Universidade Federal de Pernambuco
Centro de Informática
Departamento de Sistemas de Computação

Graduação em Ciência da Computação

Avaliando a Metodologia Pro.NET em relação ao MSF 4.0

Mauro La-Salette Costa Lima de Araújo

TRABALHO DE GRADUAÇÃO

Recife
24 de Agosto de 2005

Universidade Federal de Pernambuco
Centro de Informática
Departamento de Sistemas de Computação

Mauro La-Salette Costa Lima de Araújo

Avaliando a Metodologia Pro.NET em relação ao MSF 4.0

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Departamento de Sistemas de Computação da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: *Prof. Dr. André Luís de Medeiros Santos*

Recife
24 de Agosto de 2005

A minha família

Agradecimentos

"*Tu deviens responsable pour toujours de ce que tu as apprivoisé.*"
— ANTOINE DE SAINT-EXUPÉRY

Embora este trabalho seja fruto de apenas quatro meses de esforço, sua conclusão marca o fim de uma caminhada bem maior, de quatro anos e meio. Agradecer neste pequeno espaço a todos que , direta ou indiretamente, contribuíram para a minha formação faz-se, desta maneira, necessário.

Agradeço em primeiro lugar aos meus pais, que tornaram realidade a conclusão de meu curso.

A todos que fazem o Centro de Informática da UFPE, professores e funcionários, pelo ambiente estimulante e enriquecedor fornecido ao longo destes anos e, em especial, a dois professores: André Santos, pelas oportunidades pesquisa e monitoria a mim oferecidas, e Fernando Fonseca, cuja tutela durante minha permanência no Grupo PET foi essencial para minha formação acadêmica e pessoal.

Aos colegas de graduação, que comigo compartilharam as horas de esforço e trabalho necessárias para a conclusão dos projetos desenvolvidos ao longo do curso. Em especial a Alexandra, Julio, Marcus, Paulo e Rafael.

A todos que contribuíram de alguma forma para elaboração deste trabalho. Em especial a André Furtado, cujas minuciosas correções e sugestões foram de extrema valia.

Aos meus amigos mais próximos, que me cativaram e deixaram-se cativar por mim, estabelecendo um vínculo que, embora possa parecer desfeito em algumas oportunidades, nos tornam eternamente responsáveis uns pelos outros.

Mundo mundo vasto mundo,
se eu me chamasse Raimundo
seria uma rima, não seria uma solução
Mundo mundo vasto mundo,
mais vasto é o meu coração.

— CARLOS DRUMMOND DE ANDRADE

Resumo

Este trabalho se propõe a estabelecer uma análise comparativa entre a metodologia Pro.NET e uma das vertentes da versão 4.0 do *Microsoft Solutions Framework* (MSF), intitulada *MSF For CMMI Process Improvement*. São apresentadas breves introduções à metodologia Pro.NET e ao MSF 4.0 e a seguir, é delineada uma análise comparativa que avalia o nível de satisfação da Pro.NET em relação ao MSF 4.0. Através desta análise, este trabalho visa facilitar uma futura evolução da Pro.NET em direção ao MSF 4.0.

Palavras-chave: Processos de software, Pro.NET, CMMI, MSF

Abstract

This work intends to establish a comparative analysis between the Pro.NET methodology and a *Microsoft Solutions Framework* (MSF) variation called *MSF For CMMI Process Improvement*. It presents brief introductions to both Pro.NET and MSF 4.0, and then elaborates a comparative analysis which evaluates the level of satisfaction of one against the other. Through this analysis, this work aims to facilitate a forthcoming Pro.NET evolution towards MSF 4.0.

Keywords: Software processes, Pro.NET, CMMI, MSF

Sumário

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

"If this is true, building software will always be hard. There is inherently no silver bullet."

— F. P. BROOKS

A indústria de desenvolvimento de software tem experimentado, ao longo das últimas décadas, o que comumente convencionou-se denominar a *crise do software*. Quando da identificação desta crise em 1968, durante uma reunião do *NATO Science Comittee* [NB68], foi cunhado o termo "Engenharia de Software" para sintetizar a aspiração, naquele momento, de tornar a tarefa de produção de software uma disciplina da engenharia. Entretanto, como bem exposto por Brooks [Bro87], existem dificuldades essencialmente atreladas à tarefa de produção de software e, décadas depois, a Engenharia de Software mostra-se incapaz de lidar, de maneira completa, com a natureza abstrata de seu principal produto. De fato, dados de 2000 [Gro01] mostram que: 23% dos projetos de software são cancelados antes de sua conclusão; 49% são concluídos ultrapassando os custos e prazos previstos; e apenas 28% são concluídos dentro dos custos e prazos previstos inicialmente.

Ao longo dos anos, muitas têm sido as abordagens para dirimir as dificuldades presentes na tarefa de construção de artefatos de software. A utilização de linguagens de alto nível, ambientes integrados de desenvolvimento e linguagens gráficas para modelagem, por exemplo, aumentaram significativamente a produtividade das equipes de desenvolvimento, mas atacaram apenas dificuldades *acidentais* [Bro87]. O êxito de um projeto de software, sobretudo os de grande porte, depende não apenas da produtividade da equipe de desenvolvimento, mas também de planejamento adequado. Mesmo os melhores programadores serão suplantados por dificuldades de comunicação, cronogramas arbitrários, treinamento insuficiente, documentação precária, mudanças não gerenciadas, etc. Uma seqüência planejada de passos, tarefas e atividades para a produção de software se faz necessária. Em outras palavras, o uso de um processo.

Erradicar abordagens artesanais através do uso de processos tem sido alvo de esforços por parte da comunidade de engenharia de software, sendo vários modelos de processos desenvolvidos nos últimos anos. Durante este período, modelos inflexíveis como o Cascata [Roy70] deram lugar a abordagens mais dinâmicas e baseadas em conceitos como iterações, prototipação evolucionária e orientação a riscos. Alguns modelos de processo (*frameworks*), como o *Rational Unified Process* (RUP) [BJR99], provêem um nível detalhado do fluxo das atividades a serem seguidas, enquanto outros, como o *Microsoft Solution Framework* (MSF) [Mic03], fornecem apenas linhas mestras e recomendam boas práticas para a instanciação de um processo.

Ainda que de longa data, o problema do estabelecimento de processos para controlar o desenvolvimento de software continua crítico: muitas organizações utilizam processos ineficientes que resultam em produtos de baixa qualidade; e mesmo tentativas de utilizar processos reconhecidos e amplos como o RUP resultam em fracasso por dificuldades para adaptá-los às suas realidades.

A filosofia de que a melhoria contínua no processo de produção de software acarretaria um acréscimo de qualidade nos produtos foi influenciada por trabalhos em outras disciplinas da engenharia (sobretudo os de W. E. Deming), e culminou com a iniciativa do *Software Engineering Institute* (SEI) [Ins04], a partir de uma requisição do DoD[1] dos EUA em meados da década de 1980, para desenvolver um modelo de avaliação do nível de maturidade do processo de desenvolvimento de software para organizações. Tal modelo foi denominado *Capability Maturity Model for Software* (SW-CMM ou simplesmente CMM) [PCCW93]. O CMM evoluiu durante os últimos anos e absorveu outros padrões de avaliação de processos como o *Electronic Industries Alliance Interim Standard* (EIA/IS), para engenharia de sistemas, e o *Integrated Product Development Capability Maturity Model* (IPD-CMM), passando a chamar-se *Capability Maturity Model Integration* (CMMI) [CKS03].

Durante a década de 1990, diversas organizações, impulsionadas pelo mercado extremamente competitivo, buscaram a adequação ao CMM como uma maneira de comprovar a qualidade do seu processo de desenvolvimento de software. E agora, com o advento do CMMI e descontinuidade do CMM, tais organizações estão realizando a migração entre os modelos. Assim, o CMMI evolui rapidamente para se tornar um padrão no que concerne à avaliação do nível do processo de desenvolvimento de software.

Também nos processos de desenvolvimento a influência do modelo da SEI pode ser sentida: o MSF, que consiste em um conjunto de boas práticas de projeto utilizadas pela Microsoft, terá em sua nova versão (4.0) uma variante com práticas direcionadas aos níveis 2 e 3 do CMMI e chamar-se-á *MSF for CMMI Process Improvement*.

A metodologia Pro.NET [CTXa], desenvolvida no *Centro de Tecnologia XML* de Recife, foi concebida com base no RUP, no MSF, em conceitos de desenvolvimento ágil (*eXtreme Programming*) e é voltada para o desenvolvimento de projetos que utilizem a plataforma tecnológica .NET. Um trabalho anterior realizou uma análise comparativa entre a metodologia Pro.NET e CMM-2 [Mar03] apresentando sugestões de melhorias na metodologia. Diante do contexto de renovação de uma das principais bases da Pro.NET (o MSF) este trabalho terá por objetivo avaliar a metodologia Pro.NET em relação à nova versão do MSF. O produto do trabalho, além da análise comparativa, será um conjunto inicial de sugestões de melhoria na metodologia.

É importante frisar que este trabalho baseia-se em uma versão preliminar (*beta*) do MSF 4.0, que, além de apresentar algumas inconsistências, pode vir a ser modificada quando do lançamento de versões mais estáveis. Assim, o conteúdo deste trabalho pode apresentar discrepâncias em relação à versão final do MSF 4.0.

O restante do trabalho está organizado da seguinte forma: no Capítulo 2 são analisadas comparativamente as versões 3 e 4 do MSF; o Capítulo 3 apresenta uma visão geral da metodologia Pro.NET; o Capítulo 4 realiza uma comparação entre a metodologia Pro.NET e o MSF

[1] *Departament of Defense*

4.0 e, por fim, o Capítulo 5 apresenta conclusões e sugestões para trabalhos futuros.

CAPÍTULO 2

Uma visão comparativa entre versões do MSF

"If you can't describe what you are doing as a process, you don't know what you're doing."
— W. EDWARDS DEMING

Este capítulo descreve as principais características do MSF 3.0 e, a seguir, apresenta uma variação de sua nova versão, o *MSF For CMMI Process Improvement*.

2.1 Origem e visão geral do MSF 3.0

O *Microsoft Solutions Framework* surgiu em 1994 como um conjunto de boa práticas compiladas pela Microsoft a partir de sua experiência na produção de software e em serviços de consultoria. Desde então, o MSF evoluiu, tornando-se um *framework* flexível para nortear o desenvolvimento de projetos de software.

Os elementos básicos que compõe o MSF 3.0 são os seguintes:

- *Princípios fundamentais*
- *Modelo de Processo*
- *Modelo de Equipe*
- *Disciplinas*

A próximas subseções apresentam cada um destes elementos

2.1.1 Princípios fundamentais

Os *Princípios Fundamentais* formam a base do *framework* e resumem sua filosofia, desta forma suas influências podem ser sentidas pelo restante do MSF, guiando a organização dos Modelos de Equipe e Processo. Os princípios fundamentais são os seguintes:

- Encorajar comunicação ampla e aberta;
- Trabalhar em direção a uma visão integrada;
- Fortalecer os membros da equipe;

- Estabelecer prestação de contas clara e responsabilidade compartilhada;
- Concentrar no fornecimento de valor de negócio;
- Permanecer ágil, esperando mudanças;
- Investir em qualidade;
- Aprender a partir de todas as experiências.

2.1.2 Modelo de Equipe

O *Modelo de Equipe* do MSF [Mic02b] tem por objetivo definir os papéis e responsabilidades das pessoas que estarão envolvidas em um projeto. Uma característica do referido modelo é que o mesmo prega a formação de uma *equipe de pares*, isto é, não com uma organização hierárquica *top-down* tradicional, mas um modelo para uma equipe colaborativa com responsabilidades compartilhadas. Subjacente ao modelo está o conceito de que qualquer projeto de software, para ser considerado bem sucedido, deve satisfazer a alguns objetivos chave de qualidade. Assim, o Modelo de Equipe do MSF define cada um de seus papéis baseado na necessidade de adequação a um objetivo de qualidade (Tabela 2.1). Para atingir tais objetivos, cada grupo de papéis deve atuar em um conjunto de *áreas funcionais* com *responsabilidades* associadas.

Objetivo chave de qualidade	*Papéis do Modelo de Equipe*
Entregar a solução dentro das restrições do projeto	Gerência de Programa
Construir a solução obedecendo às especificações	Desenvolvimento
Garantir que a solução só será entregue após a identificação de seus defeitos	Teste
Implantar a solução sem maiores impactos	Gerência de *Release*
Maximizar a performance do usuário	Experiência do Usuário
Satisfazer os clientes	Gerência de Produto

Tabela 2.1 Grupos de papéis do MSF e objetivos chave de qualidade

Vale ressaltar que cada papel pertencente ao Modelo de Equipe do MSF não corresponde, necessariamente, a uma única pessoa. Cada um deles pode ser melhor definido como um *grupo de papéis* (*role cluster* no original) no qual um ou mais membros da equipe podem estar inseridos. Desta forma, o modelo mostra-se bastante flexível para fins de adaptação ao tamanho do projeto e, conseqüentemente, ao tamanho da equipe envolvida. Pode-se, inclusive, adaptar o modelo à realidade de um projeto pequeno, onde haja a necessidade de mais de um papel ser exercido por um único membro da equipe. Neste caso, o modelo faz restrições quanto às possibilidades de acúmulo de papéis.

Os membros da equipe que fazem parte da *Gerência de Produto* têm como principal objetivo satisfazer as necessidades do cliente. Convém lembrar a distinção entre cliente e usuário, pois apesar de haver coincidência entre estes dois papéis em certos projetos, nem sempre o

Figura 2.1 Grupos de papéis do Modelo de Equipe do MSF

principal beneficiado pela solução (o cliente) representa o seu usuário final: uma organização que solicitou o desenvolvimento de um dado sistema pode ser considerada o cliente do projeto, enquanto os indivíduos que fazem parte desta, e que o utilizarão efetivamente, os usuários. Para satisfazer as necessidades do cliente os membros deste grupo de papéis devem atuar nas áreas funcionais de *marketing*, *valorização do negócio*, *advocacia do cliente* e *planejamento de produto*. Os integrantes da Gerência de Produto são responsáveis por manter uma comunicação constante entre o restante da equipe e o cliente, de maneira a gerenciar as expectativas do último com relação à solução. Outras importantes responsabilidades incluem a coleta e priorização dos requisitos de negócio, o desenvolvimento de um plano de *release* multi-versão (importante dentro de um contexto iterativo) e a determinação dos critérios de sucesso em relação ao cliente.

A *Gerência de Programa* é responsável por garantir a entrega da solução obedecendo às restrições impostas pelo projeto. Tais restrições variam desde o comprometimento com cronogramas e recursos limitados, à garantia da conformidade da arquitetura projetada com a especificação funcional. As áreas funcionais que agrupam as responsabilidades deste grupo de papéis são: *gerência de projeto*, *arquitetura da solução*, *garantia de processo* e *serviços administrativos*. As responsabilidades dos integrantes deste grupo de papéis incluem, mas não se limitam a: o acompanhamento e gerenciamento do orçamento e da alocação de recursos; o comando do projeto da arquitetura da solução e do processo de garantia de qualidade; execução de atividades administrativas.

Tendo como principal objetivo a conversão da especificação funcional na aplicação propriamente dita, o grupo de papéis de *Desenvolvimento* agrega as áreas funcionais de *consul-*

toria em tecnologia, *análise e projeto*, *desenvolvimento da aplicação* e *desenvolvimento da infra-estrutura*. É responsabilidade deste grupo de papéis servir como consultoria tecnológica durante todo o projeto, provendo pareceres no que diz respeito à tecnologia e construindo protótipos para validar a mesma. É também de sua competência o refinamento da arquitetura em um modelo mais próximo do da implementação (atividade semelhante ao fluxo de análise e projeto do RUP), a condução de testes unitários e as atividades de revisão de código.

Identificar e tratar todos os defeitos apresentados pela solução antes de seu lançamento é a principal meta de qualidade do grupo de papéis de *Teste*. Atuando nas áreas funcionais de *planejamento de testes*, *engenharia de testes* e *relatório de testes*, os membros deste grupo devem conceber a abordagem a ser utilizada e planejar o processo, além de elaborar os casos de teste e conduzir a sua execução. Proporcionar à equipe um panorama do patamar de qualidade da solução também é uma atribuição deste grupo, que alcança tal objetivo a partir do rastreamento e relato dos defeitos atuais da solução.

Atender às necessidades do usuário é o principal objetivo do grupo *Experiência do Usuário*. As áreas funcionais abordadas por este grupo são as de *acessibilidade*, *internacionalização*, *comunicações técnicas*, *treinamento*, *usabilidade* e *design gráfico*. As atividades agregadas sob estas áreas corroboram para maximizar a performance do usuário quando da utilização da solução, isto é, fornecer ao mesmo um ambiente produtivo e que, ao mesmo tempo, seja o mais intuitivo possível. Membros deste grupo agem como advogados do usuário durante o processo de desenvolvimento, gerenciando seus interesses a partir dos requisitos coletados nas fases iniciais.

Finda a construção da solução e o processo de testes, cabe à *Gerência de Release* garantir que a transição para o ambiente de produção seja bem sucedida e sem maiores impactos. Através das áreas funcionais de *infra-estrutura*, *suporte*, *operação* e *gerenciamento comercial de release*, os membros deste grupo tentarão preencher a lacuna existente entre os ambientes de desenvolvimento e produção. Gerenciarão o processo de implantação para que a manutenção da solução e o suporte ao usuário sejam otimizados.

Como já foi dito anteriormente, o Modelo de Equipe do MSF opõe-se aos modelos *top-down* tradicionais nos quais o gerente de projeto está no comando absoluto da equipe. A filosofia de responsabilidade compartilhada prevê que cada membro da equipe tem poderes para atuar na área de sua competência sem a intervenção da gerência. Quando conflitos de interesses ocorrem, os integrantes da equipe devem se reunir e procurar a solução cujo benefício para o projeto como um todo seja maior. Muitas vezes, entretanto, não é possível chegar a um consenso. Nestes casos, o MSF prevê que o grupo de papéis de Gerência de Programa será o responsável pelas decisões.

Outro questionamento que surge no contexto de uma equipe de pares é: quem é responsável pela prestação de contas (comunicação externa) do projeto, uma vez que a gerência no sentido convencional não existe? O Modelo de Equipe do MSF prega que cada grupo de papéis é responsável pela prestação de contas na área de sua competência (Figura 2.2). A exceção dos grupos de papéis de Desenvolvimento e Teste, que devem permanecer isolados do mundo exterior, os outros papéis corroboram para a comunicação externa do projeto como um todo.

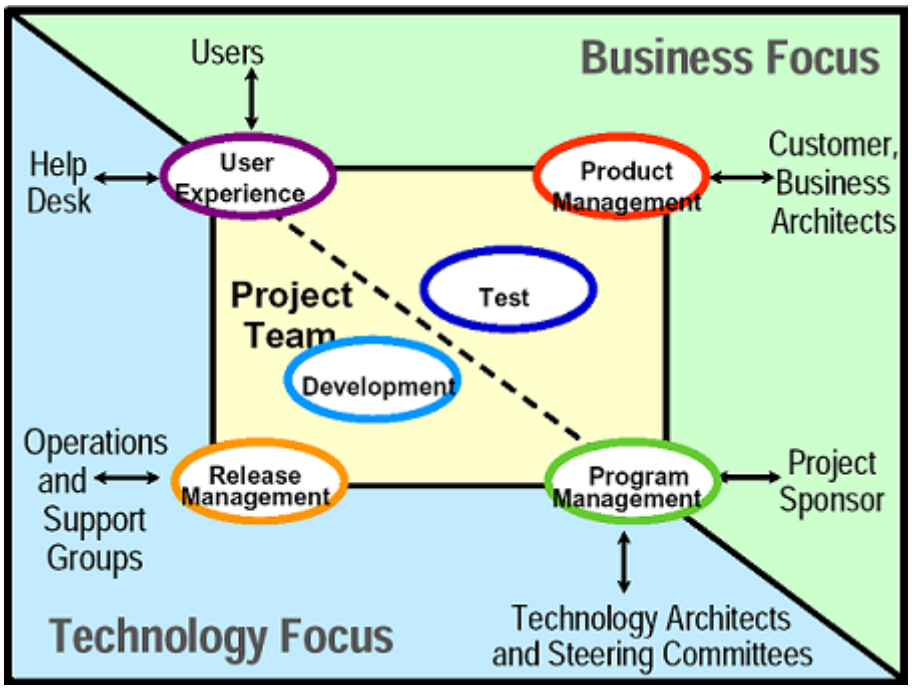

Figura 2.2 Comunicação com o mundo exterior no Modelo de Equipe do MSF

2.1.3 Modelo de Processo

O *Modelo de Processo* do MSF [Mic02a] tem por finalidade estabelecer a ordem de execução das atividades do projeto, assim como apontar os responsáveis pelas mesmas. Combinando as abordagens dos modelos de ciclo de vida cascata e espiral, o Modelo de Processo do MSF estabelece a execução de sucessivas iterações subdivididas em 5 fases cada uma (Figura 2.3).

