UNIVERSIDADE FEDERAL DE PELOTAS

Instituto de Física e Matemática

Departamento de Informática

Trabalho Acadêmico

UM MODELO DE PROCESSO PARA O DESENVOLVIMENTO DE APLICAÇÕES INTERATIVAS EM TV DIGITAL

Piero Silva Salaberri

Pelotas, 2008

PIERO SILVA SALABERRI

Um Modelo de Processo para o Desenvolvimento de Aplicações Interativas em TV Digital

Trabalho de conclusão de curso apresentado ao curso de Bacharelado em Ciência da Computação da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Eliane da Silva Alcoforado Diniz, MSc.

Pelotas, 2008

BANCA EXAMINADORA

Dedico este trabalho e tudo mais ao Tuta, à Tita e ao Gi. Esses sim, merecem todo o meu melhor. Também aos amigos, sem eles os dias seriam mais cinzentos.

AGRADECIMENTOS

Se tiver que agradecer, agradeço ao destino. Ele não acertou em tudo, mas o que é perfeito? Acertou bem mais. A **família** que tenho é o maior exemplo. Há muito sei que sem eles os objetivos traçados de nada valeriam. Logo, o maior agradecimento vai para todos os momentos de outrora e, principalmente, aos vindouros com minha família. Ao **Batuta** e a **Tita**, que mostram, diariamente, o amor irrestrito que por mim cultuam, através das preocupações bobas, nas tentativas de ajuda... Vocês não podem ser melhores. Sou eternamente grato por ter tido como principais educadores pessoas tão honestas, companheiras, alegres, dignas. Ao **Giuri**, meu irmão, melhor amigo, maior companheiro. Gratificado fui de crescer junto contigo, ao teu lado tenho certeza que nunca me sentirei desamparado.

Agradeço aos amigos, claro. Agradeço pelas risadas, pelo companheirismo, pelas concentrações, pelas reuniões improvisadas no domingo, pelas histórias divididas, pelo incentivo frente a quaisquer dificuldades. Sou grato por ter tido a oportunidade de dividir importantes passagens da minha vida com todos vocês, sei que serão os alvoroçados arroio-grandenses gritando ao som do meu nome na minha primeira grande conquista. Meu muito obrigado Bruno, Cristiano, Germano, Giuri, José Antônio, Mário Antônio e Rafael. Aos que não foram tão presentes, mas importantes também, Mika e Lucas, obrigado pelo companheirismo**!** Agradeço, também, às gurias que embelezam nossa turma. A amizade, alegria e sinceridade de cada uma levarei sempre comigo.

Um obrigado à “Turma do Barulho”. Dudu, Fabinhu e Robs, com vocês as aulas foram mais divertidas, os estudos mais proveitosos, as festas com mais risadas. Sou grato aos colegas não muito loucos que tive, por sempre estarem dispostos a ajudar em quaisquer dúvidas de trabalhos ou provas. Mesmo que juntos não tenhamos aproveitado o tempo extra-classe com esmero, os tenho como grandes amigos.

Agradeço a todos os professores que me mostraram a medida certa entre companheirismo e hierarquia, principalmente, à minha orientadora muito louca, Profa. Eliane Diniz, pelo apoio e correção de cada linha deste documento.

"Sê senhor da tua vontade e escravo da tua consciência".

Aristóteles

RESUMO

SALABERRI, Piero Silva. **Um Modelo de Processo para o Desenvolvimento de Aplicações Interativas em TV Digital.** 2008. 73p. Trabalho acadêmico – Curso de Bacharelado em Ciência da Computação. Universidade Federal de Pelotas, Pelotas.

É notório que se vive em uma época de transição entre as tecnologias de comunicação de massa. Diariamente, é visto que mais localidades já dispõem da transmissão digital. Todavia a maior mudança não é a digitalização do sinal da TV Aberta, mas sim a possibilidade da interação entre as produtoras da programação e os usuários. Portanto, vê-se a necessidade da elaboração de aplicações voltadas à interação do usuário/telespectador com as emissoras. Para a construção de tais aplicações faz-se necessário um estudo mais aprofundado de qual é a metodologia de software mais apta a esse cenário. Uma metodologia de software provê subsídios para que cada fase de um projeto tenha aporte necessário para sua conclusão. O trabalho sugere uma nova abordagem direcionada ao desenvolvimento de aplicações interativas em TV Digital, para tanto se baseia no que já é conhecido das metodologias vigentes para criar um híbrido, unindo as partes que melhor se adaptam das metodologias ao desenvolvimento de tais aplicações.

Palavras-chave: TV Digital Interativa. Metodologias de Desenvolvimento de Software. Aplicações Interativas. Xlet.

ABSTRACT

SALABERRI, Piero Silva. **Um Modelo de Processo para o Desenvolvimento de Aplicações Interativas em TV Digital.** 2008. 73p. Trabalho acadêmico – Curso de Bacharelado em Ciência da Computação. Universidade Federal de Pelotas, Pelotas.

It's well-known that people are living in a time of transition between mass communication technologies. Daily more and more places have the digital transmission available. Though the greatest change isn't the Open TV signal digitalization, but the possibility that now users have to interact with program producers. Therefore, there is the need to create applications adapted to user/spectator broadcasting channel interaction. To build such applications, a deep study of which software methodology is fits better this role is necessary. A software methodology provides subsidies, so each phase of a project can have the necessary contribution to it's conclusion. This work suggests a new approach directed to Digital TV interactive application development, and it's based on what already is available from effective methodologies to create an hybrid one, connecting the parts from these methodologies that better adapt to such applications development.

Keywords: Interactive Digital TV. Software Development Methodologies. Interactive Aplicattions. Xlet.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API – *Application Programming Interface*

ASD – *Adaptive Software Development*

AWT – *Abstract Window Toolkit*

DAVIC – *Digital Audio-Video Counci*

DVB – *Digital Vídeo Broadcasting*

EPG – *Eletronic Program Guide*

ES – Engenharia de Software

FDD – *Feature Driven Development*

Havi – *Home Audio-Video interoperability*

JAD – *Joint Application Development*

JMF – *Java Media Framework*

JVM – *Java Virtual Machine*

MHP – *Media Home Plataform*

OOAD – *Object-oriented analysis and design*

RTOS – *Real Time Operation System*

SI – *Service Information*

TDD – *Test Driven Development*

TV – Televisão

TVDI – Televisão Digital Interativa

UML – *Unified Modeling Language*

URL – *Uniform Resource Locator*

XP – *Extreme Programming*

SUMÁRIO

1 INTRODUÇÃO

O sinal da TV digital é composto por dados em forma de dígitos binários (bits), a transmissão não será composta somente por áudio e vídeo como nos dias atuais. Aliado ao som e a imagem incorporam-se aplicativos com intuito de realizar interatividade entre a audiência e o programa de TV, passando a TV Digital a ser Interativa. Através de mecanismos de acesso remoto, algumas dessas interações como votações e compras, poderão ser enviadas à emissora por um canal de interatividade (JUNOT, 2007).

Utilizando mecanismos de acesso remoto, o telespectador deixa de ser uma testemunha e passa a ser visto como um usuário ativo. Esses mecanismos possibilitarão um maior nível de interação, assim como é observado na Internet. Atualmente o controle remoto é a única forma de se interagir com o equipamento, posteriormente acredita-se que se terá uma gama de novos componentes com a capacidade de fazer esta comunicação. Porém, o desenvolvimento atual de programas não contempla esse novo nicho de mercado, sendo necessário um estudo mais aprofundado para verificar se as atuais modelagens para desenvolvimento de softwares permitem que estas aplicações sejam produzidas.

Para Rezende (2002), "uma metodologia de desenvolvimento de software é um processo dinâmico e iterativo para desenvolver (ou manter) de forma estruturada projetos, sistemas ou software". Quando se está focado em TV Digital Interativa – TVDI deve-se manter uma metodologia que privilegie o desenvolvimento de software da maneira mais eficiente possível e para isso algumas práticas devem ser analisadas, as quais segundo Veiga e Tavares (2007) são: planejamento**,** na metodologia para TVDI deve-se buscar, além da concepção do programa, a

priorização das tarefas e a estimativa de custo de cada iteração; ciclos curtos, o processo de desenvolvimento deve ser feito em ciclos curtos e iterativos, garantindo que a qualquer momento do projeto possa se verificar o andamento do projeto, utilizar um ciclo de desenvolvimento longo acaba trazendo lentidão ao processo; papéis multidisciplinares, já que é elevado o número de profissionais que compõem uma equipe de desenvolvimento de um programa para TVDI, então, o modelo de processo para desenvolvimento de programas interativos deve prever a identificação dos mais variados papéis e suas respectivas atuações; versões pequenas e funcionais, fazer a entrega de pequenas versões garante ao usuário/cliente a noção do avanço do projeto, reduz o risco da falta da entrega, ajuda na detecção prévia de falhas do sistema e fortalece a comunicação entre o desenvolvedor e o usuário/cliente; simplicidade, para garantir a comunicação constante trabalhando numa equipe multidisciplinar, nada mais natural que a solução adotada no desenvolvimento de programa para TVDI deve ser o mais simples possível, modularidade, garante o reuso do serviço em vários programas para TVDI uma vez que um mesmo serviço pode se repetir em diferentes programas.

1.1 Motivação

Para uma melhor construção de softwares que contemplem as funcionalidades inerentes aos processos interativos, faz-se necessário a realização de uma abordagem que defina quais são os métodos mais eficazes para tal construção. Muitos estudos estão focados na construção do projeto em um nível mais baixo, porém ainda se tem carência de métodos/ferramentas que auxiliem o desenvolvimento de aplicações para a TVDI de modo que se possa ter projetos bem especificados, desenvolvidos e de manutenção facilitada. Decorrente desses fatos, a busca por uma metodologia que venha contemplar todos esses elementos foi agente motivador para a realização desse trabalho.

1.2 Objetivos

Esse trabalho tem como objetivo a sugestão de um modelo de processo que contemple as necessidades do desenvolvimento de aplicações interativas em TV

Digital. Para tanto, se faz necessário realizar um estudo aprofundado das metodologias vigentes, que resulte em definir quais são as metodologias e ferramentas da Engenharia de Software que melhor se adaptam às aplicações interativas em TV Digital, bem como, a validação da abordagem sugerida.

1.3 Organização do Trabalho

Este documento foi dividido em seis capítulos, iniciando por capítulos introdutórios, de conceituação, até os de desenvolvimento e de conclusão. O capítulo 2 apresenta conceitos referentes à Engenharia de Software. No capítulo 3 são apresentados conceitos sobre TVDI. O capítulo 4 traz um comparativo entre as metodologias ágeis escolhidas para a elaboração do que se propõe este trabalho. O capítulo 5 apresenta a sugestão de uma abordagem metodológica, bem como a respectiva validação. O capítulo 6 apresenta as conclusões obtidas com a realização deste trabalho.

2 ENGENHARIA DE SOFTWARE

Independentemente do tamanho e da complexidade, o desenvolvimento de um software requer conhecimentos e habilidades que sejam condizentes com seu grau de dificuldade, nota-se a importância de se ter um processo de desenvolvimento que preze a normatização de técnicas voltadas ao cumprimento de prazos, custos e de qualidade. Sendo esses fatores elementos observados na área de Engenharia de Software – ES. Sommerville (2000) entende que a ES deve adotar uma abordagem sistemática e organizada para o processo de desenvolvimento de um software, usando ferramentas e técnicas apropriadas ao problema a ser resolvido, às restrições de desenvolvimento e aos recursos disponíveis.

Um processo de desenvolvimento de software pode ser visto como um conjunto estruturado de práticas necessárias para o bom desenvolvimento de um produto de software. Baseando-se nestas informações pode-se definir que processos de software são atividades, tais como: técnicas, padrões e métodos; que aliadas facilitam a construção de software, buscando sempre qualidade máxima ao alcançar o objetivo final.

Os motivos que levam a uma definição padrão de processos de software podem ser descritos como observado em Humphrey (1990):

a) Redução dos problemas relacionados a treinamento, revisões e suporte a ferramentas;
b) as experiências adquiridas nos projetos são incorporadas ao processo padrão e contribuem para melhorias em todos os processos definidos e
c) economia de tempo e esforço na definição de novos processos adequados a projetos.

Ao contrário do que possa parecer não existe uma seqüência obrigatória de fases, sendo que diversos autores apontam a natureza não-simultânea das fases como uma realidade na aplicação de processos de software, e também defendem que o processo de software é muito mais iterativo e cíclico do que a idéia de fases simples pode sugerir (REIS, 2001).

2.1 Atividades do Processo de Software

Cada fase de um processo de software executa atividades básicas para que o objetivo final seja atingido. Segundo Sommerville (2000 apud REIS, 2001) podem ser identificadas as seguintes atividades:

1. Especificação
   a) Engenharia de Sistema - estabelecimento de uma solução geral para o problema, envolvendo questões extra-software.
   b) Análise de Requisitos - levantamento das necessidades do software a ser implementado. A Análise tem como objetivo produzir uma especificação de requisitos, que convencionalmente é um documento.
   c) Especificação de Sistema - descrição funcional do sistema. Pode incluir um plano de testes para verificar adequação.
2. Projeto
   a) Projeto Arquitetural - onde é desenvolvido um modelo conceitual para o sistema, composto de módulos mais ou menos independentes.
   b) Projeto de Interface - onde cada módulo tem sua interface de comunicação estudada e definida.
   c) Projeto Detalhado - onde os módulos em si são definidos, e possivelmente traduzidos para pseudo-código.
3. Implementação
   a) Codificação - a implementação em si do sistema em uma linguagem de computador.
4. Validação
   a) Teste de Unidade e Módulo - a realização de testes para verificar a presença de erros e comportamento adequado a nível das funções e módulos básicos do sistema.

b) Integração - a reunião dos diferentes módulos em um produto de software homogêneo, e a verificação da interação entre esses quando operando em conjunto.

5. Manutenção e Evolução

a) Nesta fase, o software em geral entra em um ciclo iterativo que abrange todas as fases anteriores.

2.2 Metodologias de Desenvolvimento

As metodologias existentes estão incluídas em dois grupos distintos: As Metodologias Pesadas e as Ágeis. As metodologias Pesadas ou tradicionais podem ser aplicadas apenas em situações em que os requisitos do sistema são estáveis e requisitos futuros são previsíveis. Por outro lado, em projetos dinâmicos, nota-se a necessidade de se trabalhar com grupos pequenos, prazos curtos, re-implementação de trechos do código por alteração de requisitos, agilidade no desenvolvimento de software e outras características que alicerçam as metodologias ágeis. As quais podem ser vistas como alternativa às abordagens tradicionais ou pesadas no desenvolvimento de software.

2.2.1 Metodologias Tradicionais

As metodologias tradicionais são também chamadas de pesadas ou orientadas a documentação. O software era todo planejado e documentado antes de ser implementado (ROYCE, 1970). A principal metodologia tradicional é o modelo Clássico (PRESSMAN, 2001). É um modelo em que existe uma seqüência de etapas a serem seguidas. Cada etapa tem associada ao seu fim uma documentação padrão que deve ser aprovada para que a próxima etapa se inicie. De uma forma geral fazem parte do modelo Clássico as etapas de definição de requisitos, projeto do software, implementação e teste unitário, integração e teste do sistema, operação e manutenção. A inflexibilidade na divisão das fases no Modelo Clássico é um entrave no desenvolvimento de software, pois ocasiona um problema de adaptabilidade em grande parte. É um modelo que deve ser usado somente quando os requisitos forem bem compreendidos. A Fig. 2.1 ilustra graficamente o modelo Clássico.

