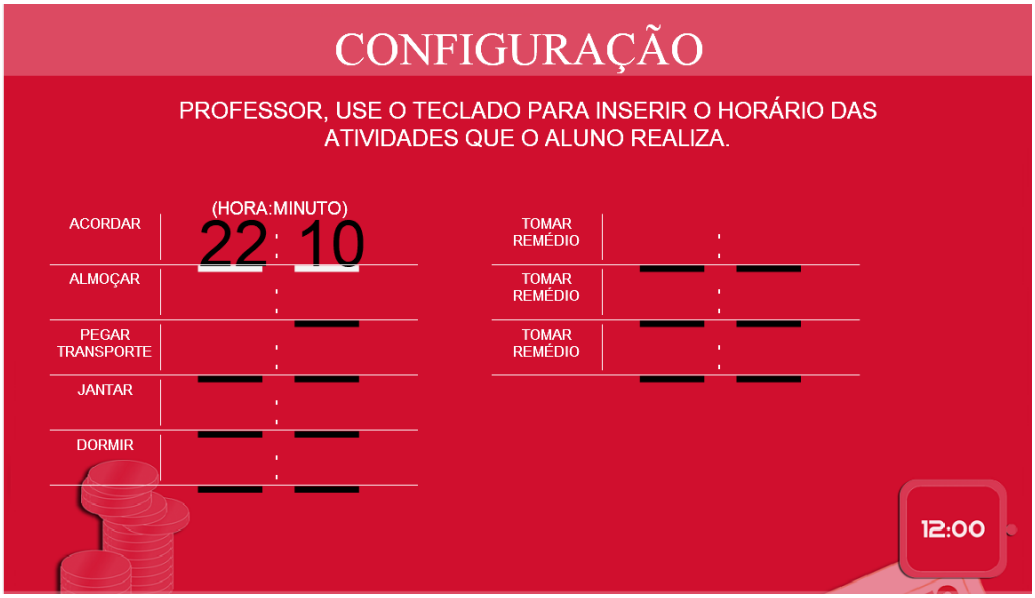


Figura 2.10: Exemplo de uso do Software E2.

Essa é uma tela interativa onde o usuário deve digitar nos campos de horários, um horário válido no formato HH:MM, sabe-se por experiência que o valor de hora pode variar entre 0 e 23, e o do minuto pode variar entre 0 e 59. Com base nessas informações, os valores limite, e os valores próximos ao limite que também são interessantes para os casos de teste desse tipo, são facilmente determinados.

Grafo causa-efeito

A técnica de utilização de um grafo causa-efeito ajuda na definição de um conjunto de casos de teste que exploram ambiguidades e incompletude das especificações [7]. Portanto, são especialmente úteis para especificações onde existem combinações de causas e diversos efeitos associados, pois nesses casos a linguagem natural de especificação de requisitos pode

tornar a descrição da funcionalidade, ineficiente. Um grafo causa-efeito é uma linguagem formal para a qual uma especificação em linguagem natural é traduzida[22].

Para derivar o grafo causa-efeito de uma especificação, é preciso primeiramente dividi-la em pedaços menores, isso é necessário pois o grafo torna-se complexo muito facilmente. A estratégia é dividir um grande processo em subprocessos menores que possam ser mais facilmente compreendidos e definidos.

Em um segundo momento é necessário que sejam identificadas causas e efeitos para o processo escolhido. Na tela da figura a seguir, o usuário deve simular uma compra, de forma que os produtos escolhidos não ultrapassem um valor anteriormente definido. Caso o usuário ultrapasse o limite o processo de transição para uma próxima etapa ou processo deve ser impedido.

Figura 2.11: Tela interativa do Software E2.

Depois disso causas e efeitos devem ser relacionados usando conectivos lógicos conhecidos, como, e, ou. Esses relacionamentos devem ser então passados para uma representação concreta e gráfica que é o próprio grafo causa-efeito.

A partir disso é possível extrair casos de teste diretamente, percorrendo os diversos caminhos do grafo, isso pode ser estruturado de forma mais simples numa tabela de decisão. A figura a seguir demonstra uma representação simplificada do grafo causa-efeito para o processo de interação na tela anteriormente descrita.

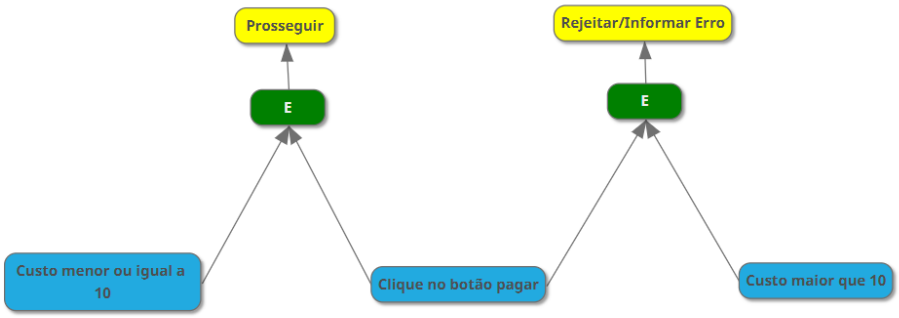


Figura 2.12: Exemplo de grafo causa-efeito.

Nesse processo simplificado, existem três causas apenas, elas se relacionam através de conectivos lógicos que por sua vez, se ligam à um dos efeitos definidos.

Num caso onde é exigido um maior rigor, teriam de ser descritas todas as causas isoladamente e todas as suas possíveis conexões lógicas também, excluindo aquelas que são antagônicas, como por exemplo a causa "Custo menor ou igual a 10" conectada logicamente com "Custo maior que 10" através do símbolo lógico "E", é uma situação inválida para esse processo em questão. A estruturação em tabela de decisão desse grafo seria simples, pois existem apenas dois casos descritos. A valoração é booleana, 1 representa verdadeiro e 0 falso.

Tabela 2.1: Tabela de decisão gerada pelo grafo causa-efeito.

Causa/Efeito	**Prosseguir**	**Rejeitar/ Informar Erro**
C >=10 E Clique no botão	0	1
C < 10 E Clique no botão	1	0

Error guessing e teste exploratório

Nesses métodos a principal fonte de casos de teste é uma mistura de experiência prévia com intuição do testador. Em um caso onde um testador é fortemente familiarizado com um tipo de aplicação, por exemplo, Editores de Texto. Pode-se concluir intuitivamente que existe uma maior probabilidade de que ele conseguirá construir casos de teste mais interessantes, do que outro testador que não possui a mesma experiência.

É difícil definir e descrever rigorosamente um procedimento para essa técnica, pois é um processo intuitivo e de natureza *ad-hoc*. Por exemplo, podemos dizer de forma geral que a presença do 0 em uma entrada é uma situação propensa a revelar defeitos, portanto deve-se produzir um caso de teste com a entrada 0, e também casos para os quais a saída é 0 [22]

Um conceito importante e emergente na área e relacionado diretamente com Error Guessing é o teste exploratório, que é definido como um método onde o testador desenvolve e executa os casos de teste, na medida em que se familiariza com o sistema [3]. Esse método pode ser particularmente interessante quando um software é pequeno e os requisitos explicitados em documentos não são adequados, estão incompletos, ou até mesmo quando a documentação está ausente.

De certa forma todo o teste é exploratório em algum grau, pois até mesmo os processos de testes mais documentados envolvem o aprendizado da interface por parte do responsável pelo teste. Um exemplo de caso de teste derivado de error guessing é mostrado a seguir.

Figura 2.13: Exemplo de uso da técnica Error Guessing.

O botão fotografia permite que o usuário escolha um arquivo de foto para fazer uma auto-representação numa tela adiante. Ao ser clicado no botão de escolha, uma caixa de diálogo é exibida onde o usuário pode navegar até uma foto e fazer sua escolha. É feita

uma restrição simples para o tipo de arquivo baseada em extensão, sendo aceitos apenas arquivos com extensões conhecidas para imagens. Com o objetivo de derivar casos de teste interessantes, pode ser observado, por exemplo, o comportamento do software ao ser escolhido um arquivo qualquer com uma falsa extensão de imagem.

2.3.4 Automatização de testes

A atividade de testes geralmente envolve uma repetição exaustiva de certos procedimentos, isso acarreta alguns problemas. O mais óbvio deles é o tempo excessivo consumido durante essas atividades, outro aspecto importante é o fator de erro humano envolvido em testes manuais. Dada uma probabilidade fixa para que aconteça um erro humano durante um processo, obviamente, quanto mais vezes o processo se repete, maior é a chance do erro humano ser introduzido.

Um ponto importante para o sucesso no teste de um software é a automação. Diversos tipos de ferramentas têm sido utilizadas para buscar aumentar a produtividade nessa atividade, que tende a ser extremamente dispendiosa [7].

O objetivo de utilizar ferramentas de automação de testes, é livrar o testador de todos os procedimentos cansativos e repetitivos associados à atividade. Também é desejável explorar a possibilidade que as ferramentas oferecem de armazenar e gerenciar grandes quantidades de dados [10].

Qualquer etapa de desenvolvimento de software que envolva testes pode ser auxiliada por essas ferramentas. Desde as primeiras etapas de teste de unidade, até os testes funcionais de mais alto-nível, todas elas envolvem alguns processos repetitivos, portanto podem ser abordadas por alguma ferramenta de automação.

2.3.5 Ferramentas para automatização

Como as ferramentas de auxílio à automação de testes são de diversos tipos e têm objetivos diferentes, é importante conhecer uma classificação que possa guiar na direção das ferramentas mais adaptadas ao contexto em que serão utilizadas. Essa é uma classificação de ferramentas baseada em suas funcionalidades [4].

- Frameworks de testes automatizados: Fornece um ambiente controlado no qual os testes podem ser iniciados e a saída do teste pode ser registrada. Para executar partes específicas de um programa, drivers e stubs têm a função de representar módulos que chamam, ou são chamados respectivamente;
- Geradores de teste: Fornece assistência no processo de gerar casos de teste. A geração pode ser aleatória, baseada em caminhos, baseada em modelos, ou uma mistura disso tudo;
- Ferramentas de captura e replay: Automaticamente reexecutam testes previamente executados que têm suas entradas e saídas gravadas, são ferramentas relacionadas à testes funcionais e testes de regressão;
- Oráculos/Comparadores de arquivos/asserção: Auxiliam no processo de decisão de sucesso ou falha de um teste;
- Analisadores de cobertura e instrumentadores: Avaliam quais e quantas entidades, através de um gráfico de fluxo, do programa foram exercidos dentre todos aqueles exigidos pela critério de cobertura de teste adotado. Isso é feito graças aos instrumentadores;
- Tracers: Gravam o histórico de caminhos da execução de um programa;
- Ferramentas para teste de regressão: Dão suporte à reexecução de uma suíte de testes após mudanças no código do software. Também podem ajudar a selecionar um subconjunto de testes de acordo com a mudança promovida;
- Ferramentas de análise de confiabilidade: Auxiliam na análise de resultados de testes e na visualização gráfica a fim de avaliar as medidas relacionadas à confiabilidade de acordo com modelos selecionados;

SikuliX

SikuliX é uma ferramenta open-source sob licença MIT e é publicamente disponível para qualquer uso [12], é também multi-plataforma operando em todos os sistemas operacionais mais populares: Windows, Linux e Mac OS.

A ferramenta permite automatizar interações de usuários com aplicativos através da criação de scripts, que podem ser escritos em Python, Ruby ou Javascript. As interações mencionadas podem ser através do uso de um mouse, como clique, clique duplo, e também com os eventos do teclado, para todas essas ações existem métodos e funções equivalentes de simulação em script. Como mouse e teclado são os dois dispositivos básicos de entrada com os quais as pessoas interagem no computador, uma vez que a ferramenta tem controle desses dois dispositivos de forma robusta, seu poder torna-se grande o suficiente para automatizar qualquer tipo de tarefa. Segundo o autor da ferramenta, existem três tipos de ferramentas para automação de tarefas [11]:

- Recorder: Esse tipo de ferramenta é capaz de gravar todo o tipo de interação do usuário de forma com que seja possível reproduzir fielmente essa interação sempre que necessário. Geralmente é possível editar essas gravações e acrescentar novos elementos desejados pelo usuário;
- GUI aware: Permite que sejam operados elementos gráficos como botões. Isso é feito com base no conhecimento da estrutura interna (código) e nomes dos elementos da interface gráfica, algumas dessas ferramentas também possuem mecanismos de gravação e reprodução;
- Visually: A ferramenta enxerga imagens através da técnica de visão computacional (geralmente áreas de pixels de formato retangular) na tela e permite que sejam feitas as interações simuladas já mencionadas. Também podem ter mecanismos de gravação, mas não necessariamente.

SikuliX pertence a terceira categoria. Através do uso de técnicas de visão computacional, o software reconhece imagens de interesse, no caso de testes de interface gráfica as imagens de interesse são todos os elementos gráficos que sejam importantes e que interfiram de alguma forma no funcionamento do programa, isso inclui no caso de testes de interface, botões, títulos de telas, menus, dentre outros elementos. A imagem a seguir demonstra algumas das áreas de interesse numeradas, para testes de uma tela em particular.

Figura 2.14: Áreas de interesse para teste da interface.

Airdroid

A ferramenta AirDroid possibilita o controle de dispositivos Android como celulares e tablets através de um PC e uma conexão em rede. A aplicação cliente que roda no PC, controla em tempo real o dispositivo Android que possui a instância servidor do Airdroid. A funcionalidade AirMirror espelha a tela do dispositivo e permite o controle de sua interface gráfica, todos os comandos gestuais como tocar, arrastar, digitar são emuladas através do mouse e teclado.

Dessa forma é possível executar virtualmente qualquer software nativo do sistema operacional Android no PC, durante o trabalho essa funcionalidade foi utilizada juntamente com o Sikulix para elaboração dos teste de interface automatizados.

A figura a seguir demonstra uma conexão simples de um PC e um Dispositivo Android através da ferramenta.

Figura 2.15: Dispositivo Android sendo espelhado.

AirDroid não é uma ferramenta específica para testes, mas pode auxiliar na execução em casos onde os softwares avaliados rodam em sistemas operacionais diferentes.

Capítulo 3

Metodologia

A base metodológica para o planejamento de avaliação foram as normas presentes na família ISO/IEC 14598. São aqui explicados os conceitos mais importantes relativos à avaliação e planejamento.

Quanto ao primeiro grupo de sub-atividades descritas na norma, nomeadas respectivamente: Estabelecer o propósito da avaliação, Identificar tipo(s) de produto(s), Especificar modelo de qualidade.

1. O propósito da avaliação é construir uma base de conhecimento sobre o funcionamento de todos os softwares para que sejam identificados pontos que possam ser melhorados em versões futuras ou até mesmo serem lançadas correções e melhorias para as versões atuais, além disso os softwares possuem muitas características semelhantes e público similar, assim é importante compará-los de alguma forma. Também é importante, devido à natureza dos softwares, verificar a correspondência entre itens especificados em documentação e as funcionalidades;

2. Para identificar as características dos softwares, a estratégia foi analisá-los em busca de pontos em comum e que pudessem ser explorados de uma forma mais sistemática em termos de suas características. Feito isso, aplica-se uma abordagem com ferramentas e processos efetivos para dar suporte às atividades.

3. O modelo de qualidade básico estabelecido na ISO/IEC 9126-1 foi adaptado às necessidades da avaliação, e com base na natureza e no conhecimento dos softwares apresentados foram escolhidos os dois fatores de qualidade considerados mais relevantes: Usabilidade e Confiabilidade. Também foi avaliada e considerada como uma característica básica do modelo a Documentação, pela sua importância para esse tipo de software;

Quanto ao segundo grupo de sub-atividades descritas na norma. Selecionar Medidas, Estabelecer níveis de pontuação e Estabelecer critérios para julgamento e Plano de Avaliação

4.

(a) Para a avaliação de Usabilidade, foi utilizado o método de inspeção baseado nas heurísticas de Nielsen. Cada um dos softwares teve todas as suas telas acessadas e inspecionadas. É essencial que se conheça o software em questão e todas as suas funcionalidades e possibilidades de fluxo entre as diferentes telas, para alcançar tal objetivo foram utilizadas técnicas e métodos exploratórios, onde o avaliador testa e explora o software simultaneamente. Pela natureza dos softwares avaliados, e por não serem tão complexos e numerosos em termos de funcionalidades, os métodos exploratórios foram suficientes para que todas as telas fossem percorridas, em alguns poucos casos de dúvidas sobre o funcionamento, a Documentação foi utilizada. Assim, todas as telas foram inspecionadas e essa análise foi transformada e documentada em formato de uma tabela. Durante testes de usabilidade, aspectos da funcionalidade do software também acabam sendo testados, portanto os relatórios de usabilidade também incluem informações sobre a adequação dos requisitos funcionais. O relatório completo inclui o nome ou descrição da tela onde o incidente foi descoberto, e o caminho percorrido pela interface do software para encontrá-la. A medida derivada da inspeção é a contagem de incidentes, e a classe à que pertencem;

(b) Na análise da Documentação foi feita checagem e confrontamento entre as funcionalidades apresentadas nas interfaces do software e as informações fornecidas nos documentos. Na prática, foi respondida uma única questão é: *A funcionalidade X apresentada na documentação encontra-se no software?* Com os valores possíveis de resposta Sim e Não, e depois é feita a contagem. Assim como na avaliação de Usabilidade, o conhecimento profundo de cada software torna esse processo de verificação mais ágil e confiável;

(c) Para avaliar a Confiabilidade, uma vez que todo os softwares estavam cobertos por testes automatizados, foi possível escolher determinadas funcionalidades e analisá-las conjuntamente quanto à sua confiabilidade em relação ao tempo de uso que tinha duração de uma sessão de teste, 30 minutos. O conjunto de funcionalidades escolhidas para serem alvo do teste formam o /emphCiclo de Atividades que é descrito no relatório completo. A métrica utilizada e calculada para o resultado final foi o tempo médio entre falhas (MTBF).

5.

(a) Medida de Usabilidade

i. Cada incidente possui o valor unitário (1) e pertence ao conjunto de heurísticas de Nielsen enumerado de 1 a 10;

(b) Medida de Confiabilidade

i. Para cada software foi encontrado um resultado no domínio do tempo para a métrica MTBF, esse resultado é a média aritmética do MTBF encontrado em cinco testes feitos, idealmente, sob as mesmas condições

ii. Pode ser expresso pela fórmula

$$mtbf = (mtbf1 + mtbf2... + mtbf5)/5 \tag{3.1}$$

(c) Medida de Documentação

i. Para Sim será atribuído 1, para Não será atribuído 0;

ii. O resultado final r será o número de divergências encontradas entre documentação e interface do software.

6. Para todos os requisitos de qualidade será calculada uma média de pontuação global, a partir disso é possível encontrar aqueles que estão acima e abaixo da média comparativa para cada um dos requisitos;

7. O plano de avaliação foi feito com base nos recursos disponíveis à serem utilizados e sob o ponto de vista de uma avaliação independente feita por apenas um indivíduo.

(a) As sessões de análise de usabilidade foram feitas manualmente. Foram utilizadas e acessadas todas as telas e todas as funcionalidades de cada um dos softwares, e cada incidente foi reportado de acordo com a classe (heurística) à qual pertence numa tabela no mesmo momento em que foi descoberto, adicionalmente foi tirada uma foto da tela para ilustração em relatório;

(b) Na análise de Documentação, foi aberto simultaneamente uma janela do software e uma do documento, e então cada uma das funcionalidades descritas foram procuradas e acessadas dentro do software. No caso de um incidente encontrado, a página do documento foi marcada utilizando a própria funcionalidade do leitor Adobe Acrobat Reader, posteriormente a funcionalidade ausente foi documentada numa tabela de relatório;

(c) Para a Confiabilidade foi utilizada a ferramenta Sikulix em ambiente PC/Windows, foi programada para executar um ciclo de atividades durante um determinado período de tempo (sessão de teste). A função de oráculo não foi automatizada

durante a atividade, coube ao avaliador o papel de observador e relator. Cada incidente disparado foi anotado, mas só foi documentado no final de cada sessão para não prejudicar a atividade. No caso dos softwares nativos do Tablet Android, foi utilizada a ferramenta Airdroid já abordada anteriormente;

(d) O equipamento de hardware utilizado foi o seguinte: PC Intel Core i5-3330 3.0 GHz 8 GB RAM, rodando Windows, a resolução de tela utilizada para todos os testes e atividades foi de 1360 x 768. O Tablet Android tem a seguinte especificação: Samsung Galaxy Tab 3 10.1 polegadas (1600x2560) , modelo GT-P5200, 1 GB de ram e 16 GB de HD rodando Android 4.2.2.

