

3

Extreme Programming (XP)

Este capítulo apresenta o processo ágil de desenvolvimento de software Extreme Programming (XP). A seção 3.1 apresenta uma visão geral do capítulo e a seção 3.2 uma introdução geral sobre processos de software. A seção 3.3 explica alguns conceitos básicos de XP como valores e princípios. A seção 3.4 procura mostrar uma visão geral da estrutura e dinâmica de XP. A seção 3.5 descreve quando não se deve usar XP. A seção 3.6 apresenta uma visão geral sobre a modelagem de processos de software e a seção 3.7 mostra a modelagem de XP usando SPEM. Finalmente, a seção 3.8 apresenta as considerações finais do capítulo.

3.1 Visão geral

Um processo de software consiste de um conjunto de atividades realizadas para a sua construção e manutenção. Ao longo dos últimos anos, a crescente demanda por software de qualidade, aliada à pressão em relação aos prazos de entrega fizeram com que a eficiência de muitas técnicas e processos tradicionalmente utilizados passasse a ser questionada.

Diante deste contexto, os processos ágeis como Extreme Programming [28, 29], passaram a atrair o interesse dos meios acadêmico e industrial. Devido à simplicidade e agilidade adotada na solução de problemas através do uso de eficientes práticas de programação, as equipes que usam XP vêm obtendo sucesso em seus projetos o que vem contribuindo para o aumento da sua popularidade.

Este capítulo procura apresentar Extreme Programming e sua modelagem utilizando a linguagem SPEM [48]. São apresentados os principais conceitos relativos a XP que são seus valores, princípios, práticas, papéis e seu ciclo de vida. O final do capítulo apresenta uma visão geral de SPEM e a sua utilização na modelagem de XP.

3.2 Processos de software

Um processo de software pode ser definido como um conjunto de atividades executadas para desenvolver, dar manutenção e gerenciar sistemas de software [47,51,52,53]. Cada atividade pode ser composta de outras atividades e são realizadas por pessoas, que possuem um determinado papel no processo como: programador, gerente, cliente e outros. Tais pessoas podem utilizar ferramentas e modelos que automatizem e facilitem os seus trabalhos, e à medida que o processo flui, artefatos (código, documentos, modelos, diagramas, etc...) são produzidos, atualizados e consumidos nas atividades realizadas.

Ao longo dos anos, muitos padrões de processos de software foram definidos com o objetivo de ajudar as organizações a construir software de uma forma controlada. Inicialmente, tais padrões tinham o objetivo de ser genéricos no sentido de procurar atender a qualquer tipo de projeto e detalhados no sentido de definir uma grande quantidade de atividades e papéis para produzir muitos artefatos e documentos detalhados. Exemplos de padrões desse tipo são os processos Rational Unified Process (RUP) [20] e OPEN [21] que vêm sendo utilizados com muito sucesso no meio empresarial. Em [54,55] pode ser visto que estes processos são mais adequados a empresas com equipes grandes e projetos complexos onde existe a necessidade de uma extensa e detalhada documentação para redução de riscos.

Recentemente, muitos pesquisadores e renomados consultores empresariais especialistas na área de desenvolvimento de software, passaram a questionar a eficiência dos processos comentados acima – passando a chamá-los de processos pesados. Eles afirmam que os processos pesados “burocratizam” o desenvolvimento de software devido ao excessivo número de atividades e artefatos o que, na opinião deles, acaba desviando o processo daquilo que deveria ser prioritário - a entrega constante de software que “roda” para o cliente.

Esses especialistas fundaram o que veio a se chamar de Aliança Ágil [57] que tem por objetivo encontrar abordagens mais simples e eficientes para o desenvolvimento de software. A idéia desses especialistas foi publicada através de um

documento chamado “Manifesto para o Desenvolvimento Ágil de Software” que valoriza os seguintes itens:

- **Indivíduos e interações** acima de processos e ferramentas.
- **Software funcionando** acima de documentação abrangente.
- **Colaboração com o cliente** acima de negociação de contratos.
- **Responder a mudanças** acima de seguir um plano.

Isso não quer dizer que esse manifesto não considere os itens à direita importantes, mas sim, que os itens à esquerda são considerados de maior valor. Portanto, os processos ágeis, como Extreme Programming (XP) [28,29], DSDM [23], Crystal [24] e outros, focam-se em entrega constante de funcionalidade e interação constante entre os membros da equipe e clientes [28,29,54,55]. Estes processos vêm se mostrando eficientes em projetos onde os requisitos são constantemente modificados, os ciclos de desenvolvimento são curtos e as equipes que implementam o projeto são pequenas [54,55,56].

Ambos os grupos de processos (ágeis e pesados) possuem vantagens e desvantagens. Cada grupo pode ser mais adequado para determinados tipos de projeto, dependendo de fatores como: natureza e complexidade da aplicação a ser construída; tamanho e distribuição da equipe; e restrições de prazo e custo impostas pelo cliente.

A descrição detalhada de XP será vista ao longo de todo este capítulo que também fará uma análise dos principais defeitos e virtudes de XP em relação ao desenvolvimento de aplicações Web, bem como apresentará a justificativa pela sua escolha. Na próxima seção serão apresentados alguns conceitos básicos de XP.

3.3 Conceitos básicos de XP

O processo ágil eXtreme Programming (XP) resultou da experiência no projeto C3 Payroll na empresa Chrysler. Este projeto consistia da implementação de um sistema

de folha de pagamento que já havia fracassado anteriormente utilizando outras metodologias. Após o sucesso nesse projeto, XP começou a despontar no meio acadêmico e empresarial e se tornou alvo de inúmeras pesquisas e discussões, além de ser adotado em diversas empresas de software do mundo inteiro e apoiado por grandes “gurus” da orientação a objeto como Kent Beck, Ron Jeffries, Martin Fowler e Grady Booch.

O sucesso e popularidade adquiridos por XP se devem principalmente aos relatos de bons resultados obtidos em projetos, a motivação dos profissionais envolvidos com XP e também devido a sua natureza simples e objetiva por se basear em práticas que já provaram sua eficiência no cenário do desenvolvimento de software. Essas práticas têm como objetivo entregar funcionalidades de forma rápida e eficiente ao cliente. Além disso, XP foi criado considerando que mudanças são inevitáveis e que devem ser incorporadas constantemente. XP se baseia em alguns valores e princípios que serão mostrados na próxima seção.

3.3.1 Valores e Princípios de XP

Extreme Programming (XP) é um processo de desenvolvimento de software que adota os valores de comunicação, simplicidade, feedback e coragem [28,29]. Estes quatro valores servem como critérios que norteiam as pessoas envolvidas no desenvolvimento de software e serão detalhados a seguir.

Comunicação: XP foca em construir um entendimento pessoa-a-pessoa do problema, com o uso mínimo de documentação formal e com o uso máximo de interação “cara-a-cara” entre as pessoas envolvidas no projeto. As práticas de XP como programação em pares, testes e comunicação com o cliente têm o objetivo de estimular a comunicação entre gerentes, programadores e clientes.

