ANDRÉ LUIZ DIAS RIBEIRO

**UM ROTEIRO PARA A REDUÇÃO DE TEMPO NO
DESENVOLVIMENTO DE PROJETOS DE SOFTWARE**

SÃO PAULO

2006

ANDRÉ LUIZ DIAS RIBEIRO

**UM ROTEIRO PARA A REDUÇÃO DO TEMPO NO
DESENVOLVIMENTO DE PROJETOS DE SOFTWARE**

Dissertação apresentada à Escola
Politécnica da Universidade de São
Paulo para a obtenção do título de
Mestre em Engenharia.

Área de Concentração:
Sistemas Digitais

Orientador:
Prof. Dr. Reginaldo Arakaki

SÃO PAULO

2006

ANDRÉ LUIZ DIAS RIBEIRO

**UM ROTEIRO PARA A REDUÇÃO DO TEMPO NO
DESENVOLVIMENTO DE PROJETOS DE SOFTWARE**

Dissertação apresentada à Escola
Politécnica da Universidade de São
Paulo para a obtenção do título de
Mestre em Engenharia.

Área de Concentração:
Sistemas Digitais

Orientador:
Prof. Dr. Reginaldo Arakaki

SÃO PAULO

2006

FICHA CATALOGRÁFICA

Ribeiro, Andre Luiz Dias

**Um roteiro para a redução de tempo no desenvolvimento de projetos de software / A.L.D. Ribeiro. -- São Paulo, 2006.
p. 124.**

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

**1.Engenharia de software 2.Desenvolvimento de software
I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.**

AGRADECIMENTOS

Agradeço a Deus pela vida e saúde em todas as minhas realizações.

Aos meus pais e irmãos pela base de conhecimento, perseverança e dignidade na realização de meus objetivos.

Ao amigo e orientador Reginaldo Arakaki pelo incentivo, apoio e diretrizes essenciais para esta realização.

A todos que, direta ou indiretamente, colaboraram na execução deste trabalho, em especial a amiga Cristina Coelho de Abreu Pinna e ao amigo Renato Manzan pelas contribuições, revisões e indicações.

À minha querida esposa Patrícia Salles Souza Ribeiro e às minhas filhas Andreza e Natália pelo companheirismo, compreensão e constante incentivo durante toda esta etapa de nossas vidas.

RESUMO

A realização de projetos dentro do prazo estabelecido é um fator comum em diversas áreas de produção como a engenharia civil, de aviação, química, transportes, indústria em geral, entre outras.

No entanto, na engenharia de software, a questão do tempo na construção de um produto é um desafio de processo a ser superado em cada novo projeto. O cumprimento de prazos no desenvolvimento de software é tão crítico que o próprio controle de atrasos no ciclo de produção é um fator a ser considerado na análise de redução do tempo de desenvolvimento.

A complexidade do ambiente de software, a competitividade de mercado, as mudanças de requisitos constantes durante o projeto e o tempo disponível cada vez mais restrito, aumentam as chances de insucesso quando analisado o indicador de tempo na produção de software.

O objetivo deste trabalho é reunir e organizar as práticas e técnicas de engenharia de software em um roteiro que permita a redução do tempo no desenvolvimento do software. Neste roteiro, é descrita a utilização organizada e planejada das práticas de engenharia de software que auxiliam no planejamento, na criação da arquitetura de solução, na definição da infra-estrutura técnica para reutilização e a utilização da engenharia simultânea, visando proporcionar ganhos reais no tempo de produção do software e no aumento da produtividade.

Palavras-Chaves: Engenharia de Software, Redução de Tempo, Arquitetura de Software, Gerenciamento de Projetos.

ABSTRACT

The completion of software project within schedule is a common goal in several industries like building engineering, aviation, chemical, transport, wares and so on.

However, in software engineering, the schedule is a process challenge from the beginning of each new project. The time is so critical that the delay control is an analysis point for cycle time reduction in software development.

The complex environment, the pressure to reduce time-to-market, frequent requirements changes during the project life-cycle, increase the failure chance of software projects when we analyze the time indicator in the software development process.

The dissertation goal is to meet and to organize of software engineering practices and techniques in an organized roadmap aiming cycle time reduction in software development. In this roadmap, the practices are organized to help software planning, solution architecture, component based development definition, to promote reuse and concurrent engineering with purpose to reduce cycle time software development and improve productivity.

Keywords: Software Engineering, Cycle Time Reduction, Software Architecture, Project Management.

LISTA DE FIGURAS

FIGURA 1.1 – REPRESENTAÇÃO GRÁFICA DA ORGANIZAÇÃO DO TRABALHO	19
FIGURA 2.1 – GRUPO DE PROCESSOS DE PLANEJAMENTO DO PMBOK (PMI, 2004)	25
FIGURA 2.2 – INTERAÇÕES ENTRE OS PROCESSOS DE CONTROLE DO PMBOK (PMI, 2004).....	26
FIGURA 2.3 – ATIVIDADES DE PLANEJAMENTO E CONTROLE DE PROJETO (KERZNER, 2003)	28
FIGURA 2.4 – MODELO CASCATA	30
FIGURA 2.5 – MODELO PROTOTIPAÇÃO EVOLUTIVA (McCONNELL, 1996)	31
FIGURA 2.6 – MODELO ESPIRAL DE BOEHM (PRESSMAN, 2005)	31
FIGURA 2.7 – MODELO ITERATIVO (PRESSMAN, 2005)	33
FIGURA 3.1 – PAPEL DA ARQUITETURA DE SOFTWARE (GARLAN, 2000)	37
FIGURA 3.2 – ATIVIDADES DO PROJETO DE ARQUITETURA (HOFMEISTER ET AL., 2005).....	39
FIGURA 3.3 – DESENVOLVIMENTO <i>TIMEBOX</i> (McCONNELL, 1996)	42
FIGURA 3.4 – CICLO DE PROTOTIPAÇÃO ADAPTADO (LUQI, 1989)	44
FIGURA 3.5 – REPRESENTAÇÃO DE UM PARTICIONAMENTO.....	45
FIGURA 3.6 – DESENVOLVIMENTO BASEADO EM COMPONENTES (PRESSMAN, 2005)	47
FIGURA 3.7 – OBJETIVOS DA ENGENHARIA SIMULTÂNEA (DWIVEDI ET AL., 1990).....	53
FIGURA 4.1 – FATORES DE REDUÇÃO DE TEMPO (BLACKBURN; SCUDDER; WASSENHOVE, 2000)	59
FIGURA 4.2 – ALOCAÇÃO DE TEMPO POR FASE (BLACKBURN; SCUDDER; WASSENHOVE, 2000)	60
FIGURA 5.1 – PRÁTICAS DE DESENVOLVIMENTO EM ENGENHARIA DE SOFTWARE (MCT,2002).....	65
FIGURA 5.2 – PRÁTICAS DE GESTÃO DE PROJETOS EM ENGENHARIA DE SOFTWARE (MCT, 2002)	66
FIGURA 5.3 – UTILIZAÇÃO DAS PRÁTICAS DE ENGENHARIA DE SOFTWARE DO ROTEIRO PROPOSTO	70
FIGURA 6.1 – MAPEAMENTO DOS CONCEITOS QUE IMPACTAM NA REDUÇÃO DE TEMPO.....	77
FIGURA 6.2 – MAPEAMENTO DOS ELEMENTOS QUE COMPÕEM O ROTEIRO PROPOSTO	79
FIGURA 6.3 – FLUXO DE ATIVIDADES DO ROTEIRO.....	87

LISTA DE TABELAS

TABELA 2.1 – PARÂMETROS DE CAPACIDADE DOS CICLOS DE VIDA ADAPTADO (McCONNELL, 1996)	33
TABELA 3.1 – COMPARAÇÃO DAS ESTRATÉGIAS DE REUSO (RAVICHANDRAN E ROTHENBERGER, 2003).....	50
TABELA 3.2 – ELEMENTOS CHAVES DA ENGENHARIA SIMULTÂNEA ADAPTADO (KARANDIKAR <i>ET AL</i> , 1993).....	53
TABELA 4.1 – OBJETIVOS PARA REDUÇÃO DO CICLO DE DESENVOLVIMENTO (COLLIER; COLLOFELLO, 1995).	58
TABELA 5.1 – PRÁTICAS ENGENHARIA DE SOFTWARE – SEPIN – (MCT, 2002).....	64
TABELA 5.2 – PERFIL DE QUALIDADE DAS EMPRESAS PARTICIPANTES DA ENQUETE	69
TABELA 5.3 – PRÁTICAS DE GERENCIAMENTO DE PROJETOS	69
TABELA 5.4 – MODELOS DE CICLOS DE VIDA UTILIZADOS.....	70
TABELA 5.5 – NÍVEL DE NÃO APLICAÇÃO DAS PRÁTICAS DE ENGENHARIA DE SOFTWARE ...	71
TABELA 5.6 – RESULTADO DA APLICAÇÃO DA PRÁTICA DE PROTOTIPAÇÃO	71
TABELA 5.7 – RESULTADO DA APLICAÇÃO DAS PRÁTICAS DE COMPONENTIZAÇÃO E REUSO	72
TABELA 5.8 – RESULTADO DA APLICAÇÃO DA PRÁTICA DE ENGENHARIA SIMULTÂNEA	72
TABELA 6.1 – QUADRO COMPARATIVO ENTRE AS ABORDAGENS DE REDUÇÃO DE TEMPO ...	75
TABELA 6.2 – COMPARATIVO ENTRE OS OBJETIVOS DAS ABORDAGENS DE REDUÇÃO DE TEMPO	76
TABELA 6.3 – TÉCNICAS E PRÁTICAS QUE CONTRIBUEM PARA A REDUÇÃO DE TEMPO.....	76
TABELA 6.4 – MAPEAMENTO DAS PRÁTICAS E TÉCNICAS COM OS OBJETIVOS DE REDUÇÃO DE TEMPO.....	78
TABELA 6.5 – PONTOS FORTES E PONTOS FRACOS NO USO DA ENGENHARIA SIMULTÂNEA.....	85
TABELA 6.6 – PRODUTOS RESULTANTES DA APLICAÇÃO DO ROTEIRO	99
TABELA 7.1 – RESULTADOS OBTIDOS COM A APLICAÇÃO DO ROTEIRO NO PROJETO 1	104
TABELA 7.2 – RESULTADOS OBTIDOS COM A APLICAÇÃO DO ROTEIRO NO PROJETO 2	106
TABELA 7.3 – AÇÕES CORRETIVAS PARA TRATAR AS DIFICULDADES ENCONTRADAS COM A EQUIPE DE PROJETO	109
TABELA 7.4 – AÇÕES CORRETIVAS PARA TRATAR AS DIFICULDADES ENCONTRADAS COM O CLIENTE	110

SUMÁRIO

1 INTRODUÇÃO	13
Objetivo do Capítulo	13
1.1. O Problema do Tempo no Desenvolvimento de Software	13
1.2. Motivação	15
1.3. Justificativa do Trabalho.....	16
1.4. Objetivos do Trabalho	18
1.5. Organização do Trabalho	18
1.6. Notações e Convenções Utilizadas.....	20
2 O PLANEJAMENTO DE PROJETOS DE SOFTWARE.....	21
Objetivo do Capítulo	21
2.1. A Importância do Planejamento de Projetos.....	21
2.2. Monitoramento e Controle do Projeto	25
2.3. Planejamento do Ciclo de Vida para Desenvolvimento do Software.....	29
2.4. Conclusões do Capítulo	34
3 CONCEITOS DAS TÉCNICAS E PRÁTICAS DE ENGENHARIA DE SOFTWARE	35
Objetivo do Capítulo	35
3.1. A Arquitetura de Software.....	36
3.2. Desenvolvimento <i>Timebox</i>	41
3.3. Prototipação do Sistema	43
3.4. Particionamento do Software.....	44
3.5. Desenvolvimento Baseado em Componentes e Reutilização	45
3.6. Engenharia Simultânea.....	51
3.7. Conclusões do Capítulo	54

4 ABORDAGENS PARA A REDUÇÃO DE TEMPO NO DESENVOLVIMENTO	
SOFTWARE	55
Objetivo do Capítulo	55
4.1. Principais Iniciativas para a Redução do Ciclo de Desenvolvimento de Software	55
4.2. Conclusões do Capítulo	62
5 AVALIAÇÕES SOBRE A UTILIZAÇÃO DAS PRÁTICAS E TÉCNICAS DE ENGENHARIA DE SOFTWARE NO BRASIL	63
Objetivo do Capítulo	63
5.1. Pesquisa sobre Qualidade e Produtividade no Setor de Software Brasileiro	63
5.2. Enquete para Verificação da Utilização das Práticas de Engenharia de Software	66
5.3. Conclusões do Capítulo	72
6 O ROTEIRO PARA REDUÇÃO DE TEMPO NO DESENVOLVIMENTO DE SOFTWARE.....	74
Objetivo do Capítulo	74
6.1. Definição da Estrutura do Roteiro para a Redução de Tempo.....	74
6.2. Os Elementos do Roteiro para Redução de Tempo no Desenvolvimento de Software	79
6.3. Fluxo de Atividades para a Aplicação do Roteiro para Redução do Tempo de Desenvolvimento de Software	86
6.4. Conclusões do Capítulo	100
7 APLICAÇÃO DO ROTEIRO EM PROJETOS DE SOFTWARE.....	101
Objetivo do Capítulo	101
7.1. Critérios de Escolha do Projeto.....	101
7.2. Características dos Projetos e Condições de Aplicação	101
7.3. Resultados da Aplicação do Roteiro nos Projetos.....	103
7.4. Dificuldades Encontradas na Aplicação do Roteiro.....	107
7.5. Conclusões do Capítulo	110

8 CONSIDERAÇÕES FINAIS.....	111
Objetivo do Capítulo	111
8.1. Resumo das Contribuições do Trabalho	111
8.2. Conclusões do Trabalho	113
8.3. Trabalhos Futuros	114
REFERÊNCIAS	116
ANEXO 1 - QUESTIONÁRIO APLICADO NA ENQUETE	121

1 INTRODUÇÃO

Objetivo do Capítulo

O objetivo deste capítulo é apresentar os motivos que levaram ao desenvolvimento deste trabalho, destacando sua relevância e objetivos. Também é apresentada a metodologia utilizada e a estruturação da dissertação.

1.1. O Problema do Tempo no Desenvolvimento de Software

Com o avanço da tecnologia e com o aumento da competitividade no mercado, os clientes estão se tornando cada vez mais exigentes em relação ao aumento da qualidade do software e à redução do tempo de desenvolvimento.

Além da exigência do cliente e da demanda do mercado, a adequação às diretrizes e normas governamentais que alteram os procedimentos e regras de mercado e que necessitam de soluções rápidas e dentro do prazo definido é mais um fator relevante a ser considerado na análise do tempo de produção de software.

Diversas pesquisas realizadas em âmbito internacional mostram que mais da metade dos projetos de software resultaram em insucesso, apresentando problemas em relação ao escopo, prazo, custo e a qualidade. Em pesquisa realizada pelo Standish Group (2004), a questão do tempo foi classificada como um dos cinco maiores fatores de insucesso dos projetos, sendo responsável por dez por cento dos problemas no universo pesquisado. Essa proporção é estável no decorrer do tempo.

Algumas dificuldades apontadas por McConnell (1996) como causas de insucesso no processo de desenvolvimento de software ainda persistem nos dias atuais. Os estudos recentes do Standish Group (2004) mostram que a maioria das empresas de desenvolvimento de software continuam enfrentando problemas na implementação de soluções que eliminem ou minimizem seus efeitos na produção de software. Entre estas dificuldades destacam-se as seguintes:

- 1) Falta de motivação da equipe;
- 2) Planejamento deficiente;
- 3) Abandono do planejamento original em função da pressão do tempo;
- 4) Erro nas estimativas de duração das atividades;
- 5) Características necessárias ao projeto excedidas;
- 6) Falha no contrato comercial.

A alternativa mais utilizada pelas empresas para eliminar os efeitos destas dificuldades é a aplicação de normas e modelos de qualidade de software que descrevem diretrizes, processos e guias de avaliação, visando a aplicação de técnicas e práticas de engenharia para a obtenção de melhoria em seus processos. As principais características e objetivos de algumas dessas normas e modelos são descritas a seguir.

A ISO IEC 12207 é uma norma que tem como objetivo descrever os processos necessários ao ciclo de vida de desenvolvimento de software, bem como aumentar o controle e permitir melhorias no ciclo de produção. É constituída por três processos principais: os processos fundamentais que descrevem a execução do processo de desenvolvimento, os processos de apoio que descrevem os procedimentos para a melhoria da qualidade do produto e os processos organizacionais que descrevem os procedimentos de gerenciamento e melhoria do processo de desenvolvimento.

A Norma NBR 13596 (ISO IEC 9126) – Tecnologia da Informação - Características e Métricas de Qualidade de Software – descreve os aspectos de qualidade de software referentes à funcionalidade, confiabilidade, usabilidade, eficiência, manutenção e portabilidade na visão do usuário sobre a avaliação da qualidade do produto. Essa norma deve ser aplicada em conjunto com a Norma ISO IEC 14598 que é um guia que orienta o planejamento e execução do processo de avaliação da qualidade e têm como objetivo principal fornecer resultados de qualidade dos produtos desenvolvidos sob a ótica de aceitação pelos envolvidos.

O modelo de software CMMI (*Capability Maturity Model Integration*), desenvolvido pelo *Software Engineering Institute* (SEI/CMU), é um modelo de referência para melhoria da capacidade de processo das empresas abordando várias disciplinas que permitem a melhoria na qualidade do produto, aumento da produtividade, maior controle sobre o prazo e orçamento e aumento da satisfação do cliente.

Embora estes trabalhos forneçam procedimentos e processos que permitam melhorias no desenvolvimento de software, os resultados apresentados pelo mercado de produção de software mostram que muitas das recomendações não são aplicadas de maneira adequada e que a maioria dos problemas encontrados nos projetos estão de alguma forma relacionados à pressão que o fator tempo impõe aos projetos. Como resultados destas dificuldades têm-se, ao final do projeto, o prazo comprometido, custos excedidos, baixa qualidade e confiabilidade do produto e a insatisfação do cliente.

Baseado nessas informações é possível concluir que se torna necessário organizar e aplicar as técnicas de engenharia de software de forma conjunta e iterativa, visando reduzir o ciclo de vida de desenvolvimento do produto para atender o requisito de tempo sem afetar os demais requisitos do projeto.

1.2. Motivação

A realização de projetos dentro do prazo estabelecido é um fator comum em diversas áreas de produção como a engenharia civil, de aviação, química, transportes, indústria em geral, entre outras.

No entanto, na engenharia de software, a questão do tempo de construção de um produto é um desafio de processo a ser superado em cada novo projeto. O cumprimento de prazos no desenvolvimento de software é tão crítico que, além do objetivo de execução do projeto dentro do prazo, o próprio controle de atrasos no ciclo de produção é um fator a ser considerado na análise de redução do tempo de desenvolvimento. A complexidade do ambiente de software, a competitividade de mercado, as mudanças de requisitos constantes durante o projeto e o tempo disponível cada vez mais restrito aumentam as chances de insucesso quando analisado o indicador de tempo na produção de software.

A agilidade e rapidez com que os ambientes de produção de software evoluem atualmente tornam o fator de redução do tempo no ciclo de vida do desenvolvimento um diferencial para as empresas de desenvolvimento de sistemas.

Outro fator relevante para o desenvolvimento de software é que, embora as práticas e técnicas de engenharia de software sejam amplamente divulgadas e de

conhecimento da maioria das empresas de desenvolvimento, estas não utilizam ou utilizam-nas de forma isolada não obtendo os benefícios que estas poderiam proporcionar durante a construção dos sistemas. Algumas causas relacionadas a este cenário podem ser descritas através dos seguintes itens (Ribeiro; Arakaki, 2005):

- 1) Falta de investimentos em tecnologia;
- 2) Treinamento deficiente;
- 3) Descrença na efetividade da utilização das técnicas;
- 4) Falta de domínio técnico das equipes de projeto;
- 5) Aplicação inadequada das práticas e técnicas;
- 6) Falta de recursos financeiros.

A motivação deste trabalho é mostrar que as técnicas de engenharia de software utilizadas em conjunto, de forma organizada e interdependente permitem a redução do tempo de desenvolvimento, proporcionando ganhos reais de produtividade na execução dos projetos, ajudando as empresas atingirem o prazo estabelecido.

1.3. Justificativa do Trabalho

O aumento da produtividade e a redução do tempo de desenvolvimento de software são os principais objetivos das empresas responsáveis pela construção de aplicações (Clincy, 2003). Estes objetivos visam reduzir os custos, melhorar os serviços prestados aos clientes e obter vantagens competitivas no mercado de produção de software (Wetherbe; Frolick, 2000).

Analisando o estado da arte sobre a redução de tempo de desenvolvimento encontra-se dois modelos cujos esforços são relevantes até os dias atuais: o Desenvolvimento Rápido de Aplicação (RAD - *Rapid Application Development*) proposto por Martin em 1991 e o Desenvolvimento Rápido Evolutivo (*Rapid Evolutionary Development*) proposto por Arthur em 1992.

O RAD (*Rapid Application Development*) possui cinco pontos focais: equipes reduzidas, prototipação evolutiva, uso de ferramentas CASE, *Joint Application Design* (JAD) e rígidos limites de tempo no desenvolvimento de software (*timebox*).

Este trabalho é muito significativo para a área por servir de base para outros esforços sobre a redução de tempo de desenvolvimento.

O Desenvolvimento Rápido Evolutivo (*Rapid Evolutionary Development*) é baseado no desenvolvimento evolutivo tradicional aliado ao forte uso da prototipação, o que significa interação com o cliente e administração das alterações de requisitos. Os benefícios deste modelo são: a redução dos riscos e a produção de resultados tangíveis em tempo reduzido.