3.1 Métodos e Ambiente de Testes

Durante a execução da avaliação foram utilizados métodos e ferramentas a fim de agilizar o processo e torná-lo mais robusto e confiável. São aqui descritos métodos e ferramentas utilizados para cada uma das três características abordadas, assim é possível que os processos utilizados originalmente possam ser reproduzidos com fidelidade. Durante a execução de todos os testes e análises foi utilizado o serviço na nuvem *Google Docs* para documentação e formatação dos resultados.

3.1.1 Usabilidade

Obviamente, é necessário conhecer e relembrar todas as diferentes heurísticas para o processo de inspeção, a utilização e constante revisão de uma tabela impressa contendo as heurísticas simultaneamente ao processo de inspeção foi útil para agilizar o processo. Além disso, como o relatório de usabilidade contém screenshots de todos os incidentes reportados, a utilização de um softwares específico para captura de tela também contribuiu positivamente, foi utilizado para este fim o software de captura de tela *LightShot*. Também é importante estabelecer uma ordem de inspeção, por exemplo as telas podem ser abordadas de acordo com a ordem alfabética dos botões que as acionam. A ordem é essencial para que todas as telas sejam inspecionadas e também para que não ocorram inspeções repetidas.

3.1.2 Confiabilidade

O método definido para testes de confiabilidade foi a utilização da ferramenta Sikulix para executar testes automatizados durante pré-determinado período de tempo que definiu-se como sessão de teste (30 minutos). Foi importante conhecer o funcionamento da ferramenta e de suas funcionalidades. A visão computacional utilizada internamente pelo

Sikulix para reconhecimento de áreas da interface do software cria algumas situações de dificuldade, por exemplo, botões muito semelhantes na interface podem ser interpretados de forma errônea, o que causa um mau-funcionamento no script. Também é importante notar que diferentes resoluções de tela fazem com que imagens sejam obviamente renderizadas de forma diferente na tela, um script Sikulix construído para ser executado em uma resolução terá, provavelmente, problemas de reconhecimento ao ser executado em outras resoluções diferentes. Além disso, é essencial que exista uma imagem para cada um dos botões e áreas de interesse à serem exploradas na interface, só assim elas podem ser referenciadas e utilizadas nos scripts criados. Mais uma vez a ferramenta especialista em captura de tela *Lightshot* mostrou-se importante para agilizar o processo de captura das imagens da interface.

3.1.3 Documentação

Foi feita a inspeção no documento simultaneamente à execução e verificação de cada um dos softwares. Essa foi a análise mais simples de ser feita, nenhum tipo de dificuldade foi encontrada durante esse processo.

3.2 Softwares Avaliados

Foram avaliados 9 softwares educacionais. A tabela a seguir apresenta os softwares.

Tabela 3.1: Tabela de Softwares Avaliados.

Nome	**Descrição**	**Plataforma**
A	Software educacional para o trabalho de expressões faciais	Tablet Android
B	Software educacional para o trabalho da percepção visual para autistas	Tablet Android
C	Software educacional para o trabalho de organização de objetos	Tablet Android
D,D2	Software educacional para o trabalho de alfabetização	D para Tablet Android D2 para PC (Windows/Linux)
E,E2	Software educacional para o ensino de matemática	E para Tablet Android E2 para PC (Windows/Linux)
F	Software educacional para o trabalho de autocuidado	PC(Windows/Linux)
G	Software educacional para o ensino de gerenciamento do tempo	Tablet Android

Capítulo 4

Resultados

Os resultados finais resumidos estão contidos na tabela abaixo, para Usabilidade e Documentação são mostrados os resultados em número de incidentes, para Confiabilidade é mostrada a métrica MTBF:

Tabela 4.1: Resultados finais dos testes

Software	**Usabilidade (Total)**	**Confiabilidade (MTBF)**	**Documentação (Total)**
E2	18	16 min 27 seg	0
F	3	Sem falhas	1
E	12	143 min 5 seg	Documento Ausente
D	11	Sem falhas	0
D2	13	9 min 13 seg	0
A	7	Sem falhas	Documento Ausente
B	8	75 min 40 seg	0
G	9	Sem falhas	0
C	5	8 min 41 seg	0

A análise de Documentação demonstrou que quase todos os documentos dos softwares continham funcionalidades descritas que estavam de fato presentes na interface. Por outro lado dois dos softwares não continham documento para serem analisados. O único incidente descoberto na análise de documentação encontra-se na tabela a seguir.

Tabela 4.2: Resultados detalhados da análise da Documentação.

Software	**Página(s)**	**Descrição**
F	37	A funcionalidade “Passar Fio Dental” foi implementada incorretamente (sua primeira tela/etapa não é apresentada corretamente).

Para os resultados das análises de Usabilidade e Confiabilidade foi calculada a média aritmética para elaborar a seguinte tabela que demonstra a distância de cada software para a média calculada.

Tabela 4.3: Distância para a média global de incidentes de usabilidade para cada software.

Software	**Usabilidade (Média = 9)**	**Confiabilidade (Média = 51 min)**
E2	+9	-35 min
F	-6	Sem falhas
E	+3	+92 min
D	+2	Sem falhas
D2	+4	-42 min
A	-2	Sem falhas
B	-1	+24 min
G	0	Sem falhas
C	-4	-38 min

O valor médio apresentado de 51 min foi calculado apenas utilizando os valores dos softwares que apresentaram falhas, pois o MTBF dos demais softwares é indeterminado.

A tabela seguinte discrimina os incidentes da inspeção de Usabilidade de acordo com as classes (heurísticas de Nielsen) a que pertencem:

Tabela 4.4: Frequência absoluta de incidentes de usabilidade nos softwares de acordo com a classe heurística.

Software / Heurística Nº	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Total**
E2	1	4	0	4	0	4	0	0	4	1	18
F	0	0	0	1	0	0	0	0	2	0	3
E	0	1	0	3	1	2	0	0	3	2	12
D	0	6	0	2	0	1	0	0	1	1	11
D2	0	1	0	6	0	3	1	1	1	0	13
A	1	0	3	0	0	1	0	1	1	0	7
B	1	2	0	3	0	0	0	0	2	0	8
G	4	2	0	1	0	0	0	1	1	0	9
C	1	1	0	0	0	0	0	0	3	0	5
Total	8	17	3	20	1	11	1	3	18	4	86

A próxima tabela descreve a frequência relativa de cada classe heurística.

Tabela 4.5: Frequência relativa das classes heurísticas de Nielsen.

Heurística Nº	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Total**
Frequência Relativa	9%	20%	3,5%	23%	1%	13%	1%	3,5%	21%	5%	100%

4.1 Discussão e Sugestões

Os resultados para Documentação apontaram apenas uma divergência entre documento e interface do software, porém, dois dos softwares avaliados não apresentaram um documento de natureza descritiva similar aos outros. A ausência desses documentos pode ser prejudicial à compreensão pelo usuário, visto que o único meio e canal de aquisição e conhecimento dos softwares é a internet. Durante a análise dos documentos foi observado que podem ser formuladas outras métricas que tornariam o processo de avaliação mais robusto, um exemplo seria a utilização de uma métrica que verifica a conformidade das imagens utilizadas nos documentos com as imagens reais geradas pelo software em execução. Também foi verificado que esses documentos não são essencialmente destinados à utilização do seu público consumidor, ou seja, existem informações não relevantes "concorrendo"com as informações que são de fato destinadas ao seu público-alvo. A heurística de Nielsen do design minimalista pode ser utilizada também para documentos, e assim conclui-se que um documento destinado ao público-alvo deve conter apenas informação suficiente e necessária para o mesmo.

Todos os softwares que apresentaram falhas durante as sessões de testes de confiabilidade podem ser investigados de várias formas. Do ponto de vista do desenvolvimento pode ser analisado e verificado o código-fonte utilizando técnicas de debug e outros tipos de processos de verificação auxiliares. O relatório completo do teste de confiabilidade anexado, que contém dados importantes pode ser utilizado para reproduzir as falhas num ambiente de desenvolvimento, facilitando a busca pelos defeitos. Já pelo ponto de vista de um avaliador que deseja investigar mais a fundo, podem ser executados mais testes que exercitem as funcionalidades que apresentaram as falhas de forma que o defeito seja isolado e descoberto, testes de desempenho também podem ser úteis, pois existe a possibilidade da falha ser resultado de um mau gerenciamento de memória pelo software por exemplo.

As classes heurísticas mais citadas podem ser vistas pelos desenvolvedores dos softwares como pontos à serem estudados e aprimorados para versões futuras e/ou correções. Também é importante ressaltar a recomendação de que esse tipo de inspeção seja realizada por vários avaliadores e só então, a partir desses resultados conjunto, deve ser derivado um resultado final. Essa recomendação deve-se ao caráter altamente subjetivo dessa técnica de inspeção. Os resultados detalhados contém informações importantes que podem ser utilizadas como guia de futuras inspeções realizadas por desenvolvedores ou avaliadores.

Uma técnica produtiva de análise e desenvolvimento de software é realizar uma decomposição em termos de funcionalidades no formato de árvore, a imagem a seguir demonstra essa decomposição feita com o software B.

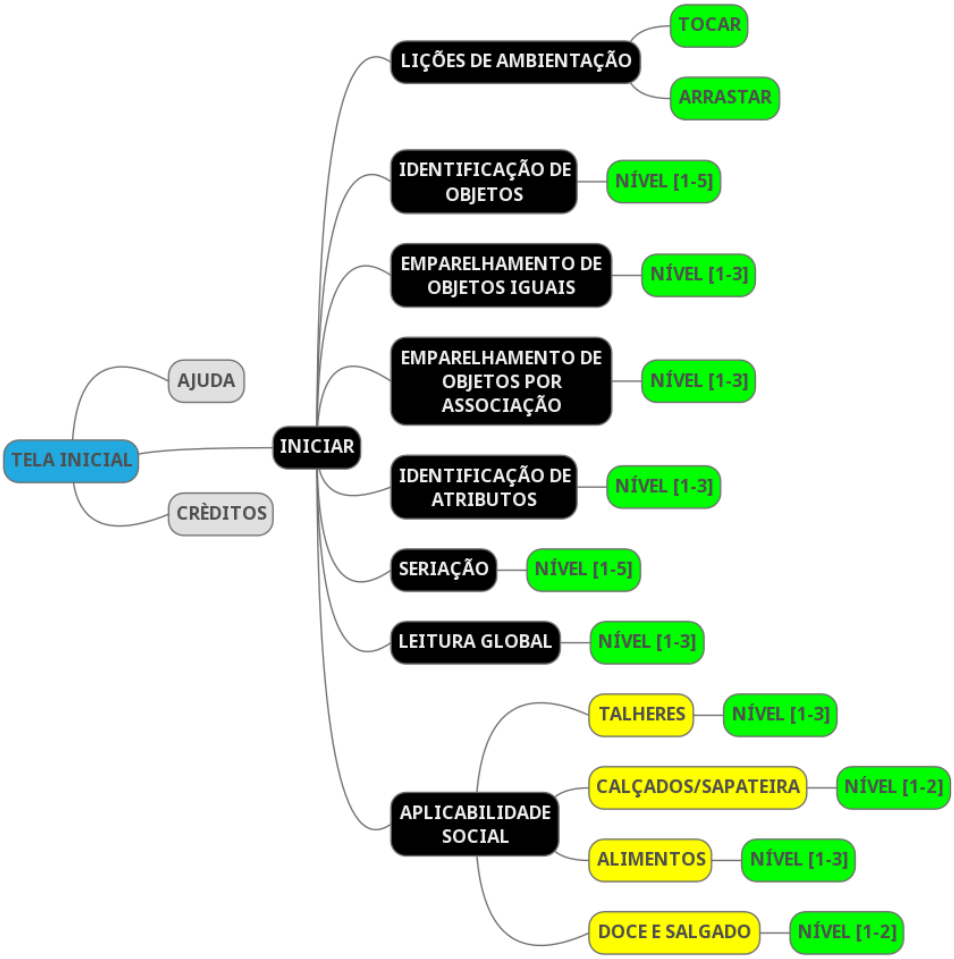


Figura 4.1: Software B decomposto em termos de suas funcionalidades.

As três características de qualidade analisadas durante o trabalho podem ser abordadas utilizando as informações contidas nas árvores. Em relação à documentação, seu processo de criação pode ser agilizado, pois o nome de todas as funcionalidades, suas relações e desdobramentos são facilmente visualizáveis. As falhas encontradas na execução do software também são mais fáceis de rastrear, pois a árvore mostra os caminhos e fluxos de execução possíveis do software e tornam o processo de isolamento de um defeito mais simples, assim a confiabilidade também é aumentada. Quanto à usabilidade, pode ser verificado, por exemplo, que uma funcionalidade tem uma localização inadequada, isso

é, o local onde a funcionalidade é apresentada, geralmente através de um botão em um menu, não condiz com seu grau de importância.

Outra abordagem interessante para a confiabilidade consiste em criar casos de testes baseados em casos reais de uso, isso pode ser feito em processos de desenvolvimento onde a equipe conta com a presença de pessoas às quais o software se destina, que no caso desse trabalho seriam os professores. Com seu auxílio, podem ser elaborados cenários de testes de confiabilidade realistas, baseados em casos de uso reais vivenciado pelos professores, ao invés de cenários pré-determinados pelos criadores do teste que não possuem o mesmo conhecimento dos reais utilizadores.

A seguir são dadas sugestões para os desenvolvedores de cada um dos softwares com base em seus resultados apresentados.

Tabela 4.6: Sugestões para os desenvolvedores dos softwares

Nome do Software	Sugestão
E2	O software apresentou baixa confiabilidade,o ciclo de atividades utilizado deve ser analisado em busca de defeitos.
F	O software apresentou o menor número de incidentes reportados para a inspeção de usabilidade, sua interface pode ser utilizada como modelo para manutenção e/ou desenvolvimento dos próximos softwares.
E	Disponibilização de algum tipo de documentação, pois o documento referido atualmente não está disponível.
D	Como esse software não apresentou falhas nos testes de confiabilidade, a abordagem para o seu desenvolvimento e manutenção podem ser feitas mais baseadas nos incidentes de usabilidade.
D2	O software apresentou baixa confiabilidade,o ciclo de atividades utilizado nos testes deve ser analisado em busca de defeitos.
A	Disponibilização de algum tipo de documentação, pois o documento referido atualmente não está disponível.
B	O software apresentou baixa confiabilidade,o ciclo de atividades utilizado deve ser analisado em busca de defeitos.
G	Como esse software não apresentou falhas nos testes de confiabilidade, a abordagem para o seu desenvolvimento e manutenção podem ser feitas mais baseadas nos incidentes de usabilidade.
C	O software apresentou baixa confiabilidade,o ciclo de atividades utilizado deve ser analisado em busca de defeitos.

Capítulo 5

Conclusão

Nesse trabalho foi feita uma avaliação de softwares educacionais utilizando normas técnicas da área de Engenharia de Software. Encontrar e mensurar a qualidade de um software é um processo longo e exige uma série de recursos, principalmente tempo. Verificou-se a grande importância em definir de forma clara e objetiva todo o processo, seus objetivos e passos necessários. Para esse objetivo a norma ISO/IEC 25040 revelou-se importante, assim como o modelo de qualidade e suas características definidos na norma ISO/IEC 25010 de onde foi derivado um modelo mais adequado aos objetivos deste trabalho.

Os métodos e técnicas de execução de testes formam uma área particularmente interessante, o processo de automatização de testes exigiu um grande esforço pra cobrir todas as funcionalidades dos softwares, mas também produz um resultado que permite várias abordagens diferentes que se integram e contribuem para a qualidade final dos softwares.

Conclui-se que uma avaliação conduzida de forma adequada ao contexto gera resultados significativos que são úteis para a mensuração da qualidade de um software. A partir desses dados os desenvolvedores podem formular planos de melhoria para os softwares, julgando as informações e sugestões aqui apresentadas e sua relevância.

Os resultados dos testes realizados nos softwares educacionais mostraram necessidades de melhorias neles em termos de documentação, usabilidade e confiabilidade. Foram testados produtos para computadores e também para tablets Android.

5.1 Trabalhos Futuros

Outras características não abordadas nos testes deste trabalho podem ser trabalhadas em futuros projetos, uma característica importante para softwares com alto nível de interatividade como estes aqui apresentados, é o desempenho. Particularmente, o tempo de resposta é um atributo importante para muitos usuários. Utilizando a ferramenta Sikulix e os scripts de automatização das telas, podem ser feitos testes desse tipo.