Simplicidade: XP sugere que cada membro da equipe adote a solução mais fácil que possa funcionar. O objetivo é fazer aquilo que é mais simples hoje e criar um ambiente em que o custo de mudanças no futuro seja baixo. O objetivo dessa abordagem adotada por XP é evitar a construção antecipada de funcionalidades, como é

feita em muitas metodologias tradicionais, que acabam muitas vezes nem sendo usadas.

Feedback: Os programadores obtêm feedback sobre a lógica dos programas escrevendo e executando casos de teste. Os clientes obtêm feedback através dos testes funcionais criados para todas as histórias (casos de uso simplificados). O feedback é importante, pois possibilita que as pessoas aprendam cada vez mais sobre o sistema e assim corrijam os erros e melhorem o sistema.

Coragem: Ela é necessária para que realmente se aplique XP como deve ser aplicado. Exemplos de atitude que exigem coragem são: alterar código já escrito e que está funcionando; jogar código fora e reescrever tudo de novo; e permitir código compartilhado por todos. Estes exemplos de atitudes podem ser necessários para trazer melhorias ao projeto e não devem ser evitadas simplesmente devido ao medo de tentá-las.

Além dos valores apresentados anteriormente XP define um conjunto de princípios que devem ser seguidos por equipes que forem usar XP em projetos. Os princípios servirão para ajudar na escolha de alternativas de solução de problemas durante o curso do projeto. Deve-se preferir uma alternativa que atenda aos princípios de uma forma mais completa do que outra que seja incompleta, ou seja, que esteja mais longe da filosofia de XP. Os princípios fundamentais são: feedback rápido, assumir simplicidade, mudança incremental, abraçando mudanças e trabalho de qualidade.

Feedback rápido: A idéia de XP é que os participantes de um projeto como clientes, programadores e gerentes devem estar sempre se comunicando para facilitar o aprendizado coletivo sobre o projeto que está sendo desenvolvido e de alertar rapidamente sobre dúvidas, riscos e problemas para facilitar eventuais ações de contingência.

Assumir simplicidade: Todo problema deve ser tratado para ser resolvido da forma mais simples possível. XP afirma que se deve fazer um bom trabalho (testes, refactoring, comunicação) para resolver hoje os problemas de hoje e confiar na sua habilidade de adicionar complexidade no futuro quando for necessário.

Mudança incremental: Quando muitas mudanças são realizadas todas de uma vez, não se obtém um bom resultado. Em vez disso, esse princípio de XP diz que as mudanças devem ser incrementais e feitas aos poucos. Os programadores têm a liberdade para estar sempre fazendo alterações de melhoria no código e as mudanças de requisitos são incorporadas de forma incremental.

Abraçando mudanças (Embracing Change): XP procura facilitar o ato de incluir alterações através do uso de vários princípios e práticas. A idéia de XP é a de que mudanças devem ser sempre bem vindas independentemente do estágio de evolução do projeto. Isso é muito benéfico em projetos cujos requisitos são bastante voláteis devido aos clientes não saberem o que querem.

Trabalho de qualidade: Em XP, qualidade não é um critério opcional, mas sim obrigatório. Embora a definição de qualidade possa variar de pessoa para pessoa, XP trata qualidade no sentido de se ter um sistema que atenda aos requisitos do cliente, que rode 100% dos casos de teste e que agregue o maior valor possível para o negócio do cliente.

3.4 Visão geral de XP

Esta seção baseia-se nas fontes de informação apresentadas em [28,29], e procura mostrar a estrutura principal e o funcionamento do processo Extreme Programming. A subseção 3.4.1 mostra as práticas que constituem a estrutura de XP. A subseção 3.4.2 mostra o ciclo de vida de um projeto XP. Finalmente, a subseção 3.4.3 descreve a gerência de projetos e os principais papéis envolvidos em um projeto XP.

3.4.1 Práticas de XP

Extreme Programming possui doze práticas que consistem no núcleo principal do processo e que foram criadas com base nos ideais pregados pelos valores e princípios apresentados anteriormente na seção 3.3.1. Segundo um dos criadores de XP, Kent

Beck, estas práticas não são novidades, mas sim práticas que já vêm sendo utilizadas a muitos anos, com eficiência, em projetos de software. Muitas das práticas de XP não são unanimidades dentro da comunidade de desenvolvimento de software, como por exemplo, programação em pares. No entanto, o valor e benefícios de tais práticas devem ser avaliados em conjunto e não individualmente, pois elas foram criadas para serem usadas coletivamente, de forma a reduzir as fraquezas umas das outras. As doze práticas de XP são comentadas abaixo.

O jogo do planejamento: O planejamento de um release e das iterações são feitos com base nas histórias (casos de uso simplificados) e conta com a colaboração de toda a equipe de desenvolvimento, inclusive o cliente, divididos em dois papéis:

- **Negócio:** Participam as pessoas que mais entendem sobre o negócio e que possam estabelecer prioridades para as funcionalidades a serem entregues.
- **Técnico:** Participam as pessoas que irão implementar as funcionalidades descritas. Os técnicos estimam qual o esforço e riscos envolvidos para implementar as funcionalidades e comunicam ao pessoal de negócios.

No final de um release é feita uma determinação rápida do escopo do próximo, através da combinação de estimativas e prioridades do negócio. Um release consiste de várias iterações e, em cada iteração, várias histórias (use case simplificados) são implementadas. Os programadores estimam cada história e dizem quantas eles podem implementar no final do release. Baseado nesses dados, os clientes escolhem as principais histórias (que agregam maior valor para o negócio) que serão implementadas. As estimativas podem ser refeitas durante as iterações à medida que os programadores aprenderem mais sobre o sistema.

Releases pequenos: Em [28] Beck diz: “cada release deve ser tão pequeno quanto possível, contendo os requisitos mais importantes para o negócio”. Isso possibilita ter releases frequentes o que resulta em maior feedback para clientes e programadores, facilitando o aprendizado e a correção dos defeitos do sistema. Beck sugere que os releases sejam de curta duração, normalmente de dois a três meses.

Metáfora: A intenção da metáfora é oferecer uma visão geral do sistema, em um formato simples, que possa ser compartilhada por clientes e programadores. A idéia da metáfora é que seja feita uma analogia entre o sistema que está sendo desenvolvido e um sistema, não necessariamente de software, que todos entendam, com o objetivo de obter um “vocabulário comum” para a posterior criação de nomes de classes, subsistemas, métodos, etc. Um exemplo de metáfora pode ser visto em [29], onde o sistema C3 payroll de folha de pagamento é descrito como uma linha de montagem onde várias partes referentes às horas trabalhadas vão sendo montadas até construir o cheque de pagamento do funcionário. A metáfora conforme dito em [28] também pode ser vista como uma forma simples de arquitetura do sistema, numa linguagem compreensível tanto por clientes como programadores.