Outros trabalhos significantes nesse tema tratam alguns aspectos importantes como a diminuição dos canais de comunicação através de equipes pequenas (até 15 pessoas), a redução do nível de comunicação formal no projeto e a produção da documentação mínima necessária para o projeto como fatores que auxiliam na redução do tempo de desenvolvimento do software (Clyncy, 2003).

Collier e Collofello (1995) descrevem que a questão de redução de tempo está relacionada às características ambientais e do produto a ser construído. As características ambientais incluem o modelo de processo, a disponibilidade de recursos, métodos, procedimentos e as práticas utilizadas durante o desenvolvimento do software. As características do produto estão relacionadas ao grau de complexidade, a independência entre os subsistemas e o tamanho do software a ser construído.

Em pesquisa realizada entre 1996 e 1999 com empresas de desenvolvimento de software nos Estados Unidos, Europa e Japão, Blackburn, Scudder e Wassenhove (2000) identificaram algumas técnicas e práticas de engenharia que auxiliam na redução de tempo de desenvolvimento. Entre estas, se destacam a engenharia simultânea, o desenvolvimento baseado em componentes, a reutilização e a prototipação como os mais efetivos nos ganhos de tempo na produção de software.

Segundo Howard (2002), existem muitas discussões sobre os efeitos e resultados dessas abordagens, desde a efetividade dos modelos em projetos maiores, questões entre a incompatibilidade de qualidade com desenvolvimento rápido até a questão de manutenção dos custos dos projetos. Nessas vertentes, há aqueles que utilizam e acreditam na materialização dos resultados tangíveis dos modelos de redução de tempo e aqueles que permanecem relutantes em adotá-los por questões de aversão ao risco ou alteração da zona de conforto na qual se encontram.

Sobre esse ambiente e abordagens citadas para redução de tempo no desenvolvimento de software foram mapeados e organizados os conceitos deste trabalho.

1.4. Objetivos do Trabalho

O principal objetivo deste trabalho é elaborar um roteiro que reúna e organize as práticas e técnicas de engenharia de software para permitir a redução de tempo no desenvolvimento de software.

No roteiro deste trabalho é descrita a utilização organizada e planejada das práticas de engenharia de software que auxiliam no planejamento, na definição da arquitetura do software, no estabelecimento do desenvolvimento baseado em componentes e reutilização e na engenharia simultânea que, aplicadas de forma conjunta, proporcionam ganhos reais na redução do tempo de produção de software e o aumento da produtividade.

1.5. Organização do Trabalho

Este trabalho foi organizado em sete capítulos conforme representação esquemática da Figura 1.1.

No **Capítulo 1** são apresentados os motivos que levaram o desenvolvimento deste trabalho, sua justificativa, seus objetivos, a estruturação do trabalho e as convenções utilizadas.

No **Capítulo 2** é descrita a importância do planejamento no desenvolvimento de software, o processo de monitoramento e controle durante a execução do projeto, bem como uma definição dos ciclos de desenvolvimento que deve ser definido no início do projeto.

No **Capítulo 3** concentram-se a fundamentação teórica dos conceitos das técnicas e práticas relacionadas ao roteiro como a arquitetura de software, a prototipação evolutiva, o desenvolvimento *timebox*, particionamento do sistema, componentização e engenharia simultânea que são partes integrantes do roteiro proposto.

No **Capítulo 4** são apresentadas as principais iniciativas existentes sobre a redução de tempo no desenvolvimento de software, seus conceitos, características e focos principais.

O **Capítulo 5** apresenta um panorama nacional sobre as práticas de Engenharia de Software adotadas pelas empresas no Brasil realizadas pela Secretaria Especial de Informática MCT (2002) e o resultado de uma enquete realizada pelo autor para verificação do grau de utilização das práticas de engenharia de software propostas pelo roteiro no mercado brasileiro.

No **Capítulo 6** é apresentado o roteiro para redução de tempo no desenvolvimento de software descrevendo sua estruturação, os elementos que o compõe e as atividades a serem realizadas na sua aplicação, seus produtos, aspectos de utilização e suas restrições.

No **Capítulo 7** são descritos os resultados da aplicação do roteiro em projetos de software vivenciados pelo autor e as principais dificuldades observadas durante a sua utilização.

No **Capítulo 8** são apresentadas as considerações finais com a descrição das conclusões, as contribuições do trabalho e os trabalhos futuros que podem ser desenvolvidos a partir deste estudo.

Figura 1.1 – Representação gráfica da organização do trabalho

1.6. Notações e Convenções Utilizadas

As tabelas e figuras que não possuem referência foram desenvolvidas pelo autor utilizando diagramas de classes e de atividades da linguagem UML – *Unified Modeling Language* (OMG, 2006) ou representações clássicas de fluxograma.

2 O PLANEJAMENTO DE PROJETOS DE SOFTWARE

Objetivo do Capítulo

O objetivo deste capítulo é descrever a importância do planejamento no desenvolvimento de software, apresentar os conceitos do processo de controle durante a execução do projeto e mostrar os conceitos dos diversos ciclos de vida que podem ser aplicados na produção de software, visando criar subsídios para a definição do ciclo de vida adequado a cada projeto.

Este capítulo foi desenvolvido com base no estudo dos seguintes trabalhos:

Boehm; Turner(2004), McConnell(1996), Bersoff; Davis(1991), Howard(2002), Pinna(2004), Kirk(2004), Krutchen(2000), Pressman(2005), PMI(2004), Kerzner (2003).

2.1. A Importância do Planejamento de Projetos

O planejamento é uma das principais atividades a ser realizada na concepção de um sistema de software (Kerzner, 2003), pois é com base na análise dos requisitos do produto a ser construído, nas suas premissas e restrições que o desenvolvimento do software é adequado ao tempo disponível e ao custo estabelecido. Além disso, é na fase de planejamento que novas dependências, requisitos, riscos e oportunidades são identificados e resolvidos pela equipe do projeto.

A inexistência ou a falta de um planejamento adequado podem trazer diversas conseqüências a um projeto, contribuindo para o insucesso do mesmo. Segundo Kerzner (2003), cinco itens se destacam como resultado dessa deficiência:

1. Início do projeto sem a definição dos requisitos;
2. Entusiasmo excessivo da equipe do projeto;

3. Decepção com os resultados não alcançados;
4. Procura por culpados pelo fracasso e punição de inocentes;
5. Premiação e reconhecimento de não participantes no projeto.

Para evitar estas conseqüências, deve ser elaborado um planejamento adequado à realidade de cada projeto, com a definição clara dos requisitos do projeto, onde a mensuração inicial de esforço e de prazo é o ponto de partida para viabilizar a execução do projeto dentro do tempo disponível.

O planejamento pode ser elaborado de forma detalhada, total ou parcial, de acordo com as características do projeto a ser desenvolvido e do método de gestão de projetos que se utiliza (Kirk, 2004). No entanto, segundo o PMBOK (PMI, 2004), o planejamento deve ser divulgado a todos os envolvidos e formalmente aprovado pelo cliente.

Para atender essas necessidades podem ser realizadas reuniões direcionadas com a presença de todos os envolvidos com o objetivo de discutir e aprovar todas as questões relevantes, permitindo o alinhamento antecipado das expectativas sobre o produto e a melhoria dos canais de comunicação (Boehm; Turner, 2004).

Na impossibilidade de elaborar o planejamento conjunto, deve-se promover reuniões de *kickoff* através da qual todos os envolvidos são informados do plano elaborado pela equipe do projeto, criando condições de avaliação e discussão sobre o planejamento apresentado. Esse plano deve ser ajustado durante a execução do projeto sob as mesmas condições de aprovação de todos os envolvidos..

Segundo Kerzner (2003), o planejamento do projeto deve ser sistemático, flexível, disciplinado por revisões e controles e capaz de aceitar e incorporar mudanças durante a realização do projeto.

Independente da abordagem utilizada pelas equipes de projeto, uma referência comum para a elaboração do planejamento de projetos é o PMBOK®, *Guide of the Project Management Body of Knowledge* (PMI, 2004). Este guia especifica os grupos de processos de iniciação, planejamento, execução, controle e fechamento para a realização de um projeto.

A seguir são descritos resumidamente os processos de planejamento de acordo com o PMBOK (PMI, 2004).

1 – Planejamento do Escopo: Descreve como o escopo do projeto deve ser definido, verificado e controlado;

2 – Definição do Escopo: Fornece o detalhamento do escopo que serve de *baseline* de requisitos;

3 – Elaboração da EAP (Estrutura Analítica do Projeto): É a estrutura que define os principais produtos de trabalho do projeto. Serve como base para a execução dos processos seguintes;

4 – Definição das Atividades: Identifica as principais atividades necessárias para a construção dos produtos de trabalho;

5 – Seqüenciamento das Atividades: Identifica as dependências entre as atividades definidas;

6 - Estimativa de Recursos das atividades: Estima a quantidade e o tipo de recursos necessários para a realização de cada atividade;

7 – Estimativa de Duração das Atividades: Estima o tempo necessário para realização de cada atividade;

8 – Desenvolvimento do Cronograma: Analisa os itens de 4 a 7 para criação do cronograma do projeto;

9 – Estimativa de Custos: Desenvolve os custos aproximados para a realização das atividades;

10 – Elaboração do Orçamento: Agrega as estimativas individuais de custo das atividades para definir o custo total do projeto;

11 – Planejamento da Qualidade: Identifica quais são os padrões de qualidades e como são atendidos;

12 – Planejamento de Recursos Humanos: Identifica os papéis e responsabilidades da equipe do projeto;

13 – Planejamento das Comunicações: Determina como serão as comunicações entre todos os envolvidos no projeto;

14 – Planejamento de Riscos: Define como os riscos serão gerenciados durante o projeto;

15 – Identificação de Riscos: Determinam quais são os riscos que afetam o projeto e documenta suas características;

16 – Análise Qualitativa de Riscos: Realiza a priorização dos riscos e avalia sua probabilidade de ocorrer e seu impacto sobre o projeto;

17 – Análise Quantitativa de Riscos: Analisa numericamente os efeitos dos riscos sobre o projeto;

18 – Elaboração do Plano de Respostas aos Riscos: Desenvolve as ações para reduzir, eliminar ou transferir os riscos do projeto;

19 – Elaboração do Plano de Terceirização: Determina o quê, quando e como adquirir recursos externos necessários ao projeto;

20 – Plano de Contratação: Define e documenta os produtos que serão contratados e identifica os potenciais fornecedores.

Os processos mostrados na Figura 2.1 podem ser executados tantas vezes quantas forem necessárias, dentro de cada fase do projeto (PMI, 2004), pois são iterativos e devem ser realizados durante todo o ciclo de vida do projeto. Entretanto, os processos são referências para a elaboração do planejamento, não sendo necessária a execução de todos os processos para a conclusão do planejamento.

Segundo Kerzner (2003), um planejamento sistemático e aderente à realidade de cada projeto permite às organizações atingirem os objetivos e contribui para:

- Eliminar e reduzir as incertezas;
- Obter melhor entendimento dos objetivos;
- Prover a base para o monitoramento e controle do trabalho;
- Melhorar a eficiência da operação.

Segundo o PMBOK (2004), o grupo de processos de monitoramento e controle tem como objetivos identificar problemas, permitir a tomada de ações corretivas durante a execução do projeto, controlar as mudanças e recomendar ações preventivas.

Figura 2.2 – Interações entre os Processos de Controle do PMBOK (PMI, 2004)

As atividades de monitoração e controle realizadas durante o desenvolvimento do projeto consistem em:

- Acompanhar o progresso das tarefas;
- Comparar os resultados atuais com os previstos no planejamento;
- Analisar o impacto das alterações que ocorrem durante o projeto;
- Realizar os ajustes necessários para manter as metas do projeto.

Para a realização dessas atividades, os seguintes processos são previstos pelo PMBOK (2004) para o controle e monitoramento do projeto e estão ilustrados na Figura 2.2.

1 – Monitorar e Controlar o Projeto: É o processo responsável por coletar, medir e disseminar informações sobre o desempenho e avaliar as tendências para efetuar as melhorias no processo;

2 – Controle Integrado das Mudanças: Controla os fatores que criam mudanças, determina se ocorreu uma mudança e gerencia as mudanças aprovadas. É realizado durante todo o projeto;

3 – Verificar o Escopo: Formaliza a aceitação das entregas de produtos realizadas;

4 – Controlar o Escopo: Controla as mudanças feitas no escopo do projeto;

5 – Controlar o Cronograma: Controla as alterações realizadas que impactam no cronograma do projeto;

6 – Controlar os Custos: Controla e monitora as variações no orçamento do projeto;

7 – Realizar o Controle da Qualidade: Monitora as entregas para determinar se estão de acordo com os padrões de qualidade;

8 – Gerenciar a Equipe do Projeto: Acompanha o desempenho dos membros da equipe, fornece *feedback* e coordena mudanças para melhorar o desempenho do projeto;

9 – Elaborar Relatórios de Desempenho: Coleta e distribui informações sobre o andamento do projeto;

10 – Gerenciar os *Stakeholders*: Gerencia a comunicação a fim de satisfazer os requisitos e resolver problemas com os *stakeholders*;

11 – Monitorar e Controlar os Riscos: Acompanha os riscos, identifica novos riscos, executa o plano de respostas aos riscos e avalia sua eficiência durante todo o projeto;

12 – Administrar Contratos: Gerencia o contrato quando o projeto envolve relação entre comprador e fornecedor.

Embora existam doze processos de controle, para cada projeto devem ser definidos quais desses processos devem ser aplicados no ciclo de desenvolvimento do software, considerando suas características específicas e o ambiente da organização que o desenvolve.

Um processo de planejamento e controle resumido é apresentado por Kerzner (2003) para se estabelecer um nível adequado de controle sobre o projeto, onde as seguintes atividades mínimas devem ser realizadas durante sua execução e são ilustradas na Figura 2.3.

As caixas superiores representam as atividades de planejamento e as caixas inferiores identificam as atividades de controle.

Figura 2.3 – Atividades de Planejamento e Controle de Projeto (Kerzner, 2003)

A partir da definição das metas e objetivos são definidos os pacotes de trabalho da Estrutura Analítica do Projeto (EAP) e o ciclo de desenvolvimento a ser utilizado durante a execução do projeto. As atividades são determinadas para a realização de cada pacote de trabalho e o cronograma e o orçamento são elaborados para servir de base para o controle do progresso do projeto.

O acompanhamento do projeto é realizado através do monitoramento das atividades em relação ao cumprimento dos prazos, gastos planejados versus o realizado e a

qualidade dos produtos a serem entregues ao cliente. Os relatórios de desempenho são utilizados para verificar a aderência ao planejamento original e corrigir possíveis desvios, gerando replanejamentos que se tornam a nova referência para o controle e monitoramento do projeto.

Este processo representa um fluxo contínuo e repetitivo de atividades que devem ocorrer durante o ciclo de vida de um projeto até a sua efetiva conclusão.

2.3. Planejamento do Ciclo de Vida para Desenvolvimento do Software

A definição do ciclo de desenvolvimento de um projeto faz parte da fase de planejamento e sua definição, como qualquer ação tomada nesta fase, pode determinar o grau de sucesso ou insucesso do projeto.

A definição do modelo de processo do ciclo de vida orienta o processo de desenvolvimento do software e determinar os critérios de avaliação da evolução do projeto. A escolha apropriada do modelo pode aumentar a velocidade do desenvolvimento, melhorar a qualidade, facilitar o acompanhamento e controle, minimizar os riscos ou melhorar a relação com o cliente. Enquanto a escolha errada do modelo pode gerar baixa produtividade, retrabalho, insatisfação do cliente e a realização de atividades desnecessárias (McConnell, 1996).

Existem muitos modelos de ciclos de vida para escolha e aplicação pelas equipes de desenvolvimento de software. Entretanto, para este trabalho são apresentadas as principais características dos quatro modelos mais utilizados atualmente: o modelo cascata, a prototipação evolutiva, o modelo espiral e o modelo iterativo.

2.3.1) Modelo Cascata

É o modelo mais antigo e o mais conhecido atualmente, sendo utilizado como referência para o desenvolvimento dos demais modelos. É denominado como modelo clássico.

O modelo consiste na execução sequencial das atividades de “especificar-projetar-construir” (Howard, 2002), onde ao final de cada fase são realizadas avaliações para determinar se o projeto passa para a próxima fase ou

permanece na fase atual. Outra característica é que os documentos produzidos em cada fase são subsídios para a fase subsequente. Esse modelo é apresentado na Figura 2.4.

O modelo cascata é adequado para projetos com baixo nível de alteração de requisitos e que não necessitam de resultados visíveis em curto prazo, pois os produtos são gerados ao final do modelo (McConnell, 1996).

Figura 2.4 – Modelo Cascata

2.3.2) Prototipação Evolutiva

É um modelo baseado no desenvolvimento de um protótipo como fonte para a identificação das necessidades do cliente e validação dos requisitos nas fases iniciais de desenvolvimento do software. Os protótipos são construídos e apresentados aos clientes para avaliação e continuam neste ciclo até a aprovação final pelo cliente, quando o produto final é construído.

Tornou-se muito popular por reduzir os custos e os riscos nas fases iniciais do projeto (Bersoff; Davis, 1991) e atualmente, é usado em conjunto com outros modelos auxiliando também, na redução do tempo de desenvolvimento do *software*.

A Figura 2.5 apresenta graficamente o modelo de prototipação evolutiva.

Figura 2.5 – Modelo Prototipação Evolutiva (McConnell, 1996)

Este modelo apresenta ainda como vantagens a melhoria da comunicação entre o cliente e a equipe do projeto, flexibilidade de adequação em cenários de constantes alterações de requisitos e proporcionar ao cliente uma visão funcional do sistema a baixo custo.

2.3.3) Modelo Espiral

O modelo espiral é um modelo orientado a riscos desenvolvido por Boehm em 1988, incorporando as melhores características do modelo clássico e da prototipação, dividindo-se em quatro atividades conforme ilustrado na Figura 2.6.

Figura 2.6 – Modelo Espiral de Boehm (Pressman, 2005)

A atividade de planejamento consiste na determinação dos objetivos, alternativas e restrições para a realização do projeto. A identificação e análise dos riscos são realizadas a seguir visando determinar e tratar os possíveis problemas nas fases iniciais do desenvolvimento e servirem como referência para monitoração durante todo o ciclo de vida. Após estas atividades inicia-se o desenvolvimento de uma versão do produto que, após a conclusão, é submetido à avaliação do cliente.

A cada iteração ao redor da espiral uma versão mais completa do software é construída (Pinna, 2004). Os requisitos vão sendo detalhados e implementados a cada iteração e, ao final de cada iteração são realizadas análises de tomada de decisão sobre o andamento do projeto.

A principal vantagem deste modelo é o caráter evolutivo e os resultados tangíveis que são alcançados à medida que o projeto evolui.

2.3.4) Modelo Iterativo

É o modelo desenvolvido a partir do modelo espiral, caracterizado pelo desenvolvimento em vários ciclos, entregas constantes de produtos e pela análise de riscos, permitindo a identificação de problemas nas fases iniciais do projeto e a tomada de ações visando corrigir o curso do desenvolvimento no tempo adequado e de maneira eficiente (Krutchen, 2000).

No modelo iterativo, a primeira iteração reúne os requisitos básicos do produto que são validados pelo cliente. Um plano é desenvolvido para a próxima interação visando satisfazer as necessidades do cliente. Esse processo é repetido após cada iteração até que o produto completo seja produzido. Diferente dos outros modelos, o modelo iterativo objetiva a elaboração de um produto operacional a cada iteração (Pressman, 2005).

O modelo iterativo permite flexibilidade em ambientes de projeto onde existem indefinições de requisitos iniciais e a determinação do foco nos pontos mais críticos do projeto.

Outras vantagens deste modelo são a detecção o mais cedo possível de inconsistências entre os requisitos e a implementação, a produção de

resultados tangíveis a cada iteração e a identificação de melhorias contínuas no processo (Krutchen, 2000).

As fases do modelo iterativo são apresentadas na Figura 2.7.

Figura 2.7 – Modelo Iterativo (Pressman, 2005)

A definição do modelo adequado para cada projeto depende de vários fatores como as características do projeto, as necessidades do cliente e o tempo disponível para realização do trabalho.

A partir dos conceitos apresentados sobre os modelos, alguns parâmetros podem ser avaliados contra as forças e fraquezas de cada modelo, conforme demonstrado na Tabela 2.1 adaptada da proposta de McConnell (1996).

Tabela 2.1 – Parâmetros de Capacidade dos Ciclos de Vida adaptado (McConnell, 1996)

Capacidade do Modelo	Cascata	Prototipação	Espiral	Iterativo
Requisitos Instáveis	Fraco	Ótimo	Ótimo	Ótimo
Arquitetura Indefinida	Fraco	Razoável	Ótimo	Ótimo
Gerenciamento Riscos	Fraco	Razoável	Ótimo	Ótimo
Flexibilidade de Adequação às mudanças	Fraco	Ótimo	Razoável	Ótimo
Visibilidade de Progresso	Fraco	Ótimo	Ótimo	Ótimo
Requer Baixo Gerenciamento	Razoável	Razoável	Ótimo	Razoável
Nível de Planejamento	Razoável	Razoável	Ótimo	Ótimo
Velocidade de Desenvolvimento	Ótimo	Ótimo	Razoável	Razoável

2.4. Conclusões do Capítulo

Este capítulo apresentou as principais características dos quatro modelos de ciclo de vida mais utilizados no desenvolvimento de software atualmente, os conceitos relacionados ao planejamento, controle e monitoramento de um projeto de software.

A escolha do modelo de ciclo de vida para o desenvolvimento de software deve estar adequado às características de cada projeto, pois a elaboração do planejamento segue as premissas e restrições que cada modelo possui e orienta as atividades de controle que são aplicadas durante a execução do projeto.

Para a realização das atividades da fase de planejamento devem ser considerados os aspectos de tempo disponível e o nível de detalhe esperado, pois o excesso de esforço nestas atividades pode gerar atraso no projeto e a preparação de cronogramas que não refletem as condições reais de desenvolvimento.

