UNIVERSIDADE FEDERAL DA PARAÍBA - UFPB

CENTRO DE CIÊNCIAS E TECNOLOGIA - CCT

DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO - DSC

WILLIAMS ALVES DANTAS

PROCESSO BYTECOM DE DESENVOLVIMENTO DE SOFTWARE

Relatório final para a conclusão da disciplina Estágio Supervisionado

Local: ByteCom Sistemas Ltda.
Orientadora: Francilene Procópio Garcia
Supervisor: Robert Kalley C. Menezes

Campina Grande, Maio/2001

Biblioteca Setorial do CDSA. Maio de 2021.

Sumé - PB

BYTECOM
Av. Aprígio Veloso , 882. Bodocongó
CEP 58109-907- Campina Grande,PB,
Brasil - Fone/Fax: (0xx83) 310-1438
E-mail: bytecom@bytecom.com.br
http://www.bytecom.com.br

AGRADECIMENTOS

Primeiramente à Deus, pela saúde e disposição que Ele me deu para romper barreiras e obstáculos durante toda esta caminhada.

Aos meus pais, Manoel e Pureza, e demais familiares, pela amizade, pelo esforço realizado e confiança depositada em mim e em meus objetivos.

Ao meu sócio Rodrigo, pelo companheirismo e dedicação à empresa durante esses anos.

Ao pessoal do POLIGENE, Lili, Águeda, Robert Kalley e Marcelo Barros pelo esforço realizado, pela dedicação, pelos ensinamentos e por terem partilhado do nosso sonho.

Aos outros colegas do curso que de uma forma ou de outra, ajudaram me durante a caminhada.

Aos demais professores do DSC, pelos ensinamentos que guardarei para sempre.

BYTECOM
Av. Aprígio Veloso , 882. Bodocongó
CEP 58109-907- Campina Grande,PB,
Brasil - Fone/Fax: (0xx83) 310-1438
E-mail: bytecom@bytecom.com.br
http://www.bytecom.com.br

Índice Analítico

ÍNDICE ANALÍTICO 1

1 APRESENTAÇÃO 4
1.1 A EMPRESA 4
1.2 ORGANIZAÇÃO DO RELATÓRIO 4

2 OBJETIVOS 6
2.1 OBJETIVOS GERAIS 6
2.2 OBJETIVOS ESPECÍFICOS 6

3 AMBIENTE DE ESTÁGIO 7
3.1 DADOS DA EMPRESA 7
3.2 NICHO DE MERCADO 7
3.3 ORGANOGRAMA DA EMPRESA 7
3.4 ESTADO DA INFORMÁTICA NA EMPRESA 8
3.5 ASPECTOS POSITIVOS 8
3.6 ASPECTOS NEGATIVOS 9

4 DESCRIÇÃO DO PROBLEMA 10

5 PROPOSTA DE SOLUÇÃO 13

6 ATIVIDADES DESENVOLVIDAS 14
6.1 REVISÃO BIBLIOGRÁFICA 14
6.1.1 Gerência de projetos: MÉTRICAS DE SOFTWARE 14
6.1.1.1 O Processo de Gerência de Projetos 14
6.1.1.2 Medida do Software 17
6.1.1.3 Integração de Métricas Dentro do Processo de Engenharia de Software 20
6.1.2 Administração de Projetos: ESTIMATIVAS 22
6.1.2.1 Observações Sobre a Realização de Estimativas 22
6.1.2.2 Objetivos do Planejamento de Projetos 23
6.1.2.3 Escopo do Software 23
6.1.2.4 Recursos 25
6.1.2.5 Estimativas de Projetos de Software 27
6.1.2.6 Técnicas de Decomposição 28
6.1.2.7 Modelos Empíricos de Estimativas 32
6.1.2.8 Métricas Orientadas a Objeto 34
6.1.3 Gerenciamento de Projetos: PLANEJAMENTO 37
6.1.3.1 Identificação dos Riscos 37
6.1.3.2 Determinação de um Cronograma para o Projeto de Software 38
6.1.4 O Processo de Desenvolvimento de Software 43
6.1.4.1 Desenvolvimento Iterativo e Incremental de Software 43
6.1.4.2 O Processo Unificado de Desenvolvimento de Software 44
6.1.4.3 Programação Extrema 49
6.1.4.4 Controle de Mudanças no Desenvolvimento de Software 50
6.1.4.5 Controle de Versões do Software 52
6.2 ESTÁGIO ATUAL DA EMPRESA NO DESENVOLVIMENTO DE SOFTWARE 56
6.2.1 Planejamento 56

6.2.2 Análise 57
6.2.3 Projeto 57
6.2.4 Implementação 57
6.2.5 Testes 58
6.2.6 Entrega do Produto 58
6.3 DEFINIÇÃO DO PROCESSO BYTECOM DE DESENVOLVIMENTO DE SOFTWARE 59
6.3.1 Critérios adotados 59
6.3.2 Planejamento 61
6.3.2.1 O Documento de Requisitos Básicos - DRB 61
6.3.2.2 O Plano de Negócio Simplificado - PNS 65
6.3.3 Elaboração 65
6.3.3.1 Modelo Conceitual – MC 66
6.3.3.2 Dicionário de Termos do Sistema – DT 66
6.3.3.3 Projeto Arquitetural do Sistema – PAS 66
6.3.3.4 Diagrama de Use Cases – DUC 67
6.3.4 Construção 67
6.3.4.1 Refinamentos de Artefatos 67
6.3.4.2 Projeto de Baixo Nível 68
6.3.4.3 Codificação 69
6.3.4.4 Planos de Testes 70
6.3.4.5 Transição para o Produto 70
6.3.5 Implantação 70
6.3.6 Manutenção e Suporte 71
6.4 FERRAMENTAS DE AUTOMAÇÃO PARA O PROCESSO BYTECOM 73
6.4.1 TeamSource 73
6.4.2 DUnit 73
6.4.3 Microsoft Project 2000 73
6.4.4 Microsoft Office 97/2000 74

7 CONCLUSÃO E SUGESTÕES 75

8 REFERÊNCIAS BIBLIOGRÁFICAS 77

ANEXO – A: FERRAMENTA QUE AUXILIA NO CÁLCULO DO NÚMERO DE PONTOS POR FUNÇÃO DE UMA APLICAÇÃO. 79

ANEXO – B: FORMULÁRIO DE PEDIDO DE MUDANÇAS 82

ANEXO – C: DECLARAÇÃO DA EMPRESA 84

ANEXO – D: PLANO DE ESTÁGIO 85

1 AMBIENTE DO ESTÁGIO 85

1.1 DADOS DA EMPRESA 85
1.2 NICHO DE MERCADO 85
1.3 ESTADO DA INFORMÁTICA NA EMPRESA 85
1.4 DADOS DO(A) ESTAGIÁRIO(A) 86

2 SUPERVISÃO 86

2.1 SUPERVISOR TÉCNICO 86
2.2 SUPERVISOR ACADÊMICO 86

3 RESUMO DO PROBLEMA OBJETO DO ESTÁGIO 87

4 PROPOSTA DE SOLUÇÃO 87

4.1 OBJETIVO GERAL 87
4.2 OBJETIVO ESPECÍFICO 87

5 ATIVIDADES A SEREM DESENVOLVIDAS E CRONOGRAMA 88

5.1 ATIVIDADES 88
5.2 CRONOGRAMA DE ATIVIDADES PROPOSTO 89
5.3 CRONOGRAMA DE ATIVIDADES REALIZADO 89

Av. Aprígio Veloso , 882. Bodocongó
CEP 58109-907- Campina Grande,PB,
Brasil - Fone/Fax: (0xx83) 310-1438
E-mail: bytecom@bytecom.com.br
http://www.bytecom.com.br

1 Apresentação

1.1 A Empresa

A ByteCom Sistemas é uma micro empresa de base tecnológica incubada no centro SOFTEX GENESIS de Campina Grande - POLIGENE. Incubada desde 1998, a empresa investe seus esforços no desenvolvimento e comercialização de sistemas de informação para micro e pequenas empresas da região Nordeste – seu mercado alvo inicial. Como toda micro empresa, a ByteCom Sistemas visa expandir seu mercado para outras regiões através da produção e comercialização de produtos de qualidade que atendam as expectativas dos seus clientes e assim contribuir com o desenvolvimento regional gerando novos empregos e renda para a população.

1.2 Organização do Relatório

O restante desses trabalho está organizado em 06 (seis) capítulos como descreverei em seguida.

No Capítulo 2 são descritos os objetivos, gerais e específicos, em relação a este Estágio Supervisionado.

No Capítulo 3 é feita uma descrição detalhada do ambiente de desenvolvimento de estágio abordando principalmente os recursos de informática disponibilizados para o desenvolvimento do mesmo no ambiente da empresa.

No Capítulo 4 é feita uma descrição de toda a problemática envolvida na forma como a ByteCom Sistemas desenvolve seus produtos.

No Capítulo 5 é feita uma descrição sucinta da proposta de solução para os problemas descritos no Capítulo 4.

No Capítulo 6 são descritas todas as atividades realizadas de acordo com o plano de estágio inicial proposto pela empresa e aprovado pelo Orientador Acadêmico e pelo Coordenador do Estágio Supervisionado do Departamento de Sistemas e Computação – DSC.

No Capítulo 7 são apresentadas conclusões deste estágio, os fatores que mais contribuíram, os que mais dificultaram a realização desse, e ainda, minhas sugestões de como teria sido mais fácil superar tais dificuldades.

No Capítulo 8 é apresentada a Bibliografia consultada durante a realização deste estágio.

Por fim, é anexada toda a documentação gerada durante a realização deste estágio.

2 Objetivos

2.1 Objetivos Gerais

Dotar a empresa ByteCom Sistemas de uma abordagem flexível de desenvolvimento para processos de software que possa ser adotada pelo seu time em situações de mudanças de requisitos, em novos projetos e no suporte ao cliente, de forma a atender as demandas do mercado.

2.2 Objetivos Específicos

Elaborar e implantar um processo de desenvolvimento de software, orientado ao desenvolvimento de sistemas de informação, que apresente condutas gerenciais para planejamento, análise, projeto, implementação, testes e disponibilização de soluções junto aos clientes; (PROCESSO BYTECOM DE DESENVOLVIMENTO DE SOFTWARE)

3 Ambiente de Estágio

3.1 Dados da Empresa

Razão Social: ByteCom Sistemas Ltda.

