Informática

Fundamentos e análise de software educativo

Rosa Lívia Freitas de Almeida
Carlos Alberto Santos de Almeida

2ª edição
Fortaleza - Ceará

Ed UECE

2015

Química | Ciências Biologicas | Artes Plásticas | Computação | Física | Matemática | Pedagogia

Editora Filiada à

ASSOCIAÇÃO BRASILEIRA DAS EDITORAS UNIVERSITÁRIAS

Editora da Universidade Estadual do Ceará – EdUECE
Av. Dr. Silas Munguba, 1700 – Campus do Itaperi – Reitoria – Fortaleza – Ceará
CEP: 60714-903 – Fone: (85) 3101-9893
Internet: www.uece.br – E-mail: eduece@uece.br

Secretaria de Apoio às Tecnologias Educacionais
Fone: (85) 3101-9962

Sumário

Apresentação

No cenário educacional, o computador é o mais novo instrumento a fazer parte do processo ensino-aprendizagem. Com a introdução do computador como mediador didático, desenvolveram-se softwares específicos para serem utilizados na interação aluno-professor-conhecimento. O software educativo é um programa de computador que possui recursos que foram projetados com a intenção e a finalidade de serem usados em contexto de ensino-aprendizagem. As características principais que distinguem um software educativo é o seu desenvolvimento fundamentado em uma teoria de aprendizagem e a capacidade que o aluno tem de construir de forma autônoma o conhecimento sobre determinado assunto.

O presente livro apresenta de forma clara e amigável os fundamentos que norteiam o desenvolvimento do software educativo bem como alguns métodos e práticas da engenharia de software utilizada no seu desenvolvimento. O conhecimento da Engenharia de Software é requerido para melhor compreensão do tema abordado.

O livro está organizado em cinco Capítulos. O primeiro capítulo fornece a visão geral do software educativo e apresenta as diferentes formas como os softwares educativos têm se inserido no ambiente formal de aprendizagem.

No Capítulo 2, Desenvolvimentos de software educativo, são apresentadas as fases do modelo de ciclo de vida do software e um modelo de desenvolvimento para o software educativo.

No Capítulo 3, são explicados os conceitos fundamentais da Engenharia de Software e alguns fluxos de atividades que normalmente participam do processo de desenvolvimento de software, focando o modelo de ciclo de vida comumente utilizado no desenvolvimento do software educativo.

O Capítulo 4 tem seu foco no conceito de Objetos de Aprendizagem (OAs). São também apresentados conceitos e metodologias de desenvolvimento de OAs, tendo em vista que um OA pode ser considerado como um produto de software.

O Capítulo 5 é dedicada à avaliação de software educativo. Apresentamos uma visão geral do assunto, abordando diversos atributos que devem ser alcançados. Especificamente em relação ao software educativo, é apresentado um conjunto de características que devem ser pontuadas em sua avaliação.

O conteúdo apresentado destina-se principalmente a professores e alunos de graduação em Ciências da Computação ou áreas afins, fornecendo um embasamento teórico e uma clara noção dos princípios e estratégias a serem seguidos no desenvolvimento de software educativo de qualidade.

Os autores

Capítulo 1

Visão geral do software educativo

Objetivo

- O material instrucional tem evoluído com as tecnologias de comunicação ao longo dos séculos. O computador e os softwares provocaram reflexões sobre o processo de aprendizagem e se inseriram no cotidiano das salas de aula presenciais e virtuais. Nesta unidade são apresentadas as diferentes categorias de software educativo.

Introdução

O domínio da educação na sociedade está pautado na transmissão dos conhecimentos conquistados para as gerações futuras. Entretanto a educação não se reduz a ensino. Como transferência de informação, o conhecimento exige a construção de representações internas que estão para além do simples acesso à informação. Vários são os métodos e os recursos utilizados para este fim. Chama-se de tecnologia educacional o conjunto de recursos, métodos e sistemas educacionais que auxiliam como ferramenta no processo de ensino.

> A tecnologia educacional está fundamentada em três teorias: da comunicação, da aprendizagem e de sistemas e não pode ser chamada de ciência.

As tecnologias têm evoluído com muita rapidez e desempenham um papel preponderante como elemento transformador do modo de acessar e organizar o universo da informação, colocando novos desafios pedagógicos na tarefa de auxiliar o aluno a organizar novos conhecimentos.

Ensinar é antes de tudo um processo em que se incluem recursos físicos e materiais que são utilizados pelo professor nos procedimentos da mediação do conhecimento.

Quando falamos de procedimentos e recursos no campo do ensino para a construção do saber teórico-prático, destaque deve ser dado para as possibilidades da informática educativa entre as novas tecnologias denominadas interativas. Tal ferramenta serve como um instrumento a mais de apoio ao professor, funcionando como meio didático.

No Brasil, a cultura da informática educativa teve início na década de 70 com a discussão do uso do computador no ensino da Física. A ideia de "sociedade do conhecimento" foi sustentada pelas transformações tecnológicas

e a sempre crescente popularização do computador pessoal tem alargado o uso desta ferramenta por diversas áreas do conhecimento como instrumento plausível na ampliação da aprendizagem.

A especificidade do conhecimento a ser trabalhado e as particularidades didáticas subjacentes evidenciam a dimensão epistemológica da educação ao mesmo tempo em que revela a importância das situações geradas em que é determinado um valor funcional aos conhecimentos, aos métodos implícitos ou explícitos que determinam a interação aluno/professor relacionada às situações e às dificuldades cognitivas que podem ser encontradas.

Softwares educacionais são programas que visam atender necessidades vinculadas à aprendizagem. Devem ter objetivos pedagógicos e sua utilização deve estar inserida em um contexto e em uma situação de ensino baseados em uma metodologia que oriente o processo, através da interação, da motivação e da descoberta, facilitando a aprendizagem de um conteúdo.

Assim, o desenvolvimento de sistemas na área educativa deve considerar a contribuição didática em seus aspectos teóricos e metodológicos. Isto implica dizer que projetos de desenvolvimento de software educacional, além de envolver uma equipe multidisciplinar, devem resultar em produtos que reflitam os objetivos educacionais propostos e o ambiente de aprendizagem almejado, criando situações que estimulem o desenvolvimento das habilidades desejadas.

O software educativo é considerado o elemento que permite ao professor introduzir o computador em sala de aula, pois sem ele o computador jamais poderá ser utilizado como ferramenta educacional.

> O aluno, o computador, o software educativo e o professor treinado para o uso do computador na sala de aula são os quatro componentes básicos para viabilizar a implantação da informática na educação.

O requisito essencial do software a ser utilizado nas escolas é a não substituição das atividades educacionais já existentes. Não deve ser simplesmente uma versão computadorizada dos atuais métodos de ensino. O computador deve ser uma ferramenta de complementação, de aperfeiçoamento e de possível mudança na qualidade do ensino.

O computador é uma ferramenta que deve propiciar as condições para os estudantes exercitarem a capacidade de procurar e selecionar informação, resolver problemas e aprender independentemente. Portanto, em vez de memorizar informação, os estudantes devem ser ensinados a procurar e a usar a informação. Esse é um dos pressupostos que deve guiar o desenvolvimento de software educativo.

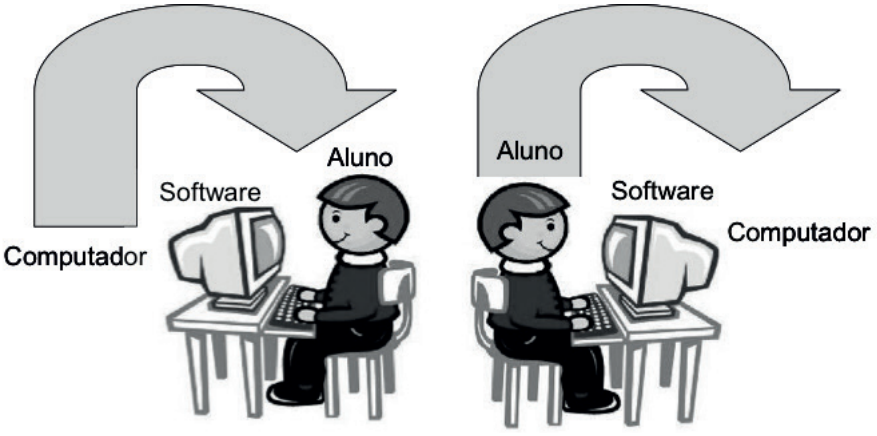
Atividades de avaliação

1. O que são softwares educacionais?
2. Quais aspectos pedagógicos devem estar contidos em um software educacional?

3. Pesquise sobre o desenvolvimento de software no Brasil.
4. Qual o impacto da evolução tecnológica e o desenvolvimento social no contexto da educação?

1. Tipos de Software Educativos

Como vimos, uma das questões fundamentais no desenvolvimento de um software educativo é o aspecto pedagógico, ou seja, o que o software se propõe a ensinar e como isto é feito. A abordagem pedagógica de como isso acontece é bastante variada, oscilando entre duas vertentes resumidas na figura abaixo:



Essas vertentes são caracterizadas pelos mesmos elementos em direções opostas. Na primeira vertente, o computador, através do software, ensina ao aluno. Já na outra, o processo é inverso: o aluno, através do software, "ensina" ao computador.

Quando o computador ensina ao aluno, o software educacional assume a responsabilidade de transformar o computador em uma máquina de ensinar. São programas que conduzem a aprendizagem de um conjunto de habilidades, quer sejam específicas, de motricidade, de percepção ou outras.

O termo ensino tem origem na palavra latina insignare e significa transmissão de conhecimento, de informação ou de esclarecimentos úteis ou indispensáveis à educação e à instrução.

Neste caso, a abordagem educacional é designada de “instrução auxiliada por computador”. Essa abordagem tem suas raízes nos métodos de instrução programada tradicionais e assume que a retenção do conhecimento se dá como consequência da contiguidade e da frequência com que ele é transmitido.

Nessa concepção de ensino, o material instrucional é uma compilação classificada do conhecimento gerado pela humanidade e hierarquizada em diferentes graus de dificuldade, para ser ministrado ao aluno do nível mais fácil para o mais difícil. Nesse modelo, o que ocorre é a substituição do papel ou do livro por um computador.

Os softwares que programam essa abordagem podem ser divididos nas categorias tutoriais, softwares de exercício-e-prática (drill-and-practice) e na categoria dos jogos educacionais e de simulação. Os tutoriais enfatizam a apresentação das lições ou a explicitação da informação. No exercício-e-prática, o processo de ensino está baseado, como o nome deixa claro, na realização de exercícios com grau de dificuldade variado. Nos jogos educacionais, a ênfase está no lúdico e a abordagem pedagógica utilizada é a exploração autodirigida em vez da instrução explícita e direta.

Na vertente em que o aluno "ensina" ao computador, a estratégia é que a máquina se torne uma ferramenta para soluções de problemas ou um realizador de tarefas como desenhar, escrever, comunicar-se. Esta categoria de software é também conhecida como programas para a aprendizagem de habilidades cognitivas amplas, ou seja, a consciência sobre o conhecimento, a fluência nas ideias, a capacidade de reorganizar e de desenvolver sua originalidade, entre outras.

Para esse tipo de abordagem normalmente usa-se uma linguagem computacional, por exemplo, BASIC, Logo, Pascal ou uma linguagem para criação de banco de dados do tipo Access, DBase, ou mesmo um processador de texto ou um software de multimídia, que permite ao aluno representar suas ideias segundo esse software.

Para "ensinar" ao computador a realizar uma determinada tarefa, o aluno deve utilizar conteúdos e estratégias. Por exemplo, programar o computador usando uma linguagem de programação requer do aluno o conhecimento prévio de uma linguagem de programação. A interação com o computador através da programação precisa da descrição de uma ideia em termos de uma linguagem formal e precisa. O computador somente é capaz de executar fielmente a descrição fornecida produzindo um resultado obtido que é fruto somente do que foi solicitado à máquina. O resultado obtido permite ao aluno refletir sobre o que foi solicitado ao computador. O resultado não correspondendo ao que foi idealizado permitirá ainda que o aluno depure ou aperfeiçoe a ideia original através da aquisição de conteúdos ou de estratégias. A construção do conhecimento acontece porque o aluno busca novas informações para complementar ou alterar o que ele já possui. Nesta perspectiva, o aluno está criando suas próprias soluções, está pensando e aprendendo sobre como buscar e usar novas informações (aprendendo a aprender).

Podemos perceber que ambas as vertentes apresentam características próprias, vantagens e desvantagens. Estas características devem ser explicitadas e discutidas de modo que as diferentes modalidades possam ser usadas nas situações de ensino-aprendizado que mais se adequam à necessidade. O uso do computador na educação não pode ser visto simplesmente como um

facilitador do processo de aprendizagem, e sim como instrumento que promove a aprendizagem dos alunos, contribuindo efetivamente na construção do processo de conceituação e no desenvolvimento de habilidades importantes para que ele se integre no coletivo da sociedade do conhecimento.