Embora o planejamento seja essencial, o acompanhamento e a monitoração do projeto exercem um papel fundamental no sucesso do projeto, visto que é através dessas atividades que os desvios são identificados e corrigidos, permitindo a adaptação às novas condições do ambiente ou a situações não previstas no plano original do projeto.

3 CONCEITOS DAS TÉCNICAS E PRÁTICAS DE ENGENHARIA DE SOFTWARE

Objetivo do Capítulo

O objetivo deste capítulo é descrever os conceitos que sustentam as práticas e técnicas de Engenharia de Software que são aplicadas no roteiro, fornecendo subsídios teóricos sobre a utilização de cada uma delas.

Neste capítulo são apresentadas as principais características e conceitos da arquitetura e sua importância na redução de tempo no desenvolvimento de software. Também são apresentados os fundamentos da prática do desenvolvimento *timebox* que busca manter o foco nas principais características do produto e na sensibilização da restrição de tempo à equipe do projeto.

São descritos os conceitos da prototipação evolutiva como ferramenta para redução de riscos e obtenção de um melhor entendimento dos requisitos, bem como as características do particionamento para a redução da complexidade e como elemento facilitador do processo de controle do projeto.

Também são conceituadas as técnicas do desenvolvimento baseado em componentes e da reutilização, seus efeitos na redução da complexidade, nos ganhos de produtividade e no aumento do paralelismo das tarefas de implementação.

Finalmente, são apresentados os conceitos da engenharia simultânea e sua significativa contribuição na redução do tempo de desenvolvimento.

Este capítulo foi desenvolvido com base no estudo dos seguintes trabalhos:

McConnell(1996), Pinna (2004), Krutchen (2000), Luqi(1989), Boehm (1994), Garlan (2000), Ran (2001), You-Sheng; Yu-Yun (2003), Kent; Howse; Lauder (2000), Park (1999), Dwiwedi et al. (1990), BlackBurn; Scudder; Wassenhove (2000), Smith (1997), Amundsen; Hutchison (1990), Brookes; Backhouse (1997), Karandikar et al (1993), IEEE 1471-2000, Subramanian e Chung (2005), Sadou, Tamzalit e Oussalah (2005), Hofmeister et al. (2005), Vitharana (2003), Bass; Clements; Kazman (2003), Ravichandran e Rothenberger (2003).

3.1. A Arquitetura de Software

3.1.1 – Conceitos de Arquitetura de Software

A definição da arquitetura é um passo chave para os efetivos ganhos em tempo de desenvolvimento e de produtividade na construção de software. Sua abrangência não está restrita a estrutura de implementação do sistema, mas também às questões de usabilidade, funcionalidade, desempenho, confiabilidade, reuso e restrições econômicas e tecnológicas (Krutchen, 2000).

De acordo com o padrão IEEE 1471-2000, a arquitetura é definida como uma prática para a organização fundamental do sistema, a incorporação de seus componentes, seus relacionamentos com outros componentes e o ambiente, os princípios que orientam o *design* e a evolução do projeto com o objetivo de identificar seus elementos básicos para o entendimento e o controle do sistema.

Para Subramanian e Chung (2005), a arquitetura de software é desenvolvida para satisfazer os requisitos funcionais de um problema e é constituída de componentes que reúnem os requisitos funcionais, as conexões entre estes componentes, suas restrições de uso para atender as necessidades do negócio e estabelece padrões e estilos para a construção do software.

Para Garlan (2000), com o aumento do tamanho e da complexidade dos sistemas de software modernos a arquitetura exerce um papel importante no desenvolvimento de software, atuando como um elo de ligação entre os requisitos e o código, conforme ilustrado na Figura 3.1.

O papel da arquitetura neste elo é facilitar o entendimento do sistema através de diferentes níveis de abstração, permitir a definição da estratégia de desenvolvimento e viabilizar as especificações de componentes para aumentar o grau de reuso na construção do software (Garlan, 2000).

Segundo Ran (2001) arquitetura de software é um conjunto de conceitos e decisões de projeto sobre a estrutura do software a ser construído que deve ser realizado nas fases iniciais do desenvolvimento para permitir a efetiva satisfação dos requisitos funcionais e sistêmicos.

Figura 3.1 – Papel da Arquitetura de Software (Garlan, 2000)

Para Sadou, Tamzalit e Oussalah (2005) a arquitetura de software oferece um alto nível de abstração para a descrição dos sistemas, permitindo a equipe do projeto direcionar esforços na estrutura global do sistema.

Essas definições mostram a importância da arquitetura na construção dos sistemas modernos, onde passa a ser parte do ciclo de vida do software e torna-se a base fundamental para viabilizar a utilização do desenvolvimento baseado em componentes, o aumento da reutilização e a maximização das atividades concorrentes para se obter ganhos reais de tempo no desenvolvimento do software.

Porém, para se obter os benefícios da arquitetura na construção de sistemas, alguns fatores críticos de sucesso devem ser observados segundo Boehm (1994):

- 1) **Domínio técnico do negócio** – maturidade do processo de desenvolvimento de software e conhecimento do ambiente;
- 2) **Desenvolvimento de componentes reutilizáveis** – orientação do desenvolvimento baseado em componentes com a preocupação de reuso;
- 3) **Treinamento das equipes de projeto** – Experiência e conhecimento da arquitetura pela equipe do projeto;
- 4) **Redução de riscos** – uso do processo de desenvolvimento e provas de conceito;
- 5) **Benefícios na redução do ciclo de vida** – experiência da equipe, componentes, grau de reuso e atividades concorrentes.

Embora esses fatores críticos tenham sido citados por Boehm em 1994, os seus conceitos permanecem atuais e aplicáveis por se tratarem de fundamentos para a verificação dos ganhos obtidos com a aplicação da arquitetura de software, visando atingir os seguintes objetivos:

- Estabelecer uma base técnica para a fase de projeto, permitindo o entendimento da estrutura do software a ser desenvolvido;
- Criar condições básicas para a implementação de componentes e sua efetiva reutilização;
- Simplificar as tarefas de codificação;
- Maximizar as atividades concorrentes.

3.1.2 – Desenvolvimento do Projeto de Arquitetura de Software

Existem diversos métodos para a implementação da prática de arquitetura de software no desenvolvimento de sistemas de informação. Esses métodos descrevem técnicas, processos e melhores práticas para facilitar o entendimento e o desenvolvimento do projeto de arquitetura de um sistema.

Hofmeister et al. (2005), desenvolveu um estudo para estabelecer um modelo geral de projeto de arquitetura de software baseado na análise comparativa entre diversos métodos, onde foram analisadas as forças e fraquezas de cada um para definir o conjunto de atividades que devem ser realizadas durante a definição da arquitetura. Os seguintes métodos de projeto de arquitetura de software foram analisados nesse estudo: *Attribute-Driven Design* (ADD) – desenvolvido pela SEI (*Software Engineering Institute*), *Siemens' 4 Views* (S4V) – desenvolvido pela *Siemens Corporate Research*, Visão “4+1” RUP – desenvolvido pela *Rational Software*, *Business Architecture Process and Organization* (BAPO) – desenvolvido pela *Philips Research* e *Architectural Separation of Concerns* (ASC) – desenvolvido pela *Nokia*.

O detalhamento desses métodos não está no escopo deste trabalho, sendo o foco apresentar o modelo para o projeto de arquitetura definido pelo estudo de Hofmeister et al. (2005).

Como resultado dessa análise comparativa, foi desenvolvido um modelo com as atividades essenciais a serem realizadas durante a definição do projeto de arquitetura.

Baseado no contexto do projeto e nos requisitos não funcionais de interesse da arquitetura, a análise arquitetural identifica os problemas que a arquitetura deve resolver e organiza os requisitos mais significativos para a solução. A síntese arquitetural tem por objetivo propor soluções alternativas, denominadas arquiteturas candidatas, que são avaliadas e medidas contra os requisitos, resultando na arquitetura final da solução (Hofmeister et al., 2005).

Figura 3.2 – Atividades do Projeto de Arquitetura (Hofmeister et al., 2005).

A seguir são descritas as principais atividades do modelo apresentado na Figura 3.2.

a) Interesses Arquiteturais:

São os requisitos não funcionais do sistema tais como: desempenho, confiabilidade, segurança, distribuição e escalabilidade. O grau de variação desses requisitos direciona e influencia as decisões técnicas que devem ser observadas durante a análise arquitetural.

b) Contexto:

Determinam o conjunto de requisitos relativos ao ambiente de desenvolvimento operacional que podem afetar o sistema. Além desses,

devem ser observados os fatores organizacionais, tecnológicos e do produto a fim de identificar suas principais características.

c) Análise Arquitetural:

Define o problema que a arquitetura irá resolver, baseado na análise dos requisitos arquiteturalmente significantes e do contexto do sistema.

As preocupações e a análise detalhada sobre os requisitos de portabilidade, integração com outros sistemas, modularidade, flexibilidade, qualidade e segurança são significativas para a definição da arquitetura.

d) Requisitos Arquiteturalmente Significantes:

A partir do conjunto de requisitos identificados, são escolhidos aqueles que afetam diretamente a arquitetura. Devem ser observados aqueles que têm maior impacto nos objetivos de negócio e na decomposição do sistema.

e) Síntese Arquitetural:

Apresenta soluções alternativas que refletem as decisões sobre a estrutura do software e o atendimento aos requisitos estabelecidos. Compreende a identificação de possíveis soluções, tomada de decisões de projeto que direcionam a avaliação arquitetural.

f) Soluções Arquiteturais Candidatas:

São as soluções alternativas que devem ser avaliadas através de simulações dos diversos cenários ou com o desenvolvimento de protótipos arquiteturais para prover os fundamentos para a avaliação arquitetural.

g) Avaliação Arquitetural:

Avalia as soluções alternativas apresentadas contra os requisitos mais significativos da arquitetura com o objetivo de definir a que melhor atende às necessidades do projeto. Os cenários e os protótipos devem ser refinados através da construção de provas de conceitos que permitam a avaliação dos resultados das soluções.

h) Arquitetura Validada:

Consiste na escolha da arquitetura consistente com os requisitos arquiteturalmente significantes e que satisfaça as premissas e restrições estabelecidas para atender às necessidades do projeto.

Em função da complexidade das tarefas, essas atividades não são executadas sequencialmente. Elas devem ocorrer de forma progressiva e repetida vezes até que a arquitetura esteja completa e validada (Hofmeister et al., 2005).

Esse modelo define as atividades fundamentais para a execução do projeto de arquitetura durante o ciclo de vida de construção do software, sendo a sua utilização adequada de acordo com as características de cada projeto ou ao método escolhido para o desenvolvimento da arquitetura do software.

3.2. Desenvolvimento *Timebox*

O desenvolvimento *timebox* é uma prática que auxilia a manter o foco nas principais características do software, evidenciar o senso de restrição de tempo à equipe do projeto e reduzir o tempo de desenvolvimento do software (McConnell, 1996).

O procedimento central desta prática é enquadrar os requisitos centrais do projeto ao tempo disponível, enquanto os demais requisitos são incorporados em outros *timeboxes* com menor prioridade. A prioridade destes requisitos é definida conjuntamente pelo cliente e pela equipe do projeto, permitindo a redução do tempo de desenvolvimento.

A aplicação dessa prática requer a utilização conjunta com a prática de prototipação, além de necessitar do envolvimento significativo do usuário final e de revisões constantes da equipe do projeto. Após a construção, o sistema é avaliado pelo cliente, podendo ser aceito completamente ou retornar para ajustes de funcionalidades ou de qualidade, conforme ilustrado na Figura 3.3.

A aplicação do desenvolvimento *timebox* é facilitada pelo suporte provido pela arquitetura definida para a construção do software, onde o atendimento aos requisitos de modularidade e flexibilidade do sistema provê melhores condições de sua utilização.

Figura 3.3 – Desenvolvimento *Timebox* (McConnell, 1996)

Embora seja uma prática da fase de projeto e de construção, a definição dos *timeboxes* deve ser parte da fase de planejamento, onde o cliente participa diretamente da definição e limitação dos principais requisitos que são atendidos dentro de cada *timebox*, bem como da validação dos protótipos e das decisões gerenciais.

Segundo McConnell (1996), existem duas recomendações que devem ser consideradas na aplicação do desenvolvimento *timebox*:

- 1) Cada *timebox* deve ter duração máxima de 120 dias, pois durações maiores podem não ter o senso de urgência necessário à aplicação desta técnica;
- 2) Envolvimento constante do cliente durante o desenvolvimento do projeto.

As principais condições de sucesso desta prática é que a data final estabelecida para cada *timebox* não seja alterada, o cliente esteja de acordo com os requisitos definidos e a limitação rígida do escopo seja atendida (McConnell, 1996). Qualquer mudança nessas condições deve ser tratada como um novo *timebox* para não comprometer os prazos estabelecidos.

3.3. Prototipação do Sistema

É uma técnica que permite que os requisitos do sistema sejam mapeados em *interfaces* amigáveis para o cliente, facilitando o entendimento das necessidades e a adequação às mudanças (McConnell, 1996). Deve ser desenvolvido em partições conforme a definição dos *timeboxes* e do particionamento do sistema.

Esta prática melhora significativamente a visibilidade do progresso das atividades do projeto junto ao cliente, permite aumentar o nível de definição dos requisitos e contribui para a redução de tempo na construção do software.

Existem duas formas para realização da prototipação:

- a) Prototipação de *Interface*
- b) Prototipação Evolutiva

Na prototipação de *interface* o objetivo é validar os requisitos do sistema junto ao cliente através de telas que demonstrem funcionalmente o software a ser construído. As ferramentas utilizadas para a elaboração do protótipo podem ser desde esboços em papel até o uso de softwares específicos para a criação de *interfaces*. Ao final do trabalho o protótipo é escrito na linguagem apropriada. Tem como principais vantagens a rapidez com que é produzido e na redução de riscos.

A prototipação evolutiva segue os mesmos objetivos da prototipação de *interface*, porém diferencia-se pelo fato do código produzido ser aproveitado no código final, portanto deve-se utilizar a linguagem de desenvolvimento do software e ser submetido às avaliações de qualidade quanto a aderência aos padrões de codificação.

Recomenda-se o uso da prototipação evolutiva quando os requisitos são mais estáveis e nas partições do sistema. Para os sistemas de médio e grande porte, ambas as abordagens podem ser utilizadas (McConnell, 1996).

No ciclo apresentado na Figura 3.4, com a determinação da parte do sistema a ser desenvolvida, inicia-se a construção do protótipo que é discutido em interações constantes de validação com o usuário, onde são capturadas as modificações necessárias até se obter o entendimento comum entre os envolvidos. Com o aceite

do usuário a construção é iniciada e o ciclo se repete até a conclusão de todo o projeto.

Figura 3.4 – Ciclo de Prototipação Adaptado (Luqi, 1989)

Os potenciais benefícios da prototipação dependem da habilidade de se realizar modificações no protótipo com esforço bem menor que o requerido para modificar o software pronto (Luqi, 1989).

3.4. Particionamento do Software

O desenvolvimento de um projeto em uma única partição é complexo, de difícil controle gerencial e os resultados produzidos são tardios para o cliente, gerando aumento de expectativas e de conflitos entre os envolvidos no projeto.

O particionamento do projeto visa subdividir o produto a ser construído em unidades menores e mais gerenciáveis permitindo o controle e avaliação do progresso das atividades que geram os produtos. Uma vez definido o escopo do trabalho, deve-se subdividir o trabalho em unidades que orientam a estratégia de desenvolvimento e os produtos a serem entregues ao cliente.

Para definir os particionamentos, a equipe do projeto deve agrupar as funcionalidades similares do sistema de forma que cada partição seja um executável independente, onde o cliente tenha condições de validar o atendimento dos requisitos do projeto.

Não há um tamanho definido para cada partição, cabendo a equipe do projeto, em conjunto com o cliente, dividir o sistema de forma a atender às principais regras de negócio e a ordem que são entregues para avaliação. Cada partição pode ser dividida em sub-partições com o objetivo de simplificar a tarefa de desenvolvimento e proporcionar resultados visíveis mais rapidamente. A Figura 3.5 exemplifica o particionamento de um projeto.

Figura 3.5 – Representação de um particionamento

Embora não reduza diretamente o tempo de desenvolvimento do software, esta prática permite a redução substancial dos riscos através da simplificação das tarefas, melhoria do controle gerencial e a produção de resultados visíveis mais rápidos para o cliente, aumentando as chances de sucesso do projeto.

3.5. Desenvolvimento Baseado em Componentes e Reutilização

3.5.1 – Desenvolvimento Baseado em Componentes

Com o aumento da complexidade do ambiente e das características do software, as preocupações das equipes de projeto não são apenas com os

requisitos funcionais, mas envolve também abordagens em relação ao desempenho, concorrência, facilidade de manutenção e reuso (You-Sheng; Yu-Yun, 2003).

Suportado pela arquitetura definida para a construção do software, o desenvolvimento de software baseado em componentes tem por objetivo definir as especificações de *interfaces* genéricas de cada elemento para que os mesmos possam ser utilizados no projeto sem a necessidade de conhecimento especializado. Estes componentes são caixas pretas que encapsulam as regras de negócio e a complexidade da tecnologia envolvida.

Para Pressman (2005), um componente é definido com uma parte não-trivial de um sistema, praticamente independente e substituível, que exerce um papel claro no contexto de uma arquitetura definida.

Segundo Vitharana (2003), um componente é definido como um pedaço de software executável com uma interface publicada.

O modelo de processo de software baseado em componentes (CBSE – *Component-Based Software Engineering*) é definido por Bass; Clements e Kazman (2003) como um processo que dá ênfase a construção de sistemas usando componentes de software reusáveis, onde a implementação dá lugar à integração como foco no processo de desenvolvimento.

No CBSE a identificação dos componentes começa a ser definida ainda na fase de entendimento dos requisitos, com foco nas necessidades de negócio, e estende-se até a fase de projeto onde são identificados os componentes relacionados à arquitetura, linguagem de programação utilizada, interação com sistemas externos e na comunicação com a infra-estrutura de hardware e software do ambiente. Cada componente é um software independente que provê o acesso às suas funcionalidades através das interfaces definidas.

Todos os componentes identificados e criados são armazenados em um repositório central para compartilhamento entre a equipe do projeto. Esse repositório central é também chamado de biblioteca de componentes.

O modelo CBSE sugere a incorporação do desenvolvimento de componentes dentro do ciclo de vida do projeto, realizando as atividades descritas na Figura 3.6, durante a fase de construção do software.

Figura 3.6 – Desenvolvimento Baseado em Componentes (Pressman, 2005)

A partir da identificação dos componentes necessários ao projeto, a primeira atividade da equipe é verificar sua existência no repositório. Se estiver disponível, o componente é extraído para utilização. Caso contrário, o componente é desenvolvido ou adquirido no mercado para atender às necessidades do projeto. Após isso, são armazenados na biblioteca para utilização posterior em outros projetos.

O desenvolvimento baseado em componentes apresenta várias vantagens se comparados com o desenvolvimento tradicional (Kent; Howse; Lauder, 2000):

- 1) **Redução dos custos de manutenção** – Uma vez construídos e testados os componentes tornam-se estáveis e o esforço de manutenção é menor;
- 2) **Aumento da produtividade através da reutilização** – Através dos componentes o esforço de implementação subsequente é reduzido com o reaproveitamento das funcionalidades e a eliminação de codificação adicional;

- 3) **Isolamento da *interface* com sistemas externos** – Os componentes são caixas pretas para o programador, isolando a necessidade de conhecimento sobre todas as interfaces do sistema;
- 4) **Maior portabilidade do sistema** – De acordo com a tecnologia utilizada, a construção de componentes permite maior flexibilidade de utilização nas diversas plataformas.

Embora as vantagens do desenvolvimento baseado em componentes sejam muitas, existem aspectos que necessitam ser observados durante a utilização dessa prática na construção de software.

Vitharana (2003) aponta os seguintes riscos e desafios para a implementação do desenvolvimento baseado em componentes:

- 1) **Evolução dos requisitos** – As evoluções dos requisitos devem ser observadas e documentadas para evitar que os componentes fiquem obsoletos;
- 2) **Alto investimento inicial** – A adoção do CBSE envolve gastos iniciais para organização da estrutura, no desenvolvimento dos componentes e no estabelecimento do processo;
- 3) **Versionamento** – A publicação das atualizações de versões dos componentes exige coordenação para manter a sincronização dos componentes;
- 4) **Testes dos componentes** – O processo de testes é mais complexo por não se conhecer exatamente como os componentes são utilizados nos diversos sistemas.

O desenvolvimento baseado em componentes é a base para a prática da reutilização e permite ganhos consideráveis de produtividade e de redução de tempo no desenvolvimento de software. Segundo Pressman (2005) os componentes podem reduzir em até 70% (setenta por cento) o tempo de desenvolvimento de um software e aumentar em até 55% (cinquenta e cinco por cento) a produtividade da equipe de projeto.

3.5.2 – Reutilização de Componentes

Reutilizar o software existente aumenta a produtividade no desenvolvimento de software (Park, 1999) e é uma consequência do desenvolvimento baseado em componentes.

Para Ravichandran e Rothenberger (2003), o reuso de software em vários projetos é uma importante estratégia para melhorar a eficiência no desenvolvimento de software e aumentar sua qualidade.

Existem três estratégias para reutilização de componentes que podem ser aplicadas ao desenvolvimento de software, cuja análise comparativa é apresentada na Tabela 3.1:

1. **Reuso Caixa-Branca:** Nessa abordagem, os desenvolvedores podem modificar o código dos componentes existentes para adequá-los às suas necessidades e aos novos requisitos.
2. **Reuso Caixa-Preta, desenvolvido pela própria empresa:** Nessa estratégia o reuso está na recuperação de componentes desenvolvidos internamente pelas equipes de projeto, não permitindo a modificação do código pelos desenvolvedores. Os componentes devem ser utilizados conforme suas especificações de interface.
3. **Reuso Caixa-Preta, adquirido no mercado:** Essa abordagem permite aos desenvolvedores procurar os componentes no mercado, aumentando as chances de identificar àquele que melhor se encaixa as necessidades do projeto.