Cada fase representa um período de tempo no qual a equipe dispensará atenção a aspectos específicos do desenvolvimento. Tais aspectos estão relacionados às responsabilidades de um ou mais papéis do Modelo de Equipe, fazendo com que a ação de alguns predomine em cada fase. O término da fase vem com a concepção de um marco (*milestone*), que pode se apresentar na forma de documento ou de artefato tecnológico e, por haver uma correspondência indireta entre os papéis do Modelo de Equipe e as fases, cada marco estará sob a responsabilidade de um grupo de papéis. Os marcos são importantes no sentido de promover uma sincronização entre os participantes do projeto já que lhes é dada oportunidade de avaliar o quão bem este caminhou até aquele momento.

Além dos marcos mencionados, o MSF sugere que as fases produzam marcos intermediários. Estes serviriam como indicadores precoces de progresso e particionariam as atividades. Como os marcos intermediários dependem bastante do tipo do projeto, o MSF apenas os sugere.

Estabelecer uma visão comum dos objetivos do projeto é a principal meta da *Fase de Visão*. Por visão comum, entende-se aquela que é compartilhada por todos os membros da equipe, assim como pelos clientes. Através dela, a equipe terá um melhor entendimento do que o cliente espera da solução e a captura dos requisitos de negócio será facilitada. O marco delimitador

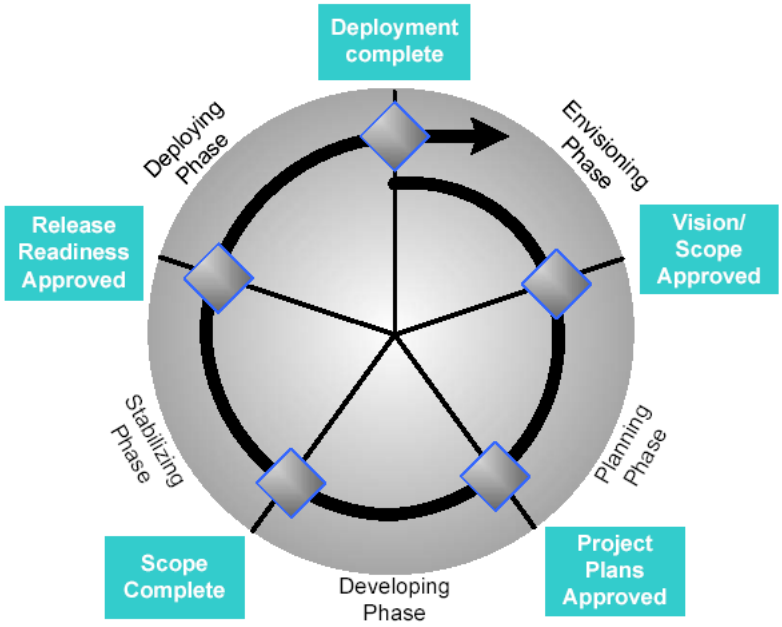

Figura 2.3 Fases que compoem cada iteração no Modelo de Processo do MSF

Marco	*Principal Responsável*
Aprovação da Visão/Escopo	Gerência de Produto
Aprovação dos Planos de Projeto	Gerência de Programa
Finalização do Escopo	Desenvolvimento e Experiência do Usuário
Aprovação de *Release*	Teste e Gerência de *Release*
Finalização da Implantação	Gerência de *Release*

Tabela 2.2 Relação entre os marcos e os seus principais responsáveis

desta fase é o *Documento de Visão e Escopo* que definirá a visão (uma perspectiva "sem limites" da solução) e o escopo (as partes da visão que podem ser atingidas considerando as restrições do projeto). O processo de gerenciamento de riscos, que continua durante todo o resto do projeto, é aqui iniciado através da elaboração de um documento de riscos. Os marcos intermediários sugeridos para esta fase são um documento de estrutura do projeto e um esboço inicial do Documento de Visão e Escopo. O primeiro tem por objetivo descrever como a equipe está organizada, definindo responsabilidades, quem exercerá os papéis do Modelo de Equipe e pontos de comunicação com o ambiente extra-equipe. Já o esboço inicial do Documento de Visão e Escopo é de extrema importância para a consolidação final do documento, uma vez que o esboço será submetido a revisões pela equipe e clientes.

Durante a *Fase de Planejamento* a idéia da solução, concebida na fase anterior, é refinada com os detalhes técnicos necessários para que ela possa ser implementada. Este refinamento tem início com a captura dos requisitos, que são divididos em 4 categorias: de negócio, de usuário, operacionais ou de sistema. Uma vez identificados os perfis de usuário do sistema,

as operações a serem realizadas pelos mesmos são refinadas em *casos de uso* e tem início a elaboração dos diferentes níveis de projeto da solução: conceitual, lógico e físico. Todos os resultados das atividades de projeto, assim como os requisitos e casos de uso, são registrados no *Documento de Especificação Funcional*. Elaborado este documento, tem-se em mãos uma especificação completa do que deverá ser implementado e o planejamento propriamente dito pode começar. O Modelo de Processo do MSF prega a utilização de técnicas *bottom-up* de planejamento, isto é, cada grupo de papéis do Modelo de Equipe deve elaborar um plano e cronograma para o artefato ou atividade pelo qual é responsável, cabendo a gerência de programa a integração de todos estes em dois documento denominados *Plano Mestre de Projeto* e *Cronograma Mestre de Projeto*. Também nesta fase é elaborado um plano de gerência de riscos. Como marcos intermediários desta fase, é sugerido que cada artefato, a medida que for finalizado, seja submetido a gerência de configuração para integrar um *baseline* do projeto. Uma vez neste estado, tais documentos só poderão ser alterados formalmente, através de solicitações de mudança.

A partir da especificação funcional e dos planos e cronogramas de projeto produzidos anteriormente, a fase de *Desenvolvimento* concentrar-se-á na implementação dos componentes da solução (documentação e código). Esta fase culmina com a *Finalização do Escopo* da solução, isto é, a implementação do que fora definido no documento de especificação funcional. Outro importante produto desta fase são as especificações de casos de teste. O principal marco intermediário desta fase é a geração de *builds* (versões internas). Os builds representam importantes pontos de sincronização e mensuração para a equipe de desenvolvimento, já que muitas atividades de codificação são feitas em paralelo.

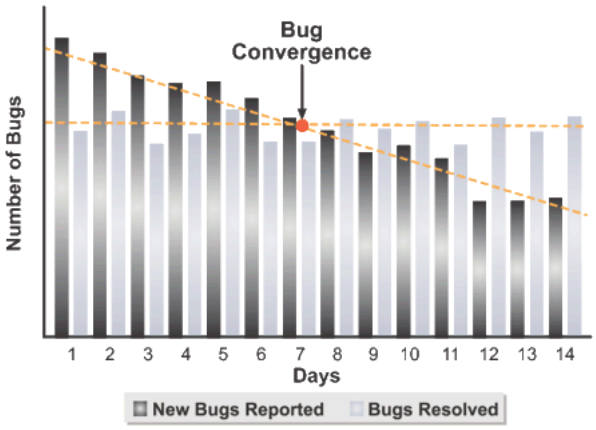

Figura 2.4 Convergência de defeitos

Após a Finalização do Escopo, a solução passa à *Fase de Estabilização*. É nesta fase que a solução é testada em um ambiente o mais próximo possível do de produção até atingir a estabilidade. Cabe ao grupo de papéis de Teste identificar e priorizar os defeitos mais críticos, reportando os mesmos à equipe para que a correção seja providenciada. Embora não seja possível prever quando todos os defeitos da solução estarão identificados, existem algumas medidas estatísticas que indicam a proximidade da estabilização da solução. Uma delas é a

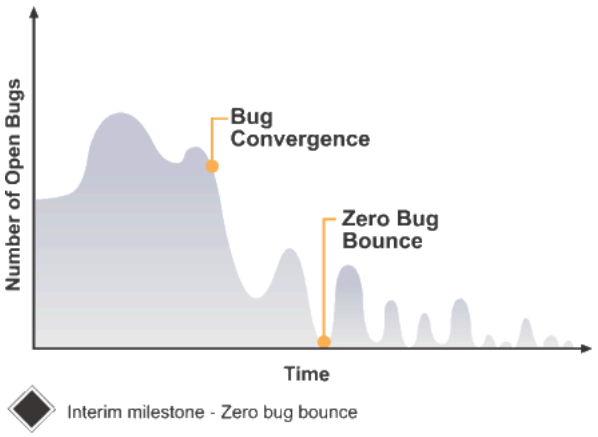

Figura 2.5 *Zero bug bounce*

convergência de defeitos (Figura 2.4), quando o número de defeitos relatados pela equipe de testes é menor do que o número de defeitos corrigidos pela equipe de desenvolvimento. A outra medida, denominada *zero bug bounce* (Figura 2.5), indica que a solução atingiu, pelo menos por um breve período de tempo, a ausência de defeitos. Ambas são sugeridas como marcos intermediários desta fase. Outro importante marco intermediário é execução de um projeto piloto, que visa testar a solução em um subconjunto do ambiente de produção. Após a identificação de todos os defeitos, ocorre a *aprovação da release* e o projeto segue para a *Fase de Implantação*.

A última das fases do Modelo de Processo do MSF tem por objetivo transferir a solução para o seu ambiente de produção, onde a mesma deverá passar a gerar o valor de negócio esperado pelo cliente. O reconhecimento, por parte do cliente, de que a equipe atingiu os objetivos a que se propôs, finaliza a Fase de Implantação.

2.1.4 Disciplinas

No MSF as disciplinas representam conjuntos de práticas e abordagens consagradas pela indústria e que estão documentadas em corpos de conhecimento (*bodies of knowledge*). As três disciplinas que compõe o MSF são *Gerenciamento de Projetos*, *Gerenciamento de Riscos* e *Gerenciamento de Conhecimento*.

A disciplina de gerenciamento de projetos apresenta uma abordagem de gerência de projeto distribuída através da equipe, em conformidade com a noção de *equipe de pares*. Não obstante, a disciplina reconhece que o grupo de papéis de Gerência de Programa, no contexto de grandes projetos, precisa ser especializado na forma de um gerente de projeto e de um arquiteto de solução, para lidar com os aspectos gerenciais e técnicos respectivamente.

O gerenciamento de riscos tornou-se requisito indispensável para o sucesso de grandes projetos envolvendo tecnologia. A natureza exploratória inerente ao processo de inovação tecnológica faz do gerenciamento de riscos uma peça chave dos atuais processos de desenvolvimento de software. No MSF a disciplina de gerenciamento de riscos lida com estes usando um ciclo

contínuo de identificação, priorização, elaboração de planos de ação e monitoramento.

A disciplina de gerenciamento de conhecimento tem por objetivo estabelecer um processo para definir, avaliar e mudar o conhecimento e as habilidades necessárias à execução do projeto. Um indivíduo que desempenhe atividades dentro de um determinado papel deve ser capaz de realizar todas as responsabilidades que lhe foram atribuídas. Neste contexto, a presente disciplina garante que ele esteja munido das habilidades e o conhecimento necessário para desempenhar o seu papel.

2.2 MSF for CMMI Process Improvement

Cada projeto de software é único devido a uma série de fatores que incluem desde o tipo de aplicação a ser construída, até restrições de cronograma e recursos, passando pelas necessidades específicas de cada cliente. A filosofia da impossibilidade da definição de um único processo de desenvolvimento de software adaptável a todas as situações, já se mostrava presente na versão 3.0 do MSF, dada sua natureza abstrata em oposição a processos bem detalhados como o RUP. Dando continuidade a esta vertente e dentro do contexto da popularização dos processos ágeis, a versão 4.0 do MSF foi lançada em duas variações: uma abordando processos ágeis e outra processos formais. A variação "ágil" do MSF 4.0 tem o nome de *MSF For Agile Software Development* e está fora do escopo deste trabalho. Esta subseção concentrar-se-á em oferecer uma visão geral de sua vertente "formal", o *MSF for CMMI Process Improvement*, doravante denominado MSF 4.0 para fins de simplificação.

A análise da versão preliminar do MSF 4.0 mostra que este continuou bastante semelhante ao seu predecessor em certos aspectos. Estes incluem principalmente os princípios fundamentais, a macroestrutura do Modelo de Processo e a maioria das boas práticas. As maiores modificações ocorreram com a concretização do modelo de equipe e das boas práticas do framework, através do estabelecimento de fluxos de trabalho (*workstreams*). A seguir, é traçada uma linha geral das modificações mais relevantes na versão 4.0 do MSF.

2.2.1 Modelo de Equipe

Os grupos de papéis presentes na versão anterior do MSF indicavam áreas de competência e responsabilidades a serem preenchidas pelos membros da equipe. Não havia uma definição clara de papéis específicos dentro destes grupos, ficando este processo a cargo de quem estivesse implantando o *framework*. O MSF 4.0, por outro lado, apresenta papéis[1] bem definidos, dividindo responsabilidades entre diferentes membros da equipe. Outra modificação significativa, foi a introdução de um novo grupo de papéis, o de arquitetura. Estas alterações refletem, em parte, o desejo de conformidade com o CMMI, como será discutido adiante. A Figura 2.6 mostra a relação entre cada grupo e os papéis associados.

A natureza destas modificações não alterou um dos conceitos fundamentais do Modelo de Equipe: cada grupo de papéis é responsável pela satisfação de um objetivo de qualidade

[1] Em oposição ao termo *grupo de papéis*, utilizado durante a descrição do MSF 3.0, o termo *papel* será utilizado para descrever o cada elemento do Modelo de Equipe do MSF 4.0, já que, neste caso, há coincidência entre os papéis definidos e a figura de um único indivíduo.

específico do projeto. A adição do grupo de *Arquitetura*, por exemplo, visa *a garantia dos alicerces técnicos sobre os quais o valor de negócio será construído*. As próximas definições apresentam uma idéia geral das responsabilidades de cada papel do Modelo de Equipe.

Pode-se notar aqui, inconsistências na documentação da versão preliminar do MSF 4.0: apesar da Figura 2.6 mostrar o papel de *Arquiteto de Infraestrutura*, não há qualquer referência a este no restante da documentação; além disso, o papel de *Especialista em Qualidade de Serviço* não é referenciado na mesma figura, embora apareça na documentação.

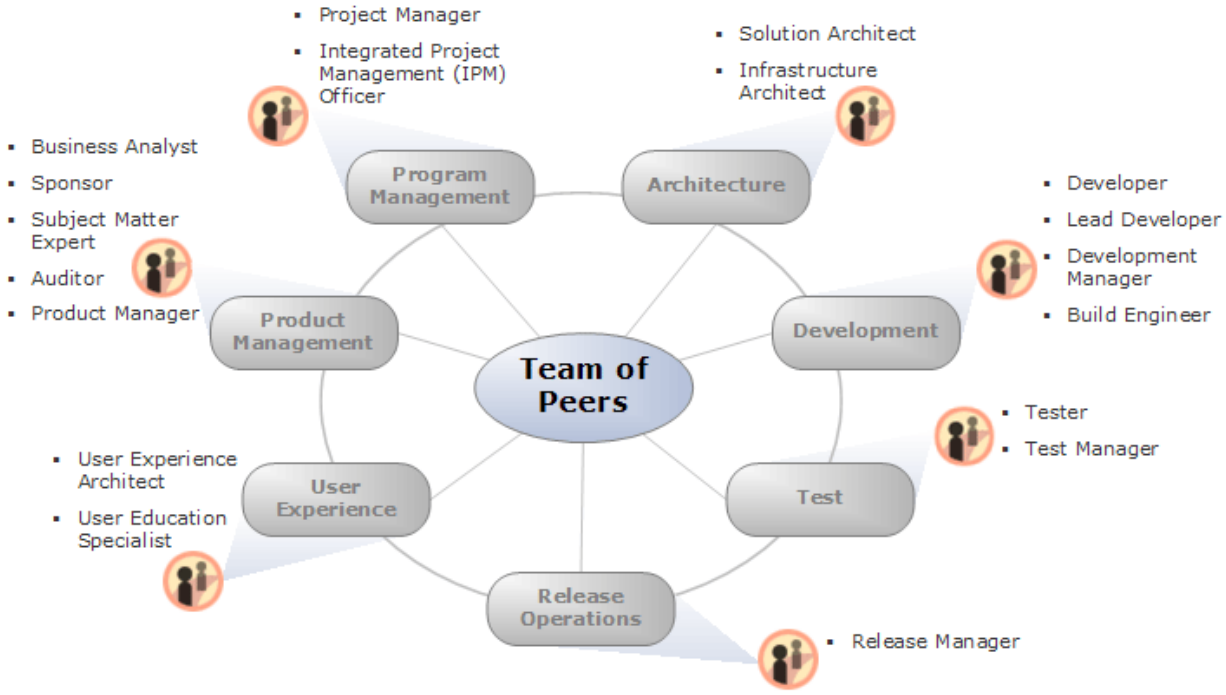

Figura 2.6 Grupos e respectivos papéis no MSF 4.0

2.2.1.1 Gerência de Programa

Gerente de Projeto: responsável por garantir a realização do projeto dentro dos prazos e custos estimados, gerando valor de negócio para o cliente. Suas responsabilidades incluem o planejamento e a construção de cronogramas do projeto e de suas iterações, monitoramento e relato da situação do projeto e a identificação e mitigação dos riscos. Suas estimativas, planejamento e cronogramas devem ser baseadas em consultas aos diversos membros da equipe.

Integrated Project Management Officer : é o executivo responsável pelo planejamento e alocação de recursos em um nível organizacional. Para coordenar os os diversos projetos pertencentes a um *portfolio*[2], o IPM Officer realiza reuniões com os gerentes de

[2]O MSF usa o termo *portfolio* para descrever um conjunto de projetos relacionados. O consagrado termo *programa* é evitado a fim de não ser confundido com o grupo de papéis de *gerência de programa*.

projeto, a fim de definir prioridades quanto a alocação de recursos e cronogramas.

2.2.1.2 Arquitetura

Arquiteto: é o papel que garante a manutenção de uma visão técnica consistente e unificada de todo o projeto. Sua principal função é projetar a arquitetura do sistema, que deverá guiar o resto da equipe durante a fase de Desenvolvimento. Durante o projeto da arquitetura, o arquiteto deverá favorecer baixo acoplamento, reuso e alta coesão dos componentes que constituem o projeto, garantindo que eles possam ser construídos e testados independentemente.

2.2.1.3 Desenvolvimento

Desenvolvedor: compõe a equipe responsável pela construção do software propriamente dito, ou seja, responsável pela etapa de codificação. Por este motivo, o desenvolvedor deve dispensar o mínimo de tempo possível a atividades que envolvam comunicação. Adicionalmente, o desenvolvedor poderá se envolver nas etapas de captura de requisitos e de análise e definição da arquitetura. O *Lider de Desenvolvimento* tem por atribuição servir como uma ponte entre a equipe de desenvolvimento e o resto do time e, por esta razão, espera-se que pelo menos 50% de seu tempo seja dedicado a atividades de comunicação, sendo o restante dedicado a liderança e assistência ao time de desenvolvimento, raramente tomando parte em atividades que envolvam codificação.