1.1. Softwares Educativos – Modalidade e aplicações

1.1.1. Instrução Auxiliada por Computador (CAI)

Como já foi mencionado, a Instrução Auxiliada por Computador (CAI) é uma versão computadorizada dos métodos de instruções programadas tradicionais. As categorias mais comuns desta modalidade são programas de reforço e programas tutoriais. Basicamente, os programas de reforço ou de exercício são utilizados para revisar a matéria ministrada em sala de aula, principalmente os conteúdos que envolvem memorização e repetição. Estes programas, em geral, requerem resposta frequente do aluno, propiciam feedback imediato, exploram as características gráficas e sonoras do computador e muitas vezes são apresentados em formas de jogos.

Os programas tutoriais são conhecidos pela sua paciência infinita, já que neste sistema de ensino a criança aprende de acordo com o seu próprio ritmo. São programas de difícil implementação e possuem custo elevado. As empresas que desenvolvem softwares educativos preferem gastar mais no aspecto entretenimento do que no aspecto pedagógico, ou nos testes de refinamento do programa. A tendência dos bons programas CAI é utilizar técnicas de inteligência artificial que permitam analisar padrões de erro e avaliar o estilo e a capacidade de aprendizagem do aluno oferecendo ainda instrução especial para os conceitos que apresentam grau de dificuldade mais elevado.

Um software dito tutorial é um programa no qual a informação está organizada de acordo com uma particular sequência pedagógica que deve ser apresentada ao estudante seguindo essa sequência. Os softwares tutoriais guardam consigo o controle da situação tanto do ensino como do conteúdo a ser apresentado ao aprendiz.

Entretanto alguns tutoriais podem permitir ao aprendiz fazer a escolha da informação que deseja apreender. Neste caso, o aprendiz tem o controle do que deseja ver. Os tutoriais que permitem ao aluno controlar a sequência da informação são organizados em forma de hipertextos e o aprendiz pode navegar entre itens de informação.

A interação entre o computador e o aluno consiste na leitura da tela ou escuta da informação fornecida. A interface de respostas pode ser via teclado ou mouse, o que caracteriza que o estudante está fazendo algo, entretanto não se sabe qual o nível de compreensão do conteúdo aplicado.

Para resolver os problemas de avaliação da aprendizagem, é necessário implementar rotinas que possam extrair do estudante uma resposta baseada em situações problemas. De forma geral, o que ocorre é uma verificação da memorização do conteúdo ou uma aplicação direta do conteúdo apresentado. Vale destacar que os programas tutoriais apresentam como limitação justamente a incapacidade de verificar se a informação processada tornou-se conhecimento agregado aos esquemas mentais do aprendiz.

Os tutoriais e os softwares do tipo exercício-e-prática enfatizam a apresentação das lições ou de exercícios, centrando a ação do estudante em virar páginas de um livro eletrônico ou realizar exercícios, cujo resultado pode ser avaliado pelo computador. Nesta modalidade, o papel do professor é fundamental para criar condições de interação com o aluno que permitam levá-lo ao nível de compreensão desejado, evitando que o aluno faça coisas sem saber o que está fazendo ou tenha conclusões erradas a respeito do que foi abordado.

1.1.2. Aprendizagem por descoberta

A aprendizagem por descoberta é uma abordagem pedagógica que explora a instrução autodirigida. A ideia subjacente a esta filosofia é a de que a criança aprende melhor quando é livre para fazer suas próprias descobertas em vez de ser ensinada. Os softwares de jogos ou de simulação estão entre os preferidos desta modalidade pedagógica.

Os jogos são considerados objetos com capacidade para permitir o aprendizado de maneira divertida e descontraída. O grande problema desse instrumento é que a competição pode desviar a atenção do objetivo do jogo. Além do mais, a maioria dos jogos educativos explora conceitos triviais e incorrem no erro de não programar técnicas de diagnóstico das falhas do jogador.

1.1.3. Jogos de Computador

Os jogos de computador são muito populares. A tecnologia utilizada para desenvolvê-los tem evoluído muito nos últimos anos, principalmente devido ao alto investimento em pesquisa e desenvolvimento, porém o grande montante destes recursos está direcionado apenas para a área comercial, sendo que os jogos que apresentam maior sucesso comercial são os que combinam violência com efeitos visuais sofisticados.

As tecnologias que combinam entretenimento com educação são ainda pouco desenvolvidas, ou seja, existe um grande campo para a pesquisa nessa área. Há estimativas que 20% dos jogos disponíveis têm algum enfo-

que educacional, além de a maioria dos programas educacionais só auxiliar o desenvolvimento de reflexos e rapidez de raciocínio.

A ciência da computação não incorporou o desenvolvimento de jogos educativos como um de seus campos de atuação por si só, mas como uma aplicação interdisciplinar envolvendo estudos em diferentes áreas, tais como áudio, gráfico, HCI (sigla de Human-Computer Interaction: interface-usuário-computador), sistemas operacionais, design de compiladores, algoritmo, redes e sistemas distribuídos, engenharia e design de software e inteligência artificial.

Na categoria jogos, existem várias classificações e cada uma delas obedece a regras que definem características aplicáveis à interface e motor, que influenciam diretamente as técnicas de programação utilizadas. Os principais tipos de jogos são: estratégias, simuladores, aventura, passatempo, RPG (Role Playing Game), esporte e educativos/infantil.

A forma como o usuário interage com o jogo ocorre de diferentes maneiras a depender do tipo de jogo. Há casos em que o usuário atua como terceira pessoa. Nesta situação, ele se vê na cena e seu personagem é denominado de ator. Diversos tipos de jogos utilizam essa representação, em jogos de luta, por exemplo, o usuário se vê como um lutador. O usuário pode também atuar em primeira pessoa. Nesse caso, a cena do jogo é exibida do ponto de vista do usuário, como se fosse o olho do jogador. Nesse sistema, o personagem é denominado de avatar. Esse tipo de interface é muito utilizado em jogos de aventura e ação, em que o personagem caminha pelo cenário desvendando enigmas ou em uma batalha competindo com outros personagens. O avatar deve marcar a presença do personagem na tela e sua modelagem não deve prejudicar a ilusão de imersão do usuário no jogo.

A interface do jogo é um elemento importante que pode variar segundo diversas projeções. Quando o jogo é feito sobre um grande mapa, é usual utilizar projeções planares ortográficas do tipo planta, ou vista lateral, vista frontal ou axonométrica isométrica. Jogos que utilizam projeções planares são o *Simcity*[1] , *Age or Empires*[2] e *Commandos*[3] . Quando o jogo utiliza projeções perspectivas, o usuário tem uma noção de profundidade.

1 Eletronic Arts
2 Microsoft Corporation
3 Eidos Interactive

> A perspectiva isométrica é uma perspectiva axonométrica em que os raios projetantes são ortogonais a um plano vertical de projeção. Os eixos x, y e z têm a mesma inclinação em relação ao plano vertical. As projeções dos eixos formam entre si ângulos de 120. Fonte: www.mat.uel.br/geometrica

1.1.3.1. Jogos de Estratégia

Os jogos classificados na categoria "estratégia" colocam o usuário em posição de tomar decisões de grande importância. Sua função principal é a conquista de um objetivo através de análise crítica da situação. Essa modalidade de jogo possibilita ao jogador uma tarefa mais intelectual do que apenas reflexo reflexiva. Como exemplo, podemos citar o jogo SimCity, cujo objetivo é administrar uma cidade, e Age of Empires, que objetiva construir um exército e dominar civilizações.

De uma maneira geral, todos os jogos seguem regras, mas em jogos de estratégia há uma maior flexibilidade. Por exemplo, para a finalização do jogo não há normalmente um regra pré-estabelecida, ou seja, o próprio jogador, baseado em sua experiência, pode definir sua própria estratégia para vencer o jogo.

O jogo de estratégia consiste em disponibilizar um conjunto de elementos básicos e regras que dão o contexto, assim como as possibilidades e limite do mundo onde será criada a história. O processo de tomada de decisão neste tipo de jogo é algo constante e, através das ferramentas e elementos oferecidos no ambiente virtual, o jogador decide o que irá fazer e o que deve utilizar. Desta forma, o jogo apresenta uma condição de incerteza ou com informações incompletas, em que o planejamento é fundamental para obter sucesso.

1.1.3.2. Simuladores

Os simuladores são um tipo particular de jogo que tenta dar ao jogador exatamente a mesma experiência que ele teria em um ambiente real. São jogos de âmbito tático, com uma visão em primeira pessoa, sendo seu principal objetivo a imersão do usuário no ambiente simulado. Este tipo de jogo representa de forma fiel os fenômenos físicos e outras características retratadas no ambiente. Cita-se, como exemplo, os simuladores de voos usados para treinamento de pilotos.

1.1.3.3. Aventura

Os jogos de aventura combinam ações baseadas em raciocínio e reflexo. O objetivo do jogador é ultrapassar estágios que envolvem a solução de enigmas e quebra-cabeças para chegar ao final do jogo.

Para pertencer a esta categoria, os quebra-cabeças devem estar implícitos. Um jogo de aventura não pode ser visto apenas como um emaranhado de quebra-cabeças em que o usuário gasta horas para encontrar a solução.

Estes jogos têm uma lógica e conduzem os usuários por meio da investigação, podendo até contar com os mesmos mecanismos de investigação da vida real. Em geral, a trama do jogo sempre aponta para um caminho

limitante, onde o jogador, explorando o mundo virtual, permanece preso em um determinado local e precisa encontrar um caminho para sair ou terminar o jogo, utilizando alguns objetos, dicas ou mecanismos encontrados durante a investigação.

O nível de dificuldade e a ordem com que são apresentados os enigmas são um fator importante no enredo do jogo, pois estimulam o jogador a encontrar a saída ou simplesmente a parar de jogar. Neste contexto, é importante que os desenvolvedores dessa categoria de jogos fiquem atentos para o equilíbrio entre os fatores de investigação, quebra-cabeças e enigmas.

Os quebra-cabeças e os enigmas tendem a seguir um padrão menos variável sendo importante não quebrar o ritmo do jogo surpreendendo o jogador com quebra-cabeças que requeiram alto nível de concentração em meio a uma sequência de ação. As dificuldades devem ser apresentadas de forma a não produzir estímulos negativos para o jogador.

Regras que facilitam o equilíbrio entre quebra-cabeça e enigma:
1. Manter a consistência entre o ritmo e a jogabilidade do jogo.
2. Usar uma arquitetura de jogo apropriada.
3. Respeitar o universo do jogo.
4. Ajudar o jogador na solução dos enigmas.

A arquitetura do jogo de aventura é um fator determinante na sua produção. Uma arquitetura apropriada deve perseguir as seguintes características:

- Sempre disponibilizar os quebra-cabeças de nível fácil e agradável no início do roteiro, pois este processo estimula a imersão do jogador na aventura.
- Fazer com que o jogador encontre facilmente os quebra-cabeças: aqueles que são potenciais "becos sem saída" devem ser facilmente identificados. É importante não deixar o jogador perdido no jogo, sem saber qual é o próximo passo a ser realizado.
- Limitar a área de investigação ao redor do quebra-cabeça: todos os elementos necessários para sua solução, tais como dicas, objetos, plantas e outros elementos devem estar próximos a ele.
- Criar quebra-cabeças contornáveis ou reaproveitáveis: a solução de um enigma pode fornecer dicas para a solução de outro.
- Evitar dar muitas dicas para o jogador: o jogador não conhece o enredo do jogo como os desenvolvedores. Informações em excesso podem prejudicar, devendo ser evitadas.

- Manter uma aventura linear: planejar bem a ordem das descobertas necessárias para a solução de cada enigma, visando manter o jogador sempre interessado no jogo e, consequentemente, na investigação dos quebra-cabeças.

1.1.3.4. Passatempo

Os jogos de passatempo são simples, normalmente formados por uma superfície composta por um conjunto de peças móveis, apresentadas com interface em 2D ou 3D. Os quebra-cabeças apresentados são rápidos e não possuem nenhuma história relacionada. Seu objetivo essencial é atingir uma pontuação alta. As peças são organizadas de acordo com a disposição dos jogadores e a superfície representa um ambiente controlado que obedecem a determinadas regras e estratégias pré-definidas.

Os passatempos proporcionam desafios ao jogador e são constituídos de uma determinada lógica que requerem paciência e muita atenção para resolvê-los. Em geral, são jogos abstratos envolvendo algumas habilidades fundamentais, tais como a combinação de padrões e os agrupamentos de cores.

1.1.3.5. RPG – Jogos de Interpretação de Papéis

Os jogos de RPG (*Role Playing Game*) permitem que o jogador assuma o papel e conte histórias sobre a vida de um personagem, Os jogadores são coautores da história, o jogo estimula a criatividade, a participação em grupo, a troca de ideias e a amizade. Entretanto, ele centra-se mais no jogador que nos personagens, afinal é ele o contador de histórias. Os jogos de RPG em computadores seguem o mesmo objetivo do RPG convencional. A programação subjacente a esse tipo de jogo é complexa, principalmente devido à gigantesca base de elementos disponibilizados ao jogador para possibilitar diferentes caminhos para a trama e diferentes finais.