Referências

[1] ABNT. Conheça a ABNT. `http://www.abnt.org.br/abnt/conheca-a-abnt`. [Acesso em 07-01-2018]. 14

[2] IEEE Standards Association et al. Systems and software engineering—vocabulary iso/iec/ieee 24765: 2010. *Iso/Iec/Ieee*, 24765:1–418, 2010. 3

[3] James Bach. Exploratory testing explained. 36

[4] Pierre Bourque, Richard E Fairley, et al. *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press, 2014. 11, 16, 38

[5] Ana Regina Cavalcanti da Rocha, José Carlos Maldonado, Kival Chaves Weber, et al. Qualidade de software: teoria e prática. *São Paulo: Prenttice Hall*, 2001. 8, 29

[6] Anamaria de Moraes. Design e avaliação de interface: ergodesign e interação humano-computador. *Rio de Janeiro: iUsEr*, 2002. 19

[7] Marcio Delamaro, Mario Jino, and José Maldonado. *Introdução ao teste de software*. Elsevier Brasil, 2017. 8, 26, 27, 28, 29, 31, 33, 37

[8] Daniel Galin. *Software quality: concepts and practice*. Wiley-IEEE Computer Society Pr, 2018. 15

[9] Ana Cervigni Guerra and Regina Maria Thienne Colombo. *Tecnologia da Informação: qualidade de produto de software*. PBQP Software, 2009. 23

[10] Anne Mette Hass. *Guide to advanced software testing*. Artech House, 2014. 19, 37

[11] Raimund Hocke. Sikuli / SikuliX Documentation for version 1.1+. `http://sikulix-2014.readthedocs.io/en/latest/index.htm`. [Acesso em 07-01-2018]. 39

[12] Raimund Hocke. Sikuli Disclaimer. `http://sikulix.com/disclaimer/`. [Acesso em 07-01-2018]. 39

[13] IEC. What we do. `http://www.iec.ch/about/activities/`. [Acesso em 07-01-2018]. 13

[14] ISO IEC. ISO/IEC 25000:2014 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE - Overview. `https://www.iso.org/standard/64764.html`. [Acesso em 07-01-2018]. 23

[15] ISO IEC. 9126-1 (2001). software engineering product quality-part 1: Quality model. *International Organization for Standardization*, 2001. 5, 14, 18, 21

[16] ISO IEC. Systems and software engineering–systems and software quality requirements and evaluation (square)–evaluation process (nd). 2015. 23

[17] IEEE et al. Ieee std 1062–1998 ieee recommended practice for software acquisition, 1998. 10

[18] ISO. About us. `https://www.iso.org/about-us.htm`. [Acesso em 07-01-2018]. 13

[19] ISO. 9241-11. ergonomic requirements for office work with visual display terminals (vdts). *The international organization for standardization*, 45:9, 1998. 19

[20] Jim A McCall, Paul K Richards, and Gene F Walters. Factors in software quality. volume i. concepts and definitions of software quality. Technical report, General Electric Co Sunnyvale Ca, 1977. 11

[21] JD Musa, A Iannino, and K Okumoto. Engineering and managing software with reliability measures, 1987. 21

[22] Glenford J Myers, Corey Sandler, and Tom Badgett. *The art of software testing*. John Wiley & Sons, 2011. 27, 33, 34, 36

[23] Jakob Nielsen. Enhancing the explanatory power of usability heuristics. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 152–158. ACM, 1994. 20

[24] Shari Lawrence Pfleegerand and Joanne M Atlee. *Software engineering: theory and practice*. Pearson Education India, 1998. 4, 6, 11

[25] Jenny Preece, Yvonne Rogers, and Helen Sharp. *Design de interação*. bookman, 2005. 20

[26] Roger S. Pressman. *Engenharia de Software: Uma Abordagem Profissional*. 2011. 1, 4, 6, 9, 12, 13, 19, 21, 22, 27, 30

[27] Jeffrey Rubin and Dana Chisnell. *Handbook of usability testing: how to plan, design and conduct effective tests*. John Wiley & Sons, 2008. 19

[28] Ian Sommerville, Reginaldo Arakaki, and Selma Shin Shimizu Melnikoff. *Engenharia de software*. Pearson Prentice Hall, 2008. 6, 7, 21, 27

[29] Stefan Wagner. *Software product quality control*. Springer, 2013. 18, 24

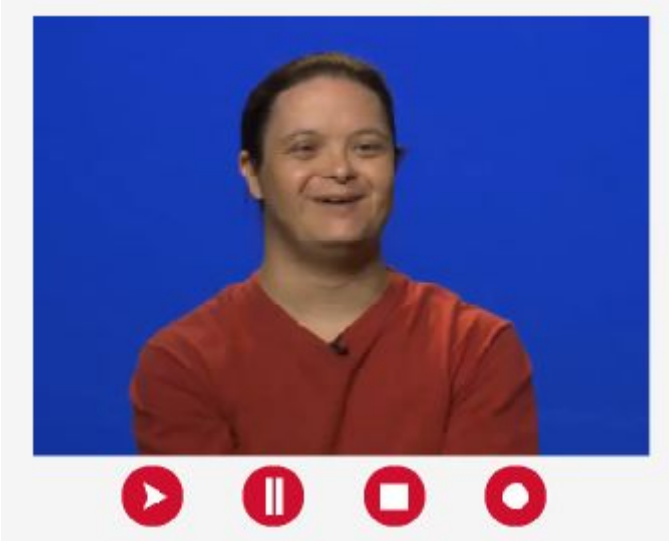
Apêndice A

Apêndice A - Resultados detalhados dos testes

Relatório de usabilidade - Software E2

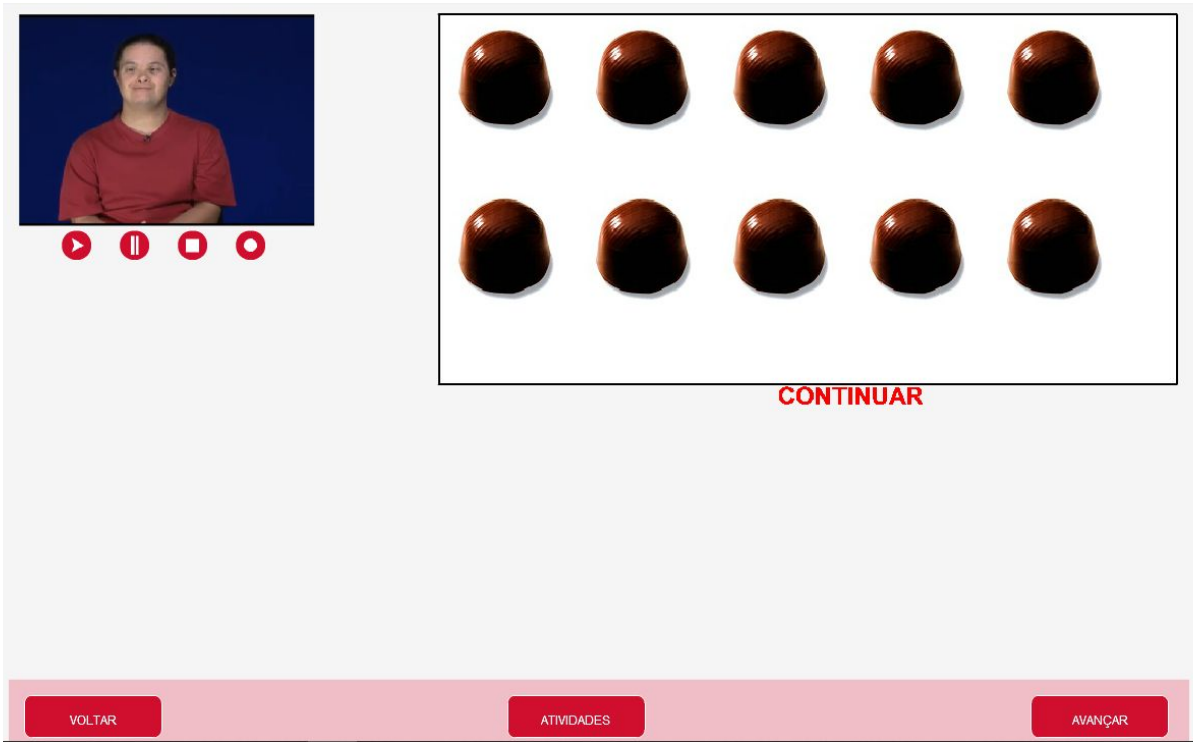
Id	Heurística nº	Descrição	Imagem	Atividade
#1	2	Os botões do player de vídeo não tem o comportamento esperado e compatível com experiência prévia dos usuários.	1	Todas que possuem vídeos
#2	9	Em muitas atividades, a primeira tela possui o botão "**Voltar",** exercendo a mesma função do botão **"Atividades"**.	2	Várias
#3	9	Em muitas atividades, a última tela possui o botão "**Avançar",** exercendo a mesma função do botão **"Atividades"**.	3	Várias
#4	10	Não é possível escolher apenas uma atividade por turno, pois existem 5 atividades que obrigatoriamente precisam ser escolhidas, e 3 turnos diferentes. Assim, obrigatoriamente haverá turno com mais de uma atividade.	4	Horas -> **Turno**
#5	1	O software não dá feedback do seu estado atual quando o usuário preenche o formulário equivocadamente e clica no botão "**Avançar**".	5	Horas -> **Turno**
#6	6	O software não previne o erro no preenchimento do formulário.	6	Horas -> Turno -> **Configuração**
#7	9	O software não permite voltar à tela de configuração após o início da subatividade **Ajuste o relógio.** O botão **"Voltar"** funciona para voltar à atividade anterior a partir da segunda tela, mas na primeira tela o botão não volta à tela anterior.	7	Horas -> Turno -> Configuração -> **Ajuste o relógio**
#8	4	Em algumas atividades com números, a última subatividade consiste em arrastar os objetos no quadro. Um novo botão de prosseguimento "**Continuar**" é utilizado, quebrando o protocolo e padronização do botão "**Avançar**".	8	Números -> **Numerais, Dezena,Dúzia,**
#9	2	No quadro da subatividade, onde o usuário pode arrastar os objetos, é possível sobrepor esses objetos, o que pode prejudicar a assimilação da lição.	9	Números -> **Numerais, Dezena,Dúzia,**
#10	9	Nas subatividades, **escolha uma (moeda, cédula)**. O botão **"Avançar"** não possui função.	10	Dinheiro-> **Cédula,Moeda**

#11	2	Nas subatividades, **escolha uma (moeda, cédula)**. Ao ser feita a escolha, o texto que descreve o valor por extenso e numericamente é difícil de ver devido ao contraste entre texto e fundo.	11	Dinheiro-> **Cédula,Moeda**
#12	4	A atividade **Calculadora** encontra-se dentro das atividades **Dinheiro,** quebrando a consistência de hierarquia na representação das funcionalidades.	12	Dinheiro-> **Calculadora**
#13	6	Nas atividades que envolvem arrastar o troco para a carteira, quando existem muitos objetos no troco (cédulas e moedas), é fácil cometer erros, pois o software, muitas vezes, não reconhece que o troco foi guardado e impossibilita o avanço à próxima tela	13	Dinheiro -> **Mercado, Padaria,Farmácia**
#14	6	Nas atividades que envolvem arrastar o troco para a carteira, quando existem muitos objetos no troco (cédulas e moedas), o software não permite que o usuário recupere-se do erro que é introduzido por **#13**	13	Dinheiro -> **Mercado, Padaria,Farmácia**
#15	2	Ao calcular o troco subtraindo o valor da cédula e o valor da compra, o software aceita como corretos, tanto a subtração, quanto a adição dos valores.	14	Calculadora -> **Troco**
#16	6	Nas sub-atividades em que se pode usar a calculadora livremente, o software permite que sejam inseridos mais do que um símbolo de pontuação (.). Ao efetuar operação com números assim, a calculadora mostra na tela **NaN** (not a number).	15	Calculadora-> **Calculadora, Troco, Dinheiro, Produto**
#17	4	O software permite utilizar a calculadora livremente em todas as telas que possuem soma e subtração de produtos, enquanto que em outras ocasiões só é permitido pressionar os botões corretos	16	Calculadora-> **Troco, Dinheiro, Produto**
#18	4	Ao clicar no botão **"Voltar"**, dentro das atividades de **Calculadora,** o software direciona para as atividades de **Dinheiro,** essa quebra de consistência é consequência direta de #12	17	Calculadora-> **Calculadora**



Software E2: Imagem 1

Software E2: Imagem 2



Software E2: Imagem 3

Software E2: Imagem 4

Software E2: Imagem 5

Software E2: Imagem 6

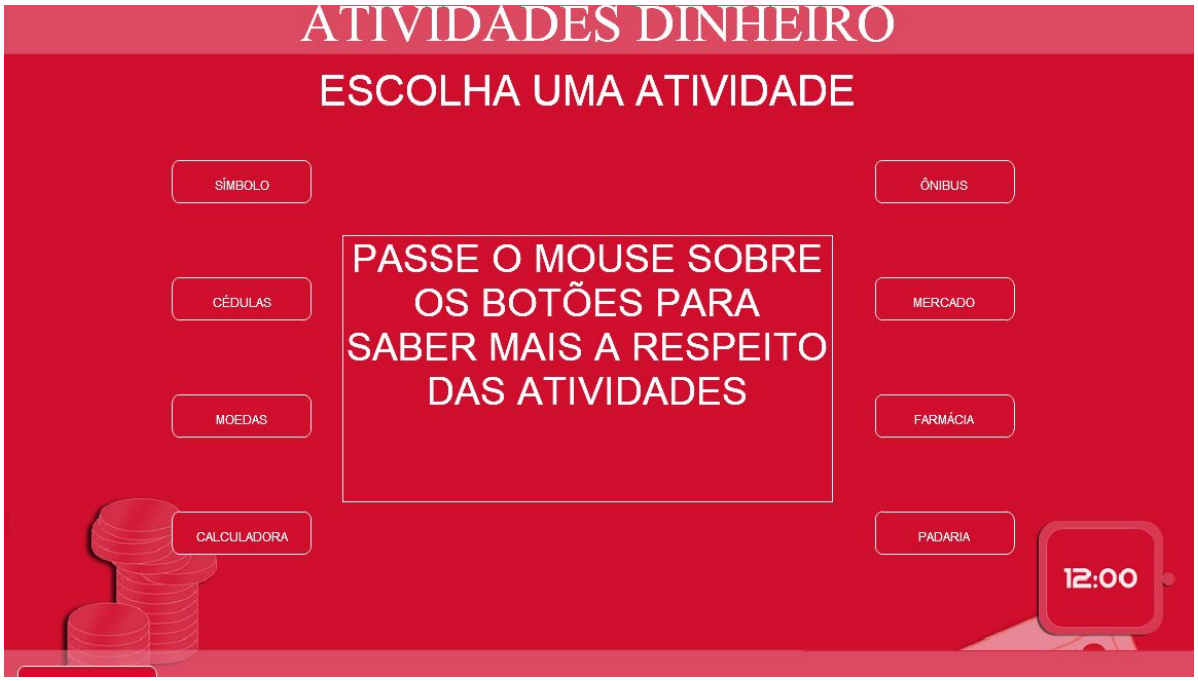
Software E2: Imagem 7

Software E2: Imagem 8

Software E2: Imagem 9

Software E2: Imagem 10

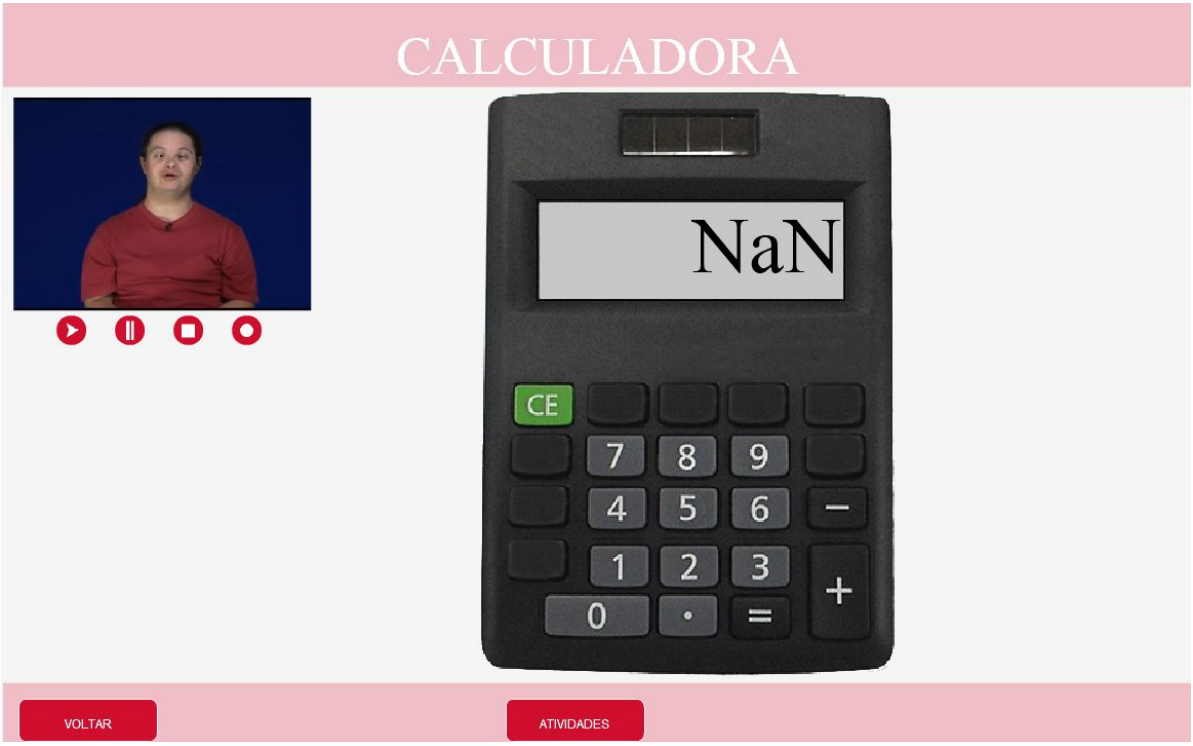
Software E2: Imagem 11



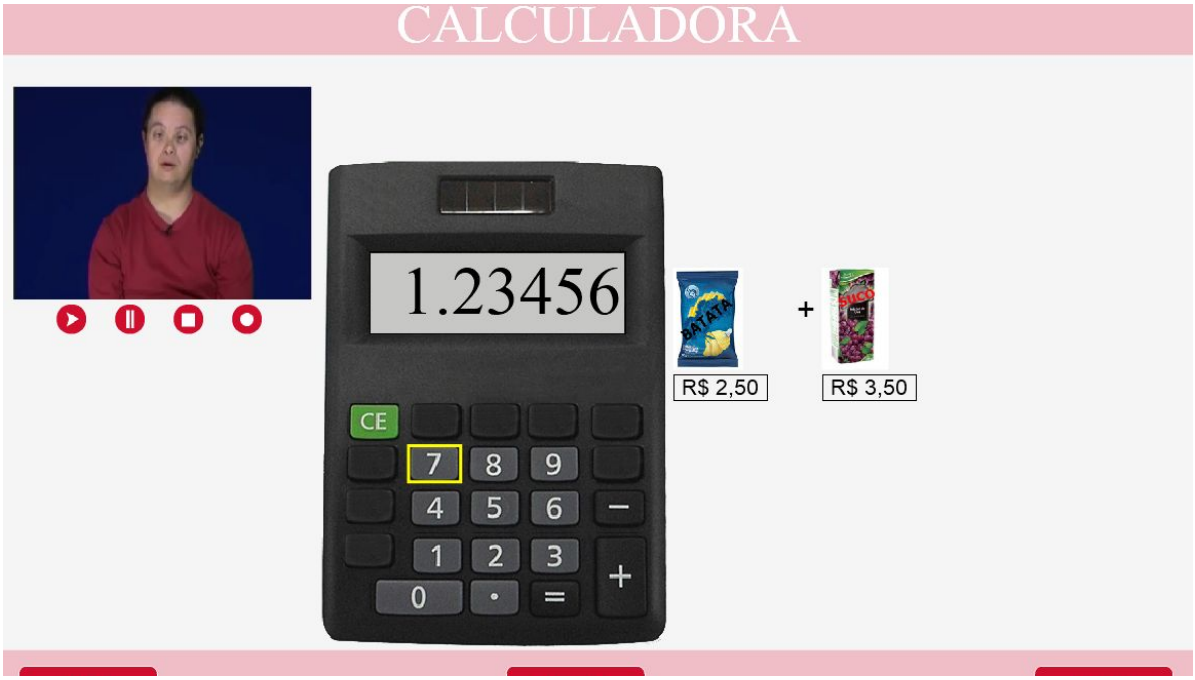
Software E2: Imagem 12

Software E2: Imagem 13

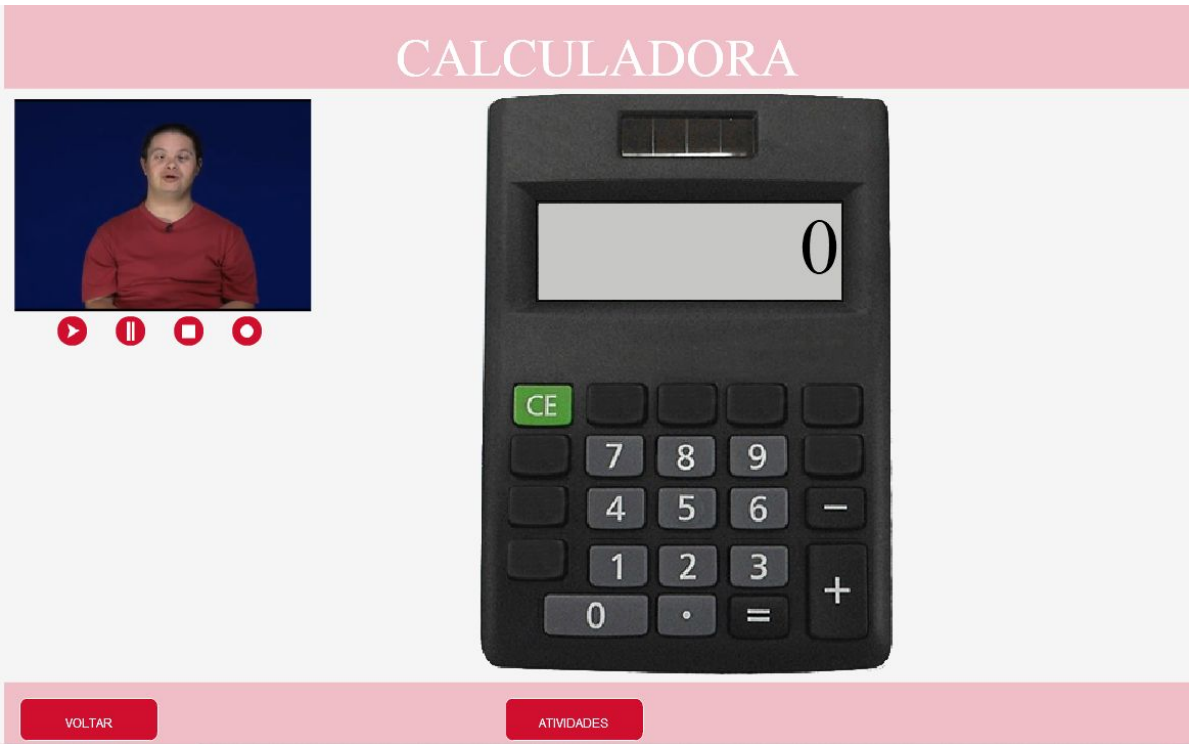
Software E2: Imagem 14



Software E2: Imagem 15



Software E2: Imagem 16



Software E2: Imagem 17

Relatório de usabilidade - Software F

Id	Heurística nº	Descrição	Imagem	Atividade
#1	2	Os botões do player dos vídeos não tem o comportamento esperado e compatível com a experiência prévia dos usuários.	1	Todos os vídeos do software
#2	9	Em muitas atividades, a primeira tela possui o botão "**Voltar" (**e/ou o botão "**Avançar"),** exercendo a mesma função do botão **"Início"**.	2	Todas as atividades
#3	9	O botão "**refazer**" aparece mesmo quando o usuário executa a atividade pela primeira vez.	3	Todas as atividades
#4	4	A atividade **fio dental** não segue os padrões das demais atividades, sua primeira tela possui algum defeito que impede sua exibição correta.	4	**Atividade fio dental**