Gerente de Desenvolvimento: encarregado do gerenciamento da equipe de desenvolvimento, recebendo relatórios por parte dos desenvolvedores e, mais freqüentemente do Líder de Desenvolvimento. Deve gerenciar o trabalho da equipe através de métricas e análises que indiquem a produtividade e a qualidade do trabalho produzido, colaborando com o gerente de projeto para garantir o bom andamento do processo durante a fase de Desenvolvimento.

Engenheiro de Build: tem por função promover a integração do código fonte do projeto durante a geração de versões. Para isso deve possuir uma visão global do projeto, gerar *scripts* de automação, etc.

Especialista em Qualidade de Serviço: é um desenvolvedor especializado cuja principal meta é zelar por requisitos como performance, escalabilidade e segurança. Através da concentração das habilidades necessárias para satisfação de tais requisitos em um único papel, os outros membros da equipe de desenvolvimento podem manter a sua atenção concentrada na satisfação dos requisitos de negócio da aplicação. Freqüentemente, o especialista deverá oferecer suporte aos outros membros da equipe a fim de orientá-los quanto a padrões que aumentem a qualidade de serviço da aplicação.

2.2.1.4 Teste

Testador: a principal atribuição deste papel é garantir que a solução gerada esteja livre de defeitos que venham a prejudicar o seu valor de negócio. Para isso deve testar a solução

após a fase de Desenvolvimento e reportar as falhas aos desenvolvedores provendo o contexto no qual o defeito foi encontrado.

Gerente de Testes: assim como o Gerente de Desenvolvimento, o Gerente de Testes é responsável pela coordenação de sua equipe. Ele deve trabalhar conjuntamente com o gerente de projeto para garantir o sucesso da fase de testes.

2.2.1.5 Operações de *Release*

Gerente de *Release*: garante que o produto está pronto para entrar na Fase de Implantação e gerencia a mesma. Para isto, cria planos e cronogramas e certifica os candidatos à versão final.

2.2.1.6 Experiência do Usuário

Arquiteto de Experiência do Usuário: é responsável pelo *design* do produto, gerenciando a concepção e construção da interface do produto, suas funcionalidades, estética e usabilidade. Toma parte na captura dos requisitos a fim de identificar as necessidades do usuário e garantir que estas sejam contempladas na interface. Assim, o Arquiteto de Experiência do Usuário constrói cenários, identifica os perfis de usuários, desenvolve protótipos e conduz testes de usabilidade durante as fases do projeto.

Especialista em educação do usuário: assim como o Arquiteto de Experiência do Usuário, o especialista em educação trabalha para a maximização da performance do usuário do sistema, estando concentrado porém, na produção de documentação técnica direcionada ao mesmo. Manuais de produto, ajudas *on-line*, manuais de operação, manuais de manutenção e manuais de treinamento são exemplos do que pode ser produzido por este papel.

2.2.1.7 Gerência de Produto

Especialista no domínio da aplicação: este papel pode ser preenchido por qualquer um que tenha conhecimento sobre o negócio alvo da solução. Sua principal responsabilidade é transferir seu conhecimento para os outros membros da equipe, principalmente com a finalidade de estabelecer um visão consistente do produto a ser construído.

Sponsor: é responsável por prover o suporte financeiro ao projeto, além de determinar o valor de negócio que o projeto deve atingir e avaliar o andamento do projeto.

Analista de negócio: tem por atribuição a análise e a definição dos requisitos de negócio da aplicação. Define os atores, escreve cenários e serve como interface entre os usuários e clientes e a equipe técnica.

Auditor: é um indivíduo externo ao projeto que tem por função oferecer uma análise independente e objetiva do mesmo. Zela pela qualidade do produto na medida em que realiza

medições que registram variações e não-conformidades nas especificações, planos e definição de processos. Suas medições podem ser utilizadas para avaliar a qualidade do produto e o nível de controle que uma organização detém de seus processos.

Gerente de Produto: constitui o principal responsável pela garantia do valor de negócio do produto. Para tal, deve garantir que os requisitos capturados satisfazem a visão do produto e a realização de testes de aceitação com o usuário. Desta maneira, representa o principal ponto de contato entre o usuário final da solução e o restante da equipe. Adicionalmente, deve ser capaz de mostrar que o produto está alinhado com o planejamento estratégico da organização e que possui um segmento alvo de mercado bem definido.

2.2.2 Fluxos e items de trabalho

Uma importante modificação no MSF 4.0, foi a concretização da metodologia através do estabelecimento de fluxos de trabalho. Estes são grupos de atividades associadas a um ou mais papéis do Modelo de Equipe, cujo objetivo é guiar a execução de tarefas específicas durante a execução do projeto.

Por restrições de espaço, não será promovida uma análise detalhada de todos os fluxos de trabalho do MSF 4.0 (um total de 25), estes podem ser encontrados em sua documentação [Mica]. Será analisado, entretanto, um de seus fluxos, para que possa ser melhor compreendida a estrutura geral dos mesmos. Não obstante, a Tabela 2.3 apresenta a relação que os fluxos de trabalho guardam com as fases do MSF.

Cada fluxo de trabalho é constituído por: uma *visão geral*, que determina o principal objetivo a ser atingido pelo fluxo; *critérios de entrada*, que explicitam dependências em relação a outras atividades, assim como determinam quando o fluxo realizar-se-á; uma seqüência de *atividades* a serem executadas; *critérios de saída*, que deixam claras as pós-condições a serem satisfeitas após o término do fluxo; um conjunto de *papéis envolvidos* que deixa claro os membros da equipe que estarão envolvidos durante o fluxo, e em que nível.

A Figura 2.7 ilustra os diferentes níveis nos quais os membros da equipe podem estar envolvidos no fluxo de trabalho. O grau de participação vai desde os membros que são *responsáveis* e *respondem* a respeito do fluxo, até os que são *consultados* ou meramente *informados* a respeito das atividades.

Não obstante a seqüencia de atividades provida pela descrição do fluxo de trabalho, estas não constituem elementos atômicos, isto é, no MSF 4.0 cada atividade pode ser decomposta em passos concretos para a realização da mesma. Tomando como exemplo a atividade *Selecionar regras e guidelines de análise estática*, notamos uma estrutura bastante semelhante a que encontramos em um fluxo de trabalho. De fato, a análise da Figura 2.8 revela tal semelhança, mostrando que uma atividade possui critérios de entrada e de saída, papéis envolvidos e uma decomposição em elementos, agora atômicos, chamados de sub-atividades.

Além dos fluxos, o MSF 4.0 introduziu os *itens de trabalho* (*work items*), os quais correspondem a artefatos que sofrem alterações durante o processo de desenvolvimento como resultado da evolução natural do projeto. Os sete itens de trabalho cujos modelos (*templates*) são fornecidos pelo MSF 4.0 são: tarefa, solicitação de mudança, risco, revisão, requisito, defeito e ocorrência.

Fluxo de Trabalho	*Fases*
Estabelecer Processo do Projeto	Visão
Capturar Visão do Produto	Visão
Análise	Planejamento
Criar Requisito de Qualidade de Serviço	Planejamento
Criar Arquitetura da Solução	Planejamento
Estabelecer Ambientes	Planejamento
Criar Cenário	Planejamento
Criar Requisitos do Produto	Planejamento
Planejar Iteração	Planejamento
Planejar Projeto	Planejamento
Estabelecer Baseline da Gerência de Configuração	Planejamento
Implementar Tarefa de Desenvolvimento	Construção
Finalizar um Defeito	Construção, Estabilização
Fazer *Build* do Produto	Construção, Estabilização
Gerenciar Solicitações de Mudança	Construção, Estabilização
Consertar Defeito	Construção, Estabilização
Testar Requisito de Qualidade de Serviço	Construção, Estabilização
Testar Cenário	Construção, Estabilização
Verificar Requisito Funcional	Construção, Estabilização
Verificar Requisito Operacional	Construção, Estabilização
Desenvolver Documentação	Implantação
Lançar o Produto	Implantação
Gerenciamento de Riscos	Qualquer
Gerenciar Ocorrência	Qualquer

Tabela 2.3 Relação entre fluxos de trabalho e as fases do MSF responsáveis

Tarefa: é uma indicação da necessidade de realização de trabalho por parte de algum membro da equipe (escrever um caso de teste, desenvolver código a partir de um cenário, etc).

Solicitações de mudança: faz-se necessária quando da modificação de qualquer item que integre um *baseline* do projeto e deve ser analisada pelo comitê de controle de mudança (*change control board*) a fim de ser rejeitada ou aprovada.

Risco: documenta uma condição ou evento provável que pode ter efeitos negativos para o projeto. É parte essencial do processo de gerenciamento do projeto identificar e gerenciar os riscos inerentes ao projeto.

Revisão: documenta os resultados de uma revisão de código ou de modelo. Deve trazer dados a respeito de como o código adequa-se a padrões de corretude de nomes, relevância, extensibilidade, complexidade algorítmica e segurança.

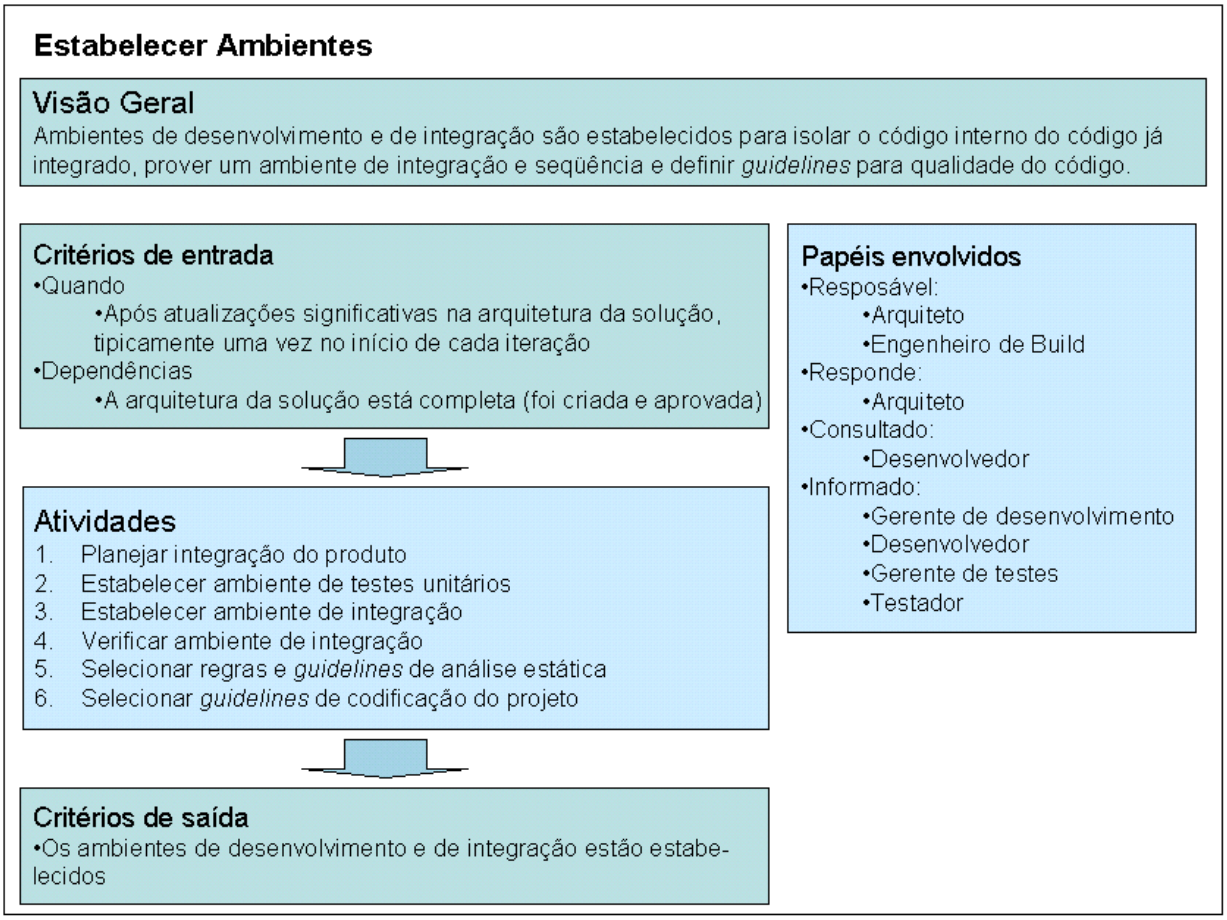

Figura 2.7 Fluxo de trabalho *Estabelecer Ambientes*

Requisito: captura o que o produto precisa ter para satisfazer as necessidades do cliente. Vários são os tipos de requisitos: cenários, qualidade de serviço, funcionais, operacionais e de interface. Este talvez seja o item de trabalho cuja presença é a mais pervasiva durante o ciclo de desenvolvimento, já que é capturado nas fases iniciais e guia as atividades de análise e projeto, codificação e testes.

Defeito: comunica que um problema potencial existe no produto. Sua descrição deve conter em que contexto o defeito foi encontrado, de maneira que este seja facilmente reproduzido, facilitando sua resolução.

Ocorrência: documenta um evento ou situação que tem a possibilidade de bloquear o andamento do projeto. Diferem dos riscos por serem identificadas de forma espontânea em reuniões de equipe. Uma vez identificadas, são geradas tarefas que, quando completadas com sucesso, implicam a finalização da ocorrência.

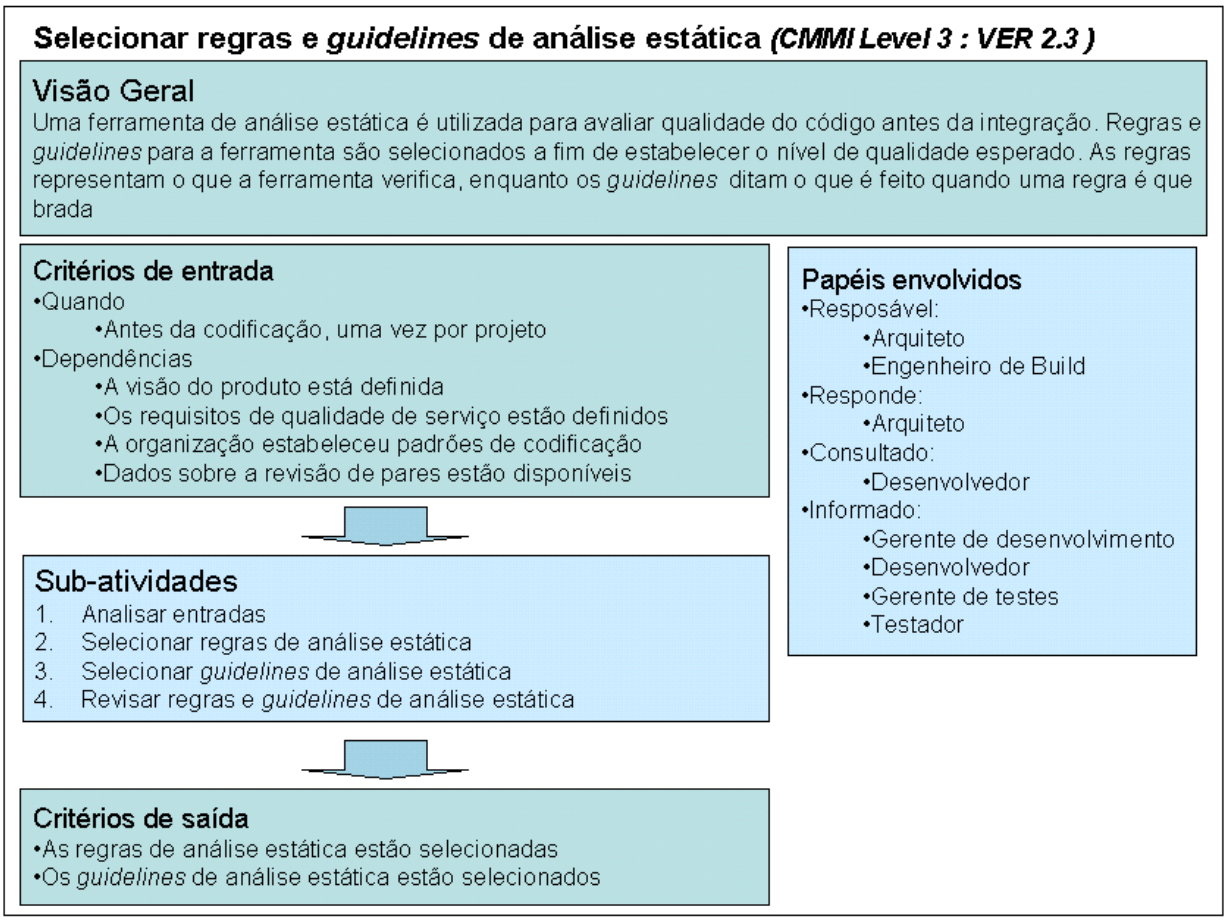

Figura 2.8 Atividade *Selecionar Regras e Guidelines de Análise Estática*

2.2.3 Relação com o CMMI

Conforme explicitado no Capítulo 1 deste trabalho, é crescente a demanda por conformidade das organizações com padrões de melhoria de processos. Como conseqüência desta tendência, e com a evidência dos modelos CMM e CMMI, diversos trabalhos têm realizado análises de processos existentes em relação aos modelos do SEI [MP03, Mel04, Mar03]. Desta maneira, o aparecimento de um processo ou *framework* cuja atenção estivesse voltada para o CMMI não passava de uma questão de tempo.

A variação do MSF 4.0 aqui analisada, o *MSF For CMMI Process Improvement*, tem por objetivo facilitar o processo de uma organização atingir os níveis de maturidade 2 e 3. O MSF 4.0 deixa claro, entretanto, que sua adoção não constitui garantia de avaliação positiva com relação aos referidos níveis. De fato, apenas 17 das 22 áreas de processo (*Process Areas - PA's*) são contempladas pelo *MSF For CMMI Process Improvement*. Para uma melhor compreensão da relação existente entre o CMMI e o MSF 4.0, faz-se necessária uma breve descrição do modelo do SEI. Uma abordagem mais detalhada pode ser encontrada na literatura específica sobre o tema [CKS03].

O modelo de avaliação por estágios do CMMI propõe a classificação das organizações em uma escala de 5 níveis de *maturidade* do processo que estas utilizam[3]. Cada nível deste modelo é um indicativo de como se encontra o processo de desenvolvimento da organização, compreendendo desde processos executados de maneira *ad hoc* àqueles cuja atenção recai sobre a melhoria contínua dos mesmos (Figura 2.9).

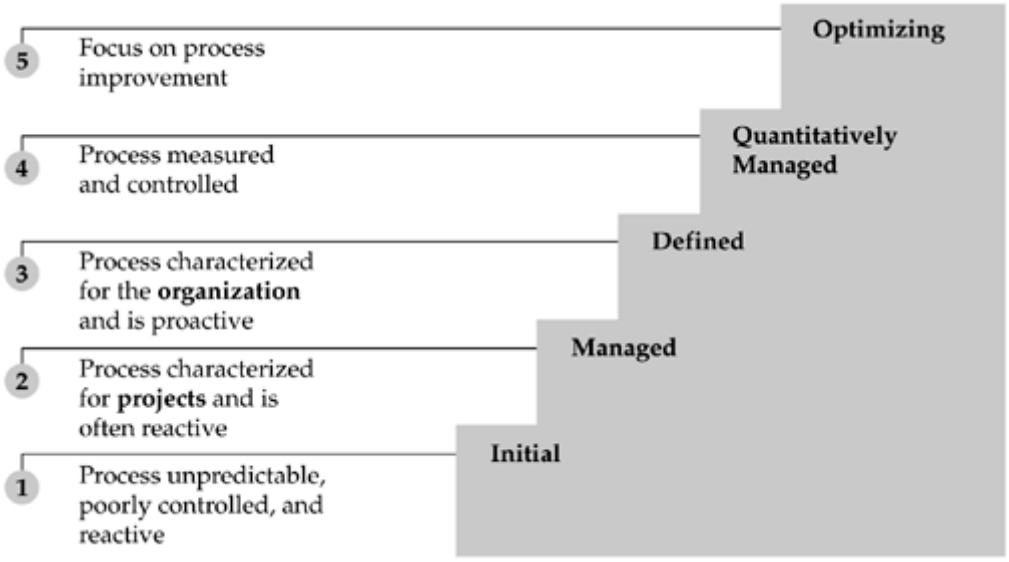

Figura 2.9 Níveis de maturidade no CMMI

A fim de guiar a evolução através do modelo, cada nível de maturidade define áreas de processo que resumem objetivos a serem atingidos pela organização. Uma vez que todas as PA's de um determinado nível são satisfeitas, o processo da organização é avaliado como pertencente a este nível. A Tabela 2.4 mostra as PA's pertencentes a cada nível de maturidade.