A escolha de qual estratégia deve ser adotada pela organização deve ser avaliada contra os custos de aquisição do componente no mercado, os custos de customização dos componentes existentes e da taxa de reuso alcançada com cada estratégia (Ravichandran;Rothenberger, 2003).

Uma possível seqüência para a escolha da estratégia de reuso, apresentada por Ravichandran e Rothenberger (2003), pode ser descrita da seguinte maneira:

- a) Adotar o reuso caixa-preta interno, se os componentes existentes no repositório atendem às necessidades;
- b) Se não existe componentes caixa-preta para atender ao projeto e o custo de customização é reduzido, adotar a estratégia de reuso caixa-branca;
- c) Se o custo de customização é elevado e não existe componente caixa-preta no repositório, deve-se buscar um componente no mercado ou desenvolver um novo componente. Nesse caso, a decisão deve ser avaliada em relação ao custo e tempo para desenvolvimento ou obtenção do componente.

Tabela 3.1 – Comparação das Estratégias de Reuso (Ravichandran e Rothenberger, 2003)

Item	Caixa-Branca	Caixa-Preta	
		Interno	Mercado
Taxa de reuso	Alta taxa de reuso pela capacidade de modificação dos componentes existentes.	Provavelmente, menor taxa de reuso pela necessidade de encontrar um componente exato.	Reuso moderado, pela maior possibilidade de encontrar um componente adequado.
Flexibilidade	Alta Flexibilidade.	Flexibilidade limitada.	Flexibilidade limitada.
Escopo do reuso	Reuso local e restrito ao grupo da mesma localização.	Reuso local, mas pode ser ampliado para uso em diversos lugares.	Componentes adquiridos do mercado. Uso diversificado.
Custo de Construção	Baixo custo de construção	Maior custo de construção para parametrização das interfaces.	Maior custo de construção para parametrização das interfaces.
Custo de Uso	Custo moderada de procura e alto custo de customização.	Baixo custo de busca e de customização.	Alto custo de busca e baixo custo de customização.
Qualidade	Conhecida pela organização.	Conhecida pela organização.	Pode não ser conhecida. Maior risco de uso.
Tempo de Retorno	Retorno somente após existir massa crítica no repositório. Taxa de reuso moderada para cobrir o investimento.	Retorno somente após existir massa crítica no repositório. Alta taxa de reuso para cobrir o investimento.	Não requer investimento para construir o repositório.
Repositório	Crescimento exponencial do tamanho do repositório. Controle de versões é essencial.	Crescimento linear do tamanho do repositório. Controle de versões é importante e mais simples.	Limitada necessidade de um repositório local. Responsabilidade de versões é do vendedor.

Independente da estratégia de reuso utilizada, a definição clara do processo para criação das bibliotecas, de publicação e recuperação dos componentes existentes e o treinamento da equipe no processo são fundamentais para a

eficiência desta prática, pois a maior barreira para a reutilização são as dificuldades na localização e uso desses componentes e a cultura dos desenvolvedores.

Com o aumento do grau de reuso do software, obtém-se ganhos reais no tempo de desenvolvimento através da redução do esforço de codificação e da redução da necessidade de testes sobre o software reutilizado.

3.6. Engenharia Simultânea

O termo engenharia simultânea vem sendo aplicado desde o final dos anos 80 no processo de desenvolvimento de software, onde o paralelismo das atividades e a integração funcional são partes do processo de desenvolvimento. As idéias atrás do conceito da engenharia simultânea são simples, mas produzem resultados significativos no tempo de produção do software (Smith, 1997) e estabelece mecanismos de melhoria de desempenho para as empresas (Brookes; BackHouse, 2000).

Segundo Amundsen e Hutchison (1990), engenharia simultânea é uma abordagem sistemática para criar um produto que considera todos os elementos do ciclo de vida de desenvolvimento. Engenharia simultânea não é a eliminação arbitrária de alguma fase do ciclo de vida, mas a realização de atividades em paralelo para a otimização do tempo de produção.

Para Blackburn, Scudder e Wassenhove (2000) a engenharia simultânea é o gerenciamento de técnicas para reduzir o *time-to-market* e aumentar a produtividade no desenvolvimento de um produto, através da execução de tarefas em paralelo por diferentes grupos e do fluxo controlado de informações para suportar as equipes de desenvolvimento.

O suporte e a flexibilidade de desenvolvimento fornecido pela arquitetura e pelo desenvolvimento baseado em componentes são essenciais para se alcançar um alto grau de execução de tarefas concorrentes, atingido através da estruturação do sistema de forma mais independente possível entre as diversas partições, da divisão do software em camadas e do aumento do nível de reutilização de componentes.

A aplicação da engenharia simultânea no desenvolvimento de software consiste em quatro elementos (Amundsen; Hutchison, 1990):

- 1) **Tarefas em paralelo:** o produto é dividido em subsistemas que tenham o mínimo de interação entre si;
- 2) **Compartilhamento de informações:** a comunicação entre os grupos recebe atenção especial para permitir a troca de informações e alinhamento das expectativas;
- 3) **Suporte e confiabilidade:** prover a equipe de desenvolvimento condições de avaliar a qualidade do produto;
- 4) **Equipe de projeto:** Gerenciamento colaborativo com os envolvidos no projeto durante todas as fases do ciclo de vida e dos subsistemas.

Com a utilização da engenharia simultânea na produção de software pretende-se atingir os seguintes objetivos representados esquematicamente na Figura 3.7 (Dwivedi *et al*, 1990).

- a) **Redução dos custos** – Distribuir as atividades de forma que possam ser realizadas no menor tempo possível;
- b) **Eliminação de gastos** – Alocar os recursos com o perfil adequado necessário à realização da tarefa;
- c) **Redução do tempo de desenvolvimento** – Com a execução das atividades em paralelo é possível reduzir o tempo de desenvolvimento do software;
- d) **Aumento da qualidade** – Realizar as atividades de análise e projeto antes da construção do produto para garantir a aderência aos requisitos do produto;
- e) **Melhoria contínua do produto** – Verificação progressiva da qualidade dos produtos gerados.

Esses objetivos são amplos e ambiciosos. Porém, através da realização desses objetivos da engenharia simultânea é possível atingir resultados tangíveis mais breves durante o ciclo de vida do projeto (Dwivedi et al., 1990).

Figura 3.7 – Objetivos da Engenharia Simultânea (Dwivedi et al., 1990)

A execução da engenharia simultânea é uma tarefa complexa que exige um nível de gerenciamento diferenciado, onde alguns elementos chaves precisam ser observados, conforme ilustrado na Tabela 3.2 (Karandikar *et al*, 1993).

Tabela 3.2 – Elementos chaves da Engenharia Simultânea Adaptado (Karandikar *et al*, 1993)

Elementos Chaves	Requisitos
Foco no Cliente	<ul style="list-style-type: none"> - Interação com o cliente - Constante atenção na satisfação do cliente
Foco no Processo	<ul style="list-style-type: none"> - Definição da metodologia - Identificação e controle do processo - Definição do gerenciamento
Estratégia do projeto	<ul style="list-style-type: none"> - Formação da equipe - Treinamento da equipe - Definição de medidas de desempenho
Sistema de Gerenciamento	<ul style="list-style-type: none"> - Gerenciamento de riscos
Mecanismos de produção rápida	<ul style="list-style-type: none"> - Prototipação - Utilização de ferramentas - Adoção de técnicas e práticas apropriadas
Agilidade	<ul style="list-style-type: none"> - Habilidade de responder as alterações
Disciplina	<ul style="list-style-type: none"> - Atenção nos objetivos das tarefas - Métodos comuns, ferramentas e processo.

A aplicação da engenharia simultânea permite o planejamento e a execução das atividades do projeto com paralelismo, mas necessita de elevado grau de gerenciamento das atividades para a minimização do retrabalho e de melhoria da comunicação, com o objetivo de desenvolver a cultura de trabalho em grupo nas pessoas envolvidas no projeto.

3.7. Conclusões do Capítulo

Neste capítulo foram mostradas as práticas e técnicas de engenharia de software utilizadas no roteiro para redução de tempo no desenvolvimento de software.

Uma vez iniciado o projeto, a preocupação da equipe deve estar em estabelecer o particionamento adequado do software às necessidades do cliente com o objetivo de definir claramente a prioridade das partes que devem ser construídas.

O desenvolvimento *timebox* associado com a protipação evolutiva permite aos envolvidos no projeto definirem e validarem os requisitos que devem ser atendidos prioritariamente na partição, delimitando o escopo do projeto para cada *timebox* estabelecido e obtendo o compromisso de todos.

A arquitetura de software é o fundamento básico que permite a organização e aplicação da prática do desenvolvimento baseado em componentes e da reutilização. Ela deve ser definida e validada o mais breve possível dentro do ciclo de vida do projeto.

Para se estabelecer o processo de desenvolvimento baseado em componentes (CBSE) é necessário que a equipe do projeto esteja consciente dos ganhos de produtividade que podem ser obtidos e dos benefícios que o reuso proporciona ao processo de construção do software. Para isso, o processo CBSE deve ser parte do ciclo de vida do projeto e as regras de utilização do repositório de componentes publicada a todos os envolvidos no projeto.

A aplicação da técnica de engenharia simultânea resulta em uma diminuição substancial do tempo de desenvolvimento através da execução das tarefas em paralelo, proporcionada pela estrutura e organização estabelecida no planejamento do projeto, na definição clara do escopo e na arquitetura de software bem definida. Porém, necessita de elevado grau de controle gerencial e de comunicação entre todos os envolvidos no projeto.

4 ABORDAGENS PARA A REDUÇÃO DE TEMPO NO DESENVOLVIMENTO DE SOFTWARE

Objetivo do Capítulo

O objetivo deste capítulo é apresentar as principais iniciativas existentes sobre a redução de tempo no desenvolvimento de software, seus conceitos, características e focos principais.

Os fundamentos e resultados obtidos do estudo dessas abordagens são utilizados como referências para a elaboração do roteiro de redução de tempo no desenvolvimento de software proposto por este trabalho.

Este capítulo foi desenvolvido com base no estudo dos seguintes trabalhos:

BlackBurn; Scudder; Wassenhove (2000), Collier; Collofello (1995), Clincy (2003) e Wetherbe; Frolick (2000); Howard (2002).

4.1. Principais Iniciativas para a Redução do Ciclo de Desenvolvimento de Software

A indústria de desenvolvimento de software, após a atenção dispensada aos fatores de produtividade e de qualidade na produção de software durante os anos 80, iniciou os anos 90 com o objetivo de reduzir o tempo de desenvolvimento de software (Collier; Collofello, 1995).

Aumentar a produtividade e reduzir o tempo do ciclo de desenvolvimento de software são os principais objetivos das organizações responsáveis pela construção de software na atualidade (Clincy, 2003).

A partir deste objetivo, o foco das pesquisas sobre a redução do tempo do ciclo de desenvolvimento nas empresas de tecnologia está na redução dos custos, na melhoria dos serviços ao cliente e na obtenção de vantagens competitivas no mercado de software (Wetherbe; Frolick, 2000).

A primeira iniciativa para redução de tempo na construção de software foi o Desenvolvimento Rápido de Aplicação (RAD - *Rapid Application Development*) desenvolvido por James Martin em 1991, cujos objetivos principais são entregar os produtos com maior rapidez e aumentar a qualidade enquanto diminui os custos. Nesse modelo, o desenvolvimento é fortemente baseado no uso de ferramentas CASE (*Computer-Aided Software Engineering*) e constituído de cinco pontos focais:

1. **Equipes reduzidas** – Melhora a comunicação dentro do projeto;
2. **Prototipação evolutiva** – Auxilia na identificação de requisitos na fase de entendimento do sistema;
3. **Uso de ferramentas CASE** – O apoio de ferramentas CASE aumenta a produtividade através da ênfase na automação;
4. **Joint Application Design (JAD)** – Reuniões com todos os envolvidos no projeto com o objetivo de esclarecer e validar os requisitos;
5. **Rígidos limites de tempo no desenvolvimento (timebox)** – Controle do escopo e limitação dos requisitos atendidos.

Segundo Howard (2002), os benefícios obtidos com a aplicação do RAD são reconhecidos pela maioria das organizações. Porém, ainda existem discussões sobre sua aplicação em projetos de maior porte, efetividade na manutenção da qualidade e o uso excessivo de ferramentas de automação para aumentar a produtividade.

No entanto, esse trabalho é significativo para a área por servir de base para outros esforços sobre a redução de tempo de desenvolvimento (Collier; Collofello, 1995).

O Desenvolvimento Rápido Evolutivo (*Rapid Evolutionary Development*) desenvolvido por Arthur (1992) é um modelo de ciclo de vida com o objetivo de reduzir o tempo de desenvolvimento de software.

É baseado no desenvolvimento evolutivo tradicional aliado ao forte uso da prototipação, o que significa maior interação com o cliente e acomodação das alterações de requisitos durante as fases iniciais do projeto (Collier; Collofello, 1995).

Segundo Arthur (1992), as principais características desse modelo são:

- 1. Prototipação** – Facilita a captura e o entendimento dos requisitos;
- 2. Uso da Lei de Pareto** – Chamada regra 80/20. Sugere que 20% dos produtos possuem 80% do valor esperado. O foco desse modelo é desenvolver os requisitos que se enquadram nos 20% dessa regra;
- 3. Interação com o cliente** – Permite a validação constante dos requisitos e produtos durante o projeto.

A partir dessas características, o *Rapid Evolutionary Development* visa reduzir os riscos, o retrabalho e a complexidade dos produtos e dos processos durante o desenvolvimento do software.

Segundo Collier e Collofello (1995), a questão de redução de tempo está relacionada às características ambientais e às características do produto a ser construído.

As características ambientais incluem o modelo de processo, a disponibilidade de recursos, métodos, procedimentos e práticas utilizadas durante o desenvolvimento do software.

As características do produto estão relacionadas ao grau de complexidade, a independência entre os subsistemas e o tamanho do software a ser produzido.

Para cada projeto deve ser realizada a análise dessas características identificando-se aquelas que podem contribuir para a redução de tempo das tarefas, categorizando-as de acordo com o impacto e aplicando-as de forma a obter os resultados esperados.

Uma vez realizada essa análise inicial, a avaliação dessas características deve ser constante durante o ciclo de desenvolvimento do software, a fim de permitir correções e melhorias na realização das tarefas.

A combinação dessas características ambientais e do produto proporciona a base para estabelecer um conjunto de objetivos para reduzir o ciclo de desenvolvimento de software, conforme ilustrado na Tabela 4.1.

Uma importante preocupação dessa abordagem é não reduzir os requisitos de qualidade em função da redução do tempo de desenvolvimento. A qualidade pode ser mantida pela redução do acoplamento entre os módulos, o aumento da coesão e redução da complexidade das tarefas, proporcionando melhorias na capacidade de manutenção e na confiabilidade do software.

Tabela 4.1 – Objetivos para Redução do Ciclo de Desenvolvimento (Collier; Collofello, 1995).

Características Ambientais	Características do produto	Objetivos
Reuso de Software	Desenvolver novos componentes para o repositório	Maximização do reuso
Alterações de Requisitos	Determinar as interdependências das partições e sua complexidade	Partições independentes Redução da complexidade
Gerenciamento de Riscos	Identificar os riscos e determinar o grau de impacto	Minimizar a ocorrência dos riscos
Produtividade Pessoal	Nenhum impacto	
Disponibilidade de Recursos	Nenhum impacto	
Validação e Verificação	Determinar os efeitos do retrabalho	Garantia de correção e completeza
Qualidade	Determinar os efeitos do retrabalho	Garantia da qualidade
Estratégia de Integração	Determinar tarefas concorrentes Reduzir o caminho crítico	Baixo acoplamento Simplificação das tarefas

Outro estudo relacionado à redução de tempo foi realizado por Blackburn, Scudder e Wassenhove em um levantamento realizado entre 1996 e 1999, com 145 empresas de desenvolvimento de software nos Estados Unidos, Europa e Japão, onde foram identificados onze fatores que auxiliam na redução do tempo de desenvolvimento de software, apresentados na Figura 4.1.

Nesse estudo, as empresas foram divididas em dois grupos: um grupo com as empresas consideradas rápidas (diminuíram em mais de 25% o tempo de desenvolvimento nos últimos cinco anos) e empresas com tempo de desenvolvimento normal.

Esses fatores receberam classificação de um a cinco e estão ordenados de acordo com o grau de importância atribuído pelas empresas pesquisadas:

- 1) Equipes treinadas e capacitadas;
- 2) Melhoria da comunicação entre os membros do projeto;
- 3) Melhoria das especificações iniciais do cliente;
- 4) Melhoria do processo de gestão de projetos;
- 5) Estratégias de testes adequadas;
- 6) Engenharia Simultânea;
- 7) Reuso de código ou módulos;
- 8) Redução do retrabalho;
- 9) Uso de prototipação;
- 10)Uso adequado da tecnologia e ferramentas CASE;
- 11)Particionamento.

Na comparação entre os fatores adotados pelos dois tipos de empresas, os que mais contribuíram para a melhoria do tempo de desenvolvimento são os relacionados a: equipes treinadas e capacitadas, prototipação e redução do retrabalho, sendo responsáveis por 35% dessa melhoria.

Figura 4.1 – Fatores de Redução de Tempo (Blackburn; Scudder; Wassenhove, 2000)

Outras observações sobre o efeito desses fatores sobre a redução do tempo de desenvolvimento foram:

- A aplicação da engenharia simultânea no ciclo de desenvolvimento melhora a produtividade das equipes de desenvolvimento e reduz substancialmente o tempo de desenvolvimento;
- Equipes reduzidas tendem a ser mais produtivas e reduzem os problemas de comunicação;
- A utilização de ferramentas CASE recebeu baixa importância em termos de redução de tempo de desenvolvimento;
- A produtividade e de redução de tempo não são perfeitamente correlatos, pois é possível reduzir o tempo de desenvolvimento sem aumentar a produtividade.

Outra avaliação realizada por este estudo foi verificar o relacionamento entre a velocidade do desenvolvimento e a alocação do tempo em cada fase do projeto.

O resultado é ilustrado no Figura 4.2, onde é possível observar que as empresas consideradas mais rápidas consumiram mais tempo nas fases iniciais que nas fases subseqüentes, exceto na fase de codificação.

Figura 4.2 – Alocação de Tempo por Fase (Blackburn; Scudder; Wassenhove, 2000)

Esta conclusão mostra que para reduzir o tempo no ciclo de desenvolvimento, é necessário ser mais lento nas fases iniciais para se atingir o objetivo. Para aumentar a velocidade e produtividade deve-se investir mais tempo em entender as

necessidades, produzir especificações sem ambigüidades e elaborar um planejamento adequado ao projeto (Blackburn; Scudder; Wassenhove, 2000).

Segundo Clincy (2003), para reduzir o tempo de desenvolvimento também é necessário reduzir os canais de comunicação com a alocação de equipes pequenas (até 15 pessoas), diminuir o nível de comunicação formal dentro do projeto e produzir a documentação mínima essencial ao desenvolvimento do projeto.

A partir de uma análise comparativa entre essas abordagens, pode-se observar que vários aspectos e práticas de engenharia de software utilizadas são comuns e apresentam resultados efetivos na redução do tempo de desenvolvimento de software. Para os objetivos deste trabalho, os seguintes pontos comuns devem ser destacados:

- 1. Equipes Reduzidas:** Diminui os canais de comunicação e facilita o controle da execução das tarefas;
- 2. Redução dos Canais de Comunicação:** Reduz os problemas de comunicação através da diminuição do nível de formalidade e de documentação do projeto;
- 3. Interação Constante com o Cliente:** Diminui os conflitos com a equipe do projeto e melhora a validação dos requisitos do software;
- 4. Prototipação:** Facilita a captura e entendimento dos requisitos, reduzindo o retrabalho e simplificando as tarefas.

Além desses pontos comuns, as práticas de engenharia de software relacionadas ao desenvolvimento baseado em componentes (CBSE), ao reuso, à engenharia simultânea e ao desenvolvimento *timebox* também são citadas como relevantes na redução do tempo de desenvolvimento, auxiliando no aumento da produtividade, na redução da complexidade, na simplificação das tarefas do projeto e na maximização das tarefas concorrentes.

4.2. Conclusões do Capítulo

Neste capítulo foram descritas as principais abordagens sobre a redução do tempo no desenvolvimento de software, suas características e práticas utilizadas.

Desde a definição do modelo RAD (Desenvolvimento Rápido de Aplicação), diversos trabalhos buscam abordar a questão da redução de tempo na produção de software seguindo as diretrizes básicas de aumento da produtividade, redução do retrabalho, diminuição da complexidade e simplificação das tarefas. Além dessas diretrizes, as abordagens descritas neste capítulo mostram que a aplicação da engenharia simultânea é uma prática que contribui diretamente na redução do ciclo de desenvolvimento através da maximização de tarefas concorrentes dentro do processo de construção do software.

Embora essas práticas sejam eficientes na diminuição do ciclo de produção de software, a sua utilização conjunta dentro de um processo bem definido, maximiza os resultados finais de redução do tempo de desenvolvimento de software.

A partir dos resultados da análise dessas abordagens, obtém-se os subsídios técnicos para a elaboração do roteiro deste trabalho, permitindo identificar e selecionar as práticas de engenharia de software que contribuem para a redução do tempo e a manutenção dos níveis de qualidade nos projetos.

5 AVALIAÇÕES SOBRE A UTILIZAÇÃO DAS PRÁTICAS E TÉCNICAS DE ENGENHARIA DE SOFTWARE NO BRASIL

Objetivo do Capítulo

O objetivo deste capítulo é apresentar um panorama nacional sobre a utilização das técnicas e práticas de engenharia de software, através dos resultados da pesquisa realizada pelo Ministério da Ciência e Tecnologia (MCT) sobre qualidade e produtividade no setor de software brasileiro em 2001. São apresentados também, os resultados de uma enquete realizada pelo autor com o objetivo de avaliar o grau de utilização das técnicas e práticas que contribuem para a redução de tempo no desenvolvimento de software.