Os jogos de RPG para computadores podem ser classificados em jogos clássicos, jogos de múltiplos jogadores e os mundos virtuais persistentes ou mundos persistentes.

1.1.3.6. Educativos

Os jogos educativos podem conter uma ou mais características dos jogos citados acima. O que diferencia os jogos de educação dos demais que visam apenas diversão é que os educativos necessariamente levam em conta critérios didáticos e pedagógicos. Neste sentido, os jogos educacionais envolvem profissionais de diversas áreas de conhecimentos, como pedagogos, psicólogos,

educadores e os especialistas em ciência da computação, além do especialista na área a que o jogo se destina: línguas, história, geografia, etc.

O sucesso de um jogo de computador está associado à perfeita combinação de suas três partes básicas: enredo, motor e interface interativa.

O enredo define a trama do jogo, juntamente com os objetivos que o jogador deverá atingir em cada série de passos para cada nível do jogo até o seu final.

O motor é o sistema de controle do jogo. Ele controla a reação do jogo em função das ações do jogador. Diversos aspectos computacionais devem ser considerados na programação do motor, tais como a escolha apropriada da linguagem de programação em função de sua facilidade de uso e portabilidade, o desenvolvimento de algoritmos específicos e o tipo de interface com o usuário.

A interface interativa controla a comunicação entre o motor e o usuário, mostrando graficamente um novo estado do jogo. O desenvolvimento da interface envolve aspectos artísticos, cognitivos e técnicos. O valor artístico da interface valoriza a apresentação do jogo e propicia aos usuários maior satisfação ao jogar. O aspecto cognitivo está relacionado à correta interpretação gráfica pelo usuário, enquanto o aspecto técnico envolve desempenho, portabilidade e complexidade dos elementos gráficos.

1.1.4. Ferramenta para alunos e professores

Os programas de uso geral - como os de processamento de texto, planilhas, manipulação de arquivos, construção e transformação de gráficos; os de apresentação e, sobretudo os de busca e pesquisa na internet - são também ferramentas extremamente úteis tanto para o aluno quanto para o professor. Estas ferramentas se constituem atualmente como uma das fontes de mudança do ensino e do processo de manipular a informação, enquanto as outras modalidades de software educativo descritas anteriormente podem ser caracterizadas como uma tentativa de computadorizar o processo de ensino-aprendizagem tradicional.

De maneira mais ampla, podemos resumir a classificação dos softwares educacionais quanto à liberdade de atuação do professor no ambiente de aprendizagem, quanto ao nível de aprendizagem proporcionada para o aluno e quanto aos objetivos pedagógicos, resultando em uma configuração, como explicitada no quadro abaixo.

Quadro resumo da classificação dos softwares educativos

Classificação das características dos softwares educativos	Quanto à liberdade de criação para atuação dos professores	Aberto Fechado
	Quanto ao nível de aprendizagem do aluno	Sequencial Relacional Criativo
	Quanto aos objetivos pedagógicos	Tutoriais Aplicativos Programação Exercício e prática Multimídia Internet Simulação Jogos

A informática não pode se distanciar dos processos educativos, haja vista a abrangência e a permeabilidade desta ferramenta na sociedade. Apesar desta relevância, a utilização da informática na escola ainda está a passos lentos e ainda pairam dúvidas sobre a real contribuição dos computadores ao ensino. Ressalta-se a necessidade de preparação de professores para que eles tenham condições de utilizar os recursos computacionais da melhor forma possível, sendo este um fator impactante no desenvolvimento de softwares educativos de qualidade.

Atividades de avaliação

1. Pesquise sobre os softwares educacionais utilizados no seu município e classifique-os segundo o modelo pedagógico adotado.

Referências

RAMOS, E.M.F. (Org.). Informática na escola: um olhar multidisciplinar. Fortaleza: Editora UFC, 2003.

SEÁRA, E.F.R; BENITTI, F.B.V; RAABE, A.A.; SCHLINDWEIN, L.M. Da concepção à validação de um cenário virtual 3D para o Ensino Fundamental. In: WORKSHOP SOBRE INFORMÁTICA NA ESCOLA, 10., 2004, Salvador. Anais. Salvador: UFBA. 2004. (CD-ROM)

VALENTE, J.A. (1993a). Diferentes Usos do Computador na Educação. Em J.A. Valente (Org.), Computadores e Conhecimento: repensando a educação (pp.1-23). Campinas, SP: Gráfica da UNICAMP.

VALENTE, J.A. (1993b). Por Quê o Computador na Educação. Em J.A. Valente (Org.), Computadores e Conhecimento: repensando a educação (pp. 24-44). Campinas, SP: Gráfica da UNICAMP.

WENGER, E. (1987) Artificial Intelligence and Tutoring System: Computational and Cognitive Approaches to the Communication of Knowledge. Califórnia: Morgan Kaufmann Publishers.

Capítulo 2

Desenvolvimento de software educativo

Objetivo

- Vários são os modelos adequados ao desenvolvimento de softwares. Nesta unidade, apresentamos o ciclo de vida do software com as definições de cada etapa e um processo para desenvolvimento de software educacional fundamentado tanto nos conceitos computacionais quanto nos conceitos educacionais.

1. Desenvolvimento de software

As primeiras metodologias de desenvolvimento de software se estruturaram a partir de um modelo que ficou conhecido como ciclo de vida de desenvolvimento do software. O ciclo de vida de um software descreve as fases pelas quais o software passa desde a sua concepção até ficar sem uso algum. Importante não confundir ciclo de vida do software com ciclo de vida de desenvolvimento do software.

1.1. Ciclo de vida de um software

As fases do ciclo de vida de um software recebem várias denominações, segundo a ótica de seus autores. Em essência se referem a quatro fases que são delimitadas por eventos típicos em diversos ciclos de vida. Cada fase inclui um conjunto de atividades que devem ser realizadas pelos atores envolvidos. Essas fases são: definição, desenvolvimento, operação e retirada.

1.1.1. Fase de Definição

A fase de definição do software ocorre em conjunto com outras atividades como a modelagem de processos de negócios e análise de sistemas. Nesta etapa é importante o conhecimento da situação atual e a identificação de problemas para que as propostas de solução de sistemas computacionais elaboradas resolvam os problemas identificados. É importante nesta etapa realizar um estudo de viabilidade, incluindo análise custo-benefício, para se decidir qual solução será a escolhida.

Nesta fase, os profissionais envolvidos com a engenharia de software atuam objetivando identificar os requisitos de software e modelos de domínio que serão utilizados na fase de desenvolvimento. Os requisitos são também

fundamentais para que o engenheiro possa elaborar um plano de desenvolvimento de software, indicando em detalhes os recursos necessários, quer sejam os humanos e materiais, quer sejam os relativos a estimativas de prazos e custos, ou seja, ao cronograma de execução e o orçamento.

1.1.2. Fase de Desenvolvimento

A fase de desenvolvimento ou de produção do software inclui todas as atividades que têm por objetivo a construção do produto. Ela inclui principalmente o design, a programação e a verificação e validação do software.

1.1.2.1. Design

Abrange vários aspectos do design: conceitual, da interface de usuário, da arquitetura do software, dos algoritmos e estruturas de dados.

O design conceitual envolve a elaboração das ideias e conceitos básicos que determinam os elementos fundamentais do software em questão. O design conceitual exerce influência na interface de usuário e na arquitetura do software.

O design da interface de usuário envolve a elaboração da maneira como o jogador pode interagir para realizar suas tarefas. A escolha dos objetos de interfaces é essencial para garantir a boa usabilidade do software e é um dos fatores de sucesso do software.

O design de arquitetura de software deve elaborar uma visão macroscópica do software definindo como os componentes interagem entre si. O conceito de componente em arquitetura varia de acordo com a visão arquitetônica adotada.

O design de algoritmos e a estrutura de dados, também conhecidos como design detalhado, visam determinar, de maneira independente da linguagem de programação adotada, as soluções algorítmicas e as estruturas de dados associados.

1.1.2.2. Programação

A fase de programação envolve as atividades de codificação, compilação, integração e testes. A codificação visa traduzir o design num programa, utilizando linguagens e ferramentas adequadas. A codificação deve refletir a estrutura e o comportamento descrito no design. Os componentes arquiteturais devem ser codificados de forma independente e depois integrados. Os testes podem ser iniciados durante a fase de programação. A depuração de erros ocorre durante a programação utilizando algumas técnicas e ferramentas. É fundamental um controle e gerenciamento de versões para que se tenha um controle correto de tudo o que está sendo codificado.

1.1.2.3. Verificação e validação

Verificação e validação destinam-se a mostrar que o sistema está de acordo com o que foi pensado e especificado. A validação visa assegurar se o programa está fazendo aquilo que foi definido na sua especificação. A verificação visa inspecionar se o programa está correto, isto é, se não possui erros de execução.

1.1.3. Fase de Operação

Na fase de operação, estão envolvidas as atividades de distribuição e entrega, instalação e configuração, utilização, manutenção.

A distribuição e entrega pode ser feita tanto pelo desenvolvedor como nos casos do software personalizado, ou em um pacote a ser vendido em prateleiras de lojas ou ainda por outras formas ditadas pelas regras do comércio eletrônico e virtual.

O processo de instalação e configuração normalmente pode ser feito com a ajuda de software de instalação disponibilizados pelos fabricantes dos ambientes operacionais.

A atividade de utilização é o objeto do desenvolvimento do software. A qualidade da utilização é a usabilidade do software.

1.1.3.1. Fase de manutenção

A manutenção normalmente ocorre de duas formas: corretiva e evolutiva. A manutenção corretiva visa à resolução de problemas referentes à qualidade do software (falhas, baixo desempenho, baixa usabilidade, falta de confiabilidade, etc.). A manutenção evolutiva ou adaptativa visa a produção de novas versões do software de forma a atender a novos requisitos ou à adaptação de novas tecnologias.

1.1.4. Fase de retirada

A fase de retirada é aquela na qual se decide que o software, apesar de já possuir um excelente nível de confiabilidade e de correção, outras necessidades prioritárias estão em maior evidência fazendo com que se opte pela troca do software seja trocado devido a outras necessidades prioritárias que estão em maior evidência. Pode-se considerar também que manutenções adicionais não são mais eficazes em termos de custo. Algumas vezes, as mudanças propostas são tão drásticas que o projeto como um todo teria de ser modificado. Em tais casos, custa menos reprojetar e recodificar o produto inteiro. Seja qual for a razão, a versão corrente é substituída por uma nova versão e o processo de software continua.

Por outro lado, a verdadeira retirada do produto é um evento um tanto raro, que ocorre quando um produto houver suplantado sua utilidade. A organização do cliente não requer mais a funcionalidade fornecida pelo produto, e

ele finalmente é desinstalado dos computadores. Por ser uma fase que ocorre raramente, muitos autores não a citam como fase do ciclo de vida do software.

2. Desenvolvimento de Software Educacional

Como vimos, o processo de desenvolvimento de software é uma etapa do ciclo de vida do software. Importante ressaltar que o processo de desenvolvimento do software é também realizado por meio de ciclos de vida. Os modelos de ciclo de vida mais difundidos na literatura são: Ciclo de vida tradicional ou cascata, Espiral e Prototipagem.

Há certamente propriedades vantajosas em cada um desses modelos. De forma geral, recomenda-se adequar o modelo de ciclo de vida a ser adotado ao tipo de sistema que se objetiva desenvolver, considerando características vitais que possuam o modelo sistêmico, tais como: grau de complexidade dos algoritmos, relevância interativa do usuário na definição das regras de negócio, tecnologias disponíveis para arquitetura dos modelos, e outras relacionadas à metodologia de desenvolvimento.

Apresentamos nesta seção um processo para desenvolvimento de software educacional fundamentada tanto nos conceitos computacionais quanto nos conceitos educacionais. Vários são os modelos adequados ao desenvolvimento de software cada um com vantagens e desvantagens. A proposta aqui considerada se insere no modelo de desenvolvimento interativo e incremental.

2.1. Modelo de processo de desenvolvimento iterativo e incremental

O modelo interativo e incremental é uma extensão do modelo espiral sendo, porém, mais formal e rigoroso. Iterações são passos em fluxo de trabalho e incrementos significam crescimento do produto.

O princípio subjacente ao processo interativo e incremental é o de que os envolvidos trabalham refinando e ampliando paulatinamente a qualidade, o detalhe e o âmbito do sistema envolvido. A principal consequência da aproximação iterativa é que os produtos finais de todo o processo vão sendo amadurecidos e completados ao longo do tempo, mas cada iteração produz sempre um conjunto de produtos finais.

A cada iteração são realizadas as seguintes tarefas:

- Análise - refinamento de requisitos, refinamento do modelo conceitual.
- Projeto - refinamento do projeto arquitetural; projeto de baixo nível.
- Implementação - codificação e testes.
-Transição para produto - documentação, instalação.