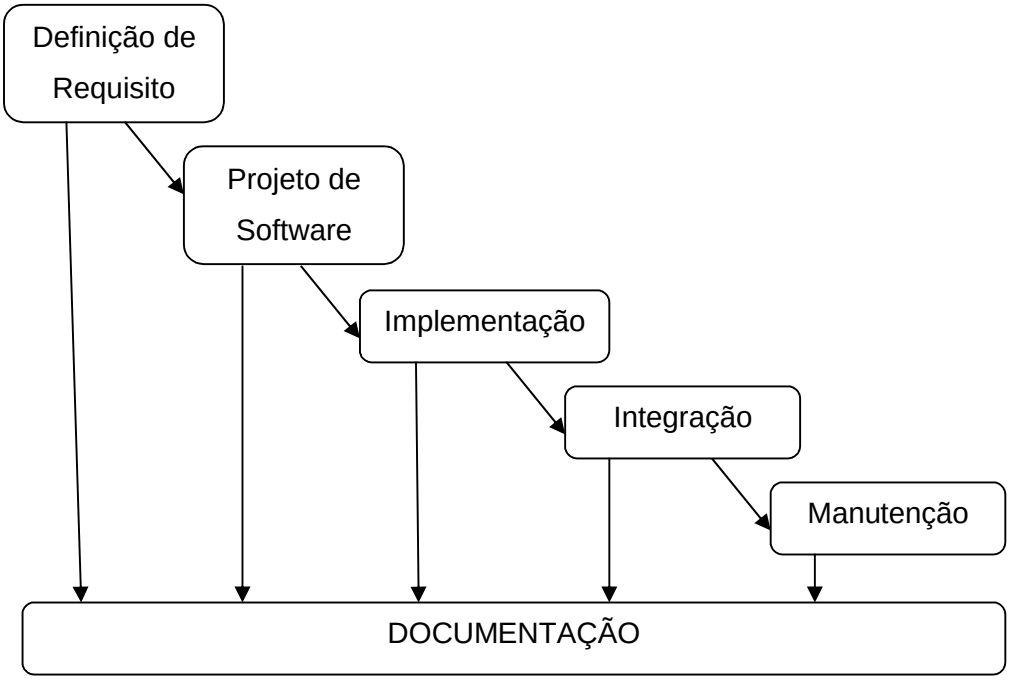

Figura 2.1– Modelo Cascata.
Fonte: adaptado de PRESSMAN, 2001.

a) Definição de Requisito - é a fase inicial, onde o programador deve se interar sobre o problema e reconhecer as necessidades do cliente.
b) Projeto de Software - nesta fase, o projetista começa a esboçar o projeto para futuramente ser implementado através de algum processo de modelagem previamente definido.
c) Implementação e teste unitário - onde ocorre a codificação do sistema. O programador codifica em uma linguagem de programação, com base no que o analista de sistema entendeu e validou junto ao cliente.
d) Integração - engloba a parte de validação. O programador verificará se o projeto atende a todas as necessidades averiguadas na fase de análise de requisitos através da comunicação entre os *stakeholders*[1].
e) Manutenção - se dá por novas versões decorrentes de correções de alguns trechos ou a adição de novas funcionalidades.

[1] *Stakeholders* – são todos os agentes envolvidos no processo, tais como: clientes, desenvolvedores, usuários, etc.

2.2.2 Metodologias Ágeis

Na década de 90, novos métodos surgiram expondo a idéia de que a abordagem de desenvolvimento fosse ágil, os processos adotados teriam a capacidade de adaptar-se às mudanças, apoiando a equipe no seu trabalho (BECK et. al., 2001).

Com o intuito de realizar uma nova maneira de se desenvolver software foi formulado um manifesto contendo um conjunto de princípios que definem critérios para os processos de desenvolvimento ágil de sistemas (AMBLER, 2004). Os doze princípios aos quais os métodos ágeis devem se adequar, de conformidade com Beck, et. al (2001), são:

a) A prioridade é satisfazer ao cliente através de entregas contínuas e freqüentes;
b) receber bem as mudanças de requisitos, mesmo em uma fase avançada do projeto;
c) entregas com freqüência, sempre na menor escala de tempo.
d) as equipes de negócio e de desenvolvimento devem trabalhar juntas diariamente;
e) manter uma equipe motivada fornecendo ambiente, apoio e confiança necessários;
f) a maneira mais eficiente da informação circular é através de uma conversa face-a-face;
g) ter o sistema funcionando é a melhor medida de progresso;
h) processos ágeis promovem o desenvolvimento sustentável;
i) atenção contínua a excelência técnica e a um bom projeto aumentam a agilidade;
j) simplicidade é essencial;
k) as melhores arquiteturas, requisitos e projetos provêm de equipes organizadas;
l) em intervalos regulares, a equipe deve refletir sobre como se tornar mais eficaz.

A seguir serão apresentadas metodologias: *eXtreme Programming, Scrum*, *Feature Driven Development* e *Adaptive Software Development*, pertencentes a esse

grupo, as quais foram selecionadas devidos as características que melhor se adéquam as aplicações interativas, de acordo com leituras previamente realizadas.

2.2.2.1 eXtreme Programmig

A metodologia *eXtreme Programming* - XP baseia-se em cinco valores, como pode ser visto na seção 2.2.2.1.1, para guiar o desenvolvimento de um produto de software.

2.2.2.1.1 Valores do XP

Os valores da metodologia XP, segundo Beck (1999), são:

a) Comunicação - o objetivo deste item é manter um canal aberto de interação entre os *stakeholders*.
b) Simplicidade - a simplicidade busca minimizar o código, extinguindo funções desnecessárias. Isto quer dizer que as classes e os métodos devem ser concisos e de número reduzidos. Deve-se visar somente o emergencial e o necessário.
c) *Feedback* - é caracterizado pelo contato direto e constante com o cliente a fim de elucidar qualquer dúvida em relação ao projeto através de testes periódicos e entrega de partes funcionais para que sejam feitas avaliações experimentais. O *feedback* durante todo o desenvolvimento do plano de trabalho assegura um produto final mais adequado aos anseios do cliente.
d) Coragem - a coragem se vê atrelada aos outros itens anteriores, já que o desenvolvedor deve ter facilidade para comunicação, iniciativa para tornar o projeto mais claro e enxuto e a capacidade de absorver e implementar mudanças de requisitos durante a execução do projeto.
e) Respeito - é fundamental tanto dentro da equipe de desenvolvimento como para com o cliente. Sem o respeito perde-se a comunicação e a interação com os membros da equipe.

2.2.2.1.2 Práticas do XP

A XP baseia-se nas 12 práticas a seguir (BECK, 1999):

a) Planejamento - evidencia o que há de emergencial para o desenvolvimento.
b) Entregas freqüentes - a entrega de pequenas implementações funcionais visa a constante aprovação do software e a incorporação rápida de mudanças por parte dos contratantes através do *feedback*.
c) Metáfora são as descrições de um software sem a utilização de termos técnicos, com o intuito de guiar o desenvolvimento do software.
d) Projeto simples - o programa desenvolvido pelo método XP deve ser o mais simples possível e satisfazer os requisitos atuais.
e) Testes - busca a validação do projeto durante todo o processo de desenvolvimento. Os programadores desenvolvem o software criando primeiramente os testes.
f) Programação em pares - dois desenvolvedores trabalham em um único computador. O desenvolvedor que está com o controle do teclado e do mouse implementa o código, enquanto o outro observa o trabalho, buscando encontrar erros sintáticos e semânticos e buscando uma forma de se otimizar o que está sendo implementado pela sua dupla.
g) Refatoração - quando se percebe que é possível simplificar o módulo atual sem perder nenhuma funcionalidade.
h) Propriedade coletiva - o código do projeto pertence a todos os membros da equipe. Isto significa que qualquer pessoa que percebe que pode fazer alguma melhoria, pode fazê-lo, desde que valide as mudanças. A propriedade coletiva é uma segurança ao projeto caso algum dos membros deixe a equipe.
i) Integração contínua - é a prática de interagir e construir o sistema de software várias vezes por dia, mantendo os programadores em sintonia, além de possibilitar processos rápidos. Integrar apenas um conjunto de modificações de cada vez é uma prática que funciona bem, porque fica óbvio quem deve fazer as correções quando os testes falham, a última equipe que integrou código novo ao software. Esta prática é facilitada

com o uso de apenas uma máquina de integração, que deve ter livre acesso a todos os membros da equipe.

j) 40 horas de trabalho semanal - caso seja necessário trabalhar mais de 40 horas pela segunda semana consecutiva, existe um problema sério no projeto que deve ser resolvido com melhor planejamento ou outra atitude, não com horas extras.

k) Cliente presente - a participação do cliente durante todo o desenvolvimento do projeto é essencial. O cliente deve estar sempre disponível para elucidar questões acerca de requisitos, evitando atrasos e até mesmo construções erradas. Uma idéia interessante é manter o cliente como parte integrante da equipe de desenvolvimento.

l) Código padrão - padronização na arquitetura do código, para que este possa ser compartilhado entre todos os programadores.

2.2.2.1.3 Papéis da Equipe

A gerência em XP é dividida entre o treinador e o rastreador. Esses papéis podem ser executados pela mesma pessoa. As atribuições são divididas como segue:

a) treinador- se preocupa principalmente com a evolução do processo. O treinador ideal deve saber se comunicar, ter um bom conhecimento técnico e ser confiante.

b) rastreador - coleta métricas sobre o que está sendo desenvolvido no máximo duas vezes por semana. Deve garantir que o projeto continue extremo, mantendo a visão do projeto.

c) Programador - ocupa o principal papel em XP. Ele analisa, projeta, codifica, testa e integra o sistema. Além de estimar prazos, definir os cartões de tarefas, implementar refatoração sempre que necessário etc.

d) Cliente - escolhe o que deve ser feito primeiramente. Ele quem escreve os cartões de histórias e está sempre a disposição para verificar e implementar testes funcionais.

e) Testador - irá ajudar o cliente na definição e escrita dos testes funcionais. Ele não precisa ser uma pessoa com apenas essa função, pode também ser um dos programadores.

f) Consultor - é alguém que domina alguma área de interesse que está impedindo a continuidade do projeto. O papel do consultor não é solucionar o problema, mas propiciar ferramentas para que a própria equipe consiga encontrar essa solução.

2.2.2.1.4 Ciclo de Vida de um projeto XP

O ciclo de vida do XP está embasado em seis fases: Exploração, planejamento, iterações para primeira versão, produção, manutenção e fim do projeto. Em Wake (2002) e Beck (2000), essas fases são descritas, como segue:

a) Fase de Exploração - o objetivo desta fase é entender o que o sistema deve realizar, a partir desta compreensão pode-se estimar o que deverá ser feito.

b) Fase de Planejamento - nesta fase é definida a menor data e o maior conjunto de histórias que serão realizadas na primeira versão. O cliente enumera as tarefas por valor, os programadores as qualificam e os clientes então escolhem as histórias para a próxima versão.

c) Iteração para primeira versão - A primeira iteração expõe como a arquitetura do sistema se comportará. Logo, as histórias devem ser escolhidas no sentido de representar todo o projeto.

d) Fase de Produção - quando o processo de *feedback* é finalizado a produção pode ser iniciada. Existem alguns métodos para verificar se o software está pronto para entrar em produção, validados através de testes.

e) Fase de Manutenção - nesta fase podem ser incorporadas novas funcionalidades, manter o sistema existente rodando, substituir membros da equipe.

f) Fim do Projeto - não existindo novas histórias o projeto chegou ao seu fim. Deverá ser feita a documentação das funcionalidades implementadas para que, no futuro, programadores envolvidos em alterações possam compreender como este código foi desenvolvido.

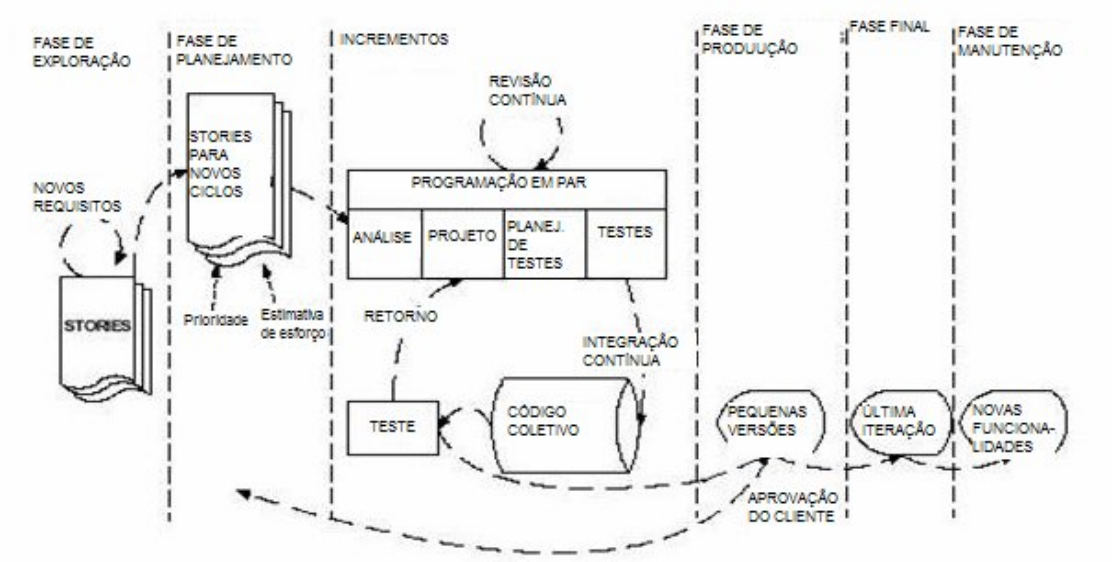

Figura 2.2 – Ciclo de Vida de um projeto XP
Fonte: Adaptado de http://emhain.wit.ie/~p02ac03/analysis&design.htm

2.2.2.2 SCRUM

Segundo Schwaber (1995), o *Scrum* não tem práticas pré-definidas para todas as situações. É utilizado em projetos altamente complexos onde não se pode prever tudo o que irá ocorrer. Essa Metodologia oferece um *framework* e um conjunto de práticas que facilitam a visão geral do projeto, permitindo aos desenvolvedores acompanhar todo o processo. As decisões e as estratégias que serão seguidas em busca da produtividade e o planejamento das entregas fica por conta de quem estiver aplicando. Um dos aspectos positivos do *Scrum* é a adaptabilidade.

2.2.2.2.1 Papéis da Equipe

O *Scrum* implementa um esqueleto interativo e incremental através de três papéis principais: o *Product Owner* (cliente), o *Scrum Team* (equipe de desenvolvimento), e o *Scrum Máster* (gerente) como descrito a seguir (BEEDLE et al., 1998):

a) *Cliente* - é a pessoa que define os itens que compõem o *Product Backlog* (requisitos) e os prioriza nas *Sprint Planning Meetings* (reuniões de planejamento).
b) Equipe - são todos integrantes comprometidos em desenvolver o jogo do projeto, alcançando os objetivos.

c) Gerente - intermedeia negociações entre o cliente e o Time. Seu principal papel é garantir a plena utilização do *Scrum*.

2.2.2.2.2 Ciclo de Vida de um projeto *SCRUM*

Em um projeto, existem basicamente as seguintes etapas (TEAMSYSTEM, 2005):

a) Preparação - onde se define os casos de negócio, o conceito do jogo, os requisitos iniciais e outras premissas ligadas aos projetos;
b) *Sprints*- são iterações realizadas para entregar gradativamente as estórias que compõem o jogo, cada uma com duração de 2 a 4 semanas. Dentro de cada *Sprint* são realizadas algumas atividades:
   - Reunião de Planejamento - é o início de um *Sprint*, onde o cliente prioriza os itens do *Product Backlog* e define com a equipe as funcionalidades (*Product Increment*) a serem entregues ao cliente ao final do *Sprint*;
   - *Daily Scrum Meeting* - é um encontro diário realizado pela equipe onde os membros discutem o que foi trabalhado, no que irá ser trabalhado e nos possíveis impedimentos ao progresso do trabalho. Não mais que 15 minutos de duração.
   - Criação do *Product Increment* - a finalização das estórias definidas para um determinado *Sprint*.
   - Revisão da *Sprint* - a equipe apresenta o que foi desenvolvido ao cliente, que é responsável por validar e solicitar ajustes visando o objetivo exigido pelo cliente.
   - Retrospectiva da *Sprint* - objetiva identificar os pontos positivos e negativos do *Sprint*.
   - Atualização dos requisitos - o cliente é responsável por re-priorizar toda lista de itens do *Product Backlog* para que um próximo *Sprint* possa ser iniciado motivado pelos mais prioritários itens.
c) Encerramento - ocorre ao fim de todos *Sprints* e é marcada pela entrega do jogo final que motivou a criação do projeto.