Os resultados dessas avaliações, juntamente com as abordagens sobre redução de tempo apresentadas no Capítulo 4, orientaram a definição das técnicas e práticas de engenharia de software que compõem o roteiro deste trabalho.

Este capítulo foi desenvolvido com base no estudo dos seguintes trabalhos:

MCT (2002), Westfall (1987) e contribuição do autor.

5.1. Pesquisa sobre Qualidade e Produtividade no Setor de Software Brasileiro

Em 2001, foi realizada pela Secretaria de Política de Informática a 5ª edição da pesquisa "Qualidade e Produtividade no Setor de Software Brasileiro" e a 2ª edição da "Produtividade Sistêmica no Setor de Software Brasileiro", em parceria com o Instituto Brasileiro de Qualidade e Produtividade do Paraná - IBQP-PR, sobre a utilização das práticas de engenharia no desenvolvimento de software no Brasil (MCT, 2002).

A pesquisa é amostral, aplicada sobre uma população alvo constituída pelas empresas de desenvolvimento de software dos tipos: pacote para comercialização – *packaged software*, software sob encomenda para terceiros – *custom software*, software para Internet – *Internet software*, software embarcado – *bundled, embedded software*.

A amostra efetiva da pesquisa foi de 446 empresas, cujos dados preenchidos em formulário próprio estruturado e não disfarçado, foram criticados e consistidos utilizando estrutura própria criada pela Secretaria de Política de Informática.

O erro calculado da amostra é de 4,5%, considerando uma população estimada para o ano 2000 de 10.713 empresas com atividades em software no Brasil, para um nível de confiança de 95% sobre os resultados da pesquisa (MCT, 2002).

5.1.1 Resultados da Pesquisa

A Tabela 5.1 apresenta os resultados globais da utilização das práticas de engenharia de software nas empresas de tecnologia no Brasil em 2001:

Tabela 5.1 – Práticas Engenharia de Software – SEPIN – (MCT, 2002)

Práticas de Engenharia de Software	Empresas	%
Modelagem de dados	302	70,10
Controle de versão de produto	298	69,10
Especificação de projetos	279	64,70
Especificação de requisitos	262	60,80
Especificação de programas	261	60,60
Projeto da interface com o usuário	244	56,60
Estimativa de custos	237	55,00
Métodos orientados a objetos	232	53,80
Prototipação	220	51,00
Estimativa de esforço	197	45,70
Métodos estruturados	173	40,10
Normas e padrões da organização	173	40,10
Análise crítica conjunta	167	38,70
Planejamento formal de testes	163	37,80
Estimativa de tamanho	125	29,00
Gerência de requisitos	105	24,40
Gerência de configuração	101	23,40
Engenharia da informação	92	21,30
Gerência de risco	51	11,80
Gestão de mudança	45	10,40
<i>Joint Application Design – JAD</i>	35	8,1

Os resultados apresentados na Tabela 5.1 mostram que a maioria das empresas pesquisadas adotam as práticas essenciais ao desenvolvimento de software como a modelagem de dados, a especificação de requisitos, as especificações de projetos e programas, entre outras. No entanto, algumas práticas que auxiliam na redução do tempo de desenvolvimento têm sua utilização limitada, como a prototipação utilizada por 51% das empresas, as normas e padrões da organização por 40,1%, o planejamento formal de testes por 37,8%, a gestão de riscos por 11,8% e a *Joint Application Design* (JAD) por 8,1%.

Figura 5.1 – Práticas de Desenvolvimento em Engenharia de Software (MCT,2002)

A Figura 5.1 apresenta graficamente os resultados da Tabela 5.1, segmentado nas práticas de desenvolvimento em engenharia de software, onde se observa um aumento na utilização dos métodos orientados a objetos (53,8%) em relação aos métodos estruturados (40,1%).

Esse aumento verificado sobre o uso da orientação a objetos é importante, pois técnicas como a componentização e a reutilização, que contribuem para a redução do tempo na produção do software, são mais efetivos com a utilização dessa prática.

Na Figura 5.2, observa-se que poucas empresas, menos de 25%, direcionam seus esforços para a questão do gerenciamento de requisitos, riscos e para a gestão de mudanças. A ausência dessas práticas afeta diretamente o controle de prazos e de

custos dos projetos e, conseqüentemente, são fatores críticos para a redução de tempo do ciclo de vida do projeto.

Figura 5.2 – Práticas de Gestão de Projetos em Engenharia de Software (MCT, 2002)

Verifica-se também um baixo índice de utilização de planejamento formal de testes (37,8%), cujo objetivo principal é a verificação da qualidade do software.

O resultado da pesquisa do MCT (2002) mostra que, embora exista ampla divulgação de normas, modelos e recomendações sobre as melhores práticas de engenharia de software, um número reduzido de empresas aplicam essas práticas dentro do seu processo de desenvolvimento.

5.2. Enquete para Verificação da Utilização das Práticas de Engenharia de Software

Entre os meses de novembro de 2005 e janeiro de 2006 foi realizada pelo autor uma enquete com as empresas de desenvolvimento de software nos estados de São Paulo, Rio de Janeiro, Minas Gerais e Distrito Federal para avaliar o nível de utilização das práticas e técnicas do roteiro proposto por este trabalho.

5.2.1. Objetivos da Enquete

O objetivo desta enquete foi avaliar o nível de conhecimento e grau de aplicação das técnicas e práticas de engenharia de software, visando consolidar seus conceitos e verificar sua efetividade na redução do tempo no desenvolvimento de projetos de software.

5.2.2. Natureza e Escopo

Foram distribuídos cem questionários (Anexo 1) descrevendo vinte e sete perguntas sobre o uso das técnicas e práticas de engenharia de software para as empresas de desenvolvimento de software, cada uma delas contendo de três a quatro opções de múltipla escolha.

As empresas pesquisadas são constituídas por 97,7% do setor privado de produção de software e 2,3% de empresas do governo.

5.2.3. Técnica de Amostragem

A enquete realizada é interseccional e não probabilística, aplicada sobre um universo constituído pelas empresas de desenvolvimento de software no Brasil atuantes nos segmentos de consultoria e prestação de serviços, indústria, telecomunicações, financeiro, educação e governo, escolhidas pelo método intencional ou por julgamento.

A amostra efetiva desta enquete foi de 44 empresas, cujos dados foram preenchidos em formulário próprio e criticados utilizando estrutura própria do autor.

Constituíram o universo desta amostragem 31 consultorias e prestadoras de serviços, 4 empresas do ramo da indústria, 2 do segmento de telecomunicações, 5 do segmento financeiro, 1 do segmento de educação e 1 estatal.

A maior concentração de empresas ocorreu no estado de São Paulo (62%), seguido pelo Rio de Janeiro (28%), Minas Gerais (7%) e Distrito Federal (3%).

5.2.4. Método Utilizado para Coleta dos Dados

A enquete foi realizada baseada em um questionário simples, constituído de vinte e sete perguntas, distribuídos e recebidos através de meio eletrônico aos responsáveis diretos pelo processo de desenvolvimento de software das empresas pesquisadas.

A estrutura do questionário pode ser analisada no Anexo 1.

5.2.5. Análise e Interpretação

A partir das respostas das empresas, as informações foram mapeadas e agrupadas de acordo com os grupos objeto da pesquisa.

Para efeitos de interpretação, as respostas marcadas com a opção “às vezes” foram mapeadas como negativas, pois o objetivo é avaliar a utilização completa da técnica ou prática em questão.

5.2.6. Limitações

Os questionários dessa enquete foram distribuídos para diversas empresas de desenvolvimento de software do círculo de relacionamento do autor e da Escola Politécnica da Universidade de São Paulo e não contou com a colaboração de nenhum órgão relacionado às empresas alvo.

5.2.7. Resultados da Enquete

As empresas que responderam aos questionários constituem um universo de quarenta e quatro de um total de cem questionários enviados.

Deste universo, 47,73% não possuem nenhuma certificação de processo de qualidade, enquanto 52,27% possuem certificação ISO9000 ou CMM (*Capability Maturity Model* – SEI/CMU), conforme ilustrado na Tabela 5.2.

Ainda em relação às empresas que participaram da enquete, 57% podem ser consideradas de grande porte (mais de 1000 funcionários), 14% de médio porte

(entre 100 e 1000 funcionários) e 29% de pequeno porte (menos de 100 funcionários).

Em relação às questões sobre gerenciamento de projetos foi verificado que 61,36% das empresas possuem profissionais com certificação PMP (*Project Management Professional*) do *Project Management Institute* (PMI).

Tabela 5.2 – Perfil de Qualidade das empresas participantes da enquete

Requisito	Nr. Empresas	%
Sem Certificação de Qualidade	21	47,73
Com Certificação de Qualidade	23	52,27
Com CMM Nível 2	13	29,55
Com CMM Nível 3	3	6,82
Com CMM Nível 4	1	2,27
Com ISO 9001	6	13,63

Foram observados também fatores que indicam uma melhoria na definição do processo de gerenciamento de projetos e no uso das técnicas de gestão de risco e gestão de mudanças, se comparadas com a pesquisa realizada pelo MCT (2002), como apresentado na Tabela 5.3.

Tabela 5.3 – Práticas de Gerenciamento de Projetos

Prática	Nr. Empresas	Enquete (%)	MCT (%)
Processo de Gestão de Projetos	28	63,64	40,1
Métrica para Estimativa de Tamanho	29	65,91	29,0
Gestão de Riscos	19	43,18	11,8
Gestão de Mudanças	23	52,27	10,4

A análise sobre a utilização dos ciclos de vida utilizados pelas empresas apresenta ainda uma grande participação do modelo cascata na maioria das organizações. Porém, pode ser verificado um bom crescimento na utilização do modelo iterativo dentro das organizações, conforme ilustrado na Tabela 5.4.

Outra observação importante sobre os ciclos de vida é que a maioria das empresas com certificação de qualidade utiliza mais de um modelo em seu processo de desenvolvimento.

Ainda no universo pesquisado, 9% das organizações não utilizam nenhum dos modelos de ciclo de vida apresentados pela enquete.

Tabela 5.4 – Modelos de Ciclos de Vida Utilizados

Modelo de Ciclo de Vida	Nr. Empresas	%
Iterativo	16	36,36
Cascata	13	29,55
Espiral	11	25,00
Outros	4	9,09

Em relação ao uso das práticas e técnicas de engenharia de software, a enquete mostra que a utilização da orientação a objetos, da prototipação e a definição da arquitetura são atividades amplamente utilizadas pelas organizações e atingiram altos índices de aplicação.

Porém, as demais práticas ainda possuem baixos indicadores de emprego no processo de desenvolvimento de software, conforme apresentado na Figura 5.3.

Figura 5.3 – Utilização das Práticas de Engenharia de Software do Roteiro Proposto

Vale ressaltar que, embora as empresas realizem as reuniões de *kickoff* no início dos projetos, 84,09% das empresas não utilizam esse tipo de reunião nas situações de mudanças de escopo ou troca de fase no processo de desenvolvimento do projeto. A realização da reunião de *kickoff* nessas situações visa diminuir os conflitos com o cliente em relação às expectativas de entregas de produtos e cumprimento dos prazos estabelecidos.

Foi observado ainda que, em 52,27% das empresas, os clientes não participam efetivamente das validações dos artefatos e subprodutos dos projetos.

Outro fator relevante observado é o nível de desconhecimento e de não utilização de algumas práticas de engenharia de software por parte das organizações pesquisadas, o que indica que existem melhorias a serem aplicadas no processo de desenvolvimento de software, conforme ilustrado na Tabela 5.5.

Tabela 5.5 – Nível de Não Aplicação das Práticas de Engenharia de Software

Práticas de Engenharia de Software	Nr. Empresas	%
Desenvolvimento <i>Timebox</i>	36	81,82
Participação efetiva do cliente	23	52,27
Particionamento	29	65,91
Desenvolvimento Baseado em Componentes	28	63,64
Reuso de Componentes	25	56,81
Engenharia Simultânea	39	88,64

Como análise final sobre os resultados dessa enquete, considerando os resultados mostrados nas Tabelas 5.6 e 5.7, observa-se que houve redução no tempo de produção do software pelas empresas, quando essas práticas são utilizadas no processo de desenvolvimento de software.

Tabela 5.6 – Resultado da aplicação da prática de Prototipação

Uso da Prototipação	%
Reduziu o conflito com o cliente	58,06
Houve redução do tempo de desenvolvimento	41,94

Tabela 5.7 – Resultado da aplicação das práticas de Componentização e Reuso

Uso da Componentização e Reuso	%
Houve melhoria da qualidade dos projetos	69,70
Houve redução do retrabalho	63,64
Reduziu o tempo de desenvolvimento do software	72,73

Para as práticas de desenvolvimento *timebox* e uso da engenharia simultânea, os resultados sobre a redução do tempo de desenvolvimento não podem ser considerados como conclusivos em função do baixo índice de utilização dessas práticas pelas empresas pesquisadas, menos de 14%.

No entanto, considerando apenas o universo das organizações que fazem uso constante da engenharia simultânea durante o ciclo de desenvolvimento, pode ser observado que em 73,73% das empresas houve melhoria da produtividade e em 63,64% houve redução do tempo de desenvolvimento, conforme ilustrado na Tabela 5.8.

Tabela 5.8 – Resultado da aplicação da prática de Engenharia Simultânea

Uso da Engenharia Simultânea	%
Houve Melhoria de Produtividade	63,64
Reduziu o Tempo de desenvolvimento	73,73

A partir dos resultados apresentados pela enquete verifica-se que, com a aplicação destas técnicas e práticas de engenharia de software, é possível promover a melhoria da produtividade, a diminuição do retrabalho e a redução de tempo no desenvolvimento de software.

5.3. Conclusões do Capítulo

Neste capítulo foi apresentado o resultado da pesquisa realizada pelo MCT (2002) sobre qualidade e produtividade de software no Brasil e os resultados sobre a avaliação da utilização das técnicas e práticas propostas pelo roteiro deste trabalho,

através de uma enquete realizada pelo autor com as empresas de desenvolvimento de software no Brasil.

O resultado da pesquisa do MCT evidencia uma baixa maturidade das empresas em relação ao processo de gestão de projetos, apresentando baixos índices de utilização das técnicas de gestão de riscos e gestão de mudanças que são fundamentais durante o processo de desenvolvimento de software. Porém, essa mesma pesquisa mostra um bom crescimento da utilização de práticas como a orientação a objetos, que melhora o grau do desenvolvimento baseado em componentes e do reuso, além do crescimento do uso da prototipação no processo de desenvolvimento de software.

Esse resultado mostra que as organizações necessitam aprimorar o seu processo de gestão de projetos para melhoria dos resultados de prazo e custo dos projetos e que as práticas de componentização, reutilização e prototipação auxiliam na melhoria da qualidade e da produtividade do software.

A enquete foi realizada com o objetivo de avaliar o grau de utilização das técnicas e práticas de engenharia de software propostas pelo roteiro deste trabalho e a efetividade destas na redução de tempo no desenvolvimento de software.

Os resultados mostram que estas práticas têm baixo nível de uso no processo de desenvolvimento das organizações, principalmente pelo desconhecimento por parte das empresas das técnicas para aplicação das mesmas. No entanto, nas organizações que aplicam essas práticas, têm-se bons resultados no aumento da produtividade, na diminuição do retrabalho e na redução de tempo no desenvolvimento dos projetos.

Como resultado da análise dessas avaliações é possível verificar que a adequada utilização das técnicas e práticas de engenharia de software podem efetivamente auxiliar as organizações no processo de desenvolvimento, proporcionando aumento na produtividade e permitindo a obtenção de ganhos reais no tempo desenvolvimento de software.

6 O ROTEIRO PARA REDUÇÃO DE TEMPO NO DESENVOLVIMENTO DE SOFTWARE

Objetivo do Capítulo

O objetivo deste capítulo é apresentar a estrutura do roteiro para a redução de tempo no desenvolvimento de projetos de software.

A estrutura do roteiro é baseada na análise realizada sobre as principais abordagens sobre a redução de tempo no desenvolvimento de software discutidas no Capítulo 4 e nos resultados das avaliações apresentadas no Capítulo 5.

Os resultados dessa análise permitem definir as práticas e técnicas de engenharia de software para a composição do roteiro e que proporcionam ganhos reais na redução de tempo no desenvolvimento de software.

Este capítulo constitui a contribuição do autor.

6.1. Definição da Estrutura do Roteiro para a Redução de Tempo

A estruturação do roteiro foi elaborada considerando o estudo realizado sobre as principais abordagens sobre a redução de tempo no desenvolvimento de software discutidas no Capítulo 4 e nas avaliações realizadas através da pesquisa do MCT (2002) e da enquete realizada pelo o autor, discutidas no Capítulo 5.

A partir da análise sobre os modelos com foco na redução do tempo de produção de software como o Desenvolvimento Rápido de Aplicação (RAD), o Desenvolvimento Rápido Evolutivo e a abordagem sobre redução do ciclo de desenvolvimento proposta por Collier e Collofello (1995), observa-se que algumas técnicas e práticas são comuns entre os modelos, conforme ilustrado na Tabela 6.1.

Tabela 6.1 – Quadro Comparativo entre as abordagens de Redução de Tempo

Práticas e Técnicas de Engenharia de Software	RAD	Rápido Evolutivo	Collier e Collofello
Prototipação Evolutiva	X	X	X
Interação com o cliente	X	X	X
Componentização			X
Reuso			X
Atividades Concorrentes			X
Desenvolvimento <i>Timebox</i>	X		
Equipes reduzidas	X	X	X
Ferramentas CASE	X		
Simplificação de Tarefas			X
JAD	X		

O uso da prototipação para a identificação e validação de requisitos, o aumento da interação com o cliente durante a execução do projeto e a redução do tamanho das equipes de desenvolvimento (menos de 15 membros) são elementos consolidados em vários modelos e mostram-se efetivos na melhoria da comunicação e na redução dos riscos nos projetos de software (Wetherbe; Frolick, 2000).

O desenvolvimento timebox, embora abordado apenas pelo modelo RAD, é uma prática com características que auxiliam na redução da complexidade do software e na redução dos riscos, estabelecendo prioridades ao desenvolvimento do software e obtendo o comprometimento do cliente com o projeto (Howard, 2002).

As práticas relacionadas ao desenvolvimento baseado em componentes, a reutilização e a execução de atividades concorrentes atuam de forma a aumentar a produtividade e a reduzir substancialmente o tempo de desenvolvimento do software (Blackburn; Scudder; Wassenhove, 2000).

As abordagens desses modelos podem ser classificadas em seis objetivos principais visando promover a redução de tempo no desenvolvimento de software, conforme apresentado na Tabela 6.2.

Tabela 6.2 – Comparativo entre os objetivos das abordagens de Redução de Tempo.

Objetivos	RAD	Rápido Evolutivo	Collier e Collofello
Melhoria da Comunicação	X	X	X
Aumento da Produtividade	X		X
Redução de Riscos		X	X
Controle da Complexidade	X	X	X
Redução do Retrabalho	X	X	X
Manutenção da Qualidade			X

No estudo realizado por Blackburn, Scudder e Wassenhove (2000) com empresas dos EUA, Europa e Japão e na enquete realizada pelo autor, os resultados mostram que a utilização das técnicas e práticas apresentadas na Tabela 6.3, produzem resultados tangíveis em relação à redução do ciclo de desenvolvimento.

Tabela 6.3 – Técnicas e Práticas que Contribuem para a Redução de Tempo

Práticas e Técnicas de Engenharia de Software	Blackburn 2000	Enquete 2005
Engenharia Simultânea	X	X
Componentização	X	X
Reuso	X	X
Prototipação Evolutiva	X	
Prototipação de Interface		X
Simplificação de tarefas	X	X

Considerando os objetivos que orientam os esforços desses modelos para a redução do ciclo de desenvolvimento e os resultados das avaliações sobre a utilização das técnicas e práticas de engenharia de software é possível realizar um mapeamento entre os conceitos que impactam na diminuição de tempo no desenvolvimento de software, conforme ilustrado na Figura 6.1.

5. **Prototipação:** Facilita o entendimento do cliente e auxilia na estabilização dos requisitos;
6. **Desenvolvimento Baseado em Componentes:** Permite aumentar o paralelismo das atividades e viabiliza a reutilização;
7. **Reutilização:** Aumenta a produtividade e provê melhoria da qualidade;
8. **Engenharia Simultânea:** Permite a realização de tarefas concorrentes em todas as fases do projeto, reduzindo o tempo de desenvolvimento.

A Tabela 6.4 mostra a relação entre as práticas e técnicas do roteiro com os objetivos das abordagens de redução de tempo.

Tabela 6.4 – Mapeamento das Práticas e Técnicas com os Objetivos de Redução de Tempo

Objetivos	Práticas e Técnicas
Melhoria da Comunicação	<ul style="list-style-type: none"> • Planejamento Colaborativo • Prototipação
Aumento da produtividade	<ul style="list-style-type: none"> • Arquitetura de Software • Reuso • Engenharia Simultânea
Redução de Riscos	<ul style="list-style-type: none"> • Arquitetura de Software • Prototipação • Timebox
Controle da Complexidade	<ul style="list-style-type: none"> • Arquitetura de Software • Particionamento • Timebox • Reuso
Redução do Retrabalho	<ul style="list-style-type: none"> • Desenvolvimento baseado em Componentes • Reuso • Prototipação
Manutenção da Qualidade	<ul style="list-style-type: none"> • Arquitetura de Software • Desenvolvimento baseado em Componentes • Reuso

Embora essas técnicas e práticas tenham sido selecionadas para constituírem este roteiro, o grupo de atividades apresentado na Figura 6.1 pode ser utilizado como referência para a criação de outros roteiros ou para inclusão de outras técnicas ou práticas de engenharia de software que auxiliem na redução de tempo no desenvolvimento de software.

