Universidade Federal de Santa Catarina

Sistemas de Informação

# PEP – Pair Eclipse Programming

## Plugin Eclipse para Programação Colaborativa em Dupla

Giorgio Santos Costa Merize

Leandro Pfleger de Aguiar

Florianópolis, Janeiro de 2007

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE SISTEMAS DE INFORMAÇÃO

# PEP – Pair Eclipse Programming

## Plugin Eclipse para Programação Colaborativa em Dupla

Giorgio Santos Costa Merize

Leandro Pfleger de Aguiar

Trabalho de conclusão de curso
apresentado como parte dos
requisitos para obtenção do grau de
Bacharel em Sistemas de Informação

Florianópolis - SC
2006/2

Giorgio Santos Costa Merize
Leandro Pfleger de Aguiar

# PEP – Pair Eclipse Programming
## Plugin Eclipse para Programação Colaborativa em Dupla

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação

Orientador: Frank Augusto Siqueira

Banca examinadora

José Eduardo De Lucca
Luiz Fernando Bier Melgarejo

# Sumário

# Apresentação

**Título:** Plugin Eclipse para Programação Colaborativa em Dupla

**Instituição de Desenvolvimento:** Universidade Federal de Santa Catarina, Departamento de Informática e Estatística (INE)

**Local de Desenvolvimento:** Florianópolis, Santa Catarina - Brasil

**Autores:** Giorgio Santos Costa Merize e Leandro Pfleger de Aguiar

**Orientação:** Prof. Ph.D. Frank Siqueira

**Banca Avaliadora Convidada:** Prof. José Eduardo De Lucca / Prof. Luiz Fernando Bier Melgarejo

# 1 Introdução

Com o avanço das tecnologias de rede e o uso crescente da Internet de alta velocidade, têm se tornado populares diversas facilidades tais como sistemas de informação com bancos de dados distribuídos, sistemas de voz sobre IP e videoconferência. O surgimento de um ambiente de programação colaborativo sobre a internet, neste âmbito, pode ser considerado uma continuação natural deste processo, capaz de interferir positivamente no desenvolvimento de software e até mesmo na educação à distância. A implantação de uma metodologia com este nível de sofisticação tecnológica, entretanto, depende da existência de recursos tecnológicos ou ferramental de suporte ao desenvolvimento, capaz de, além de dirimir as barreiras impostas pela comunicação ao longo da rede, garantindo confiabilidade ao processo, atuar em conjunto com a vasta cadeia de facilidades oferecida pelos modernos ambientes de desenvolvimento.

O "*Plugin* Eclipse para Programação Colaborativa em Dupla", aqui denominado PEP – Pair Eclipse Programming, é uma extensão do Eclipse, um IDE (*Integrated Development Environment* - Ambiente de Desenvolvimento Integrado) que agrega, entre outros benefícios, popularidade e uma vasta comunidade de suporte. Sua proposta é suprimir as distâncias no processo de programação em dupla, permitindo a prática da programação em par distribuída (DPP), dando suporte a metodologias de desenvolvimento que fazem uso deste recurso, como *Extreme Programming*, ao mesmo tempo em que garantindo acesso aos benefícios oferecidos pelo Eclipse.

## *1.1 Objetivos*

O objetivo geral do sistema idealizado é disponibilizar uma forma inteligente e usual de praticar a programação em par distribuída – DPP (*Distributed Pair Programming*) permitindo trabalho cooperativo ou CSCW (*Computer Supported Cooperative Work*) através do Eclipse. Resultados satisfatórios neste aspecto permitiriam a realização da prática da programação em par, requisito de algumas metodologias como *Extreme Programming*, de forma distribuída, servindo de meio para a prática do *Distributed Extreme Programming*, uma extensão do *Extreme Programming* que permite a execução do trabalho distribuído.

O *Plugin* que será desenvolvido objetiva permitir a conexão entre duas estações de trabalho geograficamente distribuídas, de forma que um mesmo ambiente de trabalho na IDE possa ser compartilhado, viabilizando a troca de dados entre as estações. Esta troca de dados, neste contexto, se traduz na possibilidade de dois indivíduos, interconectados através da Internet e, portanto, não necessariamente próximos entre si, praticarem a programação de uma mesma classe, por exemplo, de forma sincronizada e colaborativa. Este sistema, portanto, uma vez implementado, deve ser suficientemente automatizado para controlar esta troca de dados de maneira consistente, garantindo a confiabilidade por parte dos seus usuários.

O desenvolvimento de um plugin como o PEP, para que possa cumprir com sua missão com praticidade, depende da conquista com sucesso de alguns objetivos específicos de cunho teórico e, posteriormente, prático, baseado nos conhecimentos obtidos com os estudos. Destacam-se, assim, dentre estas tarefas,

o estudo da tecnologia dos Plugins para o Eclipse, bem como sua integração e processo de desenvolvimento dentro da IDE, o estudo das alternativas disponíveis como solução para comunicação pela rede, incluindo a identificação dos problemas conhecidos na troca confiável de dados através da Internet e controle de concorrência, e a implementação das soluções identificadas, aplicando uma metodologia de desenvolvimento ágil contemplada por uma etapa de testes bem definida.

## *1.2 Justificativa*

A fonte motivacional para o investimento no desenvolvimento do *plugin* se fundamenta em três grandes pilares. O primeiro deles é embasado pela grande e crescente popularidade com que conta o IDE Eclipse. Por se apresentar como um ambiente completo e extensível, devido a sua característica peculiar de ser este próprio, construído a partir da junção de uma série de *plugins*, o Eclipse vem cada vez mais ganhando espaço entre os desenvolvedores Java e de outras linguagens, também suportadas. Além de se propor como uma plataforma de desenvolvimento extensível, o Eclipse conta com os benefícios da comunidade de desenvolvimento de Software Livre, o que acaba transformando-o em uma verdadeira "comunidade *open source*" que, por focar no desenvolvimento aberto, livre de código proprietário (*vendor-neutral*), atrai cada vez mais a atenção dos interessados na interoperabilidade efetiva, acima de tudo.

O segundo dos pilares motivacionais é fortemente apoiado no avanço das técnicas para o estabelecimento da prática da engenharia de software. Com a difusão cada vez maior da validade e da importância do estabelecimento de boas

práticas no desenvolvimento de software que facilitem os fatores como a corretude, a manutenibilidade, a escalabilidade, a reusabilidade e a própria produtividade, entre outras, ocorre o crescimento da busca, também das pequenas e médias "*software houses*" por práticas que garantam competitividade neste mercado. Isto, em uma análise pouco substancial, associado ao sucesso obtido na gerência da mudança de requisitos e nos projetos praticados sobre metodologias ágeis de desenvolvimento, proporciona a popularidade das técnicas como "*Extreme Programming*" (ou, simplesmente, XP), que, entre outras práticas altamente focadas na produtividade, utiliza do recurso da programação em par como suporte à promoção da uniformidade de conhecimentos em uma equipe. Embora experimentos práticos já tenham atestado a validade da prática da Programação em Par Distribuída (DPP) e até mesmo da execução da metodologia XP de forma distribuída - DXP (*Distributed Extreme Programming*), é unânime a opinião de que há carência de aplicações que suportem a tarefa adequadamente.

O último, e não menos importante dos pilares que motivam o desenvolvimento do plugin PEP, está centrado na utilidade bastante prática para ferramentas de programação colaborativas deste gênero. Além de promover o estímulo ao trabalho de equipe e ao próprio desenvolvimento de habilidades relacionadas ao desenvolvimento colaborativo, uma ferramenta de tal porte serviria como incentivo ao uso de software livre na comunidade acadêmica, contribuindo com a filosofia proposta pelos órgãos internacionais do setor para a difusão do seu uso.

## *1.3 Escopo*

Embora seja conhecida a extensão e o caráter noviço da dialética que envolve a prática distribuída da programação em par e o seu uso na modalidade distribuída de *Extreme Programming*, este trabalho será limitado à simples apresentação dos conceitos que dão embasamento à implementação prática aqui proposta, ao invés de contribuir com a discussão em torno do tema. Nossa contribuição teórica, neste sentido, se restringe à sumarização e apresentação do conceito no estado da arte no que diz respeito à programação em par distribuída. Da mesma forma, não pretendemos explorar aqui toda a extensão dos aspectos que envolvem a programação colaborativa CSCW, tampouco correlacioná-la teoricamente com a programação em par, analisando a validade ou não de todos os recursos aplicados, mas apenas utilizar de suas teorias para fundamentar a importância deste recurso como ferramenta capaz de gerar produtividade na produção cooperativa de código.

Em relação à implementação prática, a intenção do trabalho não é desenvolver com a mais detalhada e completa das formas, um *plugin* comercial para a programação colaborativa, embora, conforme será apresentado na fundamentação teórica em capítulo posterior, a completude de recursos no que tange aos recursos multimídia tenha sua importância reconhecida. Tais cursos serão inicialmente postergados a uma versão futura, sendo o produto de software perfeitamente planejado para a extensibilidade pela equipe ou pela comunidade de software livre, uma vez que faz parte dos objetivos sua publicação.

# 2 Fundamentação Teórica

O PEP se baseia em diversas técnicas, tecnologias, metodologias e áreas de estudo cuja menção se faz necessária para o entendimento da validade do trabalho a ser desenvolvido. Tais recursos, além de constituírem uma sólida base teórica de suporte, servem como sustentação para o entendimento e viabilização do desenvolvimento do projeto.

## *2.1 CSCW – Computer Supported Cooperative Work*

O termo CSCW, do inglês *Computer Supported Cooperative Work* ou Trabalho Cooperativo Suportado por Computador, surgiu em 1981 com a introdução do conceito através de alguns artigos isolados. Com o crescimento e a massificação da Internet e a popularização do acesso aos meios de comunicação de alta velocidade (banda larga), surgiram propósitos comerciais mais claros e o tema foi ganhando relevância maior. Seu conceito se solidificou quando os primeiros experimentos de trabalho em grupo foram aplicados, dando origem a uma série de novos artigos citando os benefícios e as aplicações do CSCW.

O conceito que provavelmente mais bem descreve CSCW foi apresentado em 1994 por James D. Palmer e Ann Fields, da Universidade George Mason, EUA. Em seu artigo (PALMER et. Al., 1994), Palmer afirma que CSCW é “um sistema que integra processamento de informação e atividades de comunicação para ajudar indivíduos a trabalharem juntos como um grupo”. Hoje, CSCW evoluiu e é considerado uma área de pesquisa preocupada com o desenvolvimento de sistemas baseados em computador para o suporte e melhoria do trabalho em

grupo. Surgiram uma série de aplicações colaborativas e o conceito foi aplicado direta e indiretamente por programas de fins acadêmicos e comerciais, o que criou um conjunto de novas denominações práticas, como “groupware”, e especialidades como as ferramentas de gestão de ambientes virtuais. Embora muitas vezes o termo Groupware seja empregado como sinônimo para trabalho cooperativo suportado por computador, afirmar que são conceitos idênticos é incorreto. Enquanto CSCW é a pesquisa na área do trabalho em grupo e como os computadores podem apoiá-lo, *Groupware* é empregado para designar a tecnologia (hardware e software) gerada pela pesquisa em CSCW. Desta forma, correio eletrônico, teleconferência, suporte à decisão e editores de texto colaborativos são exemplos de *Groupware*, pois promovem a comunicação entre os membros de um grupo de trabalho e, portanto, contribuem para que o resultado final seja maior que a soma dos trabalhos de cada integrante do grupo.

O CSCW pode ser classificado de 4 maneiras (PALMER et. Al., 1994), levando-se em consideração o modo como é praticado pelos usuários:

- Modo síncrono: no mesmo local, no mesmo tempo, trabalhando em uma mesma atividade;
- Modo síncrono distribuído: atividades sendo realizadas na mesma hora, mas com os participantes estando em diferentes locais;
- Modo assíncrono (temporalmente distribuído): as tarefas são executadas em horas diferentes, mas no mesmo local;
- Modo assíncrono distribuído: diferentes locais em horas diferentes.

Há ainda, quando se discute trabalho cooperativo suportado por computador, o uso de alguns termos comuns à área, cuja menção é relevante:

- Grupo Homogêneo: grupo composto de indivíduos com nível de experiência e conhecimento similar;
- WYSINWIS (*what you see is not what i see*): “o que você vê não é o que eu vejo” é um termo empregado como paradigma para descrever sistemas que operam em modo distribuído em que o conteúdo observado pelos participantes durante o trabalho distribuído não é o mesmo;
- WYSIWIS(*what you see is what i see*): “o que você vê é o que eu vejo” é empregado quando se descreve sistemas que operam em modo distribuído e todos os participantes do grupo compartilham a mesma visão.

Embora muitas aplicações modernas façam uso do conceito de CSCW para denominar softwares distribuídos e utilizados em grupo, muitas delas não conseguem operar a questão da produtividade em níveis adequados. De acordo com Palmer, para que o CSCW possa ser, de fato, trazido ao dia-a-dia satisfatoriamente, é necessário cumprir uma série de requisitos mínimos à aplicação que implementa o conceito:

- ser um software amigável (com boa usabilidade);
- suportar e, adequadamente, mapear a dinâmica das atividades em grupo;
- padronizar os termos (definições, palavras, símbolos) em uso;
- gerenciar a dificuldade criada pelo composto “várias tarefas e várias pessoas” .

Embora aparentemente todos os aspectos citados pareçam manipular apenas questões relativas às características funcionais das aplicações que

implementam CSCW, há também vários estudos na área que abrangem as características sociais envolvidas no uso das ferramentas de grupo, uma vez que, por se tratar de uma área relativamente nova, em termos práticos, as ferramentas de CSCW, na grande maioria das vezes, quando são introduzidas, mudam a forma tradicional de trabalho. Em um artigo publicado pelo departamento de pesquisa da Siemens sobre as ferramentas para CSCW (REINHARD et. Al., 1994), os autores afirmam que “os processos psicológicos, sociais e culturais ativos dentro dos grupos de colaboradores são as chaves reais para a aceitação e o sucesso de sistemas CSCW”.

## *2.2 PP – Pair Programming*

Programação em Par (do inglês *pair programming*) ou programação colaborativa é onde dois programadores desenvolvem software lado a lado em um computador (COCKBURN 2000). Tal conceito, embora não seja muito novo, ganhou popularidade apenas recentemente, a partir de 1996 quando foi introduzido com a metodologia de desenvolvimento ágil *Extreme Programming* (WILLIAMS et. Al., 2000). Em PP os programadores trabalham em conjunto para apresentar a melhor solução para um problema proposto. Há um membro responsável por comandar (pilotar) o computador e digitar e outro integrante que revisa o código, seguindo padrões e auxiliando na análise mais detalhada do problema para entender outros pontos de vista em busca da melhor solução.

Desde que surgiu, a programação em pares foi vista pelos gerentes de projeto menos críticos como desvantajosa. Tais gerentes, por enxergarem os programadores como um recurso, relutavam em aceitar a prática afirmando que,

embora tenha alguns benefícios, seria inevitável dobrar o número de pessoas necessárias para desenvolver uma mesma porção de código. Este fato motivou um dos mais conhecidos experimentos práticos relacionados à programação em par: o experimento da Universidade de Utah de 1999. No artigo "*Experimenting with Industry´s Pair-Programming Model in the Computer Science Classroom*", de Laurie Williams e Robert Kessler (WILLIAMS; KESSLER, 1999) os autores descrevem este experimento prático controlado realizado com estudantes de Ciência da Computação da Universidade de Utah, EUA, cujo objetivo principal foi quantificar, em termos práticos, quais são os reais benefícios da programação em par. No projeto de software proposto no experimento, enquanto parte da turma foi orientada a codificar na forma tradicional o restante recebeu a tarefa de aplicar a técnica. Além de apresentar um custo de desenvolvimento de apenas 15% a mais, contrariando as expectativas que afirmavam que o custo seria dobrado, o time de desenvolvimento em pares produziu um código de melhor qualidade, com uma menor quantidade de linhas e com um percentual de erros 15% menor que o time que não operou em duplas.

Os benefícios da programação em par não estão restritos ao campo econômico, embora este seja, obviamente, o principal alvo de interesse do setor de produção de software. Logo depois da apresentação das conclusões do experimento de Utah, Laurie Williams e Alistair Cockburn reuniram em um artigo denominado "*The Costs and Benefits of Pair Programming*" (em português: os custos e benefícios da programação em par). Não somente estes resultados, mas conclusões de experiências relatadas por outras organizações que resolveram apostar na programação em par, levaram os autores a concluir que "com um custo

de até 15% a mais, a programação em par melhora a qualidade da arquitetura do software, reduz defeitos, reduz risco de perda de membro da equipe, melhora as habilidades técnicas, melhora a comunicação no time e é considerada mais 'agradável' em níveis estatísticos significativos" (COCKBURN 2000). No artigo, são apresentados ainda oito campos da engenharia de software e eficácia organizacional beneficiados com a programação em par:

- **Economia:** adicionar uma segunda pessoa, como provou o experimento de Utah, tem um custo pequeno, comparativamente à economia de tempo na remoção de *bugs* que tradicionalmente surgem. Em média, a programação em pares tem um custo em horas de programação 15% superior, somente, e não o dobro como muitos imaginam.
- **Satisfação:** a resistência dos programadores ao trabalho em dupla também foi avaliada. Em níveis estatísticos consideráveis, times que iniciaram a prática da programação em par admitiram que seus membros acharam a experiência mais agradável do que o trabalho individual. Em Utah, o experimento mostrou uma satisfação de mais de 80% dos funcionários envolvidos. Há também o efeito de aumento de confiança no código produzido, já que a prática da revisão é intensificada.
- **Qualidade no design:** pares produzem programas mais curtos do que na programação solo e com uma melhor arquitetura do sistema. Uma justificativa para este fato é apresentada pela ciência cognitiva, que

afirma que na programação colaborativa “a mente não trabalha sozinha” ampliando o espaço de alternativas[1].