O conjunto de objetivos resumidos por uma PA (Figura 2.10) divide-se em dois grupos: genéricos e específicos. Os *objetivos genéricos* estão diretamente ligados a institucionalização do processo e são assim chamados por serem comuns a mais de uma PA. A satisfação de um objetivo genérico se dá através da implantação de *práticas genéricas* que estão agrupadas em *características comuns*. Já os *objetivos específicos* recebem este nome pela relação intrínsica que guardam com cada PA. Este último tipo de objetivo é atingido através de *práticas específicas*.

O suporte do MSF 4.0 ao CMMI dá-se através do alinhamento de seus fluxos de trabalho e atividades com as práticas específicas do CMMI. Quando há uma relação direta entre um fluxo ou atividade do MSF e uma prática específica do CMMI, esta relação é indicada na descrição da atividade ou fluxo. Tome-se como exemplo a atividade *Selecionar regras e guidelines de análise estática* apresentada na Figura 2.8: sua descrição traz a indicação "*CMMI Level 3 :*

[3]Uma outra visão possível do CMMI é o modelo de avaliação *contínua* que classifica áreas de processo em níveis de *capacidade*

Nível de Maturidade	*Sigla*	*Áreas de Processo*
Gerenciado	REQM	Gerenciamento de Requisitos
	PP	Planejamento de Projeto
	PMC	Controle e Monitoramento de Projeto
	SAM	Gerenciamento de Acordos com Fornecedores
	MA	Mensuração e Análise
	PPQA	Garantia de Qualidade do Projeto e do Produto
	CM	Gerenciamento de Configuração
Definido	RD	Desenvolvimento de Requisitos
	TS	Solução Técnica
	PI	Integração de Produto
	VER	Verificação
	VAL	Validação
	OPF	Foco no Processo Organizacional
	OPD	Definição do Processo Organizacional
	OT	Treinamento Organizacional
	IPM	Gerenciamento Integrado de Projeto
	RSKM	Gerenciamento de Riscos
	IT	Composição de Equipes Integradas
	ISM	Gerenciamento Integrado de Fornecedores
	DAR	Análise e Resolução de Decisões
	OEI	Ambiente Organizacional para Integração
Gerenciado Quantitativamente	OPP	Performance do Processo Organizacional
	QPM	Gerenciamento Quantitativo de Projeto
Otimizado	OID	Inovação e Implantação Organizacional
	CAR	Análise e Resolução Causal

Tabela 2.4 Níveis de maturidade, áreas de processo e suas respectivas siglas (em inglês)

VER 2.3" significando que esta atividade tem relação direta com a prática específica 2.3[4] da área chave *Verificação* (VER) pertencente ao nível 3 do modelo. De fato, a prática específica em questão intitula-se *Analisar dados da revisão de pares* e a existência de tais dados faz parte dos critérios de entrada da atividade do MSF, sendo a própria seleção dos *guidelines* e regras de análise estática baseada nestes. A relação que que um elemento do MSF (atividade ou fluxo) estabelece com uma prática específica do CMMI não é, entretanto, única. A atividade *Selecionar guidelines de codificação do projeto* também referencia, por exemplo, a referida prática especifica. Assim, pode ser necessária mais de uma atividade e/ou fluxo para atingir alguma prática específica do CMMI.

[4] A notação do tipo *n.m* indica que esta prática está atrelada ao n-ésimo objetivo específico da área de processo, sendo a m-ésima prática executada para atingi-lo

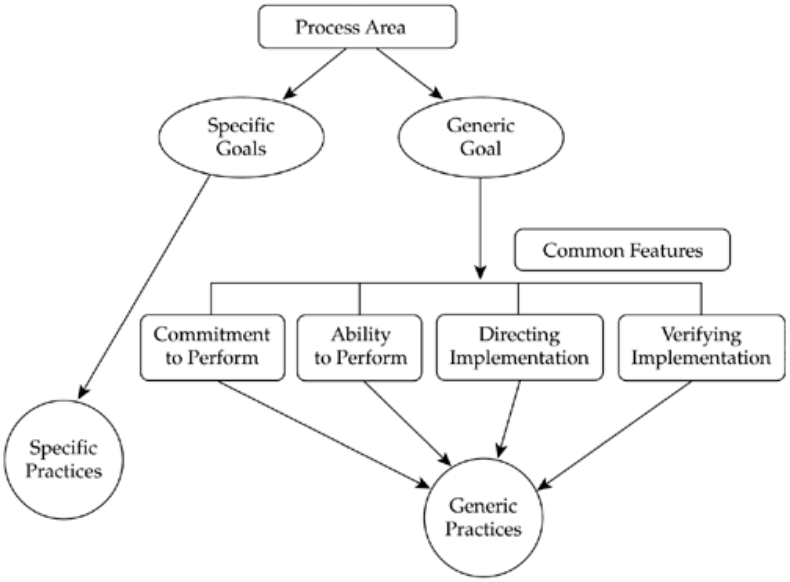

Figura 2.10 Estrutura interna de uma área de processo

2.3 Comparando as duas versões

A partir da visão geral apresentada neste capítulo, fica claro que a concretização do MSF foi principal mudança entre as versões 3.0 e 4.0. O estabelecimento de papéis bem definidos, a instanciação das boas práticas em fluxos de trabalho e atividades, além da criação de itens de trabalho, corroboraram para a referida concretização. Ainda houve a adição do grupo de papéis de arquitetura, provavelmente oriunda do reconhecimento de que, em projetos de grande porte, é mister não sobrecarregar gerência de programa com atividades demasiado técnicas, permitindo que esta se dedique integralmente a gerência do projeto.

Os esforços em direção a concretização do *framework* são justificados pelo desejo de conformidade com o CMMI, uma vez que seria virtualmente impossível oferecer qualquer suporte às práticas exigidas pelo modelo, sem a criação de algo semelhante às atividades e aos fluxos de trabalho.

Muito embora tenha havido um perda de flexibilidade em virtude da concretização do *framework*, o MSF 4.0 continua, tanto na sua vertente "ágil" quanto na "formal", fiel à filosofia de sua versão anterior. O MSF 4.0 abre espaço, inclusive, para a adaptação de seus fluxos de trabalhos à realidade do projeto a ser executado, estando assim ainda de acordo, mesmo que em menor grau, com o princípio de que todo projeto de software é único.

CAPÍTULO 3

Pro.NET

Este capítulo tem por objetivo prover uma visão geral da Pro.NET afim de facilitar a compreensão da análise comparativa apresentada no Capítulo 5. O conteúdo deste capítulo foi parcialmente baseado na documentação da metodologia Pro.NET [CTXb].

3.1 Introdução a metodologia

3.1.1 Motivação

O mercado de desenvolvimento de software, para ser competitivo, necessita cada vez mais de um conjunto de ferramentas (metodologias, linguagens, ambientes de desenvolvimento, etc) que o faça produzir software, atendendo a três importantes variáveis: baixo custo, alta qualidade e o mínimo de tempo. Aliado a esse contexto, o mercado de desenvolvimento de software tem uma demanda cada vez mais crescente de dominar e utilizar tecnologias e ferramentas, tais como MSF, .NET, C#, *MS Project*, *Visual Studio*, *eXtreme Programming*, RUP, PMBOK[1] do PMI (*Project Management Institute*), etc.

O *Microsoft Solutions Framework* (MSF) foi criado em 1994 pela *Microsoft Consulting Services* (MCS), com o objetivo de aumentar sua taxa de sucesso no desenvolvimento de projetos em soluções de tecnologia. O MSF é baseado em boas práticas utilizadas em projetos bem sucedidos de soluções de tecnologia. Essa necessidade deu-se pela razão da grande quantidade de projetos que não obtiveram resultados satisfatórios. O MSF é um conjunto de práticas para planejar, construir e implantar soluções de tecnologia da informação. Ao contrário de uma metodologia, MSF é um framework flexível e escalável que atende às necessidades de uma organização ou time de qualquer tamanho. O MSF contém um conjunto de princípios, modelos e disciplinas para gerenciar pessoas, processos e elementos tecnológicos que a maioria dos projetos enfrentam. Vale destacar sua reputação de ser o framework utilizado pela maior empresa de desenvolvimento de software do mundo.

Sendo a Plataforma .NET uma tecnologia idealizada pela *Microsoft*, é interessante que os usuários da plataforma utilizem os princípios do MSF, pois a própria plataforma foi desenvolvida utilizando esse *framework*. Neste contexto, vale ressaltar que o MSF utiliza a "linguagem Microsoft". Isso quer dizer que alguns termos utilizados na documentação da Plataforma .NET também são utilizados pelo MSF.

[1] *Project Management Body of Knowledge*

3.1.2 Objetivo

O objetivo da Pro.NET é estabelecer uma metodologia que viabilize baixo custo, alta qualidade e mínimo de tempo no desenvolvimento de projetos que utilizem a Plataforma .NET.

3.1.3 Tecnologias-alvo

A Pro.NET é uma metodologia direcionada para a Plataforma .NET e desenvolvida com base no MSF para estabelecer um processo de desenvolvimento altamente produtivo. O fato do MSF ser um *framework* flexível e escalável permitiu sua extensão para a criação da metodologia Pro.NET.

Ao contrário de grande parte dos processos de desenvolvimento existentes atualmente no mercado, a Pro.NET direciona a tecnologia a ser utilizada. O desenvolvimento da metodologia considerou a necessidade de um grupo específico de projetos de tecnologia da informação: os que utilizarão como base tecnológica a Plataforma .NET.

Apesar dessa característica, a Pro.NET pode ser utilizada em qualquer projeto de desenvolvimento de software. A maior parte da customização para a tecnologia-alvo está nos artefatos e guias, existindo uma independência nos processos e fluxos das atividades.

3.1.4 O que a Pro.NET não cobre

A Pro.NET não aborda os seguintes tópicos:

- Processo de operação de um produto depois da implantação;
- Gestão de pessoas: contratação, acompanhamento etc;
- Gestão de orçamentos: definição, alocação, etc;
- Gestão de contratos com fornecedores, clientes e empresas subcontratadas;
- Processo de instanciação da Pro.NET para a realidade de uma organização ou projeto.

3.2 Estrutura da Pro.NET

Assim como no MSF, são dois os principais componentes metodologia Pro.NET: o *Modelo de Equipe* e o *Modelo de Processo*.

O *Modelo de Equipe* descreve como estruturar a equipe e suas responsabilidades para atingir sucesso do projeto. O modelo utiliza o conceito de papéis para dividir as responsabilidades pelos membros do projeto. Além disso, apresenta guias de como a equipe deve agir e se relacionar. A seção *Modelo de Equipe* apresenta maiores informações sobre esse tópico.

O *Modelo de Processo* descreve processos e métodos a serem realizados pela equipe do projeto para planejar, desenvolver e implantar soluções de tecnologia da informação.

O modelo de ciclo de vida de desenvolvimento de software da Pro.NET combina dois outros: cascata e espiral. Esse modelo define fases a serem seguidas em seqüência, como o

modelo cascata, e que devem ser executadas várias vezes no mesmo projeto, definindo as iterações do modelo espiral. O final de cada fase é um marco e durante cada fase ocorrem marcos internos.

Cada iteração possui sua fase de *Visão, Planejamento, Desenvolvimento, Estabilização* e *Implantação*. O desenvolvimento deve seguir iterações de curto tempo, para acomodar mudanças dos requisitos da solução. A idéia é desenvolver o projeto de modo iterativo e incremental. A seção *Modelo de Processo* apresenta maiores informações sobre esse tópico.

3.3 Modelo de Equipe

O Modelo de Equipe da Pro.NET é inteiramente baseado no Modelo de Equipe do MSF 3.0, tendo fundamento nos seguintes princípios:

Responsabilidade compartilhada: cada membro da equipe é ciente das suas atribuições e divide a responsabilidade pelo sucesso do projeto.

Divisão do trabalho cada membro da equipe é ciente das atividades que deve realizar e pode requisitar atividades a outros membros. Cada membro deve estar preparado para receber solicitações da equipe e deve atendê-las conforme previamente acordado, comunicando quando o acordo estabelecido estiver em risco por qualquer motivo.

Visão Única: a equipe deve compartilhar uma mesma visão, entendendo claramente quais são os objetivos do projeto e do processo de desenvolvimento.

Foco no cliente: a equipe deve estar atenta ao que é realmente importante para o negócio do cliente. A tecnologia é utilizada como meio e não como foco.

Disposição a mudanças: todos os membros da equipe devem estar cientes de que mudanças ocorrerão e modificarão seu trabalho. Deve ser estimulada a prática de revisões e sugestões no trabalho realizado por outros membros.

Ampla Comunicação: a comunicação entre os integrantes deve ser estimulada e coordenada. Isso reduz os enganos provenientes da falta de informação. Se precisarem existir informações secretas, a equipe deve estar ciente que existe o sigilo e que ele contribui para o sucesso do projeto.

Assim como no MSF, a equipe é organizada como uma equipe de colaboradores, isto é, cada papel da equipe é único, valioso e igualmente valorizado. Esse conceito está representado na Figura 3.1. Como ilustra a figura, os papéis do Modelo de Equipe são: *Gerente de Produto*, *Desenvolvedor*, *Analista de Testes*, *Gerente de Release*, *Analista de Usuário* e *Gerente de Projeto*.

Cada papel é responsável por um conjunto de responsabilidades de determinadas áreas de funcionalidade que atingem um objetivo específico. As responsabilidades de cada papel do Modelo de Equipe são resumidas na Tabela 3.1.

Papel	*Responsabilidades*
Gerente de Produto	Age como advogado do cliente Gerencia a visão/escopo da solução Desenvolve e mantém análise de viabilidade Gerencia as expectativas do cliente Gerencia as decisões de escopo vs. cronograma vs. recursos Gerencia *marketing* e relações públicas Desenvolve, mantém e executa o plano de comunicação
Gerente de Projeto	Desenvolve processo de desenvolvimento para a entrega da solução conforme o planejado Gerencia especificação da solução Facilita comunicação e negociação com a equipe Reporta status do projeto Participa de decisões críticas do projeto Desenvolve, mantém e executa o Plano de Projeto e cronograma Gerencia todo processo de riscos do projeto
Desenvolvedor	Especifica requisitos de instalação Estima tempo e esforço para desenvolver cada caso de uso Constrói e supervisiona a implementação de casos de uso Prepara aplicação para implantação Oferece consultoria em tecnologia para a equipe
Analista de testes	Garante que todos os defeitos estão identificados Desenvolve planejamento de testes Conduz testes
Analista de Usuário	Age como advogado da equipe Gerencia definição de requisitos do usuário Cria e desenvolve sistemas de suporte a produtividade Gerencia decisões de usabilidade vs. aumento da produtividade do usuário Especifica funcionalidades e mecanismos de educação dos usuários Desenvolve e fornece treinamento de usuários
Gerente de *Release*	Age como advogado para operação, suporte e canais de distribuição Gerencia implantação da solução Gerencia decisões para facilitar a administração do sistema e suporte ao usuário Gerencia decisões de usabilidade vs. aumento da produtividade do usuário Gerencia relacionamento entre operação, suporte e canais de distribuição

Tabela 3.1 Papéis e responsabilidades na Pro.NET

Figura 3.1 Modelo de Equipe da Pro.NET

A Pro.NET não define quem é o gerente do trabalho da equipe, sob o ponto de vista administrativo. Isso condiz com as orientações do MSF, que afirma que cada papel é igualmente valorizado. Inclusive, em muitos casos, a equipe do projeto inclui membros de diferentes organizações, que estão subordinados a gerentes diferentes. No entanto, existem ocasiões em que a equipe não entra em consenso sobre um problema. Nessas situações, o Gerente de Projeto assume o controle e decide o rumo a seguir, já que esse papel deve garantir a entrega da solução dentro das limitações (incluindo tempo) do projeto. Depois de resolvido o problema, a equipe volta a dividir as responsabilidades e ser igualmente valorizada.

Para que os objetivos do projeto sejam alcançados com sucesso, é necessário existir comunicação com os *stakeholders* externos. É importante definir uma visão clara de quem será o responsável por facilitar essa comunicação. O *Gerente de Produto* se comunica com o cliente, o *Analista de Usuário* se comunica com os usuários, o *Gerente de Projeto* se comunica com o financiador do projeto e com líderes técnicos do cliente e o *Gerente de Relase* se comunica com técnicos de operação e suporte do cliente. O *Desenvolvedor* e *Analista de Testes* devem ser isolados da comunicação externa, que é executada por outros papéis. Isso não quer dizer que, por exemplo, um *Desenvolvedor* não poderá se comunicar com um cliente, no entanto, uma comunicação externa formal não é assumida por esse papel.

Cada papel pode ser desempenhado por várias pessoas. Nesse caso, a equipe que desempenha o papel é chamada *equipe de um papel* e deve existir uma hierarquia para determinar quem é o líder daquele papel. Por exemplo, vários Desenvolvedores podem ser subordinados a um Desenvolvedor. Essa hierarquia também pode ser definida para áreas de funcionalidade.

	G. de Prod.	*G. de Proj.*	*Desenv.*	*A. de Testes*	*A. de Usu.*	*G. de Rel.*
G. de Prod.	X	NR	NR	R	R	P
G. de Proj.	NR	X	NR	P	P	R
Desenv.	NR	NR	X	NR	NR	NR
A. de Testes	R	P	NR	X	R	R
A. de Usu.	R	P	NR	R	X	P
G. de Rel.	P	R	NR	R	P	X

Tabela 3.2 Possíveis combinações de papéis na Pro.NET

Outro modo de se dividir a equipe é formar pequenas equipes multidisciplinares. Essa organização forma uma equipe de equipes, sendo estas equipes chamadas de *equipes multidisciplinares* e com uma delas sendo a líder. Essas equipes menores não precisam ter todos os seis papéis e trabalharão em paralelo.

Numa equipe pequena, ocorre o caso em que uma mesma pessoa pode desempenhar vários papéis. Neste caso, deve-se ter cuidado para que uma pessoa não seja sobrecarregada de atribuições. A Tabela 3.2 ilustra o risco de se combinar papéis. A legenda utilizada é a seguinte: R (recomendável), P (possível), NR (não recomendável), X (não faz sentido, pois são os mesmos papéis).

Analisando-se a Tabela 3.2, percebe-se que algumas combinações não são recomendadas. A primeira é que membros do papel Desenvolvedor nunca devem exercer outro papel, já que as atividades do Desenvolvedor são de prioridade máxima para o projeto e o seu atraso tem grande impacto no cronograma geral do projeto. Outra combinação não recomendada é do Gerente de Projeto com o Gerente de Produto, pois tais papéis representam interesses conflitantes. Enquanto o Gerente de Produto defende o interesse do cliente, o Gerente de Projeto defende o interesse da equipe.

3.4 Modelo de Processo

O Modelo de Processo corresponde a todo trabalho de desenvolver a aplicação. Este modelo descreve os processos e métodos a serem realizados pela equipe do projeto para planejar, construir e implantar soluções de tecnologia da informação.

O Modelo de Processo da Pro.NET é baseado no Process Model do MSF. A seção *Pro.NET vs. MSF* apresenta maiores informações sobre as diferenças entre esses dois modelos. O modelo de ciclo de vida de desenvolvimento de software da Pro.NET é dividido em iterações. A Figura 3.2 ilustra como o desenvolvimento das iterações gera versões da solução. Os pontos da figura representam o final das fases a serem seguidas em cada iteração.

A decisão da ordem em que os casos de uso devem ser implementados é baseada na importância do caso de uso para o cliente e os usuários, nos seus riscos associados e na sua importância para o estabelecimento de uma arquitetura estável. É através dessa decisão que são estabelecidos o escopo de cada iteração e a ordem de implementação dos casos de uso dentro de cada versão.