Software F: Imagem 1

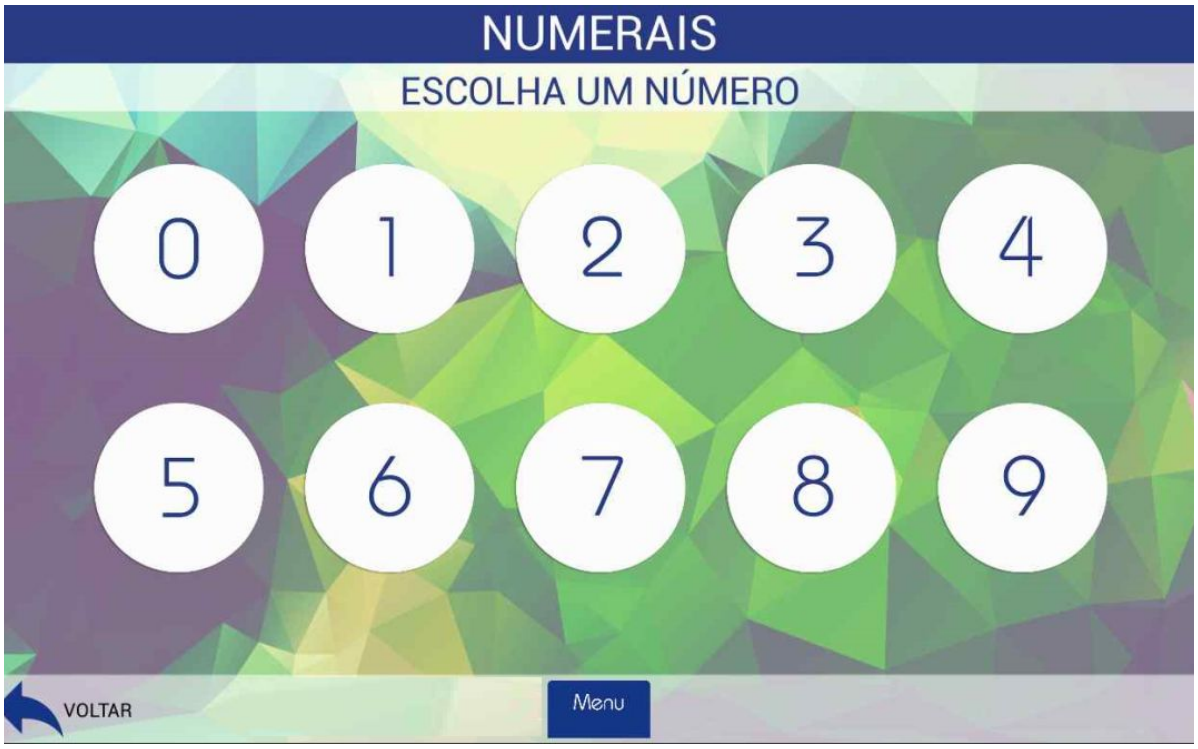
Software F: Imagem 2

Software F: Imagem 3

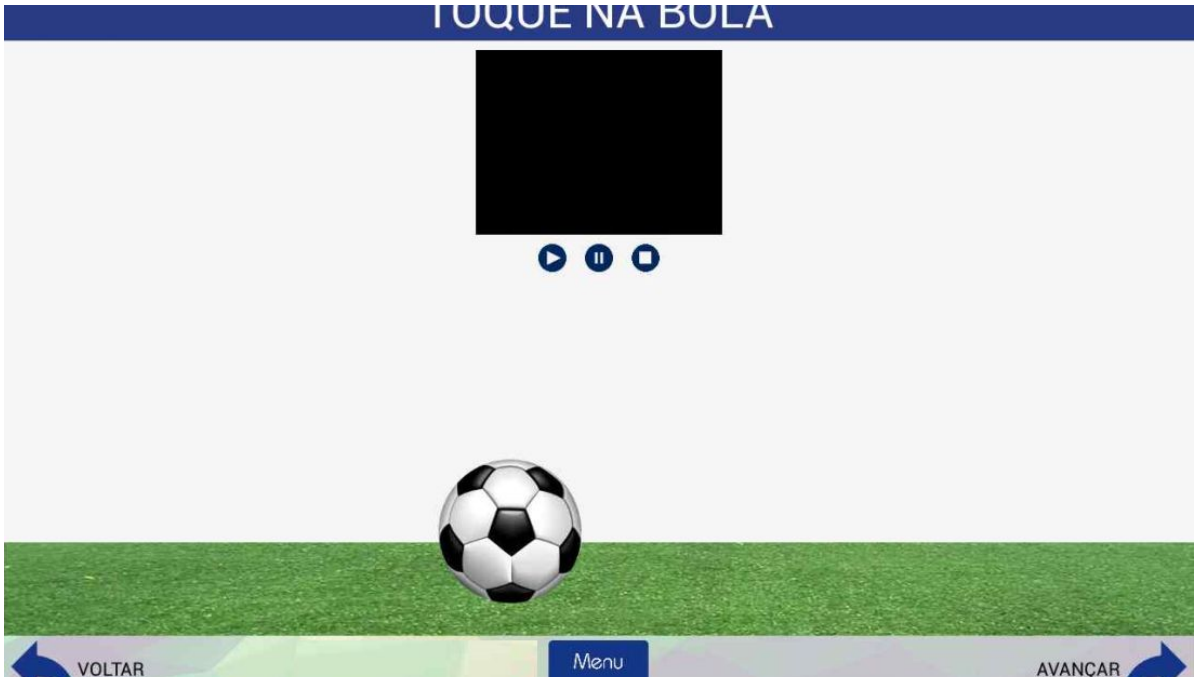
Software F: Imagem 4

Relatório de usabilidade - Software E

Id	Heurística nº	Descrição	Imagem	Atividade
#1	9	Em muitas atividades, a primeira tela possui o botão "**Voltar",** ou a última tela possui o botão **"Avançar",** exercendo a mesma função do botão **"Menu"**.	1	Várias atividades
#2	9	Essas atividades não possuem vídeos, mas os players estão posicionados nas tela, ocupando espaço desnecessário.	2	**Tocar** **Arrastar**
#3	10	Não é possível escolher apenas uma atividade por turno, pois existem 5 atividades que obrigatoriamente precisam ser escolhidas, e 3 turnos diferentes. Assim, obrigatoriamente haverá turno com mais de uma atividade.	2	Horas ->**Turno**
#4	4	A descrição da instrução ("escrever nome cédula") da tarefa no topo da tela está inconsistente com as instruções de outras telas, em relação à sua legibilidade.	3	Cédulas,Moedas -> **Escrever nome e valor numeral**
#5	10	A descrição da instrução da tarefa no topo da tela para moedas está descrita como "Escrever valor cédula".	4	Moedas -> **Escrever valor numeral**
#6	4	O botão "**Avançar**" reinicia a atividade (a partir de sua primeira tela), ao invés de finalizar e voltar ao **Menu**.	5	**Tocar** **Arrastar** **Cédulas** **Moedas**
#7	6	O software não previne o erro no preenchimento do formulário .	6	Horas -> Turno -> **Configuração**
#8	5	A mensagem de erro na marcação não sugere uma solução.	7	Horas -> Turno -> **Configuração**