CNPJ: 03.688.196/0001-47

Endereço: Av. Aprígio Veloso,882 - Bodocongó, Campina Grande, Paraíba

Tele/Fax: (83)310-1438

E-mail: bytecom@bytecom.com.br

URL: http://www.bytecom.com.br

3.2 Nicho de mercado

A ByteCom Sistemas é uma empresa incubada no CENTRO SOFTEX GENESIS de Campina Grande - POLIGENE que atua principalmente na informatização de indústrias/empresas do setor de Alimentação. Os principais clientes da ByteCom Sistemas são panificadoras, confeitarias, restaurantes, fast foods e pizzarias. Além desses, a empresa desenvolve soluções customizadas instituições publicas e privadas de outros setores de atividade.

3.3 Organograma da Empresa

ByteCom Sistemas - Organograma

3.4 Estado da Informática na Empresa

A empresa atualmente utiliza a estrutura do laboratório de prototipagem de software do centro SOFTEX GENESIS de Campina Grande – POLIGENE. No total são 08 (oito) computadores Compaq com processadores Pentium II Celeron 466 MHz, com 64 MB de memória RAM rodando sobre o sistema operacional Windows 98® e um Servidor Netfinity 3500 da IBM rodando o sistemas operacional Windows NT® Server. O laboratório dispõe de uma linha telefônica e fax para auxiliar os negócios da empresa. Além disso, o mesmo está disponível 24 horas por dia incluindo sábados, domingos e feriados. Todos os computadores possuem acesso à Internet 24 horas por dia.

A empresa adotou o Borland Delphi 5.0 na versão Enterprise para desenvolver suas aplicações convencionais e utiliza a linguagem Java® para desenvolver aplicações web. Os pacotes de software instalados nas máquinas são em sua grande maioria registrados e o restante são cópias grátis de avaliação adquiridas na Internet.

A equipe de desenvolvimento é formada basicamente por alunos da graduação do curso de Ciências da Computação e do curso de Desenho Industrial da UFPB, Campus II. O supervisor técnico é **Robert Kalley Menezes**, coordenador do centro SOFTEX GENESIS de Campina Grande – POLIGENE e professor do Departamento de Sistemas e Computação – DSC.

3.5 Aspectos Positivos

Apesar da empresa não ter , ainda, em seu quadro de pessoal nenhum profissional formado na área de informática, a mesma como empresa incubada, usufrui de toda a qualificação, dedicação e experiência do quadro de professores do Departamento de Sistemas e Computação que sempre se mostrou receptivo quando a equipe de desenvolvimento não conseguia resolver problemas de natureza técnica por falta de experiência. Assim, essa "deficiência" sempre pôde ser rapidamente resolvida.

Av. Aprígio Veloso , 882. Bodocongó
CEP 58109-907- Campina Grande,PB,
Brasil - Fone/Fax: (0xx83) 310-1438
E-mail: bytecom@bytecom.com.br
http://www.bytecom.com.br

Um outro aspecto positivo foi o fato de ser um dos sócios da empresa em questão, pois como estaríamos trabalhando em algo nosso, a motivação seria muito maior que se estivéssemos em um outro ambiente, onde provavelmente não estaríamos sendo remunerados e a única motivação seria concluir a graduação, o que poderia afetar a qualidade do trabalho.

3.6 Aspectos Negativos

Apesar da moderna estrutura disponibilizada e facilidade de acesso à mesma, o ambiente utilizado pela ByteCom Sistemas é compartilhado com outras empresas incubadas, o que reduz os recursos disponíveis por empresa e limita algumas de suas ações no sentido de desenvolver mais rapidamente seus negócios.

4 Descrição do Problema

A empresa não possui nenhum funcionário propriamente dito, isto é , todos os outros membros da equipe de desenvolvimento são estagiários alunos do curso de graduação em Ciências da Computação do Centro de Ciências Tecnologia da UFPB/CAMPUS II. Como estudantes que ainda são, eles não trabalham em tempo integral e sua permanência na empresa está limitada ao término do seu curso de graduação, o que ocorre em no máximo 03 (três) períodos letivos após sua contratação. Este fato gera uma grande rotatividade de mão-de-obra que implica em constantes atrasos no cronograma dos projetos e custos com treinamento de novos estagiários.

O negócio da empresa é o desenvolvimento de sistemas de informação para micro e pequenas empresas com características em comuns. No entanto, existem um certo grau de customização de um cliente para outro o que geralmente implica em mudança de requisitos nos projetos originais.

Não é adotado nenhum padrão durante a fase de planejamento dos projetos. As decisões eram tomadas de forma bastante subjetiva, pois não havia quase nenhuma documentação de projetos passados no qual a mesma pudesse se espelhar.

A estimativa dos custos do projeto também era bastante empírica e no máximo se baseava numa pesquisa junto aos concorrentes. A estimativa de tempo de duração ou de entrega do produto ao cliente era baseada em projetos passados, mas devido a pouquíssima documentação, só se tinha uma idéia aproximada de quanto tempo teria durando tal projeto. Quando o projeto era totalmente novo e fora do domínio do conhecimento da equipe, este projeto tinha o tempo de duração alongado o máximo possível e tolerado pelo cliente.

O planejamento de recurso necessário não era realizado de forma adequada. Eram sempre dois coordenadores e três programadores: um dos coordenadores era responsável pelo contato com o cliente e o outro pelo projeto do banco de dados. Um dos programadores era responsável pelo projeto das interfaces com o usuário e codificação, um outro era responsável somente pela codificação e por último, um programador era responsável pelos relatórios do sistema. Estas funcionalidades eram exercidas independentemente da quantidade de projetos

em andamento. Um membro da equipe poderia trabalhar em vários projetos ao mesmo tempo e exercer funções diferentes em cada um deles sem que existisse nenhum plano de atividade que o orientasse nesta transição.

Não era realizada nenhuma análise dos riscos que poderiam posteriormente afetar de qualquer forma a seqüência do projeto.

Existe uma preocupação muito grande com relação a análise do sistema. Há consciência de que sem uma análise não se pode concluir um projeto. Assim, o cliente é entrevistado inúmeras vezes até que todas as dúvidas com relação ao escopo do problema estejam sanadas. No entanto, todo o escopo do projeto sempre fica na cabeça do coordenador e na maioria das vezes apenas dois artefatos são gerados: um Documento de Requisitos Funcionais – DRF e um Diagrama de Entidades e Relacionamentos – DER, com um agravante, não existia nenhum mecanismo de controle de atualizações destes artefatos.

Durante a fase de **projeto**, a divisão de tarefas não era realizada de forma clara ou definida com antecedência. Assim, não era possível se definir um cronograma de acompanhamento e controle de progresso do projeto. A importância dessa parte do processo era invariavelmente negligenciada.

A implementação era (re)iniciada tão logo já existisse um (novo)protótipo do banco de dados ou fosse detectado um outro requisito ou um erro de especificação desse. Um programador somente saberia o que fazer no dia seguinte se a tarefa do dia atual não tivesse sido concluída. Aqui detectou-se o maior índice de retrabalho devido a existência de falha na especificação, análise, projeto e até mesmo falha na comunicação entre os membros da equipe e o coordenador do projeto.

Existia a consciência de que todos os métodos deveriam ser comentados, mas não se tinha noção do quanto seria necessário para se fazer entender ou como fazê-lo usando uma linguagem padronizada e não ambígua.

Não existia um planejamento dos testes o que causava um número muito alto de retornos dos clientes até que o sistema se tornasse progressivamente mais estável. No caso de uma atualização, os teste nunca eram refeitos ou sequer documentados. Fruto de falhas durante a coleta de requisito não-funcionais e de um mal planejamento desses testes, algumas restrições

Av. Aprígio Veloso , 882. Bodocongó
CEP 58109-907- Campina Grande,PB,
Brasil - Fone/Fax: (0xx83) 310-1438
E-mail: bytecom@bytecom.com.br
http://www.bytecom.com.br

somente eram detectadas meses após a liberação do produto para o cliente, o que agravava mais ainda a situação do projeto.

Não existia nenhuma preocupação com a embalagem do produto. Quando o produto estava "pronto" era instalado na máquina do cliente sem manual do usuário, ajuda on line e nenhum controle de versões . O procedimento era o mesmo tanto na primeira entrega do produto quanto nas entregas das atualizações seguintes.

A empresa possui uma estrutura organizacional centralizadora. Todo o processo de tomada de decisão está concentrado nos dois sócios da empresa. Este fato poderia dificultar uma tentativa de implantação de um processo produtivo de qualidade na empresa.

5 Proposta de Solução

Adequar e implantar um processo de desenvolvimento de software que apresente condutas gerenciais para planejamento, análise, projeto, implementação, testes, disponibilização de soluções junto aos clientes e manutenção dos produtos, ou seja, determinar o PROCESSO BYTECOM DE DESENVOLVIMENTO DE SOFTWARE.

Antes de qualquer ação no sentido de implantar o PROCESSO BYTECOM DE DESENVOLVIMENTO DE SOFTWARE, é necessário definir um organograma para a empresa e sobre cada departamento deste, definir seu fluxograma de informações, embora não seja objeto do plano de estágio, pois sem estas definições o processo implantado dificilmente será repetitível e facilmente cairá em desuso.

O próximo passo será realizar um levantamento do nível atual de maturidade da empresa no desenvolvimento de produtos de software, envolvendo as fases de planejamento, análise, projeto, implementação, teste e entrega do produto ao cliente.

O passo seguinte será elaborar um "**Processo ByteCom**", centrado na abordagem iterativa, capaz de se adaptar às mudanças nos requisitos do cliente, contemplando: condutas para estimar tempo e custos, templates de controle, planos de testes, e gestão de versão e tudo baseado no levantamento das deficiências detectadas anteriormente. Dando seqüência a este procedimento serão gerados documentos básicos para mapeamento de atividades chaves durante o processo de desenvolvimento - **Formato ByteCom**.

Após definir o Processo ByteCom, será eleita um conjunto de ferramentas de apoio que facilite a implantação do processo na empresa e aumente a produtividade dos membros da equipe de desenvolvimento.