A Fig. 2.3 lustra o ciclo de vida da metodologia SCRUM.

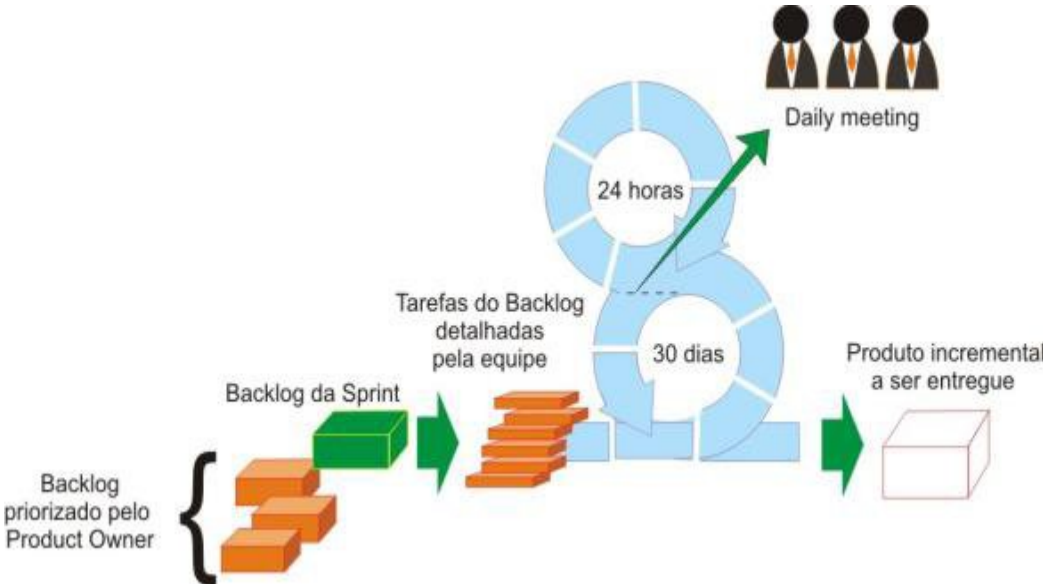

Figura 2.3 – Ciclo de Vida do SCRUM
Fonte: adaptado de BEDLEE, 2004.

2.2.2.2.3 Artefatos

A equipe deve utilizar alguns artefatos como apoio ao gerenciamento descentralizado e simples (SCHWABER, 1995):

a) Product Backlog - é uma lista priorizada pelo *Product Owner* de todas as estórias de usuário que o jogo deve conter (COHN, 2006);
b) Sprint Backlog - é uma lista das estórias mais relevantes, quebrada em tarefas com suas respectivas estimativas de duração do presente momento, até o fim do *Sprint*. É definida pelo time, e negociada com o *Product Owner*. O time deve procurar manter esta lista sempre atualizada;
c) *Impediment List* - É uma lista que contém os impedimentos para o time alcançar os objetivos. Tais impedimentos devem ser resolvidos pelo *ScrumMaster*.
d) *Reports*- São relatórios capazes de mostrar o andamento do projeto.

2.2.2.3 Feature Driven Development

O Desenvolvimento Guiado por Funcionalidades (Feature Driven Development – FDD) é uma metodologia ágil[2] para gerenciamento e desenvolvimento de software. É a união da Implantação das metodologias de *Object-oriented analysis and design* – OOAD de Peter Coad e de gerência de projetos de Jeff De Luca (DE LUCA, 2001).

A FDD, segundo Highsmith (2002), chama a atenção para algumas características peculiares:

a) Resultados úteis a cada duas semanas ou menos.
b) Blocos bem pequenos de funcionalidade valorizada pelo cliente, chamados "*Features*".
c) Planejamento detalhado e guia para medição.
d) Rastreabilidade e relatórios com incrível precisão.
e) Monitoramento detalhado dentro do projeto, com resumos de alto nível para clientes e gerentes, tudo em termos de negócio.
f) Fornece uma forma de saber, dentro dos primeiros 10% de um projeto, se o plano e a estimativa são sólidos.

[2] A Metodologia FDD foi criada em 1997 num grande projeto em Java para o United Overseas Bank, em Singapura. Teve como objetivo facilitar o desenvolvimento de um grande sistema de empréstimos para um banco internacional

2.2.2.3.1 Ciclo de Vida de um projeto FDD

A Fig. 2.4 traz o Ciclo de Vida da metodologia FDD.

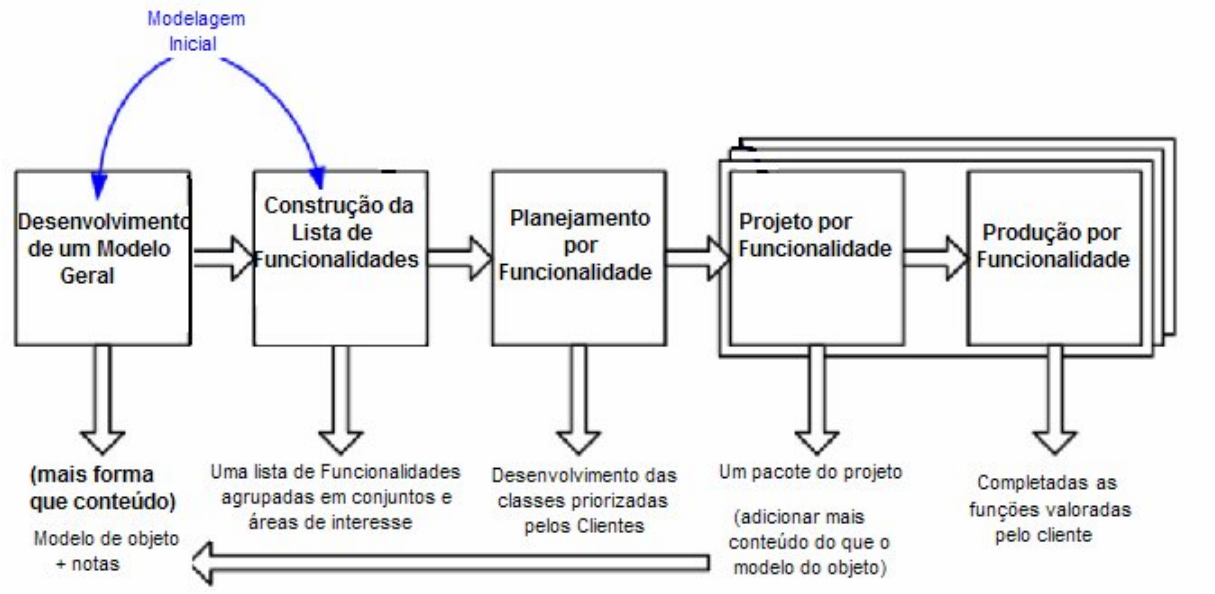

Figura 2.4 – Ciclo de Vida do FDD

Fonte: www.agilemodeling.com/essays/fdd

2.2.2.4 ASD – Adaptive Software Development

A Metodologia *Adaptive Software Development*[3] – ASD concentra-se no problema do desenvolvimento de sistemas grandes e complexos. Este método utiliza uma abordagem incremental e iterativa, com prototipagem contínua. Pretende atingir "o rigor estritamente necessário", e propõe para isso a execução do mínimo controle necessário.

Características específicas:

a) Planejamento dirigido por missões (feitas pelo cliente);
b) Foco no desenvolvimento baseado em componentes;
c) Uso de "*time-boxing*" (janelas temporais);
d) Consideração explícita dos riscos;
e) Ênfase na colaboração para levantamento de requisitos;

[3] A Metodologia Adaptive Software Development foi desenvolvida por Jeff Sutherland e Ken Schwaber em 1993.

f) Ênfase na “aprendizagem” ao longo do processo.

2.2.2.4.1 Ciclo de Vida de um projeto ASD

A Fig. 2.5 mostra o Ciclo de Vida da metodologia ASD.

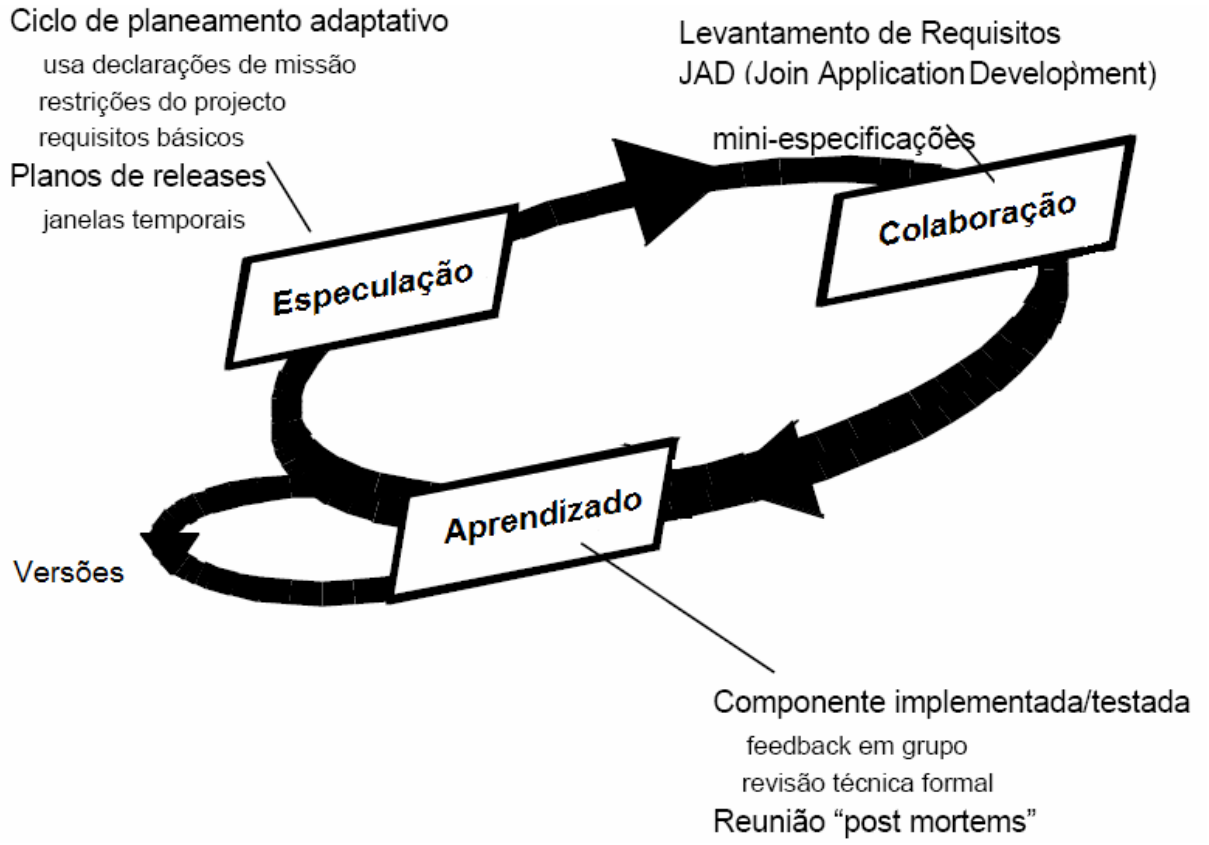

Figura 2.5 – Ciclo de Vida do ASD
Fonte: Adaptado de PRESSMAN, 2005.

Nesse capítulo foi mostrado como a Engenharia do Software traz ganhos quanto à organização de todas as fases de um projeto, bem como sendo uma ferramenta facilitadora de um ambiente próspero ao respeito de limites orçamentários e prazos pré-estabelecidos.

3 INTERFACE DE PROGRAMAÇÃO DE APLICAÇÃO

Segundo Ribeiro (2004), TVDI é uma evolução da TV convencional, associando a esta sinais digitais e interatividade. Ou seja, essa nova tecnologia oferece uma nova gama de aplicações que agrupam avançados recursos de interação.

A construção de aplicações interativas não conta ainda com uma estrutura pronta para o seu desenvolvimento. Portanto, para a elaboração de tais aplicações é utilizado o emulador *XletView* (Seção 3.2.1), que simula a execução de um *Xlet* (Seção 3.2) em um *Set-top Box*, estes conceitos serão abordados no transcorrer deste capítulo.

Este capítulo tratará acerca dos componentes em um projeto de TV Digital, bem como das rotinas que facilitam a programação de aplicações na programação e dos recursos gráficos disponíveis.

3.1 JavaTV

O JavaTV é uma interface de programação de aplicação (*Aplication Programming Interface* – API), e é uma extensão da linguagem Java da Sun[4] que tem por objetivo facilitar o trabalho dos desenvolvedores de aplicações de Televisão Interativa para set-top boxes. A API JavaTV torna possível a criação de aplicações interativas independente da tecnologia utilizada nos protocolos de transmissão, do

[4] Sun Microsystems é uma empresa fabricante de computadores, semicondutores e software com sede na Vale do Silício. O nome Sun vem de *Stanford University Network*.

sistema operacional dos set-top boxes e da camada de hardware das redes de difusão (JAVATV API OVERVIEW, 2008).

A JavaTV API está sendo projetada para fornecer o acesso de funcionalidades exclusivas aos receptores de televisão digital, incluindo:

a) Fluxo de áudio e vídeo;
b) Acesso condicional;
c) Acesso aos dados nos canais de transmissão;
d) Acesso aos dados do *Service Information*;
e) Controle do sintonizador de canais;
f) Sincronização de mídia, para permitir que conteúdo interativo seja sincronizado com o vídeo e o áudio do programa;

Além disso, a JavaTV API está sendo projetada para prover funcionalidade adicional sob a forma de sincronização de mídia e controle da execução das aplicações (ciclo de vida da execução). A sincronização de mídia permitirá que o conteúdo da TVDI seja sincronizado com vídeo e áudio de fundo em um programa de televisão. O controle da execução de uma aplicação permitirá a coexistência de seu conteúdo com o conteúdo da programação televisiva como, por exemplo, os comerciais.

A Fig. 3.1 mostra uma pilha de software para o ambiente Java TV API quando implementado em um receptor de televisão digital:

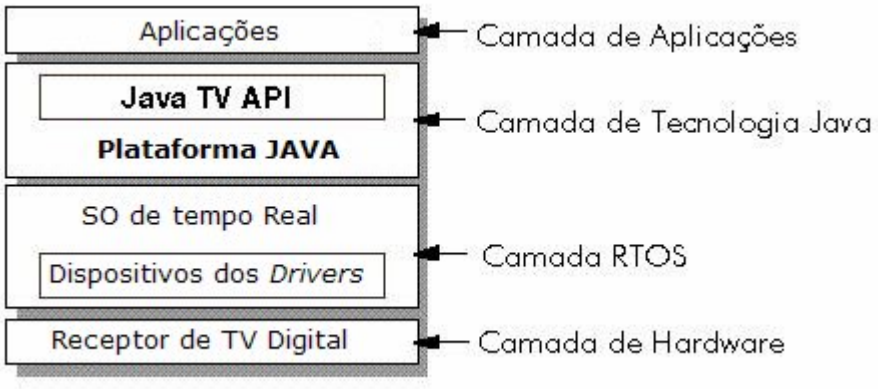

Figura 3.1 – Pilha de software para o ambiente JavaTV API
Fonte: Adaptado de http://java.sun.com/products/javatv/overview.html

No nível mais alto da pilha tem-se aplicações que fazem uso de JavaTV e das bibliotecas da plataforma Java, que geralmente será a Java 2, Micro Edition – J2ME devido ao pouco recurso de memória disponível num Set-top Box. Logo abaixo encontra-se o Sistema Operacional que dá o suporte necessário à

implementação da máquina virtual Java – JVM. Na camada de hardware, tem-se um receptor de televisão digital que suporta *broadcast*[5] e *pipeline*[6] de dados. A API Java provê uma abstração em relação à camada de hardware, fazendo com que os desenvolvedores de aplicações não se preocupem com o ambiente desta camada.