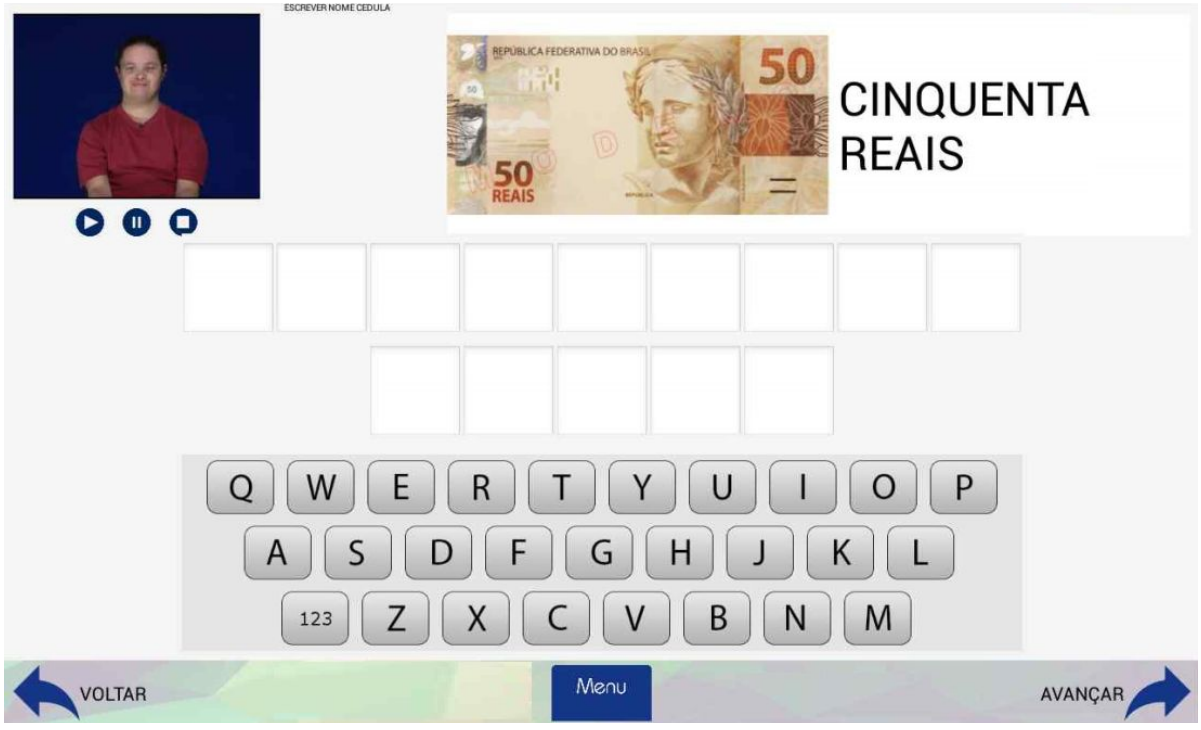


Software E: Imagem 1

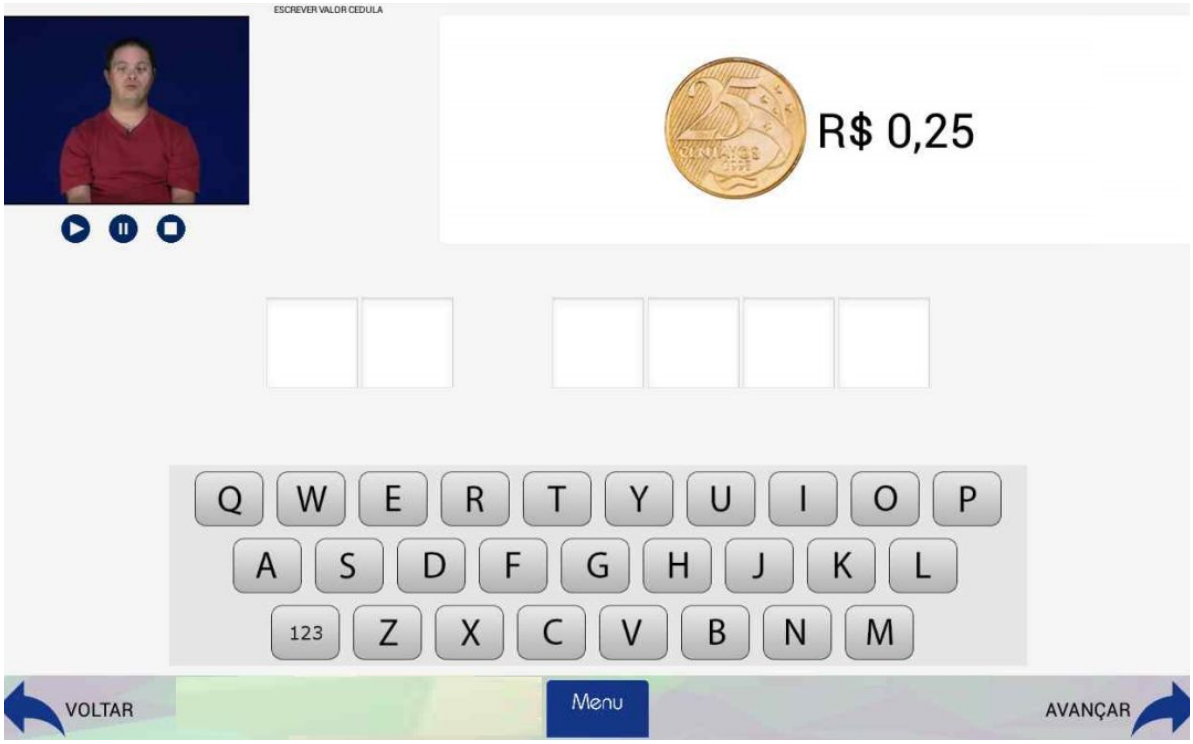


Software E: Imagem 2

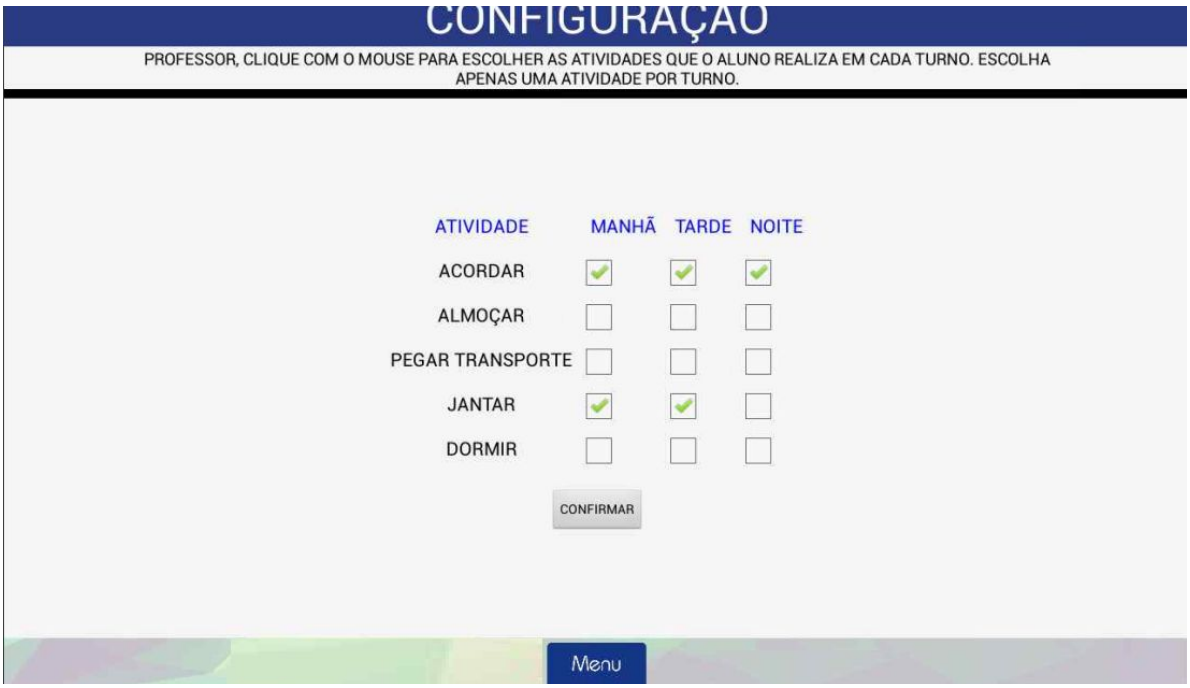
Software E: Imagem 3



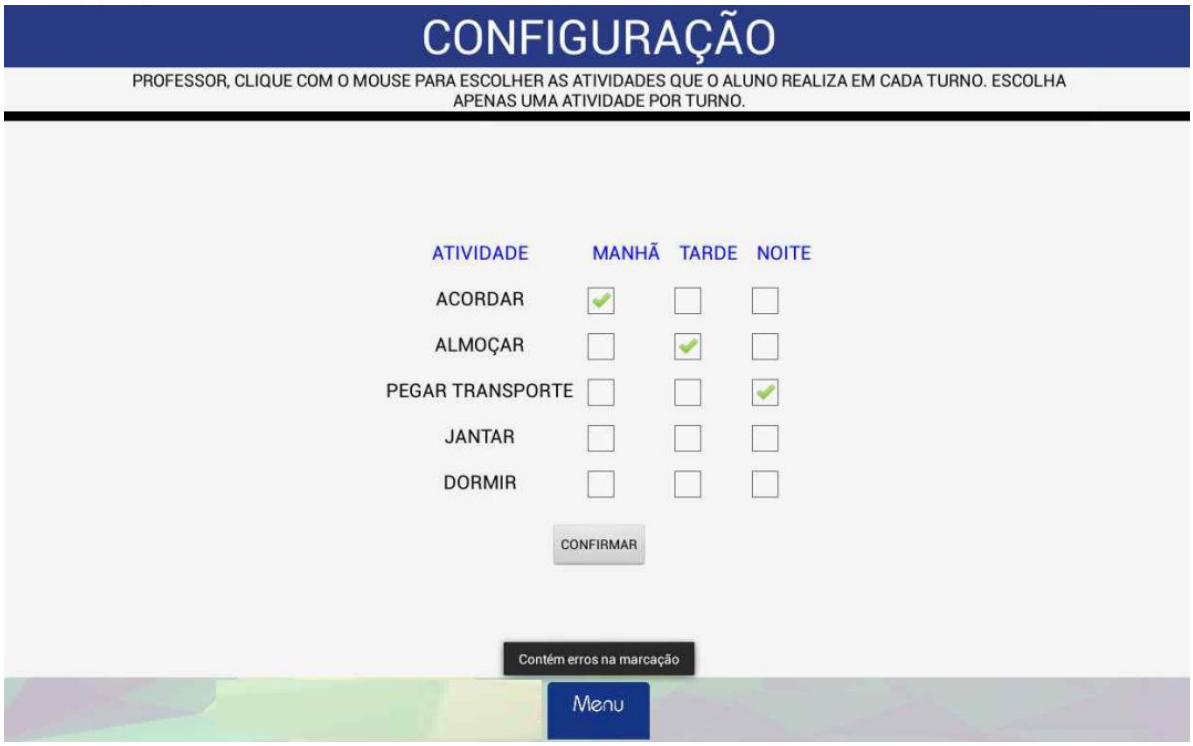
Software E: Imagem 4



Software E: Imagem 5



Software E: Imagem 6



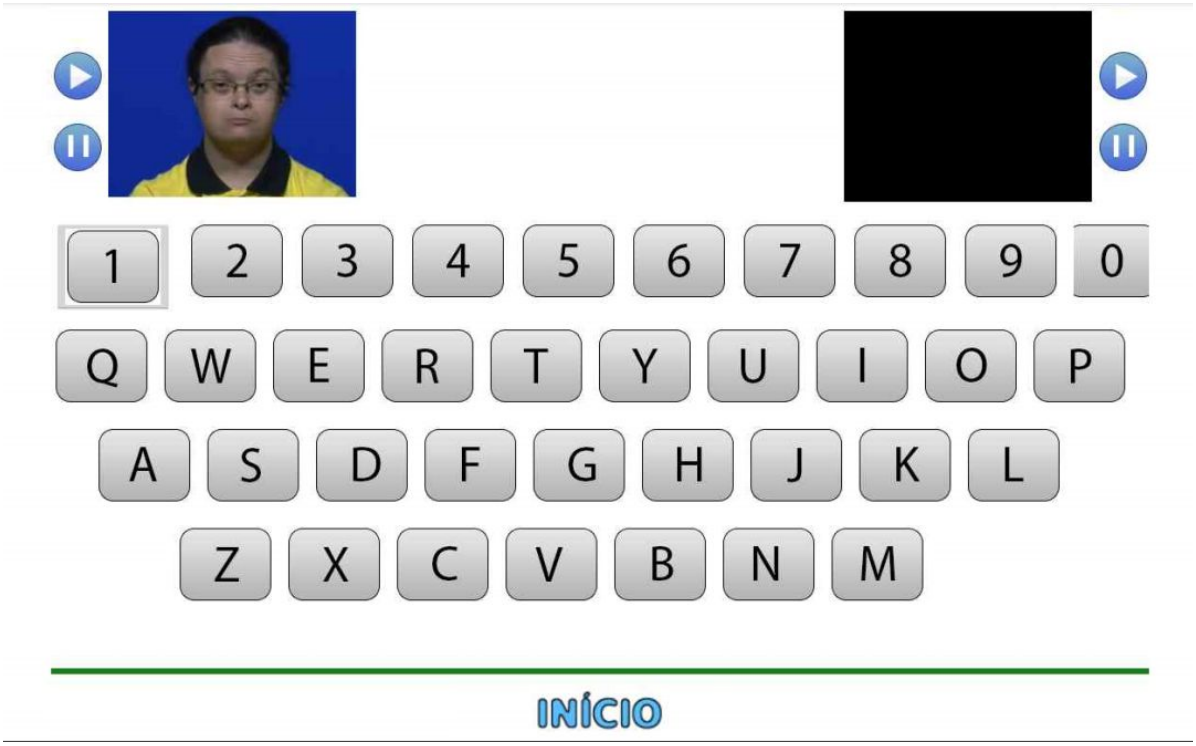
Software E: Imagem 7

Relatório de usabilidade - Software D

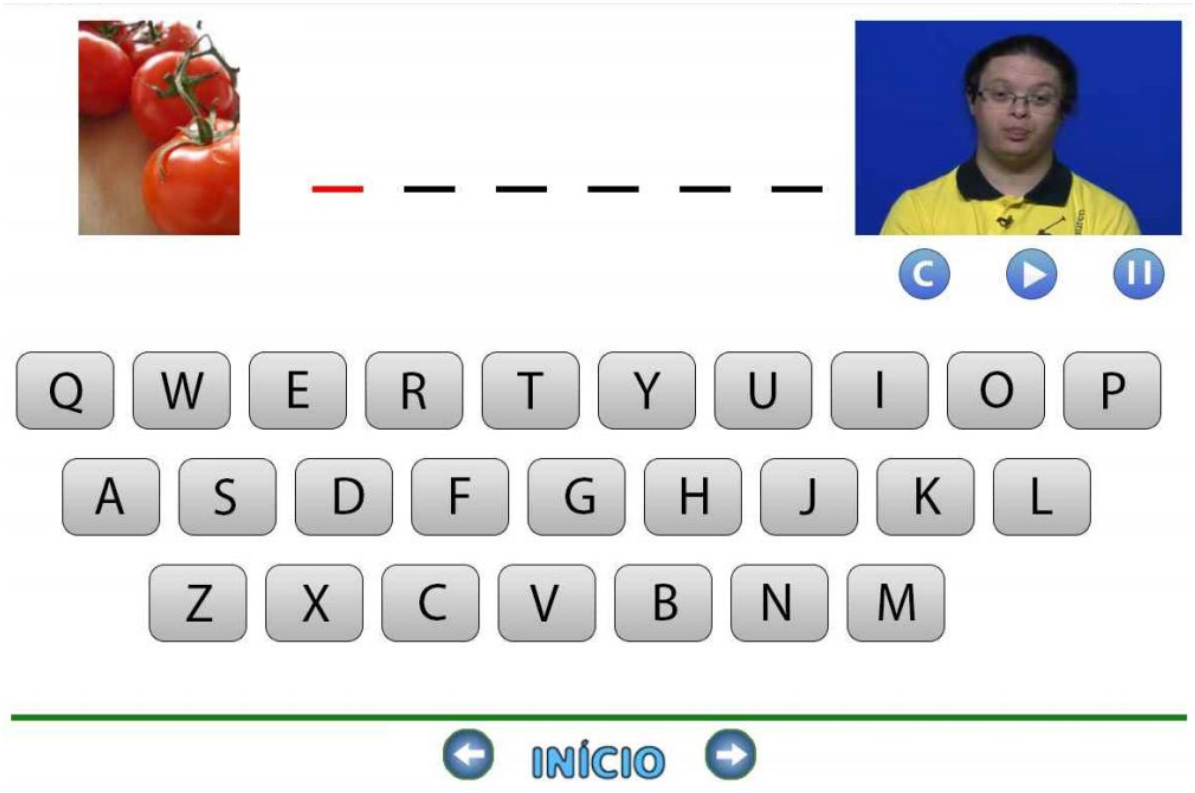
Id	Heurística nº	Descrição	Imagem	Atividade
#1	2	Na tela inicial não fica evidente que a caixa com o texto apresentado na parte esquerda deve ser rolada para baixo para que o texto seja lido completamente.	1	**Tela Inicial**
#2	2	Na tela inicial, a palavra "**deficiência**" está grafada incorretamente.	2	**Tela Inicial**
#3	2	Os botões do player dos vídeos não tem o comportamento esperado e compatível com a experiência prévia dos usuários.	3	Todos os vídeos de atividades
#4	9	Em muitas atividades, a primeira tela possui o botão "**Voltar" (**e/ou o botão "**Avançar"),** exercendo a mesma função do botão **"Início"**.	4	Várias atividades
#5	2	Nas atividades de exercícios, um botão de controle de vídeo é introduzido (letra C), sem que seu significado seja conhecido, ou explicado.	5	Todos os vídeos de exercícios
#6	2	A palavra "variação" está grafada incorretamente.	6	**Configuração dos Exercícios**
#7	10	O enunciado diz "Escolha quais opções estarão visíveis para o aluno", mas somente é possível visualizar uma opção por vez.	6	**Configuração dos Exercícios**
#8	4	O quadro verde ao centro serve para visualizar mais informações sobre uma opção, mas ele somente fornece informações sobre as variações (dificuldades).	6	**Configuração dos Exercícios**
#9	4	A atividade **Bate-Papo** encontra-se dentro das atividades **Exercícios,** quebrando a consistência de hierarquia na apresentação das funcionalidades.	6	**Configuração dos Exercícios**
#10	6	Não há feedback do sistema no encerramento da conversa (esgotamento das questões).	7	Exercícios-> **Bate-Papo**
#11	2	Na **atividade da letra B**, a segunda palavra do jogo é ilustrada pela figura de uma BOTA, mas a grafia da estrutura da palavra no jogo é BOCA.	8	Escolha uma lição -> **B**