Por fim, o "Processo Bytecom" será aplicado em sistemas existentes para validação/refinamento de atividades chaves sugeridas.

6 Atividades Desenvolvidas

Segue o resumo dos principais tópicos revisados:

6.1 Revisão Bibliográfica

Primeiramente foi realizada uma ampla revisão bibliográfica que abrangeu desde a fase de engenharia de sistemas até a fase de distribuição do produto.

6.1.1 Gerência de projetos: MÉTRICAS DE SOFTWARE

6.1.1.1 O Processo de Gerência de Projetos

A gerência de projetos é a primeira camada do processo de engenharia de software. Nós chamamos de camada porque ela abrange o processo desde o início até o fim.

Para conduzir um projeto de software bem-sucedido, devemos compreender o problema, o objetivo, o escopo do trabalho a ser feito, os riscos que incorremos, os recursos exigidos, as tarefas a serem executadas, os marcos de referencia a serem acompanhados, o esforço (custo) despendido e a programação a ser seguida. A gerência de projeto de software oferece essa compreensão. Ela começa antes do trabalho técnico, prossegue à medida que o software se desenvolve do modelo conceitual para a realidade e encerra somente quando o software se torna obsoleto e é aposentado.

6.1.1.1.1 Iniciando um Projeto de Software

Antes que um projeto de software possa ser planejado, os objetivos e escopo devem ser estabelecidos, soluções alternativas devem ser consideradas e as restrições administrativas e técnicas devem ser identificadas. Sem essas informações, é impossível definir estimativas de custo razoáveis (e mais precisas), uma divisão realística de tarefas de projeto ou uma programação administrável que ofereça indícios significativos de progresso.

O desenvolvedor de software e o cliente devem reunir-se para definir os objetivos e o escopo do projeto. Os objetivo identificam as metas globais sem levar em consideração como

essas metas serão atingidas. O escopo identifica as funções primárias que o software deve realizar, e o que é mais importante, tenta delimitar essas funções de uma forma quantitativa.

Logo que os objetivos e o escopo do projeto forem compreendidos, soluções alternativas serão consideradas. Ainda que poucos detalhes sejam discutidos, as alternativas possibilitam que gerentes e profissionais selecionem uma abordagem "melhor", dadas as restrições impostas por prazos de entrega, orçamento, disponibilidade de pessoal, interfaces técnicas, e uma infinidade de outros fatores.

6.1.1.1.2 Medidas e Métricas

Na maioria dos empreendimentos técnicos, as medições e as métricas ajudam-nos a entender o processo técnico usado para desenvolver um produto, como também o próprio produto resultado desse processo. O processo é medido, num esforço para melhorá-lo. O produto é medido, num esforço para aumentar sua qualidade.

6.1.1.1.3 Estimativas

Uma das atividades fundamentais do processo de gerenciamento de projetos de software é o planejamento. Quando o projeto de software é planejado, estimativa do esforço humano exigido (comumente, pessoa-mês), duração cronológica do projeto (em tempo de calendário) , e custo (em reais) devem ser realizadas. Desta forma, uma série de técnicas de estimativas foram disponibilizadas para o desenvolvimento de software. Sendo que todas têm os seguintes atributos em comum:

- O escopo do projeto deve ser estabelecido antecipadamente;
- Métricas de software são utilizadas e o histórico das aferições passadas é usado como base a partir da qual estimativas são feitas;
- O projeto é dividido em pequenas partes que são estimadas individualmente.

6.1.1.1.4 Análise dos Riscos

Sempre que um programa de computador for construído haverá áreas de incerteza. Com isso aparecem os seguintes questionamentos:

- As necessidades do computador são de fato entendidas ?
- As funções que devem ser implementadas podem ser realizadas antes do prazo final do projeto ?
- Surgirão problemas técnicos difíceis que atualmente estão fora de nossa visão?
- As mudanças que invariavelmente ocorrem durante qualquer projeto farão com que a programação se desvie muito do curso ?

A análise dos riscos é crucial para um bom gerenciamento de um projeto de software. Assim , a identificação, a avaliação e a disposição dos riscos por ordem de prioridade, a estratégia de administração, a resolução e a monitoração dos riscos são passos que nos possibilitam ter um maior controle sobre os riscos.

6.1.1.1.5 Determinação de Prazos

Todo o software tem uma programação de atividades para sua realização, mas nem todos os programas são criados igualmente. A diferença está nas respostas às seguintes questões:

- A programação foi desenvolvida por si mesma ou foi planejada antecipadamente?
- O trabalho foi feito sem critérios ou um conjunto de atividades bem definida foi identificado?
- Os gerente se concentraram somente na data final de entrega ou um curso bem definido foi identificado para garantir que o prazo final seja cumprido?
- O progresso foi medido por "Nós já o fizemos?" ou um conjunto de marcos de referencia uniformemente espacejados foi estabelecido?

A programação de um projeto de software não é , de fato, absolutamente diferente de qualquer projeto de engenharia.

- Um conjunto de tarefas de projeto é identificado;
- Interdependências entre tarefas são estabelecidas;
- O esforço (custo) associado a cada tarefa é estimado;
- Pessoas e outros recursos são atribuídos;
- Uma "rede de tarefa " é criada;
- Um gráfico de Gant (time line) é desenvolvido.

6.1.1.1.6 Monitoração e Controle

Logo que a programação de desenvolvimento for estabelecida, iniciar-se-á a atividade de monitoração e controle. Cada tarefa anotada no programa é rastreada pelo gerente de projeto. Se a tarefa não acompanhar a programação, o gerente pode usar uma ferramenta de planejamento e controle de projetos automatizada para determinar o impacto do não cumprimento sobre os marcos de referencia intermediários e a data de entrega global. Recursos podem ser redirecionados, tarefas reorganizadas ou (por último recurso) , compromissos de entrega modificados para acomodar o problema que foi descoberto. Dessa forma, o desenvolvimento de software pode ser bem mais controlado.

6.1.1.2 Medida do Software

A medição é uma coisa comum no mundo da engenharia. Infelizmente, a medição está longe de ser uma atividade comum no mundo da engenharia de software. Temos dificuldade em concordar sobre o que medir e dificuldades para avaliar as medidas que são obtidas. Mesmo assim, o software é medido por muitas razões:

- indicar a qualidade do produto;
- avaliar a produtividade das pessoas que produzem o produto;
- avaliar os benefícios (em termos de produtividade e qualidade) derivados de novos métodos e ferramentas de software;
- formar uma linha básica para estimativas;
- ajudar a justificar os pedidos de novas ferramentas ou treinamento adicional;

afinal, se não medirmos, não haverá nenhuma maneira real de determinarmos se estamos ou não melhorando. E se não estamos melhorando, estamos perdidos.

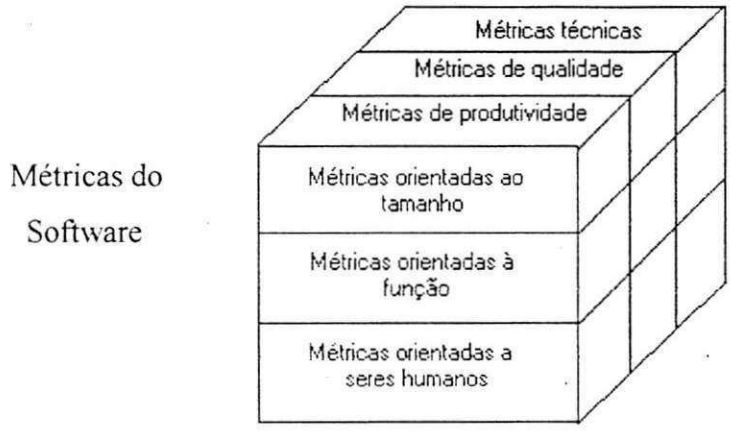


Figura: Divisão das métricas em categoria

As medições do mundo físico podem ser divididas em duas categorias: medidas diretas e medidas indiretas. De acordo com a figura 4.1, as métricas de software podem de ser divididas em categorias:

- **Métricas de produtividade:** concentram-se na saída do processo de engenharia de software;
- **Métricas de qualidade:** oferecem uma indicação de quão estreitamente o software conforma-se às exigências implícitas e explícitas do cliente;
- **Métricas técnicas:** concentram-se na característica do software (por exemplo, complexidade lógica, grau de modularidade) e não no processo por meio do qual o software foi desenvolvido.

De acordo com a figura 4.1, existe mais uma divisão em categorias:

- **Métricas orientadas ao tamanho:** são medidas diretas do software e do processo por meio do qual ele é desenvolvido.
- **Métricas orientadas à função:** oferecem medidas indiretas do software e do processo pelo qual o mesmo é desenvolvido;

- **Métricas orientadas a seres humanos:** compilam informações sobre a maneira segundo a qual as pessoas desenvolvem software de computador e percepções humanas sobre a efetividade das ferramentas e métodos.

O custo e o esforço exigidos para se construir o software, o número de linhas de código produzido e outras **medidas diretas** são relativamente fáceis de ser reunidas, desde que convenções específicas para medição sejam estabelecidas antecipadamente. Porém, a qualidade e a funcionalidade do software, ou sua eficiência e capacidade de manutenção, são mais difíceis de ser avaliadas e somente podem ser **medidas indiretamente**.

6.1.1.2.1 Métricas Orientadas ao Tamanho

A unidade básica de medida é a **LOC** - *line of code*, ou simplesmente, linhas de código, que é obtida a partir da contagem das linhas de código do programa fonte. A partir deste número, outras variáveis podem ser derivadas:

Produtividade = KLOC/pessoa-mês

Qualidade = defeitos/KLOC

Custo = R$/LOC

Documentação = Nº de pág. de documentação/KLOC

As métricas orientadas ao tamanho provocam controvérsias e não são universalmente aceitas como a melhor maneira de se medir o processo de desenvolvimento de software. A maior parte da controvérsia gira em torno do uso da linha de código como uma medida-chave. Os opositores a esta forma de medir argumentam que LOC é uma medida dependente da linguagem de programação que penaliza programas bem projetados, e assim, mais curtos, e que o nível de detalhes necessário para se poder estimar é muito difícil de ser alcançado, visto que o planejador deve estimar as linhas de código (LOC) a serem produzidas muito antes de obter detalhes dos requisitos do projeto.