Alguns conceitos são importantes para o bom entendimento de tarefas e das partes componentes em um projeto de TV Digital. Segundo Loureiro (2004) alguns dos principais conceitos são:

a) Serviço - pode ser definido como um canal de televisão, composto por áudio, vídeo ou dados. Cada serviço possui um Service Information – SI associado a ele e cada entrada do SI define uma propriedade do serviço.
b) Componente do Serviço - são *streams* elementares como de áudio e vídeo, aplicação Java ou outro tipo de dado que pode ser apresentado sem uma informação descritiva adicional. Um componente pode ser compartilhado por mais de um serviço.
c) *Service Information* - SI - Descreve a estrutura do fluxo, além de conter informações relativas ao serviço como, por exemplo, um conteúdo descritivo, dados referentes aos horários de apresentação dos programas, a freqüência do canal para possibilitar a sintonização no devido serviço, informações de demultiplexação, a linguagem utilizada, informações da rede, além de meta-informações como uma lista dos atores ou categorização do serviço.
d) *Service Locator* - É a informação necessária que possibilita a sintonização do canal, análogo a uma URL.
e) SI *Database* - Representa a base de dados que armazena o SI.
f) SI *Manager* - É um objeto *singleton*, pois só existe um no receptor de televisão digital. SI *Manager* é o ponto de acesso para a base de dados do SI e ele reporta qualquer alteração que ocorra na base.
g) *SI Factory* - Responsável em criar objetos do tipo SI *Manager*.

---

[5] É o processo pelo qual se transmite ou difunde determinada informação, tendo como principal característica que a mesma informação está sendo enviada para muitos receptores ao mesmo tempo.

[6] *Pipeline* é uma técnica de hardware que permite que a CPU realize a busca de uma ou mais instruções além da próxima a ser executada.

3.1.1 A API

Java TV requer um ambiente *PersonalJava*™ e usa um sub-conjunto do *Abstract Window Toolkit* – AWT para construir as interfaces do usuário e Java *Media Framewok* – JMF para o controle da mídia. A API é definida pelos seguintes pacotes (INTRODUCTION TO DIGITAL TV APPLICATIONS PROGRAMMING, 2008):

a) *javax.tv.carousel* - provê acesso a arquivos *broadcast* e diretórios de dados através de APIs que trabalham com o pacote java.io.

b) *javax.tv.graphics* - permite que *Xlets* possam obter seu repositório principal e descreve mecanismos para *alpha blending*[7].

c) *javax.tv.locator* - provê uma forma para referenciar dados ou programas acessíveis pela API Java TV.

d) *javax.tv.media* - define uma extensão para JMF com a finalidade de gerenciar mídia em tempo real.

e) *javax.tv.media.protocol* - provê acesso a um fluxo de dados broadcast genérico.

f) *javax.tv.net* - permite acesso a datagramas *Internet Protocol* transmitidos num *stream broadcast*.

g) *javax.tv.service* - provê mecanismos para acessar a base de dados do SI e APIs comuns aos seus sub-pacotes.

h) *javax.tv.service.guide* - este pacote dá suporte a *electronic program guides* - EPGs, incluindo a programação, eventos e a classificação de programas.

i) *javax.tv.service.navigation* - permite navegar através dos serviços.

j) *javax.tv.service.selection* - provê mecanismos para selecionar os serviços que serão exibidos.

---

[7] O Alpha Blending consiste em juntar duas imagens levando em conta o canal alpha, ou seja, a transparência em cada pixel.

k) *javax.tv.service.transport* - permite consultar informações adicionais sobre os mecanismos de transporte, tais como TS[8], redes *broadcast* ou *bouquets*[9].

l) *javax.tv.util*- suporta a criação e o gerenciamento de eventos do timer.

m) *javax.tv.xlet* - provê interfaces para o desenvolvimento de aplicações e para a comunicação entre as aplicações e o gerenciador.

3.2 XLets

O conceito de *Xlet* foi introduzido pela Sun através da API JavaTV e foi adotado por vários padrões de *middleware*. Um *applet* Java em um *browser* para Computadores Pessoais se equivale ao *Xlet* da televisão interativa. O *Xlet* é propagado em uma seqüência de transporte MPEG-2 e carregado pelo set-top box assim que o telespectador seleciona um serviço.

A grande diferença em relação a uma *applet* é que o *Xlet* pode ser pausado e depois ter sua execução continuada. Isto se deve ao fato de que, em um ambiente de TVDI, vários *Xlets* podem ser executados em simultâneo, e um set-top box é desprovido de grandes recursos de hardware. Posteriormente, o *Xlet* paralisado pode retornar a execução.

Como pode ser visto na Fig. 3.2, primeiramente o gerenciador de aplicação carrega a classe principal do *Xlet* e cria uma instância da mesma. Assim, o *Xlet* entra no estado ‘carregado’. Quando o telespectador escolhe iniciar determinado serviço que contém o *Xlet*, ou quando outra aplicação determina que a mesma deve ser iniciada automaticamente, o gerenciador de aplicação invoca o método de inicialização do *Xlet*, passando, para o mesmo, um objeto contendo o contexto no qual ele se encontra. O *Xlet* usa esse objeto para sua inicialização e para pré-carregar todos os recursos da seqüência de transporte da qual fará uso. Após isso, o

[8] Transport Stream transmite apenas a parte da figura que foi alterada de um frame para o próximo ao invés de enviar o frame completo.

[9] Expressão que significa “agregar serviços”.

Xlet entra no estado ‘paralisado’, no qual está pronto para ser iniciado imediatamente.

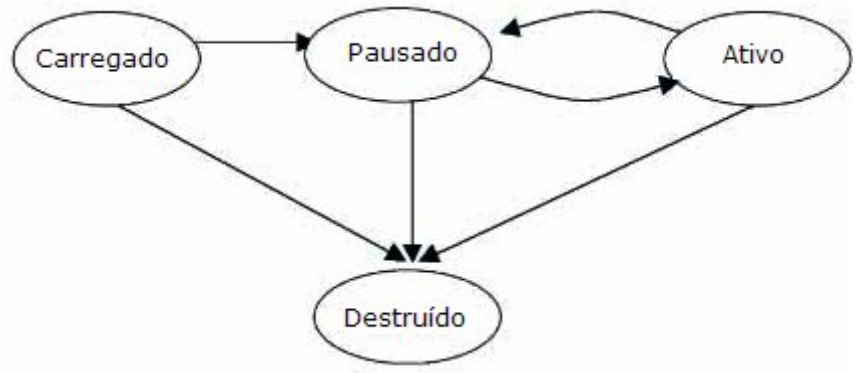

Figura 3.2 – Ciclo de Vida do XLet
Fonte:http://www2.sys-con.com/ITSG/virtualcd/Java/archives/0807/wang/index.html

3.2.1 XletView

Possui o código aberto sob a licença GPL, além de uma implementação de referência da API JavaTV, traz consigo implementações de outras APIs especificadas no padrão Media Home Plataform - MHP, como a *Home Audio-Video interoperability* - Havi, *Digital Audio-Video Council* - DAVIC e implementações especificadas pela própria *Digital Video Broadcasting* - DVB, além das bibliotecas do *PersonalJava* que o mesmo padrão faz uso (XLETVIEW, 2008).

Como é programado totalmente em Java, pode ser executado tanto em uma plataforma Linux ou Windows, bastando para isso utilizar o *Java2Standard Development Kit* para compilar *Xlets* e executar o *XleTView*.

Este capítulo trouxe um aspecto geral da arquitetura de um projeto em TV Digital, explicando como cada componente se comporta e como uma aplicação pode ser desenvolvida para que a emissora possa distribuir tais aplicações à massa de telespectadores. Abordou também sobre como as aplicações são desenvolvidas graficamente e sobre as ferramentas para sua visualização.

4 COMPARATIVO ENTRE PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE

A partir dos conhecimentos adquiridos através de leitura da descrição de metodologias de software, pré-definidas a partir da compatibilidade de suas características com os processos envolvidos na construção de aplicações interativas, descritas no Capítulo 5, chegou-se a conclusão que dentre as metodologias estudadas, quatro delas são as mais indicadas a fim de se criar um híbrido de metodologias de desenvolvimento de Software, visando tal processo de desenvolvimento. São estas as metodologias de desenvolvimento abordadas neste trabalho: *Extreme Programming* – XP (BECK, 2000), Scrum (BEEDLE et al, 1998), *Feature Driven Development* – FDD (DE LUCA, 2001) e *Adaptive Software Development* – ASD (HIGHSMITH, 2002).

Toma-se por base o Desenvolvimento Incremental para que se possa fazer uma análise detalhada das partes de um projeto. Sommerville (2003) afirma que os métodos ágeis são fundamentados no Desenvolvimento Incremental, onde os clientes, inicialmente, identificam por um esboço os requisitos do sistema, após, os identificam por grau de importância. Logo em seguida, atribuem-se esses requisitos às fases do projeto seguindo a ordem dada na primeira etapa. Projeta-se a arquitetura do sistema. Segue-se com o desenvolvimento do projeto em incrementos, validando-os ao final de cada ciclo. Após é feita a integração dos incrementos desenvolvidos pelos programadores e os testes cabíveis efetuados juntamente ao cliente. A partir da Fig. 4.1 pode-se identificar como se dispõe cada etapa do Desenvolvimento Incremental.

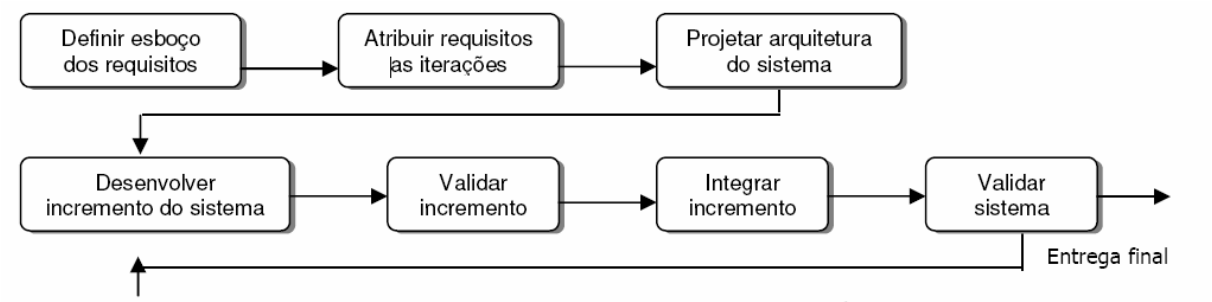

Figura 4.1 – Desenvolvimento Incremental
Fonte: Adaptado de Sommerville, 2003

Quando os requisitos já estão atrelados aos seus respectivos incrementos, as funcionalidades a serem entregues na primeira iteração são detalhadas e desenvolvidas. Porém, novas funcionalidades ainda desconhecidas do projeto podem ser adicionadas sem problemas no ciclo. Assim que cada iteração é completa, já está disponível para validação do cliente, isto acarreta o bom desenvolvimento do projeto, pois o cliente vai, gradualmente, observando a sua ampliação, adicionando novas funcionalidades ou desprezando outras desnecessárias (SOMMERVILLE, 2003).

Para uma melhor compreensão haverá no final de cada item uma tabela identificando, de forma resumida, como cada processo de desenvolvimento das metodologias XP, Scrum, FDD e ASD enfrentam as etapas do Desenvolvimento Incremental.

4.1 Definição do Esboço dos Requisitos

A definição do esboço dos requisitos no **XP** é feita através da escrita das *user stories* pelos clientes, embasados no que já conhecem dos requisitos. São as descrições das funcionalidades do sistema (BECK, 2000). No **Scrum**, os requisitos já conhecidos pelos clientes formam o *Product Backlog*, onde os requisitos serão ordenados observando a prioridade de cada um, apresentando a respectiva descrição, o tempo estimado para seu desenvolvimento e o responsável por sua elaboração (SCHWABER; BEEDLE, 2002). No **FDD**, os requisitos são divididos em

atividades contidas em áreas de interesse, para cada atividade há uma funcionalidade que a satisfaça. São construídos diagramas de classes *Unified Modeling Language* - UML para que se tenha uma visão clara de como funcionará o sistema (HIGHSMITH, 2002). O **ASD** não apresenta nenhuma atividade explícita em relação à definição preliminar dos requisitos (HIGHSMITH, 2002). Porém existe um conjunto de ferramentas facilitadoras para a execução de um projeto, conhecido como *Joint Application Development* – JAD. Sessões JAD objetivam identificar os requisitos através de reuniões previamente estruturadas guiadas por um líder. A Tab. 1 apresenta a especificação das atividades da etapa de definição do esboço dos requisitos.

Tabela 1 – Definição do Esboço dos Requisitos

Metodologia	Práticas
XP	Escrita das *user stories*
Scrum	Produção do *Product Backlog*
FDD	Atividades reunidas em Áreas de Negócios
ASD	Sessões JAD

4.2 Atribuição dos Requisitos às Iterações

A atribuição dos requisitos aos ciclos no **XP** é feita juntamente com os clientes (BECK, 2000). No **Scrum**, o *product backlog* é decomposto no *Sprint Backlog*, isto é, alocam-se atividades durante todas as *sprints* obedecendo a priorização dada na primeira etapa (BEEDLE et al, 1998). No **FDD**, as funcionalidades responsáveis pelo desenvolvimento dos requisitos são agrupados por áreas de interesses e divididos entre as fases do projeto (HIGHSMITH, 2002). O **ASD** define o número de iterações e os requisitos implementados em cada uma delas (ABRAHAMSSON, 2002). A Tab. 2 apresenta a especificação das atividades da etapa de atribuição dos requisitos às iterações.

Tabela 2 – Atribuição dos Requisitos às Iterações

Metodologia	Práticas
XP	Tarefa executada juntamente com os clientes
Scrum	*Sprint Backlog* gerada a partir da decomposição do *Product Backlog*
FDD	Requisitos agrupados por áreas de interesse
ASD	Definição dos incrementos e dos requisitos que comporão cada ciclo

4.3 Projeto da Arquitetura do Sistema

O projeto da arquitetura do sistema no **XP** é feito simultaneamente à escrita das *user stories*, todavia o processo para elaboração não explicita como se chega à arquitetura e aos artefatos (BECK, 2000). O **Scrum** enfatiza o uso do *product backlog* para a criação da arquitetura do sistema, também sem mencionar as etapas para sua construção (SCHWABER; BEEDLE, 2002). O **FDD** enfatiza o uso do diagrama de classes e de seqüência UML para esboçar a arquitetura do sistema (DE LUCA, 2001; HIGHSMITH, 2002). O **ASD** não menciona qualquer técnica que possa gerar a arquitetura do sistema. A Tab. 3 apresenta a especificação das atividades relacionadas à modelagem da arquitetura do sistema.

Tabela 3 – Projeto da Arquitetura do Sistema

Metodologia	Práticas
XP	Sem especificação sobre a arquitetura do sistema
Scrum	Sem especificação sobre a arquitetura do sistema
FDD	Construção de Diagramas UML
ASD	Sem especificação sobre a arquitetura do sistema

4.4 Desenvolver Incremento do Sistema

O desenvolvimento de incrementos do sistema acontece no **XP** paralelamente ao desenvolvimento dos requisitos (BECK, 2000). Para as *user*

stories que farão parte do ciclo são realizadas as atividades de modelagem e programação que satisfaçam o problema. No **Scrum**, desenvolve-se o incremento durante a *sprint* (SCHWABER, 1995). No **FDD,** embasa-se na arquitetura elaborada através dos diagramas UML e na documentação escrita até o momento para desenvolver os incrementos (HIGHSMITH, 2002). No **ASD** os requisitos são desenvolvidos, de preferência, simultaneamente dentro das iterações pré-definidas (ABRAHAMSSON, 2002). A Tab. 4 apresenta a especificação das atividades relacionadas ao desenvolvimento dos incrementos.