- **Revisão Contínua:** a prática da inspeção de código é uma comprovada forma de eliminar defeitos de software. Apesar disso, grande parte dos programadores não considera a prática agradável. Na programação em pares a inspeção ocorre através da revisão contínua, o que aumenta a velocidade da inspeção e elimina o desgosto dos programadores pela tarefa. Outro benefício da revisão de código que ocorre na programação em par é que os programadores aprendem mais sobre a linguagem à medida que aprendem mais sobre o sistema, criando um potencial educativo positivo. Os erros são encontrados à medida que são inseridos, padrões de codificação são seguidos com maior precisão e membros do time melhoram sua capacidade de trabalho em equipe.
- **Solução de problemas:** os participantes adquirem a habilidade de resolver “problemas impossíveis” mais rapidamente. No estudo, o código resultante apresentou 15% menos defeitos, o que mostra mais qualidade no código final e um menor desperdício de tempo com correções. Ocorre o chamado “*pair relaying*”, efeito onde a intensa troca de idéias entre os membros facilita a identificação de soluções com agilidade.
- **Aprendizado:** programadores em pares repetidamente citam o quanto aprenderam com seus pares, desde técnicas de uso até regras da

1 A “pesquisa em amplo espaço de alternativas” é uma teoria defendida pela ciência cognitiva. Em (COCKBURN 2000) os autores mencionam um estudo realizado por Nick Flor em 1991 que apresenta três razões para justificar a qualidade da produção na programação em par: (1) os atores possuem diferentes níveis de experiência no assunto (2) eles podem ter acesso a diferentes informações relevantes sobre o tema (3) eles mantém relações diferentes com o problema em decorrência de seus papéis funcionais.

linguagem de programação e do sistema. Com a alternância de papéis de professor e estudante constante, cria-se também o aprendizado de habilidades não verbais e hábitos. Na programação em par ocorre também a “quebra de fronteira” que tradicionalmente separa os programadores inexperientes dos experientes. Enquanto que para os inexperientes cria-se a possibilidade de um rápido aprendizado sem a necessidade de um treinamento formal, os experientes aprendem que é possível obter conhecimento também dos programadores inexperientes. Como resultado final, os projetos acabam com um conhecimento pleno e uniforme de todos os membros do time sobre os detalhes do projeto.

- **Comunicação:** à medida que a capacidade de cooperação é ampliada no time, as pessoas passam a compartilhar problemas e soluções com maior velocidade. As formas de comunicação são otimizadas e a freqüência de contato é ampliada, crescendo também o fluxo de informações. Com o rotacionamento, prática na qual as duplas são alternadas, ocorre uma melhor distribuição da informação pelo time.
- **Gerenciamento de Projeto e de Equipe:** à medida que o projeto caminha, as habilidades dos membros do time se ampliam e se equalizam, reduzindo-se o risco de uma eventual perda de um membro na equipe. Várias pessoas sentem-se familiarizadas em um nível maior de detalhes em relação ao sistema, reduzindo riscos no projeto.

Embora a programação tenha sido considerada, por vários anos, uma atividade solitária e programadores experientes se recusem a adotar a prática, afirmando que isto causaria problemas de atraso e coordenação nos projetos, há

vários experimentos e relatos de times de desenvolvimento que optaram pela utilização da programação em par que apresentaram resultados satisfatórios, contrariando a crença inicial sobre a prática.

### 2.2.1 Distributed Pair Programming – DPP

Embora a adoção da programação em par já tenham sido claramente apresentada como benéfica, comparativamente à atividade solitária, várias pessoas questionam se tais benefícios se manteriam quando os membros do par não se encontram fisicamente próximos um do outro. Para testar estes questionamentos, um estudo conduzido na Universidade da Carolina do Norte em 2002 (BAHETI; WILLIAMS; et. Al., 2002), aplicou um experimento com o objetivo de determinar uma plataforma técnica eficiente para permitir a programação em par distribuída. No projeto do experimento, foram realizadas comparações com pares de estudantes executando programação em par localmente e pares realizando DPP através da Internet utilizando compartilhamento de desktop através da ferramenta Microsoft NetMeeting, que provê também as funcionalidades de compartilhamento de texto e vídeo, utilizando câmeras, fones de ouvido e microfones. Nos objetivos do experimento foram comparados a qualidade do código gerado, a produtividade e, por fim, o desempenho na comunicação dentro dos times. Como resultado, os autores apresentaram as seguintes conclusões (BAHETI; WILLIAMS; et. Al., 2002):

- Desenvolvimento de software envolvendo DPP parece ser comparável ao desenvolvimento local em termos de duas métricas: produtividade (em

termos de linhas de código por hora) e qualidade (em termos de classes obtidas);

- Times locais não produzem, estatísticamente e significativamente, melhores resultados do que times distribuídos;
- O respaldo dado pelos estudantes que participaram do experimento indica que DPP promove trabalho em equipe e comunicação dentro do time virtual.

## *2.3 XP - Extreme Programming*

Extreme Programming, ou XP, foi a primeira metodologia de desenvolvimento popular a implementar a programação em par. Embora a prática, quando foi proposta, não tenha necessariamente sido associada a uma metodologia de desenvolvimento ágil, foi a partir de XP que efetivamente ganhou os olhos da comunidade de desenvolvimento de software.

### 2.3.1 Metodologias Ágeis

Historicamente, o termo “metodologia ágil” começou a ser utilizado após a publicação do “manifesto pelo desenvolvimento ágil de software” ou simplesmente “manifesto ágil“.

O manifesto ágil surgiu de uma reunião de um grupo de renomados veteranos da área de software como Alistair Cockburn, Martin Fowler, Ron Jeffries e Kent Backe, em Fevereiro de 2001 em Utah, EUA. Durante a troca de experiências, embora o grupo contivesse indivíduos com práticas e conceitos próprios, muitos deles com vários artigos publicados defendendo suas próprias

teorias na área de engenharia de software, todos concordavam haver um conjunto de princípios em comum aos projetos de sucesso e características negativas similares em projetos distintos que falhavam. Os princípios de sucesso foram então resumidos, surgindo o termo Desenvolvimento Ágil que deve seguí-los. São eles:

- Valorizar mais indivíduos e a interação entre eles do que processos e ferramentas;
- Valorizar mais software funcional do que uma documentação detalhada;
- Valorizar mais a colaboração com o cliente do que a negociação de contratos;
- Responder mais a mudanças do que seguir um plano com rigor.

Com a publicação do manifesto ágil, metodologias como SCRUM[2] e *Extreme Programming*, que já havia sido publicado por Kent Beck no final dos anos 90, passaram a ser utilizadas com mais freqüência.

### 2.3.2 Extreme Programming

*Extreme Programming* ou XP é uma metodologia criada por Kent Beck, Ward Cunninghan e Ron Jeffries, que mais tarde vieram a participar do manifesto ágil de software. Historicamente, XP surgiu com o projeto C3 em 1996, um projeto de software de folha de pagamentos para a empresa Chrysler que, após uma primeira experiência fracassada, teve sua liderança assumida por Kent Beck. Embora o projeto tenha sido cancelado mais tarde, a partir de 1999 vários artigos

2 SCRUM é uma metodologia desenvolvida por Ken Schwaber, que participou da criação do manifesto ágil, e Jedd Sutherland e é inspirada nas idéias de desenvolvimento rápido que faz analogias à agilidade de um jogo de Rugby (a palavra scrum é dada à jogada em que uma equipe age em conjunto para tentar mover a bola pelo campo).

sobre o tema foram publicados e as práticas passaram a ser utilizadas em vários projetos como metodologia de desenvolvimento.

No artigo “Aceitando Mudanças com Extreme Programming” (do inglês “*Embracing Change with Extreme Programming*”) (BECK, 1999), Kent Beck afirma que, enquanto a comunidade de engenharia de software tenta criar mecanismos para gerenciar o alto custo de mudança de requisitos durante o desenvolvimento, o que é tido como um dos mais impactantes e caros problemas nas metodologias tradicionais como Cascata, XP se propõe a tratar esta questão com outra estratégia de abordagem. XP explora a redução dos custos da mudança e software por praticar as atividades de planificação, análise e desenvolvimento em pequenas porções a cada intervalo de tempo ao longo do ciclo de desenvolvimento (BECK, 1999). Nas metodologias tradicionais era necessário que o cliente esclarecesse antecipadamente e com exatidão os requisitos e, a partir de então, partia-se para o desenvolvimento o que causava vários problemas, uma vez que os próprios usuários, na grande maioria das vezes, não sabiam exteriorizar suas intenções, criando contradições e mudanças de idéia constantes.

XP se apóia em um conjunto de valores e um conjunto de práticas que devem ser aplicadas seguindo-os. Enquanto os valores se concentram mais na determinação de um perfil comportamental que o programador e o time como um todo deve possuir para trabalhar no dia-a-dia, as práticas guiam as atividades necessárias para que os objetivos de XP sejam atingidos. São valores em *Extreme Programming*:

- *Feedback*: em XP procura-se reduzir ao máximo o intervalo de tempo entre o momento em que uma ação é tomada e o seu resultado é obtido. Um exemplo da ação deste valor é a entrega de versões com maior freqüência;
- Comunicação: em XP se prioriza a prática do diálogo como forma de ampliação do nível de comunicação internamente ao time de desenvolvedores e também com o cliente, que deve participar o máximo possível do processo;
- Simplicidade: XP utiliza este conceito em diversos aspectos para que nada além do necessário seja feito, visando à economia de tempo;
- Coragem: XP não se opõe a mudanças. Ao invés disso a metodologia se propõe a lidar com as mudanças e mantê-las sob controle. Para que isto seja possível, é necessária coragem para aceitar as mudanças e aplicá-las com segurança.

São práticas de XP (BECK, 1999):

- *Planining Game* : são os clientes que decidem escopo e tempo da entrega das versões funcionais baseando-se nas estimativas dos programadores que implementam apenas o necessário para cada iteração;
- *Small Releases* (Entrega em Pequenas Versões): o sistema é colocado em funcionamento mesmo antes de estar completamente pronto. As entregas são feitas em partes;
- *Metaphor* (Metáfora): a arquitetura do sistema é definida através de uma série de metáforas entre o cliente e os programadores;

- *Simple Design* (Design Simples): a programação é sempre feita baseando-se na simplicidade com a menor quantidade possível de classes e métodos e evitando duplicidade;
- *Tests* (Testes): testes de unidade são desenvolvidos à medida que o sistema é desenvolvido e o cliente define testes de aceitação;
- *Refactoring* (Refatoração): a arquitetura do sistema pode passar por refatorações para melhorias na arquitetura do projeto. A cada mudança os testes são executados novamente;
- *Pair Programming* (Programação em Par): tudo é produzido por duas pessoas utilizando apenas um computador;
- Integração contínua: a cada no máximo algumas horas todo o código é integrado (testes de integração);
- *Collective ownership* (Propriedade Coletiva de Código): qualquer membro do time pode mudar qualquer parte do código a qualquer momento (propriedade coletiva de código);
- *On-site customer* (Cliente Participativo): o cliente ou o seu representante está sempre junto com a equipe (*full time*);
- *40-hour weeks* (Semanas de 40 Horas): a necessidade de horas extras indica problemas que necessitam de correção;
- Open workspace (Ambiente de Trabalho Amplo): o time trabalha em uma sala grande com pequenos cubículos na periferia. Os pares trabalham em computadores no centro da sala;

- *Just rules* (Apenas Regras): todos do time devem seguir as regras com rigor, embora as regras possam mudar caso seja uma decisão do time.

Para que os riscos do projeto sejam dissolvidos, XP propõe ainda dividir a produção em "*stories*", que é a divisão das funcionalidades em pacotes testáveis, estimáveis e orientados ao negócio. Depois de identificadas, as *stories* passam pela aprovação do cliente, que é quem as ordena de acordo com suas prioridades. A cada iteração, um conjunto de *stories* é implementado e testado pelo time, através de testes unitários, e pelo cliente, ao final da iteração. Para a implementação, cada *storie* é quebrada em tarefas menores e cada tarefa é então implementada por uma dupla, uma vez que todo o código deve ser escrito com duas pessoas em uma máquina.

Em relação aos testes, que são "o coração da metodologia" (BECK, 1999), toda implementação deve se iniciar pelos testes, facilitando o aprendizado e criando testes mais eficazes e condizentes com o conteúdo implementado. Esta prática também antecipa a detecção de erros. Em XP testes também são provenientes do cliente, pois é este que os especifica, criando um aumento da sua confiança na correta operação do sistema.

XP é uma metodologia que não deve ser entendida como "uma idéia polida a ser seguida". Ao contrário, seus autores recomendam como estratégia de implantação uma abordagem incremental e adaptativa e somente em situações em que é claramente aplicável[3].

3 Em seu artigo (BECK, 1999), Kent Beck afirma que nunca deve-se implementar XP completamente de uma vez. Ao invés disso, pode-se escolher apenas um problema do processo em desenvolvimento e tentar resolvê-lo com XP. O artigo sugere ainda que XP pode ser remoldado ou adaptado às necessidades de cada empresa, o que abre a discussão para a aplicação em grupos de desenvolvimento distribuído.

### 2.3.3 Distributed Extreme Programming

Para que o valor da Comunicação possa ser aplicado, como recomenda fortemente a metodologia, XP enfatiza a necessidade da presença física dos membros do time e da proximidade entre eles, idealmente. Da mesma forma, XP defende como prática indispensável o envolvimento do cliente durante todo o ciclo de desenvolvimento, possuindo obrigações claramente definidas no projeto. Várias pessoas argumentam, entretanto, que este nível de proximidade do cliente e do próprio time de desenvolvimento muitas vezes não é possível, especialmente quando se trata de organizações geograficamente distribuídas.

Em um artigo publicado em 2001, denominado *Extreme Programming* Distribuído (em inglês "*Distributed Extremme Programming*" - DXP) (KIRCHIER et. Al., 2001), os autores conceituam DXP como *"Extreme Programming* com certos relaxamentos em requisitos de proximidade física dos membros do time*"*, permitindo a aplicação dos princípios de XP em um ambiente distribuído e móvel. Eles afirmam que, para que isto seja possível, a DXP deve tratar adequadamente as práticas *Planning Game*, *Pair Programming*, *Continuous-Integration* e *On-Site Customer* e, para isto, assume que alguns requisitos devem estar presentes:

- Conectividade: é necessária a disponibilidade de alguma forma de conexão entre os participantes;
- E-mail: que pode ser usado como uma forma conveniente de trocar informação e agendar sessões DXP;

- Gerência de Configuração: para que os artefatos de programação possam ser gerenciados é mandatário o uso de alguma ferramenta do gênero, o que também habilita a propriedade coletiva de código;
- Compartilhamento de aplicação: *"*para aplicar as práticas XP em um ambiente distribuído, alguma forma de aplicação ou software de compartilhamento precisa estar disponível ao time*"* (KIRCHIER et. Al., 2001);
- Vídeo Conferência: alguma forma de vídeo conferência (suporte a áudio e vídeo simultâneo) deve estar disponível;
- Familiaridade: DXP afirma que o sucesso somente pode ser obtido quando os membros do time conhecem um ao outro bem e são capazes de enxergar a prática do DXP como uma extensão natural do seu modo de trabalho tradicional.

Para que aplicabilidade e as conseqüências do DXP pudessem ser comprovadas, foi realizado um experimento (KIRCHIER et. Al., 2001)onde um projeto foi desenvolvido por um time do qual faziam parte 4 membros: um na Índia, outro na Alemanha, outro nos Estados Unidos e um quarto membro viajando constantemente entre os Estados Unidos e a Itália. Embora algumas dificuldades tenham sido encontradas, como a falta de um aplicativo mais adequado para o compartilhamento de aplicação e problemas com a taxa de transferência, os autores concluíram que, de fato, "DXP pode eficientemente integrar membros remotos e móveis em um processo de desenvolvimento. Então se tornando uma valiosa extensão ao tradicional XP". Eles afirmam ainda que, embora tenham a

clara noção de que uma reunião virtual não pode substituir a interação humana direta, “DXP permite um envolvimento muito mais efetivo do cliente em relação ao XP, especialmente em situações onde parece ser impossível se ter um cliente on-site”.