Projeto simples: Pode-se explicar esta prática em duas partes: A primeira diz que devem ser projetadas as funcionalidades que já foram definidas e não as que poderão ser definidas futuramente. A segunda diz que deve ser feito o melhor projeto que possa entregar tais funcionalidades. Esta prática tem o intuito de enfatizar que o projeto simples deve se concentrar em soluções simples e bem estruturadas para os problemas de hoje e que não se deve perder tempo investindo em soluções genéricas que procurem atender a funcionalidades futuras, pois como os requisitos mudam constantemente tais soluções genéricas podem não ser mais a realidade do futuro. Algumas características de projeto simples citadas em [28,29] são:

- Todos os testes executam com sucesso.
- O projeto expressa a idéia que o programador teve.
- O projeto não possui lógica duplicada.
- O projeto contém o menor número possível de classes e métodos.

Testes constantes: Os testes em XP são feitos antes da programação. Existem dois tipos de teste: teste de unidade e teste funcional. Os testes de unidade são feitos para verificar tudo que possa dar errado. Não é preciso escrever testes de unidade para todos os métodos, conforme mencionado em [28-30]. Os testes unitários são automatizados, e toda vez que o programador escrever código, ele irá verificar se o

sistema passa em todos os testes. Os testes funcionais são usados para verificação, junto ao cliente, do sistema como um todo. Os testes servem como um mecanismo para assegurar que o sistema está sempre rodando livre de erros, e também servem para dar feedback aos programadores e clientes sobre as falhas encontradas.

Refatoramento: São constantes melhorias no projeto do software para aumentar sua capacidade de se adaptar a mudanças. O refatoramento consiste em aplicar uma série de passos, como mostrado em [31], para melhorar o projeto do código existente, tornando-o mais simples e melhor estruturado, sem alterar sua funcionalidade.

Programação em pares: Todo o código produzido em XP é escrito por um par de programadores, que possuem papéis distintos, sentados lado-a-lado e olhando para o computador. Um parceiro será responsável pela codificação e pensará nos algoritmos e na lógica de programação. O outro parceiro observa o código produzido e tenta pensar mais estrategicamente em como melhorá-lo e torná-lo mais simples, além de apontar possíveis erros e pontos de falha. Além disso, as duplas são constantemente trocadas e os papéis também com o objetivo de que todos os membros da equipe possam ter conhecimento sobre todas as partes do sistema.

Propriedade coletiva do código: A programação em pares encoraja duas pessoas a trabalharem juntas procurando atingir o melhor resultado possível. A propriedade coletiva encoraja a equipe inteira a trabalhar mais unida em busca de qualidade no código fazendo melhorias e refatoramentos em qualquer parte do código a qualquer tempo. A princípio pode-se pensar que esta prática possa gerar problemas, mas como todos devem respeitar um padrão de codificação e devem realizar todos os testes para verificação de erros esta atividade é feita de uma forma controlada e pode ser facilitada com o uso de uma ferramenta de controle de versão.

Integração contínua: O código das funcionalidades implementadas pode ser integrado várias vezes ao dia. Um modo simples de fazer isso é ter uma máquina dedicada para integração. Quando a máquina estiver livre, um par com código a integrar ocupa a máquina, carrega a versão corrente do sistema, carrega as alterações que fizeram (resolvendo as colisões), e rodam os testes até que eles passem (100% corretos).

O importante é que na integração as funcionalidades só podem ser integradas se não houver erros, caso contrário os erros devem ser corrigidos.

Semana de quarenta horas: Essa não é uma regra que obriga as equipes em projetos XP a trabalharem somente 40 horas por semana. No entanto, Beck diz que não se deve trabalhar duas semanas seguidas além desse tempo, pois o cansaço e a insatisfação de trabalhar horas extras pode resultar numa queda de qualidade do código.

Cliente no local: Deve ser incluído na equipe uma pessoa da parte do cliente, que irá usar o sistema, para trabalhar junto com os outros e responder as perguntas e dúvidas. Mesmo que não seja possível obter alguém da parte do cliente deve-se ter alguém que tenha conhecimento suficiente do negócio para exercer este papel. O cliente tem um papel importante dentro de um projeto XP já que ele participa do planejamento do projeto escrevendo as histórias e priorizando-as.

Padrões de codificação: Como XP prega a propriedade coletiva de código, onde todos podem alterar e fazer refatoramento de qualquer parte do código a qualquer momento, então é mais do que necessário que se tenha padrões de codificação. O objetivo é que todos programem da mesma forma, facilitando o entendimento do código e as alterações.

3.4.2 Ciclo de vida de um projeto XP

Um projeto XP atravessa algumas fases durante o seu ciclo de vida. Essas fases são compostas de várias tarefas que são executadas. Abaixo será dada uma explicação das principais fases de um projeto XP de modo a se ter uma idéia de como o projeto flui ao longo do tempo.

Um projeto XP passa pelas seguintes fases: exploração, planejamento inicial, iterações do release, produção, manutenção e morte.

A fase de exploração é anterior à construção do sistema. Nela, investigações de possíveis soluções são feitas e verifica-se a viabilidade de tais soluções. Os programadores elaboram possíveis arquiteturas e tentam visualizar como o sistema funcionará considerando o ambiente tecnológico (hardware, rede, software,

performance, tráfego) onde o sistema irá rodar. Com isso, os programadores e os clientes vão ganhando confiança, e quando eles possuírem histórias suficientes, já poderão começar a construir o primeiro release do sistema. Em [28] sugere-se que seja gasto no máximo duas semanas nessa fase.

A fase de planejamento inicial deve ser usada para que os clientes concordem em uma data para lançamento do primeiro release. O planejamento funciona da seguinte forma: Os programadores, juntamente com o cliente, definem as histórias (use case simplificados) a serem implementadas e as descrevem em cartões. Os programadores assinalam uma certa dificuldade para cada história e, baseados na sua velocidade de implementação, dizem quantas histórias podem implementar em uma iteração. Depois, os clientes escolhem as histórias de maior valor para serem implementadas na iteração – isso é chamado planejamento de iteração. O processo então se repete até terminar as iterações do release. O tempo para cada iteração deve ser de uma a três semanas e para cada release de dois a quatro meses.

Na fase das iterações do release são escritos os casos de teste funcionais e de unidade. Os programadores vão seguindo mais ou menos o seguinte fluxo de atividades na seguinte ordem (Em cada iteração): escrita dos casos de testes; projeto e refatoramento; codificação; realização dos testes; e integração. À medida que esse fluxo vai sendo seguido, o sistema vai sendo construído segundo os princípios, valores e práticas apresentados nas seções anteriores. Depois de terminado o primeiro release, já se terá uma idéia melhor das tecnologias e do domínio do problema de modo que as iterações poderão ser mais curtas nos releases subseqüentes e já se podem fazer estimativas mais confiáveis com o que se aprendeu das iterações passadas.

Depois do final do primeiro release, considera-se o início da **fase de produção** onde cada release subseqüente do sistema, depois de construído, é colocado para rodar em um ambiente que simula o ambiente de produção para ver seu comportamento em termos de performance. Pode-se fazer testes de aceitação adicionais para simular o funcionamento real do sistema no ambiente alvo.