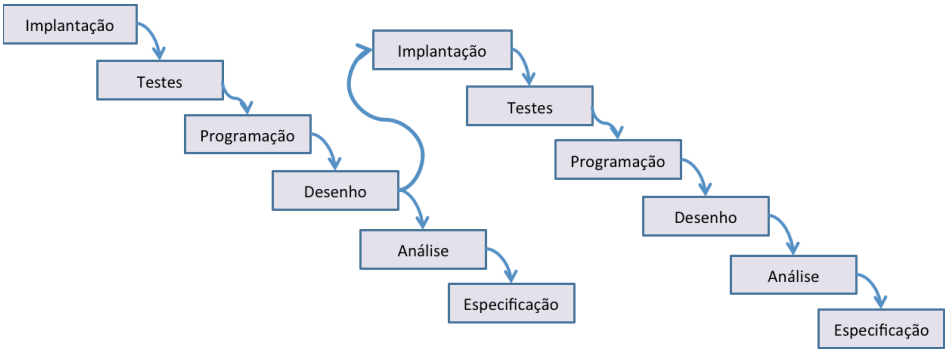


Figura x Esquema do modelo de desenvolvimento incremental e interativo

Vantagens do processo incremental e iterativo:

- Possibilidade de avaliar mais cedo os riscos e pontos críticos do projeto, e identificar medidas para eliminá-los ou controlá-los;
- Redução dos riscos envolvendo custos a um único incremento. Se a equipe que desenvolve o software precisar repetir a iteração, a organização perde somente o esforço mal direcionado de uma iteração, não o valor de um produto inteiro;
- Definição de uma arquitetura que melhor possa orientar todo o desenvolvimento;
- Disponibilização natural de um conjunto de regras para melhor controlar os inevitáveis pedidos de alterações futuras;
- Possibilidade de os vários intervenientes puderem trabalhar mais efetivamente pela interação e partilha de comunicação daí resultante.

No desenvolvimento do software educacional, estas etapas interativas são importantes porque permite a validação dos objetivos de aprendizagem para cada estágio do software. Recomenda-se que a validação e a decisão sobre o início da nova interação seja feita pela equipe que participou da concepção do software. Isto porque é na etapa de concepção do software educacional que se definem as diretrizes e os objetivos de aprendizagem que o software deve prover e são organizados em requisitos computacionais estagiados.

A primeira atividade da etapa de concepção do software concentra-se na definição dos objetivos de aprendizagem e requisitos do software, além de definir o escopo, o público-alvo e identificar a infraestrutura disponível na escola. Após a definição desta atividade, organizam-se em estágios, quando viáveis, os requisitos identificados para cada fase do processo.

Cada estágio deve ser um incremento do modelo e ser operacional, ou seja, deve possuir uma funcionalidade que lhes dê característica de produto,

o que na prática significa que possam ser utilizadas de forma independente. Como resultados da execução completa desta etapa são gerados os seguintes documentos:

- Planejamento geral do software educacional;
- Visão geral;
- Infraestrutura disponível;
- Características do público-alvo;
- Objetivos de aprendizado;
- Requisitos do software;
- Planejamento do Processo;
- Identificação dos estágios e divisão dos requisitos do software;
- Cronograma geral.

Atividades de avaliação

1. Na Engenharia de Software, há diversos modelos de ciclo de vida, definidos com variados níveis de formalidade. Pesquise três deles e descreva suas fases.
2. No caso do desenvolvimento de um jogo educacional, qual modelo de desenvolvimento você sugere? Justifique.

Referências

MAFFEO. Bruno. Engenharia de Software e Especificação de Sistemas, Ed. Campus. Rio de Janeiro.

PRESMAN, Rogers. Engenharia de Software Ed. Makron Books São Paulo – SP

SEÁRA, E.F.R; BENITTI, F.B.V; RAABE, A.A.; SCHLINDWEIN, L.M. Da concepção à validação de um cenário virtual 3D para o Ensino Fundamental. In: WORKSHOP SOBRE INFORMÁTICA NA ESCOLA, 10., 2004, Salvador. Anais. Salvador: UFBA. 2004. (CD-ROM)

Capítulo 3

Engenharia de software educativo

Objetivo

- Apresentar os principais conceitos sobre engenharia de software e abordar uma proposta de elicitação de requisitos para o software educativo.

1. Engenharia de Software

A Engenharia do Software (ES) é uma área do conhecimento da informática voltada para a especificação, o desenvolvimento e a manutenção de sistemas de softwares aplicando tecnologias e práticas de ciência da computação, gerência de projetos e outras disciplinas, objetivando organização, produtividade e qualidade. Neste sentido, a engenharia de softwares, como disciplina, tem por objetivo a compreensão e o controle da complexidade inerente ao processo de desenvolvimento de softwares e seu gerenciamento.

A ES é uma disciplina em constante evolução, com orientações que mudam segundo as exigências associadas à sua função. Ela surgiu em meados dos anos 70 numa tentativa de contornar a crise do software e dar um tratamento de engenharia, basicamente a sistematização e o controle, ao desenvolvimento de softwares complexos.

Na época, o que se denominou de "Crise do Software" foi a constatação no desenvolvimento do software, dentre outros, dos seguintes fatores:

- 25% dos projetos de software eram cancelados.
- O tempo de desenvolvimento era maior que o tempo estimado.
- 75% dos sistemas não funcionavam como planejados.
- A manutenção e a utilização eram difíceis.

Os problemas eram proporcionais à complexidade do software. Entre as razões para a Crise do Software, citavam-se:

- Falta de envolvimento do usuário.
- Análise e projetos inadequados.
- Falta de flexibilidade no projeto.
- Prazos longos.
- Elevada rotatividade do pessoal envolvido no desenvolvimento.

- Má qualidade dos métodos, linguagens, ferramentas e processos.
- Velocidade de mudança tecnológica.
- Dificuldade de formalização.

Atualmente, as tecnologias e práticas aplicadas na engenharia de software englobam linguagens de programação, base de dados, ferramentas, plataformas, bibliotecas padrões, processos e a questão da qualidade do software.

Os fundamentos científicos para a engenharia do software envolvem o uso de modelo abstratos e precisos que permitem ao engenheiro especificar, projetar, programar e manter sistemas de softwares, avaliando e garantindo suas qualidades. Além disso, a engenharia de software deve oferecer mecanismos para se planejar e gerenciar o processo de desenvolvimento de um sistema de informação.

Neste contexto, a Engenharia de Software é a aplicação de conhecimento científico para criação e manutenção de um software, o que envolve a aplicação de processos, métodos e ferramentas.

Os processos se constituem na ligação entre os métodos e as ferramentas, garantindo que haja uma sequência em que os métodos sejam aplicados e que os produtos que se exigem sejam entregues, ou ainda estabelecendo controles que ajudem a assegurar a qualidade e coordenar as alterações de requisitos e marcos de referência que possibilitam administrar o progresso do software.

> Processo de software, ou processo de engenharia de software, é uma sequência coerente de práticas, que objetiva o desenvolvimento ou evolução de sistemas de software. Estas práticas englobam as atividades de especificação, projeto, programação, testes e caracterizam-se pela interação de ferramentas, pessoas e métodos.

Os métodos proporcionam os detalhes de como fazer para construir o software, seja com relação ao planejamento e à estimativa de projeto, ou à análise de requisitos de software e de sistemas, ou ao projeto da estrutura de dados, entre outros. Por fim, as ferramentas dão o suporte automatizado aos métodos.

A Engenharia de Software tem como objetivos finais: melhorar a qualidade de produtos de software, incrementar a produtividade do pessoal técnico e aumentar a satisfação do cliente.

Atividades de avaliação

1. O que é Engenharia de Software?
2. Qual a diferença entre Engenharia de Software e Ciência da Computação?
3. Qual a diferença entre Engenharia de Software e Engenharia de Sistemas?
4. Quais são os custos da Engenharia de Software?
5. O que é processo de software?

2. Engenharia de Software Educacional (ESE)

No campo do software educativo, a Engenharia de Software tem enfrentado um de seus maiores desafios. Produzir um software educativo não se limita a um bom gerenciamento de projeto aliado a uma boa ideia de docentes aguardando o apoio da tecnologia da informática. A análise de um sistema computacional com finalidades educacionais enfrenta o desafio de incorporar os mecanismos pedagógicos e didáticos que constituem a base de todo instrumento de ensino e de aprendizagem.

Neste contexto, podemos elencar como elementos que fazem parte do complexo campo de saber da engenharia de software educativo:

- A adequada compreensão do problema que gera a necessidade do software educativo;
- A correta especificação do cenário demandante;
- A análise de possibilidades;
- A adequada arquitetura das informações disponíveis;
- O cumprimento de prazos e orçamentos;
- O levantamento claro e inequívoco de requisitos;
- Recursos humanos qualificados
- Seleção das ferramentas computacionais apropriadas.

Como se vê, não se trata de um processo banal.

O processo de desenvolvimento de softwares educativos exige especificidades que os distanciam sobremaneira dos procedimentos aplicados ao desenvolvimento de softwares comerciais, bancários ou domésticos.

Um sistema educativo, por mais simples que seja, traduz e delimita conhecimentos em processo dinâmico de comunicação e percepção, impli-

cando um olhar para um conjunto de aspectos subjetivos que caracterizam o fenômeno educativo.

Bem diferente dos outros tipos de sistema, classificados como sistemas fechados, nos quais usuários e proprietários de sistemas e subsistemas interagem entre si através de procedimentos pré-estabelecidos, previsíveis e perfeitamente traduzíveis em operações automáticas e informatizáveis.

Do ponto de vista estrito da engenharia de software, um software educativo - apesar de apresentar características e princípios que os diferenciam de outros sistemas computacionais - poderia ser considerado como software comum. Contudo não se pode prescindir da compreensão de que um software educativo é um conjunto de artefatos criados para funcionar como mediadores em atividades educativas de formação em distintas áreas do conhecimento, o que significa dizer que o software educativo é uma interface que atua em meio à negociação de significados e à construção de conhecimento específico em contextos específicos.

A função de um software educativo é promover aprendizagem para o uso, mas também analisar a aprendizagem, essência de seu objeto, que ocorre com o uso do software. Portanto, não se trata apenas de aprender a FAZER algo a partir do software, mas de aprender a lidar com o software para APRENDER algum conceito. Em outras palavras, os softwares educativos contemplam seus requisitos quando os usuários aprendem a usá-los e ao fazê-lo aprendem algo ou algum conceito. Os softwares educativos interferem, portanto, no desenvolvimento cognitivo dos usuários, tendo impacto na aprendizagem de campos conceituais determinados.

A natureza dos conhecimentos a serem veiculados e as estratégias de abordagem estão fortemente associadas às dificuldades encontradas na produção de software educativo. No processo de concepção do software, há uma diferença significativa entre as representações que designers, programadores e professores têm acerca dos processos de ensino e aprendizagem.

O problema central da Engenharia de Software Educativo (ESE) está associado aos elementos que o designer dispõe e não é um problema do tipo "o programador faz o que o educador especifica". A interação entre programadores e educadores é problemática devido às dificuldades em compartilhar conceitos das diferentes áreas.

Esta realidade parece implicar também na necessidade de um modelo de desenvolvimento que contemple a construção de mecanismos de interação adequados à proposta instrucional. Afinal, quando um software educativo é desenvolvido entende-se que uma das etapas no seu desenvolvimento é definir a concepção pedagógica daqueles que estão envolvidos na sua modelagem, pois o tipo de uso a que se destina reflete a concepção pedagógica do software.

> Quando um software é utilizado para fins educacionais, invariavelmente ele, ou o uso que se faz dele, reflete um dos paradigmas educacionais: comportamentalista ou construtivista.

Quadro 1 – Relação entre os paradigmas educacionais, suas características e algumas modalidades de Software Educacional.

Paradigma Educacional	Comportamentalista	Construtivista
Visão da natureza humana	Empirista e racionalista	Interacionalista
Quanto à atividade do aprendiz.	Algorítmico	Heurístico
Quanto ao direcionamento na utilização do software	Dura	Branda
Modalidades de Software Educacional	Tutoriais, Exercício e prática	Simulação, Jogos

Com relação à atividade do aprendiz, um software pode ser algorítmico ou heurístico. No software algorítmico, predomina a ênfase na transmissão de conhecimentos do sujeito que sabe para o sujeito que deseja aprender, sendo função do desenvolvedor do software projetar uma sequência bem planejada de atividades que conduzam o aluno ao conhecimento desejado. No software de concepção heurística, predomina a aprendizagem experimental ou por descobrimento, o que implica na criação de um ambiente rico em situações que o aluno deve explorar.

Quanto ao direcionamento na utilização do software, podem ser consideradas duas abordagens: dura e branda. Na abordagem dura, os planos são previamente traçados para uso do computador e as atividades dos alunos resumem-se a responder a perguntas apresentadas, registrando-se e contabilizando-se erros e acertos. Na abordagem branda a atividade e interação com o computador não parecem ter um objetivo definido, fazendo com que o aluno esteja no comando, fazendo uma série de atividades consideradas interessantes por ele, onde há desafio. Os erros são fontes de reflexão e desenvolvimento de novos projetos.