6.1.1.2.2 Métricas Orientadas à Função

A unidade básica são os **FP-** *Function Point*, ou simplesmente, ponto-por-função, que são obtidos usando-se uma relação empírica baseada em informações e complexidade do software. A partir deste número, outras variáveis podem ser derivadas:

Produtividade = FP/pessoa-mês

Qualidade = defeitos/FP

Custo = R$/FP

Documentação = Nº de pág. de documentação/FP

A métrica de ponto-por-função , como as linhas de código (LOC) é controversa, apesar de resolver os principais problemas encontrados com as linhas de código. Os opositores argumentam que a contagem se baseia parcialmente em dados subjetivos, que os dados sobre o domínio da informação podem ser difíceis de ser compilados a *posteriori* e que principalmente, FP não tem nenhum significado físico direto- é apenas um número.

6.1.1.3 Integração de Métricas Dentro do Processo de Engenharia de Software

Para se conseguir a integração de uma métrica de software ao processo de engenharia de software de uma empresa é necessário fique bem claro a importância desta prática para projetos futuros principalmente. Desta forma, apesar do trabalho árduo, os benefícios da medição são tão motivadores que vale a pena o esforço.

6.1.1.3.1 Estabelecimento de uma Linha Básica (Baseline)

A linha básica consiste em uma planilha de dados compilados de projetos passados de desenvolvimento de software. Esta planilha pode ser simples ou complexa, depende da disponibilidade dos dados.

Quando se estabelece uma linha básica de métricas os benefícios podem ser obtidos em nível estratégico, técnico e de projeto. No entanto, para que auxilie efetivamente no

planejamento estratégico e/ou de estimativas de custo e esforço, os dados da linha básica devem dispor das seguintes características:

- Os dados devem ser razoavelmente precisos - evite "chutes" sobre projetos passados;
- Os dados devem ser obtidos de tantos projetos quanto for possível;
- As medições devem ser consistentes, por exemplo, a linha de codigo deve ser interpretada consistentemente ao longo de todos os projetos para os quais são compilados dados;
- As aplicações devem ser idênticas ao trabalho a ser estimado.

Segue um exemplo de uma planilha que pode ser usada para a determinação da linha básica de métricas.

Atividade	Tamanho		Homens/Horas		Erros		Inicio		Fim	
	Estimado	Real	Estimado	Real	Achados	Solucio -nados	Data Estimada	Data Real	Data Estimada	Data Real
Requisitos										
Planeja- mento										
Projeto										
Implemen- tação										
Teste										
Documen- tação										
Entrega										
Gerência										
TOTAL										

Tabela 6.1: tabela utilizada para se determinar a linha básica de métrica

6.1.1.3.2 Coleta, Computação e Avaliação das Métricas

A coleta dos dados requer uma investigação histórica dos projetos passados para se reconstruir os dados exigidos. Logo que os dados forem coletados, parte mais difícil, então será possível computação das métricas. Dependendo da amplitude dos dados coletados, as métricas podem abranger uma ampla variedade de medidas LOC ou FP. Finalmente, os dados computados deve ser avaliados e aplicados na estimativa. A avaliação dos dados concentra-se nas razões subjacentes para os resultados obtidos.

- As médias computadas são pertinentes ao projeto que se tem em mãos ?

- Quais circunstâncias invalidam certos dados em uso nessa estimativa ?

Essas e outras questões devem ser encaminhadas de forma que os dados das medições não sejam usados às cegas.

6.1.2 Administração de Projetos: ESTIMATIVAS

6.1.2.1 Observações Sobre a Realização de Estimativas

- A estimativa dos recursos, custo e programação de atividades para um esforço de desenvolvimento de software exige experiência, acesso a boas informações históricas e a coragem, para se comprometer com medidas quantitativas quando dados qualitativos forem tudo o que existir;

- A complexidade do projeto tem um forte efeito sobre a incerteza que é inerente ao planejamento. A complexidade, porém, é uma medida relativa que é afetada pela familiaridade que se tem com o esforço passado;

- O tamanho do projeto é outro fator importante que pode afetar a precisão e a eficácia das estimativas. À medida que o tamanho aumenta, a interdependência entre os vários elementos do software cresce rapidamente. A decomposição do problema se torna mais difícil, pois os elementos decompostos podem ainda ser complexos;

- O grau de estrutura do projeto também exerce um efeito sobre os riscos da estimativa. Nesse contexto, a palavra estrutura refere-se à facilidade com que as funções podem ser dispostas em compartimentos e à natureza hierárquicas das informações que devem ser processadas;

- A disponibilidade de informações históricas também determina os riscos da realização de estimativas. Ao olhar-mos para trás, podemos imitar coisas que deram certo e

melhorar áreas onde surgiram problemas. Quando métricas de software aferidas em projetos passados estão à disposição, estimativas podem ser feitas com maior segurança, prazos podem ser estabelecidos para se evitar as dificuldades passadas e os riscos globais são reduzidos;

• Os riscos são medidos pelo grau de incerteza das estimativas quantitativas estabelecidas para recursos, custos e prazos. Se o escopo de um projeto for mal compreendido ou os requisitos de projeto estiverem sujeitos a mudanças, a incerteza e os riscos tornam-se perigosamente elevados. O planejador de software deve exigir inteireza das definições de função, desempenho e interface (contida numa Especificação de Sistema). O planejador e, principalmente, o cliente devem reconhecer que a variabilidade de requisitos de software significa a instabilidade de custos e de prazos.

6.1.2.2 Objetivos do Planejamento de Projetos

O objetivo do planejamento de projeto de software é fornecer uma estrutura que possibilite ao gerente fazer estimativas razoáveis de recursos, custos e prazos. Essas estimativas são realizadas dentro de um plano de tempo limitado ao inicio de um projeto de software e devem ser regularmente atualizadas à medida que o projeto progride.

Como observamos anteriormente, o objetivo do planejamento é atingido por meio um processo de descoberta de informações que leve a estimativas razoáveis.

6.1.2.3 Escopo do Software

O escopo do software descreve a função, o desempenho, as restrições, as interfaces, e a confiabilidade do software. As funções descritas na declaração do escopo são avaliadas, e em certos casos, refinadas para oferecer mais detalhes antes do início da estimativa. Uma vez que tanto as estimativas de custo quanto as de prazo são funcionalmente orientadas, certo grau de decomposição muitas vezes é útil. As considerações referentes a desempenho abrangem os requisitos de processamento e de tempo de resposta. As restrições identificam os limites

impostos ao software pelo hardware externo, memória disponível, ou outros sistemas existentes.

O software interage com outros elementos do sistemas baseados em computador. O planejador considera a natureza e a complexidade de cada interface para determinar quaisquer efeitos sobre os recursos, custos e prazo de desenvolvimento. O conceito de interface é interpretado de maneira a ter os seguintes significados:

- Hardware (por exemplo, processador, periféricos) que executa o software e dispositivos (por exemplo, máquinas, *displays*) que são indiretamente controlados pelo software;

- Software que já existe (por exemplo, rotinas de acesso a bancos de dados, pacotes de sub-rotinas, sistema operacional) e que deve ser ligado ao novo software;

- Pessoas que fazem uso do software por meio de terminais ou outros dispositivos de entrada/saída (E/S);

- Procedimento que precedem ou sucedem o software como uma série seqüencial de operações.

Em cada caso, a transferência de informações por meio de interface deve ser claramente compreendida.

Se uma Especificação de Sistema tiver sido adequadamente desenvolvida, quase todas as informações exigidas numa descrição do escopo do software estará à disposição e será documentada antes que o planejamento do processo de software se inicie. Em casos em que uma especificação não tenha sido desenvolvida, o planejador deve assumir o papel de analista de sistemas e determinar os atributos e ligações que influenciarão as tarefas de estimativas.

6.1.2.4 Recursos

A segunda tarefa de planejamento de software é a estimativa dos recursos exigidos para se levar a efeito o esforço de desenvolvimento do software. Cada recurso é especificado segundo quatro característica:

- Descrição do recurso;
- Declaração da disponibilidade;
- Tempo cronológico em que o recurso será exigido;
- Por quanto tempo o recurso será aplicado.

A disponibilidade de um recurso para uma atividade deve ser estabelecida o mais cedo possível.

6.1.2.4.1 Recursos Humanos

O planejador começa a avaliar o escopo e a selecionar as habilidades exigidas para concluir o desenvolvimento. Tanto os postos organizacionais (por exemplo, gerente, engenheiro de software sênior, testadores etc) quanto as especialidades (por exemplo, telecomunicações, banco de dados, microprocessador) são especificados. Para projetos relativamente pequenos (1 pessoa-ano ou menos), uma única pessoa pode executar todas as etapas de engenharia de software, consultando especialistas quando necessário.

O número de pessoas exigido num projeto de software pode ser determinado somente depois de uma estimativa do esforço de desenvolvimento (por exemplo, pessoas-mês ou pessoas-ano) for feita.

6.1.2.4.2 Recursos de Hardware

Três categorias de hardware devem ser consideradas durante o planejamento do projeto:

- o **hardware de desenvolvimento** – também chamado de *host system* , é o computador e os periféricos relacionados que serão usados durante o desenvolvimento do software. É utilizado porque pode suportar múltiplos usuários, guardar grandes volumes de informação que podem ser compartilhadas com os outros membros da equipe de desenvolvimento de software;

- o **hardware de produção** – no qual o software será por fim executado;

- outros elemento de hardware do novo sistema.

Uma vez que a maioria das empresas de desenvolvimento de software possuem múltiplo clientes que requerem acesso ao hardware de desenvolvimento, o planejador deve prescrever cuidadosamente o espaço de tempo exigido e providenciar que os recursos estejam disponíveis. Cada elemento de hardware deve ser especificado pelo planejador.