## *2.4 Plataforma de Desenvolvimento Eclipse*

O início do desenvolvimento do Eclipse foi realizado pela OTI (*Object Technology International*) antes de ela ser adquirida pela IBM em 1996. A partir de 1998 a IBM começou a utilizar-se, internamente, do Eclipse para a integração de seus muitos programas de desenvolvimento, sendo que só foi transformá-lo em uma plataforma *Open Source (código aberto)* em novembro 2001. Mais recentemente, em 2003, uma fundação independente da IBM foi criada para manter e gerenciar o projeto.

Além de possuir o código fonte aberto, o que traz bastante flexibilidade para os programadores, a plataforma Eclipse apresenta também um perfil voltado à integração de componentes. Enquanto várias IDE´s limitam-se à criação de novas funcionalidades para suas empresas parceiras, o Eclipse foi criado como uma plataforma para plugins, voltada para estender as funcionalidades da IDE, possibilitando que isto seja feito por qualquer programador. Assim, o Eclipse permite a automatização de uma série de funções que os programadores comumente teriam que executar manualmente ou desenvolver em outros aparatos de software. Por ser desenvolvido com código aberto, o Eclipse, permite aos seus usuários o pronto acesso ao código fonte, assim possibilitando modificações e inovações para

atingir suas necessidades, desta forma permitindo total interação com a ferramenta, o que facilita muito a elaboração de plugins mais complexos.

O Eclipse segue os padrões da OMG (*Object Management Group*), organização que padroniza softwares orientados a objetos, permitindo interoperabilidade e portabilidade entre aplicações distribuídas.

Outro fator importante é que o Eclipse é a IDE Java dominante no mercado. Segundo "*The Dominant Java IDE*" (IEEE JNL Computer, Volume 38, July 2005) (GEER, 2005), a ferramenta domina o segmento de IDE's Java. Toda a plataforma é gratuita, enquanto outras IDE´s Java mais comuns custam até 3500 dólares cada. No entanto, sua licença permite que se criem novos plugins proprietários, cobrando por isto.

A plataforma oferece uma série de APIs para conectar as ferramentas em uma unidade, o *Generic Workbench*, que trabalha como um ambiente único com uma série de comportamentos e interfaces. Embora tenha sido desenvolvido em Java para garantir portabilidade, ele também é preparado para desenvolvimento em outras linguagens como COBOL, C e HTML.

### 2.4.1 Arquitetura

Como já foi citado, o Eclipse é um ambiente de programação voltado para plugins e sua estrutura básica consiste em um núcleo que permite conexões com inúmeros plugins. Ele é composto por um editor de código, um compilador, um *depurador*, um construtor de interfaces gráficas, além de outras ferramentas. Ao editor de texto foram adicionados recursos de suporte ao programador e

automatização, a fim de oferecer o maior conforto e agilidade para os programadores.

Outros aplicativos podem atuar dentro desta estrutura básica para criar uma aplicação melhor e que cumpra com as necessidades do usuário. Exceto ao pequeno núcleo do *runtime*, tudo em Eclipse é implementado como plugins.

Uma das principais características da IDE é a de carregar os plugins sobre demanda, o que consiste em economia de recursos, já que efetiva carga dos plugins só é realizada quando alguma das funções do mesmo for requisitada.

Na seqüência segue uma breve descrição de cada um dos componentes principais da plataforma Eclipse.

- Plataforma *Runtime* – Encarregada de descobrir quais plugins estão disponíveis no diretório de plugin do Eclipse e ativá-los.
- *Workspace* - Responsável por administrar e gerenciar os recursos do usuário, que são organizados dentro de um ou mais projetos.
- *Workbench* - Interface gráfica do usuário.
- *Team* - (realiza o controle de versão interagindo com repositórios CVS – *Version Control System* - e administra as configurações).
- Ajuda – Cuida do sistema de documentação.

A Ilustração 1(GEER, 2005) mostra a estrutura básica da plataforma do Eclipse descrita acima.

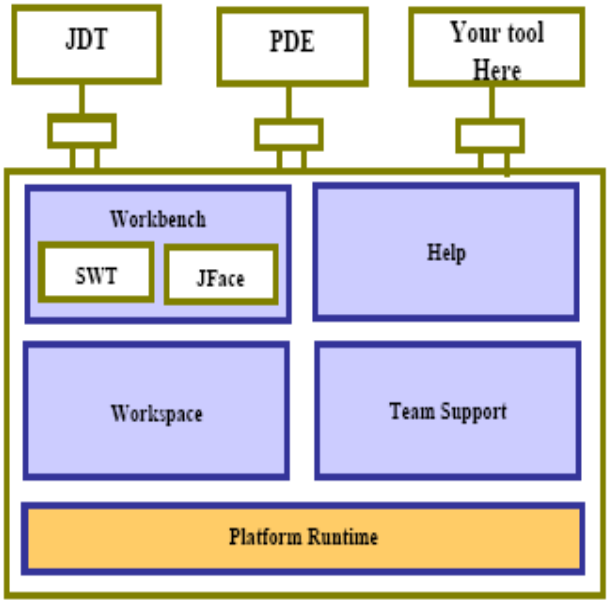


**Ilustração 1: Arquitetura do Eclipse (GEER, 2005)**

### 2.4.2 Plugins

Um *plugin* pode ser definido como um software que se conecta a um aplicativo para estender suas funcionalidades, ou seja, o *plugin* interage com um aplicativo utilizando suas bibliotecas de classes e funcionalidades, permitindo assim executar um serviço adicional no programa principal.

Planejado para ser extensível, o próprio Eclipse já inclui uma série de recursos para a produção de plugins que inclui desde assistentes de criação até depuradores e ferramenta de teste.

Inicialmente devemos saber onde se encontram os plugins instalados no Eclipse. Normalmente eles se encontram em um subdiretório chamado plugins, no interior do qual se encontram um conjunto de arquivos e pastas necessários ao seu funcionamento:

- **plugin.xml** - arquivo manifesto que fornece informações ao Eclipse tais como nome do plugin, versão, dependências, extensões, e alguns detalhes adicionais;
- **plugin.properties** - um arquivo de propriedades que define configurações para o plugin.xml;
- **about.html** - um arquivo HTML para informações de licença;
- **Lib** - um diretório para arquivos JAR (arquivo de biblioteca Java), que são necessários ao plugin;
- **Ícones** - um diretório para ícones específicos;
- **Outros arquivos.**

O desenvolvimento de um plugin no Eclipse deve seguir um conjunto de passos básicos:

- (1) Definir no arquivo de manifesto (plugin.xml), os pontos de extensão que o *plugin* irá contemplar no programa principal;
- (2) Definir uma subclasse de alguma classe da plataforma Eclipse de acordo com a extensão sendo definida;
- (3) Implementar métodos herdados destas classes juntamente com código de lógica de funcionamento do *plugin;*

### 2.4.3 Ambiente de Desenvolvimento de Plugins – PDE

Para auxiliar o desenvolvimento dos plugins, o que é uma atividade muito utilizada e necessária, a plataforma disponibiliza um plugin, chamado PDE (*Plugin Development Environment*), que facilita o desenvolvimento de novos aplicativos para o Eclipse. Algumas características da PDE são mostradas a seguir:

- Trata-se, este próprio, de um plugin construído usando os recursos da própria plataforma e o JDT(*Java Devolopment Tools*) do Eclipse;
- Permite a definição de um *plugin project*, que agrega os elementos do *plugin* (arquivo *manifest*, código-fonte, *gifs, etc...*);
- Possibilita a construção, compilação, depuração, teste e empacotamento de um *plugin;*
- Possui ferramenta para edição do arquivo de *manifest (plugin.xml);*
- Possui *wizards (templates)* para a geração de diversas extensões: ações, editores, conteúdo de ajuda, preferências, views, entre outros;
- Suporte à instalação (*deployment)* de um *plugin* num arquivo .zip ou atualização automática via Eclipse (*new feature*);
- Possibilita a importação de novos modelos de plugins;
- Exportação de plugins;
- Disponibiliza recursos para testar e depurar plugins;
- Permite a visualização e o controle dos logs.

## *2.5 Trabalhos Relacionados*

### 2.5.1 COLLAB – Módulo de Desenvolvimento Colaborativo (Netbeans)

Desenvolvido para o IDE da Sun, concorrente do Eclipse em termos de funcionalidades, o COLLAB foi desenvolvido para “permitir aos times distribuídos ou grupos de trabalho interagirem e trabalharem dinamicamente juntos em uma maneira altamente produtiva”(DEVELOPER, 2005). Publicado inicialmente em 2004 apenas para a versão não gratuita da IDE da Sun, o Java Studio Enterprise, o Collab surgiu para permitir uma “inovação no trabalho coletivo unificado”. Com a disponibilização para o Netbeans, a versão gratuita do IDE da Sun, o plugin ganhou mais popularidade sendo, por várias vezes, referenciado por revistas da área como sendo a única solução para o desenvolvimento colaborativo em grupo.

Embora tenha sido defendido pela Sun como sendo uma solução que permite agilidade no uso do trabalho de equipe para implementação de problemas em aplicações complexas, não há registro de publicações associando o plugin aos conceitos de DPP ou DXP, justificando sua criação nestes tópicos.

Dentre as características funcionais do Collab, destacam-se:

- sincronismo de código;
- sincronismo de algumas operações no IDE (ex. Comandos de criação de classes, remoção, etc);
- utilização de controle de concorrência por bloqueio de trechos de código, permitindo edição simultânea de trechos diferentes de uma mesma classe.
- utiliza um servidor centralizado, o qual necessita de registro dos usuários.
- permite múltiplos usuários logados ao mesmo tempo.

- possibilita compartilhar e descompartilhar projetos ou classes depois de estar conectado aos parceiros.

### 2.5.2 Sangam

Sangam é um plugin do Eclipse que permite aos usuários compartilharem um ambiente de desenvolvimento como se estivessem utilizando o mesmo computador, permitindo assim a prática do DPP ou *Distributed Pair Programming*. A ferramenta foi desenvolvida para sincronizar os ambientes de desenvolvimento de dois programadores geograficamente distribuídos e intenciona mostrar que, em termos de produtividade e qualidade, a programação em pares distribuída pode ser tão eficaz quanto a local, proposta pela XP (*Extreme Programming*).

Uma das motivações para o desenvolvimento do Sangam foi a resolução do problema de baixa taxa de atualização (*refresh*), especialmente para situações de pouca largura de banda, apresentado pelo compartilhamento de *desktop*, que consiste no uso de aplicativos como VNC e Netmeeting para a visualização das telas dos computadores. O plugin segue a arquitetura orientada a eventos, a qual se baseia em ações a serem realizadas para cada comportamento, e é composto de três componentes básicos: interceptador de eventos, servidor de mensagens e reprodutor de eventos:

- Interceptador de eventos - é responsável por capturar o evento quando o controlador executa um comportamento e enviar uma mensagem ao servidor;

- Gerenciamento das mensagens - é baseado em um servidor centralizado e todos que desejam participar da sessão de programação devem se conectar ao mesmo servidor;
- Reprodutor de eventos – quando ele recebe uma mensagem do servidor de mensagens ele verifica a mensagem (*parse*) e interage com o Eclipse para realizar a ação solicitada, repetindo a ação localmente.

#### 2.5.2.1 Funcionalidades Propostas pelo Sangam

- Sangam Editor - permite edição síncrona (digitar, selecionar, abrir, fechar e sincronizar);
- Sangam Launcher - permite iniciar uma aplicação Java ou um teste com Junit simultaneamente (executar e depurar);
- Sangam Toolbar - permite conectar e desconectar do servidor de mensagens e alternar entre os modos de controlador e controlado (start/stop driving);
- Sincronização de recursos - ao adicionar, deletar ou modificar arquivos as mesmas ações são refletidas no controlado;
- Sincronização no Refactoring: suporta apenas alguns casos especiais onde é possível captar os eventos gerados por operações, como por exemplo, a operação “rename”;

#### 2.5.2.2 Limitações e Problemas Conhecidos

- Atualizações – O projeto, que se encontra em um repositório de softwares livres, foi abandonado. Sendo que a última versão encontrada no

SouceForge é a 1.6.4, de 2002. Não se tem conhecimento da compatibilidade com as novas versões do Eclipse;

- Limitação ao editor Java - não pode operar, por exemplo, com o CDT (para desenvolvimento em C++);
- Suporte ao CVS – não há suporte atualmente para interação do plugin com um repositório de controle de versões como o CVS;
- Suporte a outras atividades do desenvolvimento – embora a metodologia DXP defina outras práticas além do *pair programming* que precisam ser realizadas para a efetiva colaboração distribuída, o Sangam compreende apenas a prática da programação;
- Controle de Concorrência – embora reconhecidamente o Sangam seja a primeira implementação prática de uma aplicação de suporte à programação em par distribuída, seu resultado, no que tange ao desempenho no controle da concorrência não é satisfatório em muitas situações (não há, por exemplo, como identificar com clareza que a conexão foi desfeita ou que a consistência do código pode ter sido afetada por um problema na conexão);
- Comunicação – apesar de possuir a funcionalidade de sincronismo de outras operações além da digitação de código de programas (Sincronização de Recursos), a comunicação dos eventos para o usuário controlado ocorre praticamente em segundo plano, contrariando as necessidades de clareza no processo sugeridas pelos autores da DXP, por exemplo;
- Sincronismo – não foram identificados mecanismos próprios para re-

sincronismo no caso de perda de comunicação, o que causa perda de tempo e possibilidade de inserção de erros em decorrência da necessidade de verificação manual de sincronismo e consistência entre as versões de cada participante.

# 3 Desenvolvimento do Projeto PEP

Com o conhecimento básico acerca dos fundamentos teóricos de CSCW, *Pair Programming* e *Extreme Programming*, podemos apresentar em termos gerais o que, de fato, será o plugin PEP. O PEP implementará uma modalidade de suporte ao CSCW Síncrono Distribuído, o que significa afirmar que tratará de desenvolvimento de código por um par de programadores interagindo simultaneamente com o software estando estes desenvolvedores fisicamente separados, mas logicamente conectados através da Internet. O resultado final propiciará a prática do DPP ou programação em pares distribuída na forma WYSIWIS (*What You See Is What I See*), o que significa dizer que o PEP poderá se comportar como uma ferramenta do gênero. Diferente do WYSINWIS *(What You See Is Not What I See),* onde os programadores não compartilham da mesma visão sincronizada (o que permitiria uma implementação de programação em par assíncrona, somente), o WYSIWS se enquadra mais adequadamente nas características desejadas de trabalho síncrono distribuído. Em última instância, desde que corretamente associado aos recursos multimídia necessários, o PEP permitirá o tratamento do requisito de "compartilhamento de aplicação" necessário ao DXP, possibilitando a programação em par em times de desenvolvimento que utilizam à metodologia *Extreme Programming* simultaneamente à necessidade de mobilidade ou localização remota de membros do time. Esta forma de colaboração será suportada pelas seguintes características funcionais fundamentais:

- Disponibilização de uma perspectiva de trabalho no Eclipse para suporte ao trabalho cooperativo;

- Arquitetura orientada a eventos.
- Reconhecimento automático e manual da perda de sincronismo, com possibilidade de resincronismo do código em tempo real.
- Recursos de controle de concorrência multi-thread;
- Mecanismo de “Chat” ou “bate papo” integrado;
- Extensibilidade à implantação de comunicação de voz;
- Extensibilidade ao uso vídeo conferência;
- Recurso de compartilhamento de projetos;

## *3.1 Metodologia de Desenvolvimento*

A metodologia de desenvolvimento aplicada para a criação do plugin PEP, esta própria foi planejada e executada utilizando algumas das práticas sugeridas pela metodologia XP. Foram práticas aplicadas:

- *Small Releases* (Entrega em Pequenas Versões): assim como em XP, na metodologia aplicada para o desenvolvimento do PEP, o software foi desenvolvido em etapas, normalmente pequenas, as quais chamamos Iterações. Para cada iteração, um objetivo é definido através de um “Caso de Uso de Alto Nível”, e esta funcionalidade (ou conjunto de funcionalidades) é agregada de forma a gerar uma versão sempre funcional, mesmo antes de o sistema estar completamente pronto. Desta forma, o sistema cresce de forma gradativa, em tamanho e complexidade e resultados visíveis tornam-se disponíveis mais rapidamente;

- *Simple Design* (Design Simples): todas as classes que compõem o sistema, seguindo a prática de Small Releases, foram inseridas no conjuto que compõe o produto final também de forma gradativa, à medida que se mostravam necessárias. Na prática, isto significa que, sempre que se concluía que uma nova classe seria necessária, apenas as funções necessárias naquele momento eram nela implementadas, ou seja, nenhuma classe foi concebida de maneira completa, ao contrário, todas as classes surgiram com funcionalidades básicas e foram sendo incrementadas, à medida em que novos comportamentos se faziam necessários;
- *Tests* (Testes): XP propõe, como prática para a execução de testes, um forte uso de testes unitários em associação com testes de aceitação ao final de cada iteração. Embora a utilização de testes unitários não tenha sido uma abordagem utilizada neste projeto, a cada release liberado no final de cada iteração, testes de aceitação eram executados, revalidando não somente a funcionalidade implementada na versão corrente, mas também todas as funcionalidades já implementadas até o momento;
- *Refactoring* (Refatoração): em XP, a prática de refactoring prega que a arquitetura do sistema pode passar por refatorações para melhorias na arquitetura do projeto, à medida que novos requisitos vão sendo implementados. Estas mudanças normalmente dizem respeito à melhorias do próprio código já desenvolvido e já funcional, tornando-o simplesmente mais manutenível, compreensível ou extensível. Mais adiante, neste mesmo documento, daremos exemplo de situações onde a refatoração foi aplicada e quais seus resultados ;

- *Pair Programming* (Programação em Par): assim como em XP, tudo o que foi produzido, não somente em termos de código, mas também nos artefatos documentais e resultados de pesquisas, foi gerado durante sessões de trabalho em par, contrapondo as adversidades decorrentes da falta de um ferramental de suporte adequado e integrado. Todas as sessões de trabalho ocorreram, portanto, em par, com os membros geograficamente separados, cada qual operando o seu respectivo microcomputador. Para que o método de trabalho colaborativo WYSIWIS (what you see is what i see), fosse possível, foi utilizada a ferramenta VNC, que utiliza como técnica de sincronismo a transmissão gráfica das partes da tela do "servidor" que são alteradas[4]. A ferramenta que possibilitou toda a comunicação de voz em tempo real, também gratuita, foi o Skype[5]. Além de cumprir muito bem com seu propósito (desde que haja banda de comunicação suficiente), o skype fornece ainda as funcionalidades de Chat e troca de arquivos, o que forma em conjunto com o VNC, um bom kit de ferramentas para a prática do trabalho colaborativo síncrono distribuído;
- Integração contínua: testes de integração foram executados constantemente, sempre que uma nova funcionalidade era agregada ao sistema, prática que evita a descoberta tardia de erros;

4 A ferramenta VNC, desenvolvida originalmente pelas empresas Olivetti e Oracle Corporation , hoje é utilizada com muita frequência como método de abertura de terminal remoto e, como foi mencionado em capítulo anterior, possui desvantagens no que tange à forma de sincronismo.