Tabela 4 – Desenvolver Incremento do Sistema

Metodologia	Práticas
XP	Os incrementos são desenvolvidos alicerçados nas 12 práticas do XP
Scrum	Os requisitos do *Sprint backlog* são desenvolvidos durante as *Sprints*
FDD	Os requisitos acatam o planejado na modelagem da arquitetura e do projetado para cada ciclo
ASD	Desenvolvem-se os requisitos pré-definidos a cada ciclo

4.5 Validar Incremento

A validação dos incrementos no **XP** se dá por testes de unidade aplicados pelos desenvolvedores e testes de aceitação, pelos usuários (BECK, 2000). No **Scrum** a validação ocorre no final de cada *sprint* (SCHWABER; BEEDLE 2002). No **FDD** os testes finalizam os ciclos (ABRHAMSSON, 2002). No **ASD,** qualificam a produção dos desenvolvedores (HIGHSMITH, 2002). A Tab. 5 apresenta a especificação das atividades relacionadas à validação dos incrementos.

Tabela 5 – Validar Incremento

Metodologia	Práticas
XP	Testes de unidade pelos desenvolvedores; de aceitação pelos clientes
Scrum	Ao final de cada *sprint*
FDD	Ao final de cada incremento
ASD	Testa se a produção da equipe foi satisfatória

4.6 Integrar Incremento

A integração do incremento acontece paralelamente seu desenvolvimento no **XP** (BECK, 2000). No **Scrum**, ao final de cada *sprint* o código é integrado (SCHWABER; BEEDLE, 2002). O **FDD** integra o código gerado ao final de cada iteração após a realização dos testes cabíveis (HIGHSMITH, 2002). O **ASD** não menciona qualquer especificação de como o incremento possa ser integrado ao projeto. A Tab. 6 apresenta a especificação das atividades relacionadas à integração dos incrementos.

Tabela 6 – Integrar Incremento

Metodologia	Práticas
XP	Paralelamente ao desenvolvimentos de novas funcionalidades
Scrum	Ao final de cada *sprint*
FDD	Ao final de cada incremento
ASD	Sem especificação sobre a integração de novos incrementos

4.7 Validar Sistema

Como no **XP** a validação acontece durante todo o seu desenvolvimento, não há um processo de validação final do sistema, o projeto que funciona perfeitamente e satisfaz os anseios do cliente está validado (BECK 2000). O **Scrum** valida cada

incremento ao final das *sprints. Q*uando chega ao final da última *sprint* realiza os testes de inspeção e o entrega ao cliente, porém recomenda o acompanhamento após a entrega para garantir a qualidade do produto desenvolvido (SCHWABER; BEEDLE, 2002). No **FDD** a validação é executada através de testes de integração (DE LUCA, 2001). O **ASD** não menciona sugestão sobre como deve acontecer a validação do sistema. A Tab. 7 apresenta a especificação das atividades relacionadas à validação do sistema.

Tabela 7 – Validar Sistema

Metodologia	Práticas
XP	O sistema é validado continuamente, pois a integração acontece a cada nova funcionalidade. É revalidado pelo cliente antes do encerramento do projeto
Scrum	Ao final da última *sprint* o sistema é validado pelo cliente
FDD	Através do teste de integração após a elaboração de todos os requisitos
ASD	Sem especificação sobre a validação do sistema

4.8 Entrega Final

O **XP** entrega o projeto quando o cliente estiver satisfeito e os requisitos, desenvolvidos por um todo (BECK, 2000). O **Scrum** acaba seu desenvolvimento quando os requisitos do *Product Backlog* acabam (SCHWABER; BEEDLE, 2002). Assim que todos os requisitos foram projetados e construídos, o **FDD** finaliza o desenvolvimento do sistema (HIGHSMITH, 2000; ABRAHAMSSON, 2002). O **ASD** entrega o produto final quando não mais houver requisitos (HIGHSMITH, 2002). A Tab. 8 apresenta como a entrega do projeto é executada.

Tabela 8 – Entrega Final

Metodologia	Práticas
XP	Quando o cliente estiver satisfeito
Scrum	Quando o *Product Backlog* chega ao fim
FDD	Quando todas as atividades estiverem concluídas
ASD	Finalização do desenvolvimento de todos os requisitos do sistema

O comparativo elaborado nesse capítulo fundamentou a seleção das práticas que comporão a sugestão para elaboração da abordagem metodológica sugerida.

5 SUGESTÃO DO MODELO DE PROCESSO

A partir do estudo apresentado no capítulo anterior, será apontado um processo de desenvolvimento de software para cada fase do Desenvolvimento Incremental. A escolha das etapas que compõem esta sugestão baseia-se nos conhecimentos adquiridos através de leitura e das necessidades encontradas no desenvolvimento de aplicações-testes.

Dentre todas as metodologias ágeis estudadas, as quatro já mencionadas no comparativo foram a que mais se adaptaram às necessidades do desenvolvimento de aplicações interativas para TV Digital. A XP auxilia a construção do aplicativo através das suas premissas de desenvolvimento que pregam, por exemplo: um projeto simples, o uso da metáfora, testes contínuos, a programação em par, a refatoração, a propriedade coletiva, a comunicação freqüente, o *feedback* contínuo, o período de trabalho de, no máximo, 40h semanais entre outras. A *Scrum* leva grande vantagem no gerenciamento de projetos. O FDD é vantajoso nos pequenos ciclos e no projeto da arquitetura do sistema. O ASD é representado através das sessões JAD. A Tab. 9 destaca, resumidamente, quais foram os processos escolhidos para cada fase do Desenvolvimento Incremental, bem como os processos que motivaram tais escolhas.

Tabela 9 – Processos escolhidos para elaboração do Modelo proposto

Fase	Metodologia	Processo
Esboço dos requisitos	ASD	Sessão JAD
Requisitos às Iterações	FDD	Ciclos de 2 semanas
Arquitetura do Sistema	FDD	Diagramas UML
Desenvolvimento dos Incrementos	XP	12 práticas
Validação dos Incrementos	ASD	Sessão JAD
Integração dos Incrementos	XP	Continuamente
Validação do Sistema	ASD	Sessão JAD
Entrega Final	ASD	Sessão JAD

5.1 Valores e Papeis da Metodologia Proposta

A metodologia proposta nesse trabalho possui quatro valores que norteiam todo o processo de desenvolvimento, são eles:

a) Comunicação - o projeto deve priorizar a abertura de um amplo canal de comunicação que privilegie o contato interno de sua equipe como também com o cliente. Um ambiente de trabalho de uma equipe entrosada e com facilidade para comunicar-se aumenta, consideravelmente, a probabilidade de resultados mais proveitosos. A expectativa de sucesso em atingir os objetivos do projeto aumenta conforme o nível de comunicação dentro da equipe e dessa para com os clientes.
b) Simplicidade - um projeto simples é aquele onde as rotinas são otimizadas a tal ponto que um pequeno trecho consegue executar as funcionalidades de um código mais extenso.
c) Dinamismo - esse valor engloba conceitos como adaptabilidade, inteligência e coragem. Uma equipe dinâmica tem maior facilidade para

traspassar obstáculos, pois acaba se moldando conforme o projeto necessita. O posicionamento frente a dificuldades é o que caracteriza esse valor. A equipe dinâmica é aquela que busca saídas rápidas e eficientes para contornar os problemas encontrados.

d) Respeito - esse valor está intimamente ligado à comunicação. Através da cordialidade da equipe para com seus colegas e clientes o ambiente de trabalho fica mais propício ao cumprimento de suas obrigações.

Os papéis da metodologia sugerida assemelham-se muito aos papéis das metodologias propostas como base, pois todas elas têm como papéis principais um gerente, a equipe de desenvolvimento e os clientes/usuários. Os papéis da abordagem proposta são:

a) Gerente ou facilitador - é o responsável pelo andamento da equipe. Quando algum membro está enfrentando alguma dificuldade é para o gerente que deve manifestar suas insatisfações, independente, da esfera em que o problema se encontra.
b) Equipe de desenvolvimento - é responsável pela construção do sistema. O gerente escolhe alguns membros da equipe para que tomem papéis diferentes durante a fase de esboço dos requisitos, como anotador e gerente de tempo.
c) Cliente - caracteriza tanto o patrocinador do projeto, quanto os usuários do sistema proposto. Caso esses papéis sejam distintos a primeira reunião será feita com o patrocinador e as demais reuniões com os futuros usuários, contudo o *feedback* é constante para ambos.

O gerenciamento do projeto cabe a metodologia Scrum com enfoque na figura do *Scrum Master*. Essa escolha se deve ao fato de que o *Scrum Master* reúne em si as características de gerente, facilitador e mediador. Está sempre motivando os projetistas. Procura sempre aumentar o nível de comunicação entre os *stakeholders*, porém preservando os desenvolvedores de intervenções externas desnecessárias. Transfere parte da responsabilidade para a equipe. Está sempre preocupado com as condições de trabalho do grupo e com as dificuldades enfrentadas por cada um.

5.2 Ciclo de Vida Sugerido

O ciclo de vida desta metodologia (Fig.5.1) é curto e de fácil entendimento. Inicia pelo esboço dos requisitos, após o sistema terá suas iterações definidas e a arquitetura do sistema projetada. As fases seguintes envolvem o desenvolvimento dos incrementos seguidos de validação individual e final. Ao final acontece a entrega do projeto.

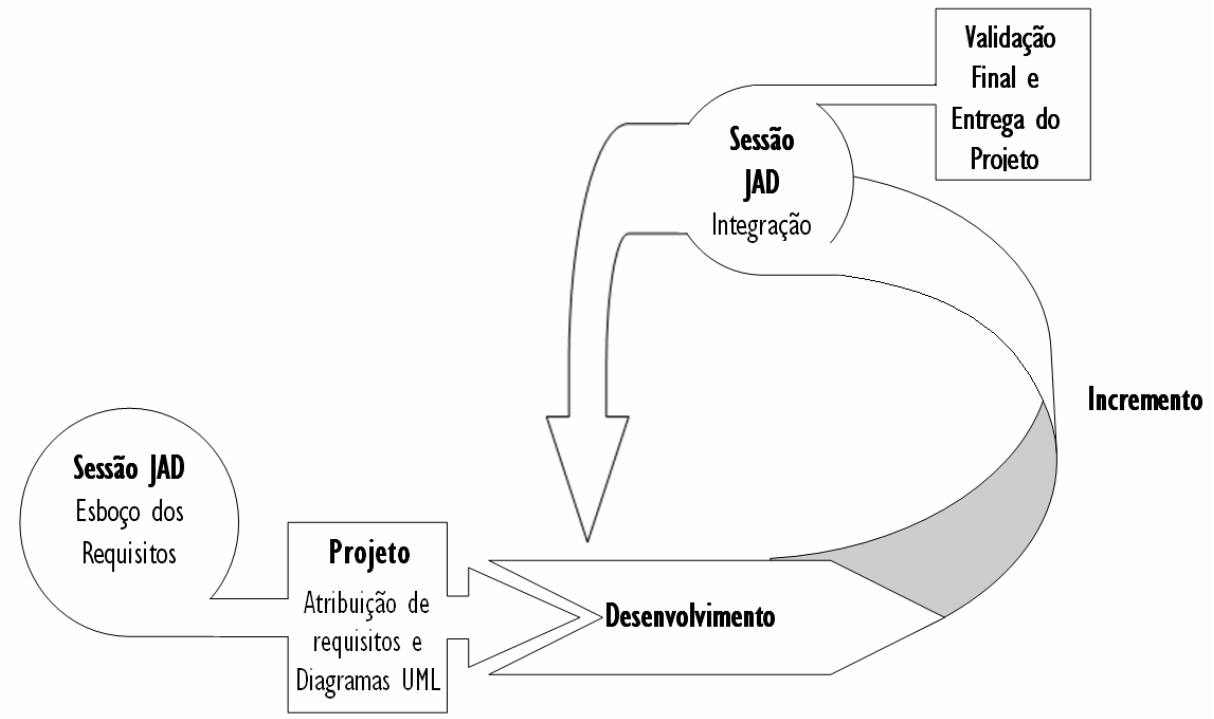

Figura 5.1 – Ciclo de Vida da Metodologia proposta

A seguir, serão apresentadas com um maior detalhamento todas as fases apresentada nesse Ciclo de Vida, quando aplicadas em um projeto desenvolvido durante a execução desse trabalho. Para esboçar esta validação foi desenvolvida uma aplicação interativa observando cada item proposto nessa seção. A aplicação constitui um jogo de cunho educativo que executa em primeiro plano enquanto a camada de vídeo continua exibindo a programação normal da emissora.

5.2.1 Definição do esboço dos requisitos

A metodologia ASD é indicada para compor essa etapa do projeto. O processo JAD é um conjunto de ferramentas facilitadoras para descoberta dos requisitos do sistema. Uma equipe JAD é composta por (JENNERICH, 2008):

a) Facilitador - geralmente o líder do projeto, deve ouvir os questionamentos e sugestões de todos os presentes na reunião com responsabilidade e autoridade.
b) Gestor - é o patrocinador do projeto.
c) Clientes - compõem o público-alvo do projeto. Um cliente também pode ser o gestor. São eles que entendem as necessidades do sistema e colaboram para o seu aperfeiçoamento.
d) Anotador - registra o que foi acertado durante a sessão JAD. Tudo o que foi dito durante a reunião deve ser anotado. A utilização de ferramentas que comprovem tudo o que está escrito pelo anotador é imprescindível, por exemplo, gravadores.
e) Gerenciador de tempo - monitora o andamento das sessões baseado na estimativa de tempo feita previamente, avisando a equipe quando o cada assunto deverá ser encerrado.

O processo do JAD está baseado em sessões, mas o sucesso da aplicação desse conjunto de ferramentas também se deve a pré-definição e término das reuniões. Antes do primeiro contato da equipe o facilitador encontra-se com o gestor para que fique inteirado do escopo do projeto.

Para servir como referência à equipe o gestor e o facilitador criam um quadro do projeto destacando quais são os objetivos principais do sistema, já prevendo uma ordem cronológica abstrata dos requisitos a serem desempenhados pelos programadores.

A condução das sessões JAD é deveras importante, já que perder o foco em uma reunião acontece facilmente. Para que a reunião esteja sempre sob o enfoque previsto a sessão JAD dispõe das seguintes técnicas:

a) O facilitador tem caráter neutro, de ouvinte. Porém pode fazer o redirecionamento dos assuntos abordados quando nota que está saindo do planejado. Também deve ter um papel de mediador para que o debate durante a reunião se dê de forma democrática.
b) O gerenciador de tempo evita que alguma discussão tome o tempo de outros assuntos.

c) O quadro do projeto tem o objetivo de direcionar as reuniões somente aos assuntos de importância imediata, evitando que aspectos futuros sejam tema de discussões sem necessidade.

Na primeira sessão os papéis de cada membro da equipe são explicitados. Todos os membros devem estar presentes, incluindo o gestor. A última sessão servirá como sessão de finalização, onde os objetivos iniciais devem ser analisados, com o intuito de se averiguar a conclusão de todos.

As sessões JAD têm a seguinte estrutura geral:

a) Abertura- apresentação dos tópicos a serem discutidos.
b) Discussões- discussão dos tópicos em seqüência, respeitando a estimativa de tempo.
c) Finalização- considerações finais e próximos passos.
d) Após as sessões, as anotações do anotador devem ser transformadas em uma ata e disponibilizadas para todos os membros da equipe. Um espaço entre as reuniões é proveitoso, porque as idéias discutidas devem ser aprimoradas.

5.2.2 Atribuição dos Requisitos

Para essa etapa será utilizada a agilidade da metodologia FDD com os requisitos divididos em ciclos curtos de no máximo duas semanas. Os requisitos vão ser decompostos em funcionalidades agrupadas em áreas de interesse, auxiliados pelo cronograma previamente estabelecido.

O líder do projeto e os desenvolvedores reúnem-se para que seja elaborado um diagrama de casos e o diagrama de classes com foco na forma do modelo, isto é, quais classes estão no domínio, como estão conectadas. Os métodos e atributos identificados são colocados nessas classes. Deve haver a elaboração de diagrama de seqüência, caso haja necessidade. Além da redação dos comentários sobre o modelo escolhido e quais alternativas poderiam ter sido consideradas.