6.2. Os Elementos do Roteiro para Redução de Tempo no Desenvolvimento de Software

A finalidade deste roteiro é organizar a utilização ordenada e conjunta das práticas e técnicas de engenharia de software descritas no item 6.1 com o objetivo de obter ganhos reais de tempo no processo de desenvolvimento de software.

Com a definição das técnicas e práticas de engenharia de software que são utilizadas neste roteiro, pode-se estabelecer um mapeamento entre esses elementos conforme ilustrado na Figura 6.2.

Figura 6.2 – Mapeamento dos Elementos que compõem o Roteiro Proposto

O processo de desenvolvimento é dividido em três abordagens principais: a elaboração do planejamento, a definição da estratégia da solução e a definição da arquitetura de software a ser utilizada durante o projeto.

No planejamento é definido o ciclo de vida a ser utilizado e os pacotes de trabalho que representam os produtos entregáveis ao cliente durante o projeto.

Orientada pela definição da arquitetura de software, a estratégia da solução avalia e estabelece como o projeto será desenvolvido utilizando as técnicas de prototipação, do desenvolvimento *timebox* e do particionamento, visando simplificar as tarefas, diminuir os riscos e aumentar o grau de atividades concorrentes.

A arquitetura de software deve orientar o desenvolvimento baseado em componentes e o reuso, aumentando a produtividade e a execução das atividades concorrentes.

A execução do projeto é apoiada e orientada por dois processos:

- 1. Processo de gestão de projetos** – Deve haver um processo de gerenciamento de projetos estabelecido com o objetivo de coordenar todas as atividades previstas no roteiro.
- 2. Metodologia de desenvolvimento de software** - Para a produção de software em escala, a metodologia de desenvolvimento deve estar bem definida e de conhecimento de toda a equipe do projeto para proporcionar a correta execução das atividades.

Esses dois processos são considerados pré-requisitos para a aplicação deste roteiro e não são detalhados neste trabalho.

A seguir são descritos os elementos que compõem o roteiro e os pré-requisitos mínimos necessários para a utilização adequada de cada prática ou técnica.

1) Planejamento do Projeto

O planejamento deve ser realizado de forma colaborativa entre a equipe do projeto e o cliente e adequado à realidade das necessidades de cada projeto.

A definição do ciclo de vida de desenvolvimento direciona a escolha entre os modelos espiral ou iterativo por serem mais adequados em função da flexibilidade e da geração de resultados tangíveis mais rápidos.

A análise de viabilidade e a identificação dos pontos de aplicação da engenharia simultânea devem começar a ser verificados a partir do planejamento.

Outro fator a ser observado durante o planejamento é a preparação da reunião de *kickoff* do projeto que deve conter as principais atividades do projeto, os pontos de controle, os principais riscos e as dependências associadas ao projeto.

Os seguintes pré-requisitos devem ser observados para a elaboração do planejamento do projeto:

- Participação efetiva do cliente na tomada de decisões;
- Conhecimento da equipe do projeto nos modelos de ciclo de vida iterativo ou espiral;
- Metodologia de desenvolvimento definida;
- Processo de gerenciamento do projeto estabelecido.

2) Definição da Estratégia da Solução

A estratégia da solução consiste na análise das características do projeto e na definição de como ele será organizado e dividido.

A equipe de projeto, em conjunto com o cliente, deve realizar reuniões para determinar os requisitos principais do projeto com a finalidade de se obter:

1. Redução do tempo de desenvolvimento;
2. Aumento progressivo da visibilidade do projeto;
3. Redução de riscos.

As principais técnicas e práticas que auxiliam na definição dessa estratégia são:

2.1 Particionamento do Projeto

O particionamento visa diminuir a complexidade do projeto e auxilia na definição das atividades concorrentes nas fases iniciais do projeto.

Os seguintes pontos devem orientar a definição das partições do projeto:

- Maximização da independência entre as partições para permitir maior paralelismo das atividades;
- Priorização das principais necessidades de negócio.

2.2 Desenvolvimento *Timebox*

O procedimento central desta prática é enquadrar as principais características do projeto em períodos de tempo que permitam o atendimento das necessidades do cliente dentro do menor tempo possível. Os *timeboxes* definidos devem ser analisados e aprovados em conjunto com o cliente.

Os seguintes pontos devem ser seguidos para a aplicação do desenvolvimento *timebox*:

- O projeto deve estar com o particionamento definido;
- Os *timeboxes* devem ter duração máxima de 120 dias para criar o senso de urgência à equipe (McConnell, 1996);
- Utilização da prototipação evolutiva para auxiliar no entendimento dos requisitos;
- O cliente deve participar da definição dos *timeboxes*.

2.3 Prototipação Evolutiva

A elaboração do protótipo permite a validação do cliente do entendimento dos requisitos e auxilia na definição dos *timeboxes*.

Os seguintes pré-requisitos devem ser atendidos para a utilização da prototipação evolutiva:

- O cliente que participa da definição do protótipo deve ter domínio dos requisitos de negócio e funcionais que deseja ver desenvolvido;

- O código produzido deve estar de acordo com a tecnologia que será utilizada no desenvolvimento para permitir o reaproveitamento do código.

3) Definição da Arquitetura de Software

A definição da arquitetura de software é de vital importância na produção de software em escala e define o sucesso ou insucesso de um projeto (Zhang,2003).

A definição da arquitetura de software a ser utilizada é o passo chave para os efetivos ganhos em tempo de desenvolvimento e produtividade pela equipe do projeto. Além de direcionar a estratégia de desenvolvimento, a arquitetura viabiliza o desenvolvimento baseado em componentes, a reutilização e a utilização da engenharia simultânea.

Para a definição da arquitetura os seguintes pré-requisitos devem ser observados:

- Conhecimento das premissas e restrições de tecnologia do ambiente de desenvolvimento;
- Utilização de desenvolvimento orientado a objetos;
- Experiência da equipe na tecnologia definida.

4) Desenvolvimento baseado em Componentes e Reutilização

Uma vez definida a arquitetura, a modelagem do sistema deve ser cuidadosa e ampla, com o objetivo de definir quais são os componentes de negócio que devem ser desenvolvidos, adquiridos ou reutilizados pelas diversas partições do sistema.

Também devem ser desenvolvidos e aplicados os componentes de infra-estrutura definidos para o ambiente de execução do software. Entre eles têm-se: conexões com a base de dados, controles de sessão, permissões de usuários, aplicações, entre outros.

Com a utilização dos componentes e, conseqüente reutilização do software, a redução de tempo de desenvolvimento dá-se através da redução do esforço de codificação e da eliminação da necessidade de aplicação de testes sobre o software reutilizado.

Para a utilização do desenvolvimento componentizado e da reutilização os seguintes pré-requisitos devem ser atendidos:

- A arquitetura de software deve orientar o desenvolvimento baseado em componentes;
- Existência do repositório ou biblioteca de componentes reutilizáveis;
- Definição do responsável pela administração dos componentes;
- Facilidade de identificação e busca de componentes;
- Documentação de utilização dos componentes atualizada e disponível;
- Processo de divulgação e atualização de componentes estabelecido;
- Treinamento e *mentoring* das equipes de projeto.

5) Engenharia Simultânea

Dentro do roteiro proposto, a engenharia simultânea é o conceito que mais contribui para a redução do tempo de produção do software.

A engenharia simultânea é uma abordagem que permite a realização de várias atividades ao mesmo tempo, sem a eliminação de passos essenciais para a construção do software. Sua aplicação desde as fases iniciais do projeto permite a execução das atividades do projeto com o paralelismo necessário para reduzir o tempo final do processo de desenvolvimento.

Porém, a realização de atividades concorrentes exige uma comunicação eficiente entre os membros da equipe e maior controle durante a execução do projeto visando evitar o retrabalho e o aumento dos riscos no projeto. A Tabela 6.5 mostra os pontos fracos e pontos fortes do uso da engenharia simultânea.

Tabela 6.5 – Pontos Fortes e Pontos Fracos no uso da Engenharia Simultânea

Pontos Fortes	Pontos Fracos
Paralelismo das atividades reduz o tempo de desenvolvimento.	Necessidade de maior grau de controle gerencial na execução das atividades.
Redução de custos.	Mudanças constantes de requisitos não controladas podem gerar retrabalho e aumento de custos.
Produção de resultados mais rápidos.	Alto grau de paralelismo pode aumentar os riscos do projeto.
Melhoria no processo de comunicação.	Resistência ao uso por equipes com aversão ao riscos.
Aumento da produtividade.	

Para reduzir os pontos fracos descritos na Tabela 6.5, as seguintes atividades devem ser consideradas:

- O maior obstáculo para o uso da engenharia simultânea é a constante alteração de requisitos. Esse problema pode ser diminuído gastando-se mais tempo nas fases iniciais do projeto para fortalecer a estabilidade dos requisitos e aderência às necessidades do cliente (Collier e Collofello, 1995);
- Uso da prototipação para auxiliar no entendimento e validação do cliente;
- Utilização de equipes reduzidas para a diminuição dos canais de comunicação;
- Fluxo de comunicação da equipe deve ser livre e informal para garantir a troca de informações;

Outra dificuldade na engenharia simultânea está relacionada em determinar o que será desenvolvido de forma concorrente dentro do projeto. Essa definição está diretamente relacionada às características do projeto que está sendo desenvolvido, ao modelo do ciclo de vida escolhido, o número de recursos disponíveis e a flexibilidade da arquitetura que está sendo utilizada. No entanto, alguns critérios podem ser utilizados como referência para auxiliar nesta definição:

1. A elaboração do protótipo e a definição dos *timeboxes* são atividades que podem ser desenvolvidas de forma concorrente;
2. As partições independentes definidas na estratégia da solução devem ser analisadas para determinar o grau de paralelismo aplicável ao projeto;

3. Essas partições independentes são candidatas a serem subprojetos com desenvolvimento completo até a entrega ao cliente;
4. A definição da arquitetura pode ser paralela à fase de definição de requisitos;
5. O levantamento e análise dos componentes reutilizáveis são atividades independentes podendo ocorrer em qualquer fase do processo de desenvolvimento.

6.3. Fluxo de Atividades para a Aplicação do Roteiro para Redução do Tempo de Desenvolvimento de Software

O conjunto de técnicas e práticas de engenharia de software definidas pelo roteiro deste trabalho devem ser realizados na seqüência estabelecida com o objetivo de viabilizar a utilização da técnica seguinte, e assim sucessivamente, até a obtenção do produto final. Este fluxo de atividades é ilustrado esquematicamente na Figura 6.3.

O fluxo de atividades do roteiro é composto por quatorze atividades, agrupadas em cinco fases de acordo com a correlação entre elas, visando organizar a seqüência de execução e facilitar a aplicação do roteiro.

A seguir são descritos os objetivos das cinco fases definidas pelo roteiro:

Fase 1 - Planejamento do projeto

Consiste na análise dos objetivos e dos requisitos iniciais do projeto visando definir quais são as atividades necessárias para a execução do projeto.

Fase 2 - Definição da Estratégia da solução

O objetivo desta fase é analisar e definir as possíveis estratégias que serão aplicadas no desenvolvimento do projeto.

Fase 3 - Formalização e Início do Desenvolvimento do Software

Visa apresentar para todos os envolvidos no projeto a estratégia de desenvolvimento a ser aplicada durante a execução do projeto e obter sua aprovação.

Fase 4 - Definição dos componentes e reutilização

O objetivo desta fase é publicar os novos componentes e orientar o reuso à equipe do projeto para eliminar as redundâncias e reduzir o tempo de desenvolvimento.

Fase 5 - Desenvolvimento das partições

Caracterizada pela construção das partições definidas na estratégia da solução.

Figura 6.3 – Fluxo de Atividades do Roteiro

Fase 1 – Planejamento do Projeto

A fase 1 consiste em executar as atividades preliminares para analisar os objetivos e requisitos iniciais do projeto a fim de revisar e determinar as atividades necessárias para a execução do projeto, a definição do ciclo de desenvolvimento a ser utilizado e identificar e reunir os principais riscos e pendências sobre o escopo do projeto. Estes dados servem de base para a execução das atividades da fase 2.

- Atividades Executadas

1) Definição da Estrutura Analítica do Projeto (EAP)

Baseado na especificação inicial dos requisitos do projeto e de acordo com o processo de desenvolvimento da empresa deve ser realizada a análise dos principais produtos de entrega ao cliente para elaboração da EAP que é o documento principal para a definição do escopo do projeto e base para futuras negociações de mudanças no projeto.

2) Definição do Ciclo de Vida

Na fase inicial do planejamento é importante a definição do ciclo de vida a ser utilizado durante a execução do projeto, pois é um fator determinante para permitir a redução do tempo de desenvolvimento.

Conforme definições apresentadas no capítulo 2, o ciclo de vida recomendado pelo roteiro é o modelo iterativo e incremental, que proporciona resultados tangíveis rapidamente e maior flexibilidade às mudanças que ocorrem durante o projeto.

3) Preparação para a reunião de *Kickoff* do projeto

A partir da definição da EAP e do ciclo de vida para o projeto, são realizadas as atividades iniciais para análise dos riscos e verificação das pendências e dúvidas sobre os requisitos do projeto e então, mapeadas no

documento de pendências ou no mapa de riscos para futuro acompanhamento.

Nesta atividade, elabora-se o cronograma preliminar de tarefas do projeto, alinhado com o ciclo de vida definido e orientado à construção dos produtos previstos na EAP. Após a conclusão do cronograma, é recomendável a criação da linha de base do projeto que serve de referência para o controle e acompanhamento do prazo e do custo do projeto.

- Produtos de Trabalho Resultantes da Fase 1

Como resultado das atividades realizadas nesta fase, os seguintes artefatos são elaborados:

- **Estrutura Analítica do Projeto (EAP):** Define os produtos que devem ser entregues ao cliente. Deve ser elaborada em formato gráfico para facilitar o entendimento e organizados por fases;
- **Mapa de Riscos:** Os riscos do projeto são identificados, analisados, as respostas aos riscos são desenvolvidas e armazenadas para o acompanhamento durante o projeto. O mapa de riscos deve conter, no mínimo, a descrição do risco, a descrição do evento que indica que o risco está para ocorrer, o grau de impacto e a probabilidade de ocorrência para definição da prioridade, a data limite para resolução, a descrição do impacto, as ações de resposta e contingência a ser aplicada caso o risco venha a ocorrer;
- **Cronograma Preliminar:** Cronograma com a definição das atividades orientado para a construção dos produtos definidos na EAP e de acordo com o ciclo de desenvolvimento definido;
- **Plano do Projeto preliminar:** São identificados os principais elementos que serão utilizados na reunião de início do projeto. Deve conter os seguintes elementos: Objetivo do projeto, as datas de início e término,

cronograma sumarizado, principais pontos de controle, organograma da equipe, EAP, as dependências, premissas e restrições do projeto e os principais riscos envolvidos.

- Restrições Identificadas

Como as atividades desta fase são preliminares, as mesmas podem sofrer alterações durante a execução da fase 2, onde é realizada uma análise mais ampla e com a participação de todos os envolvidos.

A falta de conhecimento do ambiente onde o projeto será desenvolvido pode gerar uma estimativa inicial inadequada sobre a duração das atividades e ou a identificação de riscos que afetam o projeto.

- Aspectos de utilização do roteiro

As atividades e produtos previstos nesta fase são essenciais para o entendimento dos requisitos e estabelecem uma visão inicial sobre as dificuldades do projeto, e devem ser executados obrigatoriamente.

Apenas a atividade de definição do ciclo de vida pode sofrer alterações, pois o modelo iterativo e incremental muitas vezes não é de domínio das empresas e da equipe do projeto, podendo gerar resistências ao roteiro. Neste caso, a única recomendação é a não utilização do modelo cascata, pois este apresenta diversas desvantagens, conforme apontado no Capítulo 2.

Fase 2 – Estratégia da solução

Uma vez que os principais produtos e problemas envolvidos no projeto foram identificados, a próxima etapa é a definição da estratégia a ser aplicada no desenvolvimento do projeto.

Nesta fase, a análise deve estar com foco na definição das principais características do projeto, na priorização das entregas, na escolha da arquitetura de sistema, na pesquisa dos componentes que serão reutilizados e, principalmente, a definição o grau de paralelismo das atividades a ser aplicado no projeto.

Esta fase é determinante para o sucesso do projeto e deve contar com a participação efetiva do cliente para a obtenção do comprometimento e aprovação formal da estratégia de desenvolvimento do software.

- Atividades Executadas

1) Definir o Particionamento

A definição das partições estabelece como o projeto será desenvolvido e entregue ao cliente para validação. A duração destas partições deve estar entre duas e seis semanas dependendo das características de cada projeto, permitindo a verificação rápida dos resultados do projeto.

Duas atividades devem ser executadas para complementarem essa definição:

1.1 – Desenvolver o Protótipo: Desenvolver o protótipo nesta fase pode ser de grande valia para a verificação das principais funcionalidades e entendimento claro das características pelo cliente. O protótipo desenvolvido pode ser evolutivo ou apenas de interface, mas o mais amplo possível para apoiar na definição dos *timeboxes*.

1.2 – Definir os *Timeboxes*: A partir da identificação das principais características do projeto, deve ser desenvolvido em conjunto com o cliente a priorização dos requisitos e o seu agrupamento nas partições definidas de acordo com o prazo disponível e às necessidades do negócio. Dependendo das decisões estabelecidas nos *timeboxes* podem haver revisões nas partições definidas.

2) Definir a Arquitetura do Software

A finalidade desta atividade é apresentar a solução técnica a ser desenvolvida. Devem ser observadas as características do ambiente do cliente e eventuais pré-requisitos estabelecidos sobre o *hardware* disponível, softwares básicos utilizados e as especificações técnicas como linguagem de programação e banco de dados que deverão ser utilizados.

Na inexistência destes requisitos preliminares, a equipe de projeto deve apresentar as soluções técnicas disponíveis no mercado, com as respectivas vantagens e desvantagens para a definição com a equipe técnica do cliente.

A orientação da arquitetura para os sistemas WEB deve seguir o estilo MVC (*Model-View-Controller*) e à utilização de linguagem orientada a objetos, cujas características são direcionadas para o desenvolvimento baseado em componentes e para o reuso.

3) Consultar Componentes para Reuso:

Esta atividade consiste na pesquisa e identificação de possíveis componentes existentes que podem ser reutilizados no projeto. O foco está em verificar os componentes de infra-estrutura e de negócio disponíveis para reuso e verificar a necessidade de aquisição ou desenvolvimento de novos componentes.

4) Analisar a Estratégia de Engenharia Simultânea:

O objetivo desta atividade é identificar, priorizar e verificar o grau de acoplamento entre as tarefas de forma a analisar a viabilidade da sua execução em paralelo.

Deve ser realizada em paralelo com todas as atividades desta fase, pois é através do grau de paralelismo de desenvolvimento das partições que o tempo de desenvolvimento pode ser reduzido substancialmente.

Durante toda a análise da engenharia simultânea deve ser levado em consideração o grau de risco envolvido para cada solução apresentada.

5) Preparar o Plano do Projeto Definitivo:

Baseado nas definições obtidas e validadas nesta fase, o plano do projeto preliminar deve ser alterado para conter o particionamento e a estratégia de paralelismo das partições que será apresentado no início da fase 3.

- Produtos de Trabalho Resultantes da Fase 2

Como resultado desta fase, os seguintes artefatos são produzidos:

- **Roteiro das partições:** Descreve o conteúdo das partições que serão desenvolvidas, a prioridade, a dependência entre elas e a ordem de validação e aceite a ser realizado pelo cliente. Neste artefato que é mostrado o grau de paralelismo aplicado ao projeto.
- **Protótipo:** Apresenta a visão preliminar das funcionalidades que serão desenvolvidas. É refinado durante o ciclo de desenvolvimento dentro de cada partição.
- **Documento de Arquitetura do Sistema:** Documento que define os padrões de desenvolvimento que devem ser seguidos durante a construção das partições. Deve conter o diagrama de distribuição da aplicação, um modelo da arquitetura de software e os padrões de codificação estabelecidos.
- **Lista de Componentes:** Relaciona os componentes identificados para reuso, sua localização e os documentos de referência para sua utilização. Também são descritos aqueles que devem ser adquiridos ou desenvolvidos.
- **Cronograma de Trabalho:** Desenvolvido a partir do cronograma preliminar elaborado na fase 1, acrescido das estratégias de solução definidas nesta fase. Devem ser observados os novos pontos de controle para acompanhamento do projeto e as datas de validações do cliente.
- **Plano do Projeto definitivo:** Adiciona as informações complementares da estratégia da solução ao documento preliminar elaborado na fase 1. As principais informações desta fase estão relacionadas aos *timeboxes* definidos, as partições de desenvolvimento, as definições de arquitetura e

as datas dos principais pontos de controle do projeto e as datas de entregas das partições.

- Restrições Identificadas

Durante a aplicação do roteiro foram observadas as seguintes restrições para esta fase:

- A construção do protótipo nem sempre é possível nesta fase. Neste caso, a definição dos *timeboxes* é baseada na experiência e conhecimento do negócio do representante do cliente e da equipe do projeto;
- No caso de ausência do cliente nesta fase os riscos aumentam substancialmente pela posição unilateral da solução. Este fator pode ser minimizado através da reunião de *kickoff*, cuja presença dos envolvidos é obrigatória e todas as questões sobre o projeto são apresentadas;
- Outra restrição verificada é a posição do cliente em não validar as partições após a entrega, deixando para uma avaliação única após todas as entregas. Nesta situação cabe um esclarecimento e apresentação das vantagens da validação parcial para o sucesso e cumprimento dos prazos.

- Aspectos de utilização do roteiro

A definição dos *timeboxes*, a organização das partições e a definição da arquitetura de implementação são indispensáveis para a aplicação do roteiro.