Software D: Imagem 1

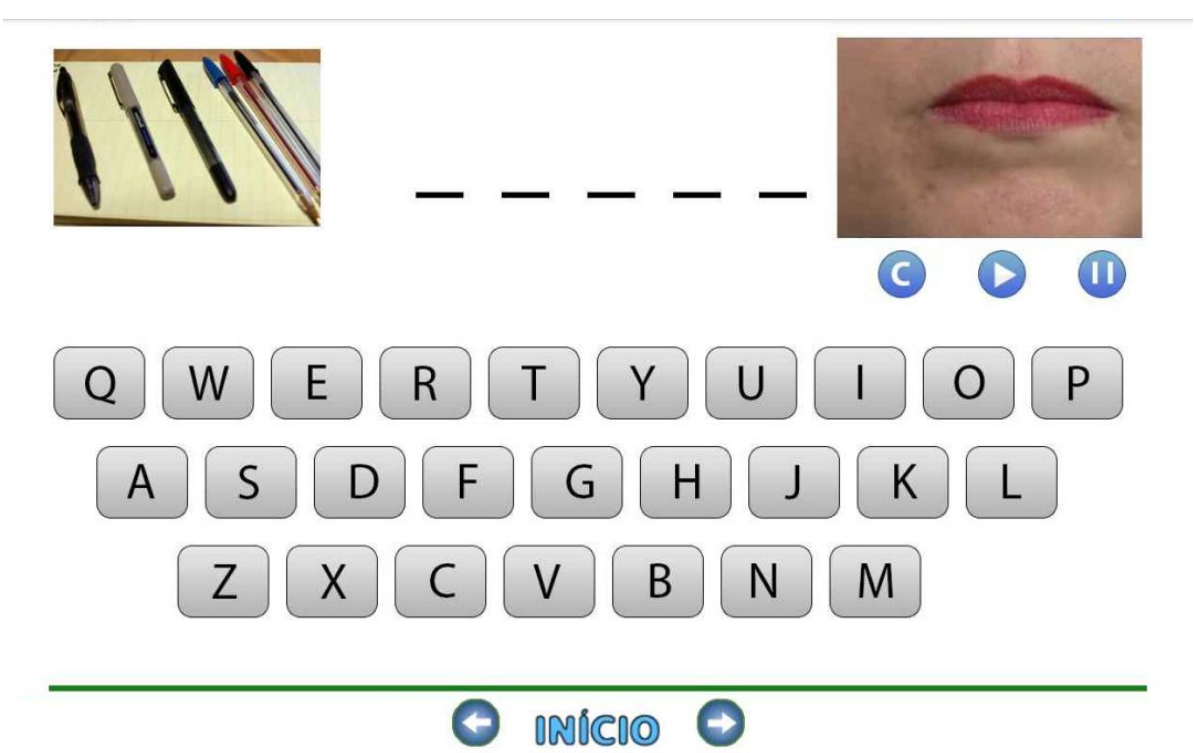
Software D: Imagem 2



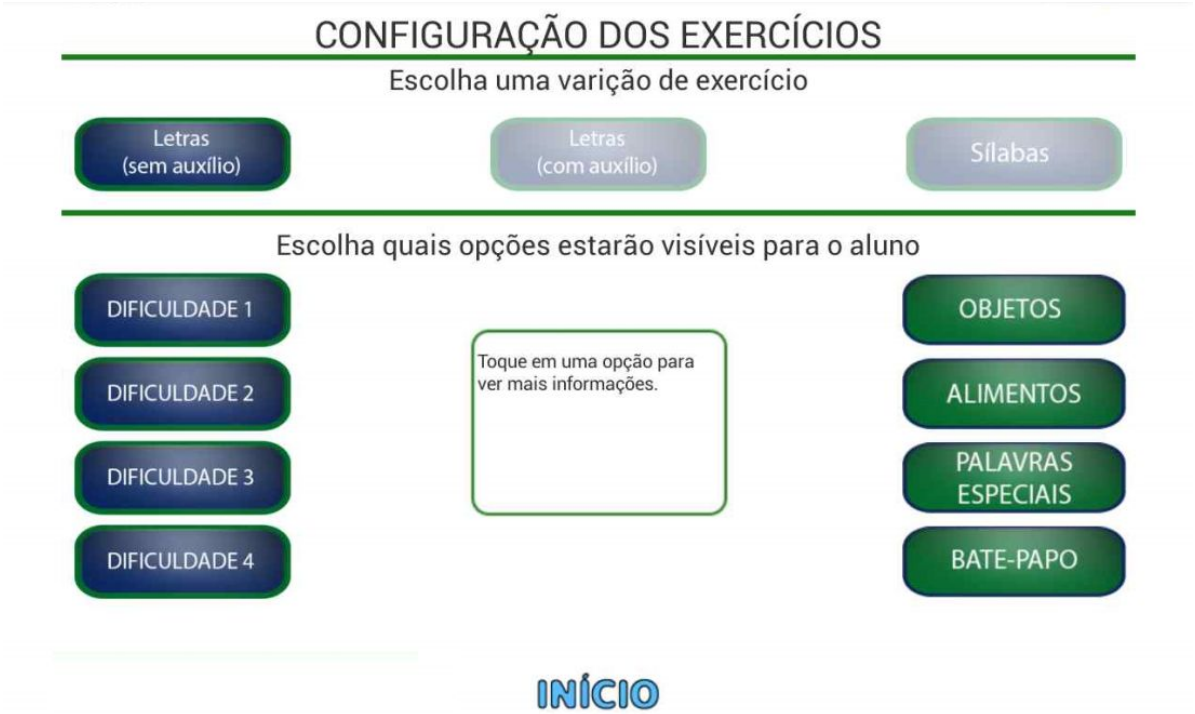
Software D: Imagem 3



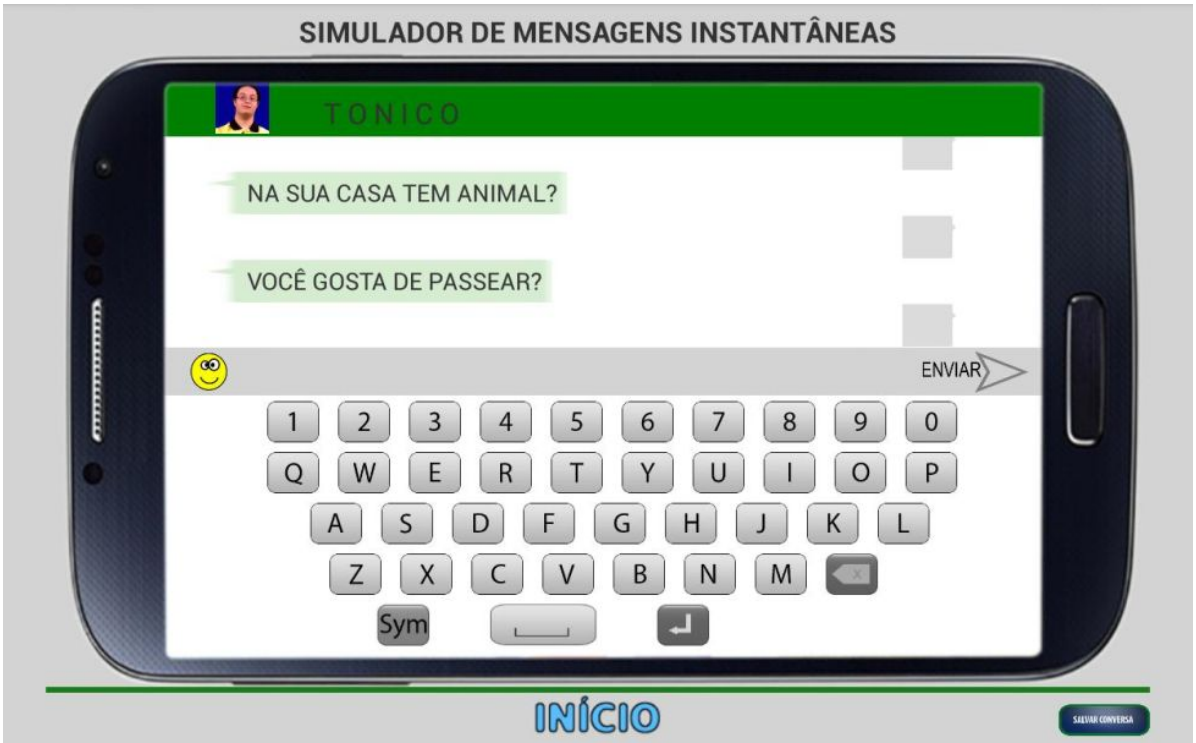
Software D: Imagem 4



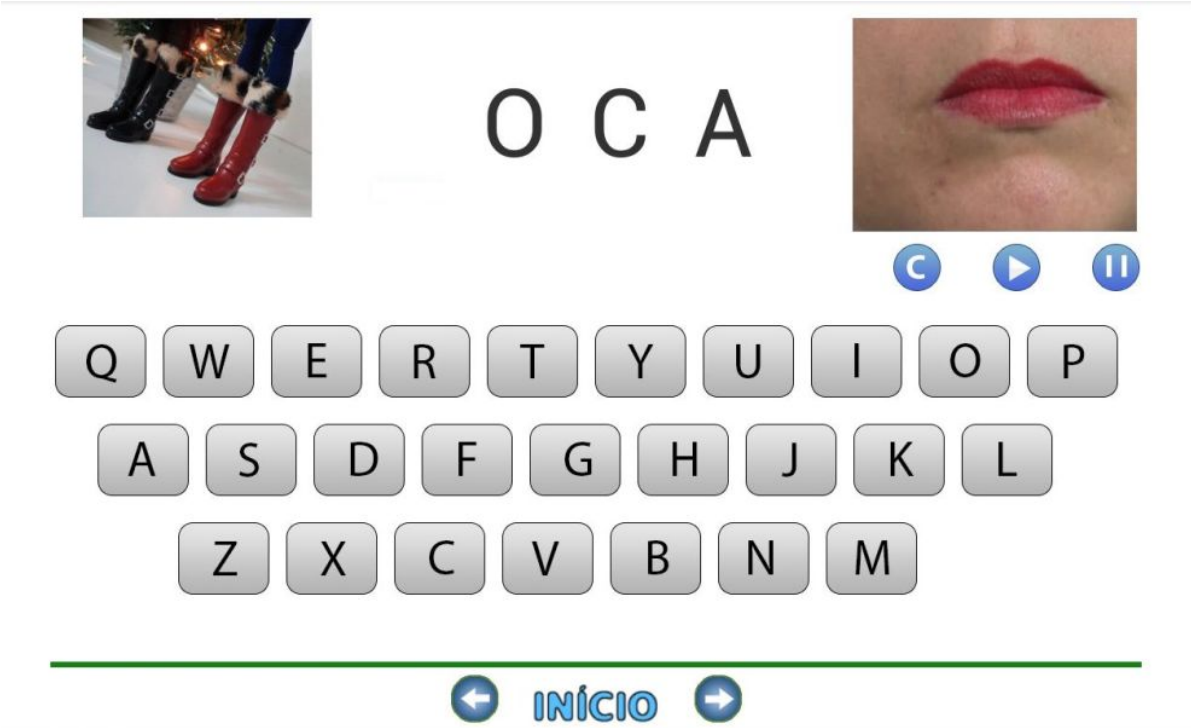
Software D: Imagem 5



Software D: Imagem 6



Software D: Imagem 7

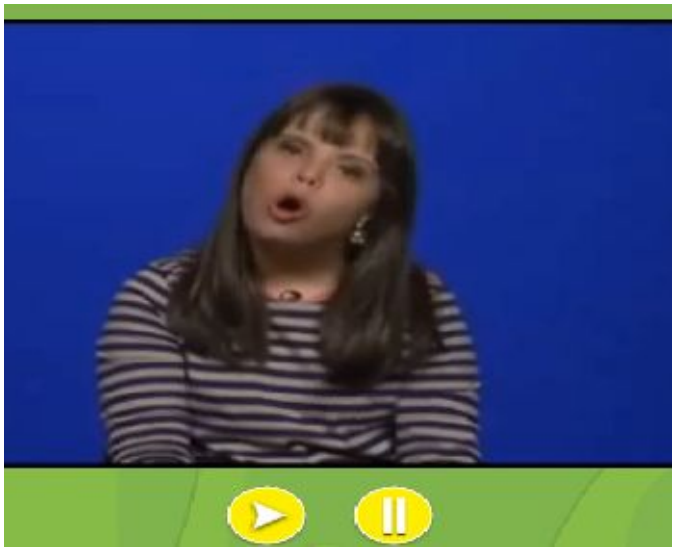


Software D: Imagem 8

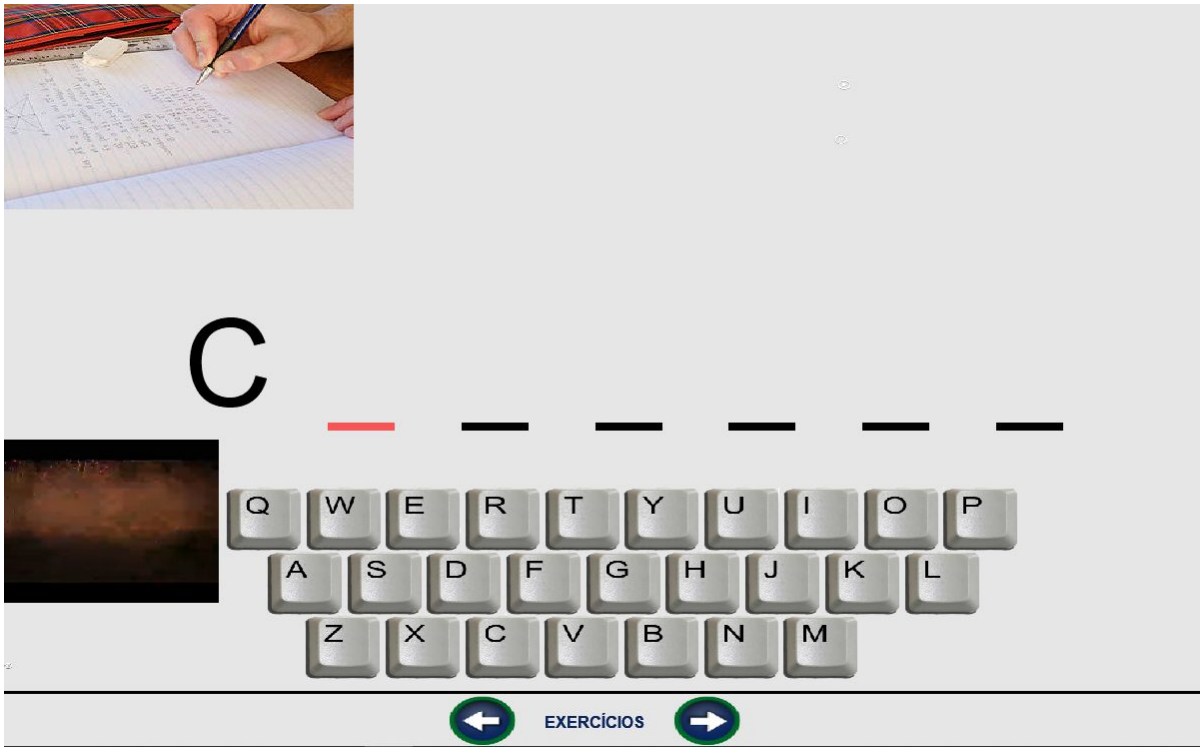
Relatório de usabilidade - Software D2

Id	Heurística nº	Descrição	Imagem	Atividade
#1	2	Os botões do player dos vídeos não tem o comportamento esperado e compatível com experiência prévia dos usuários.	1	Todos os vídeos do software
#2	7	Os botões do player dos vídeos tem uma visibilidade muito ruim.	2	Todos os vídeos de atividades
#3	9	Em muitas atividades, a primeira tela possui o botão "**Voltar" (**e/ou o botão "**Avançar"),** exercendo a mesma função do botão **"Início"**.	3	Várias atividades
#4	4	A disposição/layout de exibição dos vídeos é inconsistente dentro do software em relação às margens da tela e em relação ao posicionamento dos botões.	4 5	Todos os vídeos de atividades
#5	4	A atividade **Bate-Papo** encontra-se dentro das atividades **Exercícios,** quebrando a consistência da hierarquia na apresentação das funcionalidades.	6	Exercícios-> **Bate-Papo**
#6	6	O botão **"letras maiúsculas / letras minúsculas"** ao ser pressionado, apaga toda a conversa já realizada pelo usuário, esse é um efeito colateral não desejado.	7	Exercícios-> **Bate-Papo**
#7	6	Na tela **Configuração dos exercícios**, o erro causado por não escolher uma categoria de exercício não é prevenido pelo software.	8	Exercícios-> **Configuração dos exercícios**
#8	6	Na tela **Configuração dos exercícios**, deve ser feita uma combinação entre uma opção do primeiro grupo de botões, e uma ou mais opções do segundo grupo. Ambos os grupos são definidos na interface como "variação de exercício", na tela de erro gerada a partir do incidente #7, a mensagem de erro define o segundo grupo como "categoria de exercício".	9	Exercícios-> **Configuração dos exercícios**
#9	4	Na dificuldade **sem auxílio**, o quadro do vídeo desaparece após o acerto da primeira letra, porém os controles de vídeo ainda estão presentes na tela e são funcionais.	10	Exercícios-> **Exercício sem auxílio**
#10	4	Na dificuldade **com auxílio**, o quadro do vídeo desaparece após alguns segundos de inatividade.	10	Exercícios-> **Exercício com auxílio**
#11	4	Na dificuldade **com auxílio**, o vídeo de auxílio tocado é sempre o mesmo independente do progresso do usuário na palavra.	10	Exercícios-> **Exercício com auxílio**

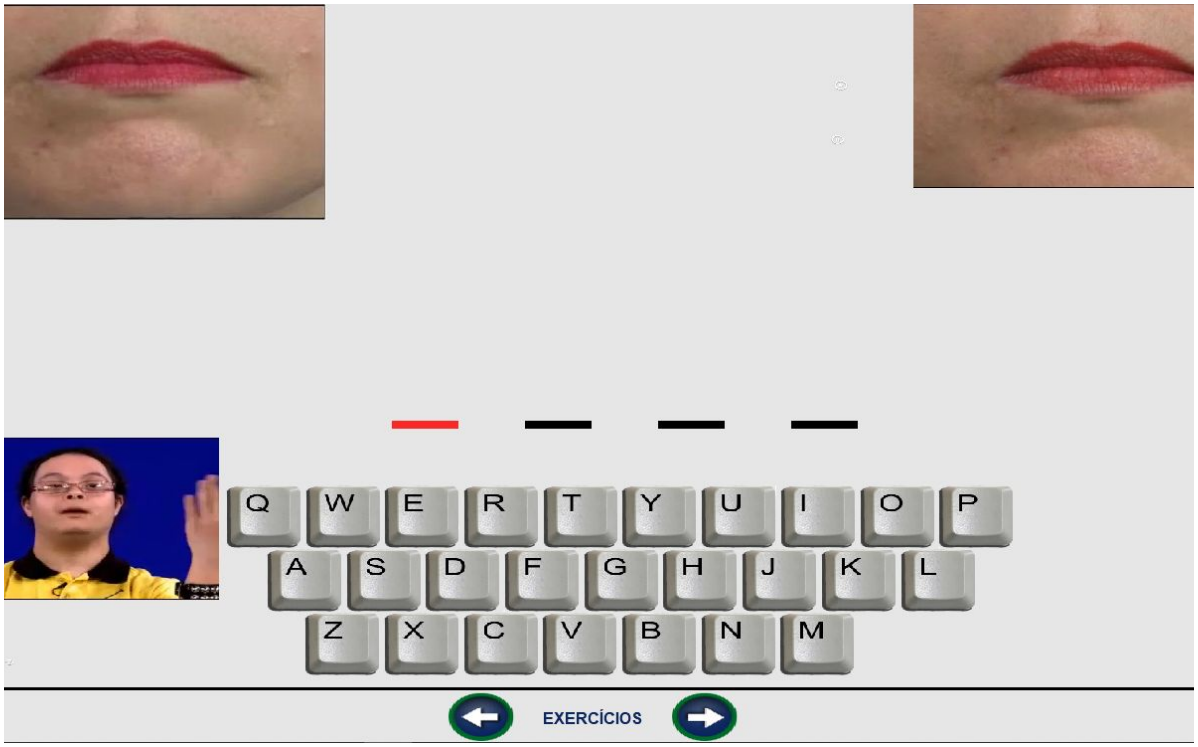
#12	8	A tela “**exercícios”**, que sucede a configuração de exercícios é sempre exibida, ainda que o usuário queira realizar a lição com apenas um tipo de exercício, essa tela impede que o usuário atinja seu objetivo mais rapidamente.	11	Exercícios> **Exercício com auxílio**
#13	4	Ao entrar em **tela cheia** e depois reverter essa operação, o layout do software torna-se inconsistente.	12	Todas as atividades com exercícios



Software D2: Imagem 1



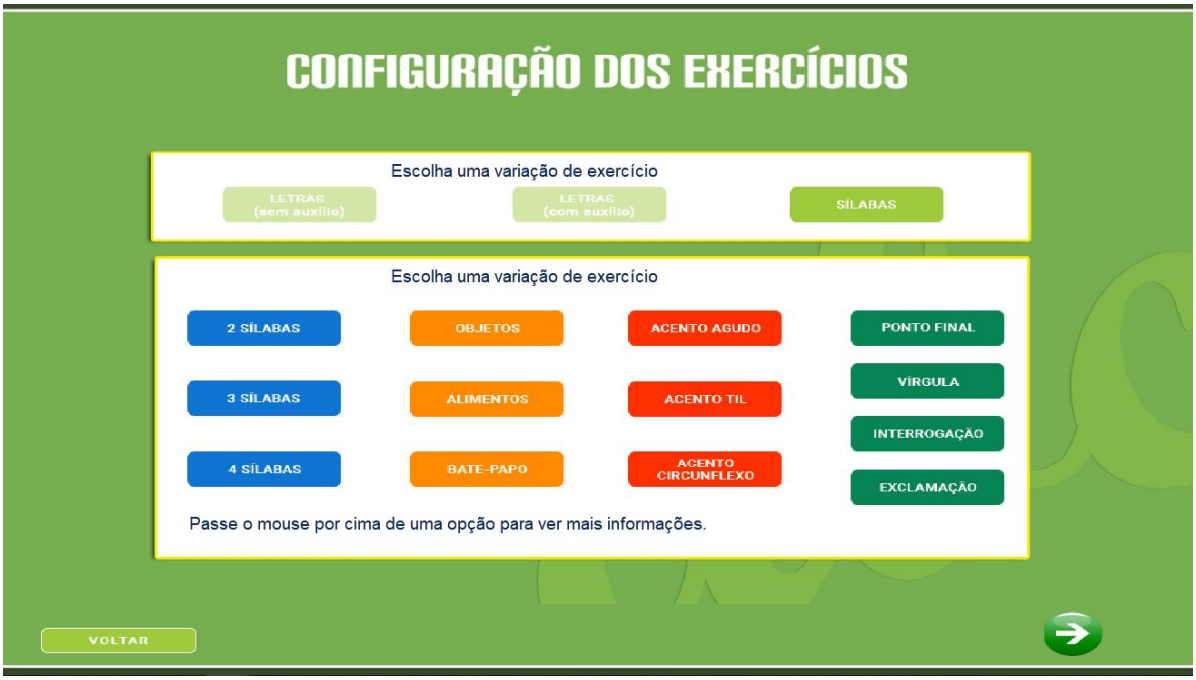
Software D2: Imagem 2



Software D2: Imagem 3

Software D2: Imagem 4

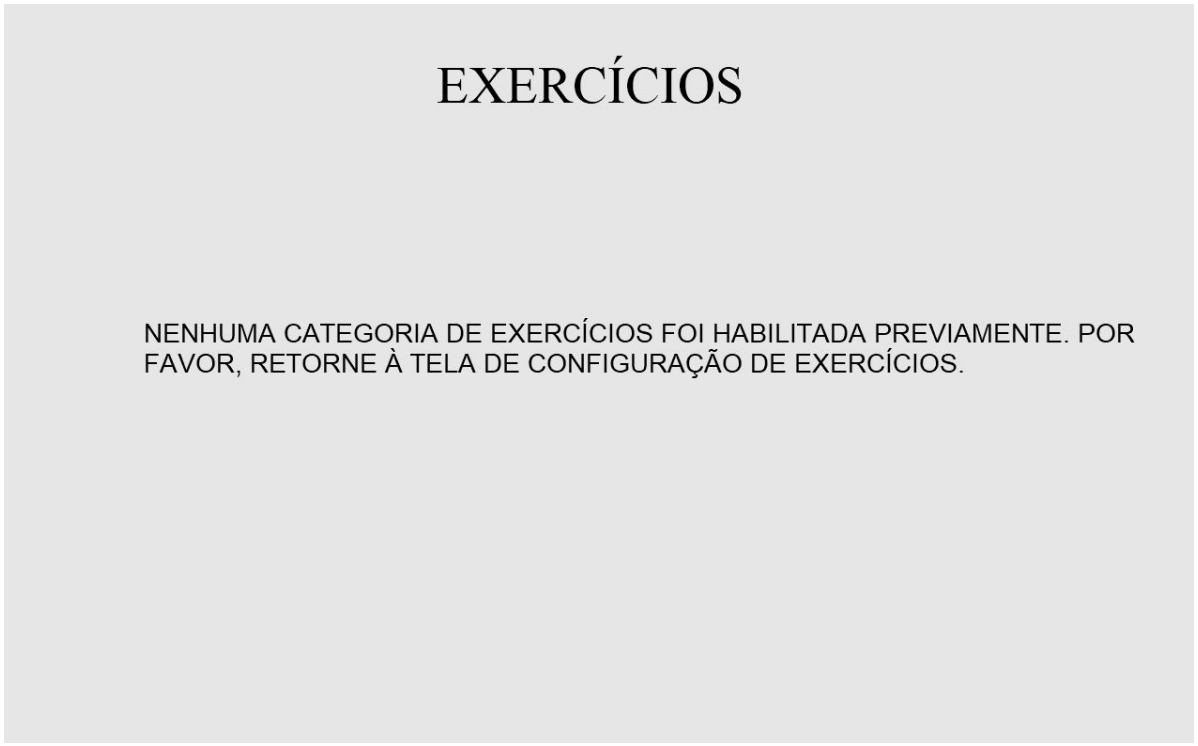
Software D2: Imagem 5



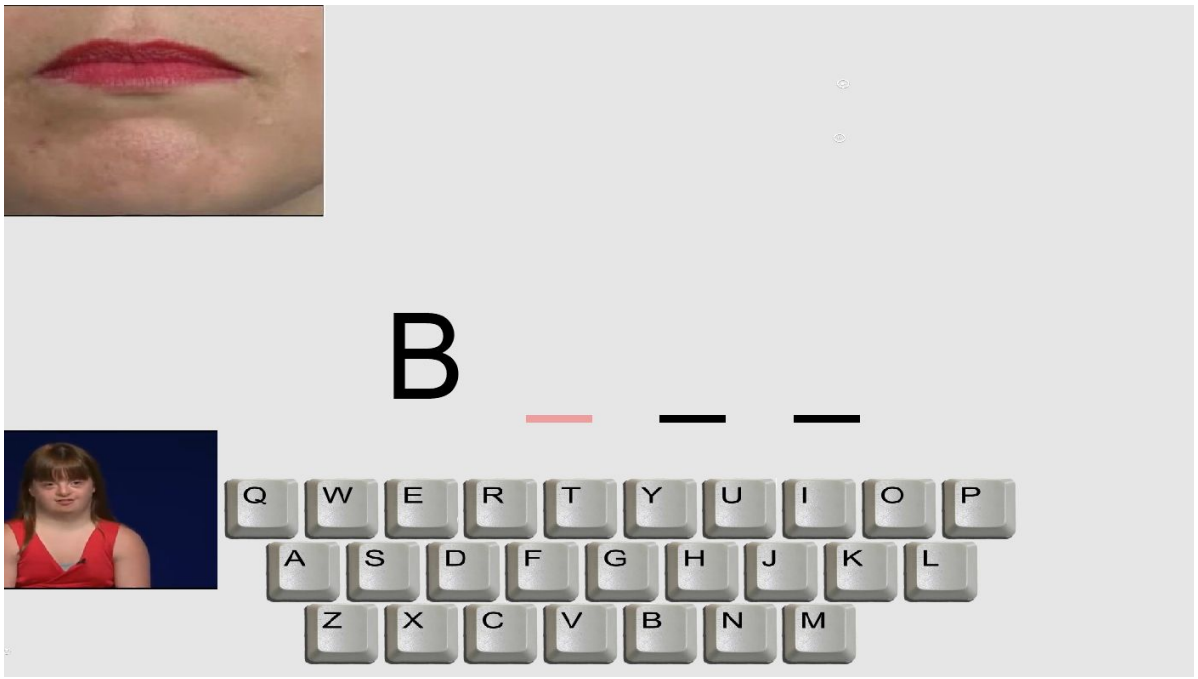
Software D2: Imagem 6

Software D2: Imagem 7

Software D2: Imagem 8



Software D2: Imagem 9



Software D2: Imagem 10

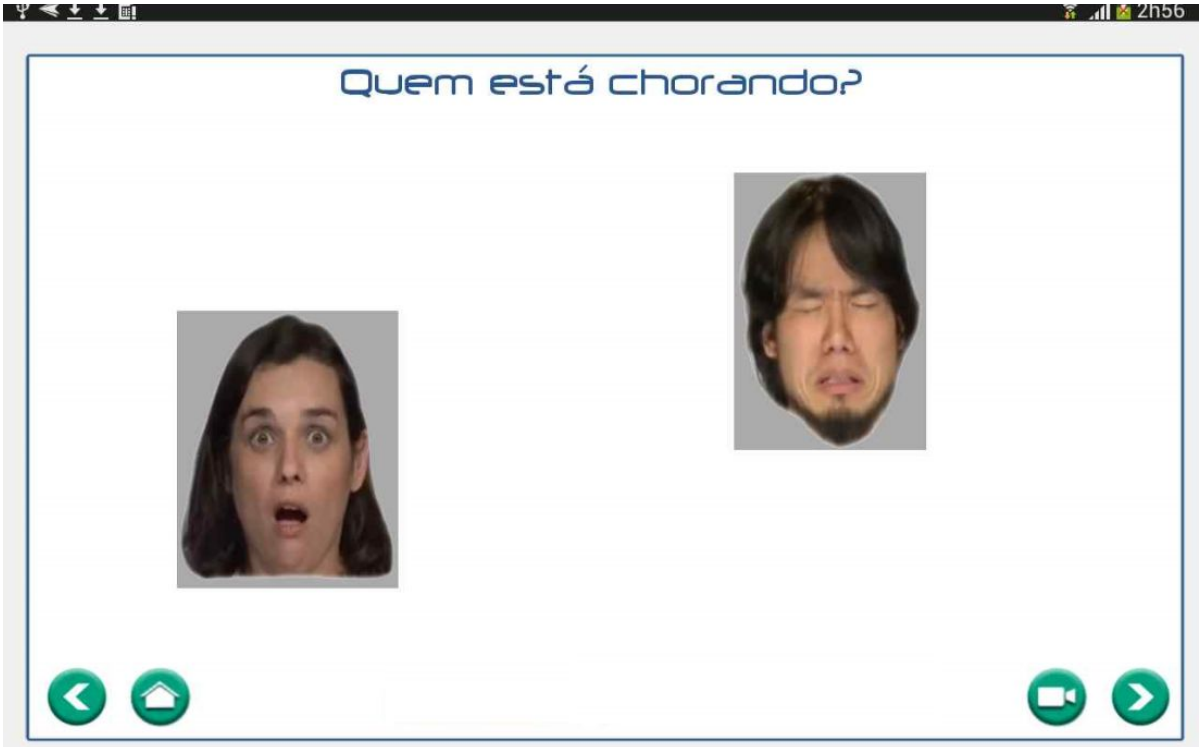
Software D2: Imagem 11

Software D2: Imagem 12

Relatório de usabilidade - Software A

Id	Heurística nº	Descrição	Imagem	Atividade
#1	8	Para iniciar as atividades, sempre são necessárias uma série de ações e configurações, não existe forma de acelerar o processo.	1	Todas atividades
#2	9	Na Lição 1.1 de todos os módulos, o botão **"Voltar"** tem a mesma função do botão **"Home"**.	2	**Lição 1.1**
#3	3	A acão "**Voltar ao Menu"** está associada com um botão com o símbolo do botão **"Avançar"**.	3	Todos os módulos
#4	3	Ao escolher os atores dos vídeos dos módulos, não é possível "desselecionar" um ator já selecionado, na mesma tela.	4	Módulos 1,2,3,4-> **Escolha de atores**
#5	3	Ao serem escolhidas 18 ou mais opções de atores para os **módulos 1,2 e 3**, é produzido um defeito que impede a escolha de qualquer ator para o **módulo 4.** Assim, torna-se impossível passar da etapa de atores, pois o software exige que a escolha seja feita.	5	Módulos 1,2,3,4-> **Escolha de atores**
#6	1	A escolha de recurso motivacional nas lições dos módulos exige uma escolha e não permite que o usuário saiba qual é o recurso que está sendo usado naquele momento.	6	Módulos 1,2,3,4-> Lições **Recurso motivacional**
#7	6	A **lista de opções de recursos** mostra a opção **Imagem/vídeo escolhido pelo Prof**. mesmo quando essa escolha não foi feita na tela de **configuração**.	6	Módulos 1,2,3,4-> Lições **Recurso motivacional**