A fase de manutenção pode ser considerada como uma característica inerente a um projeto XP. Em XP você está simultaneamente produzindo novas funcionalidades,

mantendo o sistema existente rodando, incorporando novas pessoas na equipe e melhorando o código. Mecanismos como: refatoramento, introdução de novas tecnologias, e introdução de novas idéias de arquitetura podem ser utilizados em um projeto XP. É importante ressaltar que a manutenção dada em um sistema que já está em produção deve ser feita com muita cautela, pois uma alteração errada pode paralisar o funcionamento do sistema resultando em prejuízos para o cliente.

A fase de morte corresponde ao término de um projeto XP. Existem duas razões para se chegar ao final de um projeto, uma boa e a outra ruim. A boa razão é quando o cliente já está satisfeito com o sistema existente e não enxerga nenhuma funcionalidade que possa vir a ser implementada no futuro. A má razão para a morte em XP seria a do projeto ter se tornado economicamente inviável, devido a dificuldades de adicionar funcionalidades a um custo baixo e devido a uma alta taxa de erros.

3.4.3 Gerência de projetos e papéis envolvidos em XP

A estratégia de gerência adotada em XP é mais voltada para a tomada de decisões descentralizada do que para o controle centralizado. O papel do gerente é fazer fluir o jogo do planejamento, coletar métricas, fazer com que as métricas sejam vistas por aqueles cujo trabalho esteja sendo medido, e ocasionalmente intervir em situações que não podem ser resolvidas de forma distribuída.

A ferramenta básica de gerência em XP é a métrica. A medida básica que é usada durante o jogo do planejamento é a razão entre o tempo estimado para desenvolver e o tempo realmente gasto. Se esta razão aumentar (menos tempo real para um tempo estimado dado) pode significar que a equipe está trabalhando corretamente. Por outro lado, isso também pode significar que a equipe está gastando pouco tempo na codificação e em refatoramento o que poderá prejudicar a qualidade do projeto no longo prazo. A forma usada para mostrar as métricas deve ser um gráfico atualizado constantemente (pelo menos uma vez por semana).

A gerência em XP é dividida através de dois papéis: o treinador (coach) e o rastreador (tracker). Esses papéis podem ou não ser executados pela mesma pessoa. O

treinador se preocupa principalmente com a execução técnica e evolução do processo. O treinador ideal deve ser um bom comunicador, ter um bom conhecimento técnico, não entrar em pânico e ser confiante. O papel do treinador não é de tomar decisões técnicas, mas de fazer com que todos tomem boas decisões e de facilitar o processo de desenvolvimento.

O rastreamento é outro componente da gerência em XP. O objetivo do **rastreador** (tracker) é coletar métricas sobre o que está sendo desenvolvido e confrontar com as métricas estimadas verificando possíveis divergências. O rastreador deve tomar cuidado para não perturbar muito os programadores pedindo por métricas. Uma forma recomendada é coletar os dados duas vezes por semana.

Além dos papéis gerenciais apresentados anteriormente, uma equipe que utiliza XP para desenvolver software é composta de outros papéis. Estes são: programador, cliente, testador e consultor.

O programador ocupa o principal papel em XP. Ele analisa, projeta, testa, codifica, e integra o sistema. Além disso, o programador estima a dificuldade das histórias e faz alterações nessas estimativas, caso necessário. Em XP o foco está na programação e o importante é entregar funcionalidades implementadas para o cliente. O programador está sempre escrevendo testes de unidade, codificando e fazendo refatoramento com o objetivo de produzir código de alta qualidade rapidamente.

O cliente escolhe o que vai agregar valor ao seu negócio, escolhe o que deve ser feito primeiro e o que deve ser adiado. Além disso, o cliente define com a ajuda dos testadores, os testes funcionais que irão mostrar se o sistema realmente faz o que deve ser feito.

O testador irá ajudar o cliente na definição e escrita dos testes funcionais. Ele não precisa ser uma pessoa com apenas essa função, pode desempenhar também o papel de programador.

O consultor é necessário apenas em algumas situações onde se precisa de alguém com um elevado nível de conhecimento, por exemplo, um especialista em uma determinada tecnologia sobre determinado assunto que não está sendo bem

compreendido pelas pessoas do projeto. O objetivo da equipe não é obter a solução do consultor para usá-la, mas sim entender o problema e elaborar a sua própria solução.

3.5 Quando não se deve usar XP

Os limites exatos de XP ainda não são conhecidos, mas existem alguns fatores que são fortes indicadores para não usar XP como [28,29]: equipes grandes, clientes desconfiados e tecnologia que não dá suporte facilitado para mudanças. Uma breve lista de possíveis barreiras para o sucesso de um projeto XP são mostradas a seguir.

- **Cultura:** A organização pode estar inserida dentro de uma cultura tradicional de desenvolvimento de software, produzindo muita documentação, gastando muito tempo com análise e projeto antecipado, entre outras coisas. Fazer com que a equipe passe a adotar as práticas de XP e esqueça as antigas pode ser muito difícil.
- **Tamanho da equipe:** Um projeto XP deve possuir uma equipe pequena – geralmente até 12 programadores. É difícil imaginar como ficariam alguns conceitos de XP como comunicação, programação em par e outras em uma equipe grande (Ex. 100 pessoas).
- **Tecnologia:** Não se deve usar XP quando uma tecnologia é complicada para escrever casos de teste, quando retorna feedback em um tempo longo ou quando não incorpora facilmente as mudanças.
- **Espaço físico:** A organização do espaço físico onde a equipe de XP trabalha deve facilitar a comunicação e deixar todos próximos uns dos outros.
- **Cliente:** XP exige que o cliente participe da equipe do projeto e trabalhe no mesmo local dos demais, estando a disposição, de preferência, o tempo todo para esclarecer dúvidas. Caso não possa ser alguém da organização do cliente alguém deve ser alguém que entenda do negócio

do cliente e que seja capacitado para exercer este papel. O cliente também não precisa estar disponível todo o tempo embora seja preferível.

Esta seção finaliza a descrição dos pontos mais importantes de XP, iniciada na seção 3.3. A próxima seção procura apresentar uma visão geral sobre modelagem de processos de software e sobre a sua necessidade dentro do contexto deste trabalho. Além disso, serão apresentados alguns exemplos da modelagem de XP utilizando SPEM [48].

3.6 Modelagem de Processos de Software

Um processo de software pode ser definido como um conjunto parcialmente ordenado de atividades realizadas para construir, gerenciar e dar manutenção em sistemas de software [51,52,53]. Tais atividades são realizadas por pessoas que possuem um determinado papel no processo. Durante a execução de atividades, produtos do processo, chamados artefatos, podem ser atualizados, consumidos ou produzidos. Exemplos comuns de artefatos são código fonte, modelos de análise e projeto, diagramas, documentos, manuais e outros.

Pode-se então dizer que um processo de software se compõe de diversos elementos que se relacionam segundo algum critério. Alguns elementos comuns a processos de software segundo [51,52,53] são:

- **Agente ou ator:** É uma entidade que participa da execução do processo. Pode-se dividir os atores em dois grupos: atores humanos e atores de sistema. Os atores humanos são as pessoas que participam do processo enquanto que os atores de sistema são os componentes de hardware e software que participam da execução de uma determinada parte do processo.