6.1.2.4.3 Recursos de Software

Atualmente, os engenheiros de software usam um conjunto de ferramentas denominado engenharia de software auxiliada por computador (CASE - Computer-Aided Software Engineering). Estas ferramentas são organizadas em categorias:

- Ferramenta de Planejamento de Sistemas de Informação;
- Ferramentas de Gerenciamento de Projetos;
- Ferramentas de Apoio;
- Ferramentas de Análise e Projeto;
- Ferramentas de Programação;

- Ferramentas de Integração e Testes;
- Ferramentas de Construção de Protótipos e Simulação;
- Ferramentas de Manutenção;
- Ferramentas de Framework;

6.1.2.4.4 Reusabilidade

Qualquer discussão sobre o recurso de software seria incompleta sem o reconhecimento da reusabilidade - ou seja, a criação e reuso dos blocos de construção de software. Tais blocos deve ser catalogados para que possam ser facilmente consultados, padronizados para facilitar a aplicação e validados para que sua integração seja fácil.

Duas " regras " devem ser consideradas pelo planejador de software quando um software reusável for especificado como um recurso:

1. Se o software existente cumprir os requisitos, adquira-o . O custo de aquisição de um software existente quase sempre será menor que o custo para desenvolver um software equivalente.

2. Se o software existente exigir "alguma modificação" antes que possa ser integrado ao sistema, proceda cautelosamente. O custo para modificar um software existente às vezes pode ser maior que o custo para desenvolver um software equivalente.

6.1.2.5 Estimativas de Projetos de Software

As estimativas de custo e esforço de software jamais serão uma ciência exata. Muitas variáveis – humanas, técnicas, ambientais e políticas – podem afetar o custo final do software e do esforço aplicado para desenvolvê-lo. Entretanto, as estimativas de projetos de software podem ser transformadas em uma série de passos sistemáticos que oferecem estimativas com riscos aceitáveis.

Para conseguirmos estimativas de custo e esforço confiáveis, uma série de opções se apresenta:

1. Atrasarmos as estimativas até um ponto tardio do desenvolvimento do projeto (obviamente, podemos conseguir estimativas 100% precisas depois que o projeto tiver sido concluído!).

2. Usarmos técnicas de decomposição relativamente simples para gerar estimativas de custo e de esforço de projeto.

3. Desenvolvermos um modelo empírico para o custo e o esforço de software.

4. Adquirirmos uma ou mais ferramentas de estimativas automatizadas.

Infelizmente, a primeira opção, ainda que atraente, não é prática. As estimativas de custo devem ser oferecidas " logo de início". Porém, devemos reconhecer que, quanto mais esperarmos, mais sabemos e, quanto mais sabemos, menos probabilidade temos de cometer sérios erros em nossas estimativas.

As demais opções são abordagens viáveis às estimativas de projetos de software. Idealmente, as técnicas disponíveis para cada opção devem ser aplicadas em série, cada uma sendo usada como uma verificação cruzada das demais.

6.1.2.6 Técnicas de Decomposição

Com o objetivo de simplificar a tarefa de estabelecer estimativas de custo e esforço para um projeto de software, um problema complexo deve ser decomposto e transformado em um conjunto de problemas menores, supostamente , mais administráveis.

6.1.2.6.1 Estimativas de Linhas de Código (LOC) e Pontos por Função (FP)

As técnicas de estimativas LOC e FP diferem quanto ao nível de detalhes exigidos para a decomposição. Quando LOC é usada como variável de estimativa, a decomposição funcional é absolutamente essencial e freqüentemente assume consideráveis níveis de detalhes. Uma vez que os dados exigidos para se estimar os pontos-por-função são mais macroscópicos , o nível de decomposição usado torna-se consideravelmente menos detalhado quando FP é a variável de estimativa.

Independentemente da variável de estimativa usada , o planejado do projeto tipicamente oferece um limite de valores para cada função decomposta . Usando dados históricos ou (quando tudo mais falhar) a intuição, o planejador estima um valor LOC ou FP otimista, um mais provável e um outro pessimista para cada função. Uma indicação implícita do grau de incerteza é oferecida quando um limite de valores é especificado.

6.1.2.6.2 Estimativa do Esforço

Existem várias condutas para se estimar o esforço de desenvolvimento que é um componente importante para se estimar tempo e custos de projetos de software. Em geral, tais condutas fazem uso de estimativas baseada em LOC ou PF, explorado no item anterior.

A figura que segue ilustra como as estimativas de tempo obtidas por diferentes condutas e também de forma intuitiva podem variar em termos de precisão.

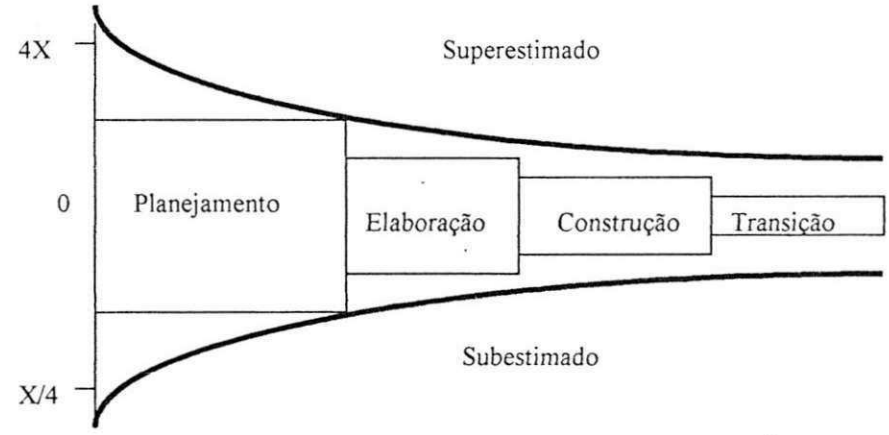


Figura 6.1:estimativas de tempo obtida por diferentes condutas

Em geral, quanto mais se estiver nas fases iniciais do desenvolvimento (Planejamento e Elaboração), mais tende-se a estimar de forma pouco precisa. Acompanhe na tabela que segue de que forma as estimativas tendem a se tornar mais precisas e maduras.

Planejamento	**Elaboração**	**Construção**	**Transição**
Fase ainda inicial	Fase Inicial de projeto	Fase pós-arquitetura	
Entradas grosseiras Estimativas pouco fiéis Requisitos em altissimo nível Arquitetura ainda conceitual	Projeto melhor entendido Estimativas mais fiéis Requisitos melhor entendidos Arquitetura melhor entendida	Projeto bem caracterizado e detalhado Estimativas alta precisão Requisitos mais estáveis Arquitetura mais estável	

Para estimarmos o tempo de desenvolvimento, com o objetivo de seguir com o cálculo de custo associados, podemos fazer uso do Modelo COCOMO na sua forma básica, por exemplo:

No modelo COCOMO, o esforço nas fases iniciais é calculado conforme ilustra as equações abaixo:

Esforço = a_b * (Tamanho) * exp(b_b),

Tempo_Desenvolvimento = c_b * (Esforço * exp(d_b)) onde;

Esforço = número de homens/hora ou homens/mês

Tempo_Desenvolvimento = tempo em meses cronológicos

Tamanho = número de pontos-por-função ou KLOC estimados

a_b e b_b são coeficientes c_b e d_b são expoentes de ajustes fornecidos na tabela abaixo.

Projeto de Software	a_b	b_b	c_b	d_b
Orgânico - projetos simples, relativamente pequenos, requisitos não muito rígidos, com pequenas equipes experientes	2,4	1,05	2,5	0,38
Semidestacado - projeto intermediário (em tamanho e complexidade), equipes com experiência heterogênea, requisitos mistos (rígidos e não rígidos)	3	1,12	2,5	0,35
Embutido - um projeto com um conjunto rígido de restrições operacionais (hardware e software)	3,6	1,2	2,5	0,32

Tabela 6.1: Tipos de COCOMO

Observe que neste modelo básico do COCOMO, o esforço e por consequência o custo do projeto é estimado em função do tamanho do código estimado - considera-se um modelo estático. Outras formas do COCOMO levam em consideração avaliações subjetivas do produto, do hardware, do pessoal e dos atributos do projeto, além do impacto de fases que direcionam os custo (análise, projeto, por exemplo). Quem tiver interesse nas outras formas mais precisas, ver em Engenharia de Software do Roger Pressman.

Uma vez estimando-se o esforço do projeto, que pode ser melhor refinado a cada iteração, estaremos em condições de fazer uma estimativa de custo para o projeto, onde os componentes abaixo são determinantes:

- Esforço em Homem/mês * Custo (incluindo impostos);
- Infra-estrutura (hardware + software + ambiente)[1];
- Overhead administrativo - cerca de 20 a 30% do valor total.

[1] Lembrar que os valores devem ser associados ao tempo do projeto. Pro exemplo, três meses de aluguel de um PC, sala mais percentual de licenças de software.

6.1.2.7 Modelos Empíricos de Estimativas

Um modelo de estimativas para software usa fórmulas empiricamente derivadas para prognosticar informações de planejamento do projeto. Os dados empíricos que sustentam a maioria dos modelos derivam de uma amostra limitada de projetos. Por essa razão, nenhum modelo de estimativa é apropriado para para todas as classes de software e em todos os ambientes de desenvolvimento.

6.1.2.7.1 Método Wideband-Delphi [Boehm]

Este método sugere que vários desenvolvedores envolvidos no projeto indiquem suas estimativas individualmente. Em seguida, aplica-se o processo Delphi para se obter uma estimativa convergente.

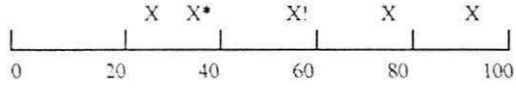
Exemplo:

Projeto: ABC

Moderador: Fulano de Tal

Data: 09.02.2001

Aqui está a faixa de estimativas obtidas na primeira coleta:



X - estimativas obtidas

X* - sua estimativa

X! - estimativa média

Favor, entre com a sua estimativa para a próxima coleta: _____ LOC.

Acrescente alguma explicação que justifique sua estimativa.

O processo Delphi tem como meta compartilhar diferentes visões dos desenvolvedores. Após algumas coletas, quando se alcança uma faixa considerada razoável, uma estimativa é colocada como finalista. Em projetos grandes, sugere-se a aplicação do processo de forma simultânea aos diferentes componentes do produto, ao final, combina-se as estimativas e elege-se uma estimativa final.