De acordo com o nível de engenharia simultânea aplicada, os riscos identificados devem ser monitorados e acompanhados com ênfase durante todo o projeto, com o objetivo de minimizar o retrabalho nas tarefas executadas.

A inexistência de componentes de infra-estrutura para reuso afeta diretamente a estratégia da solução. Neste caso, deve ser criada uma partição exclusiva para o desenvolvimento destes componentes para permitir seu reaproveitamento nas demais partições. O tempo gasto para esta construção deve ser recuperado nas partições subseqüentes.

Fase 3 – Formalização e Início do Desenvolvimento do Software

O objetivo desta fase é apresentar para todos os envolvidos no projeto as definições e a estratégia de desenvolvimento a ser aplicada durante a execução do projeto.

A participação das equipes que irão interagir durante o projeto é essencial para se obter o resultado esperado desta fase. A ausência de um destes envolvidos pode comprometer o planejamento e execução do cronograma de trabalho.

Nesta reunião são apresentados os principais marcos de controle, as dependências e os maiores riscos para o sucesso do projeto. Também, devem ser apresentadas as datas onde os clientes são envolvidos para validação das entregas das partições.

- Atividades Executadas

1) Realização da reunião de *Kickoff*:

É a formalização da estratégia de desenvolvimento adotada para o projeto, conforme descrito na fase 2. Deve ter aspecto formal e participação efetiva de todos os envolvidos no projeto para obtenção do comprometimento e aceite do projeto, bem como dos prazos estabelecidos.

Eventuais alterações devem ser registradas e o plano alterado para comportar as referidas solicitações. Porém, tratando-se de alterações relevantes, uma nova reunião deve ser realizada para o aceite definitivo.

2) Início do Ciclo de Desenvolvimento do software:

Obtido o aceite o formal da estratégia de desenvolvimento, dá-se início às atividades previstas no cronograma de trabalho.

- Produtos de Trabalho Resultantes da Fase 3

Como resultado desta fase, o seguinte produto de trabalho é elaborado:

- **Plano do Projeto aprovado:** Todos os artefatos produzidos até esta fase constituem o plano do projeto, não sendo necessária a elaboração de um novo documento. Sua aprovação formal é obtida na reunião de *kickoff*.

- Restrições Identificadas

A principal restrição identificada nesta fase refere-se a ausência de uma das partes envolvidas no projeto na reunião de *kickoff* ou a recusa na realização da mesma. Neste caso, devem ser realizadas reuniões junto ao cliente destacando a importância e os objetivos da realização da mesma.

- Aspectos de utilização do roteiro

A reunião de *kickoff* é obrigatória na aplicação do roteiro. A não realização da reunião ou a ausência de um envolvido torna-se um fator impeditivo para o início do projeto.

Fase 4 – Definição dos componentes e reutilização

À medida que as partições são desenvolvidas, outros componentes de negócio são identificados, podendo ser reutilizados em outras partições. Nesta fase, a ênfase está em publicar estes novos componentes reutilizáveis para a equipe do projeto com o objetivo de eliminar redundâncias e reduzir o tempo de desenvolvimento.

- Atividades Executadas

1) Definir os componentes:

Esta atividade consiste na publicação de novos componentes de negócio que são identificados ou adquiridos durante o desenvolvimento do projeto.

Estes componentes devem ter, no mínimo, definidas as assinaturas de acesso às suas operações e a documentação de sua utilização.

2) Definir a estratégia de reuso:

Baseado na estratégia de reutilização de componentes descrita no Capítulo 3, a equipe do projeto deve identificar os novos componentes e analisar a dependência existente entre eles, compartilhando o conhecimento através do repositório central de componentes e sob a coordenação de um membro da equipe.

- Produtos de Trabalho Resultantes da Fase 4

Como resultado desta fase, os seguintes produtos de trabalho são elaborados:

- **Novos Componentes para reuso:** Relaciona os novos componentes de negócio identificados para reuso, sua localização e os documentos de referência para sua utilização.

- Restrições Identificadas

A maior restrição verificada para esta fase está em não se utilizar linguagens orientadas a objetos na construção dos componentes, tornando-se fator impeditivo para aplicação das atividades desta fase. Pela sua estrutura, a orientação a objetos permite a organização do software em componentes e facilita o reuso em outras aplicações.

- Aspectos de utilização do roteiro

Deve ser dada atenção especial à reutilização dos componentes nos projetos com alto grau de paralelismo, pois a dependência entre as diversas tarefas concorrentes pode dificultar o reuso adequado dentro do projeto.

As atividades de identificação, armazenamento e publicação de novos componentes devem ser contínuas durante todo o projeto para permitir sua reutilização em outros projetos.

Fase 5 – Desenvolvimento das partições

Esta fase é caracterizada pela construção das partições definidas na estratégia da solução. Cada produto da partição é desenvolvido e entregue para a validação do cliente de acordo com a EAP e seguida de uma RTF (revisão técnica formal).

- Atividades Executadas

1) Desenvolvimento da partição:

Constitui na realização das tarefas necessárias para a construção do produto, conforme planejado durante a fase de estratégia da solução e definido na EAP. Devem ser realizadas revisões técnicas formais (RTF) junto ao cliente para o aceite formal de todos os produtos entregáveis, em todas as fases do ciclo de desenvolvimento do software.

- Produtos de Trabalho Resultantes da Fase 5

Como resultado desta fase, os seguintes produtos de trabalho são elaborados:

- **Pacote:** São os artefatos ou executáveis definidos na EAP para a validação do cliente, conforme o cronograma de trabalho.
- **Relatório de Revisões Técnicas Formais:** Para cada artefato ou executável definido como um produto de entrega pela EAP deve ser objeto de revisão junto ao cliente, com o objetivo de se obter o aceite formal do conteúdo do artefato ou da funcionalidade construída.
Deve conter informações relativas ao local, data de realização, participantes, artefatos objetos da revisão, outros anexos de apoio, as conclusões sobre o aceite ou não do material revisado.
- **Documento de Gestão de Mudanças:** Todas as mudanças de requisitos identificadas durante o desenvolvimento da partição devem ser registradas e acompanhadas com a finalidade de servirem de base para a negociação sobre sua incorporação ou não ao projeto.

Deve conter informações sobre a descrição da mudança, impactos sobre o projeto, solicitante da mudança, data da solicitação, grau de dificuldade, estimativas de esforço e prazo para atendimento.

- Restrições Identificadas

Conforme descrito na fase de planejamento do projeto, não é recomendada a utilização do modelo de ciclo de vida em cascata, pois é pouco flexível à incorporação de mudanças e aos resultados produzidos são tardios.

- Aspectos de utilização do roteiro

Eventuais mudanças de escopo devem ser tratadas em uma nova partição de forma a não afetar as datas de entrega planejadas, salvo quando existe uma formalização de aceite do cliente e as novas datas são publicadas a todos os envolvidos. Porém, estas alterações de planejamento devem ser realizadas somente quando a incorporação dos novos requisitos tenha caráter impeditivo para atender às necessidades do negócio.

A Tabela 6.6 apresenta, em resumo, os produtos gerados a partir do roteiro.

Tabela 6.6 – Produtos resultantes da Aplicação do Roteiro

Nr.	Produto	Fase do Roteiro
1	EAP	1 - Planejamento
2	Mapa de Riscos	1 - Planejamento
3	Cronograma Preliminar	1 - Planejamento
4	Plano do Projeto Preliminar	1 - Planejamento
5	Roteiro das partições	2 – Estratégia da Solução
6	Protótipo	2 – Estratégia da Solução
7	Documento de Arquitetura do Sistema	2 – Estratégia da Solução
8	Lista de Componentes	2 – Estratégia da Solução
9	Cronograma de trabalho	2 – Estratégia da Solução
10	Plano do Projeto Aprovado	3 – Formalização e Início
11	Lista de Novos Componentes para reuso	4 – Reuso e Componentes
12	Pacote	5 - Desenvolvimento
13	Relatório da Reunião Técnica Formal	5 – Desenvolvimento
14	Documento de Gestão de Mudanças	5 – Desenvolvimento

6.4. Conclusões do Capítulo

Neste capítulo foi descrita a estrutura do roteiro para redução de tempo no desenvolvimento de software, seus elementos, as condições de utilização de cada técnica e prática de engenharia de software e o fluxo de atividades para a aplicação do roteiro. Foram apresentadas também, as atividades a serem executadas, os produtos de trabalho, as restrições e os aspectos de utilização destas atividades em cada fase do roteiro.

O roteiro apresentado está baseado no planejamento colaborativo com o cliente e na definição da arquitetura de software que viabilizam e permitem a definição das prioridades de desenvolvimento do projeto utilizando o particionamento, os *timeboxes* e a prototipação. Aliado a estas práticas, o desenvolvimento baseado em componentes, do reuso e da execução de tarefas concorrentes aumentam a produtividade e diminuem a complexidade do software reduzindo o tempo na produção do software.

7 APLICAÇÃO DO ROTEIRO EM PROJETOS DE SOFTWARE

Objetivo do Capítulo

O objetivo deste capítulo é apresentar os resultados e as dificuldades encontradas durante a aplicação do roteiro descrito no Capítulo 6 em situações de desenvolvimento de projetos permitindo a avaliação sobre os ganhos de tempo obtidos durante a execução dos projetos.

7.1. Critérios de Escolha do Projeto

Para escolha dos projetos foi analisado o ambiente de desenvolvimento de software no ramo de consultoria e vivenciados pelo autor em sua atividade profissional.

Os projetos de software objetos deste estudo são voltados para sistemas de internet no segmento financeiro, de tamanho médio, com até oito meses de duração, utilizando os ciclos de vida de desenvolvimento espiral ou iterativo e com metodologia orientada a objetos.

Os projetos escolhidos para a avaliação da aplicação do roteiro são apresentados a seguir:

Projeto 1 - Migração de Serviços Financeiros para a plataforma internet;

Projeto 2 - Desenvolvimento de Aplicações Internet com integração com o legado.

7.2. Características dos Projetos e Condições de Aplicação

Os projetos avaliados foram desenvolvidos em um ambiente de produção de software com metodologia de desenvolvimento de sistemas implantada e em uso por toda a equipe de projeto há mais de três anos. Também possuía um processo de gerenciamento de projetos implantado com base nos padrões estabelecidos pelo PMBOK (*Project Management Body of Knowledge*) (PMI, 2004).

As equipes de projetos eram compostas por um gerente de projeto responsável pela coordenação das atividades do projeto, um líder técnico responsável pelas decisões arquiteturais, analistas de sistemas especializados em negócio e codificação, além de equipes contratadas de empresas terceirizadas para o aumento de escala na produção do software.

Os projetos avaliados estavam sob as seguintes condições de desenvolvimento, antes da utilização das técnicas e práticas propostas pelo roteiro:

Projeto 1 - Migração de Serviços Financeiros para a plataforma internet:

Utilização do ciclo de vida cascata e a arquitetura no modelo cliente-servidor. No processo de desenvolvimento do software não era utilizada a orientação a objetos e o desenvolvimento baseado em componentes não existia. O reuso era pouco relevante e a fase de codificação seqüencial. A gestão de mudanças era inexistente e a gestão de riscos estava em fase de implantação.

Projeto 2 - Desenvolvimento de Aplicações Internet com integração com o legado:

O ciclo de vida utilizado era o modelo espiral e a arquitetura definida no padrão MVC (*Model-View-Controller*). A construção do software utilizava orientação a objetos, com desenvolvimento baseado em componentes, porém sem a existência de um repositório central de componentes. A taxa de reuso dos componentes de infra-estrutura (logs, acesso a dados, controle de tempo de sessão) era alto e o reuso dos componentes de negócio incentivado. A engenharia simultânea era aplicada na fase de construção respeitando as partições definidas na fase inicial do projeto. A gestão de mudanças e a gestão de riscos eram amplamente utilizadas e aplicadas.

Com a utilização do roteiro, os projetos avaliados foram desenvolvidos utilizando metodologia orientada a objetos com notação UML (*Unified Modeling Language*) (OMG, 2006), arquitetura MVC (*Model-View-Controller*) e ambiente Java J2EE (*Java 2 Enterprise Edition*).

7.3. Resultados da Aplicação do Roteiro nos Projetos

Os principais resultados obtidos com a aplicação do roteiro proposto nos projetos descritos no item 7.2, são apresentados a seguir.

Projeto 1 - Migração de Serviços Financeiros para a plataforma internet:

- 1) O envolvimento do cliente durante a fase de planejamento permitiu a validação dos requisitos, do protótipo e a definição da estratégia de particionamento das entregas de acordo com as expectativas do cliente;
- 2) O planejamento colaborativo e a aplicação da gestão de riscos durante todo o processo de desenvolvimento permitiram a aplicação de contingências e reservas de tempo estabelecidas no planejamento, bem como verificar e evitar os impactos nos prazos de entrega;
- 3) A aplicação da gestão de mudanças foi fundamental para o cumprimento dos prazos, pois se tratava de um projeto de engenharia reversa de um sistema já existente que tiveram melhorias identificadas durante a execução do projeto. Além disso, as mudanças foram tratadas como novos projetos que geraram novas oportunidades;
- 4) A definição da arquitetura do software e a utilização da orientação a objetos viabilizaram a criação de uma biblioteca de componentes de infra-estrutura e de negócio para a reutilização neste projeto;
- 5) A definição da arquitetura, com a criação dos componentes e o reuso facilitaram a aplicação do desenvolvimento simultâneo das partições planejadas;
- 6) O desenvolvimento de tarefas concorrentes, em todas as fases do projeto, proporcionou os maiores ganhos na redução de tempo no desenvolvimento do software;
- 7) Com a reuso dos componentes houve aumento da qualidade do software e a redução dos custos através da diminuição do número de testes;

- 8) Ganhos em escala nos projetos subseqüentes com economia de custos e esforços obtidos com a padronização da arquitetura, o uso da engenharia simultânea e o aumento do reuso de componentes;
- 9) Redução do tempo de desenvolvimento em aproximadamente vinte e cinco por cento em relação ao processo anterior.

A Tabela 7.1 mostra alguns parâmetros comparativos observados sobre o processo de desenvolvimento existente e os ganhos obtidos com a utilização do roteiro.

Tabela 7.1 – Resultados obtidos com a Aplicação do Roteiro no Projeto 1

Nr	Características	Antes do roteiro	Ganhos após o roteiro
1	Gerência de Mudanças	Não existente	Auxílio na limitação do escopo e manutenção dos prazos.
2	Gerência de Riscos	Baixa utilização	Redução do retrabalho.
3	Ciclo de Vida	Cascata	Espiral.
4	Envolvimento do cliente	Excesso de retrabalho, após a validação do cliente.	Redução dos conflitos e rapidez nas validações.
5	<i>Timebox</i>	Não utilizado	Definição das funções principais junto com o cliente.
6	Uso de Prototipação	Uso limitado de prototipação	Aplicação de protótipo evolutivo na fase de requisitos.
7	Engenharia Simultânea	Uso apenas na codificação	Uso da engenharia simultânea desde as fases iniciais.
8	Desenvolvimento Componentizado	Não utilizado	Definição dos componentes de negócio e de infra-estrutura.
9	Reutilização	Não existia	Início do processo de reuso de componentes
10	Prazo do Projeto	Atrasos de 10% a 40% no prazo planejado.	Entrega das partições no prazo estabelecido.
11	Qualidade do Software	Elevado número de testes e suscetibilidade a erros durante a validação	Melhoria da qualidade do produto pelo uso de componentes e pelo nível de reutilização aplicado.
12	Encerramento do Projeto	Demora no aceite final do cliente.	Aceite realizado durante as entregas das partições facilitou o encerramento do projeto.

Projeto 2 - Desenvolvimento de Aplicações Internet com integração com o legado:

- 1) O planejamento colaborativo trouxe o comprometimento do cliente com as partições definidas e com a estratégia de desenvolvimento apresentada, facilitando as validações e cumprimento dos prazos;
- 2) Para cada partição foi elaborado e validado um protótipo evolutivo com os *timeboxes* definidos durante o planejamento colaborativo;
- 3) A prototipação de interface foi substituída pela prototipação evolutiva permitindo o reaproveitamento do código;
- 4) Como o desenvolvimento baseado em componentes e o reuso já eram utilizados, os ganhos observados foram em relação ao aumento do reuso de componentes de negócio e na definição de um responsável pela administração dos componentes;
- 5) Mesmo com a existência de um responsável pelos componentes, ainda não foi possível a criação do repositório central para compartilhamento e divulgação dos componentes;
- 6) Neste projeto houve um aumento da contratação de serviços de desenvolvimento terceirizado que foi facilitado pela arquitetura definida e a existência de componentes básicos de infra-estrutura e negócio;
- 7) A engenharia simultânea foi aplicada em todas as fases do projeto, inclusive nas atividades envolvendo as empresas terceirizadas, aumentando a escalabilidade de desenvolvimento do software e a redução de tempo no desenvolvimento;
- 8) Redução do tempo de desenvolvimento em aproximadamente quinze por cento em relação ao processo anterior.

A Tabela 7.2 mostra alguns parâmetros comparativos observados sobre o processo de desenvolvimento existente e os ganhos obtidos com a utilização do roteiro.

Tabela 7.2 – Resultados obtidos com a Aplicação do Roteiro no Projeto 2

Nr	Características	Antes do roteiro	Ganhos após o roteiro
1	Planejamento Colaborativo	Conflitos de prioridades e atraso nas validações	Estabelecimento de prioridades conjuntas e comprometimento do cliente.
2	<i>Timebox</i>	Não utilizado	Definição das entregas principais de acordo com o particionamento estabelecido.
3	Uso de Prototipação	Prototipação de interface	Forte utilização da prototipação evolutiva com o aproveitamento do código.
4	Engenharia Simultânea	Uso na codificação	Aplicação em todas as fases do desenvolvimento.
5	Desenvolvimento baseado em componentes	Utilizado	Ampliação da criação de componentes de negócio e na definição de um responsável pelos componentes.
6	Reutilização	Utilizado	Aumento do reuso de componentes de negócio.
7	Uso de Desenvolvimento Terceirizado	Limitado	Aumento de escala através da definição da arquitetura, existência de componentes e reutilização do software.

Com a característica específica desse projeto em utilizar equipes terceirizadas para aumentar a escala de desenvolvimento pôde ser observado que a definição do maior número possível de partições independentes e da arquitetura de software são pré-requisitos para criar as condições necessárias para o desenvolvimento concorrente e aumentar o nível de reutilização dos componentes.

Com a aplicação do roteiro proposto foram verificados resultados tangíveis em relação à redução de tempo no desenvolvimento de software, no aumento da produtividade e no controle da complexidade do software.

7.4. Dificuldades Encontradas na Aplicação do Roteiro

Durante a execução do roteiro em ambos os projetos descritos no item 7.3 foram observadas dificuldades na assimilação dos conceitos propostos, na confiança da equipe de projeto sobre a execução de maior número de tarefas concorrentes, na criação do repositório de componentes, nos resultados efetivos do reuso e na correta utilização da orientação a objetos.

As principais dificuldades verificadas durante a aplicação do roteiro podem ser divididas em duas visões: visão da equipe de projeto e a visão do cliente.

As principais dificuldades encontradas durante a aplicação do roteiro são apresentadas a seguir, podendo variar de acordo com os níveis de maturidade da equipe de projeto e do cliente.

a) Dificuldades encontradas com a equipe de projeto:

- 1) **Dificuldade na mudança do ciclo de desenvolvimento** – A mudança do ciclo de desenvolvimento tradicional encontrou resistência nas equipes acostumadas no processo que dominavam no dia-a-dia da produção de software;
- 2) **Riscos na aplicação da Engenharia Simultânea** – A equipe do projeto tendeu a manter a estratégia de desenvolvimento seqüencial até a codificação, por considerar a aplicação da engenharia simultânea nas fases iniciais do projeto como um risco desnecessário;
- 3) **Falta de um repositório central de componentes** – A orientação para o desenvolvimento baseado em componentes não foi suficiente para sua aplicação dentro do projeto, pois sem a facilidade de consulta e obtenção dos componentes e da documentação de utilização a equipe tendeu a criar seus próprios componentes;
- 4) **Resistência ao reuso** – Aliada a falta do repositório central de componentes, a resistência ao reuso foi outra dificuldade encontrada em função da falta de

confiança dos membros das equipes nos componentes desenvolvidos por outras equipes ou pela má qualidade da documentação desses componentes;

- 5) **Falta de eficiência do processo de comunicação** – O processo formal estabelecido causou dificuldades na interação entre os membros da equipe, afetando a sincronização e acompanhamento das tarefas concorrentes.

b) Dificuldades encontradas com o cliente:

- 1) **Falta de participação de todos os envolvidos do cliente** – Como o cliente possuía várias áreas interessadas ligadas ao projeto, a ausência de uma dessas áreas causou problemas na validação dos produtos e na estratégia de desenvolvimento do projeto;
- 2) **Resistência do cliente em realizar as reuniões de *kickoff*** – A reunião de *kickoff* foi um dos principais elementos de comunicação do projeto e a falta de entendimento de sua finalidade pelo cliente reduziu os benefícios que esta técnica poderia trazer ao projeto;
- 3) **Não aceitação da incorporação das mudanças posteriores às entregas** – O cliente solicitou mudanças durante todo o projeto e insistiu que fossem incluídas na versão que estava sendo desenvolvida. A falta de formalização de um processo de gestão de mudanças e a clareza no procedimento de incorporação de novos requisitos gerou conflitos entre a equipe de projeto e o cliente;
- 4) **Resistência do cliente em receber e validar as entregas parciais** – Mesmo tendo participado do planejamento e aprovado a estratégia do projeto, o cliente não aceitou realizar as validações das entregas parciais do sistema, aguardando que todas as entregas fossem realizadas para proceder sua validação;

- 5) Falta de eficiência do processo de comunicação** – A falta de comunicação em todos os níveis dos envolvidos no projeto, causou conflitos e alterou as expectativas do cliente em relação às mudanças que ocorreram durante o projeto.