Uma questão relevante envolvendo o desenvolvimento de aplicações educativas está na especificidade dos requisitos não funcionais de usuários, que, muitas vezes, parecem pouco integralizados neste desenvolvimento.

O conjunto de requisitos deve observar não só aspectos do processo de aprendizagem dos alunos, mas também aspectos do processo de mediação a ser promovida pelo professor, o qual pode beneficiar-se de funcionalidades específicas do sistema, como o registro de passos ou a prévia organização de sequência de problemas. Da mesma maneira, funcionalidades - muito rígidas ou pouco adaptativas - podem tornar um ambiente por demais diretivo e inadequado aos professores e a suas práticas.

2.1. Elicitação de Requisitos de Software Educativo

Elicitação de Requisitos é o nome dado às atividades envolvidas em descobrir qual o problema a ser resolvido pelo sistema, quais serviços devem ser oferecidos e qual o desempenho desejado. No desenvolvimento de software educativo, até mesmo técnicas sistemáticas apresentam problema em sua execução.

Como destacamos, dada a complexidade da natureza do objeto aprendizagem na educação, esta fase vai além das perguntas sobre o que o sistema deve fazer. Neste tipo de sistemas, há uma grande variedade de tipos de requisitos a serem identificados, relacionados ao domínio (aprendizagem de conceitos) e ao contexto de uso (atividade).

A modelagem cognitiva da ação permite aceder a informações sobre o domínio, sua aprendizagem. Já a modelagem cognitiva de atividades permite identificar requisitos relacionados às práticas sociais nas quais participam os usuários com artefatos similares àqueles em desenvolvimento. A análise de ambos ocorre mediante orientações de modelos e abordagens construtivistas.

Diante dessa complexidade e do número de pessoas e saberes envolvidos no desenvolvimento, um processo para guiar a elicitação dos requisitos torna-se crucial para o sucesso do projeto. Nesta perspectiva, diferentes modelos de elicitação de requisitos foram propostos por estudiosos que vivenciam a problemática.

Apresentamos a seguir o modelo de Elicitação de Requisitos para software educacional de Gomes e Wanderley (2003).

2.1.1. O Modelo de Elicitação de Requisitos de Gomes e Wandeley

Neste modelo deve-se ter clareza na distinção dos requisitos relacionados ao domínio e os requisitos relacionados à atividade.

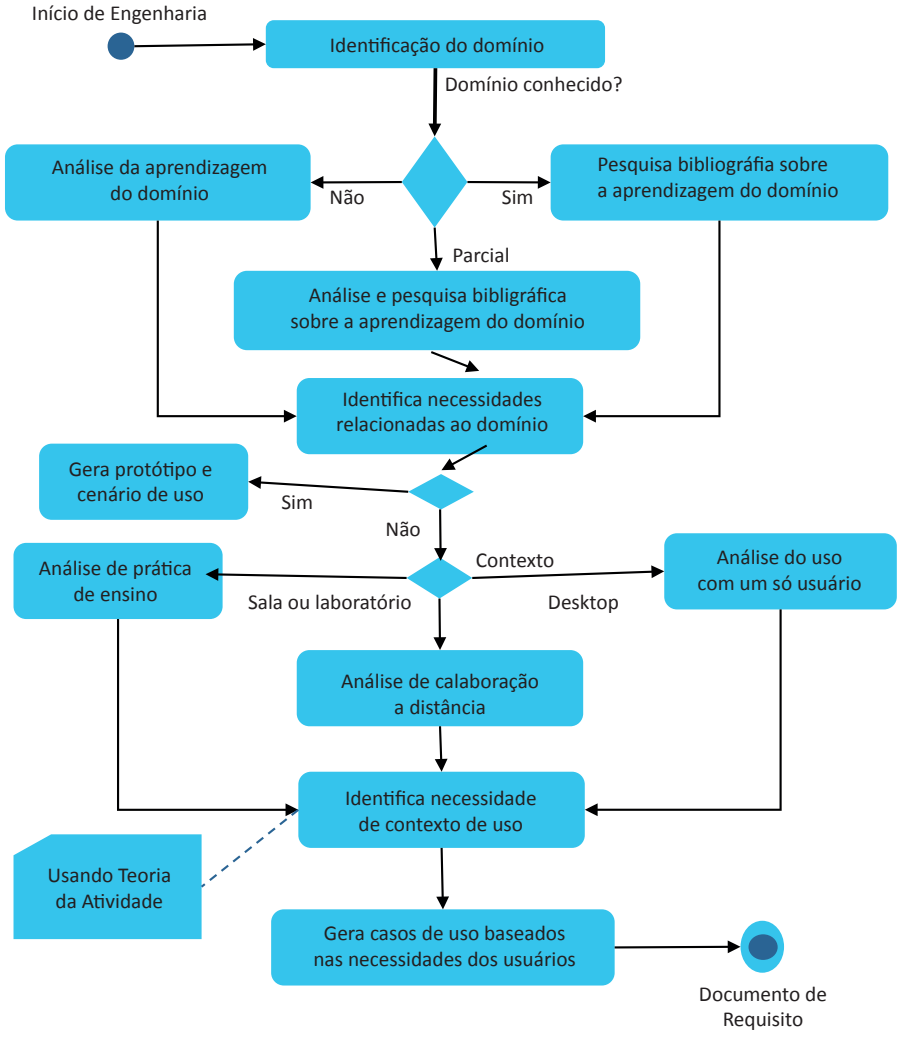
O domínio é definido como a aprendizagem dos conceitos e suas informações são obtidas através da modelagem cognitiva da ação.

A atividade é a contextualização de como o sistema é usado. A modelagem cognitiva das atividades identifica requisitos relacionados às interações sociais das quais o público-alvo participa.

Os autores propuseram um fluxo de atividades para o desenvolvimento de softwares educativos que pode ser aplicado a qualquer tipo ou nível de software educativo pretendido.

O processo é realizado através de perguntas, que são feitas durante o processo e servem para aplicá-lo a projetos mais específicos.

O fluxo de atividades está apresentado na figura abaixo. Em seguida, faremos uma breve explicação de cada atividade do fluxo que deve ser seguido para realizar a elicitação de requisitos com essa proposta.



Proposta de um modelo de processo de criação de software educativo genérico (Gomes e Vanderley 2003)

1. **Identificação do Domínio –** Entende-se por domínio o campo conceitual a ser apresentado no sistema como estatística, português, linguagem de programação. E, claro, a identificação do domínio deve se dar desde o início do projeto.

   No âmbito do domínio, os requisitos se relacionam com a interface e com o que ela deve prover em termos de aprendizagem. Em primeiro lugar, são as interfaces que servem para veicular situação e viabilizar contextos nos

quais as ações dos usuários são possíveis. Em segundo lugar, são também as interfaces que servem de mediador para os profissionais de ensino na tarefa de ensinar.

2. **Pesquisa bibliográfica sobre a aprendizagem do domínio –** No caso de já haver conhecimento sobre o domínio é fundamental compreender o processo da aprendizagem desse domínio. São importantes tanto a construção de uma bibliografia de conceitos com fonte de informações mais simples como a inclusão, na equipe multidisciplinar, de especialista de aprendizagem e desenvolvimento cognitivo.

3. **Análise da aprendizagem do domínio –** Quando a aprendizagem do domínio não está tão bem descrita na literatura, a análise da aprendizagem do domínio realizada com auxílio da observação sistemática permite identificar os requisitos associados ao domínio. A equipe de design pode também criar experimentos a partir dos quais possam ser observados os efeitos do uso do sistema em desenvolvimento, sobretudo na aprendizagem de conceitos específicos. É importante também realizar análises através de estudos empíricos, uma vez que as informações desejadas podem estar implícitas na interação dos usuários.

4. **Identificar necessidades relacionadas ao domínio –** A etapa de identificar as necessidades dos usuários a serem atendidas pelo sistema é realizada com os resultados obtidos na pesquisa bibliográfica e/ou com estudos empíricos relacionados ao domínio.

5. **Geração de protótipos e análise cenários de uso –** Neste modelo, a atividade de gerar protótipos não é obrigatória. Entretanto a geração de protótipos é importante porque permite que os usuários testem as interfaces do sistema antes que seja programado, podendo assim validar os requisitos ou refiná-los.

6. **Análise do uso do usuário aluno –** Nesta atividade são usadas técnicas de etnografia rápida para estudar a interação do aluno com a máquina no contexto de uso da aplicação. O resultado da pesquisa etnográfica orienta o projeto do software.

> Etnografia: é particularmente eficaz na descoberta de dois tipos de requisitos: a) aqueles derivados da maneira como as pessoas realmente trabalham, em vez da maneira pelas quais as definições de processo dizem como elas deveriam trabalhar; b) os derivados da cooperação e conscientização das atividades de outras pessoas.

7. **Análise da prática de ensino –** Nesta etapa, é importante compreender as interrelações no ambiente de aprendizagem, observando a interação solitária aluno-máquina e a ambientação em um laboratório de informática educativa. Para os autores deste modelo, é importante observar a atividade normal de um professor num laboratório de informática educativa para, a partir dessa observação, gerar requisitos que promovam o design de ferramentas adequadas.

8. **Análise da colaboração a distância –** Esta etapa é recomendada para descobrir requisitos de sistemas de Educação a Distância (EaD). A análise da colaboração é realizada em estudos sistemáticos envolvendo a observação como também pode fazer uso da análise pela teoria da atividade.

9. **Identificar necessidades do contexto de uso –** Após o estudo do contexto, é possível identificar os requisitos associados a ele. Essas informações devem ser organizadas de modo que os designers possam entender e traduzir as necessidades de contexto e de domínio para casos de uso.

10. **Gerar casos de uso baseado nas necessidades do usuário –** Casos de uso são artefatos importantes que ajudam designers a ganhar uma visão coerente do produto. Nos casos de uso, definem-se atores e se detalham os serviços que serão programados no sistema, de modo que todos os requisitos sejam atendidos por um ou mais casos de uso. Para os autores deste modelo, as exigências do usuário descrevem como um futuro produto pode ajudá-lo a alcançar suas metas efetiva e eficientemente, e com satisfação em seu contexto de uso.

    Como podemos perceber, as questões relacionadas à Engenharia de Requisitos de Software é extremamente crítica no desenvolvimento de sistemas educativos, pois existe o desafio de prover a aprendizagem do usuário final em um determinado domínio.

    Fica evidente que os métodos tradicionais de elicitação, como entrevistas e questionários, não são suficientes para se capturar as necessidades dos usuários desse tipo de sistema.

    Entender como um indivíduo aprende requer uma análise da sua interação com a ferramenta de aprendizado e, portanto, exige a aplicação de métodos empíricos para captar o chamado conhecimento intangível.

    A preocupação com o entendimento do contexto onde o software será usado e a necessidade de uma equipe multidisciplinar, com especialistas em educação e no domínio abordado pelo software educativo, é outro ponto que o difere dos sistemas convencionais.

A pouca interação entre os profissionais da área de Informática e de Educação ainda é uma barreira a ser transposta e ainda há muitas lacunas na literatura acerca dos requisitos para software educativo.

Atividades de avaliação

1. Que diferenças podem ser enumeradas para o desenvolvimento de software educativo e o desenvolvimento de um software comercial?
2. Que outros modelos de elicitação de requisitos podem ser adequados para o levantamento de requisitos do software educativo?
3. Pesquise com quais dificuldades os desenvolvedores se deparam no desenvolvimento de software educativo. Por que isso ocorre?

Referências

O'BRIEN, J. & MARAKAS, G. (2005) "*Introduction to Information Systems*", 13a. ed., McGraw-Hill/Irwin.

SANTO, R. E. (2009) "Portal EduES Brasil: Um Ambiente de Apoio à Pesquisa Experimental em Educação em Engenharia de Software no Brasil". Monografia de Projeto Final, IM-DCC, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brasil, 88p.

SANTOS, R. P.; SANTOS, P. S. M.; SANTO, R. E. (2010) "Projeto e Desenvolvimento de Sistemas de Informação para Gestão de Conteúdo na Web utilizando Java EE e JBoss Seam". In: Anais do VI Simpósio Brasileiro de Sistemas de Informação, Minicursos, Marabá, PA, Brasil, pp. 4.4-1 - 4.4-6.

TRAVASSOS, G. H.; GUROV, D.; AMARAL, E. A. G. (2002) "Introdução à Engenharia de Software Experimental". Relatório Técnico ES-590/02. Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

WERNER, C. M. L.; RODRIGUES, C. S. C.; SANTOS, R. P.; COSTA, H. L. C.; SANTO, R. E.; CASTRO, W. S. (2009) "Projeto Tec3ES: Tecnologias e Estratégias para Educação em Engenharia de Software". In: Proceedings of the 17th Iberian-American Conference on High Education in Computer Science (CIESC), XXXV Latin American Informatics Conference (CLEI), Pelotas, RS, Brasil, pp. 1-2.