6.1.2.7.2 Método Fuzzy-Logic [Putnam]

Putnam descreve este método da seguinte forma: "os projetistas / desenvolvedores devem avaliar o produto especificado e indicar um palpite sobre seu tamanho comparando com dados históricos existentes, coletados em produtos anteriores."

Quando o grupo alcança uma massa de dados suficiente, categorias de tamanho são atribuídas, como ilustra a tabela que segue.

Faixas	Tamanho - LOC	Mínimo - LOC	Máximo - LOC
Muito Pequeno	2000	1000	4000
Pequeno	8000	4000	16000
Médio	32000	16000	64000
Grande	128000	64000	256000
Muito Grande	512000	256000	1028000

As categorias poderão ser quebradas em mais detalhes, permitindo um acerto maior nas estimativas.

6.1.2.7.3 Método Componente Padrão [Putnam]

Da mesma forma que o método Fuzzy-Logic, Putnam propõe neste método o armazenamento de dados históricos ao longo do tempo. Porém, neste método os projetistas são estimulados a registrarem dados sobre tamanhos de componentes em diferentes níveis (subsistemas, módulos, e telas, por exemplo).

Também estimula-se que sejam realizadas avaliações sobre números mínimos e máximos.

A estimativa obtida é multiplicada por quatro e, então, adicionada dos valores mínimo e máximo. Em seguida, divide-se o total por seis para se obter o valor mais indicado para o componente. O desvio padrão do valor final deve ser aproximadamente um sexto da diferença entre o mínimo e o máximo valor obtido.

Número Estimado = [menor número admitido + 4 * (número médio estimado) + maior número admitido] / 6

A tabela que segue ilustra alguns dos valores que Putnam usa para cálculo das estimativas.

Componente Padrão	**LOC por Componente**	**P**	**M**	**G**	**X= (P + 4 * M + G) / 6**	**LOC**
LOC	1					
Arquivos	2535	3	6	10	6,17	15633
Módulos	932	11	18	22	17,5	16310
Subsistemas	8175					
Telas	818	5	9	21	10,3	8453
Relatórios	967	2	6	11	6,17	5963
Programas Interativos	1769					
Programas Batch	3214					
Total						46359

6.1.2.8 Métricas Orientadas a Objeto

As métricas orientadas a objeto estão classificadas em quatro importantes categorias:

- **Tamanho do sistema**: conhecendo, por exemplo, quantas funções chamadas e quantos objetos serão usados antecipadamente no sistema pode ajudar a fazer estimativas mais precisas;
- **Tamanho da classe ou dos seus métodos**: apesar de existir várias maneiras de se medir, classe e métodos simples e pequenos são normalmente melhores de medir que classes e métodos complexos e grandes;

- **Interdependência (Coupling) e herança**: o número de tipos desses relacionamentos indicam a interdependência entre as classes. É certo que relacionamentos mais simples são melhores que os mais complexos;
- **Classes e métodos internos**: Esta métrica revela quão complexo são as classes e métodos, e quão bem você os documentou em seu código fonte;

6.1.2.8.1 Tamanho do Sistema

A métrica de Tamanho do sistema se relaciona com total de esforços requerido para a construção do sistema. Na maioria dos sistemas orientados a objetos, os componentes da interface gráfica com o usuário (GUI) representa um significante percentual do trabalho de desenvolvimento do sistema. Portanto, acrescento-se mais dois tipos de métricas orientadas ao tamanho:

- **Número de classes**: a métrica **NOC** (number of classes) conta todas as classes do sistema, incluindo classes não visuais e classes da interface gráfica com o usuário, como controle de janelas;
- **Número de Janelas**: a métrica **NOW** (number of windows) conta o número de janelas visíveis no sistema. Este número simplesmente indica o tamanho da interface gráfica do usuário.

6.1.2.8.2 Tamanho da Classe e dos seus Métodos

Esta métrica se baseia no número de linhas de código, no número de chamadas de função, no número de métodos públicos e privados e no número de atributos por classe do sistema;

6.1.2.8.3 Interdependência (Coupling) e herança

Interdependência (Coupling) e herança ajuda a medir a qualidade de um modelo de objetos. Mais especificamente, ele ajuda a revelar o grau de interdependência entre objetos existentes. Idealmente, objetos devem ser independentes, que os faz mais fáceis de

transplantá-los de um ambiente para um outro e reusá-lo na construção de novos sistemas. Esta métrica leva alguns parâmetros em consideração :

- **Class fan-in**: número de classes que dependem de um dado objeto;
- **Class fan-out**: número de classes da qual um dada classe depende;
- **Nível de herança da classe**: a profundidade da herança da classe é o número de ancestrais diretos que ela possui. Uma profundidade desnecessária acrescenta complexidade ao sistema e revela falhas no uso desse mecanismo;
- **Número de classes filhas:** esta métrica medi o número de descendente diretos de uma determinada classe, que pode indicar uma desnecessária complexidade de hierarquia;

6.1.2.8.4 Classes e Métodos Internos

Métricas internas possibilitam várias medidas de qualidade para classes e métodos. Elas podem ser usadas para identificar áreas onde ações corretivas podem ser úteis. Podendo ainda serem usadas para comparar a qualidade de um sistema com a de um outro. Estas métricas envolvem número de referencias globais compartilhadas por classe, complexidade dos métodos, o número de atributos públicos por classe, a coesão entre os métodos, o índice de especialização das classes, a percentagem de métodos comentados e finalmente, o número de parâmetros por método;

6.1.3 Gerenciamento de Projetos: PLANEJAMENTO

O primeiro passo do planejamento de projetos de software é a estimativa. A estimativa oferece ao planejador as informações necessárias para concluir as atividades de planejamento do projeto. Nesta seção apresentaremos as atividades de identificação dos riscos e a atividade de determinação de um cronograma do projeto.

6.1.3.1 Identificação dos Riscos

Quando um risco é considerado no contexto da engenharia de software, os três pilares conceituais de Charrette estão sempre em evidencia . O futuro é nossa preocupação – quais riscos poderiam fazer com que o projeto de software saísse torto ? A mudança é nossa preocupação – com as mudanças nos requisitos do cliente, nas tecnologias de desenvolvimento, nos computadores de destino e em todas as demais entidades ligadas ao projeto afetarão o sucesso global e o cumprimento do cronegrama ? Por fim, devemos nos envolver com as escolhas – quais métodos e ferramentas devemos usar ? Quantas pessoas devem ser envolvidas ? Quanta ênfase sobre a qualidade é suficiente ?

A identificação dos riscos envolve a relação dos riscos específicos de projeto dentro das amplas categorias: riscos de projetos, riscos técnicos, riscos do negócio.

- **Riscos do projeto**: identificam problemas orçamentários, de cronegrama, de pessoal, de recursos, de clientes, e de requisitos, e o impacto dos mesmos sobre o projeto de software;
- **Riscos técnicos**: identificam potenciais problemas de projeto, implementação, interface, verificação e manutenção. Os riscos técnicos ocorrem porque os problemas eram mais difíceis de resolver do que havíamos imaginado;
- **Risco do negócio**: estes são os mais insidiosos pois podem destruir os resultados até mesmo dos melhores projetos de software. Entre os candidatos aos cinco riscos de negócio de maior destaque encontram-se: (1) construir um excelente produto que ninguém realmente quer (risco de mercado); (2) construir um produto que não se

encaixa na estratégia global da empresa; (3) construir um produto que a equipe de vendas não sabe como vender; (4) riscos administrativos; (5) risco orçamentário.

Um dos melhores métodos para entender cada um dos riscos é utilizar umconjunto de perguntas que ajude o planejador do projeto a compreender os riscos em termos técnicos ou de projeto. Boehm[BOE89] sugere o uso de um "*checklist*" de itens de risco", por exemplo:

- São as melhores pessoas disponíveis ?
- As pessoas tem a combinação certa de habilidades ?
- Há pessoas suficientes à disposição ?
- O pessoal está comprometido com toda a duração do projeto ?
- Algum membro do pessoal do projeto estará trabalhando somente em tempo parcial nesse projeto ?
- O pessoal tem as expectativas corretas sobre o trabalho que tem à mão ?
- Os membros do pessoal receberam o treinamento necessário ?
- A rotatividade entre os membros do pessoal será baixa o suficiente para permitir continuidade ?

A certeza relativa das respostas a essas perguntas permitirá que o planejador estime o impacto dos riscos na composição da equipe.

6.1.3.2 Determinação de um Cronograma para o Projeto de Software

Quando abordamos a fixação de prazos para projetos de software, uma série de perguntas podem ser feitas:

- Como relacionamos tempo cronológico com o esforço humano ?
- Quais tarefas e paralelismos devem ser esperados ?
- Quais marcos de referencia podem ser usados para mostrar o progresso ?
- Como o esforço é distribuído ao longo do processo de engenharia de software ?
- Existem métodos disponíveis de determinação de prazos ?

- Como representaremos fisicamente o cronograma e como rastrearemos o progresso quando o projeto se iniciar ?

6.1.3.2.1 Relações Pessoas-Trabalho

Há um mito comum no qual muitos gerentes que são responsáveis pelo esforço de desenvolvimento de software ainda acreditam: "... se nos atrasarmos, sempre poderemos acrescentar mais programadores e posteriormente sairmos do atraso no projeto...". Infelizmente, acrescentar pessoas tardiamente num projeto muitas vezes tem um efeito desintegrador sobre o projeto, fazendo com que o cronograma fuja ainda mais do controle.

6.1.3.2.2 Definição de Tarefas e Paralelismo

Quando mais de uma pessoa está envolvida num projeto de engenharia de software, é provável que as atividades de desenvolvimento sejam executadas em paralelo. Esta natureza concorrente dessas atividades leva a uma série de importantes exigências de cronograma. Uma vez que as tarefas paralelas ocorrem de maneira assíncrona, o planejador deve determinar as dependências intertarefas para garantir o contínuo progresso rumo à conclusão. Além disso, o gerente de projetos deve ter consciência daquelas tarefas que se encontram no caminho crítico, as tarefas que devem ser concluídas no prazo, se é que o projeto como um todo pretende ser concluído dentro do cronograma.

6.1.3.2.3 Distribuição do Esforço

As características de cada projeto devem ditar a distribuição do esforço para cada projeto, de forma que nenhuma regra geral deve ser seguida rigorosamente sob pena de incorrer em erros de estimativas.