Software A: Imagem 1

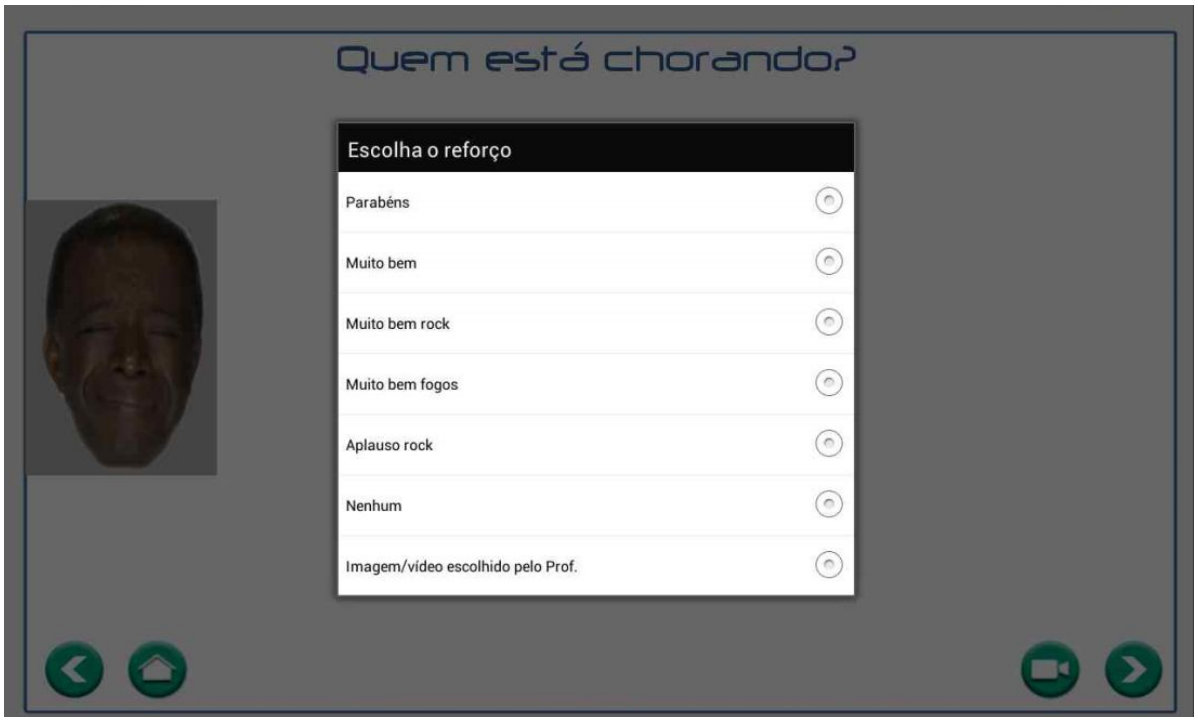


Software A: Imagem 2

Software A: Imagem 3

Software A: Imagem 4

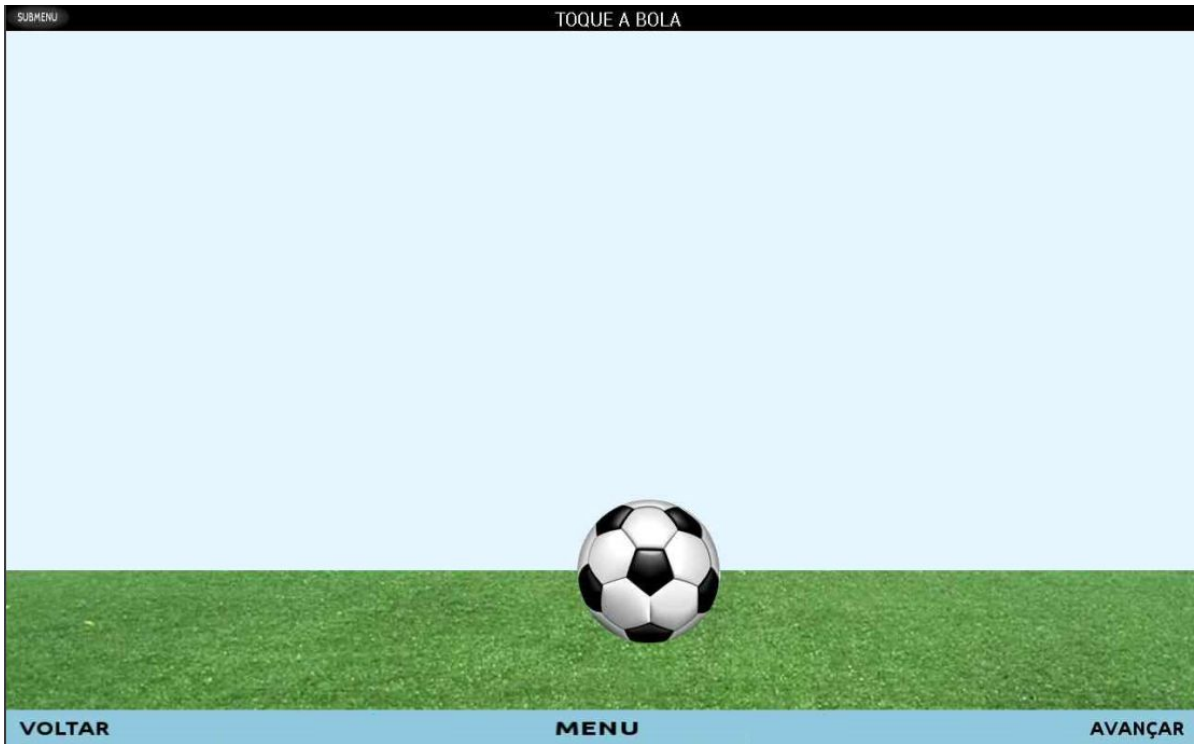
Software A: Imagem 5



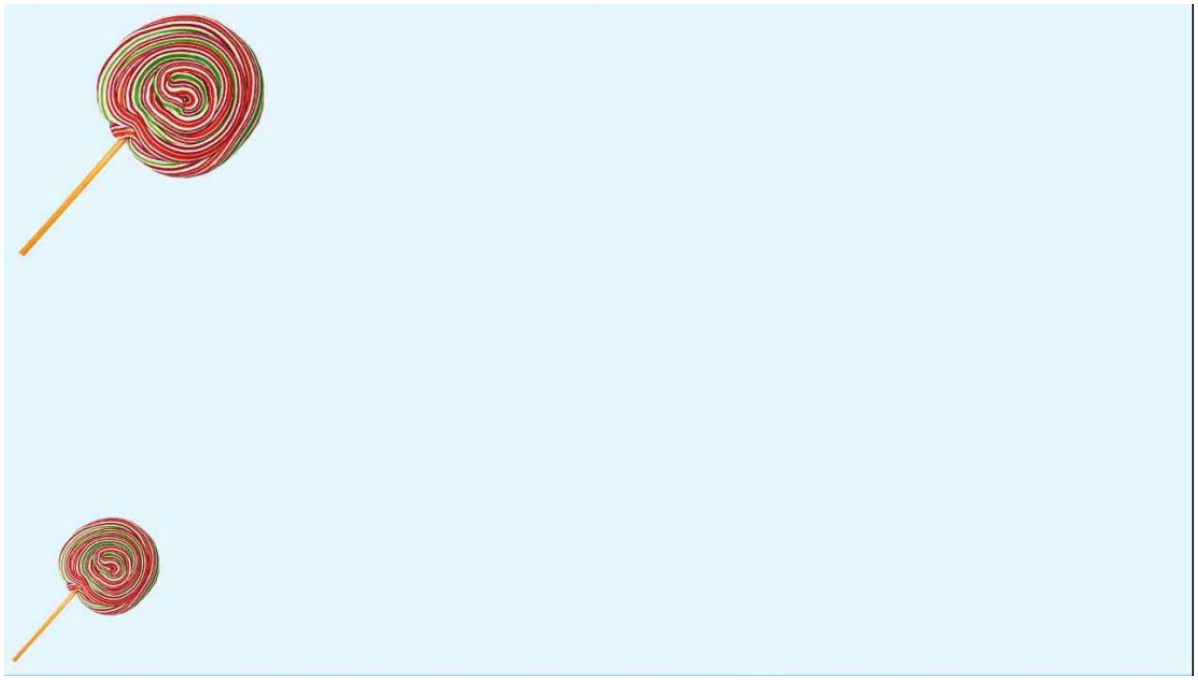
Software A: Imagem 6

Relatório de usabilidade - Software B

Id	Heurística nº	Descrição	Imagem	Atividade
#1	9	Em muitas atividades, a primeira tela possui o botão "**Voltar"** que não volta à tela anterior, mas sim à atividade anterior do ciclo de atividades.	1	Todas atividades
#2	9	Muitas atividades possuem apenas uma tela, para essas telas não há necessidade do botão Voltar e Avançar no rodapé.	1	Todas atividades que possuem apenas uma fase/tela
#3	2	Todas as atividades estão dispostas em uma lista circular, de forma que o usuário não sabe qual é a última atividade do ciclo.	2	Todas atividades
#4	4	O botão **"submenu"** presente em todas as atividades, não está consistente visualmente em relação aos outros botões da tela e sua visibilidade está prejudicada.	2	Todas atividades
#5	2	Na atividade **identificação de atributos**, a grafia do título da tela está incorretamente inserida como **"indentificação de atributos"**.	3	**Identificação de atributos**
#6	4	O software possui mecanismo que dá feedback e previne os erros do usuário, mas o mecanismo não está presente consistentemente em todas as atividades.	4	Várias atividades
#7	4	O design de "**Arroz e Feijão**" e "**Doce e Salgado"** dá a entender que eles são botões, como os botões "**Nível** 1, **Nível** 2, etc..." presentes na mesma tela.	5	**Aplicabilidade Social**
#8	1	Nas opções de vídeos motivacionais não há como saber qual vídeo está sendo usado atualmente, e a escolha de um vídeo é exigida.	6	**Opções de vídeos motivacionais**

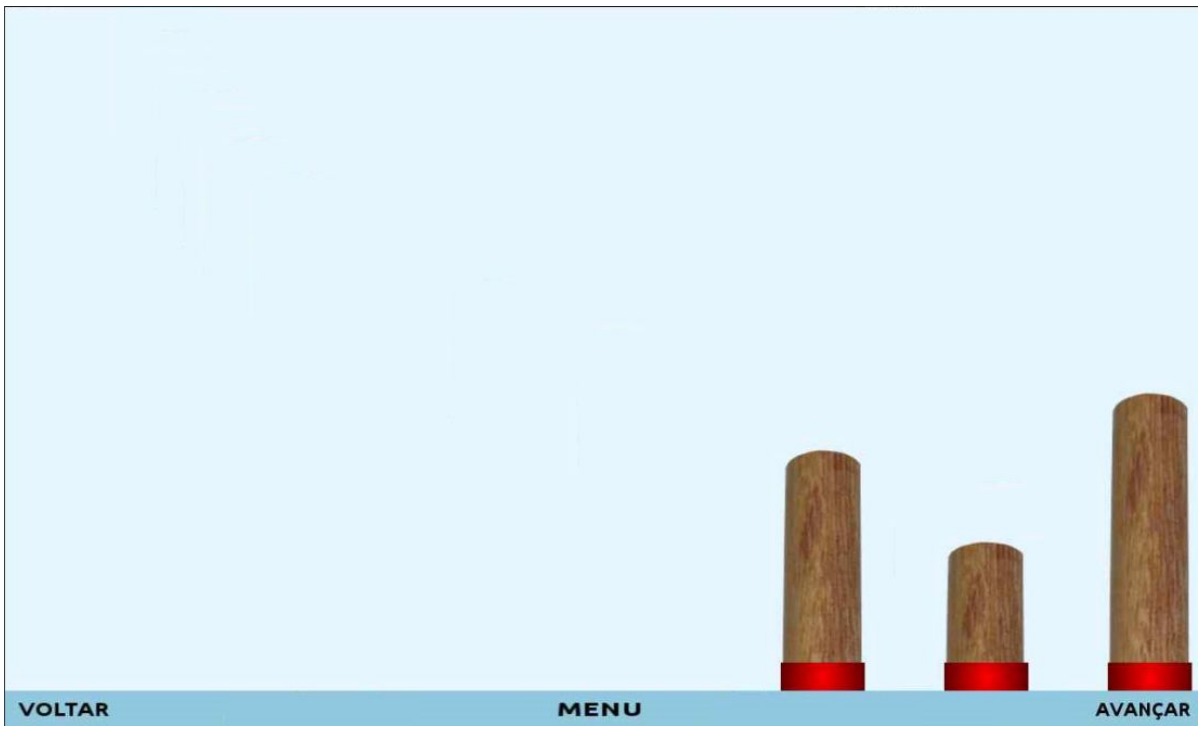


Software B : Imagem 1



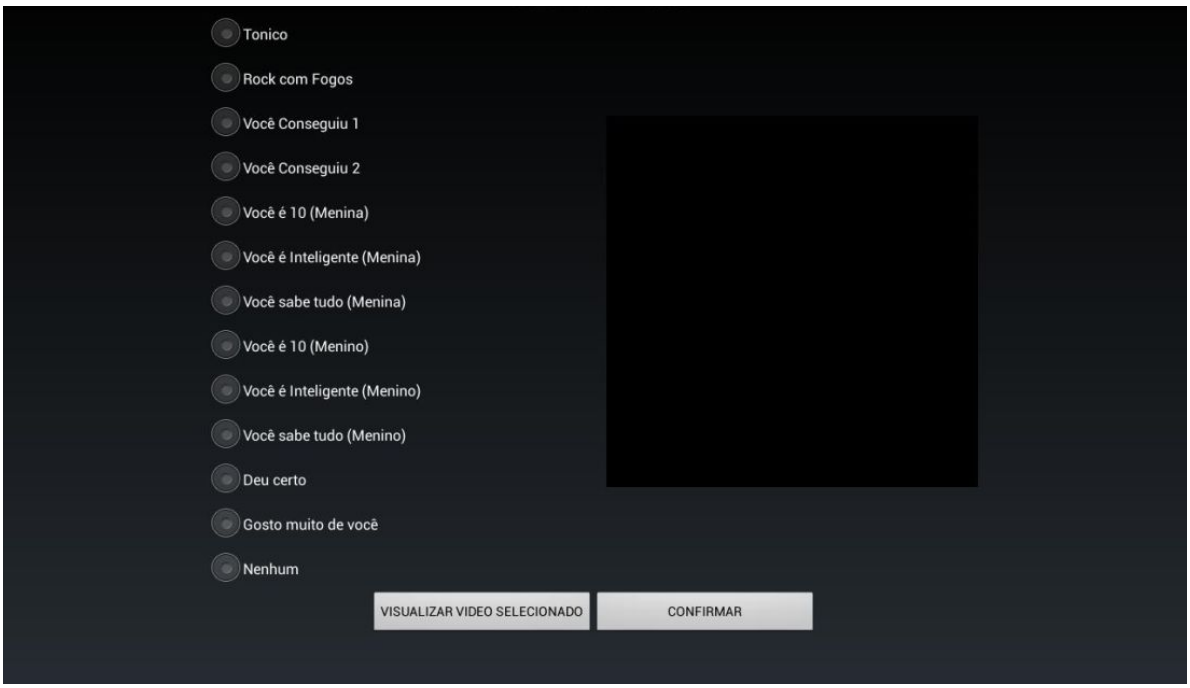
Software B : Imagem 2

Software B : Imagem 3



Software B : Imagem 4

Software B : Imagem 5



Software B : Imagem 6

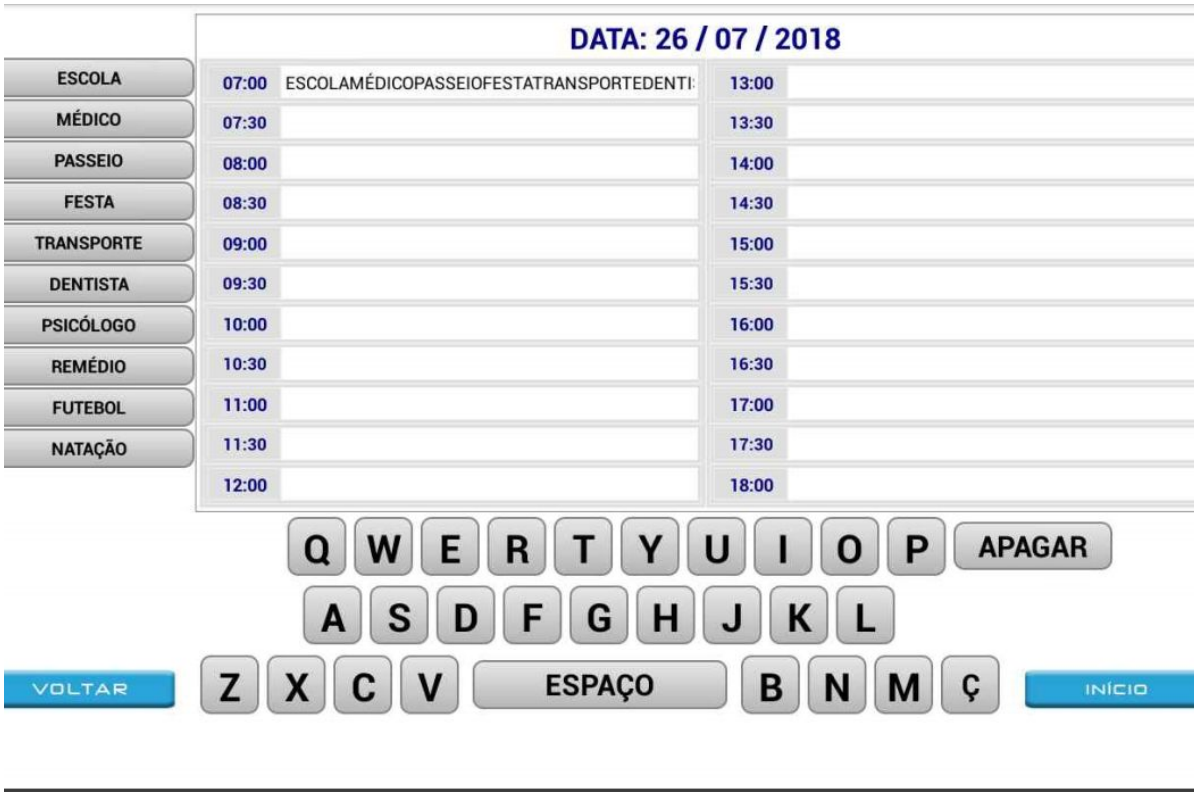
Relatório de usabilidade - Software G

Id	Heurística nº	Descrição	Imagem	Atividade
#1	9	Em muitas atividades, a primeira tela possui o botão "**Voltar"** exercendo a mesma função do botão **"Início"**.	1	Várias atividades
#2	8	Sempre é necessário o usuário configurar um vídeo motivacional e indicar o clima do dia, antes de iniciar a atividade "**Clima"**. Não existe atalho para um início mais ágil.	2	**Clima**
#3	2	Ao clicar em "**Voltar"**, o software volta à duas telas anteriores e não apenas uma.	3	Estações do ano-> Estação-> **Vídeo** Agenda-> Vídeo Explicativo-> **Tela de agendamento**
#4	2	É possível atribuir um número infinito de atividades para um horário da lista.	4	Agenda-> Vídeo Explicativo-> **Tela de agendamento**
#5	1	O software não dá feedback para identificar visualmente, à qual horário será atribuído uma atividade.	5	Agenda-> Vídeo Explicativo-> **Tela de agendamento**
#6	1	O software não dá feedback para identificar visualmente, qual será o horário que terá a atividade apagada.	5	Agenda-> Vídeo Explicativo-> **Tela de agendamento**
#7	1	O software não dá feedback para identificar visualmente, a qual campo será atribuído o texto digitado no teclado virtual.	5	Agenda-> Vídeo Explicativo-> **Tela de agendamento**
#8	4	É possível mudar o dia no calendário, mas as mudanças não se refletem nas atividades.	6	**Calendário**
#9	1	Não há feedback do sistema ao finalizar a atividade.	7	Atividades-> Estações do Ano-> Estação-> **Roupas Masculinas e Femininas**