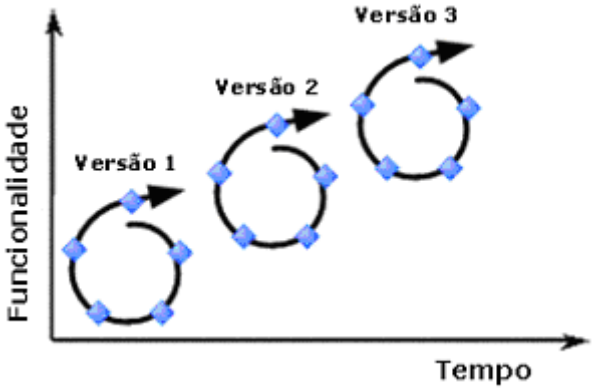

Figura 3.2 Geração de versões das soluções na Pro.NET

Cada iteração possui as seguintes fases: *Visão*, *Planejamento*, *Desenvolvimento*, *Estabilização* e *Implantação*. O final de cada fase é caracterizado por um marco conforme apresenta a Figura 3.3. Embora esta figura apresente um círculo dividido em cinco parte iguais, as fases não ocorrem necessariamente durante tempos iguais. A duração de cada fase depende da natureza do projeto.

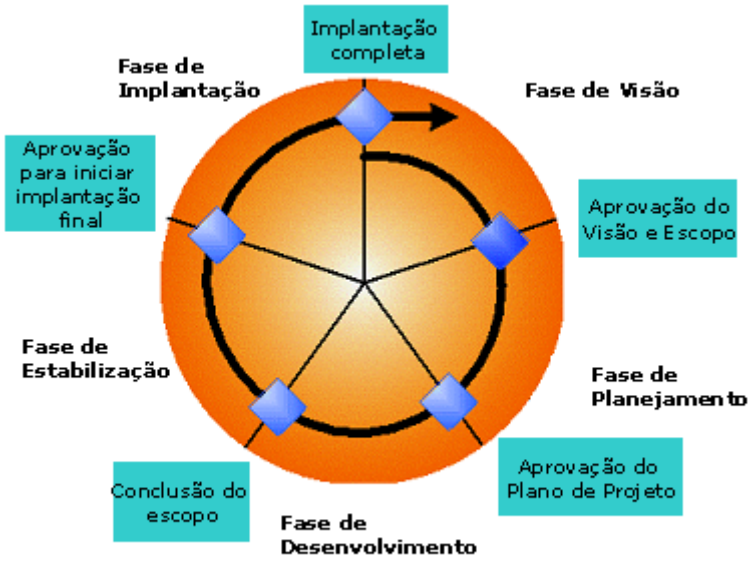

Figura 3.3 Fases e marcos de cada iteração na Pro.NET

A fase de *Visão* é a primeira da iteração. É nesse período que o cliente e o time definem, em alto nível, os objetivos do projeto. Esta fase pode ser vista como um estágio inicial do planejamento. A aprovação do documento de *visão e escopo* é o marco ao final dessa fase. Com ele, o time e o cliente estabelecem uma visão única do que será o projeto.

Durante a fase de *Planejamento*, a equipe está trabalhando na construção do Documento de Especificação Funcional da solução e no planejamento geral do projeto. O planejamento do projeto deve ponderar três fatores principais: escopo da solução, recursos do projeto e datas acordadas com o cliente. A fase termina com o marco aprovação do *plano de projeto*, indicando que os *stakeholders* concordam com o que e como será construído e quando a solução estará disponível aos usuários.

Na fase de *Desenvolvimento*, a equipe constrói o código da aplicação, desenvolve a infra-estrutura e escreve uma parte da documentação. No decorrer da fase, são gerados e testados alguns *releases* internos. O marco do final da fase caracteriza que o escopo total da aplicação já foi construído, mas ainda podem existir defeitos.

É na fase de *Estabilização* que os defeitos ainda existentes da aplicação são corrigidos. A equipe está focada em testar, priorizar e corrigir os defeitos encontrados, gerando novos builds, e em preparar a solução para a implantação no ambiente de produção. Um dos builds gerados será utilizado para implantação final da solução e marca o final desta fase.

A última fase da iteração é a *Implantação*. É nela que a implantação final é realizada e a responsabilidade pela solução passa para o time de operação e suporte. Deve ser realizado um postmortem, revisão geral das lições aprendidas durante o projeto, com objetivo de enriquecer a base de conhecimentos da organização. A implantação final da solução é o último marco da iteração e caracteriza o início da próxima iteração.

A idéia do modelo de ciclo de vida da Pro.NET é baseada no desenvolvimento iterativo e incremental. Em cada marco, ocorre validação do trabalho já executado, promovendo a comunicação entre os *stakeholders*. Além disso, a solução pode ser validada com o cliente ao final de cada iteração e é possível acomodar mudanças dos requisitos nas próximas iterações.

Além dos marcos já citados, podem existir, como no MSF, marcos internos a uma fase. Esses marcos são pontos de sincronização e revisão do trabalho realizado, mas não são necessariamente validados com o cliente e usuários.

É possível estruturar a metodologia através de dois diferentes aspectos: um aspecto atemporal e um aspecto temporal. O primeiro aspecto estrutura o conhecimento da metodologia de modo indiferente ao fluxo do tempo pelo qual evolui o projeto, através da divisão do mesmo em disciplinas, macro-atividades, atividades, artefatos e responsáveis. Já o aspecto temporal, por outro lado, estrutura o conhecimento da metodologia ao longo do tempo, através da divisão do projeto em fases e iterações.

As disciplinas servem para agrupar um conjunto de atividades relacionadas a uma mesma área de conhecimento. Existem dois tipos de disciplinas, as de processo e as de suporte.

As disciplinas de processo estão relacionadas ao trabalho de desenvolver a solução. São elas: *Requisitos*, *Análise & Projeto*, *Implementação*, *Testes e Implantação*.

A disciplina de *Requisitos* busca entender a estrutura e a dinâmica da organização, os problemas correntes, identificar melhorias de processos, derivar requisitos da organização e da solução de modo a garantir que os clientes, os usuários finais e a equipe de desenvolvimento tenham um entendimento comum da solução a ser desenvolvida e de como esta solução atende aos objetivos da organização. O uso de protótipos é estimulado para aprimorar a especificação.

Análise & Projeto é a disciplina que possibilita a transformação dos requisitos em um projeto da aplicação, sugerindo uma arquitetura robusta e uma solução que atenda aos requisitos

de implantação.

A disciplina de *Implementação* produz o código em várias camadas, implementa as funções e estruturas de dados, testa o código desenvolvido como unidade e realiza inspeção nos artefatos criados.

A disciplina de *Testes* verifica a integração de partes de código implementadas separadamente, se todos os requisitos foram corretamente implementados e identifica e garante que defeitos serão resolvidos antes da implantação da solução.

A disciplina de *Implantação* disponibiliza versões da solução para os usuários e realiza treinamentos.

As disciplinas de suporte são formadas por um conjunto de atividades de apoio que possibilitam a execução de disciplinas de processo. São elas: *Planejamento & Gerenciamento, Riscos* e *Ambiente e Gerencia de Configuração*.

A disciplina de *Planejamento & Gerenciamento* abrange os principais aspectos do planejamento e gerenciamento de projetos em um conjunto de atividades bem definidas. O planejamento agrupa vários planos criados (sendo alguns deles em outras disciplinas) em um plano integrado considerando uma data fixa para entrega da solução e o uso de *buffer time*, para acomodar futuros problemas ao longo do desenvolvimento. O gerenciamento de projeto deve assegurar que as atividades programadas estão sendo executadas, tratar mudanças e garantir a qualidade dos artefatos produzidos e dos processos executados. Ao final do projeto, devem ser capturadas as lições aprendidas.

A disciplina de *Riscos* descreve uma maneira pró-ativa de lidar com incerteza e a considera nas decisões no ciclo de vida do projeto. A disciplina inclui identificação, priorização, planejamento, monitoramento e controle dos riscos.

Ao longo do projeto, devem ser capturadas as lições aprendidas com os riscos. A disciplina Ambiente e *Gerencia de Configuração* descreve como tratar a evolução dos artefatos produzidos, a geração de builds de uma aplicação sendo desenvolvida e o controle das solicitações de mudanças na solução. Uma prática sugerida é a integração contínua do código da aplicação. Outro objetivo é planejar e gerenciar o ciclo de vida dos ambientes e dos recursos de hardware e software do projeto durante o progresso da solução.

A Figura 3.4 relaciona o aspecto atemporal e temporal da Pro.NET. As fases estão na primeira linha e as disciplinas na primeira coluna. As células centrais representam quais são as macro-atividades a serem realizadas naquelas fases e disciplinas.

Macro-atividades contêm um conjunto de atividades que são executadas conjuntamente para a realização de um objetivo. Elas fornecem um nível mais elevado de abstração na descrição do trabalho a ser realizado. Para cada macro-atividade, a metodologia descreve o fluxo de execução de suas atividades.

A Figura 3.4 apresenta cada macro-atividade associada a uma disciplina e várias fases. A barra de cada macro-atividade representa o período do projeto em que existirá esforço para executá-la.

A ordem de execução das macro-atividades durante as fases e das atividades em uma macro-atividade é apenas um direcionamento, podendo ser alterado de acordo com a necessidade do projeto.

As Figuras 3.5 e 3.6 apresentam exemplos de uma macro-atividade e uma atividade respec-

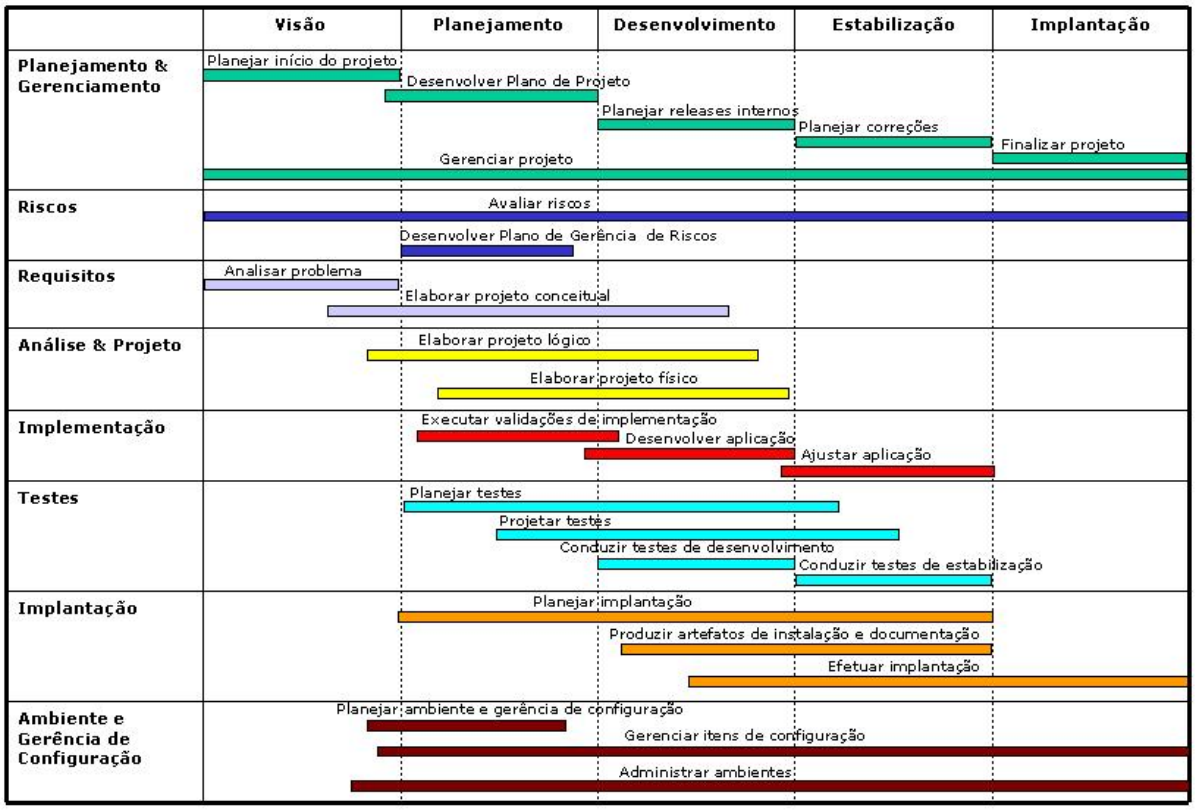

Figura 3.4 Realizações das macro-atividades na Pro.NET

tivamente.

Cada atividade possui um objetivo, um responsável, passos, artefatos de entrada e saída e, possivelmente, guias. O objetivo indica o que a realização daquela atividade pretende atingir dentro do contexto de desenvolvimento do projeto. O responsável é um membro do Modelo de Equipe que irá liderar a execução da atividade, respondendo pelo seu sucesso. Os passos são a divisão do trabalho a ser realizado na atividade em unidades menores. Os artefatos de entrada são necessários à execução da atividade, que gera artefatos de saída. Os guias contêm um conjunto de orientações para facilitar a execução da atividade.

A metodologia fornece também guias e templates para artefatos. Os guias apresentam boas práticas no contexto de determinadas atividades e templates auxiliam na geração dos artefatos durante a execução de um projeto. Tais artefatos devem ser permanentemente atualizados, refletindo as mudanças e o próprio amadurecimento da solução.

3.5 Pro.NET vs. MSF 3.0

A Pro.NET é objetiva, simples e concreta. É objetiva, pois oferece a visão de quais atividades devem ser realizadas em um determinado momento do desenvolvimento do projeto, como

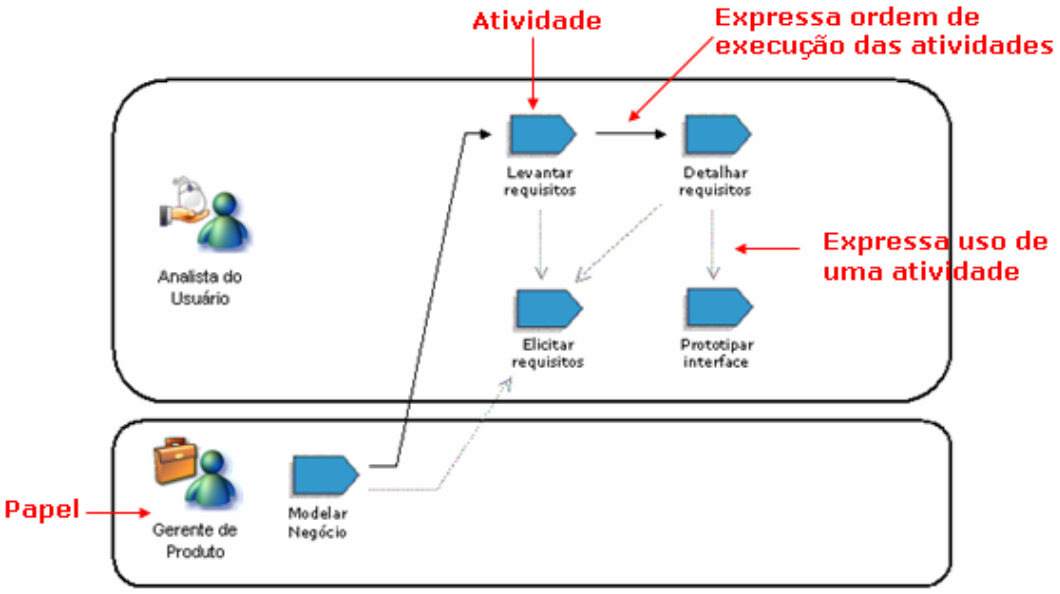

Figura 3.5 Macro-atividade *Elaborar projeto conceitual*

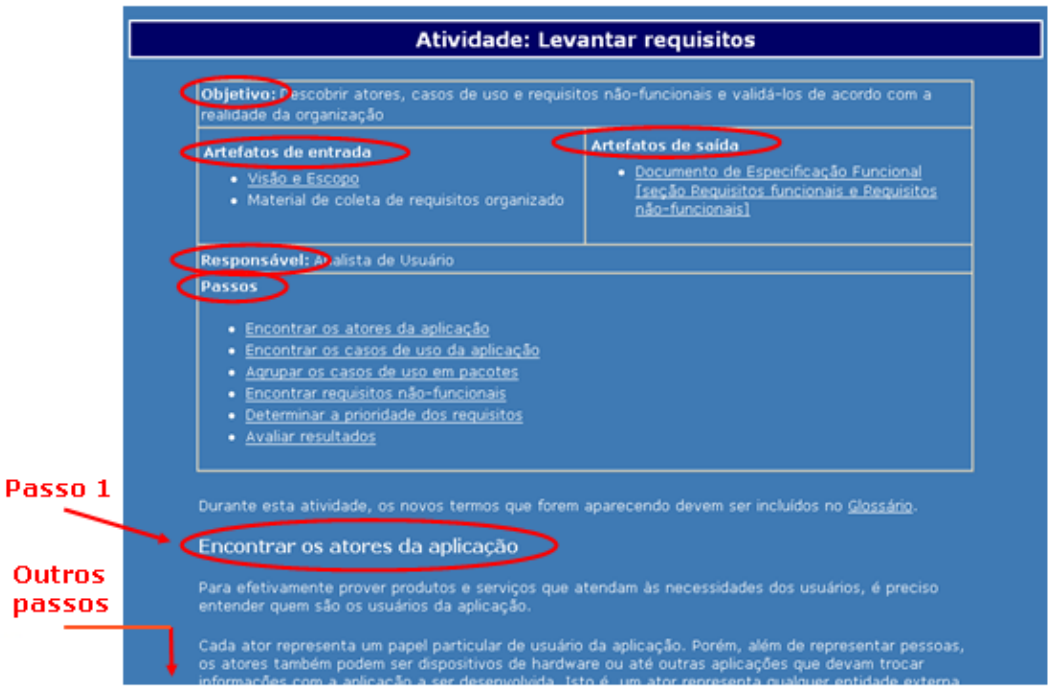

Figura 3.6 Atividade *Levantar requisitos*

explicado na seção *Modelo de Processo*. Além disso, através da definição de uma tecnologia-alvo, grande parte dos artefatos da metodologia foi desenvolvida especificamente para essa tecnologia, o que torna a Pro.NET simples. Sendo assim, seus processos e artefatos são bem próximos dos que serão realmente utilizados em um projeto que utilize a Plataforma .NET. Não será necessário interpretar processos e métodos genéricos para a realidade do projeto, pois isso foi justamente o que a equipe de desenvolvimento da Pro.NET já fez.

A Pro.NET oferece uma concretização das boas práticas definidas pelo MSF. Isso quer dizer que a Pro.NET preserva o conteúdo MSF, agregando a este descrições de como executá-los e acrescentando outros processos e métodos. Desse modo, enquanto o MSF diz o que fazer, a Pro.NET diz como. Para isso, a Pro.NET utiliza o mesmo modelo de ciclo de vida do MSF: iterativo com marcos.

No entanto, a Pro.NET não é somente uma concretização das boas práticas do MSF. O Modelo de Processo da nova metodologia também possui os seguintes diferenciais, se comparado ao MSF: direcionamento para a Plataforma .NET, novos processos, novos guias e templates para artefatos e maior estruturação.

O diferencial direcionamento para a Plataforma .NET já foi comentado quando foi apresentada a tecnologia-alvo. Sendo o MSF um framework para construir soluções de tecnologia da informação, ele não pode considerar os detalhes de uma plataforma específica. Os benefícios de se restringir a tecnologia-alvo de uma metodologia também já foram citados anteriormente. Um exemplo desse direcionamento é a indicação de ferramentas e disponibilização de alguns programas específicos para aplicações .NET.

Não se limitando a concretizar o MSF, os novos processos da Pro.NET são também são provenientes de outras metodologias, como RUP (Rational Unified Process), XP (eXtreme Programing) e PMBOK. Por exemplo, a Pro.NET define um processo para a gerência de mudanças, enquanto que no MSF apenas cita a existência de artefatos de *Solicitação de mudanças* e indica que deve existir um processo para realizar mudanças.

A Pro.NET também possui novos guias e templates para artefatos. O guia mais importante produzido pela Pro.NET é o que especifica o modelo físico do Modelo de Aplicação. Existem também novos templates e adaptação de alguns templates utilizados no MSF para a tecnologia-alvo.