- **Papel:** Descreve as responsabilidades, deveres e habilidades necessárias para realizar uma determinada atividade do processo. Um papel pode ser composto por um grupo de agentes e um mesmo agente pode atuar em papéis distintos.
- **Atividade:** Representa o trabalho realizado por um ou mais agentes em um determinado ponto de execução do processo. Os agentes agrupados em papéis realizam uma determinada atividade como responsáveis ou assistentes e podem utilizar técnicas, modelos e ferramentas para facilitar o seu trabalho. Além disso, artefatos podem ser exigidos como entrada para a realização de uma determinada atividade ou podem ser atualizados e produzidos como saída no término da atividade.
- **Artefato ou produto:** É o subproduto de um processo. Ele é criado durante a execução do processo e pode ser modificado ao longo do tempo apresentando diferentes versões. Existem artefatos que são criados para facilitar o processo de construção de software e a manutenção dos sistemas (ex: modelos de dados, diagrama de classes e outros). Nem todo artefato é entregue ao cliente ficando a critério da empresa fornecedora de software quais artefatos devem ou não ser descartados ao longo do tempo.

A descrição dos elementos do processo pode ser representada através de um diagrama de classes da UML como na Figura 3.1 a seguir.

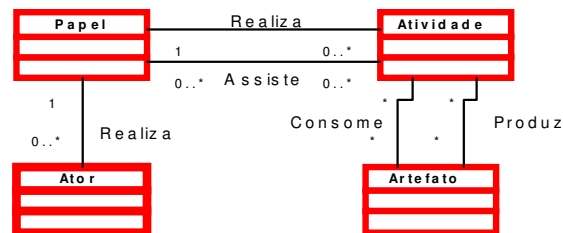


Figura 3.1 - Elementos de um processo de software

A modelagem de processos de software procura definir modelos e seus inter-relacionamentos para representar os elementos que fazem parte do processo, como, por

exemplo, na Figura 3.1. Tais modelos podem ser construídos seguindo uma determinada linguagem de modelagem e procuram retratar uma determinada visão do processo de software. Existem diversas linguagens, meta-modelos e ferramentas que possibilitam modelar diversos aspectos de processos de software segundo uma visão particular. Nas próximas seções será discutido a importância da modelagem de processos de software (seção 3.6.1) bem como algumas abordagens existentes para modelagem (seção 3.6.2).

3.6.1 Importância da modelagem de processos de software

Diversos trabalhos [51,52] têm constatado os benefícios e a importância da modelagem de processos de software. Dentre os principais benefícios advindos da modelagem pode-se citar:

- **Facilitar o entendimento e comunicação:** A modelagem do processo permite a estruturação dos seus elementos mostrando seus inter-relacionamentos e tornando mais fácil a visualização e entendimento do processo por parte das pessoas envolvidas.
- **Facilitar a gerência e controle do processo:** A modelagem pode simplificar o trabalho da gerência, que poderá verificar de uma forma mais fácil – através de uma determinada visão do processo - o andamento do processo dentro de uma seqüência de atividades a ser seguida.
- **Dar suporte para a melhoria do processo:** Uma vez que se tenha um modelo para o processo, fica mais simples evoluir e fazer modificações para atingir determinados objetivos específicos e adaptar o processo a novos conceitos e paradigmas. Ao se acostumar com o processo na forma de um modelo, as pessoas tendem a entender cada vez mais o processo e sugerir adaptações e melhorias ao longo do tempo.
- **Dar suporte para a automação de partes do processo:** Uma vez que se tenha definido um modelo para representar processos, ferramentas podem ser criadas para automatizar certas partes do processo,

eliminando algumas atividades feitas de forma manual e conseqüentemente agilizando o processo de construção de software.

Levando esses benefícios em consideração é que este trabalho procura elaborar uma modelagem para o processo de software XP. Em [65] menciona-se que processos de software ágeis, incluindo XP, carecem de uma descrição mais explícita dos seus elementos e não especificam como podem ser adaptados. Isso torna difícil a adaptação de processos ágeis para determinados tipos de propósitos como, por exemplo: adaptar XP para o desenvolvimento de software para a Web, adaptar XP para o desenvolvimento de software de tempo real, etc.

A necessidade de se adaptar processos surge do fato de que os processos de desenvolvimento de software geralmente possuem um propósito geral e são descritos de uma forma que não abrange de forma suficiente todos os tipos de domínios de problema. Então, para resolver um determinado problema específico, em alguns casos, é necessária uma adaptação no processo de software. A tarefa de adaptar o processo pode ser facilitada pelo uso linguagens de modelagem para representar o processo e ferramentas para automatizar a modelagem.

Ao longo deste capítulo serão mostradas a modelagem e as adaptações feitas em XP. A modelagem é feita usando a meta-linguagem de modelagem de processos SPEM [48] e será descrita na seção 3.7. Esta modelagem servirá para a elaboração do processo XWebProcess a ser mostrada no capítulo 4.

3.6.2 Abordagens para modelagem de processos de software

Processos de software podem ser modelados em diferentes níveis de abstração como, por exemplo: modelos genéricos, modelos adaptados e modelos instanciados. Além disso, eles também podem ser modelados com diferentes objetivos e segundo visões distintas. Em [66] são mostradas algumas das principais perspectivas de modelagem de processos:

- **Funcional:** Representa quais elementos do processo estão sendo implementados e quais fluxos de informação são importantes para tais elementos.
- **Comportamental:** Representa quando e sobre quais condições os elementos do processo são implementados.
- **Informativo ou Estrutural:** Representa a estrutura e relacionamentos entre os elementos do processo.

Os modelos usados para representar processos procuram levar em consideração as perspectivas vistas acima e são criados com o objetivo de se oferecer linguagens, abstrações e formalismos para facilitar a representação de um processo de software capturando sua estrutura e comportamento.

Ao longo dos últimos anos, surgiram diversas linguagens com o propósito de modelar processos de software. Em [67] é mostrada uma linguagem chamada DYNAMITE que se baseia em conceitos de UML e de redes de tarefas (task nets) para a modelagem de processos de software. Além da linguagem, ferramentas são fornecidas para criar os modelos e acompanhar o processo à medida que um projeto vai sendo implementado. Em [68] é proposta uma linguagem orientada a objetos para modelagem de processos de software que também possui o suporte de ferramentas.

O surgimento de várias linguagens para processos de software acaba acarretando um problema semelhante ao que aconteceu nos anos 80 em relação à modelagem orientada a objetos, quando muitas empresas usavam linguagens distintas para modelagem o que dificultava o trabalho junto aos clientes e a troca de informações entre as empresas. Esse problema foi minimizado depois do surgimento de UML que logo veio a se tornar um padrão utilizado em escala mundial.

No campo da modelagem de processos de software um grande avanço foi dado nos últimos três anos quando a OMG [69] decidiu submeter uma proposta de requisição (RFP) por uma linguagem de modelagem para processos de software. Diversas empresas importantes na área de software como Rational software, IBM, Unisys, Alcatel e outras resolveram se unir para elaborar uma linguagem que pudesse atender

aos requisitos propostos pela OMG. Em Novembro de 2002 a meta-linguagem Software Process Engineering Metamodel (SPEM) [48] foi oficializada como um padrão da OMG e encontra-se atualmente na sua versão 1.0.