Capítulo 4

Objetivos de Aprendizagem

Objetivo

- A tecnologia da reutilização evoluiu e chegou à educação sob a forma de Objetos de Aprendizagem (OAs). Neste capítulo, apresentamos conceitos e metodologias de desenvolvimento de OAs, tendo em vista que um OA pode ser considerado um produto de software.

1. Reutilização de Software

As primeiras ideias sobre reutilização de software são do ano de 1968, quando Doug McIlroy vislumbrou a possibilidade de que os softwares pudessem funcionar como "Circuitos Integrados" (CI) e fossem fabricados em larga escala a partir de padrões. A visão de McIlroy era baseada na indústria de componentes eletrônicos que podiam ser selecionados em catálogos de diferentes fabricantes, apresentando propriedades configuráveis e diferentes níveis de confiabilidade.

A partir de 1980, o aumento da complexidade dos sistemas e as necessidades do mercado forçaram as empresas produtoras de software e as universidades a acatarem a ideia da tecnologia de reuso.

A vantagem da reutilização de software está no potencial que ela exibe em relação à qualidade, ao custo e à rapidez no desenvolvimento. No entanto, para que esse potencial se torne realidade o reuso deve ser estendido para além do código das aplicações, ou seja, deve abranger os requisitos, especificações, projetos, testes e todos os outros elementos produzidos durante as fases de desenvolvimento.

De início, a reutilização de software foi uma prática sem nenhuma padronização e realizada basicamente por desenvolvedores que tinham a responsabilidade de fornecer e recuperar possíveis artefatos ao longo dos projetos. Essa forma de trabalho não era adequada porque as pessoas perdiam muito tempo para procurar e entender o funcionamento de cada pedaço candidato e, ainda assim, corriam o risco de introduzir novos erros no sistema.

Vários problemas podem ser citados, entre eles a dificuldade dos próprios desenvolvedores de reusar e confiar no trabalho feito por terceiros, mesmo aqueles adeptos que construíam software reutilizável. Essa resistência para reutilizar software desenvolvido por terceiros ficou conhecida como a Síndrome do "Não-Inventado-Aqui" (*Not-Invented-Here Syndrome,* em inglês).

As tecnologias orientadas a objetos constituiu um marco para a tecnologia de reuso de software. Isso porque elas conseguiram ampliar esse escopo e permitiram o reaproveitamento de classes de análise, projeto e implementação em diferentes projetos de software. Assim, as antigas bibliotecas de funções deram lugar ao desenvolvimento de bibliotecas de classes, que modelam com mais coesão as regras de negócio dos sistemas. Na sequência evolutiva, novos passos foram dados com a criação de frameworks reutilizáveis.

> Frameworks - estruturas de classes montadas e preparadas para um conjunto específico de problemas.

A computação distribuída de objetos - também chamada de *Distributed Object Computing* (DOC) - causou forte influência sobre a reutilização de software, pois técnicas de orientação a objetos são aplicadas com a finalidade de se distribuir serviços reutilizáveis e aplicações de forma eficiente, flexível e robusta em ambientes diferentes e normalmente heterogêneos.

Essa iniciativa surgiu quando o *Object Management Group* (OMG) elaborou uma especificação aberta sobre mediadores (middleware) para a distribuição de sistemas e desenvolveu o Common Object Request Broker Architecture (CORBA).

O objetivo à época era permitir que diferentes empresas pudessem implementar a especificação e oferecer mediadores reutilizáveis para apoiar a distribuição de objetos em aplicações de software. Com isso, elas poderiam se comunicar pela rede sem se preocupar com a localização dos objetos, linguagens de programação, sistemas operacionais, protocolos de comunicação e plataformas de hardware.

> Object Management Group – Formado em 1989, o OMG (www.omg.org) é um consórcio internacional sem fins lucrativos da indústria de software com o propósito de criar padrões para a arquitetura de componentes.

O advento da orientação a objetos e da computação distribuída foi decisivo para a definição do conceito de componente, que até então era tido como qualquer artefato de software que pudesse ser reutilizado. A introdução de interfaces bem definidas somadas ao conceito de encapsulamento trouxe a ideia do componente para uma realidade mais próxima daquela vista em componentes de outras indústrias.

Em 1991, a invenção dos controles OLE (Object Linking and Embedding) pela Microsoft desencadeou o início de uma das mais importantes tecnologias conhecidas para o desenvolvimento de software baseado em componentes:

o COM (Component Object Model). Inicialmente, esses controles OLE apresentaram limitações quanto à dificuldade de programação e robustez que forçaram a Microsoft a buscar novos caminhos.

A subsequente criação de controles ActiveX baseados no modelo COM foram mais bem sucedidos e contribuíram para o avanço da plataforma Windows e suas linguagens de programação RAD (Desenvolvimento Rápido de Aplicativos), como o Visual Basic e o Delphi.

O surgimento de novas necessidades do mercado fez a Microsoft estender o seu modelo e criar uma família de soluções que incluem o COM Distribuído (DCOM) e o COM+ [COM2006]. Essas soluções acrescentam suporte a serviços como distribuição de processamento pela rede (DCOM), transações, segurança, eventos assíncronos, filas de mensagens, etc.

A arquitetura de componentes também sofreu influência da plataforma Java. Em 1996, a Sun Microsystems lançou a especificação JavaBeans voltada para o desenvolvimento padronizado de componentes reutilizáveis.

Alguns anos mais tarde, em 1999, a evolução da tecnologia de componentes atingiu um novo estágio com o lançamento do Enterprise JavaBeans (EJB), um modelo de componentes focado no desenvolvimento e implantação de aplicações orientadas a negócio. Dentre as contribuições dessa tecnologia, citamos a possibilidade de desenvolver aplicações distribuídas, robustas e reutilizáveis, podendo, ainda, aproveitar o suporte ao gerenciamento de transações e multithreading.

Atividades de avaliação

1. A evolução tecnológica apresentada provocou mudanças substanciais no desenvolvimento de software nas últimas décadas exigindo, além de outras coisas, a reformulação das metodologias e processos de reutilização de software.
2. Pesquise sobre quais metodologias foram criadas para orientar o desenvolvimento dos componentes nas diferentes fases do ciclo de vida dos sistemas.
3. Cite pelo menos dois exemplos e especifique qual o seu papel nos processos de desenvolvimento de componentes reutilizáveis.

1.1. Objeto de aprendizagem

Inovações tecnológicas podem resultar em mudanças inteiras de paradigma. A rede de computadores conhecida como Internet é uma destas inovações.

Depois de revolucionar o modo como as pessoas se comunicam e fazem negócios, a Internet passa a provocar uma mudança de paradigma no modo como as pessoas aprendem, além de impulsionar fortemente a Educação a Distância (EaD) e a formação de redes virtuais de aprendizagem colaborativa. Em decorrência disso, mudanças importantes estão surgindo na maneira como os materiais educacionais são projetados, desenvolvidos e apresentados para aqueles que desejam aprender.

Neste contexto, surgiu o conceito de Objetos de Aprendizagem, que são entidades digitais projetadas com um determinado objetivo pedagógico e que podem ser reutilizadas em outros contextos para apoiar a aprendizagem.

Os Objetos de Aprendizagem são elementos de um novo tipo de instrução baseada no paradigma da orientação a objetos da ciência da computação. A orientação a objeto explora a criação de componentes, chamados objetos, que possam ser reusados em contextos múltiplos. Assim, a ideia subjacente aos objetos de aprendizagem é que os projetistas instrucionais podem construir componentes instrucionais pequenos, relativamente ao tamanho de um curso inteiro, que podem ser reusados diversas vezes em diferentes contextos de aprendizagem.

Importante ressaltar que os Objetos de Aprendizagem estão geralmente compreendidos como entidades digitais acessíveis via Internet, significando que um grande número de pessoas pode acessá-los e usá-los simultaneamente, diferentemente da mídia instrucional tradicional, como o projetor ou a fita de vídeo, que só podem existir em um lugar de cada vez.

Observe que os Objetos de Aprendizagem carregam consigo o conceito de reuso, portanto, para que um AO seja eficaz e possa ser reutilizado por outras pessoas como parte de uma atividade ou curso, ele deve ser desenvolvido segundo alguns critérios tecnológicos, pedagógicos e educacionais.

Nesta perspectiva, o Departamento de Defesa dos Estados Unidos e o governo americano, por meio da Advanced Distributed Learning (ADL), criaram o Sharable Courseware Object Reference Model (SCORM). O SCORM é uma especificação que utiliza metadados para lidar com aspectos de gerência e conteúdo de aprendizagem, facilitando o desenvolvimento de conteúdo reutilizável e permitindo a criação de uma biblioteca de objetos de aprendizagem que, por sua vez, oferece o recurso de recuperação desses objetos.

1.1.1. Objetos de Aprendizagem e o padrão SCORM

Quando se fala em desenvolvimento de Objetos de Aprendizagem (OAs), é importante considerar dois aspectos: o objetivo pedagógico e a tecnologia a ser utilizada. O objetivo pedagógico define o resultado esperado do aprendizado após a utilização e interação com o Objeto de Aprendizagem. O conteúdo de um OA deve ser independente de outros OAs, premissa básica para que seja reutilizável.

A tecnologia utilizada para desenvolvimento de ensino a distância deve considerar a utilização de padrões internacionalmente conhecidos e aceitos. Observar essa premissa é importante para que qualquer pessoa possa ter acesso à informação e para que a reutilização possa ocorrer com mais facilidade. Também devem ser consideradas regras de acessibilidade e navegabilidade, sendo importante utilizar uma padronização para os objetos a serem desenvolvidos, evitando gerar dificuldades para os alunos ao terem que, durante um curso, interagir com objetos muito diversificados como menus diferentes e mensagens de erro em formatos variados.

O SCORM reúne um conjunto de padrões e especificações relacionadas aos fundamentos técnicos para criação e utilização de OAs voltados para o ensino a distância. Ele é composto pelos seguintes modelos:

- Modelo de Agregação de Conteúdo (*Content Aggregation Model - CAM*);
- Ambiente de Execução (*Run-Time Environment* - RTE) para objetos educacionais baseados na Web e
- Modelo de Sequenciamento e Navegação (*Sequencing and Navigation* - SN) para apresentação dinâmica de conteúdo baseada na necessidade do aprendiz.

O CAM descreve os componentes de um pacote de conteúdo. Define as responsabilidades e os requisitos para a agregação de componentes, como cursos, lições e módulos. Traz definições para a geração de pacotes de conteúdo, para a elaboração de metadados sobre os conteúdos e para a inserção de detalhes de sequência e navegação no contexto do pacote.

O RTE descreve como objetos de conteúdo são iniciados, permitindo a interoperabilidade entre os objetos de conteúdo e Learning Management Systems (LMS) que utilizem o padrão SCORM.

O SN define métodos para representar o comportamento pretendido em uma experiência de aprendizado, de forma que qualquer LMS que utilize o padrão SCORM dará sequência às atividades de aprendizagem de forma consistente.

1.1.2. Etapas do desenvolvimento de Objetos de Aprendizagem

O desenvolvimento de um Objeto de Aprendizagem como produto de software exige uma metodologia de trabalho fundamentada em processos estruturados. O modelo ADDIE utiliza cinco estágios da engenharia de software para o desenvolvimento de OAs.

O modelo ADDIE desenvolveu-se com o ISD – *Instrucional Systems Design*. O ISD, assim como o ISDD (Instrucional Systems Design and Development), ou o SAT (*Systems Approach to Training*) ou ainda o ID (*Instrucional Design*), constituem-se em uma metodologia que propõe um processo desenvolvido em basicamente três etapas interrelacionadas: conhecimento do público-alvo (identificação das necessidades), proposta de solução para estas necessidades (desenho da solução) e avaliação dos resultados.

O ISD teve origem no período da Segunda Guerra Mundial, a partir da necessidade dos Estados Unidos melhorarem seus treinamentos. A partir daí, o ISD tornou-se a metodologia mais popular para o desenvolvimento de programas de treinamento.

Muitos são os modelos de ISD, mas a maioria é baseada no ADDIE Model, ou melhor:

A n a l y s i s – Análise

D e s i g n – Desenho

D e v e l o p m e n t – Desenvolvimento

I m p l e m e n t a t i o n – Implementação

E v a l u a t i o n – Avaliação

No modelo ADDIE, as etapas são dependentes entre si, pois cada uma alimenta a seguinte e, caso a etapa anterior não esteja definitivamente concluída, as outras ficam seriamente comprometidas.

Na fase de Análise, são determinados os pré-requisitos e também se analisa a reusabilidade do objeto, o cenário tecnológico, as mídias convenientes e os envolvidos no projeto. A primeira fase é considerada a mais importante. Nela todas as necessidades do público-alvo precisam ser claramente compreendidas. É nesta fase que se levanta o problema que será solucionado. São perguntas típicas desta fase: Quais as necessidades deste grupo? Do que eles precisam? Por que é importante desenvolver isso? O que precisam saber para o problema ser resolvido?