5 O software Skype, criado pelos mesmos fundadores do kaZaA (um dos mais conhecidos softwares de compartilhamento de arquivos), é um software que permite comunicação de voz e vídeo com qualidade,. F funcionando gratuitamente pela internet entre os usuários do software.

- *Collective ownership* (Propriedade Coletiva de Código): embora o projeto tenha sido desenvolvido com uma equipe de apenas duas pessoas, a prática da propriedade coletiva de código foi um elemento presente no desenvolvimento do plugin PEP. Nenhuma porção de código foi concebida ou idealizada como "de responsabilidade" de membro algum. Com a alternância constante de membro no comando (codificando/digitando), era comum surgir a necesside de intervenção por um membro do par em determinada porção de código cujo desenvolvimento não tenha sido originalmente de sua responsabilidade;
- *Just rules* (Apenas Regras): todas as regras que foram estipuladas para o desenvolvimento foram cumpridas à risca. Mesmo com a presença da problemática da agenda comum entre os membros, por exemplo, a prática do desenvolvimento em par se manteve com rigor e nenhuma das funcionalidades foi implementada individualmente. Quando necessário, entretanto, a metodologia foi adaptada para atender as necessidades da equipe e do projeto.

Para suportar este conjunto de práticas planejadas para o desenvolvimento ágil, a metodologia utilizada no desenvolvimento do plugin PEP foi subdividida em etapas bem distintas:

- Definição dos Requisitos: etapa onde foram definidos os requisitos principais a serem embutidos no comportamento do plugin para que seu objetivo maior fosse possível;
- Definição da Visão Geral do Sistema: etapa onde foi discutido e elaborado um esquema geral de funcionamento do software, seus componentes principais e a forma geral de interação entre eles;

- Definição dos Casos de Uso de Alto Nível: etapa onde, a partir da visão geral do sistema, foram determinadas as principais funcionalidades (funcionalidades macro) a serem elaboradas. Cada caso de uso, descrito em alto nível, introduz de maneira textual o objetivo correspondente para o mesmo;
- Ciclos Iterativos (iterações): uma vez determinados os casos de uso de alto nível, os mesmos foram agrupados ou deram origem diretamente aos ciclos iterativos, períodos nos quais ocorreram as tarefas de análise, projeto e implementação, com a conseqüente liberação de um release funcional, ao final do ciclo, contendo a nova funcionalidade implementada. Cada ciclo iterativo foi executado em períodos pequenos, geralmente de uma semana, ou menos, e o processo se repetiu até que todos os casos de uso planejados fossem executados;

### 3.1.1 Definição dos Requisitos

A definição dos requisitos iniciais ocorreu baseada principalmente em três elementos disponíveis: os requisitos mínimos para a prática do PP, os plugins já existentes e analisados (Sangam e Collab) e os novos requisitos oriundos da análise dos problemas destes. Os requisitos oriundos das práticas recomendadas pela programação em par, além de mais importantes, foram os mais facilmente identificados. Pode-se destacar como pertencentes a esta categoria os requisitos relacionados às ações que ocorrem durante uma atividade de programação em si, como a troca dos pares, o compartilhamento da mesma visão e o comando único,

que pode ser entendido como a prática que ocorre em PP na qual apenas um membro do par pode comandar o computador de cada vez. Dos plugins já existentes foram analisados e adquiridos requisitos mais elaborados, mais relacionados às ações que normalmente ocorrem durante as atividades de programação mesmo quando não se está programando em par. Tratam-se de funcionalidades necessárias à prática de programação que normalmente já existem em ambientes de desenvolvimento como o Eclipse e o Netbeans, como a rolagem de tela (scroll), seleção de texto e refactoring e que, mesmo estando em um ambiente de programação em par distribuído, também precisam estar disponíveis, permitindo que o usuário habituado ao ambiente Eclipse possa dispôr das ferramentas com as quais já trabalha em seu cotidiano. A última, e certamente a mais interessante das fontes de informação para a definição dos requisitos, os problemas dos plugins já disponíveis, trouxe, acima de tudo, tópicos cuja presença poderia determinar o sucesso do plugin enquanto ferramenta útil para a comunidade de desenvolvimento DPP. Ao identificar tais problemas, várias sugestões acerca do que poderia ou não ser feito vieram à tona, trazendo muitos dos requisitos mais importantes para o projeto. Abaixo é apresentada sucintamente a lista de requisitos planejados para o plugin PEP:

1. O plugin deverá operar com a versão mais recente da IDE Eclipse 3.2 Callisto;
2. Deverá suportar desenvolvimento com Java;
3. Deverá operar dentro de uma Perspectiva Eclipse Própria, suportando simultaneamente as principais funcionalidades já disponíveis na perspectiva Java;

4. Deverá suportar conexão apenas entre dois usuários;
5. Deverá suportar conexão ponto a ponto (sem a necessidade de um servidor intermediário, ambos podem operar como “servidor” e “cliente”);
6. Comunicação deverá suportar protocolo TCP;
7. Deverá possibilitar a digitação do endereço do servidor remoto quando operando em modo cliente;
8. Deverá possibilitar a criação de um servidor de comunicação local (ouvidor de conexões);
9. Deverá disponibilizar um comando para desconexão;
10. Deverá possuir a funcionalidade de chat de texto interno e, opcionalmente, de voz ;
11. Deverá sincronizar os seguintes comandos mínimos: digitação de código, colagem de código (copiar e colar), remoção de código, seleção de código, seleção e troca de arquivo, movimentação de cursor, movimentação de scroll, compilação, execução;
12. Deverá possibilitar a abertura de múltiplos arquivos simultaneamente;
13. Deverá disponibilizar um mecanismo de verificação de integridade sob demanda;
14. Deverá realizar verificações de integridade automáticas;
15. Deverá comunicar ao usuário quando da ocorrência de eventos como: perda de sincronismo (integridade afetada), perda de conexão, recebimento de arquivos, impossibilidade de execução de comandos e demais erros;

16. Deverá realizar um registro em log das principais operações e erros ocorridos, possibilitando o rastreamento dos eventos e depuração;
17. Deverá permitir que apenas um membro do par realize alterações de cada vez;
18. Deverá bloquear a possibilidade de edição para o usuário que não está no comando;
19. Deverá disponibilizar uma opção para solicitar/liberar o poder de alteração de código;
20. Deverá disponibilizar uma opção para compartilhamento/envio de projetos inteiros;
21. Deverá disponibilizar uma opção para envio de arquivos individuais inteiros, para as situações em que ocorrer perda de sincronismo;
22. Deverá solicitar permissão ao usuário para sobreposição de arquivos;
23. Deverá realizar o envio de alterações somente quando dentro da perspectiva própria;
24. Deverá possuir ícones característicos do plugin para os comandos disponíveis;
25. Deverá ser de fácil instalação no Eclipse.

### 3.1.2 Definição da Visão Geral do Sistema

A etapa de definição geral do sistema teve como objetivo principal tentar definir, através de um método característico, os componentes macro que fariam

parte do sistema, gerando um composto capaz de executar fundamentalmente, os seus requisitos. A exemplo do que ocorre em XP, onde não há uma preocupação em pré-definir o sistema em um projeto grande e detalhado, no método utilizado para o desenvolvimento do plugin, a etapa de visão geral do sistema não se preocupou em exaurir as minúcias do sistema que seria concebido, mas sim, através de um pequeno estudo, discussão e testes com implementação, quando necessário, tentar obter uma visão dos componentes principais e da arquitetura que melhor atenderia as necessidades estipuladas.

O modo de funcionamento do plugin, assim como já foi mencionado anteriormente, deve seguir, acima de tudo, os princípios da programação em par. Dessa forma, não há melhor cenário para apresentar a sua dinâmica, senão através da esquematização do software interagindo com os membros do par em cada uma das pontas. Para melhor entender de forma concisa uma visão geral de operação do PEP, obtida pela aplicação do método acima citado, apresentamos o seguinte esquema (Ilustração 2):

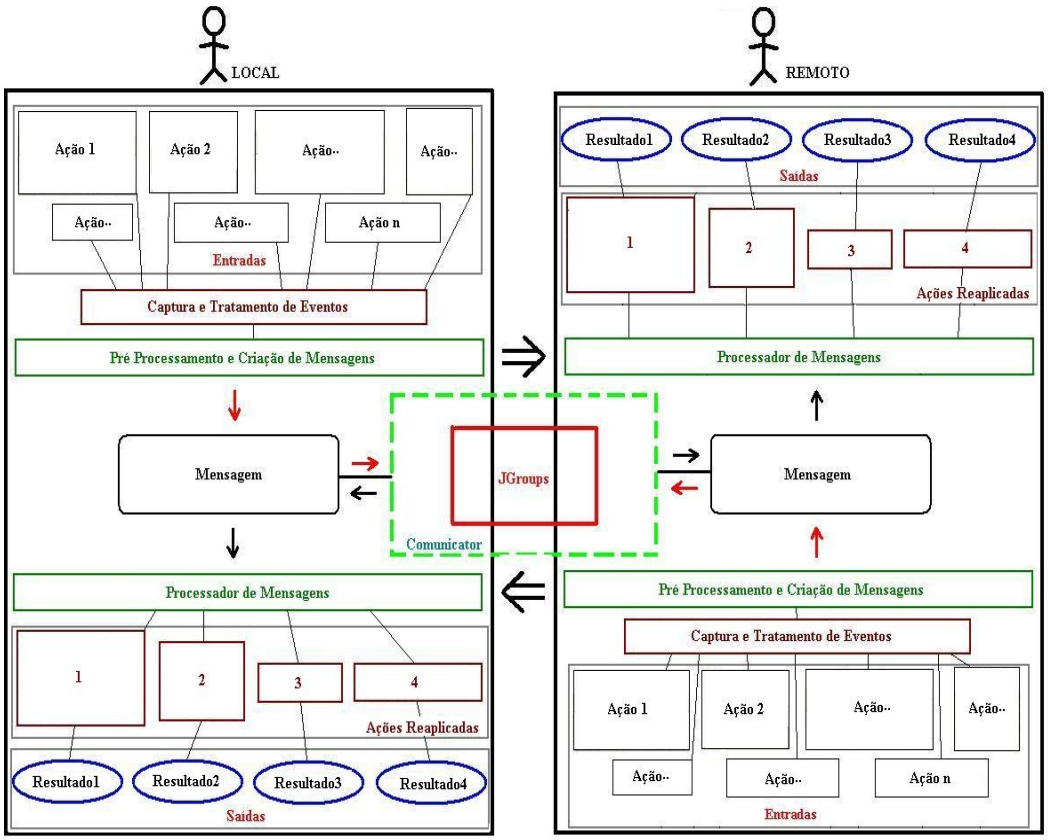


**Ilustração 2: Visão Geral do Sistema**

Como primeiro pressuposto para a análise do esquema na figura acima (Ilustração 2), pode-se observar a clara divisão em dois grandes segmentos: no segmento da esquerda está representado o membro do par que detém o controle da digitação no momento. A exemplo do que ocorre na programação em par não distribuída (PP), enquanto este membro detém o comando, o membro parceiro não deve ter acesso à execução de comandos, o que ficou representado na figura na parte direita da imagem. Neste esquema, portanto, enquanto o plugin da esquerda tem a responsabilidade de enviar as ações de alterações de código correspondentes ao que o membro esquerdo do par executou, o plugin da direita fica

com a atribuição de receber estas informações e disponibilizá-las ou aplicá-las de forma que o membro direito do par tenha a percepção de que as ações ocorreram localmente. Esta configuração, naturalmente, e, como ocorre também na programação em par não distribuída, deve permitir a inversão dos papéis a qualquer momento, desde que haja demanda através de comando executado por qualquer um dos membros do par.

O plugin executado no lado do membro que detém o controle (esquerda) é responsável pela detecção e captação de todas as ações produzidas localmente para mais tarde encaminhá-las ao membro remoto. Tais eventos podem ser provenientes tanto de ações praticadas diretamente através de botões disponibilizados ao usuário, quanto de eventos gerados pelos componentes da plataforma Eclipse ao serem acionados. Cada um destes eventos selecionados, independentemente de sua fonte originadora, devem, antes de serem encaminhados ao destino, passar por uma etapa de pré-processamento, na qual o evento originador é analisado e suas informações relevantes para o processo futuro de reprodução no ponto remoto são adequadamente selecionadas. Isto se faz necessário, pois nem sempre todas as informações geradas por um evento captado localmente são necessárias. Nesta mesma fase, após a coleta do conjunto mais relevante, este pacote é transformado em uma mensagem compreensível pelo plugin remoto e serializado para o envio. Para cada tipo de ação ou evento que é processado, portanto, é criada uma mensagem correspondente, contendo todo o conjunto de informações úteis ao processo de reprodução que será executado remotamente.

A responsabilidade pelo controle, envio e recepção das mensagens geradas fica para o elemento “comunicador”. De maneira geral, para o membro originador, suas atribuições se restringem a receber pacotes de mensagens prontas para o processo de serialização, executar a serialização e o envio propriamente dito. Para o terminal remoto, de maneira análoga, o elemento comunicador fica com a responsabilidade de receber o pacote através do canal de rede escolhido, identificar o tipo de mensagem, validar o conteúdo recebido, com base em um conjunto de mensagens conhecidas e interpretáveis, e fazer a chamada ao processador de mensagens correspondente.

Cada processador de mensagens detém o conhecimento necessário para validar o conteúdo da mensagem e para reproduzi-la, caso a validação ocorra com sucesso. Para tanto, o processador interage com os objetos da plataforma responsáveis pelo controle das ações relacionadas, ou seja, os objetos capazes de reproduzir o evento a partir das informações recebidas. Com a reprodução transparente, estes eventos geram, para o membro remoto (direita), a percepção de que houve uma intervenção.

### 3.1.3 Definição dos Casos de Uso de Alto nível

Com a definição da visão geral de sistema, em associação com a definição de requisitos foram definidos os casos de uso de alto nível para o plugin PEP. Ao invés de explorar minuciosamente cada caso de uso individualmente, como tradicionalmente este processo é executado em metodologias mais “pesadas”, na

metodologia proposta para o desenvolvimento deste projeto os casos de uso foram utilizados apenas como guias para a manutenção do foco no desenvolvimento iterativo. Por se tratar de um desenvolvimento nos moldes de uma metodologia ágil, o processo utilizado no PEP foi planejado para suportar casos de uso contendo apenas algumas informações mínimas que permitam a compreensão das metas da iteração, de forma a priorizar a produção de código correto, em detrimento de uma documentação detalhada.