A meta-linguagem SPEM foi escolhida como linguagem de modelagem a ser usada neste trabalho, pois acreditamos que ela possa se tornar um padrão muito utilizado, como é a UML, devido aos seguintes fatores:

- É um padrão oficial da OMG cuja reputação em estabelecer padrões é indiscutível;
- Possui o apoio de grandes empresas na área de software que estão dando suporte ao seu uso, através da construção de ferramentas e outras formas de apoio;
- Baseia-se em UML que é um padrão já conhecido e consolidado no meio acadêmico e empresarial.

Uma completa descrição da linguagem SPEM é mostrada no Apêndice 1, onde é apresentada uma visão geral sobre a linguagem e seus principais conceitos. É importante ressaltar que as outras linguagens possuem grande importância para a área de modelagem de processos, porém optamos por SPEM devido aos fatores mencionados acima. A próxima seção mostra a modelagem de XP feita usando SPEM.

3.7 Modelagem de XP usando SPEM

Esta seção contém uma breve descrição sobre SPEM na seção 3.7.1, e uma parte da modelagem de XP usando SPEM mostrada nas seções 3.7.2 e 3.7.3.

3.7.1 Visão geral de SPEM

O Software Process Engineering Metamodel (SPEM) [48] é um meta-modelo que pode ser usado para descrever um processo concreto de desenvolvimento de

software ou uma família de processos relacionados. SPEM adota uma abordagem orientada a objetos para modelar processos e usa UML como notação. A Figura 3.2 descreve a arquitetura de quatro níveis de modelagem definida pela OMG e respeitada por SPEM.

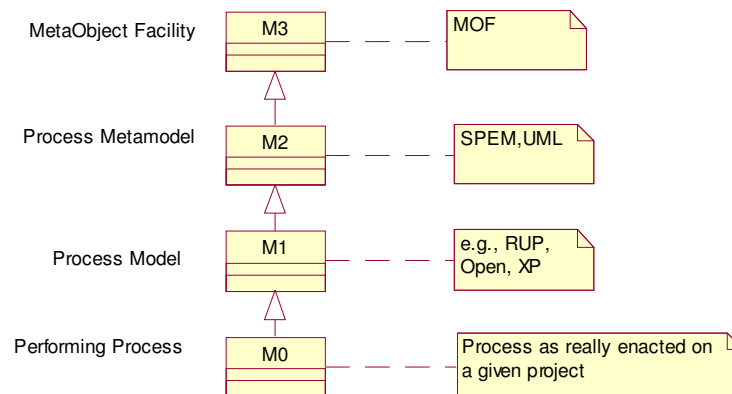


Figura 3.2 - Níveis de modelagem definidos em [48]

O processo real de produção, conforme instanciado para um projeto específico, está no nível M0. No nível M1 encontra-se a definição do processo que foi instanciado em M0 como, por exemplo, RUP, OPEN ou XP. O foco da definição de SPEM está em criar um meta-modelo que se encontra no nível M2 que possui estereótipos para a modelagem dos processos que estão no nível M1. A especificação de SPEM está estruturada como um perfil UML (UML profile)¹⁰ e também através de um meta-modelo baseado no MOF¹¹ que se encontra no nível M3. Essa abordagem facilita a utilização e construção de ferramentas para UML e ferramentas baseadas em MOF.

O objetivo do padrão SPEM é abranger uma vasta gama de processos de desenvolvimento de software já existentes ao invés de excluí-los devido ao excesso de elementos e restrições que poderiam existir na sua definição. SPEM permite que seus

¹⁰ Um UML-profile é um conjunto de uma ou mais extensões da semântica de UML com a intenção de customizá-la para um domínio ou propósito particular, como, por exemplo, para modelagem de processos no caso de SPEM.

¹¹ O Meta-Object Facility (MOF) é a tecnologia adotada pela OMG para definir metadados e representá-los como objetos CORBA. SPEM usa um subconjunto da UML para representar seus elementos como um meta-modelo MOF.

usuários (modeladores de processo) usem a UML, e define estereótipos que podem ser usados nos modelos produzidos.

O meta-modelo SPEM é construído pela extensão de um subconjunto do meta-modelo da UML 1.4. Esse subconjunto de UML é chamado em SPEM de “SPEM_Foundation”. Além disso, o modelo SPEM possui o pacote “SPEM Extensions” que adiciona as construções e semânticas requeridas para a engenharia de processos de software. A Figura 3.3 abaixo mostra esses dois pacotes.

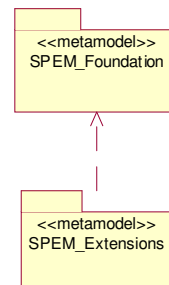


Figura 3.3 - Estrutura de pacotes de SPEM [48]

Um maior detalhamento do conteúdo de cada pacote pode ser encontrado em [48] e no Apêndice 1. A descrição do modelo SPEM apresentada até agora é suficiente para se ter um entendimento de como a linguagem foi construída e de qual é a sua utilidade. O uso real de SPEM para a modelagem de um processo de software será apresentado nas duas próximas seções através da modelagem de XP.

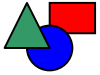




3.7.2 Modelagem dinâmica de XP usando SPEM

Na seção anterior foi apresentada uma visão geral de SPEM destacando-se como a linguagem é definida e organizada. Nesta seção, procura-se mostrar como SPEM pode ser utilizada na prática, ou seja, do ponto de vista de seus usuários. A linguagem SPEM será utilizada para modelar o processo XP, apresentado anteriormente, sob as perspectivas dinâmica (comportamental) e estrutural.


De acordo com [48] alguns diagramas básicos de UML podem ser usados para apresentar perspectivas diferentes de um modelo de processo de software. Em SPEM

podem ser utilizados alguns desses diagramas, dentre eles: diagrama de classes, de pacotes, de atividade, de caso de uso, de seqüência e de transição de estados. A linguagem SPEM oferece algumas representações e estereótipos para modelar seus principais elementos em diagramas UML. Um resumo dos principais elementos de SPEM, seus conceitos e suas representações gráficas é mostrado na Tabela 3.1.

Tabela 3.1 - Descrição de alguns elementos de SPEM¹²

ESTEREÓTIPO	COMENTÁRIO	NOTAÇÃO
WorkProduct (Artefato)	É uma descrição de algo que contém informação ou é uma entidade física produzida ou usada por atividades do processo. Ex: modelos, planos, documentos, etc.	
WorkDefinition (Conjunto de Trabalho)	É um elemento do modelo que descreve a execução, as operações realizadas e as transformações feitas nos "WorkProducts". Representa um conjunto de atividades, como, por exemplo: especificar requisitos, fazer projeto do sistema, etc.	
Guidance (Guia)	É um elemento do modelo que se associa a outros elementos e pode conter descrições adicionais, tais como técnicas, "guidelines", "templates", etc...	
Activity (Atividade)	É uma "WorkDefinition" que descreve o que um "ProcessRole" (papel) realiza.	
ProcessRole (Papel)	Descreve os papéis, responsabilidades e competências que um determinado indivíduo tem dentro do processo.	