O MSF divide suas boas práticas em disciplinas e fases. Não fica claro em que fases as boas práticas de uma disciplina do MSF devem ser realizadas. A Pro.NET define uma maior estruturação do conhecimento, através da apresentação do aspecto temporal e atemporal. A maior estruturação da Pro.NET facilita o entendimento da metodologia e a execução de seus processos. Esta estrutura está amplamente descrita na seção do *Modelo de Processo*.

Enquanto o MSF 3.0 possui três disciplinas, a Pro.NET possui oito. Somente a disciplina chamada *Gerenciamento de Conhecimento* não é contemplada pela Pro.NET. Ela detalha um modo de gerenciar conhecimento e habilidades da equipe necessárias para planejar e construir soluções de sucesso.

As comparações realizadas até aqui se referiam ao Modelo de Processo da Pro.NET. O Modelo de Equipe da Pro.NET é equivalente ao do MSF 3.0.

CAPÍTULO 4

MSF 4.0 vs. Pro.NET

Este capítulo visa estabelecer uma análise comparativa entre o MSF 4.0 e a Pro.NET. Como a busca por adequação da Pro.NET em relação ao MSF 4.0 é um dos objetivos deste trabalho, a presente análise foi concebida de forma a tentar alinhar os elementos que compõe a Pro.NET aos do MSF 4.0. Assim, os elementos do MSF 4.0 são sempre tomados como base nas comparações, procurando-se, em seguida, identificar o suporte que a Pro.NET oferece aos mesmos. Para cada elemento comparado, o nível de satisfação que a Pro.NET apresentou em relação ao MSF 4.0 foi então classificado através da seguinte escala: *alto*, quando o elemento do MSF 4.0 é suportado pela Pro.NET, ou mesmo quando este é parcialmente suportado, mas o principal objetivo do elemento não é prejudicado; *médio*, quando o elemento é parcialmente suportado e há um prejuizo em seus objetivos; *baixo*, quando a Pro.NET oferece um suporte muito escasso; *inexistente*, quando não há elemento correspondente na Pro.NET.

A presente análise abrange o modelo de equipe e os fluxos de trabalho do MSF 4.0.

4.1 Modelo de equipe

A Pro.NET apenas ofereceu uma alternativa de concretização para os grupos de papéis apresentados na versão 3.0 do *framework*, podendo ser considerado completamente baseada no mesmo. A alternativa de concretização adotada foi, conforme exposto no Capítulo 3, a de instanciar, em um único papel, cada grupo presente no MSF 3.0.

O MSF 4.0 optou, entretanto, por dividir as responsabilidades de cada grupo entre mais de um papel, além de adicionar um novo grupo, o de *Arquitetura*. Não obstante, quase a totalidade das responsabilidades de cada grupo continuou constante, de maneira que o modelo de equipe da Pro.NET satisfaz o do MSF 4.0, principalmente se for considerado que a Pro.NET não faz restrições quanto ao número de membros da equipe que podem desempenhar um mesmo papel. Assim, o mapeamento entre os papéis presentes na MSF 4.0 e no Pro.NET é direto, a exceção dos que trazem consigo novas responsabilidades (Tabela 4.1).

Como exemplo de papéis que trouxeram consigo novas responsabilidades tem-se o IPM Officer. A gerência dos múltiplos projetos executados por uma organização é a principal atribuição deste papel, e sendo a Pro.NET uma metodologia com foco voltado para a execução de um projeto em particular, é natural que não haja correspondência entre a gerência superior da organização e qualquer papel do MSF 4.0.

Os papéis *Sponsor* e Auditor representam membros externos ao projeto, e apesar de não possuírem correspondência direta com quaisquer papéis do modelo de equipe da Pro.NET, poderiam ser encaixados dentro do conjunto de *stakeholders* envolvidos no projeto.

Papel no MSF 4.0	*Papel na Pro.NET*	*Nível de Satisfação*
Gerente de Projeto	Gerente de Projeto	Alto
IPM Officer	NA	Inexistente
Arquiteto	Desenvolvedor	Médio
Desenvolvedor	Desenvolvedor	Alto
Gerente de Desenvolvimento	Desenvolvedor	Alto
Engenheiro de Build	Desenvolvedor	Alto
Especialista em Qualidade de Serviço	Desenvolvedor	Alto
Testador	Analista de Testes	Alto
Gerente de Testes	Analista de Testes	Alto
Gerente de *Release*	Gerente de *Release*	Alto
Arquiteto de Experiência do Usuário	Analista de Usuário	Alto
Especialista em Educação do Usuário	Analista de Usuário	Alto
Especialista no Domínio da Aplicação	Gerente de Produto	Alto
Sponsor	NA	Inexistente
Analista de Negócio	Gerente de Produto	Alto
Auditor	NA	Inexistente
Gerente de Produto	Gerente de Produto	Alto

Tabela 4.1 Relação entre papéis do MSF 4.0 e da Pro.NET

A última das não-conformidades observadas entre os dois modelos de equipe é a atribuição das responsabilidades arquiteturais ao desenvolvedor. Dado o impacto do projeto da arquitetura no restante do desenvolvimento do sistema, este cenário pode representar um grave risco ao andamento de projetos de grande porte. Uma atenuação possível seria indicação de um desenvolvedor sênior para a atividade, entretanto a Pro.NET não faz qualquer ressalva quanto à experiência do desenvolvedor envolvido.

4.2 Fluxos de trabalho

Conforme exposto no Capítulo 2, cada fluxo de trabalho do MSF 4.0 compreende atividades, e estas, por sua vez, agregam sub-atividades. Por restrições de espaço este trabalho limita-se a oferecer uma visão comparativa dos fluxos de trabalho e das atividades, sem adentrar em discussões a respeito das sub-atividades. As sub-seções seguintes apresentam as atividades que compoem cada fluxo de trabalho e, a seguir, procuram identificar quais macro-atividades (ou atividades) apresentam práticas semelhantes na Pro.NET. Ao final do capítulo, apresentaremos um sumário desta análise.

4.2.1 Estabelecer processo do projeto

1. Evocar processo organizacional
2. Aplicar critérios e *guidelines* de *tailoring*

3. Avaliar processo
4. Estabelecer plano de mensuração
5. Estabelecer plano de gerenciamento de dados do projeto
6. Popular o servidor de mensuração do projeto
7. Popular a biblioteca de definições do processo
8. Manter o catálogo de definições do processo

As atividades 1, 2 e 3 dizem respeito à instanciação do processo para o contexto particular de uma organização ou projeto e, como explicitado no Capítulo 2, tais atividades não são contempladas pela Pro.NET. O restante das atividades, embora tenha relação direta com a *suite* de ferramentas que suporta o MSF 4.0[1], também está relacionado com a instanciação do processo.

4.2.2 Capturar visão do produto

1. Redigir visão do produto
2. Definir *personas*
3. Desenvolver *lifestyle snapshot* (opcional)
4. Revisar visão do produto

A primeira atividade deste fluxo de trabalho é suportada integralmente pela atividade *Definir visão* pertencente à macro-atividade *Analisar problema* da Pro.NET.

Já a atividade 2 poderia ser relacionada com o passo de identificação dos *stakeholders* do projeto (dentre eles os usuários) da mesma atividade *Definir visão*, ou ainda com a identificação dos atores da aplicação, presente nas atividades *Modelar negócio* e *Levantar requisitos* da macro-atividade *Elaborar projeto conceitual*. Mas o conceito de *persona*, como introduzido por Cooper [Coo99], vai além de uma simples identificação dos atores e define um personagem fictício que representa um grupo particular de pessoas.

O desenvolvimento de *lifestyle snapshots*, uma extensão do conceito de *personas* que é contemplado pela atividade 3 do fluxo, também não é abordado pela Pro.NET, mas sendo esta uma atividade opcional, sua falta não põe em risco a conformidade da metodologia com o MSF 4.0

Por fim, a atividade 4 é suportada pelo passo de verificação contido na atividade *Definir visão*, com a ressalva que a Pro.NET não promove uma revisão comparativa entre a visão do produto e a da organização, como faz o MSF 4.0.

[1] *Visual Studio Team System*

4.2.3 Criar um cenário

1. Fazer *brainstorm* dos cenários (opcional)
2. Priorizar cenários
3. Escrever cenários
4. Fazer *storyboard* dos cenários
5. Validar cenários
6. Escrever testes de aceitação do usuário

A fim de capturar as interações do usuário com o sistema, este fluxo de trabalho utiliza a técnica de elicitação de requisitos baseada em cenários. A Pro.NET por sua vez, faz uso do equivalente a esta técnica na modelagem OO: os casos de usos. Cada caso de uso pode ser visto como o conjunto de todos cenários possíveis durante uma interação do usuário com o sistema [MF97]. Assim, as atividades deste fluxo são suportadas pela macro-atividade *Elaborar projeto conceitual*, que coleta os requisitos do sistema através de casos de uso.

A atividade 1 tem como equivalente na Pro.NET as atividades *Elicitar requisitos* e *Levantar requisitos*, que coletam uma relação inicial dos caso de usos da aplicação. Também a segunda atividade deste fluxo é suportada pela atividade *Levantar requisitos*, através do seu passo que determina a priorização dos casos de uso dentro da escala *essencial*, *importante* ou *desejável*.

O detalhamento de cada cenário (atividade 3) encontra equivalência na atividade *Detalhar requisitos*, que expande as definições iniciais dos casos de uso com a adição de fluxos de evento principais e secundários.

Apesar da técnica de *storyboarding* de cenários propriamente dita não ser abordada pela Pro.NET, sua atividade *Prototipar interface* traz os mesmos resultados da atividade 4 do presente fluxo: um modelo/protótipo de interface com usuário baseado nos casos de uso, e conseqüentemente nos cenários, da aplicação.

A validação dos cenários coletados (atividade 5) é suportada pelos dois últimos passos da atividade *Detalhar requisitos*.

Testes de aceitação do usuário podem se referir tanto àqueles cuja execução se dá na última fase do ciclo de desenvolvimento, já em ambiente de produção, quanto àqueles nos quais uma lista de passos pré-determinados é executada pelo usuário do sistema, a fim de validar um cenário ou requisito não-funcional. A noção adotada pela atividade 6 é esta última. Na atividade *Executar testes* é mencionada a realização de testes de aceitação planejados durante a atividade *Elaborar plano de aceitação*. Entretanto, a Pro.NET não menciona explicitamente que a elaboração de tais testes tem por base os cenários coletados.

4.2.4 Criar um requisito de qualidade de serviço

1. Fazer *brainstorm* dos requisitos de qualidade de serviço (opcional)
2. Priorizar requisitos de qualidade de serviço

3. Escrever requisitos de qualidade de serviço
4. Validar requisitos de qualidade de serviço
5. Escrever casos de teste de aceitação do usuário

Requisitos de qualidade de serviço, como definidos pelo MSF 4.0, representam os requisitos não-funcionais da solução. Assim, as atividades 1, 2 e 3, deste fluxo são suportadas pela atividade *Levantar requisitos* da Pro.NET, que possui passos para a identificação e priorização dos mesmos.

Assim como ocorreu com a validação dos cenários os requisitos de qualidade de serviço (atividade 4) são validados pelos passos de avaliação de resultados das atividades *Levantar requisitos* e *Detalhar requisitos*.

Como referido no fluxo de trabalho *Criar um cenário*, a Pro.NET não deixa claro que a elaboração dos testes de aceitação é necessariamente baseada em cenários ou requisitos de qualidade de serviço.

4.2.5 Criar requisitos do produto

1. Desenvolver modelo de fluxo da interface com o usuário
2. Desenvolver modelo do domínio
3. Definir requisitos funcionais
4. Definir requisitos de interface
5. Definir requisitos de segurança
6. Definir requisitos de segurança do usuário
7. Definir requisitos operacionais
8. Alocar requisitos aos componente do produto
9. Priorizar funcionalidades
10. Validar requisitos

A primeira atividade deste fluxo é suportada pela atividade *Prototipar interface* pertencente à macro-atividade *Elaborar projeto conceitual*. Um modelo de domínio pode ser visto como um caso especial de um modelo mais completo, o de negócio. Assim, a atividade 2 é satisfeita pela atividade *Modelar negócio* também da macro-atividade *Elaborar projeto conceitual*

A atividade 3 concebe os requisitos funcionais da aplicação baseada nos cenários, nos requisitos de qualidade de serviço e no modelo de domínio. Na Pro.NET a atividade correspondente seria *Detalhar requisitos* que é responsável pelo detalhamento dos casos de uso (requisitos funcionais).

Na Pro.NET, a atividade *Projetar subsistemas* possui passos para a definição das responsabilidades e interfaces dos subsistemas da aplicação, contemplando assim, a atividade 4.

Os requisitos definidos através das atividades 5, 6, 7 podem ser considerados requisitos não-funcionais sendo assim cobertos pela atividade *Levantar requisitos*, responsável pela identificação dos requisitos não-funcionais na Pro.NET

A oitava atividade deste fluxo é suportada na Pro.NET pela macro-atividade *Elaborar projeto lógico* que promove o mapeamento dos requisitos em um conjunto de subsistemas através das atividades *Projetar subsistemas* e *Analisar casos de uso*.

A priorização das funcionalidades (atividade 9) é executada durante a atividade *Levantar requisitos*.

A Pro.NET não deixa claro em qualquer uma de suas atividades que a validação dos requisitos deve ser feita junto aos *stakeholders*: os passos de validação presentes na Pro.NET são internos à equipe. A atividade 10 não é, deste modo, satisfeita diretamente pela Pro.NET já que há uma exigência explícita quanto a esta questão.

4.2.6 Criar arquitetura da solução

1. Criar alternativas de particionamento para o projeto da aplicação
2. Projetar a arquitetura e implantação dos sistemas
3. Criar provas de conceito
4. Avaliar alternativas (LAAAM[2])
5. Selecionar arquitetura
6. Desenvolver modelo de ameaças[3]
7. Desenvolver modelo de performance

Para definir a arquitetura da solução, este fluxo elabora e avalia várias alternativas de modelos arquiteturais. Esta estratégia é completamente ausente na Pro.NET, de maneira que as atividades 1, 2 e 3 têm alguma correspondência na metodologia, mas com foco no projeto de um único modelo de arquitetura. As atividade 4 e 5 não possuem correspondência por motivos óbvios, já que não há construção de múltiplos modelos arquiteturais para avaliação.

As duas primeiras atividades deste fluxo têm suas atenções voltadas para a definição de uma arquitetura orientada a serviços. Não obstante, a filosofia empregada nestas atividades é da escolha dos requisitos mais representativos para o projeto da arquitetura, e a construção de uma estrutura de subsistemas relacionada a tais requisitos. Da mesma maneira, a Pro.NET, através da atividade *Projetar subsistemas*, projeta um conjunto de subsistemas baseado nos casos de uso mais representativos da especificação funcional. Além disso, a atividade 2 é responsável pela concepção de um modelo de implantação que é validado em relação à arquitetura. Na

[2] *Lightweight Architecture Alternative Analysis Method*

[3] Um modelo de ameaças documenta falhas de segurança conhecidas e descreve como solucioná-las.

Pro.NET, a atividade *Definir requisitos de instalação*, pertencente a macro-atividade *Elaborar projeto físico*, também define um modelo de implantação que considera a concepção da topologia de rede, de dados e de subsistemas da aplicação.

A criação de provas de conceito (atividade 3) é tratada na Pro.NET pela macro-atividade *Executar Validações de Implementação* que possui as atividades *Validar tecnologia* e *Construir prova de conceito*. Na Pro.NET estas atividades atuam produzindo protótipos de validação que têm papel fundamental no estudo da viabilidade da implementação.

A Pro.NET não oferece nenhum suporte às atividades 6 e 7, que controem, respectivamente, detalhados modelos de ameaças e de performance para a aplicação.

4.2.7 Análise

1. Análise de projeto e de desenvolvimento
2. Planejamento de testes
3. Análise de desdobramento do trabalho da experiência do usuário
4. Análise de desdobramento do trabalho da educação do usuário

Baseando-se nos cenários, requisitos funcionais e não-funcionais, solicitações de mudanças e na arquitetura construída, este fluxo aloca tarefas a serem executadas pelos membros da equipe. A Pro.NET não possui uma única atividade ou macro-atividade que tenha a responsabilidade de delegar diferentes tipos de tarefas aos membros da equipe.

A primeira atividade deste fluxo gera uma lista de tarefas de desenvolvimento a partir da análise dos requisitos do produto. Cada tarefa de desenvolvimento compreende a elaboração de um modelo de análise & projeto e sua posterior codificação, a fim de satisfazer o requisito em questão. A Pro.NET é dirigida por casos de uso, isto é, estas entidades guiam o desenvolvimento e a concretização da arquitetura na medida em quem são analisados e projetados. Desta maneira os casos de uso representam por si só o equivalente às tarefas de desenvolvimento do MSF 4.0.

A alocação de tarefas de teste (atividade 2) também é , na Pro.NET, dirigida pelos casos de uso. As atividades *Elaborar plano de testes* e *Elaborar projeto de testes* são responsáveis pela análise e escolha dos requisitos a serem testados e pela derivação de casos de teste a partir dos casos de uso selecionados, respectivamente.

A atividade 3 tem por responsabilidade gerar tarefas que norteiem e validem o desenvolvimento da interface gráfica com o usuário. Na Pro.NET a atividade *Prototipar interface*, pertencente à macro-atividade *Elaborar projeto conceitual*, cumpre o propósito de oferecer um alicerce para o desenvolvimento da interface com o usuário. Já a validação da interface através do planejamento de testes de usabilidade (outro produto da atividade deste fluxo), não encontra equivalência na Pro.NET.

A construção de documentos que auxiliem na educação do usuário é o objetivo das tarefas geradas pela atividade 4 deste fluxo. Na Pro.NET, a construção da documentação do sistema não segue uma abordagem tão sistemática quanto a apresentada nesta atividade. Todos os artefatos de documentação são gerados durante a atividade *Preparar documentação* que pertence à macro-atividade *Produzir artefatos de instalação e documentação*.

4.2.8 Estabelecer *baseline* da gerência de configuração

1. Revisar *guidelines* de gerência de configuração
2. Criar plano de gerência de configuração
3. Estabelecer política de controle de acesso da gerência de configuração
4. Estabelecer sistema de gerência de configuração
5. Estabelecer registros de gerência de configuração

A atividade 1 não é suportada pela Pro.NET, uma vez que não há uma avaliação dos *guidelines* de gerência de configuração da organização em relação ao projeto.

Na Pro.NET, a atividade *Elaborar plano de gerência de configuração* é responsável pela definição do Plano de Gerência de Configuração. A atividade da Pro.NET, entretanto, produz apenas um subconjunto do plano de gerência de configuração resultante da atividade 2 deste fluxo. Questões ausentes no plano gerado pela Pro.NET incluem: definição da estratégia de *baselining*, identificação dos tipos de *baseline* a serem usados, plano de treinamento de gerência de configuração, etc. Algumas das lacunas da atividade *Elaborar plano de gerência de configuração* são preenchidas por outras atividades da Pro.NET: a definição das ferramentas utilizadas para garantir a gerência de configuração e a definição de um processo de controle de mudanças, são feitas durante as atividades *Elaborar plano de ambiente* (macro-atividade *Planejar ambiente e gerência de configuração*) e *Integrar planos* (macro-atividade *Desenvolver Plano de projeto*), respectivamente. A elaboração de um plano de auditoria de gerência de configuração é parcialmente suportada pela Pro.NET, uma vez que o Plano de Qualidade permite a definição de auditorias de qualquer tipo.

A atividade *Elaborar plano de gerência de configuração* também satisfaz a terceira atividade deste fluxo, uma vez que o referido plano contém uma seção dedicada à definição dos responsáveis por cada operação de gerência de configuração.

As atividades 4 e 5 são responsáveis por tornar operacional o ambiente de gerência de configuração planejado anteriormente. O estabelecimento do sistema propriamente dito (hardware e software) (atividade 4) e a constituição do *baseline* incial do projeto (atividade 5) são satisfeitos na Pro.NET pela atividade *Administrar ambientes*.