Essas perguntas, quando bem respondidas, transformam-se em objetivos, sendo importante identificar expectativas, anseios e objetivos profissionais, para aproximar ainda mais o "onde estou" com o "onde quero chegar". É importante também identificar o que já se sabe e o que é preciso ser eviden-

ciado. E, a partir daí, é preciso ter certeza de tudo o que precisa ser visto para alcançar o objetivo traçado.

Na fase de desenho, é fundamental definir os objetivos de aprendizagem, pois deles dependem os procedimentos de ensino e as formas de avaliação, que também devem ser pensadas neste momento. Nesta fase deve ocorrer a elaboração do conteúdo ou a distribuição dele – sem perder de vista a hierarquia de conceitos, suas relações e sua granularidade, desenho instrucional do conteúdo, elaboração de exercícios e construção da avaliação, traduzidos em um mapa ou roteiro. Esse material gerado na fase de desenho é o documento base da fase de desenvolvimento.

Na fase de desenvolvimento, o material gerado na fase anterior deve ser validado a partir dos objetivos traçados na análise. Portanto, é recomendável retornar a essa etapa e checar os objetivos. A criação do material é feita na fase de desenho. Na etapa de desenvolvimento, encontram-se eventos relatados para a criação dos OAs. Nesta etapa podem ocorrer atividades, como produção de elementos de mídia, desenvolvimento de testes de usabilidade, interface, navegação de formulários, etc.

A fase de implementação é a hora de transformar todas as ideias em códigos na linguagem de programação escolhida e mais adequada aos propósitos do OA. O programa será implementado a partir do desenvolvimento do material que foi planejado na fase do desenho. Nesse estágio, são incluídos os metadados do objeto, os testes nos diversos ambientes, além dos testes de interface no Repositório de Objetos de Aprendi\agem (ROA) para com o objeto.

A fase de avaliação é a etapa que se dedica ao estudo da eficiência do programa. Essa avaliação pode ser feita a partir de um teste com um grupo pequeno, com pessoas de perfis variados ou ainda com um grupo específico de alguma organização. Existem várias formas de checar se os objetivos foram atingidos ou não.

Com o resultado dessa avaliação em mãos, é possível comparar com os objetivos traçados na fase de análise e, a partir daí, definir novos caminhos, repensar objetivos e procedimentos, ou manter exatamente como está.

Outras metodologias de processo de desenvolvimento de software, como Modelo Cascata, Modelo Incremental, Modelos Evolutivo ou Modelos Agéis podem ser aplicados ao desenvolvimento de OAs.

Atividades de avaliação

1. No Brasil, a Secretaria de Educação a Distância (SEED), através do programa Rede Interativa Virtual de Educação (RIVED) coordena a produção de conteúdos pedagógicos digitais, na forma de objetos de aprendizagem.
2. Qual o método de desenvolvimento adotado pelo RIVED?
3. Quais os documentos produzidos pelo RIVED e qual sua utilização?
4. Pesquise sobre a utilização dos OAs nas escolas brasileiras.
5. Que críticas se pode fazer ao modelo de desenvolvimento apresentado nesse capítulo?

Referências

Albert Endres e Dieter Rombach. A Handbook of Software and Systems Engineering - Empirical Observations, Laws and Theories. Addison-Wesley 2003.

Carma McClure. Software Reuse Techniques: Adding Reuse to the System Development Process. Prentice Hall 1997.

C. A. A. Nunes; D. Neves; R. Degani; A. F. Santos; P. Gouveia; R. Okuyama; R. Góes; E. Paideti; A. M. Navas; M. E. Fejes. "O processo de autoria/produção de objetos de aprendizagem de química: Uma experiência de trabalho colaborativo universidade-escola." Virtual Educa Bilbao, 2006.

Desmond D'Souza e Alan Wills. Objects, Components, and Frameworks with UML - The Catalysis Approach. Addison-Wesley 1999.

Doug McIlroy. Mass Produced Software Components. Proceedings of the NATO Software Engineering Conference. Outubro, 1968.

Johannes Sametinger. Software Engineering with Reusable Components. Springer-Verlag 1997.

LTSC IEEE. "Standard for Information Technology: Education and Training Systems - Learning Objects and Metadata", 2002. Disponível em: http://ltsc.ieee.org/wg12/

M. C. Pessoa; F. B. V. Benitti. "Proposta de um processo para produção de objetos de aprendizagem". Hífen, v. 32, p. 172-180, 2008.

Capítulo 5

Avaliação de software educativo

Objetivos

- Identificar o que são atributos de qualidade e qual é sua influência na arquitetura de software;
- Relacionar atributos de qualidade a decisões arquiteturais que os proporcionam;
- Entender que os atributos de qualidade se relacionam e como eles se relacionam.
- Identificar aspectos particulares da avaliação de qualidade do software educativo.

1. Qualidade de Software

Para avaliar a qualidade de um software é necessário compreender que, ao falarmos de software e de sua engenharia, estamos falando ao mesmo tempo de produtos e de processos, os quais não podem ser vistos dissociados. Para lidar com qualidade, é necessário termos claro que o processo de produção deve ter qualidade e que o produto deve ter qualidade.

O objetivo maior da Engenharia de Software é produzir software de qualidade. Em engenharia de software, a qualidade perseguida abrange dois aspectos que fazem parte de domínios distintos, embora estreitamente relacionados: o domínio operativo e o domínio tecnológico, designados também de qualidade básica e qualidade extra, respectivamente.

O domínio operativo está relacionado diretamente ao produto e a sua qualidade está relacionada à satisfação do usuário ou cliente e é percebida de diferentes formas. A qualidade interna ou do domínio tecnológico está associada aos desenvolvedores.

> Existem dois tipos de avaliação para o software: avaliação ao longo do processo de desenvolvimento e avaliação de produtos de software.

A avaliação ao longo do processo de desenvolvimento é importante e exige a definição e a implantação de um programa de qualidade que garanta a avaliação do software ao longo das etapas de desenvolvimento.

A qualidade do processo é fundamental para a qualidade do produto, embora não seja suficiente para garantir um produto de qualidade, que necessita ser também avaliado quanto às suas características. A qualidade de produtos é tratada, entre outras, na série de Normas ISO/IEC 9126, na série ISO/IEC 14598 e na Norma ISO/IEC 12119, esta última focalizando os requisitos de qualidade de pacotes de software.

De acordo com a norma ISO/IEC 9126, a avaliação de qualidade de um software deve seguir parâmetros distribuídos em seis características principais com cada uma delas divididas em um conjunto de particularidades:

1. **Funcionalidade:** A capacidade de um software prover funcionalidades que satisfaçam o usuário em suas necessidades declaradas e implícitas, dentro de um determinado contexto de uso. Suas sub-características são:
   - **Adequação -** mede o quanto o conjunto de funcionalidades é adequado às necessidades do usuário;
   - **Acurácia (ou precisão) -** representa a capacidade do software de fornecer resultados precisos ou com a precisão dentro do que foi acordado/solicitado;
   - **Interoperabilidade -** trata da maneira como o software interage com outro(s) sistema(s) especificado(s);
   - **Segurança -** mede a capacidade do sistema de proteger as informações do usuário e fornecê-las apenas - e sempre - às pessoas autorizadas;
2. **Confiabilidade:** O desempenho se mantém ao longo do tempo nas condições estabelecidas. Suas sub-características são:
   - **Maturidade -** capacidade do software em evitar falhas decorrentes de defeitos no software;
   - **Tolerância a Falhas -** capacidade do software em manter o funcionamento adequado mesmo quando ocorrem defeitos nele ou nas suas interfaces externas;
   - **Recuperabilidade -** medida pela capacidade de recuperação após uma falha, restabelecendo seus níveis de desempenho e recuperando os seus dados;
3. **Usabilidade:** capacidade do produto de software ser compreendido, seu funcionamento aprendido, ser operado e ser atraente ao usuário.

   Observe que o conceito de usabilidade é bastante amplo e se aplica mesmo aos pequenos programas que não possuem uma interface para o usuário final. Por exemplo, um programa batch executado por uma ferramenta

de programação de processos também pode ser avaliado quanto à sua usabilidade, no que diz respeito a ser facilmente compreendido, aprendido, etc. Além disto, a operação de um sistema está sujeita às avaliações de usabilidade no domínio da Interface Humano-Computador (IHC). As sub-características consideradas na Usabilidade são:

- **Inteligibilidade** - facilidade com que o usuário pode compreender as suas funcionalidades e avaliar se ele pode ser usado para satisfazer as suas necessidades específicas;
- **Apreensibilidade** - identifica a facilidade de aprendizado do sistema para os seus potenciais usuários;
- **Operacionalidade** - como o produto facilita a sua operação por parte do usuário, incluindo a maneira como tolera erros de operação;
- **Atratividade** – conjunto de características que possam atrair um potencial usuário para o sistema, o que pode incluir desde a adequação das informações prestadas para o usuário até os requintes visuais utilizados na sua interface gráfica;

4. Eficiência: Os recursos e os tempos utilizados são compatíveis com o nível de desempenho requerido para o software. Suas sub características são:

- **Comportamento em Relação ao Tempo** - avalia se os tempos de resposta ou de processamento estão dentro das especificações;
- **Utilização de Recursos** - mede tanto os recursos consumidos quanto a capacidade do sistema em utilizar os recursos disponíveis;

5. Manutenibilidade: Capacidade que o produto de software apresenta para ser modificado, incluindo tanto as melhorias ou extensões de funcionalidade quanto as correções de defeitos, falhas ou erros. Suas sub características são:

- **Analisabilidade** - facilidade em se diagnosticar eventuais problemas e identificar as causas das deficiências ou falhas;
- **Modificabilidade** - facilidade com que o comportamento do software pode ser modificado;
- **Estabilidade** - capacidade do software de evitar efeitos colaterais decorrentes de modificações introduzidas;
- **Testabilidade** - capacidade de se testar o sistema modificado, tanto quanto às novas funcionalidades quanto em relação às não afetadas diretamente pela modificação.

6. Portabilidade: Capacidade de o sistema adaptar-se ao ser transferido de

um ambiente para outro. Por ambiente, devemos considerar todos os fatores de adaptação, tais como diferentes condições de infraestrutura (sistemas operacionais, versões de bancos de dados, etc.), diferentes tipos e recursos de hardware (como aproveitar um número maior de processadores ou memória). Além destes, fatores como o idioma ou a facilidade para se criar ambientes de testes devem ser considerados como características de portabilidade. Suas sub características são:

- **Adaptabilidade** - capacidade do software se adaptar a diferentes ambientes sem a necessidade de ações adicionais (configurações);
- **Capacidade para ser instalado** - identifica a facilidade com que se pode instalar o sistema em um novo ambiente;
- **Coexistência** - mede o quão facilmente um software convive com outros instalados no mesmo ambiente;
- Capacidade para substituir - representa a capacidade que o sistema tem de substituir outro sistema especificado, em um contexto de uso e ambiente específicos. Este atributo interage tanto com adaptabilidade quanto com a capacidade para ser instalado.

2. Avaliação de software Educacional

A dimensão epistemológica da didática, que considera em sua problemática a especificidade do conhecimento a ser trabalhado, torna o software educacional singular em relação às outras áreas do conhecimento. Assim, a avaliação de um software educacional não é uma tarefa restrita apenas aos parâmetros definidos em normas.

A qualidade de um software educativo está relacionada à capacidade que tem para ser inserido no ambiente de aprendizagem contemplando os requisitos de conteúdo e favorecendo a dinâmica da interação aluno/professor. Isso significa dizer que a análise de um sistema computacional com finalidades educacionais não pode ser feita sem considerar o seu contexto pedagógico de uso. Um software só pode ser tido como bom ou ruim dependendo do contexto e do modo como ele será utilizado.

Nesta perspectiva, para ser capaz de qualificar um software é necessário clareza da abordagem educacional, a partir da qual ele será utilizado e qual o papel do computador nesse contexto, ou seja, implica ser capaz de refletir sobre a aprendizagem a partir de dois eixos: a promoção do ensino e a construção do conhecimento pelo aluno.

O processo de avaliação de software deve ocorrer preferencialmente em diferentes estágios de seu desenvolvimento. O método de avaliação mais

apropriado irá depender do estágio de desenvolvimento e da disponibilidade de recursos e podem ser enquadradas em cinco categorias de avaliação.

Apresentaremos a seguir, os cinco tipos de avaliação.