¹² Estes estereótipos são definidos em [48] e no restante do trabalho serão usados em português.

ESTEREÓTIPO	COMENTÁRIO	NOTAÇÃO
Discipline (Disciplina)	É um agrupamento coerente de elementos do processo (artefatos, papéis, atividades) cujas atividades são organizados segundo algum ponto de vista ou tema comum (Ex: Análise e Projeto, teste, implementação, etc.).	

Os elementos descritos na tabela acima podem ser usados em diagramas UML para representar diversos pontos de vista do processo de software. Neste trabalho, será mostrada a modelagem feita para o processo Extreme Programming utilizando os conceitos acima. A Figura 3.4 abaixo mostra um diagrama de atividades que representa a dinâmica de um projeto realizado seguindo o processo XP.

O processo se inicia por uma etapa inicial de explorações onde são feitos testes e experimentos com as tecnologias a serem utilizadas e verifica-se a existência de viabilidade tecnológica considerando-se as restrições do ambiente onde o sistema será implantado. Uma arquitetura inicial e um levantamento prévio dos requisitos do sistema são elaborados.

Depois disso, os clientes e os programadores escrevem as histórias que representam os requisitos do release a ser implementado. As histórias correspondem a casos de uso simplificados que recebem uma estimativa de dificuldade dada pelos programadores com base em estimativas passadas e na sua experiência. Essas estimativas servem como uma base para que os clientes escolham quantas histórias serão implementadas no release – isso segue a idéia do jogo do planejamento apresentado na seção 3.4.1.

Dentro de um release, ocorrem várias iterações onde diversas histórias são implementadas e testadas. Cada iteração segue um determinado fluxo de atividades que consistem em: planejar iteração, projetar, escrever testes de unidade, codificar, testar e integrar conforme mostrado na figura abaixo. Além disso, dentro da iteração podem ocorrer diversas alterações nos requisitos onde as histórias podem ser re-escritas e revisadas para atender as necessidades de mudança do cliente. Essas alterações podem

ser realizadas durante o planejamento da iteração onde os programadores podem reavaliar as estimativas feitas anteriormente com base nas sugestões de modificação e no feedback obtido de iterações anteriores.

No término da última iteração, a versão atual do sistema é colocada em produção no ambiente do cliente e diversas versões subseqüentes são geradas até que todos os requisitos do sistema sejam implementados e entregues ao cliente.

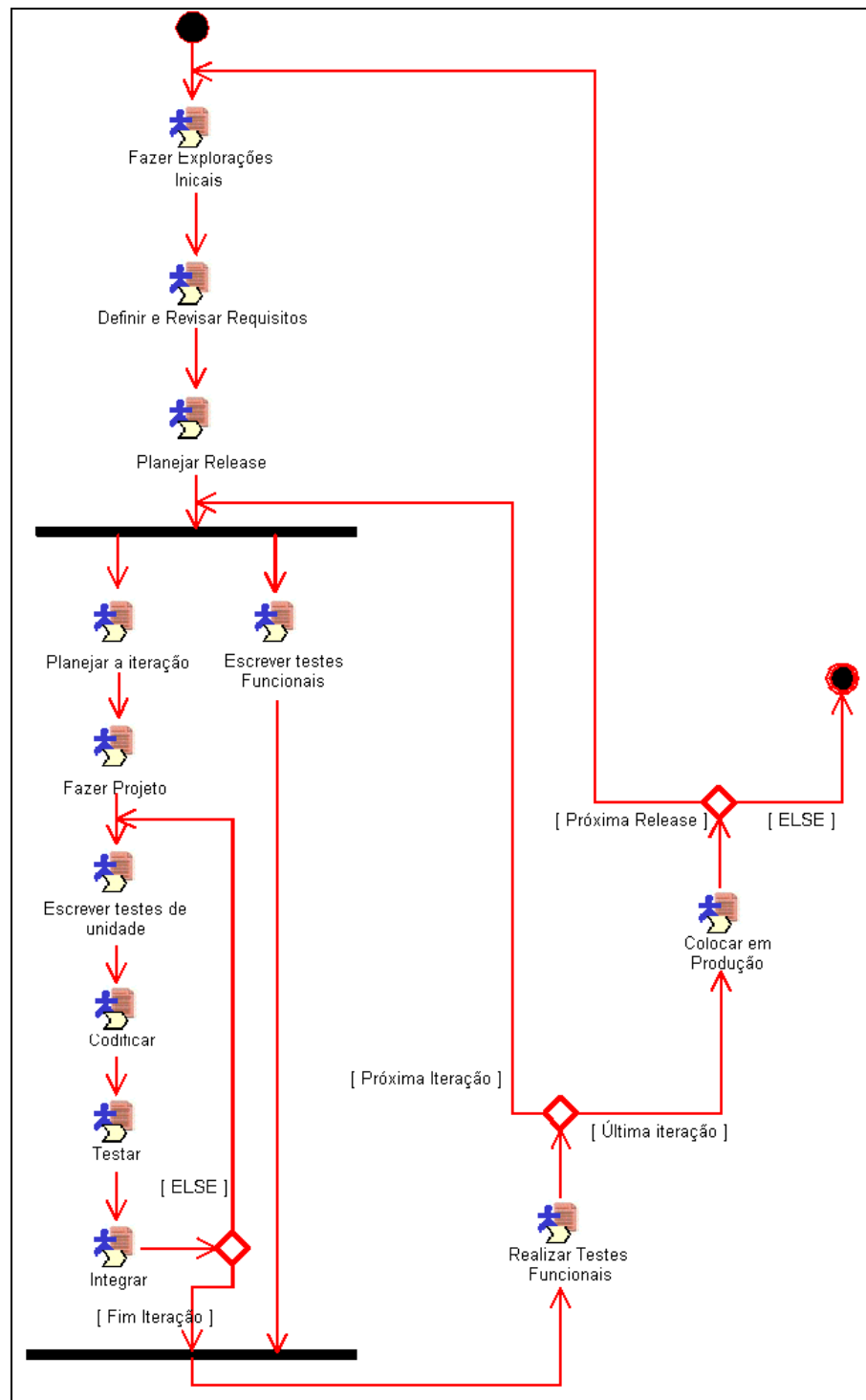


Figura 3.4 - Modelagem de XP usando diagrama de atividades da UML e estereótipos de SPEM

A modelagem dinâmica de XP mostrada na Figura 3.4 foi feita com base nos estereótipos disciplina de SPEM utilizando-se um diagrama de atividades da UML para dar uma idéia da seqüência de cada disciplina de XP ao longo do tempo. A modelagem se baseou na descrição do ciclo de vida de XP apresentada na seção 3.4.2. A próxima seção procura apresentar a modelagem estática de algumas disciplinas através de diagramas de classe UML.