Considerando as dificuldades encontradas durante a aplicação do roteiro foram executadas ações corretivas com o objetivo de diminuir ou eliminar seus efeitos, conforme apresentado nas Tabelas 7.3 e 7.4.

Essas ações corretivas tiveram resultados satisfatórios durante a realização dos projetos descritos no item 7.3, proporcionando melhor entendimento dos objetivos e a consolidação das técnicas e práticas propostas pelo roteiro.

As ações junto aos clientes também atenderam às expectativas, porém dependem do relacionamento estabelecido com a equipe de projeto, pois seus efeitos estão diretamente associados à aceitação por parte do cliente e ao nível de maturidade no desenvolvimento de software em que ele se encontra.

Tabela 7.3 – Ações Corretivas para tratar as Dificuldades encontradas com a Equipe de Projeto

	Dificuldades	Ações Corretivas
1	Dificuldade na mudança do ciclo de desenvolvimento.	Treinamento e <i>mentoring</i> com a equipe mostrando as vantagens do modelo iterativo.
2	Riscos na aplicação da engenharia simultânea.	Treinamento e <i>mentoring</i> com a equipe e uso moderado de tarefas concorrentes na fase de construção.
3	Falta de Repositório central de componentes.	Formalização do desenvolvimento baseado em componentes com o início da criação do repositório de componentes e a definição de um responsável pelo gerenciamento do mesmo.
4	Resistência ao reuso.	Com a melhoria da qualidade e a redução do número de testes foi possível mostrar os benefícios do uso de componentes e a melhoria da produtividade.
5	Falta de eficiência no processo de comunicação.	Realização de reuniões rápidas para compartilhar as informações e verificar o <i>status</i> de andamento das atividades do projeto.

Tabela 7.4 – Ações Corretivas para tratar as Dificuldades encontradas com o Cliente

	Dificuldades	Ações Corretivas
1	Falta de participação de todos os envolvidos do cliente.	Realização de reuniões apresentando informações históricas de problemas ocorridos em outros projetos pela falta de participação dos envolvidos nos projetos.
2	Resistência do cliente na realização de reuniões de <i>kickoff</i> .	Reuniões com apresentação das vantagens e objetivos esperados com o <i>kickoff</i> .
3	Não aceitação de incorporação das mudanças posteriores às entregas.	Apresentação dos impactos nos prazos das entregas, caso as mudanças fossem incorporadas imediatamente.
4	Resistência em receber e validar as entregas parciais.	Criação de partições que permitam a avaliação de um ciclo de negócio completo pelo cliente.
5	Falta de eficiência no processo de comunicação.	Realização de reuniões de <i>kickoff</i> nas principais mudanças do projeto e entre as fases para o alinhamento das expectativas de prazo com o cliente.

7.5. Conclusões do Capítulo

Neste capítulo foram descritas os principais resultados obtidos com a aplicação do roteiro e as dificuldades encontradas durante sua utilização em dois projetos de software.

As dificuldades encontradas são limitações para a aplicação do roteiro e foram tratadas conforme as ações corretivas descritas nas Tabelas 7.3 e 7.4 durante a execução dos projetos apresentando resultados satisfatórios durante o processo de desenvolvimento do software.

Os resultados obtidos com a aplicação do roteiro nos projetos apresentados no item 7.3 mostram que a utilização das técnicas e práticas propostas pelo roteiro podem reduzir o tempo no desenvolvimento do software, dependendo do nível de maturidade em que se encontra o processo de desenvolvimento da empresa, a maturidade e o conhecimento da equipe de projeto e do nível de envolvimento do cliente.

8 CONSIDERAÇÕES FINAIS

Objetivo do Capítulo

O objetivo deste capítulo é apresentar as conclusões e as contribuições deste trabalho, bem como descrever os aspectos positivos e as dificuldades encontradas na aplicação do roteiro para a redução do tempo de desenvolvimento de software. Também são descritos possíveis trabalhos futuros que podem ser desenvolvidos a partir dos assuntos aqui discutidos.

8.1. Resumo das Contribuições do Trabalho

O roteiro proposto para a redução de tempo no desenvolvimento de software apresentado neste trabalho, mostra de forma ordenada e interdependente, a utilização das práticas e técnicas de engenharia de software com o objetivo de permitir às empresas realizarem o desenvolvimento de um projeto em tempo reduzido e com maior produtividade.

Nas avaliações realizadas com a utilização do roteiro, os seguintes aspectos positivos devem ser destacados:

1. A aplicação ordenada e conjunta das práticas e técnicas de engenharia de software propostas por este trabalho permite a construção dos projetos de software em um menor espaço de tempo;
2. A participação efetiva do cliente durante o planejamento e da definição da estratégia da solução permite a priorização adequada das partições e do atendimento dos requisitos principais dentro do prazo estabelecido;
3. Durante a definição das partições deve-se procurar diminuir ao máximo o acoplamento entre elas, visando aumentar as possibilidades de desenvolvimento concorrente;

4. A utilização da prototipação evolutiva, de acordo com os padrões de codificação e alinhada à tecnologia na qual o projeto será desenvolvido permite obter ganhos no tempo de desenvolvimento;
5. A definição da arquitetura de software deve priorizar a redução da complexidade e o desacoplamento entre os elementos de software, pois será a base para viabilizar a engenharia simultânea, o desenvolvimento baseado em componentes e o reuso;
6. A aplicação da engenharia simultânea desde as fases iniciais do projeto, constitui um fator relevante para a redução do tempo de desenvolvimento e o aumento da produtividade;
7. Além da questão da redução do tempo de desenvolvimento, a definição dos *timeboxes* e a aplicação da gestão de mudanças durante a construção do software possibilitam a entrega dos produtos dentro do prazo estabelecido;
8. A utilização de equipes reduzidas no projeto melhora a comunicação e facilita a reutilização dos componentes;
9. Através da estrutura definida pelo roteiro é possível aumentar a escala de desenvolvimento com a utilização de serviços terceirizados, sem comprometer a qualidade do software produzido.

No entanto, durante a aplicação do roteiro alguns fatores críticos que afetam a execução do projeto foram observados e devem ser tratados previamente para potencializar os resultados:

1. O alto grau de paralelismo das atividades no projeto necessita de maior controle gerencial para a coordenação das atividades e de um alto nível de comunicação dentro da equipe do projeto;

2. A resistência ao reuso de componentes é um problema cultural dos desenvolvedores e deve ser tratada com especial atenção antes da utilização do roteiro;
3. A implantação do repositório de componentes deve preceder à aplicação do roteiro, pois maximiza o nível de reutilização e auxilia na cultura da equipe de projeto;
4. As práticas e técnicas discutidas no roteiro são aplicáveis a qualquer estilo de linguagem. Porém, a não utilização de linguagens orientadas a objetos pode trazer maiores dificuldades em sua aplicação, especialmente em relação ao desenvolvimento baseado em componentes e à reutilização;
5. A resistência do cliente em participar das definições do planejamento e da estratégia da solução prejudica a aplicação do roteiro, uma vez que não se obtém o comprometimento adequado no processo de desenvolvimento;
6. A incorporação das mudanças de requisitos detectadas durante o desenvolvimento das partições já definidas compromete o cumprimento dos prazos.

8.2. Conclusões do Trabalho

Este trabalho apresentou um roteiro que reúne e organiza técnicas e práticas de engenharia de software com o objetivo de criar condições que permitam a redução no tempo de desenvolvimento de projetos de software.

A arquitetura de software é a base para a implementação do roteiro criando as condições de escalabilidade e flexibilidade necessária para a definição do particionamento do sistema, do desenvolvimento baseado em componentes e da reutilização e na maximização das atividades concorrentes no processo de desenvolvimento do software.

Outro aspecto importante é o planejamento colaborativo que permite o alinhamento das expectativas do cliente com a equipe do projeto em relação a como o software será construído e avaliado por todos os envolvidos.

A engenharia simultânea é a prática que mais contribui para a redução de tempo na construção do software. Para tanto, o planejamento e a arquitetura devem direcionar a solução para permitir o máximo de desacoplamento entre as partições a serem desenvolvidas.

Nos experimentos realizados, a aplicação do roteiro proposto mostrou-se eficaz na redução do tempo de desenvolvimento de software, na redução da complexidade e no aumento da produtividade da equipe do projeto. Porém, outros tipos de projeto ou situações peculiares que não foram observadas durante este estudo podem gerar novos requisitos que necessitariam de estudos complementares para futuras adequações. No entanto, as características do roteiro proposto são genéricas e flexíveis para eventuais adequações às necessidades específicas de desenvolvimento de projetos de software.

8.3. Trabalhos Futuros

Alguns aspectos não abordados neste trabalho podem ser desenvolvidos com o objetivo de expandir as discussões sobre a redução no tempo de desenvolvimento de software e serem objeto de trabalhos futuros:

1. Aplicação do roteiro proposto em outras situações de projeto, com a finalidade de verificar possíveis ajustes e melhorias nas atividades propostas;
2. Enquadramento do roteiro a uma metodologia de desenvolvimento de software para avaliação da sua flexibilidade de adequação aos processos existentes nas empresas;
3. Verificação através de pesquisas e questionários dos resultados obtidos em empresas que utilizaram o roteiro, para avaliação complementar de sua eficácia na redução do tempo de desenvolvimento;

4. Implementação do roteiro em empresas de desenvolvimento de software com o objetivo de estabelecer medidas de ganhos na redução de tempo e indicadores de melhoria contínua no processo de desenvolvimento de software.
5. Criação de novos roteiros a partir do mapeamento dos conceitos que impactam na redução de tempo no desenvolvimento de softwares ilustrados na Figura 6.1 com outras técnicas ou práticas de engenharia de software que auxiliem na redução de tempo no desenvolvimento de software

REFERÊNCIAS BIBLIOGRÁFICAS

AMUNDSEN, M; HUTCHISON, K, **“A Synergistic Approach to Concurrent Engineering”**, ACM, 1990.

BASS; CLEMENTS; KAZMAN, **“Software Architecture in Practice”**, 2. Edição. New York: Ed. Addison-Wesley, 2003.

BERSOFF, E; DAVIS, A, **“Impacts of Life Cycle Models on Software”**, ACM, 1991.

BLACKBURN, J; SCUDDER, G; WASSENHOVE, L, **“Concurrent Software Development”**, ACM, 2000.

BOEHM, B, **“Software Architecture: Critical Success Factors and Cost Driven”**, ACM, 1994.

BOEHM, B; TURNER, R, **“Balancing Agility and Discipline: A Guide for the Perplexed”**. Boston: Ed. Pearson, 2004.

BOYD, W, **“Pesquisa Mercadológica”**. 7ª edição. Rio de Janeiro: Ed. Fundação Getúlio Vargas, 1987.

BROOKES, N.J; BACKHOUSE, C.J, **“Implementing Concurrent Engineering in Different Environment: Factors for Success”**, ACM, 2000.

CLINCY, V, **“Software Development Productivity and Cycle Time Reduction”**, ACM, Consortium Computing Sciences in Colleges, 2003.

COLLIER, K; COLLOFELLO, J, **“Issues in Software Cycle Time Reduction”**, IEEE, 1995.

DWIVEDI, S; SHARAN, R, “**Concurrent Engineering – Why and What ?**”, ACM, 1990.

GARLAN D, “**Software Architecture: a Roadmap**”. In: International Conference on *Software Engineering*. Ireland, 2000. Anais. New York: ACM Press, 2000. P.91 – 101.

HOFMEISTER, C; KRUCHTEN, P; NORD, R, “**Generalizing a Model of Software Architecture Design from Five Industrial Approaches**”, IEEE, 2005.

HOWARD, A, “**Rapid Application Development: Rough and Dirty or Value-for Money Engineering**”, Communications of the ACM, 2002.

IEEE 1471-2000, “**Recommended Practice for Architectural Description of Software -Intensive Systems**“, 2000.

ISO/IEC 12207, “**Software Life Cycle Processes**”, 1995.

ISO/IEC 9126, “**Quality Characteristics and guidelines for their use**”, 1991.

ISO/IEC 14598-2, “**Software production evaluation. Planning and management**”, 2000.

ISO/IEC 14598-3, “**Software production evaluation. Process for developers**”, 2001.

KARANDIKAR, H; FOTTA, M, “**Assessing Organizational Readiness for Implementing Concurrent Engineering Practices and Collaborative Technologies**”, IEEE, 1993.

KENT, S; HOWSE, J; LAUDER, A, “**Modeling Software Components**”, University of Brighton, Reino Unido, 1999.

KERZNER, “**Software Architecture in Practice**”, 2. Edição, Boston: Ed. Addison Wesley, 2003.

KIRK, D, “**A Flexible Software Process Model**”, IEEE, 2004.

KRUTCHEN, P, “**The Rational Unified Process: An Introduction**”, 2. Edição, New Jersey: Ed. Addison Wesley, 2000.

LUQI, “**Software Evolution Through Rapid Prototyping**”, IEEE, 1989.

MARTIN, J, “**Rapid Application Development**”. Nova York: Ed. Macmillan, 1991.

McCONNELL, S, “**Rapid Development: Taming Wild Software Schedule**”. São Paulo: Ed. Microsoft Press, 1996.

MCT WEB SITE, Ministério da Ciência e Tecnologia – Secretaria de Política de Informática, Produtividade e **Qualidade do Software Brasileiro**, Brasília, 2002, disponível em: <<http://www.mct.gov.br/Temas/info/Dsi/Quali2001/remtab2001.htm>>. Acesso em: 30 jan 2006.

OMG WEB SITE, Object Management Group. **Processos e Padrões sobre Orientação a Objetos e UML (Unified Modeling Language)**, disponível em: <<http://www.omg.org>>. Acesso em: 16 nov 2006.

PARK, Y, “**Organizing and Retrieving Class Components based on Types for Reuse**”, ACM, 1999.

PINNA, C. A, “**Um Roteiro Centrado em Arquitetura para Minimização de Riscos e Incertezas em Projetos de Software**”, Dissertação de Mestrado – Escola Politécnica, Universidade de São Paulo, São Paulo, 2004.

PMI, **PMBOK Guide – “A Guide of the Project Management Body of Knowledge”**. Pennsylvania, 2004.

PRESSMAN, R. S., “**Software engineering: A Practitioner’s Approach**”, 6a. Edição, New York: McGraw-Hill, 2005.

PYSTER, A, “**Software Development Productivity**”, ACM, 1982.

RAVICHANDRAN, T; ROTHENBERGER, A, “**Software Reuse Strategies and Component Markets**”, Communications of the ACM, 2003, Volume 46, Nr 8.

RIBEIRO, A; ARAKAKI, R, “**Proposta de um Roteiro para a Redução do Tempo de Desenvolvimento em Projetos de Software**”, in: SUCESU – 7º Seminário de Gestão de Projetos, São Paulo, 2005. **Anais**. São Paulo, 2005b.

RAN, A, “**Fundamental Concepts for Practical Software Architecture**”, ACM, 2001.

SADOU, N; TAMZALIT, D; OUSSALAH, M, “**A unified Approach for Software Architecture Evolution at different abstraction levels**”, 8º International Workshop on Principles of Software Evolution (IWPSE’05), 2005, p. 65-70.

SEI WEB SITE – Software Engineering Institute. Pittsburgh, **Capability Maturity Model Integration**, disponível em: <<http://www.sei.cmu.edu/cmml>>. Acesso em: 15 nov 2006.

STANDISH GROUP, **Chaos Research Report**, West Yarmouth, Massachusetts, USA, 2004, disponível em: <<http://www.standishgroup.com/sampleresearch/PDFpages/extremechaos.pdf>>. Acesso em: 30 jan 2006.

SUBRAMANIAN, N; CHUNG, L, “**Relationship between the Whole of Software Architecture and its Parts: An NFR Perspective**”, 6º International Conference on Software Engineering, 2005.

SMITH, R P, **“The Historical Roots of Concurrent Engineering Fundamentals”**, IEEE, 1997.

VITHARANA, P, **“Risks and Challenges of Component-Based Software Development”**, Communications of the ACM, 2003, Volume 46, Nr 8.

WETHERBE, J; FROLICK, M, **“Cycle Time Reduction: Concepts and Case Studies”**, ACM, 2000.

YOU-SHENG; YU-YUN, **“Architecture-Based Software Process Model”**, ACM, 2003.

ANEXO 1 - QUESTIONÁRIO APLICADO NA ENQUETE

	Questão	R-1	R-2	R-3	R-4	R-5
1	Defina o ramo de atividade de sua empresa:	Consultoria	Indústria	Telecom	Finanças	Outras
2	Qual o número de funcionários de sua empresa?	Até 50	50 a 100	100 a 1000	1000 a 2000	Mais de 2000
3	Sua empresa possui alguma das certificações?	ISO9000	CMM	CMMI	SPICE	
4	Se possuir CMM ou CMMI, qual o nível em que se encontra?	Nível 2	Nível 3	Nível 4	Nível 5	
5	Sua empresa possui profissionais certificados PMP?	Sim	Não			
6	Sua empresa utiliza algum processo de Gestão de Projetos estabelecido?	Não possui	Pouco Utilizado	Utilizado Sempre		
Responda às questões abaixo em relação ao processo de Gestão de Projetos utilizado pela sua empresa:						
6.1	Qual métrica é utilizada para mensuração dos projetos?	Nenhum	Pontos de função	Pontos de casos de uso	Outros	
6.2	O projeto é particionado para reduzir a complexidade?	Nunca	Sempre	Às vezes		
6.3	A gestão de riscos é realizada durante todo o projeto?	Nunca	Sempre	Às vezes		
6.4	Existe um processo de gestão de mudanças de requisitos?	Nunca	Sempre	Às vezes		
6.5	Durante a execução do projeto, qual é a frequência de reuniões de acompanhamento com o cliente?	Por Semana	A cada 15 dias	Por mês	Por produto	
7	Qual metodologia para o desenvolvimento de projetos de software é utilizada?	Não utiliza	Iterativo	Espiral	Cascata	
8	Na fase de planejamento do projeto, sua empresa gera quais dos artefatos listados abaixo?	Cronograma	EAP	Análise de Riscos	Formaliza Gestão de Mudanças	

Reuniões de <i>kickoff</i> são realizadas no início do projeto e de cada fase para alinhar as expectativas dos envolvidos e para a formalização do planejamento do projeto. Com base nesta afirmativa, responda às questões 9 e 10						
9	No início do projeto é realizada uma reunião de <i>kickoff</i> com o cliente ?	Nunca	Sempre	Às vezes		
10	Nas demais fases do projeto a reunião de <i>kickoff</i> é realizada ?	Nunca	Sempre	Às vezes		
<i>TimeBox</i> é uma técnica que tem como objetivo enquadrar as características principais do projeto à restrição do tempo estabelecido para o projeto, permitindo o atendimento das mesmas na prioridade necessária para o atendimento dos prazos. Com base nesta afirmativa responda à questão 11:						
11	Sua empresa conhece ou utiliza a prática denominada <i>TimeBox</i> ?	Não conhece	Conhece e não usa	Utiliza às vezes	Utiliza sempre	
12	Sua empresa utiliza a técnica de prototipação no desenvolvimento de software?	Não utiliza	Utiliza às vezes	Utiliza Sempre		
13	Qual técnica de prototipação é aplicada nos projetos de desenvolvimento de software?	Nenhuma	Protótipo de Interface	Protótipo Evolutivo	Ambos os protótipos	
14	O cliente participa efetivamente das validações de todos os artefatos desenvolvidos ?	Nunca	Sempre	Às vezes		
15	Se as respostas das questões de 9 a 14 foram respondidas como “Sempre”, responda às questões 15.1 e 15.2:					
15.1	Houve redução dos conflitos de escopo com o cliente?	Nunca	Sempre	Às vezes		
15.2	Foi verificado pela sua empresa melhorias no cumprimento dos prazos estabelecidos com o cliente ?	Nunca	Sempre	Às vezes		
16	Sua empresa utiliza orientação a objetos?	Não	Sim			
17	Sua empresa utiliza o modelo MVC (<i>Model-View-Controller</i>) na arquitetura para desenvolvimento de software?	Nunca	Sempre	Às vezes		
18	A arquitetura de implementação da aplicação é definida nas	Nunca	Sempre	Às vezes		

	fases iniciais do projeto?					
19	A fase de projeto orienta o desenvolvimento de componentes para a reutilização?	Nunca	Sempre	Às vezes		
20	O reuso de componentes é difundido em sua empresa de que forma?	Não é difundida	Informal	Formal, sem processo	Formal com processo	
21	Qual o nível de reutilização utilizado em sua empresa?	Nenhum	Dentro de um projeto	Entre vários projetos	Na empresa	
22	Se as respostas de 16 a 19 foram respondidas como “Sempre”, responda às questões 22.1 a 22.4:					
22.1	Houve melhoria da qualidade do software?	Nunca	Sempre	Às vezes		
22.2	Existiram ganhos de produtividade, permitindo a redução do tempo de desenvolvimento?	Nunca	Sempre	Às vezes		
22.3	Houve redução de retrabalho nas fases finais do projeto?	Nunca	Sempre	Às vezes		
22.4	Auxiliou no cumprimento do prazo com o cliente?	Nunca	Sempre	Às vezes		
A Engenharia simultânea (ES) é uma técnica que permite o planejamento da execução de atividades em paralelo durante o ciclo de desenvolvimento do software. Com base nesta afirmativa, responda as questões abaixo:						
23	No desenvolvimento de software sua empresa utiliza Engenharia Simultânea?	Não conhece	Conhece e não usa	Utiliza às vezes	Sempre	
24	Em que fase do ciclo de produção a Engenharia Simultânea é aplicada ?	Requisitos	Análise	Projeto	Código	Testes
25	A ES na fase de codificação é aplicada baseada no reuso?	Não	Sim			
26	Com a aplicação da ES foram verificados ganhos de produtividade nos projetos?	Nunca	Sempre	Às vezes		
27	Com a aplicação da ES houve redução do tempo de desenvolvimento do software?	Nunca	Sempre	Às vezes		