Logo após é construída uma Lista de Funcionalidades, isto é, a decomposição funcional do modelo do domínio. Existirá uma lista de atividades pra

cada área de interesse. E para cada atividade deverá existir uma funcionalidade que satisfaça o problema. O resultado é uma hierarquia de funcionalidades que representa o produto a ser construído

A equipe deve atribuir uma data para o término de cada atividade. Esta data deve estar baseada em dependências entre as funcionalidades, distribuição da carga de trabalho, complexidade das funcionalidades. O produto final desta etapa é o plano de desenvolvimento, com pacotes de trabalho ordenados conforme necessidade do cliente.

5.2.3 Projeto da Arquitetura do Sistema

A metodologia FDD foi a escolhida para essa etapa, pois cria uma visão mais ampla da estrutura que o projeto deverá seguir, devido à construção dos diagramas de Casos de uso, de Classes e de Seqüência, quando necessário.

5.2.4 Desenvolver Incremento

A fase Desenvolver Incremento contará com as conhecidas práticas de desenvolvimento da metodologia XP. Há uma sinergia muito grande quando se analisa o conjunto de todas as técnicas dessa metodologia:

a) Pequenas Versões - pequenas versões funcionais auxiliam o cliente no processo de aceitação;
b) Metáfora - comunicação com o cliente de forma simples.
c) Projeto Simples - simplicidade é um princípio básico do XP. O código simples não significa ser aquele fácil de desenvolver, mas sim o que executa uma operação da forma menos complexa e com maior qualidade possível.
d) Time Coeso - a equipe de desenvolvimento é formada pelo cliente e pela equipe de desenvolvimento, onde existe um bom canal de comunicação, acrescido de um bom relacionamento, dinamismo e respeito entre todos os envolvidos.

e) 40 horas semanais - A necessidade de mais horas de desenvolvimento denota problema no planejamento das iterações e deve ser revisto.
f) Reuniões em pé - são mais rápidas e menos dispersivas.
g) Posse Coletiva - todos desenvolvedores podem editar o código.
h) Programação em Pares - programação em dupla no mesmo computador. Geralmente, um iniciante operando a máquina juntamente com um mais experiente servindo de instrutor. O código acaba sendo revisado por duas pessoas, diminuindo a incidência de erros.
i) Padrões de Codificação - A equipe seguir regras estabelecidas para o desenvolvimento da aplicação.
j) Refatoração - permite a melhoria constante da codificação. Quando se encontra algum trecho passível de aperfeiçoamento, a refatoração é executada.
k) Desenvolvimento Orientado a Testes (*Test Driven Development* – TDD) - Primeiro são criados os testes unitários, logo após é criado o código apropriado a esses testes. De acordo com Beck (2002) para que se consiga extrair todo o proveito desta prática deve-se:
   - Criar um teste – antes de cada funcionalidade ser desenvolvida deve-se criar um teste. Esse teste deverá ser composto por todos os possíveis casos de uso.
   - Executar todos os testes - a rotina de testes deve estar funcionando corretamente e o novo teste deverá encontrar todas as possíveis falhas existentes na funcionalidade pretendida, antes da implementação da referida funcionalidade.
   - Escrever o código - o código que irá passar no teste, escrito anteriormente, deverá ser desenvolvido, sem se preocupar em torná-lo elegante/otimizado. É muito importante que o código implementado reflita somente o teste escrito.
   - Executar novamente todos os testes – quando todos os testes forem satisfeitos se terá certeza que essa nova funcionalidade não criou problemas de integração.

5.2.5 Validar os Incrementos

A metodologia que Validará os Incrementos será a ASD, por meio das sessões JAD. Ao término de cada iteração será feito um teste de aceitação com o cliente e um teste de unidade pelos desenvolvedores.

5.2.6 Integrar os Incrementos

A fase Integrar Incremento é feita baseado na XP. A integração acontece paralelamente ao desenvolvimento dos requisitos, o que torna o processo de integração uma tarefa mais fácil, pois a cada nova funcionalidade produzida já há um teste parcial, evitando assim um problema de incompatibilidade comparando-se ao procedimento de incremento ao final de cada iteração. A integração contínua dificulta a ocorrência de erros no código e dá o status real do projeto.

5.2.7 Validação Final do Sistema e Entrega Final

As etapas de Validar Sistema e Entrega Final utilizarão os processos da metodologia ASD, por meio das sessões JAD. Ao término de iteração final será feito um teste de aceitação com o cliente e validação pelos desenvolvedores juntamente com a revisão dos objetivos e do projeto como um todo.

5.3 Estudo de Caso

A validação se dará através do retrato das experiências advindas do processo de desenvolvimento de uma aplicação interativa para TVDI que sugere um QUIZ[10], onde quatro animais estão associados a quatro cores presentes no controle remoto. Para cada acerto, uma nova pergunta. Caso o usuário erre, uma mensagem de erro aparece na tela. A aplicação termina quando o usuário pressiona a tecla Exit no controle remoto. A codificação de tal aplicação encontra-se no Apêndice A.

[10] Quiz é um jogo de perguntas e respostas, na maioria das vezes para testar seus conhecimentos.

Para a elaboração desta aplicação as práticas sugeridas nesse capítulo foram, em grande parte, cumpridas. A seguir, cada fase do Desenvolvimento Incremental e como os processos se adaptaram sob os preceitos desta nova abordagem.

O esboço dos requisitos será definido baseado nas sessões JAD. Segue a seguinte estrutura:

Esboço dos Requisitos

a) Definição do projeto
   - Quadro do projeto (Fig. 5.2);

b) Seleção da equipe JAD

c) Sessão JAD inicial
   - Definição dos papéis e objetivos do projeto;

d) Sessões JAD (planejamento)
   - O que é esperado ao final do projeto;
   - Como o projeto será executado;
   - Elaboração de um cronograma;

Validar Incremento

e) Sessões JAD (desenvolvimento)
   - Acompanhamento do projeto
   - Acertos de rumo do projeto

Entrega Final

f) Sessão JAD final (finalização)
   - Revisão dos objetivos atingidos
   - Revisão do projeto

-

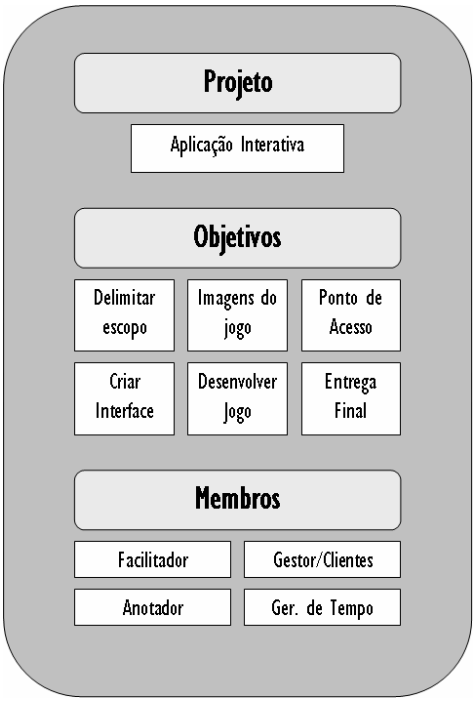

Figura 5.2 – Quadro do projeto

Através do levantamento da fase anterior a equipe deve identificar o conjunto de funcionalidades que devem ter maior precedência perante as outras. Então, cria-se uma Lista de Negócios. Para cada área que compõe a lista de negócios é elaborada uma Lista de Atividades. Para cada atividade, uma funcionalidade que atenda suas necessidades. Pode-se imaginar como áreas de negócio a Interface e a Jogabilidade. As atividades relacionadas à interface são, por exemplo, escolher figuras e dispor as imagens na tela. O desenvolvimento é a atividade atrelada à área de jogabilidade.

As sessões JAD, bem como todas as rotinas de desenvolvimento foram todas executadas pelo próprio desenvolvedor, procurando sempre tornar tais reuniões mais próximas do real, simulando faltas e desvios de assuntos durante as reuniões.

As Fig. 5.3 e 5.4 traduzem graficamente o projeto da arquitetura do sistema, através das práticas da metodologia FDD. A Fig. 5.3 traz o Diagrama de Casos de Uso e a Fig. 5.4, o Diagrama de Classes.

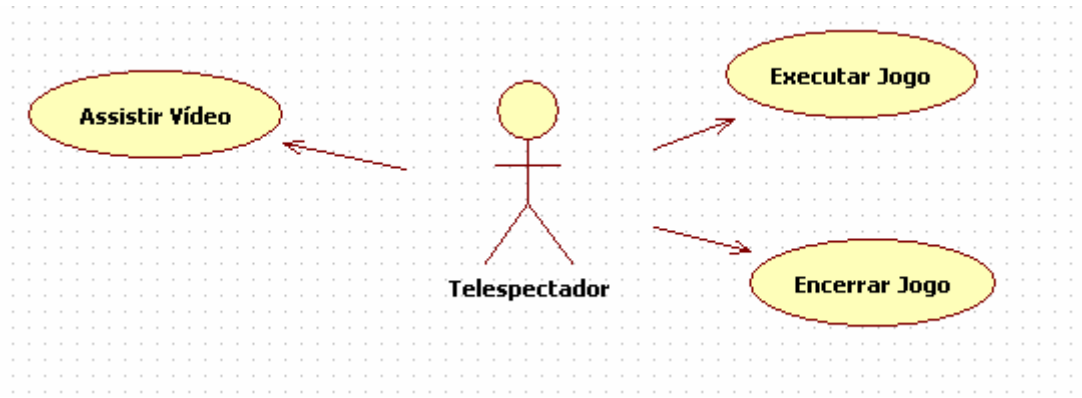

Figura 5.3 – Diagrama de Casos de uso

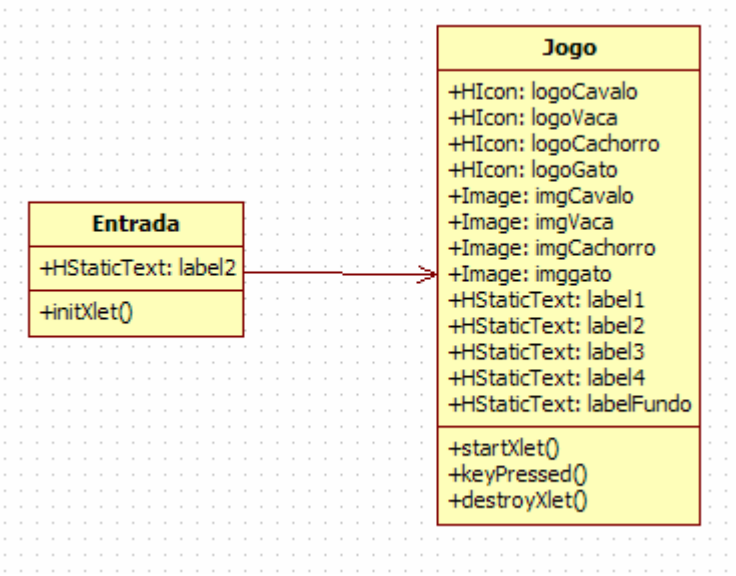

Figura 5.4 – Diagrama de classes

Como no ambiente televisivo a velocidade de informação é o mais importante, deve-se ter um processo de desenvolvimento que acompanhe tal velocidade. A aplicação dos métodos do XP auxilia a construção de tais aplicações, evidenciado pela interligação de certas características, tais como: programação em dupla e projeto simples; pequenas versões e *feedback* constante; time coeso, refatoração e propriedade coletiva dentro outras.

A experiência advinda do processo de desenvolvimento mostra que as entregas freqüentes são propiciadas pelas pequenas versões alocadas em ciclos muito curtos, de no máximo duas semanas. Através da programação em dupla é possível refatorar o projeto durante toda a sua construção. Programação em dupla

também torna o desenvolvimento mais dinâmico e ágil, pois dois programadores conseguem encontrar os erros mais rapidamente, como também deparar-se com possíveis saídas de uma maneira menos complexa. Quando se desenvolve em dupla, novas técnicas podem ser exploradas, além de quem um projetista ajuda o outro a se manter focado, porém para isso o time deve estar coeso. Um projeto simples é encontrado mais rapidamente quando dois programadores pensam como mudanças podem ser implementadas visando o aperfeiçoamento do código.

Em relação ao horário de trabalho, o gerente deverá optar pelo que mais se molde ao projeto em questão, podendo ser em jornadas de 40 horas semanais ou por demanda de atividades. As reuniões devem ser em pé, devido a maior agilidade. A técnica de metáfora deve ser utilizada nas reuniões com os clientes ou usuários, isto é, usar uma linguagem clara e objetiva, ignorando o uso de termos técnicos e jargões da área.

O *feedback* deve ser constante para o cliente, para que se verifique o grau de satisfação relacionado ao andamento do projeto e à adição de novas funcionalidades. A codificação padrão é de grande valia no desenvolvimento, pois para que a integração dos incrementos se dê da melhor maneira possível o código deve ser especificado sob as mesmas regras. A integração contínua auxilia durante o desenvolvimento da aplicação, porque é uma maneira de se simular a integração de cada incremento ao passo que novos requisitos são criados. Uma técnica muito interessante e proveitosa quando bem aplicada é a TDD. Todavia a aplicação desta técnica não é uma tarefa corriqueira durante o projeto.

Ao final de cada incremento uma nova sessão JAD valida o código e averigua se o ciclo foi proveitoso. Essa reunião aponta os pontos positivos e negativos do incremento, criando um consenso geral para que se possa estimar quais são as melhorias necessárias para que a próxima iteração não pene em nenhuma parte.

Quando os desenvolvedores encerram a última iteração a última sessão JAD é executada, analisando o código através de testes de unidade pelos projetistas e testes de aceitação por parte dos usuários e patrocinadores. Caso seja aceito por ambas as partes o sistema é entregue e o projeto é dado por encerrado.

A seguir as Figs 5.5, 5.6, 5.7, 5.8 e 5.9 mostram, respectivamente, uma seqüência das telas da aplciação desenvolvida. A Fig. 5.5 mostra a mensagem que aparecerá na tela quando o Xlet foi inicializado. A Fig. 5.6 representa o primeiro questionamento do QUIZ. A Fig. 5.7 é a tela seguinte que indica o acerto do usuário. A Fig. 5.8 é a tela que indica o erro, a pergunta permanece estática na tela, até o telespectador apertar o botão colorido que representa o animal. A Fig. 5.9 apresenta a última tela do jogo, onde finaliza perguntando se o usuário deseja termina o jogo ou recomeçá-lo.

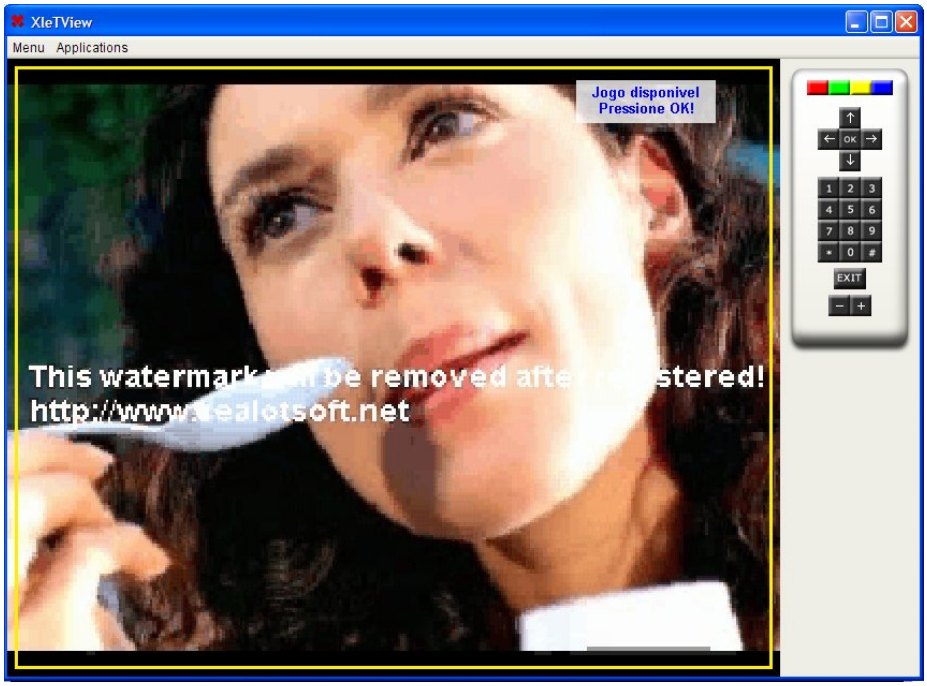

Figura 5.5 – Tela Inicial, quando o Xlet é iniciado a mensagem de “Jogo Disponível!” aparece na tela. Se o botão OK for pressionado o jogo encaminha-se para a próxima tela.