4.2.9 Planejar projeto

1. Determinar fontes e categorias dos riscos
2. Determinar parâmetros dos riscos
3. Determinar estratégia de gerenciamento dos riscos
4. Planejar recursos do projeto
5. Planejar conhecimento e habilidades requeridas pelo projeto

6. Formar equipe de projeto
7. Estabelecer contrato de conduta entre os membros do projeto
8. Definir papéis e responsabilidades no projeto
9. Definir plano de comunicação do projeto
10. Identificar *stakeholders* do projeto
11. Definir orçamento e cronograma
12. Planejar envolvimento dos stakeholders do projeto
13. Estimar projeto
14. Definir procedimentos operacionais
15. Planejar revisão do projeto
16. Revisão do projeto pelo IPM Officer
17. Obter comprometimento com o projeto

As três primeiras atividades deste fluxo concernem ao planejamento de riscos do projeto. Na Pro.NET, é a atividade *Definir gerência de riscos* (macro-atividade *Desenvolver plano de gerência de riscos*) promove a identificação das fontes dos riscos, bem como estabelece uma taxonomia para os mesmos. A atividade 2 trata da customização do artefato que representa um risco, não sendo assim contemplada pela Pro.NET. O suporte a terceira atividade é discutido mais adiante nesta sub-seção.

Definir habilidades, competências e recursos necessários para a execução do projeto, é o objetivo das atividades 4 e 5. A correspondência com a Pro.NET é estabelecida através das atividades *Elaborar planejamento inicial do projeto* e *Elaborar plano de ambiente*. A primeira estima os requisitos necessários aos recursos humanos que participarão do projeto, enquanto a última define os recursos físicos a serem alocados.

As atividades 6, 7 e 8 tratam do estabelecimento da equipe de projeto. A atividade 6 é satisfeita pela atividade *Elaborar planejamento inicial do projeto* que perfaz o recrutamento inicial da equipe de projeto. Na atividade *Elaborar plano de equipe*, a Pro.NET define os papéis e responsabilidades dos membros da equipe (atividade 8). A atividade 7 envolve a elaboração de um contrato de conduta entre os membros da equipe (*Team Charter*) que não tem correspondência direta na Pro.NET.

Definir um plano de comunicação para o projeto (atividade 9) envolve a identificação de canais de comunicação e a maneira como estes irão operar durante o projeto. A Pro.NET, através da atividade *Elaborar plano de comunicação*, define como se dá a comunicação dos membros do projeto, tanto internamente quanto com o meio externo. A Pro.NET também compactua com o MSF 4.0 quanto à necessidade de manter certos membros da equipe isolados de comunicação externa, como desenvolvedores e analistas de teste.

Conforme relatado anteriormente, a atividade *Definir visão*, apresenta um passo de identificação dos *stakeholders* envolvidos no projeto. Tal passo, no qual é sugerida uma série de questionamentos a fim de identificar os *stakeholders*, pode ser visto como equivalente ao *brainstorm* da atividade 10 deste fluxo.

As definições de cronogramas para as iterações e de orçamento para o projeto (atividade 11) são feitas, respectivamente, durante as atividades *Elaborar plano de implementação* e *Integrar Planos*, ambas pertencentes a macro-atividade *Desenvolver plano de projeto*. Assim como no MSF 4.0, a Pro.NET lança mão de uma análise de dependência entre os requisitos, a fim de decidir a ordem de implementação destes durante as iterações subseqüentes. A Pro.NET não aborda, entretanto, a estimativa de orçamento do projeto enquanto dependência direta dos recursos alocados para o mesmo: no MSF 4.0, há a determinação de que os custos do projeto devem ser sempre derivados e avaliados em relação a quanto pode ser fornecido para o projeto. A Pro.NET admite que recursos suficientes foram alocados antes do início do mesmo.

Na Pro.NET, a atividade *Definir visão* promove uma identificação dos *stakeholders* e, posteriormente, a elaboração do Plano de Equipe delineia as responsabilidades daqueles que participarão mais ativamente do projeto. Atividade 12 deste fluxo exige porém, que haja um detalhamento de como todos os *stakeholders* estarão envolvidos no projeto, mesmo aqueles que não são membros diretos da equipe. Sendo assim, tal atividade é apenas parcialmente satisfeita pela Pro.NET.

O objetivo da atividade 13 é refinar a visão do projeto em um escopo que defina o que vem a ser critico para a qualidade do projeto (critical to quality - CTQ). Isto é feito através da identificação de um conjunto de cenários mais representativos para a essência do produto, e do posterior desenvolvimento de *storyboards* que envolvam uma interação entre estes cenários. Na atividade *Definir visão* da Pro.NET há uma restrição do escopo da aplicação com base nas funcionalidades a serem entregues. A elaboração deste escopo, entretanto não leva em conta a análise de cenários, sendo direcionada pelos requisitos iniciais coletados.

A atividade 14 aborda a adaptação dos procedimentos operacionais do MSF para a realidade da organização ou do projeto. A Pro.NET define, através da atividade *Elaborar plano de qualidade*, que processos adicionais ou alterados devem ser registrados no Plano de Qualidade, estabelecendo correspondência, assim, com a atividade do MSF 4.0. Esta atividade da Pro.NET também satisfaz, portanto, a atividade 3 deste fluxo já que esta última trata da customização da estratégia de gerenciamento de riscos definida pelo MSF 4.0.

Nenhuma das três últimas atividades deste fluxo tem correspondência direta na Pro.NET. Não há atividade reponsável por coordenar reuniões para a homologação do Plano de Projeto, nem por coletar de aceites dos membros da equipe em relação ao mesmo.

4.2.10 Planejar iteração

1. Selecionar pendências da iteração
2. Análise
3. Planejar conhecimento e habilidades
4. Planejar recursos da iteração

5. Formar equipe(s) da iteração
6. Definir papéis e responsabilidades da iteração
7. Definir plano de comunicação
8. Identificar *stakeholders* da iteração
9. Planejar envolvimento dos *stakeholders* na iteração
10. Estimar iteração
11. Definir orçamento e cronograma da iteração
12. Revisão do plano de iteração
13. Revisão do plano de iteração pelo IPM Officer
14. Obter comprometimento com a iteração

O fluxo de trabalho *Planejar iteração* promove um refinamento do que foi planejado anteriormente, através do plano de projeto, em relação a iteração a ser executada. Também são revisadas as atividades de consolidação da equipe e planejamento de comunicação. Além deste refinamento, este fluxo de trabalho aloca as tarefas a serem executadas durante a iteração através da atividade 2.

Quando comparado ao presente fluxo de trabalho, o suporte oferecido pela Pro.NET ao planejamento de uma iteração é apenas parcial. Na Pro.NET, as macro-atividades *Planejar releases internos* e *Planejar correções* refinam, através da atividade *Elaborar plano de release interno*, o Plano de Projeto em relação às iterações da fase de Desenvolvimento e de Estabilização. Sub-entende-se entretanto, que, por se tratar de uma metodologia iterativa e incremental, o presente fluxo de trabalho seria suportado por revisões no Plano de Projeto.

4.2.11 Estabelecer ambientes

- Planejar integração do produto
- Estabelecer ambiente de testes de unidade
- Estabelecer ambiente de integração
- Verificar ambiente de integração
- Selecionar regras e *guidelines* de análise estática
- Selecionar guidelines de codificação do projeto

Na Pro.NET, a atividade *Elaborar plano de gerência de configuração* possui uma passo no qual podem ser definidas as políticas de integração a serem utilizadas pelo projeto, sendo assim equivalente a primeira atividade deste fluxo.

A atividade 2 visa planejar os procedimentos utilizados para a realização de testes de unidade, assim como prover os recursos de ambiente necessários para isto. A Pro.NET, faz uso do *Guia de testes de unidade* para definir como tais testes devem ser realizados. Adicionalmente, a atividade *Administrar ambientes* fornece os recursos necessários para a condução dos testes.

A terceira atividade deste fluxo demanda o estabelecimento de estratégias de *branching*, critérios e convenções de nomeação de *build* e o estabelecimento do ambiente físico de integração. Na Pro.NET não há recomendações diretas quanto ao planejamento de estratégia de *branching*. As outras demandas são satisfeitas respectivamente pelas atividades *Elaborar plano de gerência de configuração* e *Administrar ambientes*.

A verificação do ambiente de integração (atividade 4) é feita na Pro.NET pela atividade *Administrar ambientes*

As duas últimas atividades deste fluxo estabelecem planos de controle de qualidade para o código a ser integrado. Na Pro.NET este planejamento é feito pela atividade *Elaborar plano de qualidade*, na qual se deve definir guias, normas, padrões e métricas de qualidade além de de especificar como as inspeções serão realizadas. A única ressalva diz respeito ao fato da Pro.NET não recomendar diretamente o uso de ferramentas de análise estática, como o faz a atividade 5.

4.2.12 Implementar uma tarefa de desenvolvimento

1. Estimar uma tarefa de desenvolvimento
2. Elaborar modelo de projeto
3. Criar ou atualizar um teste de sistema ou verificação
4. Preparar revisão do modelo de projeto
5. Revisar modelo de projeto
6. Escrever ou atualizar teste de unidade
7. Escrever código
8. Executar análise do código
9. Executar testes de unidade
10. Fazer *refactoring* do código
11. Preparar revisão de código
12. Revisar código
13. Integrar mudanças

A implementação de uma tarefa de desenvolvimento compreende a produção de um modelo de projeto e a posterior transformação deste em código. Para que isto ocorra, o desenvolvedor toma por base a descrição da tarefa e um modelo arquitetural de alto nível do sistema.

Estimar a implementação de uma tarefa de desenvolvimento portanto, compreende estimar tanto a elaboração do modelo de projeto, quanto a implementação do código e as atividades de testes e revisão associadas. Na Pro.NET, as estimativas relacionadas a implementação são elaboradas durante a atividade *Elaborar plano de implementação*. Tal Plano de Implementação faz parte da miríade dos que terminam por integrar o Plano de Projeto e suas estimativas são baseadas numa análise geral de todos os casos de uso a serem implementados. Assim, a atividade que estima a implementação na Pro.NET suporta parcialmente a atividade deste fluxo, já que aqui os desenvolvedores estimam individualmente as suas tarefas.

A elaboração de um modelo de projeto (atividade 2) produz o que o MSF 4.0 denomina *Technical data package*: um conjunto de documentos que descrevem a tarefa de desenvolvimento em detalhes suficientes para a sua codificação. Um modelo de projeto, portanto. Na Pro.NET as atividades *Analisar casos de uso*, *Projetar casos de uso* e *Projetar banco de dados* são responsáveis pela construção gradativa do modelo de projeto a partir dos casos de uso.

A terceira atividade deste fluxo elabora testes com base no modelo de projeto produzido durante a atividade anterior. A Pro.NET não se baseia no modelo de projeto para produzir seus casos de testes. As atividades *Elaborar plano de testes* e *Elaborar projetos de testes* tomam por base o fluxo de evento de cada caso de uso, para gerar o caso de teste correspondente.

As atividades da Pro.NET responsáveis pela produção de um modelo de projeto possuem passos específicos para a avaliação destas. Entretanto, não são agendadas ou realizadas reuniões formais com um conjunto de revisores. Assim, a revisão do modelo de projeto da Pro.NET não satisfaz as atividades 4 e 5 deste fluxo, que exigem explicitamente revisões desta natureza.

Conforme explicitado no fluxo de trabalho anterior, o planejamento e execução dos testes de unidade é de responsabilidade da atividade *Conduzir testes de unidade* que satisfaz, assim, as atividades 6 e 9 deste fluxo.

A implementação do código propriamente dito (atividade 7) encontra correspondência na atividade *Implementar código da aplicação*.

A atividade 8 promove uma inspeção dos *assemblies* resultantes da compilação do código e relata não-conformidades deste com *guidelines* de desenvolvimento da plataforma .NET: convenções de nome, segurança de localização, performance, etc. Embora a Pro.NET não faça recomendações específicas com relação à inspeção de *assemblies*, o Plano de Qualidade abre espaço para a definição de inspeções de qualquer natureza, suportando, assim, parcialmente a atividade 8.

A garantia de controle de qualidade dos artefatos, inclusive do código fonte, produzidos está presente na Pro.NET através da atividade *Elaborar plano de qualidade*. As inspeções ou revisões são executadas a partir da atividade *Gerenciar qualidade*, pertencente a macro-atividade *Gerenciar projeto*.

A realização de *refactoring* no código fonte (atividade 10) é motivada pela presença de não-conformidades encontradas durante as revisões e inspeções deste. Na Pro.NET, a correspondência dá-se através da atividade *Realizar refactoring*. Ressalvas, entretanto, devem ser ser feitas. A Pro.NET não deixa claro que os resultados das inspeções motivam a realização de

refactoring, nem menciona a realização de testes de unidade após o processo, como faz o MSF 4.0.

A atividade 13, por fim, é satisfeita na Pro.NET através da atividade *Submeter documento de Build*.

4.2.13 Gerenciar solicitações de mudança

1. Criar solicitação de mudança
2. Analisar solicitação de mudança
3. Priorizar solicitações de mudança
4. Revisar solicitações de mudança

A primeira atividade deste fluxo cria uma solicitação de mudança com base na detecção de um defeito. O analista de negócio é responsável por esta atividade, propondo uma solução para o defeito, identificando os *baselines* afetados e criando a solicitação de mudança apropriada. Na Pro.NET, a atividade *Solicitar mudança* (macro-atividade *Gerenciar projeto*) mostra-se equivalente. Entretanto qualquer papel do modelo de equipe pode executar esta atividade.

A análise da solicitação de mudança (atividade 2) é feita também pelo analista de negócio. O objetivo é analisar o escopo do impacto da mudança solicitada, assim como refinar a solução proposta inicialmente. Esta análise faz-se necessária para a posterior revisão da solicitação pelo comitê de controle de mudanças. Na Pro.NET não existe esta análise: a solicitação é diretamente submetida ao comitê (representado pelo gerente de projeto).

A atividade 3 trata da priorização das solicitações e é ainda executada pelo analista de negócio, que faz uso do modelo Kano de fatores de qualidade [NSK84]. Na Pro.NET, a priorização das solicitações de mudança só ocorre quando da análise do gerente de projeto e não segue nenhum modelo específico.

A revisão das solicitações de mudanças (atividade 4) é de responsabilidade do comitê de controle de mudanças, que no MSF 4.0 é representado pelos papéis de Gerente de *Release*, Gerente de Desenvolvimento, Gerente de Testes, Arquiteto, Especialista em Eduação do Usuário e, finalmente, o Gerente de Projeto. Esta revisão aprova, rejeita ou posterga a decisão a respeito da solicitação. Na Pro.NET, a atividade *Avaliar mudança* é responsável por este julgamento, bem como pela priorização da mudança. Uma vez aprovada, a atividade *Programar mudanças aprovadas* será executada. A atualização do estado da mudança fica a cargo da atividade *Atualizar status da mudança*.

4.2.14 Corrigir um defeito

1. Reproduzir o defeito
2. Localizar a causa do defeito
3. Encaminhar defeito

4. Selecionar uma estratégia para a correção
5. Codificar correção
6. Escrever ou atualizar teste de unidade
7. Executar testes de unidade
8. Fazer *refactoring* do código
9. Preparar revisão de código
10. Revisar código
11. Integrar mudanças

Na Pro.NET a macro-atividade *Ajustar aplicação* é responsável pela correção de um defeito. As atividades 1 e 2 deste fluxo são ambas suportadas pela atividade *Corrigir defeitos*, que possui passos para a reprodução do defeito e localização da sua causa.

A Pro.NET prevê, em sua atividade *Corrigir defeitos*, a realocação da tarefa de corrigir o defeito pela localização do mesmo em uma área do código fonte não pertencente ao desenvolvedor em questão.

A atividade 4 prevê a escolha de uma estratégia para a correção do defeito, que pode variar de modificações arquiteturais às de performance. Na Pro.NET, a atividade *Priorizar defeitos* identifica uma resolução para o defeito, mas não faz menção que esta resolução possa incluir modificações arquiteturais ou de performance. Dentro da atividade *Corrigir defeitos*, há a codificação da correção necessária, satisfazendo assim a atividade 5 deste fluxo.

A partir da codificação da correção, a seqüência de atividades guarda semelhança com o que já foi apresentado no fluxo *Implementar uma tarefa de desenvolvimento*. Da mesma maneira, a macro-atividade *Ajustar aplicação* possui atividades para a escrita e execução de testes de unidade (*Conduzir testes de unidade*) e integração das mudanças do código (*Submeter documento de build*), satisfazendo assim, as atividades 6, 7 e 11 deste fluxo.

Não há a realização de refactoring durante a macro-atividade *Corrigir defeitos*. Na Pro.NET a condução da atividade *Realizar refactoring* está restrita a macro-atividade *Desenvolver Aplicação*.

Conforme exposto no fluxo de trabalho *Implementar uma tarefa de desenvolvimento*, a atividade 9 não possui equivalência direta na Pro.NET, enquanto a atividade 10 é suportada pela atividade *Gerenciar qualidade*.

4.2.15 Executar *build* do produto

1. Executar o *build*
2. Verificar o *build*
3. Aceitar o *build*

4. Corrigir o *build* (opcional)

Na Pro.NET, a macro-atividade *Gerenciar itens de configuração* compreende a atividade *Conduzir integração contínua*, que é responsável por executar um *build* do produto. Esta atividade contém passos para a construção e verificação do *build*, satisfazendo assim as atividades 1, 2 e 3.

A última das atividades objetiva a resolução do problema que levou a falha durante a construção do *build*. Se a falha não foi causada por um problema de ambiente, então os desenvolvedores responsáveis pelo código são informados a respeito do problema. Na Pro.NET a geração e o encaminhamento de um arquivo de *log* aos desenvolvedores cumpre este papel informativo.

4.2.16 Finalizar um *defeito*

1. Verificar correção

2. Finalizar *defeito*

Este fluxo está diretamente relacionado ao ciclo de vida do item de trabalho *defeito*. A Pro.NET define uma artefato denominado *Registro de Defeitos*, que serve como base para a execução da atividade *Corrigir defeitos*. O Último passo desta atividade atualiza o *status* do defeito de maneira a refletir as modificações feitas no código da aplicação, sendo assim equivalente à atividade 2. A verificação da correção não ocorre na Pro.NET, já que a correção de um defeito, feita por um desenvolvedor, não é verificada diretamente, e sim indiretamente, quando da condução de novos testes de desenvolvimento ou estabilização.

4.2.17 Verificar um requisito funcional

1. Desenvolver testes

2. Selecionar e executar testes

3. Criar um *defeito*

Na Pro.NET, a verificação de um requisito funcional é abordada pelas macro-atividades *Conduzir testes de desenvolvimento* e *Conduzir testes de estabilização*, dependendo se o projeto está na fase de *Desenvolvimento* ou *Estabilização*, respectivamente.

A atividade *Implementar testes* da Pro.NET é responsável pela produção de programas a partir dos casos de teste. Tais programas contribuirão para que as atividades de teste sejam realizadas de forma mais automática. Este também é o foco da primeira atividade deste fluxo.

A execução do teste propriamente dito, e a geração de um artefato que documente o eventual defeito encontrado (atividades 2 e 3), são responsabilidades da atividade *Executar testes*. Esta atividade gera um artefato denominado Registro de Defeitos se o teste não for bem sucedido.

4.2.18 Testar um cenário

1. Desenvolver testes
2. Selecionar e executar testes
3. Conduzir testes exploratórios
4. Criar um *defeito*

Na Pro.NET cada cenário de um caso de uso (requisito funcional) pode dar origem a um caso de teste distinto. Desta maneira, o conjunto de casos de teste utilizado para testar um requisito funcional, já cobre os testes de todos os seus cenários possíveis. Assim, os objetivos e atividades do presente fluxo são satisfeitos pelas mesmas atividades da Pro.NET citadas no fluxo anterior.

A técnica de teste exploratório (atividade 3), também chamada de teste *ad hoc*, também é suportada pela Pro.NET já que o Plano de Pestes pode prever que as atividades de teste não necessariamente serão feitas de forma automática, isto é, por meio de programas e *scripts*.