A definição dos casos de uso de alto nível foi executada com base em reuniões centradas em dois elementos principais: a lista de requisitos e a especificação da visão geral do sistema. Abaixo são destacados os casos de uso de alto nível sobre os quais todo o desenvolvimento foi baseado. No capítulo posterior, cada um deles será detalhado:

- Executar Plugin;
- Comunicar;
- Transmitir Alterações;
- Dirigir/Não Dirigir;
- Checar sincronismo de código;
- Controlar Visão de Código;
- Compartilhar Projeto;
- Comunicar via Chat;
- Forçar Resincronismo;

- Instalar Plugin;

### 3.1.4 Desenvolvimento por Ciclos Iterativos

Uma vez determinados os casos de uso de alto nível, na metodologia de desenvolvimento utilizada para o PEP, o próximo passo realizado foi a divisão dos mesmos em ciclos iterativos. Um ciclo iterativo de desenvolvimento, na prática, pode ser entendido como um período curto (normalmente não ultrapassando uma semana), no qual um objetivo definido por um caso de uso de alto nível é implementado de forma iterativa, com análise, projeto, implementação e testes, com estas atividades repetindo-se até que o objetivo seja concluído. Todo o ciclo é voltado para a obtenção de um alto nível produtivo em termos de código e, assim, não há preocupação com a geração de uma documentação completa, mas sim com um código de elevada legibilidade e corretude.

A divisão do desenvolvimento em ciclos iterativos, no planejamento da implementação, em conseqüência da elevada clareza e fácil separação obtidos na definição dos casos de uso de alto nível, resultou na simples associação de cada um destes a um ciclo iterativo. Na divisão resultante, portanto, a cada ciclo iterativo um dos casos de uso de alto nível foi explicado até que suas funcionalidades propostas estivessem operacionais por completo. Nos itens que se seguem, cada um destes ciclos é explorado em detalhes, descrevendo a etapa de desenvolvimento em si.

### 3.1.4.1 Ciclo Iterativo “Executar Plugin”

Objetivo: criar a infra-estrutura completa de um plugin para o eclipse com as funcionalidades mínimas que permitam sua execução, com a capacidade de extensibilidade e com a abertura de perspectiva própria de desenvolvimento PEP.

Requisitos Envolvidos: 1, 2, 3, 16, 25

Descrição

Uma vez que a proposta do PEP é a de operar como um plugin capaz de agregar a funcionalidade de possibilitar a programação em par distribuída dentro do ambiente Eclipse, o primeiro passo para a criação do software proposto é, portanto, a criação da estrutura básica que compõe o plugin. Embora, como já foi apresentado em capítulo anterior, a IDE já conte com uma perspectiva própria para o desenvolvimento de plugins, a PDE, a implementação de plugins requer a aquisição de um leque de habilidades e conhecimentos cuja obtenção exige um estudo e forte interação com listas de discussão sobre o tema. Subdividimos o processo, portanto, na implementação deste ciclo iterativo, nas seguintes etapas: estudo dos tópicos necessários à implementação da infra-estrutura do plugin, criação da infra-estrutura do plugin e criação da perspectiva PEP.

Em virtude da grande popularidade obtida pela plataforma Eclipse e sua capacidade de extensibilidade, por já haver sido projetado para tal fim, e, por ser um projeto de código Livre, há uma boa documentação de referência para os interessados na criação de plugins. Como documentação de base, duas referências prin-

cipais foram utilizadas: a documentação oficial do Eclipse (ECLIPSE DOC, 2007), onde há disponível uma série de tutoriais, código de exemplo e explicações sobre a plataforma, e a API de desenvolvimento da plataforma Eclipse (ECLIPSE API, 2007), que apresenta no formato HTML uma descrição completa da assinatura de todas as classes disponíveis ao programador. Para várias situações específicas, no entanto, surge à necessidade de conhecer os detalhes internos de funcionamento e programação do Eclipse, situações nas quais há a necessidade de se recorrer às listas de discussão. Nestes casos, duas fontes de informação se destacaram: as listas de discussão oficiais do Eclipse (ECLIPSE NEWS, 2007) e os canais de suporte na rede IRC[6] para o Eclipse.

Após a obtenção dos conhecimentos básicos necessários, a próxima etapa deste ciclo iterativo foi a criação da infra-estrutura mínima para o plugin. A sua criação propriamente dita, incluindo os arquivos de configuração necessários para que a plataforma reconheça sua presença, estruturação dos diretórios e configuração é feita de forma automática com assistente (wizard) de criação disponível na PDE. Em termos simplificados, basta informar uma série de dados mínimos e solicitar a geração dos arquivos para que toda a estrutura do plugin seja criada. Três elementos dentre o conjunto criado devem ser destacados: o arquivo manifest.mf, com as informações para a configuração de tempo de execução (runtime), o arquivo plugin.xml, descrevendo os pontos de extensão e as extensões já disponíveis e o arquivo build.properties, com as informações de configuração para a transfor-

6 IRC – Internet Relay Chat é um chat de comunicação escrita em tempo real criado em 1988 e baseado no protocolo TCP onde são disponíveis uma série de fóruns chamados canais onde os usuários podem se agrupar por tema alvo para estabelecer conversas grupais ou individuais.

mação do projeto criado em um arquivo JAR[7] correspondente ao plugin, caso um comando de exportação seja disparado. Todos os arquivos gerados já continham as informações básicas necessárias para que o plugin se tornasse executável.

Durante a coleta de informações pelo assistente de criação da PDE para a geração da estrutura mínima para o plugin, um dos elementos solicitados é o nome da classe que servirá como "ponto de entrada" para o plugin. Também chamada de "Activator", de acordo com a nomenclatura específica, a classe escolhida para operar como ponto de entrada é um requisito obrigatório durante a criação do plugin. Após a escolha do nome, o assistente cria automaticamente uma classe que estende a classe abstrata AbstractUIPlugin, obrigando a implementação de alguns métodos. Dentre os métodos mais importantes que necessitam de implementação se destacam os métodos start(), onde é definido o comportamento de partida do plugin e o método stop() onde, da mesma forma, se insere o código executado no momento do fechamento do software. Tais métodos são chamados automaticamente pela plataforma nos momentos apropriados, desde que o a classe seja configurada no arquivo manifest.mf, o que foi feito automaticamente pelo assistente de criação. Logo após a criação da classe PEPActivator, nome dado à classe do projeto destinada à função, e implementação do código mínimo nos métodos citados, a classe foi complementada com as funcionalidades adicionais necessárias a nesta etapa como o método configurePlugin(), onde foi adicionado o comportamento de inicialização e configuração inicial.

7 JAR – Java Archive é um formato de empacotamento de arquivos criado como forma de distribuição de um conjunto de classes Java. Trata-se de um arquivo compactado com a extensão .jar que além das classes compiladas, pode armazenar metadados que podem constituir um programa executável.

Como parte das atividades implementadas na primeira iteração, foi criada a perspectiva de desenvolvimento para programação em par. A exemplo do que ocorre com as demais perspectivas já existentes na plataforma Eclipse, a perspectiva PEP (Ilustração 3) é, na verdade, a implementação de uma extensão e sua criação, em termos práticos, se resume a implementar uma interface chamada PerspeciveFactory (do inglês “fábrica de perspectiva”) e declarar esta classe no arquivo de configuração plugin.xml. Na classe PEPPerspectiveFactory, portanto, foram definidas as características comportamentais e de aspecto principais como as configurações de tamanho, posição e elementos presentes na perspectiva, o que garante ao plugin a possibilidade de disponibilizar ao usuário, uma perspectiva própria para o desenvolvimento colaborativo.

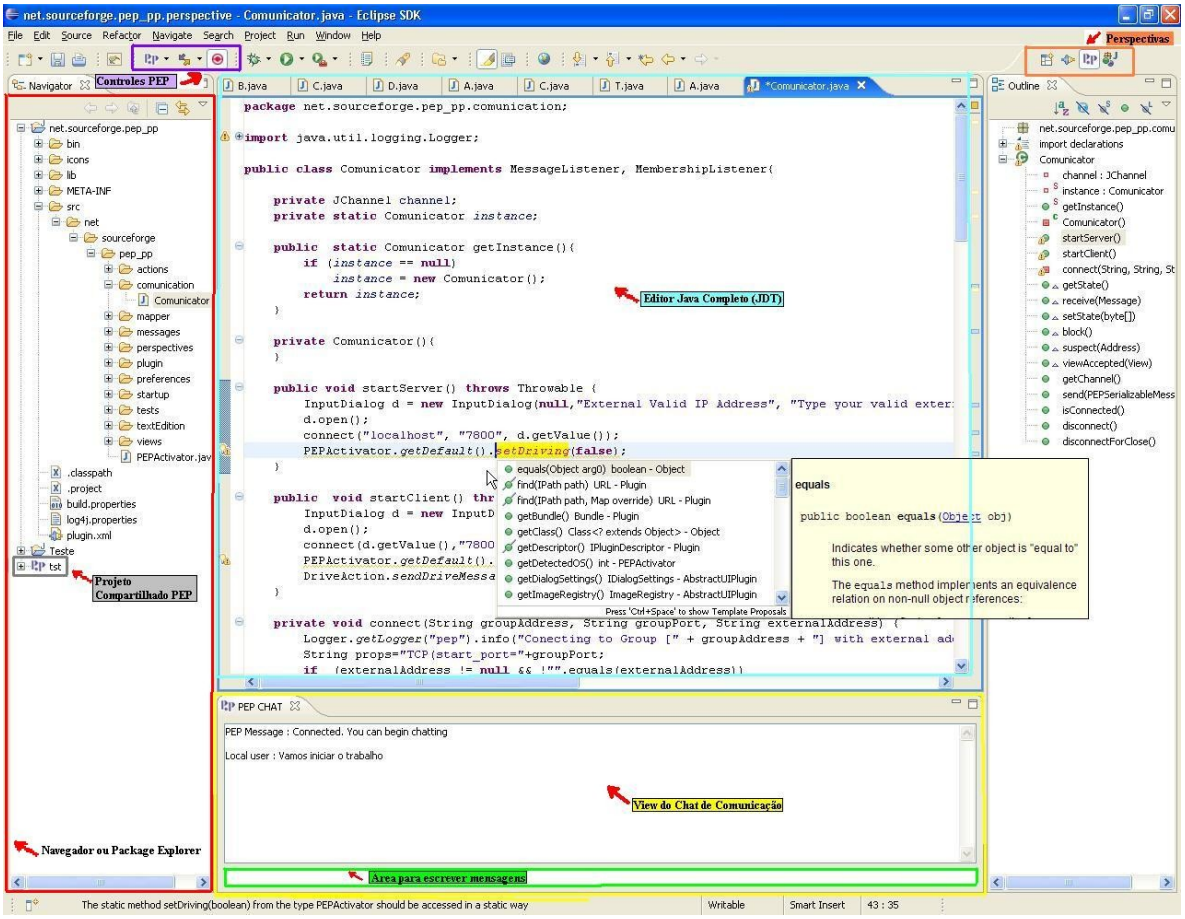


**Ilustração 3: Perspectiva PEP (já com outras funcionalidades implementadas)**

Nesta primeira etapa de desenvolvimento também foram explorados, com uma riqueza maior de detalhamento, os recursos da PDE de suporte ao desenvolvimento de plugins. Um dos principais recursos utilizados, responsável pela facilidade obtida na criação de extensões é a edição das propriedades dos arquivos de configuração (plugin.xml e manifest.mf) em alto nível, através de formulários HTML. Na figura abaixo (Ilustração 4), é apresentado um formulário no qual é possível, por exemplo, dispensando a necessidade de edição direta do arquivo no formato xml, adicionar e configurar uma nova extensão do tipo perspectiva no plugin.

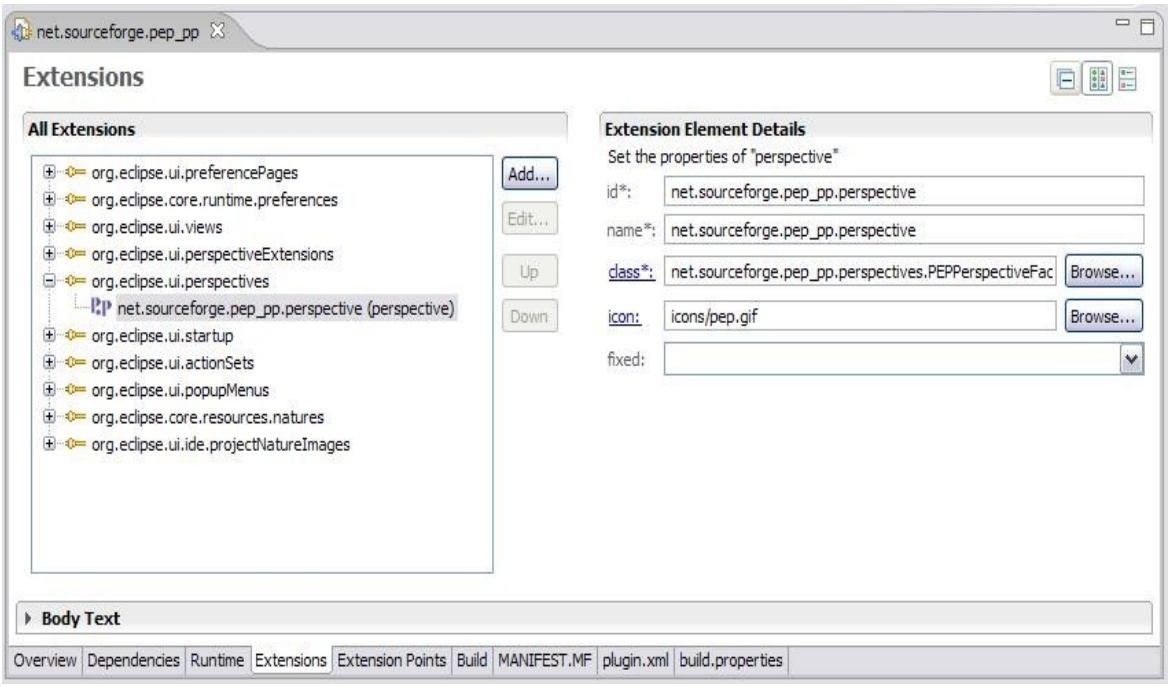


**Ilustração 4: Editor Html das Configurações**

Como última funcionalidade implementada na iteração, como parte dos elementos de infra-estrutura para o desenvolvimento do restante do software, foi adicionada a arquitetura de Log do plugin PEP. Inicialmente configurada para o ambiente de depuração, a classe de logs, que implementa o padrão de projeto Singleton (garantindo uma instância única para todo o sistema), foi inserida no planejamento de todos os ciclos de desenvolvimento, como elemento essencial para a depuração. Assim, para a maior parte das operações ocorridas dentro do sistema, são inseridas chamadas ao Logger para que o mesmo registre, seguindo as configurações aplicadas durante o carregamento do plugin, em arquivo ou diretamente no console as informações relevantes inseridas no código.

#### *3.1.4.1.1 Logger (java.util.logging)*

Recurso adicionado junto ao pacote java.util.logging, a partir do JDK versão 1.4, onde permite que aplicações Java façam o uso de Logging, algo que se assemelha ao comando System.out.println(), no entanto, atua de forma muito mais eficiente, rápida e prática.

**FINEST** – Nível mais baixo. Altíssimo nível de detalhe.
**FINER** – Alto nível de detalhe.
**FINE** – Utilizado para grandes detalhes, quando debugging/diagnosing.
**CONFIG** – Informações sobre settings/setup.
**INFO** – Informações em tempo de execução.
**WARNING** – Mensagens de Aviso.
**SEVERE** – Nível mais alto. Mensagens extremamente importantes (exceptions/erros fatais).

O Logging, como pode ser observado na lista acima, possui 5 níveis atuação, indo do mais baixo, FINEST, utilizado para informações como Debug, até o mais alto, o SEVERE utilizado para mensagens de erros graves no sistema.

Uma funcionalidade bastante interessante é que através de um arquivo de configuração o sistema decide se imprime, ou não, um determinado logger na tela do programador. Assim permitindo maior organização e clareza durante a execução da aplicação.

Como padrão, a JVM (*Java Virtual Machine*) utiliza o arquivo de configuração logging.properties, que se encontra no diretório %JAVA_HOME%/JRE/lib. Este arquivo está configurado, também por padrão, para imprimir apenas os níveis iguais ou superiores a INFO. Sendo possível a alteração do arquivo de configuração ao gosto do desenvolvedor. Os logs, além de impressos, podem também ser registrados em um arquivo do tipo java.log, tal arquivo armazena todos os logs,

com suas respectivas informações(data/hora, classe, nível, mensagem,etc.. ) no formato xml.