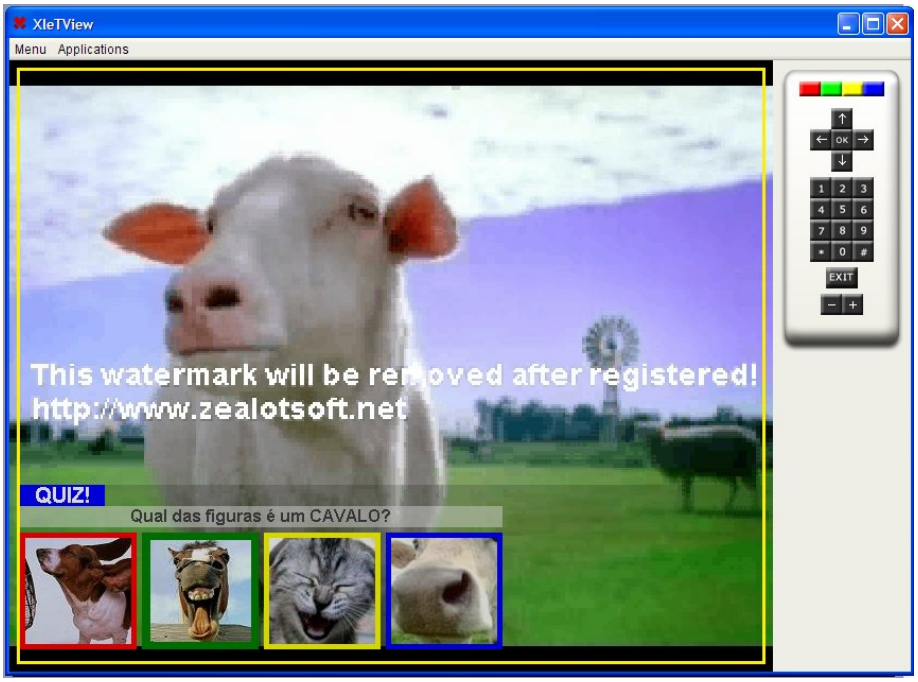

Figura 5.6 – Primeira pergunta do jogo

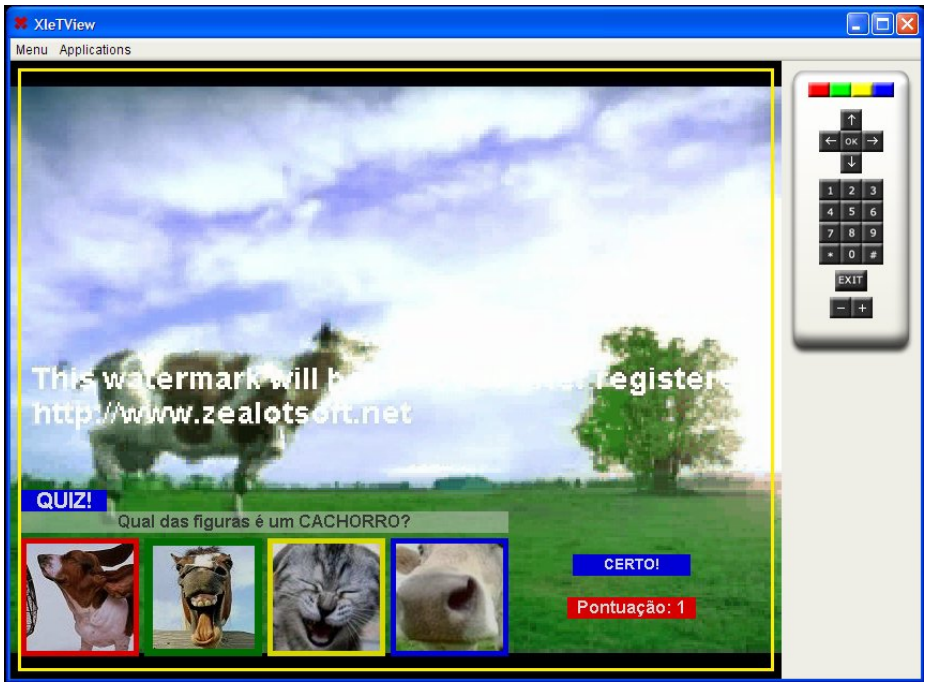

Figura 5.7 – Tela de acerto, quando o usuário pressiona no controle remoto a tecla que corresponde à borda da figura do animal, uma mensagem de que o usuário acertou aparece na tela e a pontuação é incrementada.

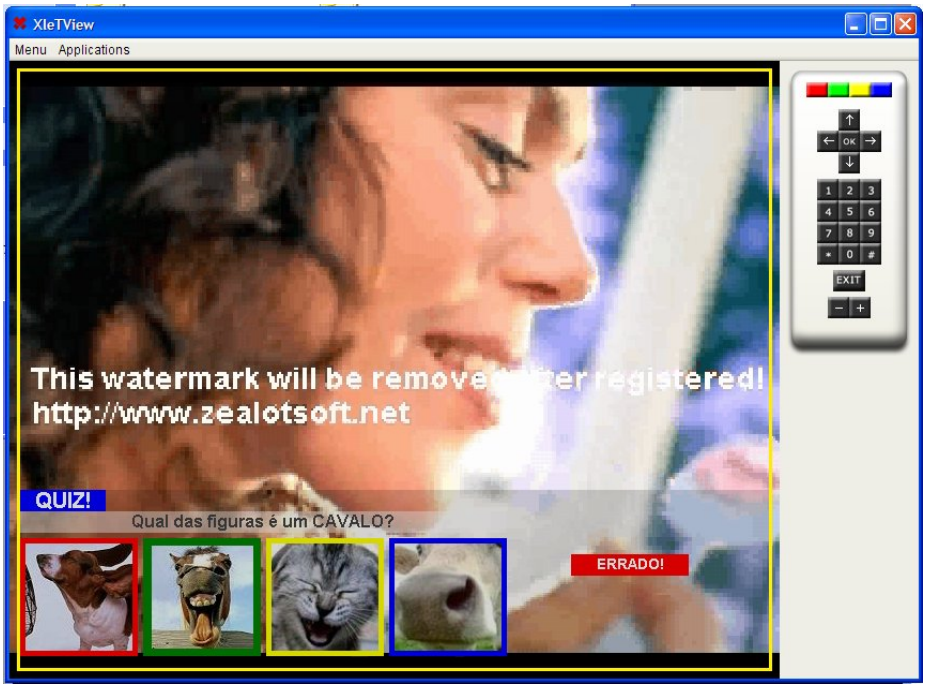

Figura 5.8 – Tela de erro, quando o usuário pressiona qualquer uma das 3 teclas coloridas que não estão associadas ao animal da pergunta.

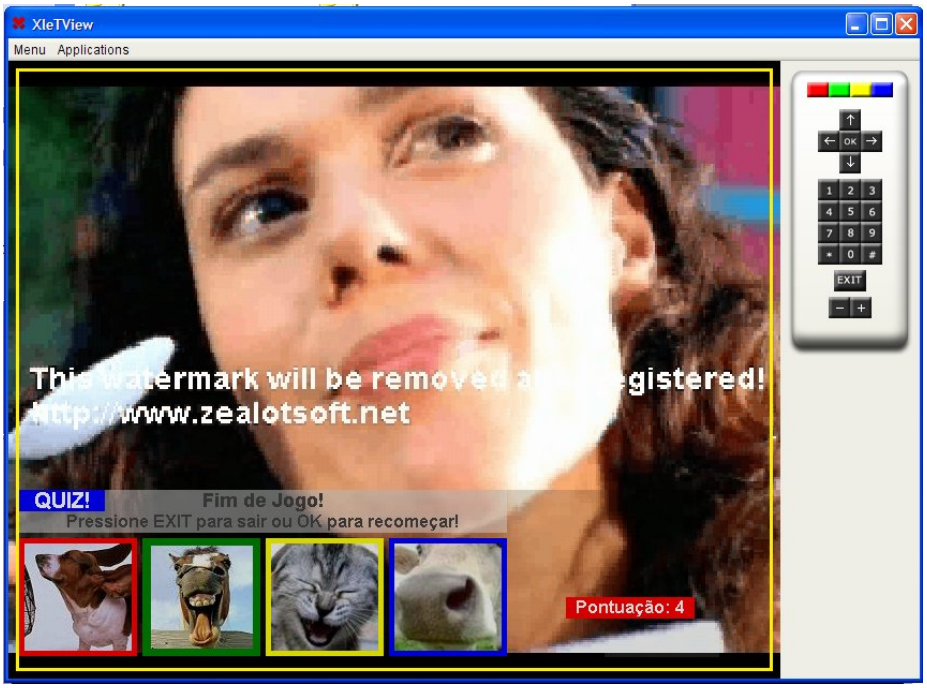

Figura 5.9 – Tela de fim, quando o usuário acerta a última pergunta essa é a tela mostrada ao telespectador. Pressionando EXIT a aplicação é destruída, caso o OK seja escolhido o jogo inicia da primeira pergunta.

Com o final desse capítulo pode-se constatar que o desenvolvimento de aplicações interativas direcionadas a TVDI agora tem um aporte definido que facilita a obtenção de aplicações mais bem elaboradas. Os processos escolhidos que compõem esse modelo têm o papel de mitigar todas as dificuldades encontradas na produção dos softwares criados para a grade televisiva que utilizam o canal de retorno.

6 CONCLUSÃO

Esse trabalho propôs a análise de toda a estrutura de um projeto direcionado ao desenvolvimento de aplicações interativas em TV Digital com o intuito de evidenciar qual a metodologia seguida. Foi constatado que ainda não se tem um processo de desenvolvimento de software definido para o ambiente televisivo, o que favorece os altos gastos e o descumprimento de prazos pré-fixados.

Os modelos de processos conhecidos não contemplam todas as áreas do projeto de software em TVDI, pois algumas práticas de tais metodologias não acompanham o ritmo de desenvolvimento, o grau de comunicação necessário dentro da equipe ou para com os clientes. Outras não possuem rotinas bem claras em relação à obtenção dos requisitos do sistema e da elaboração da arquitetura e da distribuição dos ciclos ao longo do projeto. É notória a falta de rotinas de desenvolvimento mais bem especificadas, que sirvam de base para a boa prática das metodologias propostas.

A abordagem metodológica sugerida nesse trabalho contempla todas as áreas de um projeto interativo, extraindo de cada metodologia a prática que mais convém para suprir a necessidade de cada fase do Desenvolvimento Incremental. Com isso, esse cenário, ainda desprovido de uma normalização das práticas relacionadas ao bom andamento do trabalho, ganha um Modelo de Processo apto a minorar problemas advindos do desenvolvimento de projetos para a TVDI.

A partir do que foi estudado para a elaboração desse trabalho pôde-se concluir que as metodologias de software têm um comportamento semelhante nas diversas fases do desenvolvimento incremental, o que motivou as escolhas distintas entre as metodologias foram pequenas características que se adaptaram melhor ao

contexto dos processos de desenvolvimento. Conclui-se que a metodologia *Extreme Programming* tem as melhores premissas de desenvolvimento, pois dá enfoque direto à agilidade do projeto, mantendo-se adaptada do início ao fim do desenvolvimento. A metodologia SCRUM, como já conhecido, é direcionada ao gerenciamento, pois privilegia a equipe com a figura de um gerente altamente preocupado e atuante com todos os aspectos relacionados a sua equipe de trabalho. A rotina da metodologia *Adaptive Software Development*, sessão JAD, é a que mais se adaptou à obtenção de requisitos, bem como sua mineração entre os incrementos e como parte fundamental na validação final de cada ciclo. Como o processo de incremento simultâneo ao final de cada nova funcionalidade desenvolvida, advindo da metodologia XP, foi utilizado nessa nova metodologia proposta, as sessões JAD tornaram-se mais proveitosas, já que têm papel, principalmente, de uma análise qualitativa de como foi a etapa do projeto em questão, dando um enfoque mais relacionado ao aproveitamento de tempo, relacionamento da equipe e se os objetivos traçados foram alcançados. A rotina de incremento e validação paralelos minora a possibilidade da ocorrência de erros, pois as sessões JAD não têm esse enfoque especificado. A metodologia *Feature Driven Development* é a única que, realmente, tem um processo de definição da arquitetura do projeto bem definido através dos diagramas UML, o que facilita à equipe acompanhar e não perder o foco durante a execução do projeto.

6.1 Trabalhos Futuros

Tomando-se esse trabalho como base, novas aplicações mais complexas podem ser desenvolvidas, influenciadas pelas premissas propostas nesse trabalho, para que a escolha dos processos de desenvolvimento seja ratificada. As mudanças, principalmente, ligadas ao ambiente televisivo, alvo desse trabalho, são iminentes e constantes, portanto essa metodologia poderá passar por mudanças a fim de que se molde da melhor forma possível ao desenvolvimento de aplicações direcionadas às novas tecnologias em TVDI. A possibilidade da metodologia ser adaptada aos processos relacionados a toda construção de um projeto de TV Digital Interativa a torna mais promissora, já que todas as tarefas ligadas aos custos do projeto, a qualidade do software desenvolvido e ao planejamento de tempo podem estar alicerçados em uma metodologia mais ampla e adaptada.

REFERÊNCIAS

ABRAHAMSSON, P.; WARSTA, J.; SIPONEN, M.T.; RONKAINEN, J. **New Directionson Agile Methods: A Comparative Analysis**. 2002.

AMBLER, S. **Modelagem ágil**: **práticas eficazes para a Programação Extrema e o Processo Unificado**. Bookman. 2004.

ANDERSON, D. J.. **Agile Management for Software Engineering, Applying the Theory of Constraints for Business Results**. Prentice Hall. 2003.

BECK, Kent. **Extreme Programming Explained**. Addison-Wesley. 2000.

BECK Kent. **Programação Extrema Explicada**. Bookman. 1999.

BECK, Kent. **Tests-Driven Development by Examples**. Addison-Wesley. 2002.

BECK, K. et al. **Agile Manifesto**. 2001. Disponível em: http://www.agilemanifesto.org. Acesso em: 24 jul. 2008.

BEEDLE, M; et al. **SCRUM: An extension pattern language for hyperprodutive software development**. 1998.

BOEHM, B. **A View of 20th and 21st Century Software Engineering**, ICSE. 2006.

BOEHM, B.; TURNER, R. **Balancing Agility and Discipline A Guide for the Perplexed**. AddisonWesley. 2003.

BONA, Cristina. **Avaliação de Processos de Software: Um Estudo de Caso em XP e ICONIX**. 2002. Dissertação (Mestrado em Engenharia de Produção) - Universidade Federal de Santa Catarina, Santa Catarina. 2002.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML, guia do usuário**. Rio de Janeiro. Campus. 2000.

BROOKS, F. **No Silver Bullet: Essence and Accidents of Software Engineering**. 1987. p. 10-19.

CHARETTE, R. **Fair Fight? Agile Versus Heavy Methodologies**. Cutter Consortium E-project Management Advisory Service. 2001.

COCKBURN, A.; HIGHSMITH, J. **Agile Software Development: The Business of Innovation**, IEEE Computer. 2001. p. 120-122.

COHN, Mike. **Agile Estimating and Planning**. Prentice Hall. 2006. 330 p.

DE LUCA, J. **Feature-Driven Development (FDD) Overview Presentation**. 2002.