- **Avaliação analítica:** Definida como uma avaliação que realiza uma descrição formal ou semi-formal da interface para predizer o desempenho do usuário no que se refere às operações físicas e cognitivas que devem ser realizadas. Esta avaliação pode ser aplicada nos estágios iniciais do desenvolvimento do software, pois demanda poucos recursos. Trata-se de uma avaliação preliminar, que não exige testagem com o usuário, possui um foco muito estreito e não gera resultados diagnósticos para um replanejamento.
- **Avaliação por peritos:** Realizada por expertises, que são solicitados a julgar o software e a identificar os problemas potenciais de usabilidade, assumindo o papel de usuários menos experientes. É um método barato e bastante eficiente. Não envolve testagem com usuários nem exige um grande número de especialistas, e pode ser usado nos primeiros protótipos ou especificações do sistema. Deve-se ter cuidado, ao escolher os especialistas, para não introduzir vieses.
- **Avaliação observacional**: Consiste na coleta de dados a respeito do comportamento do usuário ao utilizar o software. As técnicas mais comuns são: observação direta, gravação de vídeo, autolog do software (gravação automática das interações do usuário com o sistema), e protocolos verbais (o usuário é convidado a expressar em voz alta observações e pensamentos). É um método a ser usado com protótipos que atingiram um estágio mais avançado de desenvolvimento.
- **Avaliação por inspeção:** Utiliza-se entrevistas e questionários com o propósito de evocar as opiniões subjetivas e a compreensão dos usuários a respeito da interface. Os questionários podem ser usados com um número maior de usuários do que as entrevistas, as quais podem consumir muito tempo.
- **Avaliação experimental:** Tipo de avaliação na qual o avaliador manipula diversos fatores associados com o projeto da interface, a fim de estudar o seu efeito no desempenho do usuário. Normalmente, este método é aplicado em protótipos completamente desenvolvidos, requer um bom conhecimento de métodos experimentais e demanda grande quantidade de recursos e de tempo.

Como já foi dito, a qualidade de um software educativo está relacionada à capacidade que tem de atender a requisitos e necessidades dos usuários relacionadas à aprendizagem e aos aspectos contextuais que determinam a interação aluno/professor. Para tal, faz-se necessário definir métricas para cada variável pertinente considerando os paradigmas pedagógicos para os quais foi projetado.

As diferentes modalidades dos softwares educacionais - como exercício e prática, tutoriais, simulações, jogos, hipertexto/hipermídia, e sistemas baseados em conhecimento (sistemas especialistas e tutores inteligentes -, bem como as especificidades da população alvo para quem o software foi escrito, apresentam características diferentes, sendo necessária a elaboração de critérios de avaliação específicos para cada tipo.

Apresentamos a seguir algumas características importantes de qualidade que deverão ser avaliadas e organizadas segundo seus paradigmas instrucionais.

2.1. Características de Qualidades dos Softwares Educacionais

2.1.1. Exercício e prática

- Acesso a ajudas, para encaminhar o aluno a respostas certas, sendo que o acesso a ajudas deve ser diminuído de acordo com a crescente complexidade da instrução.
- Existência de mensagens de erro que encaminhem o aluno para a resposta adequada.
- Uso de ilustrações, de animação, de cor, de recursos sonoros, para despertar, manter e reforçar a atenção e a motivação do aluno.
- Controle da sequenciação do programa pelo usuário, para respostas também sequenciadas e capazes de oferecer situações para a aquisição e retenção do conteúdo.
- Fornecimento de feedback para melhorar o desempenho do aluno e assegurar o conhecimento da habilidade desejada.
- Tratamento de erro do usuário, para conduzi-lo ao domínio do conteúdo.
- Geração randômica de atividades, para a retenção, melhoria do desempenho e atenção.
- Capacidade de armazenamento das respostas, a fim de verificar o desempenho do aluno.
- Adaptabilidade ao nível do usuário, assegurando o domínio das habilidades necessárias.

- Adequação do programa ao currículo da escola.
- Integração do programa com outros recursos ou materiais instrucionais, para o enriquecimento do aluno.
- Apresentação dos escores e resultados dos alunos para que eles saibam sua posição diante do conteúdo apresentado.
- Resistência do programa a respostas inadequadas assegurando a continuidade do programa.
- Facilidade de leitura da tela a fim de obter uma interação adequada com o aluno.

2.1.2. Tutorial

- Existência de recursos motivacionais, a fim de garantir a atenção do aluno.
- Fornecimento de feedback encaminhando para respostas corretas.
- Possibilidade de adequação do vocabulário ao nível do usuário que garanta a compreensão do conteúdo e do que está sendo pedido.
- Controle da sequência do programa pelo usuário especificando caminhos alternativos que possam ser escolhidos e seu grau de dificuldade.
- Mensagens de erro com o intuito de conduzir o aluno à resposta correta, desejada ou adequada.
- Uso de Ilustrações, de animação, de cor, e de recursos sonoros, para despertar a atenção do aluno e mantê-la.
- Definição do tempo de resposta para permitir que o aluno aprenda em seu próprio ritmo.
- Adaptabilidade ao nível do usuário, promovendo a aplicação das habilidades já adquiridas e a aprendizagem das novas.
- Existência de ramificação para enfoques alternativos da instrução, facilitando a retenção e a transferência.
- Existência do tratamento de erro do usuário, analisando as respostas do aluno, determinando ou propondo segmentos corretivos ou mesmo fornecendo novas abordagens do conteúdo apresentado.
- Capacidade de armazenamento das respostas, para a verificação do desempenho final do aluno.

- Possibilidade de inclusão de novos elementos e/ou estruturas de conteúdo, tornando a substância do programa sempre atualizada e ao nível da clientela.
- Indicação de outros materiais para enriquecimento do aluno.
- Adequação do programa às necessidades curriculares da escola e a consequente integração ao currículo.
- Verificação dos pré-requisitos necessários à instrução, a fim de diagnosticar as necessidades do aluno e situá-lo no programa.
- Orientação da aprendizagem através de segmentos propostos no menu.
- Resistência do programa a situações hostis.
- Facilidade de leitura da tela, ou seja, interação humano computador adequada ao perfil do aluno.
- Passos da solução de problemas implícitos no programa.
- Capacidade de reformular o conhecimento, inferindo sobre o conhecimento já existente.
- Existência de recursos motivacionais.
- Adequação do vocabulário.
- Controle da sequência do conteúdo da aprendizagem pelo aluno, pelo do programa ou pelo professor.
- Mensagens de erro.
- Representação do conhecimento utilizando som, cor, animação e gráficos.
- Ramificações do processo instrucional.
- Armazenamento de respostas.
- Diagnóstico dos pré-requisitos.
- Orientação da aprendizagem.
- Robustez.
- Facilidade de leitura das telas.
- Suporte para análises heurísticas.
- Respostas satisfatórias.

2.1.3. Simulações e Modelagem.

- Fornecimento de feedback.
- Clareza nos comandos pedidos pelo programa.
- Controle das sequências reprodutoras do evento pelo aluno, facilitando a simulação da realidade.
- Mensagens de erro claras e indicadoras do caminho correto a ser seguido pelo aluno.
- Uso de ilustrações, de cor, animação e recursos sonoros para fornecer dados mais reais ao aluno, suprindo deficiências que a palavra escrita possa apresentar.
- Facilidade de leitura da tela.
- Apresentação dos resultados ao aluno, tanto parcialmente quanto ao final da simulação.
- Ramificações para enfoques alternativos, apresentando as possibilidades diante do problema simulado.
- Capacidade de armazenamento das respostas, a fim de se conhecer a estrutura do raciocínio do aluno diante do problema dado.
- Possibilidade de inclusão de novas estruturas e/ou segmentos de programa, a fim de manter o conteúdo sempre atualizado e reproduzindo a realidade.
- Possibilidade de correção de erros realizados pelo aluno e detectados pelo próprio antes do registro.

2.1.4. Jogos

- Existência de recursos motivacionais para despertar, manter e fixar a atenção do aluno.
- Clareza dos comandos a serem pedidos pelo programa antes de seu início.
- Controle da sequência pelo aluno.
- Mensagens de erro claras.
- Fornecimento de diretrizes no início do jogo e sua manutenção, a não ser quando a descoberta for parte do jogo.
- Diagramação da tela de forma clara.

- Apresentação dos resultados do aluno e de seu nível de desempenho.
- Adaptabilidade ao nível do usuário, promovendo interações que facilitem o alcance do objetivo do jogo.
- Capacidade de resistência a situações hostis.
- Fornecimento de feedback para facilitar o aumento do conhecimento e estimular o aluno-usuário.

2.1.5. Hipertexto / Hipermídia

- Qualidade da documentação.
- Facilidade de entendimento da estrutura do hiperdocumento.
- Documento disponível sobre os nós.
- Alterabilidade corretiva.
- Clareza das informações.
- Facilidade de aprendizado.
- Eficiência de utilização.
- Facilidade de lembrança.
- Facilidade de localização.
- Clareza dos comandos.
- Adequação do vocabulário ao nível do usuário.
- Estabilidade.
- Existência de recursos motivacionais.
- Controle da sequenciação do hiperdocumento.
- Diagramação das telas.
- Uso de ilustrações, animação, vídeo, cor.
- Uso de marcas especiais.
- Uso de recursos sonoros.
- Facilidade de leitura de textos na tela.
- Tempo de troca de nós.
- Suporte de múltiplas janelas.
- Rolamento de telas e janelas.

- Uso de ícones.
- Previsão de realimentação.
- Seleção de auxílio.
- Integração.
- Tutorial para leitura.
- Tempo de exposição de telas.

2.1.6. Tutores Inteligentes

- Conhecimento do domínio do problema.
- Representação do entendimento do aluno sobre o assunto durante a sessão de tutoria.
- Conhecimento de estratégias pedagógicas.
- Possibilidade de reconhecimento do raciocínio do aluno.
- Diálogo intensivo e ativo com o aluno.
- Modelagem do ensino.
- Interface que modela o sistema, oferecendo possibilidades de utilização de outras modalidades como simulação.
- Possibilidade de uso de animação e uso de símbolos expressivos (ícones).
- Possibilidade de diagnóstico do conhecimento do aluno sobre o problema.
- Pedido de resposta do aluno, a fim de oferecer tutoria constante.
- Módulo Especialista.
- Base de + conhecimento.
- Máquina de inferência.
- Modelo do aluno.
- Módulo de Ensino.
- Interface.
- Possibilidade de oferecimento de ajuda para solução do problema.
- Possibilidade de análise estatística do desempenho do aluno.
- Possibilidade de inferência do domínio do aluno sobre o tópico.

De uma forma geral, a qualidade de um software educativo está relacionada à finalidade de uso e à possibilidade de permitir a emergência de situações que levem os usuários a um nível de esforço mental que os faça compreender parte das propriedades de um novo conceito. O entendimento das necessidades do usuário é visto como um fator de sucesso para o desenvolvimento de um produto.

Este entendimento deve ocorrer no início do desenvolvimento, quando os requisitos são definidos, caso contrário o sistema não será bem aceito pelos usuários, gerando a necessidade de um re-trabalho para que venha atender às reais necessidades.

Atividades de avaliação

1. Quais os atributos de um bom software?
2. Quais particularidades possuem o software educativo?
3. Como avaliar os software educativos no contexto de uso?
4. Como avaliar um software educativo no contexto aprendizagem?

Referências

MORAN, José Manuel; MASETTO, Marcos; BEHRENS, Marilda Aparecida. **Novas tecnologias e mediação pedagógica.** São Paulo: Papirus, 2003.

_____ **Ensino e aprendizagem inovadores com tecnologia audiovisuais e telemáticas**. 2000. Disponível em: <www.eca.usp.br/prof/moran/inov.htm>. Acesso em: 05 abril. 2012.

RAMOS, Kátia. Educação a Distância provoca mudanças na gestão da aprendizagem e do conhecimento. Disponível em: http://www.educacaoetecnologia.org.br/?p=2508. Acesso em: 05 abril 2012.

ROCHA, Ana Regina Cavalcanti da. **Qualidade de software** – teoria e prática. Ed. Prentice Hall, 2001.

Sommerville, I. **Engenharia de Software**. 6. ed. São Paulo: Pearson Education do Brasil, 2003.

Trebien, E.S.E. **Software educacional:** modelo de desenvolvimento. União da Vitória: Face, 2003.

Sobre os autores

Carlos Alberto Santos de Almeida – Professor do Departamento de Física da Universidade Federal do Ceará (UFC) desde 1985. Atua no Mestrado em Ensino de Ciências e Matemática da UFC nas linhas de pesquisa Métodos Pedagógicos no Ensino de Ciências e Divulgação Científica e Espaços Não Formais para o Ensino de Ciências. Atua também no curso de Pós-graduação em Física da UFC, tendo orientado várias teses e dissertações. Ministra palestras nas áreas de Física e Divulgação de Ciências. É bacharel e mestre em Física pela UFC e doutor em Física pelo Centro Brasileiro de Pesquisas Físicas (Rio de Janeiro).

Rosa Lívia Freitas de Almeida – Engenheira Elétrica pela Universidade de Fortaleza, especialista em Analista de Sistemas pela Pontifícia Universidade Católica (PUC), do Rio de Janeiro (1989), mestre e doutora em Saúde Coletiva pela Universidade Federal do Ceará. Atua como Consultora em Tecnologia da Informação e Estatística, gerência de projetos de informática, operacionalização e suporte em Ambientes Virtuais de Aprendizagens (AVA). É especialista em georreferenciamento e estatística espacial, análise de sistemas, desenvolvimento e implantação de sistema de informação.