4.2.19 Testar um requisito de qualidade de serviço

1. Desenvolver testes
2. Selecionar e executar testes
3. Conduzir testes exploratórios
4. Criar um *defeito*

O desenvolvimento e execução de testes dos requisitos de qualidade de serviço (requisitos não-funcionais) é abordado na Pro.NET como exposto nos dois últimos fluxos, ou seja, através das atividades *Implementar testes* e *Executar testes*. Entretanto, tais atividades devem ser preferencialmente executadas como parte da macro-atividade *Conduzir testes de estabilização*, já que, durante a execução desta, os requisitos funcionais estão mais estáveis e o produto encontra-se em um ambiente mais próximo do de produção.

4.2.20 Verificar um requisito operacional

1. Desenvolver testes
2. Selecionar e executar testes
3. Criar um *defeito*

A verificação de requisitos operacionais encontra seu suporte nas mesmas atividades da Pro.NET explicitadas nos fluxos anteriores. E assim como os testes dos requisitos de qualidade de serviço, também deve ser realizada durante a macro-atividade *Conduzir testes de estabilização* pela própria natureza dos requisitos a serem verificados, operacionais.

4.2.21 Desenvolver documentação

1. Criar rascunhos da documentação
2. Criar plano de documentação
3. Escrever documentação
4. Editar documentação
5. Executar revisão da documentação por pares
6. Testar documentação

O MSF 4.0 utiliza uma abordagem mais metódica que a empregada pela Pro.NET em relação ao desenvolvimento da documentação. Assim, as atividades 1 e 2 não encontram correspondência na Pro.NET, cuja única atividade que contempla a produção de documentação, *Preparar documentação*, não alicerça a escrita dos documentos em qualquer planejamento. As revisões da documentação produzida (atividades 4 e 5) também são de responsabilidade da atividade *Preparar documentação*, entretanto a Pro.NET não faz menção explícita sobre a participação de outros membros da equipe durante este processo. A condução de procedimentos de testes para a documentação é prevista pela macro-atividade *Conduzir testes de estabilização*.

4.2.22 Lançar um produto

1. Estabelecer ambiente dos testes de aceitação
2. Estabelecer procedimentos e critérios de aceitação do usuário
3. Selecionar candidato a *release*
4. Criar plano de implantação
5. Executar testes de aceitação do usuário
6. Analisar resultados dos testes de aceitação do usuário
7. Aprovar produto para lançamento
8. Criar pacote de instalação
9. Executar plano de implantação
10. Criar *Release notes*

Assim como quaisquer outros ambientes necessários à realização das atividades do projeto, o ambiente de testes de aceitação é estabelecido na Pro.NET através da atividade *Administrar ambientes*.

Os critérios e procedimentos adotados para os testes de aceitação são definidos quando da execução da atividade *Elaborar plano de aceitação*, durante o desenvolvimento do Plano de Projeto.

A seleção de um *build* candidato a *release* constitui responsabilidade conjunta da macro-atividade *Conduzir testes de estabilização* e da atividade *Avaliar release interno de estabilização*. Durante a macro-atividade, os testes são implementados e executados repetidamente até que a solução seja homologada. O critério definido para a homologação é a progressiva estabilização do produto. A atividade *Avaliar release interno de estabilização* decide se um dado *release* interno pode ganhar o status de candidato.

A atividade 4, cujo foco é planejamento do processo de implantação, é coberta na Pro.NET pela macro-atividade *Planejar implantação*, que contém as atividades *Planejar capacitação dos usuário* e *Planejar disponibilização e instalação*.

A Pro.NET não define que os testes de aceitação serão executados utilizando um candidato a *release*. A responsabilidade destes testes recai sobre a atividade *Executar testes*, ou seja, ocorre antes da homologação do produto e sua consequente promoção a *release* candidato. Uma vez executados os testes, a análise de seus resultados (atividade 6) é realizada a fim de verificar a conformidade com os critérios de aceitação definidos anteriormente. Tal análise é suportada pela atividade *Avaliar release interno de estabilização*.

Na Pro.NET, a decisão de implantação definitiva da solução não cabe a qualquer atividade específica. Uma vez que o desempenho da atividade *Testar piloto* foi satisfatório, o produto é gradualmente implantado no restante do ambiente de produção.

A criação de um pacote de instalação é executada em conjunto pelas atividades *Produzir artefatos de instalação* e *Instalar a aplicação*, que criam os programas de instalação e unem estes à documentação a ser distribuída ao usuário, respectivamente.

A execução do plano de instalação é de responsabilidade da atividade *Efetuar Instalação* da Pro.NET.

A última atividade deste fluxo produz o documento de *release notes*, que é concebido pela atividade *Preparar documentação*.

4.2.23 Gerenciamento de ocorrência

1. Revisar *log* de ocorrência
2. Analisar ocorrência
3. Tomar ação corretiva
4. Monitorar ação corretiva
5. Revisar ação e verificar solução

Na Pro.NET, o gerenciamento de ocorrências que representam impedimentos para a execução bem sucedida do projeto é feita de maneira proativa. As reuniões realizadas durante a atividade *Monitorar status do projeto* visam a identificação de prossíveis problemas. As eventuais ocorrências são reportadas através de relatórios de comunicação, que servem como

artefatos de entrada para a atividade *Controlar problemas*. Tal atividade analisa, define medidas corretivas e faz as solicitações de mudança necessárias. Posteriormente, a atividade *Monitorar status do projeto* revisa e acompanha o andamdento das solicitações.

Desta maneira, o gerenciamento de ocorrências deste fluxo é plenamente suportado pela Pro.NET

4.2.24 Gerenciamento de risco

A versão do MSF 4.0 analisada por este trabalho não apresenta uma descrição detalhada deste fluxo. Acredita-se que esta ausência seja devida ao MSF 4.0 encontrar-se em uma versão de *pre-release* e que a mesma será corrigida em futuras versões. De todo modo, a gerência de riscos é suportada na Pro.NET a partir das macro-atividades *Desenvolver plano de gerência de riscos* e *Avaliar riscos*.

4.3 Considerações finais

Uma comparação entre duas metodologias não envolve apenas a verificação da presença de práticas semelhantes em ambas. O aspecto temporal, isto é, a ordem em que estas atividades são executadas, também deve ser levado em consideração. Além disso, o papel responsável pela execução de cada atividade constitui outro ponto de comparação, assim como os artefatos produzidos. Este capítulo representa apenas um passo inicial da comparação entre o MSF 4.0 e a metodologia Pro.NET. As Tabelas 4.2 e 4.3 sintetizam a discussão realizada. A classificação do nível de satisfação que a Pro.NET apresenta em relação a cada fluxo do MSF 4.0 segue a escala delineada no início deste capítulo.

Dos 24 fluxos de trabalho analisados, mais da metade mostrou-se bem suportada pelas atividades e macro-atividades da Pro.NET (nível de satisfação *alto*). As não-conformidades apresentadas pela Pro.NET em relação ao restante dos fluxos são decorrentes de alguns fatores:

- Diferenças na ordem de execução das atividades dentro de um fluxo, papel responsável pelas mesmas, nível de detalhamento dos artefatos produzidos ou na motivação para a realização das atividades;
- Ausência de certas atividades;
- Diferenças que levam em conta o aspecto temporal do MSF 4.0, isto é, em como *framework* distribui as responsabilidades e tarefas entre seus fluxos de trabalho através das fases;
- O uso, pelo MSF 4.0, de técnicas não abordadas pela Pro.NET para a execução de certas atividades. Por exemplo: o MSF 4.0 usa a técnica LAAAM para a avaliação de possíveis opções de arquitetura da solução, a Pro.NET não;
- Diferenças de ordem tecnológica, já que o MSF 4.0 foi projetado para interagir fortemente com o *Visual Studio Team System* [Micb]

As não-conformidades cujas resoluções dar-se-iam de maneira mais direta seriam as que estão ligadas aos dois primeiros conjuntos de fatores, pois as demais exigem análises comparativas mais elaboradas e uma reestruturação mais profunda da metodologia Pro.NET.

Um conjuto de sugestões iniciais incluiria, mas não estaria limitado a:

- A Pro.NET prevê a realização de revisões/inspeções de seus principais artefatos no Plano de Qualidade. Tais inspeções contemplam a formação de uma equipe de revisores, estando de acordo com as revisões encontradas no MSF 4.0. Entretanto, a execução destas revisões não é mencionada explicitamente durante as macro-atividades da Pro.NET. A inclusão explícita de atividades desta natureza aumentaria, portanto, o nível de satisfação da Pro.NET em relação ao MSF 4.0.;
- Modificação da macro-atividade *Desenvolver aplicação* de forma a tornar a execução da atividade *Realizar refactoring* uma conseqüência da realização de inspeções de qualidade no código;
- Introdução de uma atividade específica para realizar o planejamento de qualquer iteração, e não só das que compoem as fases de Desenvolvimento e Estabilização;
- Na macro-atividade *Conduzir testes de estabilização*, postergar a execução dos testes de aceitação do usuário da atividade *Executar testes* para a atividade *Testar piloto*. O envolvimento do usuário em atividades de teste de aceitação antes da homologação da solução não parece fazer sentido, uma vez que a freqüência da descoberta dos defeitos nesta fase pode frustrar as expectativas do usuário;
- Introdução de atividades que planejem e executem de testes de usabilidade;
- Introdução do papel de Arquiteto no modelo de equipe.
- Introdução de atividades que validem os principais marcos do projeto junto aos *stakeholders*, pois embora haja uma recomendação da Pro.NET quanto a isto, este tipo de validação não está explicitado em nenhuma atividade.
- Adição de uma atividade que, como no MSF 4.0, contemple a construção de um projeto de arquitetura da solução. No entanto, considerando as macro-atividades *Analisar casos de uso* e *Projetar casos de uso*, parece ser mais adequada à Pro.NET a abordagem utilizada pelo RUP: inserção de uma atividade de projeto de arquitetura entre o passo de análise e o passo de projeto.

Fluxo de Trabalho (MSF 4.0)	*Atividades (Pro.NET)*	*Satisfação*
Estabelecer processo do projeto	NA	Inexistente
Capturar visão do produto	Definir visão	Alto
	Modelar negócio	
	Levantar requisitos	
Criar cenário	Levantar requisitos	Alto
	Elicitar requisitos	
	Detalhar requisitos	
	Prototipar interface	
	Elaborar plano de aceitação	
Criar requisito de qualidade de serviço	Levantar requisitos	Alto
	Detalhar requisitos	
	Elaborar plano de aceitação	
Criar Requisitos do Produto	Levantar requisitos	Alto
	Elicitar requisitos	
	Detalhar requisitos	
	Modelar negócio	
	Prototipar interface	
	Projetar subsistemas	
	Analisar casos de uso	
Criar Arquitetura da Solução	Projetar subsistemas	Baixo
	Analisar casos de uso	
	Definir requisitos de instalação	
	Validar tecnologia	
	Construir prova de conceito	
Análise	Planejar testes	Baixo
	Projetar testes	
	Prototipar interface	
	Preparar documentação	
Estabelecer *baseline* da gerência de configuração	Elaborar plano de ger. de configuração	Médio
	Integrar planos	
	Elaborar plano de ambiente	
	Elaborar plano de qualidade	
	Administrar ambientes	
Planejar projeto	Definir gerência de riscos	Alto
	Elaborar planejamento inicial do projeto	
	Elaborar plano de ambiente	
	Elaborar plano de equipe	
	Elaborar plano de comunicação	
	Definir visão	
	Elaborar plano de implementação	
	Integrar planos	
	Elaborar plano de qualidade	
Planejar Iteração	Elaborar plano de release interno	Baixo

Tabela 4.2 Relação entre fluxos de trabalho do MSF 4.0 e as atividades da Pro.NET

Fluxo de Trabalho (MSF 4.0)	*Atividades (Pro.NET)*	*Satisfação*
Estabelecer ambientes	Elaborar plano de ger. de configuração	Alto
	Administrar ambientes	
	Elaborar plano de qualidade	
Implementar uma tarefa de desenvolvimento	Elaborar plano de implementação	Médio
	Analisar casos de uso	
	Projetar casos de uso	
	Projetar banco de dados	
	Conduzir testes de unidade	
	Implementar código da aplicação	
	Realizar *refactoring*	
	Gerenciar qualidade	
	Submeter documento de *build*	
Gerenciar solicitações de mudança	Solicitar mudança	Alto
	Avaliar mudança	
	Atualizar status da mudança	
	Programar mudanças aprovadas	
Consertar um defeito	Corrigir defeitos	Médio
	Priorizar defeitos	
	Conduzir testes de unidade	
	Gerenciar qualidade	
	Submeter documento de *build*	
Fazer *Build* do Produto	Conduzir integração contínua	Alto
Finalizar um defeito	Corrigir defeitos	Médio
Testar um cenário	Implementar testes	Alto
	Executar testes	
Testar um requisito de qualidade de serviço	Implementar testes	Alto
	Executar testes	
Verificar um requisito funcional	Implementar testes	Alto
	Executar testes	
Verificar um requisito operacional	Implementar testes	Alto
	Executar testes	
Desenvolver documentação	Preparar documentação	Baixo
	Executar testes	
Lançar o produto	Administrar ambientes	Médio
	Elaborar plano de aceitação	
	Avaliar *release* interno de estabilização	
	Planejar disponibilização e instalação	
	Executar testes	
	Produzir artefatos de instalação	
	Instalar aplicação	
	Preparar documentação	
Gerenciamento de ocorrências	Monitorar status do projeto	Alto
	Controlar problemas	
Gerenciamento de Riscos		NA

Tabela 4.3 Relação entre fluxos de trabalho do MSF 4.0 e as atividades da Pro.NET

CAPÍTULO 5

Conclusões e trabalhos futuros

Este trabalho apresentou o MSF 4.0 e promoveu uma análise comparativa deste com a metodologia Pro.NET. Sua principal contribuição foi estabelecer um mapeamento inicial dos fluxos e atividades do MSF 4.0 com as macro-atividades e atividades da Pro.NET.

Como mostrou a análise comparativa apresentada, a Pro.NET atende a certos fluxos de trabalho do MSF 4.0 de maneira distribuída, isto é, muitas vezes atividades pertencentes a várias macro-atividades concorrem para a satisfação de um único fluxo de trabalho do MSF 4.0. Isto demonstra, em certo grau, que há uma diferença fundamental na maneira como as duas metodologias organizam suas atividades. Assim, uma adequação total da Pro.NET em relação ao MSF 4.0 acarretaria mudanças abruptas na metodologia. Sendo o MSF uma das bases da Pro.NET, é interessante que haja uma evolução neste sentido, mas por outro lado a estrutura da Pro.NET deveria ser preservada. Por tanto, talvez a melhor abordagem a ser adotada seja a de uma evolução independente do MSF 4.0, na qual haja a incorporação de aspectos que enriqueçam a metodologia, mas que não alterem a estrutura da Pro.NET. Sugestões como as apresentadas no capítulo anterior exemplificam alterações que aumentariam nível de conformidade da Pro.NET em relação ao MSF 4.0, mas não representariam mudanças bruscas na metodologia.

Por tratar-se de um material extremamente novo, a análise e compreensão do MSF 4.0 representou a principal dificuldade na elaboração deste trabalho. A não existência de literatura especializada sobre o mesmo, bem como a incompletude e inconsistências em sua documentação, foram sobremaneira responsáveis por grande parte do esforço despendido durante a elaboração da visão geral do MSF 4.0 (Capítulo 2) e da análise comparativa entra o MSF 4.0 e a Pro.NET (Capítulo 4).

A elaboração de trabalhos futuros tendo como base a análise aqui apresentada será parte fundamental da evolução da metodologia Pro.NET em direção ao MSF 4.0. É bastante provável que este trabalho seja utilizado pelo Centro de Tecnologia XML de Recife como ponto de partida para trabalhos que incluiriam:

- Revisar este trabalho em relação à versão final do MSF 4.0;
- Estender a análise aqui apresentada de maneira a abordar as sub-atividades do MSF 4.0;
- Elaborar uma análise comparativa que leve em consideração o aspecto temporal do MSF 4.0 e da Pro.NET;
- Elaborar uma análise comparativa entre os itens de trabalho do MSF 4.0 e os artefatos da Pro.NET;

- Analisar a viabilidade de uma possível integração entre a Pro.NET e o *Visual Studio Team System*.

Referências Bibliográficas

[BJR99] G. Booch, I. Jarcobson, and J. Rumbaugh. *The Unified Software Development Process*. Addison-wesley, 1999.

[Bro87] F. P. Brooks. No silver bullet: Essence and accidents of software engineering. *Computer Magazine*, April 1987.

[CKS03] Mary Beth Chrissis, Mike Konrad, and Sandy Shrum. *CMMI: Guidelines for Process Integration and Product Improvement*. Addison Wesley, February 2003.

[Coo99] Alan Cooper. *The Inmates Are Running the Asylum: Why High-Tech Products Drive Us Crazy and How to Restore the Sanity*. Sams, 1999.

[CTXa] CTXML. Site contendo toda a metodologia PRO.NET. Disponível em : http://www.cin.ufpe.br/∼ctxmlrec. Data de acesso: 18/08/2005.

[CTXb] CTXML. *Visão Geral da Metodologia Pro.NET*. Disponível em :http://www.cin.ufpe.br/∼ctxmlrec/SiteProNet/. Data de acesso: 18/02/2005.

[Gro01] Standish Group. *2001 update to the CHAOS report*, 2001. Disponível em: http://www.standishgroup.com/sample_research/. Data de acesso: 18/08/2005.

[Ins04] Software Engineering Institute. *Software Engineering Institute*, 2004. http://www.sei.cmu.edu/.

[Mar03] Suzana Maranhão. *Análise Comparativa entre a PRO.NET e o CMM nível 2*, August 2003.

[Mel04] Walcelio L. Melo. *Enhancing RUP for CMMI compliance: A methodological approach*, July 2004. Disponível em : http://www-128.ibm.com/developerworks/rational/library/5318.html#notes. Data de Acesso: 18/08/2005.

[MF97] K. Scott M. Fowler. *UML Distilled: applying the standard object modelling language*. Addison Wesley, 1997.

[Mica] Microsof. *MSF For CMMI Process Improvement*. Disponível em: http://lab.msdn.microsoft.com/teamsystem/workshop/msfcmmi/default.aspx. Data de acesso: 18/08/2005.

[Micb] Microsoft. *Visual Studio Team System*. Disponível em :http://lab.msdn.microsoft.com/teamsystem/. Data de acesso: 18/02/2005.

[Mic02a] Microsoft. *MSF Process Model v. 3.1*, June 2002. Disponível em :http://www.microsoft.com/technet/itsolutions/msf/default.mspx. Data de acesso: 18/08/2005.

[Mic02b] Microsoft. *MSF Team Model v. 3.1*, June 2002. Disponível em :http://www.microsoft.com/technet/itsolutions/msf/default.mspx. Data de acesso: 18/08/2005.

[Mic03] Microsoft. *Microsoft Solutions Framework version 3.0 Overview*, July 2003. Disponível em :http://www.microsoft.com/technet/itsolutions/msf/default.mspx. Data de acesso: 18/08/2005.

[MP03] Lisandra V. Manzoni and Roberto T. Price. Identifying extensions required by rup (rational unified process) to comply with cmm (capability maturity model) levels 2 and 3. *IEEE Trans. Softw. Eng.*, 29(2):181–192, 2003.

[NB68] P. Naur and Randell B., editors. *Proceedings of the NATO Conference on Software Engineering*, Garmish, Germany, October 1968. NATO Science Committee.

[NSK84] F & Tsuji S. N. Seraku Kano, Takahashi N. Attractive quality and must-be quality. *Journal of the Japanese Society for Quality Control*, 1(4):39–48, 1984.

[PCCW93] M. C. Paulk, B. Curtis, M. B. C. Chrissis, and C. Weber. Capability maturity model for software, version 1.1. Technical Report ADA263403, Software Engineering Institute, February 1993.

[Roy70] W. W. Royce. Managing the development of large software systems. In *Proceedings of IEEE WESCON*, August 1970.

Este volume foi tipografado em LaTeX na classe UFPEThesis (www.cin.ufpe.br/~paguso/ufpethesis).