A figura a seguir (Ilustração 5) mostra um exemplo da exibição dos logs durante a depuração do plugin na IDE Eclipse:

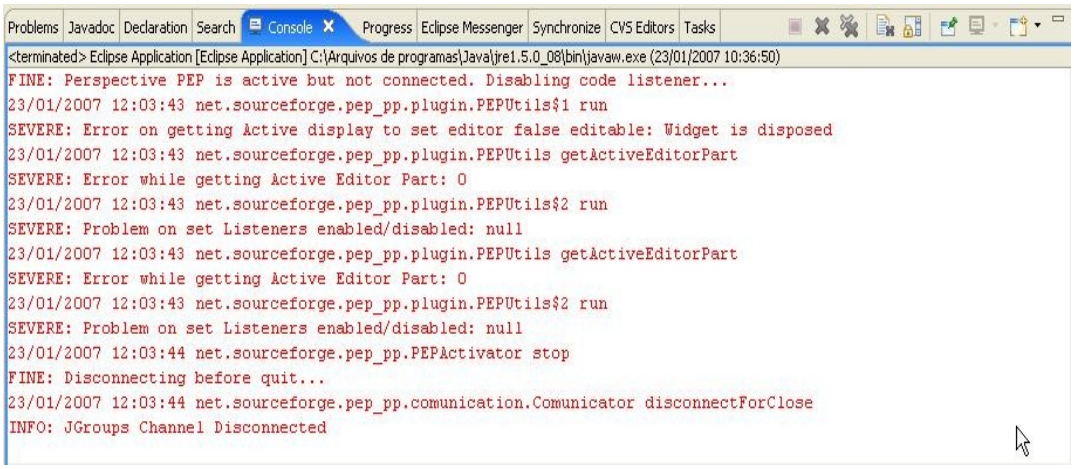


**Ilustração 5:Exemplo de Exibição dos Logs Registrados**

Quando o pacote “java.util.logging” foi criado, baseou seu funcionamento no LOG4J, que também é uma API para logging desenvolvida pelo grupo Jakarta/Apache, a qual surgiu antes da especificação de logging do JDK e é bastante utilizada em vários projetos Java em todo o mundo.

### 3.1.4.2 Ciclo Iterativo “Comunicar”

Objetivo: realizar um processo completo de comunicação fim-a-fim utilizando uma API de comunicação adequada disponibilizando uma infra-estrutura de comunicação de alto nível para a utilização.

Requisitos Envolvidos: 4, 5, 6, 7, 8, 9

Descrição

Um dos elementos cruciais para o sucesso na criação de um ambiente distribuído, certamente se encontra no processo e nas estratégias de comunicação utilizadas. A definição e implementação de uma infra-estrutura de comunicação básica para suportar o envio e recebimento de mensagens para as operações do PEP passou, basicamente, pela execução das seguintes etapas: definição da API de comunicação e protocolos, criação de uma abstração de acesso ao processo de envio e definição da arquitetura mínima para a criação de um mecanismo extensível de envio e recebimento de mensagens.

Embora a API nativa do Java dê suporte a muitos dos protocolos e estratégias de comunicação que se possa utilizar ou criar, atualmente há vários projetos de código livre criados pela comunidade ou pelas empresas para facilitar o uso destes protocolos. Se por um lado a criação de uma abstração própria utilizando as classes nativas garante uma excelente flexibilidade no projeto, por outro, implica em correrrmos o risco da reimplementação de uma solução já disponível na internet, por exemplo. Com o avanço mundial contínuo da engenharia de software, muitas das aplicações modernas já utilizam dos benefícios da reusabilidade, deixando de adotar estratégias de implementação própria em prol da adoção de componentes de código livre conhecidos em larga escala. Em termos de API´s de comunicação, são exemplos deste tipo o projeto JXTA, JGroups, XMPP, entre outras. Embora nem todas estabeleçam seus processos de comunicação exatamente da mesma maneira, tratam-se de componentes que disponibilizam ao projetista, formas inteligentes, livres, gratuitas (na maioria das vezes), código amplamente

utilizado na comunidade de desenvolvedores, o que garante confiabilidade, além de tudo. Com base nestes e em outros critérios definidos como requisitos para o projeto, a API de comunicação eleita como a mais adequada aos propósitos do plugin PEP foi o JGroups, que além de grande flexibilidade, agrega características de excelente desempenho e confiabilidade.

#### *3.1.4.2.1 JGroups*

O JGroups é um Framework para comunicação confiável em grupo escrito inteiramente em Java. Tem como princípio a confiança na comunicação entre membros de um grupo.

Confiança:

- Garantia de transmissão de uma mensagem para todos membros do grupo(com o retransmissão de mensagens perdidas).
- Fragmentação de mensagens grandes e reconstrução da mensagem original do lado do receptor.
- Atomicidade: ou uma mensagem é recebida por todos os membros do grupo, ou por nenhum.

Grupo:

- Conhecimento de quem são os membros do grupo
- Notificação de entrada e saída dos membros e se ocorrer alguma conflito.

A tabela a seguir (Tabela 1) mostra realmente onde o Jgroups pretende trabalhar, no entanto, o framework permite diversas configurações diferentes.

**Tabela 1: JGroups e Modos de Operação**

| | Não Confiável | Confiável |
|---|---|---|
| Unicast | UDP | TCP |
| Multicast | IP MultiCast | **JGroups** |

A comunicação pode ser estabelecida utilizando um dos dois protocolos de transporte, o UDP ou o TCP. No entanto, nos dois casos a comunicação é Unicast, o que significa que apenas dois usuários trocam informações por vez (1-1). No caso do UDP, a transmissão não é confiável devido a perda de pacotes. Já utilizando TCP, que possui retransmissão de pacotes perdidos, a comunicação é confiável. Quando se pretende comunicar ao mesmo tempo um grupo de usuários, podemos utilizar um grupo Multicast, que consiste em usar um IP da faixa designada a esta função, que através de suas características, retransmite as mensagens recebidas, através do protocolo UDP, a todos os membros desse grupo Multicast. No entanto, por utilizar o protocolo UDP, se torna um recurso não confiável, além de que, a quantidade e a identificação dos membros do grupo não podem ser determinadas. O JGroups tem por objetivo estabelecer conexões um para muitos (1-N) e normalmente é configurado de uma das seguintes formas: usando TCP e encaminhando com confiabilidade as mensagens recebidas no IP do servidor para todos os membros do grupo, ou com UDP Multicast, onde o JGroups não se preocupa com o encaminhamento das mensagens, mas sim, com a retransmissão de pacotes perdidos durante a transmissão. Para criar um grupo e poder enviar mensagens, uma aplicação tem que instanciar um JChanel (canal) e conectar-se através desse canal ao grupo usando o mesmo nome (todos canais com o mesmo nome formam um grupo). O canal é o une o grupo, sendo assim, enquanto conectado, um membro pode enviar e receber mensagens para todos os outros membros do grupo. Na construção de um JChanel deve-se passar como parâmetro uma String (chamada normalmente “props”) contendo todas as configurações da comunicação, incluindo: nome, protocolo, endereço, porta, número máximo de

membros, entre várias outras. A seguir podemos observar (Ilustração 6) um exemplo de criação de um JChanel passando uma string props configurando-o para o protocolo UDP:

```
String props="UDP(mcast_addr=228.1.2.3;mcast_port=45566;ip_ttl=32):" +
"PING(timeout=3000;num_initial_members=6):" +
"FD(timeout=5000):" +
"VERIFY_SUSPECT(timeout=1500):" +
"pbcast.STABLE(desired_avg_gossip=10000):" +
"pbcast.NAKACK(gc_lag=10;retransmit_timeout=3000):" +
"UNICAST(timeout=5000;min_wait_time=2000):" +
"FRAG:" +
"pbcast.GMS(initial_mbrs_timeout=4000;join_timeout=5000;" +
"join_retry_timeout=2000;shun=false;print_local_addr=false)";
 JChannel channel;
 try {
 channel=new JChannel(props);
 }
 catch(Exception ex) {
 // channel creation failed
 }
```

**Ilustração 6:Exemplo de String de Configuração do JChanel**

A classe encarregada no projeto de estabelecer a conexão deve criar o JChanel, como descrito acima, e além disso, implementar as interfaces MessageListener, MembershipListener. A seguir destacamos os principais métodos dessas classes e suas respectivas funções:

- PullPushAdapter – objeto que habilita o recebimento das mensagens para o JChanel.
- receive() – Interface MessageListener: recebe as mensagens do grupo
- viewaccepted() – Interface MembershipListener: recebe requisições de novos membros.
- send() – envia mensagens para o grupo através do JChanel.

O JGroups é um projeto bem conceituado e seu funcionamento satisfaz os objetivos propostos, é por isso que projetos como Jakarta, JBoss e Tomcat utilizam o JGroups para auxiliar na comunicação. O Framework possui licença LGPL, o que permite que ele possa ser utilizado em qualquer projeto, mesmo que este não seja open source, o que não é o caso do PEP.

Embora a API Jgroups tenha sido uma opção adequada para os propósitos do projeto PEP, é sabido que uma boa prática em termos de qualidade de software é o estabelecimento de garantia de fraco acoplamento de implementações de terceiros. A solução adotada para resolver este ponto foi a criação de uma classe de abstração, denominada “Comunicator”, cujo objetivo maior seria o de encapsular as funções acessadas da API importada, permitindo, em caso de necessidade futura, uma fácil substituição ou adaptação a outra API que se faça necessária. Além disso, a classe comunicator permitiu a dissociação das funções relativas à comunicação de dados ainda presentes no plugin, possibilitando uma melhor organização das funções relacionadas em um único ponto. Como resultado final, a classe Comunicator ficou com as seguintes responsabilidades:

- Manutenção da configuração: ficando com a definição da pilha de protocolos necessária para a conexão com o JGroups;
- Estabelecimento de conexão com o grupo: onde ocorre a criação propriamente dita do JChannel a partir da pilha de protocolos configurada;

- Desconexão do grupo: quando necessário, a classe fica responsável também por chamar o método de desconexão do JChannel criado, além de disponibilizar um ponto para a introdução de código que se faça necessário antes ou após a desconexão;
- Envio de pacotes de mensagens: embora a API do JGroups forneça suporte ao envio de objetos, estes objetos no entanto, devem ser encapsulados por um outro objeto do tipo “JGroupsMessage”, o que fica a cargo da classe Comunicator;
- Recebimento de pacotes de mensagens: com a implementação das interfaces MessageListener e MembershipListener, há a necessidade da implementação dos métodos para o tratamento adequado quando os eventos são recebidos, o que é feito também na classe Comunicator.

Como vimos acima, o envio e recebimento de pacotes, é feito pela classe Comunicator que fica responsável pela conversão deste conteúdo em uma mensagem do tipo JGroupsMessage. Para que um objeto “Message” possa ser enviado, no entanto, ele deve estar em um formato serializável. Em java, a responsabilidade pela identificação de objetos serializáveis é do programador que, para transformar o objeto em serializável, deve implementar a interface Serializable. Para criar uma abstração que simplificasse este processo, eliminando a duplicação dos métodos necessários para todas as mensagens a serem enviadas (como por exemplo, o método para a configuração do tipo da mensagem) para cada novo tipo de mensagem  criado no sistema, foi criada a classe “PEPSerializableMessage”. Des-

sa forma, para criar um novo tipo de mensagem, basta estender esta classe, sobrescrevendo na medida do necessário, os métodos da classe mãe.

Cada novo tipo de mensagem criado no sistema está associado a uma ação que se deseja reproduzir no plugin remoto. Dessa forma, para cada nova mensagem que se cria, há também a necessidade de criação de um processador específico para esta mensagem. É tarefa do manipulador de mensagens, representado pela classe “PEPMessageHandler”, após o recebimento de uma mensagem pelo Comunicator, para cada mensagem que chega, identificar o tipo da mensagem, chamando o método genérico getType() da classe PEPSerializableMessage e, depois disso, chamar o processador correspondente para esta mensagem. A solução mais adequada para criar um código extensível de fraco acoplamento entre as classes foi a utilização do padrão de projeto “Command” baseado em reflexão computacional[8]. Após a identificação do tipo da mensagem recebida, obtido através da chamada ao método getType(), comum a qualquer tipo de mensagem do sistema, este tipo é convertido em um String que é concatenado ao sufixo “Processor”. Depois disso, através do método Class.forName(), que é capaz de instanciar um objeto a partir de um String que representa o seu nome (reflexão), o processador correspondente é criado, e o seu método processMessage() é chamado. O esquema abaixo sintetiza este fluxo:

1. Uma mensagem é recebida;
2. O Comunicator desconverte o pacote recebido em uma mensagem do tipo PEPSerializableMessage;

8 Reflexão computacional: No domínio da informática, o termo reflexão computacional está ligado à capacidade de um programa conhecer e examinar sua própria estrutura e de alterar seu comportamento através do redirecionamento ou interceptação de operações efetuadas.

3. O Comunicator instancia um manipulador de mensagens PEPMessageHandler e encaminha a mensagem para este manipulador.
4. O manipulador extrai o tipo da mensagem através do método getType() e concatena este nome com o sufixo “Processor”.Ex. para a classe “DriveMessage” o handler tentaria localizar a classe “DriveMessageProcessor”;
5. O manipulador tenta instanciar um objeto a partir do String concatenado;
6. Depois de instanciado, o manipulador chama o método processMessage do processador correspondente que contém o código necessário para reproduzir o evento recebido.

Como finalização da iteração, foram implementados também os botões de controle para acesso às funções de conexão e desconexão utilizando a extensão correspondente (Ilustração 7).


**Ilustração 7: Botões da Interface**

### 3.1.4.3 Ciclo Iterativo “Transmitir Alterações”

Objetivo: Realizar um processo completo, fim a fim, de detecção de texto de código digitado, envio destas informações ao membro remoto do par e reprodução das alterações para este membro.

Requisitos Envolvidos: 11, 23

Descrição

Nesta iteração, pode-se afirmar, que foram implementados alguns dos requisitos mais relevantes para o sistema PEP. O desenvolvimento com sucesso de uma arquitetura que suporte a dinâmica de envio e recepção de alterações, uma vez que isto se constitui um dos seus maiores objetivos, faz parte das atividades consideradas essenciais para o sucesso da ferramenta.

Em termos teóricos, sincronizar código é uma tarefa simples que pode ser resumida a: detectar toda e qualquer alteração aplicada ao código, transformar estas informações em mensagens apropriadas para o envio, enviar as alterações pelo canal de comunicação, recebê-las com um tratamento adequado e aplicar as informações de alteração extraídas da mensagem no plugin receptor.

A primeira das grandes tarefas implementadas para a criação da funcionalidade de envio de alterações foi a criação de código para a detecção de alterações. O trabalho de detecção propriamente dito foi realizado através da implementação da Interface ITextListener, nativa da plataforma, criada para operar como uma forma de geração de eventos para objetos do tipo ITextViewer, presentes na composição de todos os editores que são abertos na IDE. Na prática, foi criada a classe PEPTextListener através da implementação do método textChanged() da interface referida. Assim, para qualquer editor que se queira observar as alterações, basta efetuar o registro desta classe “ouvidora”, chamando o método apro-

priado addTextListener(), que recebe como argumento justamente o objeto PEP-TextListener. O funcionamento pode ser esquematizado assim:

1. O objeto PEPTextListener implementa a interface ItextViewer que possui o método textChanged();
2. Assim que um Editor é ativado pelo usuário, este editor ativo é criado um objeto do tipo PEPTextListener;
3. Este objeto é então registrado no Editor;
4. A cada alteração de código, a plataforma trata de chamar o método text-Changed() da classe implementada.

Para cada alteração, um evento é, então, gerado na classe PEPTextListe-ner. Dentre as informações recuperados no evento gerado estão: o offset (posição no texto onde a mudança ocorreu), o length (tamanho do texto alterado), o text (texto inserido) e o replacedText (texto que foi substituído). De posse destas infor-mações é criado, então, um objeto do tipo DocumentMessage, encapsulando as informações relevantes para o transporte até o membro remoto. A cada alteração, todo o ciclo se repete.

O envio da mensagem é feito com uma chamada ao objeto Comunicator que espera, como já foi mencionado anteriormente, objetos que estendam a clas-se PEPSerializableMessage.

Quando uma mensagem é recebida, o tratamento fica a cargo da classe DocumentMessageProcessor. Durante este tratamento, as informações são cole-tadas da DocumentMessage e é iniciado o processo de reprodução do evento.

Este processo é relativamente simples, guardadas as devidas proporções em relação à complexidade da plataforma Eclipse. Simplificando o processo, podemos afirmar que, quando uma alteração é recebida, é criada uma thread que fica responsável pela extração das informações serializadas no objeto DocumentMessage. Tendo o texto que foi inserido, o texto que foi removido e a posição, esta String é substituída no texto do Editor ativo através de uma chamada ao método Document.replace().

A última das funcionalidades implementadas na iteração, cumprindo a definição dos requisitos, foi a criação do mecanismo de ativação e desativação do ouvidor de alterações. Como foram definidos, os eventos de detecção de alterações somente podem ser dispadados quando a perspectiva PEP está ativa, evitando overhead de processamento nas situações em que isto não é necessário. Para que isto fosse possível, foi necessária a criação de um novo objeto ouvidor, aqui chamado de PEPPerspectiveListener, cuja responsabilidade foi a de observar os eventos de ativação e desativação da perspectiva de forma a identificar as horas em que o ouvidor de código deveria estar ativo.