6.1.3.2.4 Métodos de Determinação de Cronogramas

A determinação de um cronograma para um projeto de software não difere significativamente da fixação de prazos para qualquer esforço de desenvolvimento de multitarefas. Portanto, ferramentas e técnicas para determinação de um cronograma de projeto podem ser aplicadas no software com poucas modificações.

O *PERT – Program Evalution and Review Tecnique* e o *CPM- Critical Path Method* , são dois métodos de determinação de cronogramas que podem ser aplicados no desenvolvimento de software. Ambas as técnicas desenvolvem uma descrição da rede de tarefas de um projeto, ou seja, uma representação pictórica ou tabular das tarefas que devem ser levadas a efeito o início até o final de um projeto. A rede é definida ao se desenvolver uma lista de todas as tarefas associadas a um projeto específico, a qual às vezes é chamada *WBS - Work Breakdown Structure,* e uma lista de restrições que indica em que ordem as tarefas devem ser executadas.

Tanto o PERT quanto o CPM proporcionam ferramentas quantitativas que permitem ao planejador de software:

1. determinar o caminho crítico- cadeia de tarefas que determina a duração do projeto;
2. estabelecer as estimativas de tempo mais prováveis para tarefas individuais ao aplicar modelos estatísticos;
3. calcular limites de tempo que definam um "tempo de reserva" para uma tarefa em particular;

Cálculos de limites de tempo podem ser úteis na determinação de um cronograma para projetos de software. O "resvalamento" no prazo de um projeto de uma função por exemplo, pode retardar ainda mais o prazo de desenvolvimento de outras funções. Riggs [RIG81] descreve importantes limites de tempo que podem ser reconhecidos numa rede PERT ou CPM:

1. o limite mais cedo em que uma tarefa pode iniciar-se é quando todas as tarefas precedentes foram completadas no tempo mais curto possível;

2. o limite mais tarde para o início de uma tarefa é antes que o tempo de conclusão mínimo do projeto seja atrasado;
3. o término mais breve- a soma do início com a duração da tarefa;
4. o término mais tardio – o momento de início mais tarde somado à duração da tarefa;
5. a flutuação total- a quantidade de superávit de tempo ou de margem de segurança permitida nas tarefas de determinação de prazos de forma que o caminho crítico da rede seja mantido dentro do prazo.

Os cálculos do tempo-limite leva, à determinação do caminho crítico e proporcionam ao gerente, um método quantitativo para avaliar o progresso à medida que as tarefas são concluídas.

O planejador deve reconhecer que o esforço despendido no software não se encerra no final do desenvolvimento. O esforço de manutenção, ainda que não seja fácil programar neste estágio, tornar-se-á, por fim, o maior fator de custo. Uma meta primordial da engenharia de software é ajudar a reduzir esse custo.

6.1.3.2.5 Rastreamento e Controle de Projeto

Há um antigo ditado que diz: "Os projetos de software atrasam-se em seu cronograma um dia de cada vez". O atraso de um dia na programação raramente será fatal para o projeto. Mas os dias se somam e ao longo da extensão de um projeto, pequenos atrasos podem resultar em grandes problemas.

Já observamos que um dos papeis da administração é rastrear e controlar o projeto, assim que ele se põe a caminho. O rastreamento (*tracking*) pode ser levado a efeito de muitas maneiras diferentes:

- Realizando-se reuniões periódicas sobre a situação do projeto, em que cada membro da equipe relate o progresso e os problemas.
- Avaliando-se os resultados de todas as revisões levadas a efeito ao longo do processo de engenharia de software.

- Determinando-se se os marcos de referencia do projeto foram atingidos até a data programada.
- Comparando-se a data de início real com a data de início planejada para cada tarefa de projeto relacionada previamente.
- Reunindo-se informalmente com profissionais para se obter suas avaliações subjetivas do progresso até o momento e os problemas que aparecem no horizonte.

O controle é empregado por um gerente de projetos de software para administrar os recursos do projeto, enfrentar problemas e dirigir o pessoal de projeto. Se as coisas estiverem indo bem, o controle é leve. Mas quando ocorrerem problemas, o gerente de projetos deverá exercer o controle para saná-los o mais rapidamente possível. Depois que o problema tiver sido diagnosticado, recursos adicionais podem ser concentrados na área do problema; o pessoal pode ser novamente disposto ou a programação do projeto, redefinida.

6.1.4 O Processo de Desenvolvimento de Software

Um processo de desenvolvimento de software é um método para organizar as atividades relacionadas ao Planejamento, Elaboração, Criação, Entrega e Manutenção de sistemas de software de maneira que fique claro **quem** faz **o quê**, **quando**, **como** e até mesmo **onde**, para atingir um certo alvo. Em Engenharia de Software, um processo tem como meta a construção de artefatos de software, ou a melhoria de um produto existente, através da participação de vários usuários (clientes, usuário final, desenvolvedor, gerentes etc.), cujos limites dependem de alguns elementos tais como: tecnologia, ferramentas, habilidades pessoais dos envolvidos e organização.

Desenvolver sistemas de software constitui hoje um imenso desafio às empresas. São duas as classes de problema que tornam esta tarefa dificultosa:

- as **pressões do negócio**, que ocorrem devido à necessidade das empresas de implementarem novos sistemas muito rapidamente, a fim de utilizarem a tecnologia como diferencial competitivo;
- as **pressões tecnológicas**, que forçam mudanças nos sistemas e tomam seu desenvolvimento muito mais complexo.

Devido a tais problemas, o que se espera de um processo de desenvolvimento atual é que ele possua características tais como melhor flexibilidade e adaptabilidade às necessidades de mudança dos requisitos do negócio; proporcione um menor tempo de desenvolvimento, com baixo orçamento e melhor integração entre as diversas fases de desenvolvimento de software. A chave para a solução desses problemas é o Desenvolvimento Iterativo e Incremental de software.

6.1.4.1 Desenvolvimento Iterativo e Incremental de Software

Hoje, é considerado errado ter um processo que gere um "grande bum!". Ou seja, não se deve ter o software inteiro funcionando por completo no primeiro release, pois o risco é muito grande. Assim, um processo de desenvolvimento deve ser:

- Iterativo: deve ter várias iterações no tempo;

- Incremental: deve gerar novas versões incrementadas a cada release;

onde uma iteração dura entre 2 semanas e 2 meses pois:

- Sempre tem algo para entregar para o cliente apressado (a última iteração)
- Os requisitos mudam com tempo e um processo iterativo mantém freqüentes contatos com o cliente o que ajuda a manter os requisitos sincronizados
- Altamente motivador para a equipe de desenvolvimento (e o cliente) ver o software funcionando cedo

Para lidar com ciclos de desenvolvimento cada vez mais complexos e longos, são utilizadas iterações para "quebrar" esta complexidade e obter uma resposta mais rapidamente dos resultados de sua aplicação.

Sob esta ótica, um sistema é desenvolvido através de múltiplos ciclos de desenvolvimento de análise, projeto, implementação e teste. Onde cada ciclo adiciona uma nova funcionalidade, ou melhora uma existente, resultando em um produto parcial funcionando completamente.

A seguir abordaremos dois processo de desenvolvimento de software baseados no modelo iterativo e incremental: O Processo Unificado e a Programação Extrema.

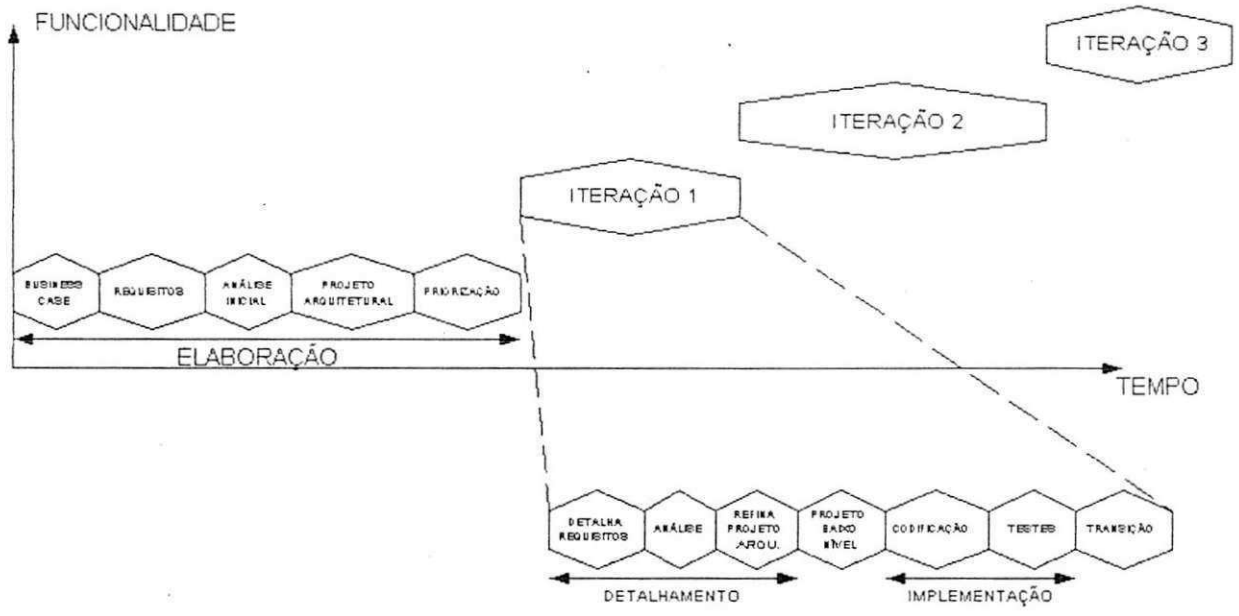
6.1.4.2 O Processo Unificado de Desenvolvimento de Software

O Processo Unificado de Desenvolvimento de Software – PU, é um processo de desenvolvimento que suporta um conjunto de atividades necessárias para se transformar "requisitos" num sistema de software de maneira iterativa e incremental. Apresenta-se como um framework genérico para uma ampla gama de classes de software e faz uso de UML-Unified Modeling Language, como sua linguagem de modelagem padrão.