Disponível em: http://www.nebulon.com/articles/fdd/download/ fddoverview.pdf . Acesso em: 10 jul. 2008.

DIGITAL ÁUDIO-VIDEO COUNCIL. Disponível em: http://www.davic.org Acesso em: 15 jul. 2008.

DIGITAL VÍDEO BROADCASTING. Disponível em: http://www.dvb.org Acesso em: 15 jul. 2008.

GILB, T. **Principles of Software Engineering Management**. Addison-Wesley. 1988.

HIGHSMITH, J. **Agile Project Management, Creating innovative products**. AddisonWesley. 2004.

HIGHSMITH, J. **Agile Software Development Ecosystems**. Boston: Addison Wesley, 2002.

HIGHSMITH, J.; ORR, K.; COCKBURN, A. **Extreme Programming**. E-Business Application Delivery. 2000. p. 4-17.

HOME ÁUDIO-VIDEO COUNCIL. Disponível em: http://www.havi.com. Acesso em: 15 jul. 2008.

HUMPHREY, Watts. **Managing the Software Process.** Addison-Wesley Publishing,Company. 1990.

INTRODUCTION TO DIGITAL TV APPLICATIONS PROGRAMMING. Disponível em: http://java.sun.com/developer/technicalArticles/javatv/apiintro. 2001. Acesso em: 20 ago. 2008.

JALOTE, P. **An integrated approach to software engineering.** New York: Springer-Verlag, 1997.

JAVATV TECHNOLOGY. Disponível em: http://java.sun.com/products/javatv. Acesso em: 17 nov. 2003.

JENNERICH, Bill. **Joint Application Design - Business Requirements Analysis for Successful Re-engineering**. Disponível em: http://www.bee.net/bluebird/jaddoc.htm. Acesso em 15 set. 2008.

JUNOT, Régis Alvim. TV Digital Interativa - O Ponto de Partida. Disponível em: http://www.via.multimidia.nom.br/tvdi/tvdi_o_ponto_de_partida.pdf. Acesso em 13 jun. 2008.

KNIBERG, Henrik. **Scrum and XP from the Trenches, How we do Scrum**. 2006. 90 p.

LOUREIRO, Janine de Aguiar. **Interface de Programação para o Desenvolvimento de Aplicações para TV Digital**. Universidade de Pernambuco. 2004.

MOUNTAIN Goat. **The Scrum Development Process**. Disponível em: http://www.mountaingoatsoftware.com/Scrum. Acesso em: 15 jun. 2006.

PRESSMAN, R. **Engenharia de Software**. McGraw-Hill. 2001.

PRESSMAN, R., **Software engineering: A practitioner's approach**. McGraw-Hill. 1997. p. 22-53.

REIS, Christian. **Caracterização de um Modelo de Processo para Projetos de Software Livre**. 2001. Dissertação (Mestrado em Ciências da

Computação e Matemática Computacional)- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Paulo.

REZENDE, Denis A. **Engenharia de Software e Sistemas de Informação**. 2002.

RIBEIRO, Nuno. **Tecnologia de Informação: Multimídia e Tecnologias Interativas**. Lisboa: FCA Editora de Informática. 2004.

ROYCE, W. W. **Managing the development of large software systems**. 1970. p. 1-9.

SCHWABER, K. **Scrum Development Process**. Springer-Verlag. 1995.

SCHWABER, K.; BEEDLE, M. **Agile Software Development with Scrum**. Prentice-Hall. 2002.

SOMMERVILLE, Ian. **Engenharia de Software**. Editora Addison-Wesley. 2003. 592p.

SOMMERVILLE, Ian**. Software Engineering**. 6th edition. 2000.

TEAMSYSTEM. **Scrum For Team System**. 2005. Disponível: http:www.scrumforteamsystem.com. Acesso em: 17 jul. 2008.

VEIGA, Elba Guimarães; TAVARES, Tatiana Aires. **Um Modelo de Processo para o Desenvolvimento de Programas para TV Digital e Interativa baseado em Metodologias Ágeis**. 2007. 8p.

WAKE, William. **Extreme Programming Explored**. Addison-Wesley. 2002.

XLETVIEW. Disponível em: http://www.xletview.org/ . Acesso em: 23 jul. 2008.

XPLANNER. Disponível em: http://www.xplanner.org. Acesso em: 20 jul. 2008.

APÊNDICE A

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.ImageIcon;
import org.dvb.ui.DVBColor;
import org.havi.ui.*;
import xjavax.tv.xlet.Xlet;
import xjavax.tv.xlet.XletContext;
import xjavax.tv.xlet.XletStateChangeException;

public class ExemploXlet extends HContainer implements Xlet, KeyListener {
   private XletContext context;
   private HScene scene;
   private HStaticText label1, label2, label3, label4, labelFundo;
   private HIcon logoVaca, logoCachorro, logoCavalo, logoGato, logoVermelho, logoVerde, logoAmarelo, logoAzul;
   private Image imgVaca, imgCachorro, imgCavalo, imgGato, imgVermelho, imgVerde, imgAmarelo, imgAzul;
   private int est1=0, est2=0, est3=0, est4=0, sair=0, ok=0;
   private HDefaultTextLayoutManager manager;

   public ExemploXlet() {
   }

   public void initXlet(XletContext xletContext)
   throws XletStateChangeException {
      /* guardando o contexto... */
      this.context = xletContext;
      HSceneFactory hsceneFactory = HSceneFactory.getInstance();
      scene = hsceneFactory.getFullScreenScene(HScreen.getDefaultHScreen().
      getDefaultHGraphicsDevice());
      scene.setSize(800, 640);
      manager = new HDefaultTextLayoutManager();
      scene.setLayout(null);
      scene.addKeyListener(this); //o próprio Xlet é o listener
      label1 = new HStaticText("QUIZ!", 10,400,80,20,
      new Font("Arial_Black", 1, 22),
      Color.white, Color.blue, manager);
      imgCachorro = new ImageIcon("c://pacote/cachorro.jpg").getImage();
      logoCachorro = new HIcon(imgCachorro);
      logoCachorro.setSize(100,100);
      logoCachorro.setLocation(15,450);
      logoCachorro.setVisible(true);
      logoCachorro.addKeyListener(this);
      imgCavalo = new ImageIcon("c://pacote/cavalo.jpg").getImage();
      logoCavalo = new HIcon(imgCavalo);
      logoCavalo.setSize(100,100);
      logoCavalo.setLocation(130,450);
      logoCavalo.setVisible(true);
      logoCavalo.addKeyListener(this);
      imgGato = new ImageIcon("c://pacote/gato.jpg").getImage();
      logoGato = new HIcon(imgGato);
      logoGato.setSize(100,100);
      logoGato.setLocation(245,450);
      logoGato.setVisible(true);
      logoGato.addKeyListener(this);
      imgVaca = new ImageIcon("c://pacote/vaca.jpg").getImage();
      logoVaca = new HIcon(imgVaca);
      logoVaca.setSize(100,100);
      logoVaca.setLocation(360,450);
      logoVaca.setVisible(true);
      logoVaca.addKeyListener(this);
      imgVermelho = new ImageIcon("c://pacote/vermelho.jpg").getImage();
      logoVermelho = new HIcon(imgVermelho);
      logoVermelho.setSize(110,110);
      logoVermelho.setLocation(10,445);
      logoVermelho.setVisible(true);
      logoVermelho.addKeyListener(this);
      imgVerde = new ImageIcon("c://pacote/verde.jpg").getImage();
      logoVerde = new HIcon(imgVerde);
      logoVerde.setSize(110,110);
      logoVerde.setLocation(125,445);
      logoVerde.setVisible(true);
      logoVerde.addKeyListener(this);
```

```
    imgAmarelo = new ImageIcon("c://pacote/amarelo.jpg").getImage();
    logoAmarelo = new HIcon(imgAmarelo);
    logoAmarelo.setSize(110,110);
    logoAmarelo.setLocation(240,445);
    logoAmarelo.setVisible(true);
    logoAmarelo.addKeyListener(this);
    imgAzul = new ImageIcon("c://pacote/azul.jpg").getImage();
    logoAzul = new HIcon(imgAzul);
    logoAzul.setSize(110,110);
    logoAzul.setLocation(355,445);
    logoAzul.setVisible(true);
    logoAzul.addKeyListener(this);
}

public void startXlet()
throws XletStateChangeException {
    labelFundo = new HStaticText("", 10, 400, 700, 170,
    new Font("Tiresias", 1, 15),
    Color.blue,new DVBColor(0, 0, 0, 40),manager );
    String mensagem = "Jogo disponivel\nPressione OK!";
    label2 = new HStaticText(mensagem, 530, 20, 130, 40,
    new Font("Tiresias", 1, 15),
    Color.blue,new DVBColor(255, 255, 255, 200),manager );
    scene.add(label2);
    scene.setVisible(true);
    scene.requestFocus();
}
public void pauseXlet() { }
public void destroyXlet(boolean unconditional)
throws XletStateChangeException {

    if (scene!=null) {
        scene.setVisible(false);
        scene.removeAll();
        scene = null;
    }
    context.notifyDestroyed();
}
/* Método de java.awt.event.KeyListener */
public void keyTyped(KeyEvent keyevent) {
}
/* Método de java.awt.event.KeyListener */
public void keyReleased(KeyEvent keyevent) {
}
/* Método de java.awt.event.KeyListener */
public void keyPressed(KeyEvent e) {
    String mensagem = "";
    int codigo = e.getKeyCode();
            /*
             * 403 - vermelho
             * 404 - verde
             * 405 - amarelo
             * 406 - azul
             *
             * 27 - exit
             * * (asterisco) - 151
             * # (grade) - 520
             *
             * seta para cima - 38
             * seta para baixo - 40
             * seta para esquerda - 37
             * seta para direita - 39
             * ok - 10
             * números - número + 48 (ex. 2 é 50)
             */
   switch (codigo) {
        case 403:
            if(ok==1)
            {
                if(est1==1 && est2==1 && est3==0 && est4==0)
                {
                    mensagem = "Qual das figuras é uma VACA?";
                    label2 = new HStaticText(mensagem, 10,420,455,20,
                    new Font("Tiresias", 1, 18),
                    Color.DARK_GRAY, new DVBColor(250, 250, 250, 120),manager );
```

```
            mensagem = "CERTO!";
            label3 = new HStaticText(mensagem, 525, 460, 110, 20,
            new Font("Tiresias", 1, 16),
            Color.WHITE, Color.blue,
            new HDefaultTextLayoutManager());
            mensagem = "Pontuação: 2";
            label4 = new HStaticText(mensagem, 520, 500, 120, 20,
            new Font("Tiresias", 1, 18),
            Color.white, Color.red, manager);
            setar();
            est3=1;
         }
          else
         {
            setarElse();
         }
      }
         break;
   case 404:
      if(ok==1)
      {
         if(est1==1 && est2==0 && est3==0 && est4==0)
         {
            mensagem = "Qual das figuras é um CACHORRO?";
           label2 = new HStaticText(mensagem, 10,420,455,20,
            new Font("Tiresias", 1, 18),
            Color.DARK_GRAY, new DVBColor(250, 250, 250, 120),manager );
            mensagem = "CERTO!";
            label3 = new HStaticText(mensagem, 525, 460, 110, 20,
            new Font("Tiresias", 1, 16),
            Color.WHITE, Color.blue,
            new HDefaultTextLayoutManager());
            mensagem = "Pontuação: 1";
            label4 = new HStaticText(mensagem, 520, 500, 120, 20,
            new Font("Tiresias", 1, 18),
            Color.white, Color.red, manager);
            setar();
            est2=1;
         }
         else
         {
            setarElse();
         }
      }
      break;
   case 405:
      if(ok==1)
      {
         if(est1==1 && est2==1 && est3==1 && est4==1)
         {
            est1=0;est2=0;est3=0;est4=0;ok=0;
            mensagem = "Fim de Jogo!";
            label2 = new HStaticText(mensagem, 10,400,455,20,
            new Font("Tiresias", 1, 20),
            Color.DARK_GRAY, new DVBColor(250, 250, 250, 120),manager );
            mensagem = "Pressione EXIT para sair ou OK para recomeçar!";
            label3 = new HStaticText(mensagem, 10,420,455,20,
            new Font("Tiresias", 1, 18),
            Color.DARK_GRAY, new DVBColor(250, 250, 250, 120),manager );
            mensagem = "Pontuação: 4";
            label4 = new HStaticText(mensagem, 520, 500, 120, 20,
            new Font("Tiresias", 1, 18),
            Color.white, Color.red, manager);
            setar();
         }
          else
         {
            setarElse();
         }
      }
         break;

      case 406:
      if(ok==1)
      {
```

```
        if(est1==1 && est2==1 && est3==1 && est4==0)
        {
            mensagem = "Qual das figuras é um GATO?";
           label2 = new HStaticText(mensagem, 10,420,455,20,
            new Font("Tiresias", 1, 18),
            Color.DARK_GRAY, new DVBColor(250, 250, 250, 120),manager );
            mensagem = "CERTO!";
            label3 = new HStaticText(mensagem, 525, 460, 110, 20,
            new Font("Tiresias", 1, 16),
            Color.WHITE, Color.blue,
            new HDefaultTextLayoutManager());
            mensagem = "Pontuação: 3";
            label4 = new HStaticText(mensagem, 520, 500, 120, 20,
            new Font("Tiresias", 1, 18),
            Color.white, Color.red, manager);
            setar();
            est4=1;
        }
         else
        {
            setarElse();
        }
    }
        break;
case 27:
    try{

      Thread.sleep(1000);
                  }
    catch(InterruptedException ie){

    }

    try {
        destroyXlet(true);
        }
        catch(XletStateChangeException xletstatechangeexception) {
        }
case 10:
    if(est1==0)
    {
        ok=1;
        mensagem = "Qual das figuras é um CAVALO?";
        label2 = new HStaticText(mensagem, 10,420,455,20,
        new Font("Tiresias", 1, 18),
        Color.DARK_GRAY, new DVBColor(250, 250, 250, 120),manager );
        est2=0; est3=0; est4=0;
        scene.removeAll();
        scene.add(labelFundo);
        scene.add(label1);
        scene.add(label2);
        scene.add(logoVaca);
        scene.add(logoGato);
        scene.add(logoCachorro);
        scene.add(logoCavalo);
        scene.add(logoVermelho);
        scene.add(logoVerde);
        scene.add(logoAmarelo);
        scene.add(logoAzul);
        scene.repaint();
        est1=1;
    }
    break;

default:
    if(ok!=1)
    {
        mensagem = "Jogo disponível\nPressione OK!";
        label2 = new HStaticText(mensagem, 530, 20, 130, 40,
        new Font("Tiresias", 1, 15),
        Color.blue,new DVBColor(255, 255, 255, 200),manager );
        scene.removeAll();
        scene.add(label2);
        scene.repaint();
    }
```

```
                break;
            }
        }
        void setar(){
            scene.removeAll();
            scene.add(labelFundo);
            scene.add(label1);
            scene.add(label2);
            scene.add(label3);
            scene.add(logoVaca);
            scene.add(logoGato);
            scene.add(logoCachorro);
            scene.add(logoCavalo);
            scene.add(logoVermelho);
            scene.add(logoVerde);
            scene.add(logoAmarelo);
            scene.add(logoAzul);
            scene.add(label4);
            label2.repaint();
            label3.repaint();
            scene.repaint();
        }
        void setarElse(){
            label3 = new HStaticText("ERRADO!", 525, 460, 110, 20,
            new Font("Tiresias", 1, 16),
            Color.WHITE, Color.red,
            new HDefaultTextLayoutManager());
            scene.removeAll();
            scene.add(labelFundo);
            scene.add(label1);
            scene.add(label2);
            scene.add(label3);
            scene.add(logoVaca);
            scene.add(logoGato);
            scene.add(logoCachorro);
            scene.add(logoCavalo);
            scene.add(logoVermelho);
            scene.add(logoVerde);
            scene.add(logoAmarelo);
            scene.add(logoAzul);
            label2.repaint();
            label3.repaint();
            scene.repaint();
        }
    }
```