#### 3.1.4.4 Ciclo Iterativo "Dirigir/Não Dirigir"

Objetivo: disponibilizar ao usuário um mecanismo de troca de indivíduo no comando, bloqueando a possibilidade de alteração quando não for apropriado.

Requisitos Envolvidos: 17, 18, 19

Descrição: O mecanismo escolhido para simular a troca explícita de membro do par no comando da digitação foi a inclusão de um botão na interface (Ilustração 8) para possibilitar este comportamento. A nova “action” incluída, nome efetivamente dado à extensão que representa o botão, foi configurada para operar no modo push/pull, ou seja, com um clique o botão se ativa e com um novo clique o botão é desativado. A criação desta extensão é uma atividade simples que pode ser executada diretamente através do uso da interface HTML de edição do arquivo plugin.xml da PDE. Dentre as suas configurações estão o tipo de botão, aqui definido como “push-pull” e ícone que será utilizado quando o plugin estiver em execução. Durante este processo, o próprio assistente disponível na página de edição realiza a criação da classe onde ficará o comportamento que ocorre quando o botão é acionado.

**Ilustração 8: Botão Dirigir**

**Ilustração 9: Botão Dirigir Ativo**

Quando o botão Dirigir é pressionado, ele deve estar em um dos dois estados descritos na figura acima (Ilustração 8, Ilustração 9), este evento deve ser comunicado ao plugin remoto. Assim como nas demais ações já explicadas no funci-

onamento do plugin, para cada evento de acionamento do botão gerado, é criada uma mensagem específica para o envio. A mensagem DriveMessage, portanto, possui a responsabilidade de avisar ao elemento remoto que o mesmo deseja comandar  ou liberar o comando de controle do código. Embora sutil, este detalhe que permite tanto ao comandante quanto ao comandado solicitar ou liberar o comando é uma característica cuja intenção é criar uma associação cognitiva que permita o aumento da usabilidade, uma vez que gera um link simbólico com a atividade prática em si (na programação em par, qualquer um dos membros, com facilidade, pode migrar de comandante para comandado e vice-versa).

Duas atividades são necessárias quando uma mensagem DriveMessage é recebida: configurar o plugin local de acordo com a situação (comandando ou não comandando) e, programaticamente, alterar a situação do botão de controle de acordo com esta mesma situação.

Como última funcionalidade implementada para a finalização da iteração, foi idealizado um mecanismo de controle de bloqueio e desbloqueio de edição com o seguinte comportamento: quando o usuário local está dirigindo, o usuário remoto não deve ter acesso à modificação (somente leitura e visualização) e quando o usuário remoto estiver dirigindo o contrario deve ocorrer. A partir do momento em que o plugin é instalado na IDE, estes controles entram em execução, bloqueando a edição para o usuário local, caso o mesmo não esteja programando em Par.

#### 3.1.4.5 Ciclo Iterativo “Checar Sincronismo de Código”

Descrição: disponibilizar duas formas, uma automática e uma sob demanda de se verificar se uma determinada classe ou arquivo compartilhados estão sincronizados adequadamente ou se há diferenças entre os arquivos de cada membro do par.

Requisitos Envolvidos: 13,14

Descrição

Uma das principais deficiências das ferramentas Sangam e Collab, no que tange ao processo de transmissão e sincronismo de código, é a inexistência de mecanismos de detecção de inconsistências entre as versões local e remota no par. Deixar de tratar esta situação significa aceitar que os eventos de comunicação e processamento de alterações são infalíveis, o que não é verdade. Desde problemas no tráfego, queda repentina do link de comunicação e até mesmo em decorrência de problemas locais que podem ocorrer durante a aplicação das alterações recebidas, todas são situações que devem ser previstas para que o nível de confiabilidade e robustez do software implique na sua aceitação pelos usuários.

Os mecanismos de sincronismo implementados no PEP para a disponibilização da funcionalidade de verificação de sincronismo de código podem ser divididos em dois segmentos: uma verificação simples por tamanho e uma verificação completa através de hash. A verificação por tamanho consiste em uma checagem de sincronismo através da comparação do tamanho do texto contido no editor sendo sincronizado no momento. Por ser um processo relativamente rápido, uma vez que consiste na simples contagem da quantidade de caracteres do String que representa o texto contido no editor, a verificação de integridade por tamanho é apli-

cada após a ocorrência de cada alteração. A informação sobre o tamanho do texto é recalculada a cada alteração local e inserida junto no pacote que representa este evento (DocumentMessage).

O processo de verificação de sincronismo através de hash utiliza como método de comparação o algoritmo de “Hash MD5”. Algoritmos de hash, também chamados de algoritmos de dispersão, têm como característica principal a capacidade de gerar, para determinado conteúdo de entrada, uma seqüência de letras ou números de tamanho fixo cuja probabilidade de se repetir, para um texto diferente na entrada, é muito pequena. Em outras palavras, devido a esta característica, algoritmos de hash podem ser utilizados para identificar um arquivo ou uma informação qualquer unicamente. O algoritmo MD5 é uma função de hash que gera uma cadeia de saída de 128 bits e que pode ser utilizado para este propósito.

Apesar de mais seguro, o processo de verificação de integridade no sincronismo através de hash consome um tempo maior de processamento, o que inviabiliza o seu uso após cada evento de alteração, a exemplo do que foi implementado com a verificação por tamanho. Este processo é executado, portanto, em duas situações: sob demanda, assim que o usuário solicita uma verificação de integridade no sincronismo clicando no botão correspondente na interface do plugin (Ilustração 10), e durante a verificação completa de integridade, que ocorre sempre que um erro de integridade por tamanho é detectado.

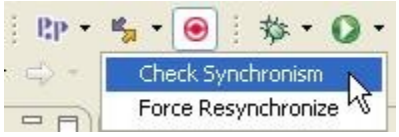


**Ilustração 10: Botão para Checar Sincronismo**

Para o processo de verificação de integridade por hash, ao contrário da verificação por tamanho, que é encapsulada na própria mensagem DocumentMessage, foi criada uma mensagem específica, contendo o hash calculado para o documento suspeito. Sempre que o processo se inicia, ocorre uma troca de mensagens entre plugin local e remoto, onde cada ponto calcula o hash do documento ativo e envia ao plugin remoto. Caso a comparação dos hash´s recebidos resulte em diferença o sincronismo é considerado inválido e mensagens correspondentes são apresentadas aos usuários.

#### 3.1.4.6 Ciclo Iterativo "Controlar Visão"

Objetivo: Criar, através do envio de informações relativas à visão e o controle do editor utilizado, a impressão de uso local para o membro remoto.

Requisitos Envolvidos: 11,12

Descrição

Causar a impressão de acesso local para a operação distribuída de edição de código é uma tarefa que depende da detecção e envio de várias informações além do texto que é digitado dentro do editor. Operações como a seleção de texto, scroll de tela e a mudança de posição do cursor são exemplos de atividades básicas cuja implementação facilita a criação desta percepção aos usuários.

As operações eleitas para cumprir o ciclo iterativo foram as seguintes:

- Detecção e transmissão do evento de seleção de texto;
- Detecção e transmissão do evento de mudança da posição do cursor no texto do editor;
- Detecção e transmissão do evento de “scroll” com teclado ou mouse;
- Detecção e transmissão do evento de seleção de arquivo aberto.

A implementação dos três primeiros dos itens acima foi, em muito, facilitada pela possibilidade de reuso de software disponível no software Sangam. Apesar de a estrutura das mensagens não se assemelhar com o esquema criado para o PEP, grande parte do código de detecção dos eventos e extração das informações relevantes pôde ser aproveitado com pequenas alterações. A detecção e transmissão do evento de seleção do arquivo aberto, no entanto, não havia sido implementada no referido plugin e, portanto, não foi possível reutilizar o código para esta função.

#### 3.1.4.7 Ciclo Iterativo “Compartilhar Projeto”

Objetivo: disponibilizar ao usuário um mecanismo simples para o compartilhamento inicial dos projetos.

Requisitos Envolvidos: 20

Descrição

Uma das grandes deficiências do projeto Sangam, cuja proposta se assemelhava com a proposta do plugin PEP, era a necessidade que existia de se utilizar uma ferramenta externa para realizar o processo de sincronismo inicial de determinado projeto. Além de causar desconforto para o usuário, a exigência de um utilitário externo se traduz em uma forte desvantagem da ferramenta como proposta de solução para o problema de programação em par.

No projeto da função de compartilhamento as seguintes operações foram planejadas para o atingimento da funcionalidade: criação de um menu do tipo “popup” para acionamento da função(Ilustração 11), criação de funções de compactação e descompactação de pacote de projetos e implementação de código para abertura programática pós recebimento.

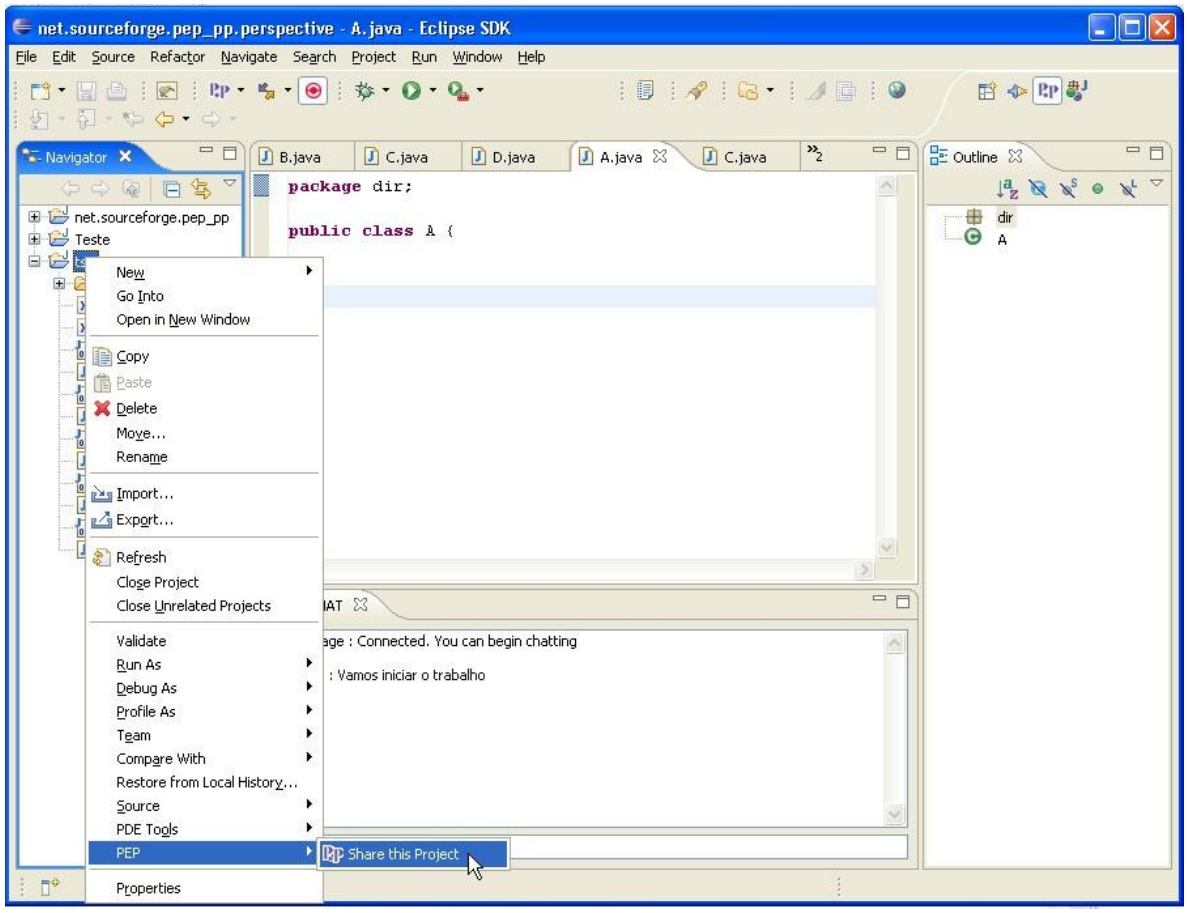


**Ilustração 11: Popup Menu para Compartilhamento do Projeto**

O menu “popup”, criado como uma forma de acionamento da função de compartilhamento, foi implementado através da aplicação de uma extensão denominada “popupMenus” pertencente à plataforma. A configuração para acionamento foi feita de forma programática, de forma que o menu somente se tornasse disponível com o clique do botão direito do mouse sobre um projeto válido.

A melhor solução identificada para a transmissão do projeto, que na prática é um conjunto de arquivos e diretórios, foi a transformação deste composto em um arquivo único, compactado através da utilização a API “java.util.zip”. A tarefa maior de implementação, neste caso, foi a criação de métodos para a compactação e descompactação de diretórios, a partir da documentação e exemplos disponíveis

na internet. Para o trânsito do arquivo compactado, foi implementado um novo tipo de mensagem, desta vez contendo, como informação mais relevante, um conjunto de bytes representando o arquivo zip enviado.

Quando uma mensagem de compartilhamento de projeto é recebida, há a necessidade de realizar a importação e abertura programática deste projeto dentro do ambiente de trabalho (workspace) do usuário. Após a extração do arquivo zip da mensagem e descompactação do mesmo, foi necessária a criação de código específico para a execução destas tarefas.

#### 3.1.4.8 Ciclo Iterativo "Comunicar via Chat"

Objetivo: Disponibilizar um mecanismo de comunicação textual interno ao plugin, capaz de operar como chat.

Requisitos Envolvidos: 10

Descrição

A criação do PEPChat foi, em grande parte, facilitada pela existência de código reusável disponibilizado de forma aberta e gratuita através do plugin Sangam. Embora várias de suas características exigissem alteração e adaptação, toda a infra-estrutura do chat pôde ser analisada como fonte de informações sobre a melhor forma para se implementar tal funcionalidade. De forma simplificada, nesta ite-

ração, para que a funcionalidade do Chat pudesse ser disponibilizada, as seguintes tarefas foram executadas: criação da área específica na perspectiva para a janela do chat, criação da mensagem para o tráfego de frases entre os membros do par e tratamento de mensagens recebidas.

A criação da janela para a inserção da área do chat foi feita através da extensão do ponto de extensão “views’, ou seja, a área do chat (Ilustração 12) é, na verdade, uma view, que é o nome dado à região da plataforma destinada à criação de conteúdo em pequenas janelas no interior do ambiente. Assim como nas demais extensões criadas até o momento, para esta extensão também foi necessária a criação de uma classe específica para o tratamento dos seus eventos relacionados. Assim, na classe PEPChat foi inserido todo o comportamento relacionado ao tratamento do tráfego de mensagens na view do Chat, incluindo o comportamento de scroll da tela automático, após a alteração do conteúdo, e tratamento do “ouvidor” para a tecla “enter” permitindo envio de mensagens através da chamada pelo teclado.

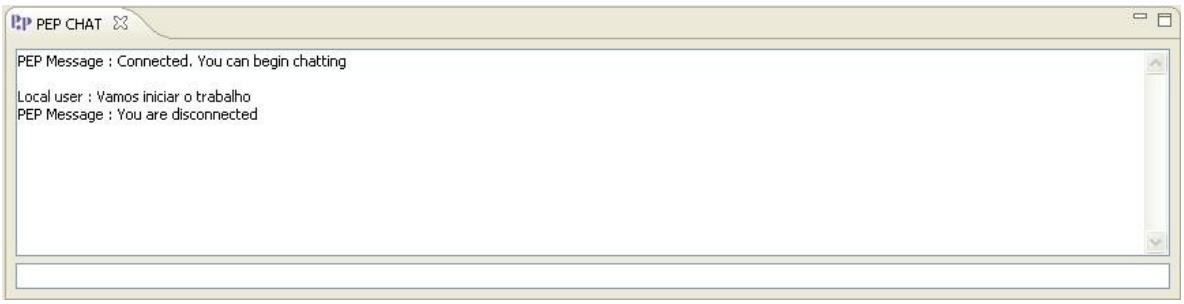


**Ilustração 12: View do Chat**

Além do tratamento de mensagens recebidas para apresentação na view, foi tratada também, nesta iteração, a infra-estrutura para a apresentação das men-

sagens informacionais, como mensagens de conexão, desconexão e recebimento de arquivos, diretamente através do chat.

#### 3.1.4.9 Ciclo Iterativo "Forçar Resincronismo"

Objetivo: disponibilizar, para as situações onde houver a necessidade de resincronismo de arquivo após detecção de inconsistência, uma opção para restabelecer a consistência.