3.7.3 Modelagem estática das disciplinas de XP

Nesta seção são apresentados diagramas de classe que descrevem o detalhamento de duas disciplinas da Figura 3.4 - mostradas nas Figuras Figura 3.5 e Figura 3.6. A apresentação completa de todos os diagramas é feita no [apêndice 2](#).

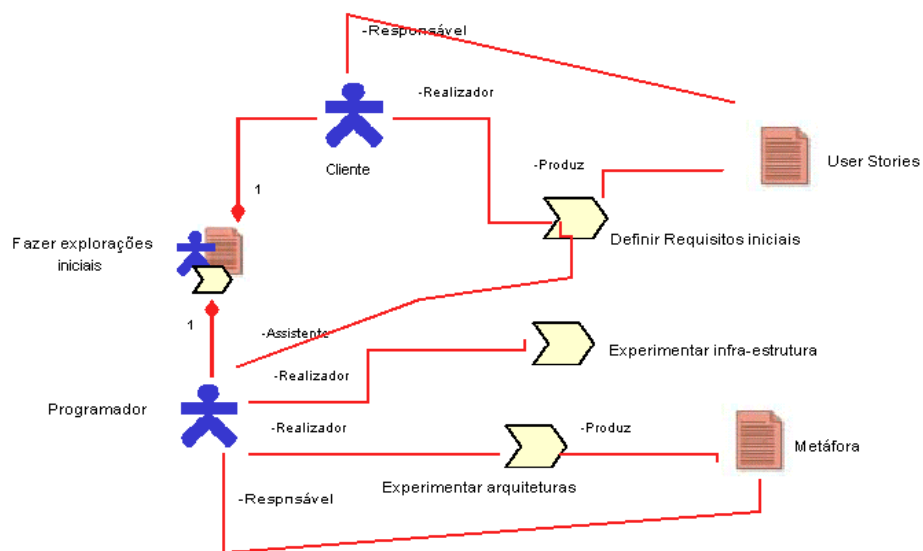


Figura 3.5 - Fazer explorações iniciais

A modelagem da Figura 3.5 procura representar o que é feito na fase de exploração de XP descrita na seção 3.4.2. Esta parte do processo XP é onde são feitas as primeiras explorações e investigações sobre o projeto a ser implementado. Os clientes são consultados e trazidos para dentro da equipe para realizar a definição prévia dos requisitos e do escopo do sistema. Os programadores são responsáveis por fazer experimentos com a possível tecnologia e infra-estrutura a ser escolhida para verificar a

viabilidade da solução e elaborar uma idéia de arquitetura através da metáfora. Algumas histórias iniciais podem ser descritas, mas o importante aqui é levantar as informações necessárias para decidir se o projeto é viável ou não. Outra disciplina de XP é descrita na figura abaixo.

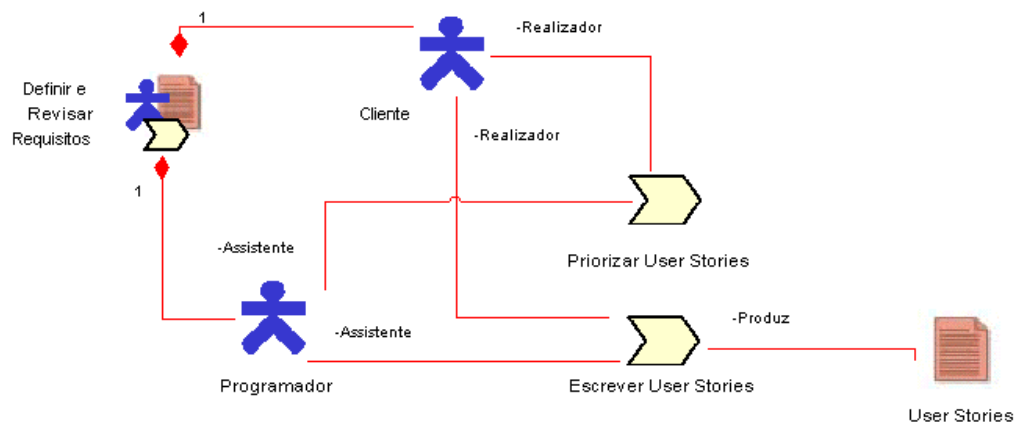


Figura 3.6 - Definir e Revisar requisitos

A modelagem da Figura 3.6 procura representar o que é descrito nas seções 3.4.1 e 3.4.2 no que se refere ao levantamento de requisitos, que é feito basicamente pelo cliente e pelos programadores, através da escrita dos cartões de história.

Esta parte do processo XP se preocupa em definir e revisar os requisitos que irão fazer parte de uma versão (release) do sistema. Como XP admite mudanças constantes nos requisitos então é importante mencionar que esses requisitos podem ser alterados a qualquer instante e que eles também serão revistos na parte de XP que trata dos requisitos da iteração. Em XP, o cliente, ou alguém que o represente, trabalha juntamente com a equipe de desenvolvimento (prática on-site customer) e é responsável por escrever e priorizar as histórias com o auxílio dos programadores. Os clientes escrevem os cartões contendo as descrições de cada história, a qual é atribuída uma certa prioridade pelo cliente. O principal resultado dessa disciplina são os cartões de história com a descrição de suas funcionalidades e as estimativas de esforço.

Ao longo desta seção, a meta-linguagem de modelagem SPEM foi usada para modelar a estrutura do processo de software XP, através da elaboração de dois

diagramas de classe contendo estereótipos de SPEM. Alguns dos benefícios da modelagem de processos já foram citados na seção 3.6.1 e a justificativa da escolha de SPEM foi mencionada na seção 3.6.2. É importante também ressaltar que o fato de se ter modelado XP não implica em nenhuma perda de eficiência, pois a modelagem não tem o objetivo de alterar a forma como se trabalha utilizando XP, nem tampouco sugere nenhuma modificação na sua estrutura.

A modelagem de XP realizada traz os seguintes benefícios:

- Simplifica o entendimento dos elementos de XP tornando mais fácil seu aprendizado.
- Simplifica o trabalho de uma organização que pretende adotar XP como processo base de engenharia de software.
- Simplifica o trabalho de uma organização que já utiliza XP e precisa fazer adaptações para atender certas características de projetos como, por exemplo, desenvolvimento de software para a Web e de software de tempo real usando XP.
- A forma de trabalhar com XP não é alterada pelo modelo, logo XP não deixará de ser ágil ou leve porque foi modelado.

A modelagem dinâmica e estática de XP produzidas nesta seção e na anterior serve como base para as adaptações feitas no próximo capítulo. A organização e a estruturação de XP em modelos facilitam o seu entendimento e também a elaboração do processo XWebProcess.

3.8 Considerações finais

Este capítulo apresentou os principais conceitos relativos a Extreme Programming. Foi dada uma explicação sobre a estrutura de XP através da descrição de seus valores, princípios e práticas e também foi mostrado o ciclo de vida que segue um projeto XP. Além disso, foi apresentada uma parte da modelagem de XP com SPEM

que traz alguns benefícios como facilitar o entendimento e a organização dos elementos de XP. Esta modelagem servirá também para facilitar a criação do processo XWebProcess a ser apresentado no próximo capítulo.