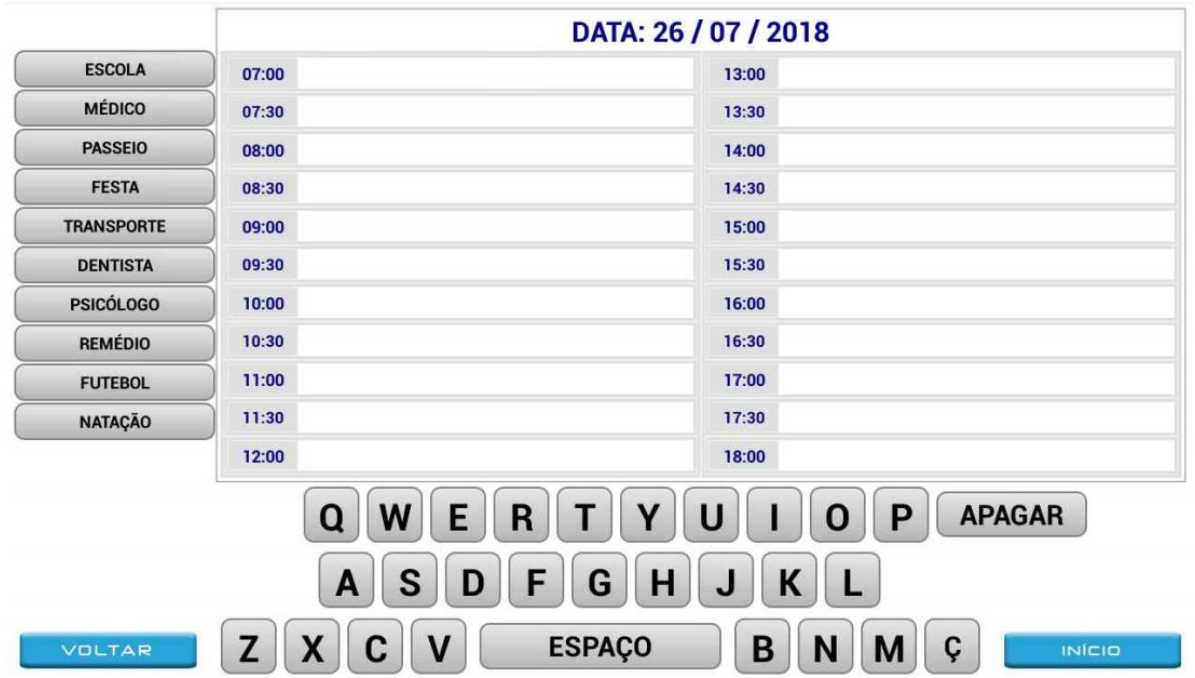
Software G: Imagem 1

Software G: Imagem 2

Software G: Imagem 3

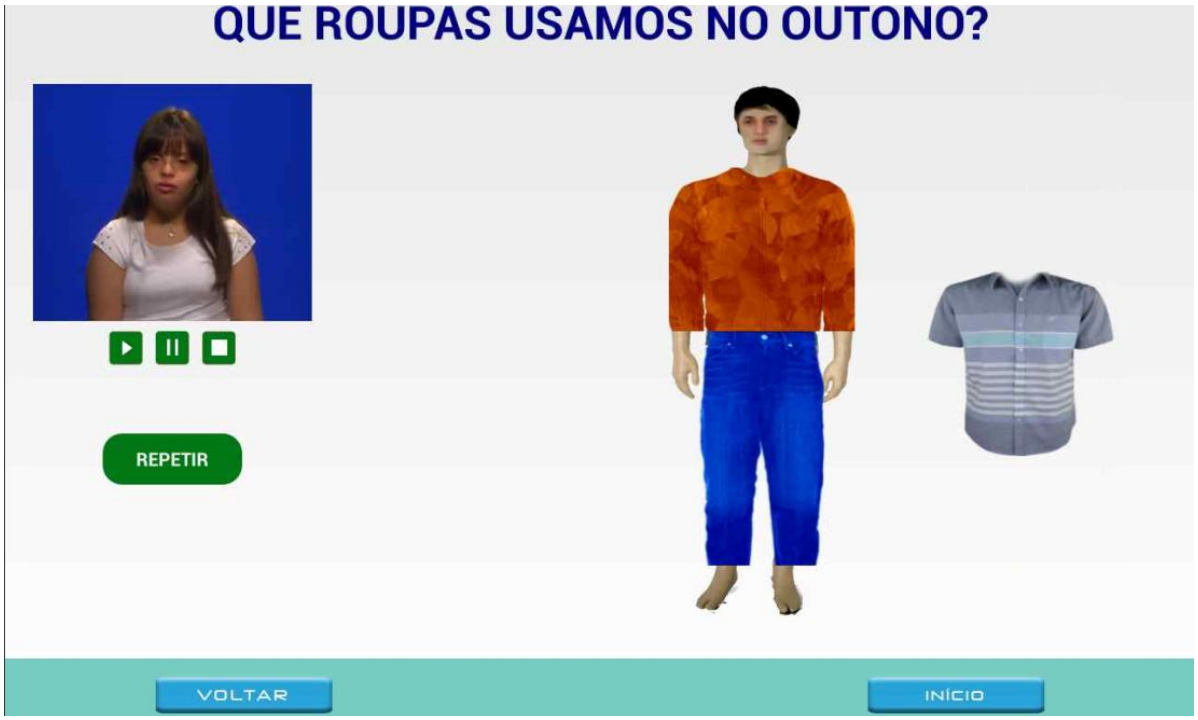


Software G: Imagem 4



Software G: Imagem 5

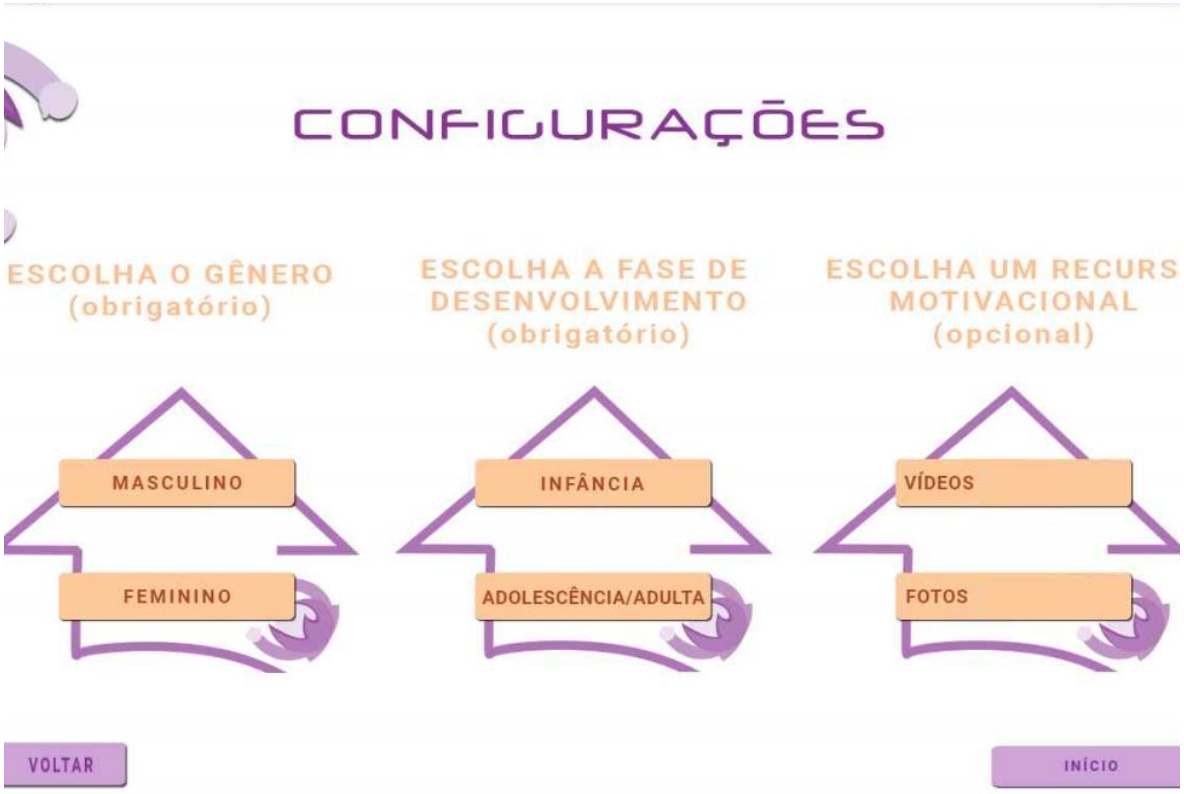
Software G: Imagem 6



Software G: Imagem 7

Relatório de usabilidade - Software C

Id	Heurística nº	Descrição	Imagem	Atividade
#1	1	Não fica claro se a escolha de gênero, fase de desenvolvimento e recurso motivacional foram realizadas, o feedback do estado do software é ineficiente.	1	**Configurações (Iniciar)**
#2	9	Os dois botões de escolha de recurso motivacional estão desabilitados e sem efeito.	1	**Configurações (Iniciar)**
#3	9	O botão "**Refazer**" aparece mesmo quando o usuário está iniciando a atividade pela primeira vez.	2	Todas as atividades
#4	9	Os botões "**Voltar"** e "**Avançar",** nas primeiras e últimas telas das atividades, respectivamente, não tem função alguma.	2	Todas as atividades exceto: **Ambientação, Sons dos Objetos e Guardar Talheres**
#5	2	O botão "**Voltar"** não leva de volta ao menu "**Sons dos objetos",** e o botão **"Avançar"** não leva ao próximo objeto de acordo com a **ordem** estabelecida no Menu.	3	Sons dos objetos-> **Qualquer objeto**



Software C: Imagem 1

Software C: Imagem 2

Software C: Imagem 3

Relatório completo dos testes de confiabilidade

Software	Atividades	Tempo de Ciclo	Nº de Falhas	Local/Tempo de Falha
E2 (Somar PC)	1.Dinheiro(Cédulas) 2.Dinheiro(Moedas) 3.Números (Dezena) 4.Números(Dúzia) 5.Calculadora(Troco)	6 min	5	1.Dinheiro(Moedas) 2 min 10s 2.Dinheiro(Cédulas) 19 min 5s 3.Números(Dezena) 15 min 40s 4.Números(Dezena) 21 min 50s 5.Dinheiro(Cédulas) 18 min 30s
F (ADV)	1.Passar desodorante(M) 2.Passar blush 3.Passar desodorante(F) 4.Limpar orelhas 5.Cortar unha das mãos	4 min 20s	0	N/A
E (Somar Tab)	1.Calculadora(Produto) 2.Calculadora(Dinheiro) 3.Ambientação(Toque) 4.Números(Numerais) 5.Números(Dúzia)	9 min 30s	1	5.Números(Numerais) 23 min 5s (FI)
D (Participar)	1.Exercício(Dificuldade 1) 2.Exercício(Dificuldade 2) 3.Exercício(Dificuldade 3) 4.Exercício (Palavras Especiais) 4.Exercício (Palavras Alimentos)	10 min	0	N/A
D2(PC)	1.Escolha uma lição[0-9] 2.Acentuação 3.Pontuação 4.Exercício (Palavras Alimentos) 4.Exercício (Palavras Objetos)	8 min	5	1.Escolha uma lição 8 min 32s (FI) 2.Escolha uma lição 8 min 40s (FI) 3.Escolha uma lição 8 min 30s (FI) 4.Escolha uma lição 9 min 2s (FI) 5.Escolha uma lição 8 min 3s (FI)

A(Expressar)	1.Atividades (Módulo 1) 2.Atividades (Módulo 2) 3.Atividades (Módulo 3) 4.Atividades (Módulo 4) 5.Atividades (Módulo 4)	4 min	0	N/A
Software	Atividades	Tempo de Ciclo	Nº de Falhas	Local/Tempo de Falha
B(Perceber)	1.Ambientação (Toque) 2.Ambientação(Arrastar) 3.Emparelhamento por associação 4.Identificação de atributos 5.Identificação de objetos	9 min 30s	1	3.Ambientação (Arrastar) 15 min 0s
G(Organizar)	1.Calendário (Dia da Semana) 2.Calendário (Dia do Mês) 3.Calendário (Número do Mês) 4.Calendário (Nome do Mês) 5.Calendário (Número do Ano)	1 min 30s	0	N/A
C(Ambientar)	1.Ambientação (Toque) 2.Ambientação(Arrastar) 3.Arrumar cama 4.Guardar talheres 5.Abrir/Fechar porta	5 min	5	1.Menu de escolha 5 min 50s (FR) 2.Menu de escolha 8 min 40s (FR) 3.Menu de escolha 8 min 30s (FR) 4.Menu de escolha 9 min 2s (FR) 5.Menu de escolha 8 min 3s (FR)

Apêndice B

Apêndice B - Scripts de automatização dos testes no Sikulix

Scripts de automatização dos testes no Sikulix

```
ImagePath.setBundlePath("C:/sikulix/screenshots/adv/");

Debug.on(3)
Settings.DelayAfterDrag = 0.2
Settings.MoveMouseDelay = 0.3
Settings.MinSimilarity= 0.9

regiaoCima = Region(804,150,322,223)
regiaoPia = Region(811,435,358,185)

setShowActions(True)
setAutoWaitTimeout(5)

click("/botoes/cortarUnhaDasMaos")

dragDrop("/objetos/cortadorUnha",regiaoPia)
click("/botoes/avancar")
click("/botoes/avancar")
click("/objetos/clickCortador")
click("/botoes/avancar")
click("/botoes/avancar")
dragDrop("/objetos/maoCortador",regiaoPia)
click("/botoes/avancar")
```

Software F: Script de automatização da atividade “Cortar Unha das Mãos”

```
#ESCOLHE UM VALOR PARA GASTAR (10 reais)
click("10Mercado.png")
sleep(2)
#ARRASTA O ARROZ PARA O CARRINHO
dragDrop("arrozMercado.png","carrinhoMercado.png")
sleep(3)
Settings.DelayAfterDrag = 2
Settings.MoveMouseDelay = 0.6
#ARRASTA O MACARRÃO PARA O CARRINHO
dragDrop("macarraoMercado.png","carrinhoMercado.png")
Settings.DelayAfterDrag = 2
Settings.MoveMouseDelay = 0.6
#CLICA EM PAGAR
click("btnPagar.png")
sleep(2)
#ARRASTA O DINHEIRO PARA PAGAMENTO
dragDrop("10MercadoCarteira.png","caixaMercado.png")
sleep(4)
click("btnAvancarGrande.png")
```

Software E: Script de automatização da atividade “Mercado”

```
origem=Region(0,0,0,0)
botaoAvancar=Region(1135,720,127,38)
click("btnDinheiro")
click(origem)
type("2")
click("pontoCalculadora")
type("00")
sleep(2)
click(botaoAvancar)
click(origem)
type("5")
click("pontoCalculadora")
type("00")
sleep(2)
click(botaoAvancar)
sleep(2)
click(origem)
type("5")
click("maisCalculadora.png")
type("2")
click("igualCalculadora.png")
click(botaoAvancar)
sleep(2)
```

Software E2: Script de automatização da atividade “Dinheiro”

```
Settings.MinSimilarity=0.5
ImagePath.setBundlePath("C:/Users/Victor/Documents/AirDroid/ScreenShot/ambientar");
Settings.DelayAfterDrag = 2.5
Settings.DelayBeforeDrop = 2.5
Settings.MoveMouseDelay = 4

dragDrop("objetos/arrasteTravesseiro","objetos/arrastadaCama")

Settings.DelayAfterDrag = 1
Settings.DelayBeforeDrop = 1
Settings.MoveMouseDelay = 1

click()
sleep(2)

```

Software C: Script de automatização da atividade “Arrastar”

```
j=0
i=1
sufixos = ("Pequeno","Medio","Grande")
while(i<6):

    if wait("/botoes/nivel" + str(i)):
        click("/botoes/nivel" + str(i))

    if(i<4):
        dragDrop("/objetos/bastao" + sufixos[i-1],"/objetos/posicaoVaga" + str(i))
    else:
        nMovimentos = i-2

        while(j<nMovimentos):
            dragDrop("/objetos/bastao" + sufixos[j],"/objetos/posicaoVaga" + str(j+1))
            j=j+1

        j=0

    if wait("/telas/sucesso",2):
        wait(5)
    i = i+1

    hover("/botoes/submenu")
    mouseDown(Button.LEFT)
    wait(1)
```

Software B: Script de automatização da atividade “Seriação”

```
#TESTANDO DIFICULDADE 2

dif2 = ["meia","roupa","padaria","gelo","celular","lixeira","gilete","salada"]
click("btnDif2.png")

for nWord in range(0, len(dif2)):
    for i in range (0, len(dif2[nWord])):
        click(str(dif2[nWord][i]) + ".png")

    click("btnAvancar.png")
    sleep(1)
```

Software D: Script de automatização da atividade “Exercícios Dificuldade 2”

```
pontuacao = [".",".",".",".",".",".",".",","]

for nWord in range(0, len(pontuacao)):
    for i in range (0, len(pontuacao[nWord])):
        type(pontuacao[nWord][i])
        sleep(0.2)



    click("btnAvancar.png")
    hover(origem)
    sleep(1)
```

Software D2: Script de automatização da atividade “Exercícios de Ponto Final”

```
# TESTE DAS LICOES 1 ATÉ LICAO 5 - MODULO 1 - SORRISO

click("botoes/licao1-1")
wait("telas/quemEstaSorrindo")

a=1

while (True and a<20):

    if regiaoExpressoes.exists("expressoes/sorriso"+str(a),0):
        regiaoExpressoes.click("expressoes/sorriso"+str(a))
        click("botoes/flechaDireitaBranco")
        break
    a+=1

a=1

wait("telas/quemEstaSorrindo")

while (True and a<20):

    if regiaoExpressoes.exists("expressoes/sorriso"+str(a),0):
        regiaoExpressoes.click("expressoes/sorriso"+str(a))
```

Software A: Script de automatização da atividade “Módulo 1”

```
#TESTE DA ATIVIDADE 4 ESTAÇÕES ( TESTE VERAO - ESTAÇÃO)

click("btn4Estacoes.png")
sleep(1)
click("btnVerao.png")
click("btn4Estacoes.png")

if exists("verao1.png"):
        fotoVerao = "verao1.png"
elif exists("verao2.png"):
        fotoVerao = "verao2.png"
else:
        fotoVerao="verao3.png"

dragDrop("palavraVerao.png", fotoVerao)
```

Software G: Script de automatização da atividade “Estações do ano / Verão”