Requisitos Envolvidos: 21,22

Descrição

Como já foi apresentado anteriormente, o plugin PEP contou com um mecanismo confiável de detecção de inconsistência no sincronismo dos arquivos local e remoto (Ilustração 13). Até o momento, no entanto, não foi apresentada nenhuma alternativa para a recuperação do sincronismo para estes casos. No planejamento inicial do projeto, a intenção idealizada como solução para este problema envolvia a implementação de um sistema mais flexível de resincronismo com suporte, se possível a merge, a exemplo do plugin CVS[9] já disponível originalmente no Eclipse. Durante o planejamento do cronograma, no entanto, chegou-se à conclusão que esta tarefa, de menor prioridade, seria executada em uma forma mais simples

9 CVS – Control Version System, ou sistema de controle de versões é um plugin que acompanha o pacote de instalação default do eclipse com suporte a vários recursos de forma visual e simplificada para permitir a gerência de configuração de projetos.

de forma a otimizar o uso do tempo. A maneira mais simples encontrada para resolver este problema foi, através do reuso da infra-estrutura criada para o compartilhamento de projetos, permitir o resincronismo de outros arquivos além de projetos, forçando a sua sobrescrita no terminal remoto. Para garantir uma robustez maior ao processo, foi implementada também a solicitação de confirmação para sobrescrita no terminal remoto, assim que a mensagem de resincronismo é recebida. Caso a confirmação seja fornecida pelo usuário remoto, a IDE, automaticamente, realiza a atualização do arquivo aberto na tela, de forma que esta funcionalidade não implicou na necessidade de implementação de código específico para o fim.


**Ilustração 13: Forçar Sincronismo**

### 3.1.4.10Ciclo Iterativo “Instalar Plugin”

Objetivo: Disponibilizar um pacote de instalação para permitir a implantação do plugin no Eclipse de forma simplificada.

Requisitos Envolvidos: 24,25

Descrição

Para que o plugin possa ser disponibilizado para download e instalação no Eclipse, há a necessidade da disponibilização deste projeto na forma de um arquivo ".jar". Basicamente, para que isto seja possível, foi necessária a realização de uma chamada ao assistente de exportação. Um detalhe relevante a observar, neste ponto, é que sempre que um pacote é importado no projeto, o arquivo manifest também deve ser atualizado para conter as referências corretas aos pacotes importatos, caso contrário, exceções de classe não definida serão chamadas.

### 3.1.5 Outras Ferramentas Utilizadas

Além das ferramentas já citadas utilizadas durante o desenvolvimento do PEP, dois outros recursos foram facilitaram o desenvolvimento do software: o portal Sourceforge, que fornece recursos gratuitos para o desenvolvimento de software livre, e o plugin CVS do Eclipse, utilizado como ferramenta para o controle de versões.

#### 3.1.5.1 Sourceforge

O Sourceforge é um web site reconhecido pela comunidade na internet como sendo o maior repositório para o desenvolvimento de software livre do mundo. Sem cobrar pelos serviços, o Sourceforge oferece aos interessados no desenvolvimento ou colaboração par ao desenvolvimento de software livre um conjunto de serviços de suporte. Os principais deles são:

- Repositório para controle de versões com CVS ou SVN;

- Web site para controle de bugs e novas funcionalidades;
- Espaço na web para publicação de uma home page para o projeto;
- Listas de discussão;
- Fórum;
- Espaço para disponibilização de arquivos para download;
- Espaço para publicação da documentação;
- Sistema para controle de acompanhamento o projeto (milestones).

Deste conjunto, três recursos foram utilizados com mais ênfase: o repositório CVS, utilizado como sistema de controle de versões durante todo o desenvolvimento, o espaço para publicação da home page, onde foi inserido um conteúdo web simples com informações adicionais sobre o projeto, e o espaço para a disponibilização de arquivos para download, onde foram inseridos os “releases” gerados do PEP.

#### 3.1.5.2 Plugin Eclipse para Controle de Versões com CVS

Já vimos que o repositório de controle de versões utilizado durante o desenvolvimento do pep foi disponibilizado no portal Sourceforge, em conjunto com os outros serviços. O serviço de lá utilizado, no entanto, se restringe ao espaço para a criação de repositórios para controle de versões, ou seja, o servidor CVS. Todo

o processo de inserção, alteração e remoção de arquivos do repositório, no entanto, deve ser feito com uma ferramenta externa, tradicionalmente chamada de “cliente cvs”.

O pacote de instalação do Eclipse na versão mais nova já acompanha o plugin CVS, que implementa um cliente CVS de controle de versões. Vários recursos avançados são lá disponíveis como o suporte a “merge” e “diff” diretamente no interior do editor.

# 4 Conclusão

## *4.1 Considerações Finais*

O desenvolvimento do plugin PEP pode ser considerado uma obra com uma contribuição multilateral, capaz de atuar beneficamente em vários sentidos. Em primeira instância, o plugin foi planejado com um propósito verdadeiro de contribuir com um problema ainda não resolvido, surgido com a criação da nova metodologia DXP (*KIRCHIER et. Al., 2001*) na disponibilização de ferramentas que adequadamente suportem a atividade. Em uma segunda análise, sob o ponto de vista da excessiva complexidade inerente do projeto, pode-se analisá-lo como uma forma desafiadora de se obter habilidades em vários aspectos técnicos tais como a criação de plugins para o Eclipse. Em último grau, a própria metodologia de desenvolvimento aplicada, que apesar de cientificamente já haver tido seus benefícios comprovados, ainda não obteve o respaldo e a

popularidade na comunidade de desenvolvimento, tornou-se uma fonte ideal para o exercício de suas práticas e uma possível fonte de informações para novos aspirantes ao método. Nos capítulos que se seguem estas e outras conclusões obtidas no desenvolvimento do PEP serão apresentadas.

## Sobre a Metodologia e as Ferramentas Utilizadas no Desenvolvimento

Um dos grandes desafios que lançamos, logo que a idéia da criação de uma ferramenta de suporte ao Pair Programming foi concebida, foi a de tentar utilizar a metodologia DPP no próprio desenvolvimento do plugin. Embora o julgamento da bilbiografia referenciada já houvesse atestado a carência de um ferramental adequado para suportar esta decisão, consideramos o exercício da disciplina em si como um fator positivo a mais, capaz de facilitar a identificação de requisitos para a criação de um software realmente útil. Com esta filosofia em mente, e, em conjunto com uma série de conhecimentos recém adquiridos com o estudo mais aprofundado sobre a própria metodologia XP, optamos pela criação de uma metodologia própria, aplicando as próprias recomendações de XP que orientam pela adaptação sempre que necessário da mesma. Embora, de duas práticas, a grande maioria não tenha sido aplicada, podemos afirmar que o pequeno conjunto de regras criado neste desenvolvimento a partir de XP foi de grande valia para o processo. Certamente esta foi uma das melhores das alternativas que poderia ter sido eleita como metodologia de desenvolvimento, considerando a elevada complexidade do software, equipe reduzida, requisitos mutantes, prazo curto e uma boa interação com o usuário do software: a própria equipe. Embora à primeira vista esta última colocação possa parecer inadequada,

podemos dizer que nosso grau de experiência com a atividade de trabalho colaborativo distribuído, embora nem sempre nos moldes precisos de PP, é bastante avançado, considerando a experiência em trabalhos acadêmicos executados desta forma durante praticamente todo o período da graduação, ou seja, 4,5 anos.

Podemos afirmar que o conjunto de ferramentas utilizadas como suporte para o desenvolvimento do plugin atendeu adequadamente a demanda. Esta conclusão certamente é colocada ignorando as dificuldades existentes na utilização do aplicativo VNC como ferramenta de visualização e/ou alteração remoto, como a excessida lentidão em vários momentos surgida em horários de pico no uso da rede, por exemplo. Em relação às demais ferramentas, cabe ressaltar a excelente contribuição recebida gratuitamente quando optamos pela hospedagem do projeto no Sourceforge. Todos os serviços são oferecidos de forma online e com alta disponibilidade, criando um ambiente confiável de trabalho. Por fim, todo o trabalho de criação do plugin somente foi possível devido à plataforma de desenvolvimento de plugins do Eclipse, a PDE que, de fato, cumpriu com êxito a tarefa a que se propôs, facilitando e automatizando um processo que de outra forma seria inviável neste tempo.

Como resultados em termos da metodologia de desenvolvimento aplicada em conjunto com as ferramentas, podemos concluir que o cronograma do projeto, de fato, pôde ser executado dentro dos prazos estipulados, garantindo a agilidade planejada para o desenvolvimento. O maior dos aprendizados conquistados, entretanto, neste âmbito, certamente veio em conseqüência do uso do distributed pair programming e a comprovação prática de todos os benefícios pregados pela

metodologia, mesmo com a precariedade da ferramenta de suporte e a reduzida equipe de desenvolvimento. Dentre os mais notáveis, destacamos a excelente compreensão do sistema, a todo tempo, entre os membros da dupla.

Sobre as Dificuldades no Desenvolvimento

No desenvolvimento de um sistema complexo, certamente as maiores dificuldades se concentram justamente no esforço despendido para a identificação de soluções que garantam o seu funcionamento propriamente dito.

Por ser um plugin, várias das atividades necessárias para o cumprimento dos requisitos definidos para o PEP exigiram uma forte atividade de interação com a plataforma Eclipse. Embora uma completa e documentada API de programação seja disponibilizada no site eclipse.org, inclusive para os interessados no desenvolvimento de extensões para a plataforma, na grande maioria das vezes nem a documentação de apoio, nem o código disponibilizado como exemplo eram suficientes e a alternativa restante era a pesquisa e a interação com fóruns de discussão sobre o tema. Por vários ocasiões o desenvolvimento foi atrasado, aguardando respostas e sugestões do time de desenvolvimento do projeto Eclipse o que, felizmente, era satisfeito em todas as situações.

Sobre Pair Programming Distribuído Usando o PEP

Já vimos que há disponível na internet duas ferramentas projetadas especificamente para o suporte da atividade de programação em par distribuída. Vimos também, entretanto, que o nível de maturidade de tais ferramentas, principalmente em termos de robustez e confiabilidade, torna tais aplicativos

possíveis, mas não prováveis ferramentas a serem utilizadas pelos adeptos da metodologia Pair Programming. Assim, além das atividades básicas de sincronismo de código e chat, uma das grandes metas planejadas para o PEP foi a de criar uma alternativa mais confiável como ambiente de desenvolvimento colaborativo e distribuído. A consequência direta desta decisão, a priori, foi a concepção dos novos requisitos de verificação de consistência e resincronismo, embora outras melhorias como o compartilhamento automatizado do projeto e o uso de uma boa API de comunicação tenham sido aplicadas. Como resultado, embora a grandeza deste trabalho não tenha permitido a sua extensão até um estudo empírico comprovando ou atestando sua aceitação, vários dos problemas antes disponíveis foram eliminados, o que cria uma perspectiva favorável neste sentido.

Sobre o Desenvolvimento de Software Livre

O desenvolvimento do plugin PEP em si, por se tratar de um trabalho acadêmico, já pode ser considerado, em termos, como um projeto de software livre. A opção pela hospedagem do projeto no Sourceforge, no entanto, trouxe uma perspectiva de tratamento mais responsável do projeto, considerando que seus frutos são, em muito, passíveis de contribuição com a comunidade de software livre. Assim, a partir do momento em que esta opção foi aceita, criou-se a possibilidade de iniciação de um software com chances de reconhecimento nesta comunidade, na internet e, sobretudo no âmbito dos interessados na programação em par, como os praticantes ativos pra metodologia XP, por exemplo. Na prática, isto significa que este trabalho não deve mais ser somente visto como uma

atividade de aprendizado acadêmica, mas, ao contrário, como um projeto de software livre que, por oferecer uma contribuição real à comunidade na qual se insere, possui excelentes chances de obter continuidade, se não pela dupla que o iniciou, por outros colaboradores que possam surgir e pelos que, mesmo antes do término de uma versão estável do projeto, já se ofereceram como voluntários para colaborar com a causa.

## *4.2 Trabalhos Futuros*

Como comentamos anteriormente, o projeto PEP que apresentamos neste momento não se trata de um trabalho acadêmico na sua forma final, mas sim de um software livre planejado para suprir uma demanda de ferramentas confiáveis aos praticantes da programação em par distribuída. Sendo assim, podemos considerar o trabalho concluído até aqui somente como sendo a primeira de uma provável sequência de etapas que serão executadas até o atingimento desta meta em sua plenitude. No que tange ao planejado para o escopo deste projeto como trabalho acadêmico, os objetivos definidos foram atingidos, restanto, naturalmente, atividades de pouca representatividade como tarefas de implementação. Devido à opção pelo enfrentamento das metas sob esta ótica mais abrangente, porém, preferimos definir como trabalhos futuros, o conjunto de requisitos identificados como necessários e relevantes para a continuidade do PEP, trabalhos que certamente serão parte da próxima lista de melhorias constantes nas páginas do projeto no Sourceforge e na internet. São eles:

- Melhorias no resincronismo: atualmente o resincronismo após um evento de perda de sincronismo (botão “forçar resincronismo), possui um comportamento “pouco inteligente” e, portanto, insuficiente sob olhares mais críticos de desenvolvedores. A intenção é substituir a “simples troca do arquivo” pela utilização de um mecanismo mais sofisticado como o merge do plugin CVS, por exemplo;
- Testes unitários e automatização dos testes: embora as práticas de XP defendam o forte uso de testes unitários e até mesmo de desenvolvimento dirigido por testes (TDD), algumas características inerentes ao processo de desenvolvimento e à própria complexidade do software resultaram na opção pela “não utilização” do TDD. Uma das próximas tarefas a ser executada, portanto, é a criação de alguns testes unitarios de forma a “cobrir” com completude todo o código desenvolvido e estabelecer a prática para implementações futuras;
- Transmissão de voz e imagem: indiscutivelmente a criação da “sensação de presença física” é um requisito fundamental para que as características de boa produtividade obtidas com a DPP possam ser obtidas. Assim, julgamos que o uso de uma ferramenta de comunicação de voz e até mesmo de vídeo, é um requisito cuja implementação interna ao plugin facilitaria o seu uso e, portanto, merece uma atenção futura;
- Internacionalização: embora todo o código tenha sido escrito no idioma inglês, de forma que facilite a intervenção futura por desenvolvedores de outros países que conheçam o idioma, requisitos de internacionalização de

software devem ser inseridos no plugin, de forma que o mesmo possa ser “entendido” também por outras comunidades, ampliando sua utilidade para a comunidade como um todo;

- Suporte a outras linguagens: embora a arquitetura do plugin tenha sido planejada para o desenvolvimento utilizando a linguagem Java, uma vez que faz uso durante suas operações da API JDT (Java Development Tools), seria adequado e relevante criar o suporte a outras linguagens suportadas pelo Eclipse através de outros plugins, como C++ e PHP.

# 5 Referências Bibliográficas

(PALMER et. Al., 1994) PALMER, James D., FIELDS, Ann. “Computer-Supported Cooperative Work”. IEEE May 1994.

(REINHARD et. Al., 1994) REINHARD, Walter, SCHWITZER, Jean, VÖLKSEN, Gerd. “CWCS Tools Concepts and Architectures”. Siemens Corporate Research and Development.

(COCKBURN 2000) COCKBURN, A., WILLIAMS, L. “The Costs and Benefits of Pair Programming”. IEEE.

(WILLIAMS et. Al., 2000) WILLIAMS, Laurie, KESSLER, Robert R., CUNNINGHAM, Ward, JEFFRIES, Ron. “Strengthening the Case for Pair-Programming”. IEEE.

(WILLIAMS; KESSLER, 1999) WILLIAMS, Laurie, KESSLER, Robert R. “Experimenting with Industry´s Pair-Programming Model inf the Computer Science Classrom”. IEEE 1999.

(BECK, 1999) BECK, KENT. “Embracing Change with Extreme Programming”. IEEE 1999.

(KIRCHIER et. Al., 2001)KIRCHER, Michael, JAIN, Prashant, CORSARO, Angelo, LEVINE, David. “Distributed Extreme Programming”. Siemens AG, 2001.

(BAHETI; WILLIAMS; et. Al., 2002) BAHETI, Prashant, WILLIAMS, Laurie, GEHRINGER, Edward. “Distributed Pair Programming: Empirical Studies and Supporting Environments” - Technical Report. Department of Computer Science Univ of Norh Carolina at Chapel Hill.

(GEER, 2005) GEER, D. “Eclipse Becomes The Dominant Java IDE”. IEEE July 2005.

Referências na Internet:

(DEVELOPER, 2005) Revista Eletrônica Developer.com
“Sun Presents Java Studio Enterprise Tool to Further Code-Aware Team Collaboration”. Disponível em<http://www.developer.com/java/ent/article.php/3434771>. Acesso em 30 de Julho de 2006.

(ECLIPSE DOC, 2007) Documentação Oficial do Eclipse
http://www.eclipse.org/documentation/. Acessado em 31 de Janeiro de 2007.

(ECLIPSE API, 2007) API – Application Program Interface da Plataforma Eclipse http://inf.ntb.ch/doc/eclipse/eclipse_doc/reference/api/. Acessado em 31 de Janeiro de 2007.
(ECLIPSE NEWS, 2007) Listas de Discussçao Eclipse. www.eclipse.org/newsgroups/. Acessado em 31 de Janeiro de 2007.

# 6 Bibliografia

Manifest for Agile Software Development. Disponível em:<http://agilemanifesto.org/>. Acessado em 30 de Julho de 2006.

SCRUM. Disponível em <http://www.controlchaos.com>. Acessado em 30 de Julho de 2006.