6.1.4.2.1 Fases do Processo Unificado de Desenvolvimento de Software

As quatro grandes fases presentes em qualquer processo de desenvolvimento de software são as seguintes:

- **Planejamento e Elaboração**, que envolve o planejamento, definição de requisitos, construção de protótipos (opcional);
- **Construção** do sistema, que envolve a codificação e teste;
- **Implantação**, que engloba as etapas de publicação instalação, treinamento de usuários etc.



6.1.4.2.1.1 Planejamento e Elaboração

A fase de Planejamento e Elaboração de um projeto de software inclui as seguintes etapas:

- **Criação de um relatório inicial de investigação**, do qual será construído um documento contendo informações estratégicas e financeiras sobre o projeto;

- **Levantamento de requisitos funcionais e não funcionais**

  Os requisitos são partes da solução do problema ou o entendimento do que o usuário quer que seja resolvido. O resultado desta fase é uma promessa para o cliente. Nesta fase não só requisitos são levantados requisitos funcionais, mas também os não funcionais:

  - Facilidade de uso necessária ;
  - Quem utilizará o produto;
  - Hardware e software alvo para o produto;
  - Qualidade/robustez;
  - Desempenho;
  - Segurança;
  - Compatibilidade com outros produtos/versões e necessidades de migração;
  - Necessidades de internacionalização do produto;
  - Suporte;
  - Preço da solução;
  - Documentação necessária;
  - Uso de padrões;
  - Aspectos legais;
  - Integração com outros produtos;
  - Embalagem etc.

Nessa fase ainda não se fala "como" as coisas serão feitas e os diagramas de use cases descrevem os cenários sobre a funcionalidade desejada ;

- **Construção de um dicionário** de termos e qualquer informação associada, tais como regras e restrições;
- **Definição do modelo conceitual inicial;**
- **Priorização das funcionalidades e distribuição destas entre as iterações a serem desenvolvidas;**

- **Sugestão de novas funcionalidades** que possam gerar uma nova versão do produto no futuro.

6.1.4.2.1.2 Construção

A fase de construção de um projeto envolve ciclos repetidos de desenvolvimento, as iterações, dentro dos quais o sistema é estendido. O objetivo final desta fase é a obtenção de um sistema de software completo e de acordo com seus requisitos.

A cada iteração, as seguintes etapas são repetidas:

- **Análise**

A análise gera um modelo para entender o domínio do problema tratando o mesmo em forma genérica (alto nível) como uma solução possível que pode ser montada para atender aos requisitos. Muitos diagramas UML podem ser usados aqui, fato que acaba gerando uma especificação, mas sempre do ponto de vista do usuário e tratando apenas do domínio do problema e jamais dos detalhes de implementação. Este modelo gerado é sempre para o cliente, e não para programadores;

As atividades típicas realizadas durante a análise são:

- Refinar use cases
- Refinar modelo conceitual
- Refinar o dicionário de termos
- Definir diagramas de seqüência (opcional)
- Definir contratos de operação (opcional)
- Definir diagramas de estado (opcional)

- **Projeto**

A etapa de projeto gera uma extensão do modelo de análise, visando sua implementação num computador. É dividida em duas partes: Projeto Arquitetural (ou projeto de alto nível) e projeto detalhado (ou projeto de baixo nível).

- Projeto de Alto Nível

- Definição de pacotes (módulos), interfaces entre pacotes;
- Decisão sobre uso/criação de bibliotecas e/ou componentes;

- **Projeto de Baixo Nível**

  - Atribuição de responsabilidades entre os objetos
  - Construção de diagramas de classes
  - Construção de diagramas de interação (opcional)
  - Levantamento de necessidades de concorrência
  - Considerações de tratamento de faltas
  - Detalhamento do formato de saída (interface com usuário, relatórios, transações enviadas para outros sistemas, ...)
  - Definição do esquema do BD
  - Mapeamento de objetos para tabelas se o BD for relacional

- **Implementação**

  Esta etapa envolve as atividades de codificação e várias fases de testes . A codificação é feita a partir de artefatos produzidos na fase de projeto. Os testes possuem a seguinte classificação:

  - **Testes de Unidades**: teste de classes individuais (ou de grupos de classes relacionadas) ;
  - **Teste de Funcionalidades**: teste de funções inteiras (item de menu, p. ex.) ;
  - **Teste de Componentes**: teste de componentes inteiros (exe, dll, ...);
  - **Testes de Sistemas**: testa a integração entre todos os componentes do produto;
  - **Teste Alpha**: teste de produto inteiro na própria empresa;
  - **Teste Beta**: teste de produto inteiro no cliente;

  Sempre que possível, os testes deveriam ser automatizados.

- **Transição para produto**

  Esta etapa envolve atividades de documentação, empacotamento, instalação etc.

6.1.4.2.1.3 Implantação

A fase de implantação de um projeto consiste em:

- gerar documentação técnica para o suporte;
- na geração de documentação básica para o usuário;
- implementação do software de instalação
- confecção da embalagem;
- preparação do treinamento etc.

6.1.4.3 Programação Extrema

Programação Extrema é um processo simples de desenvolvimento de software que segue o modelo iterativo e incremental de desenvolvimento de software, como havíamos dito, que visa principalmente reduzir o tempo e custo de desenvolvimento de software.

Idealizada pelos fundadores de SmallTalk, Ward Cunningham, Ron Jeffries, Martin Fowler, Kent Beck, a Programação Extrema elimina todos os "tradicionais" artefatos UML, tipo Use cases, diagrama de classes etc, deixando apenas um que eles chamam de *UserStores*, onde o cliente escreve os requisitos do sistema e define as prioridades desses. Então os desenvolvedores planejam as iterações, duração máxima de uma semana, e pronto, inicia-se a fase de **codificação** seguida de **testes** e mais testes e sempre que necessário, **refatoração**!

Além disso, a Programação Extremas tem regras bem definidas para a equipe de desenvolvimento:

- A iteração mais crítica é codificada primeiro;
- Nada é feito sem que seja extremamente necessário agora;
- A equipe deve ser formada programadores bastante experientes;
- O trabalho é realizado em pares, pois melhora o código no geral, apesar de diminuir o rendimento;
- Para requisitos: UserStories e Teste Funcionais;
- Para análise e projeto: nenhum documento, só cartões CRC ;

- Para codificação: um comentário por classe, se necessário;
- Se o usuário quiser um documento, manual, etc. é só fazer um UserStory pedindo!

6.1.4.4 Controle de Mudanças no Desenvolvimento de Software

O processo de controle de mudanças é um processo definido a ser seguido quando uma mudança a um documento controlado está sendo proposto. Este processo deve ser utilizado para efetuar mudanças nos documentos de Requisitos e Especificação. Por exemplo, o documento de requisitos deve ser colocado sob tal sistema de forma a não aceitar a implementação de qualquer mudança pedida por usuários, digamos, sem análise do seu impacto. Motivos levam a colocar os outros documentos sob controle de mudanças. Como resultado, o projeto permanece sob controle e todos seus componentes sincronizados com a documentação disponível.

Ao propor uma mudança, as seguintes perguntas devem ser feitas:

- A mudança é necessária?
- Em qual *release* do produto dever-se-ia efetuar a mudança?
- Quem será afetado pelas mudanças?
- De que forma o cronograma será mudado?
- Que documentação deve ser produzida em função da mudança?
- Como e quando efetuar a mudança no sentido de minimizar seu impacto?

As mudanças são avaliadas por um grupo consistindo do gerente de produto, líder do desenvolvimento, além de alguns programadores experientes do time de desenvolvimento.

A aceitação (ou não) da mudança se faz em dois estágios. Após a submissão de uma mudança para avaliação, o grupo aprova ou não a realização de um estudo de impacto. O conteúdo deste estudo ficará claro com a apresentação da informação de status associada à

mudança, dada adiante. Após o recebimento do estudo de impacto, o grupo aprova ou não a realização da mudança.

A seguinte informação deve ser mantida para cada mudança proposta, seja ela efetuada ou não:

- Número único de controle;
- Nome da pessoa que sugere a mudança;
- Data de abertura da mudança;
- Pessoa responsável pelo estudo de impacto;
- Status;
- Data de fechamento necessária;
- Data-alvo de fechamento;
- Data real de fechamento;
- Data de aceitação para estudo de impacto;
- Data de aceitação para inclusão no produto;
- Mudança congelada até data/release;
- Mudança rejeitada;
- Mudança cancelada;
- Título da mudança;
- Descrição da mudança;
- Justificativa para a mudança;
- Estudo de impacto;
- Linhas de código afetadas (mudadas e novas);
- Componentes ou módulos afetados;
- Tempo estimado para design e data de provável fechamento;
- Tempo estimado para codificação e data de provável fechamento;
- Tempo estimado para teste de unidade e data de provável fechamento;
- Tempo estimado para teste de função e data de provável fechamento;
- Tempo estimado para componente e data de provável fechamento;

- Tempo estimado para documentação e data de provável fechamento;
- Novas especificações prontas (e anexadas);
- Observações.

6.1.4.5 Controle de Versões do Software

O controle de versões de software é de extrema importância no processo de desenvolvimento de software de qualidade. É através dele que podemos, por exemplo, corrigir defeitos em versões que estão liberadas à medida que o desenvolvimento das novas versões prossegue.

Costuma-se definir como versão ou configuração de um sistema de software, um pacote composto de todos os artefatos gerados para funcionar numa dada plataforma. Outra nomenclatura muito usada é a de "release", indicando revisões que são desenvolvidas e liberadas para uso interno ou externo. Neste contexto, é muito comum encontrar empresas rotulando seus produtos da seguinte forma:

- Produto Versão n.m, onde

n - indica a versão corrente (definindo uma configuração de hardware específica)

m - indica a release corrente.

Uma mudança de plataforma operacional, por exemplo, o porte do produto para Linux, representaria uma nova versão (n+1). Uma melhoria nas funcionalidades inciais, por outro lado, representaria uma nova release (m+1).

6.1.4.5.1 Controle de Versões e o Processo Iterativo de Desenvolvimento

Quando se fala em processo de desenvolvimento incremental e iterativo, popularizado mais recentemente, o conceito inicial de versão permanece o mesmo, porém, algumas diferenças são trazidas aqui.

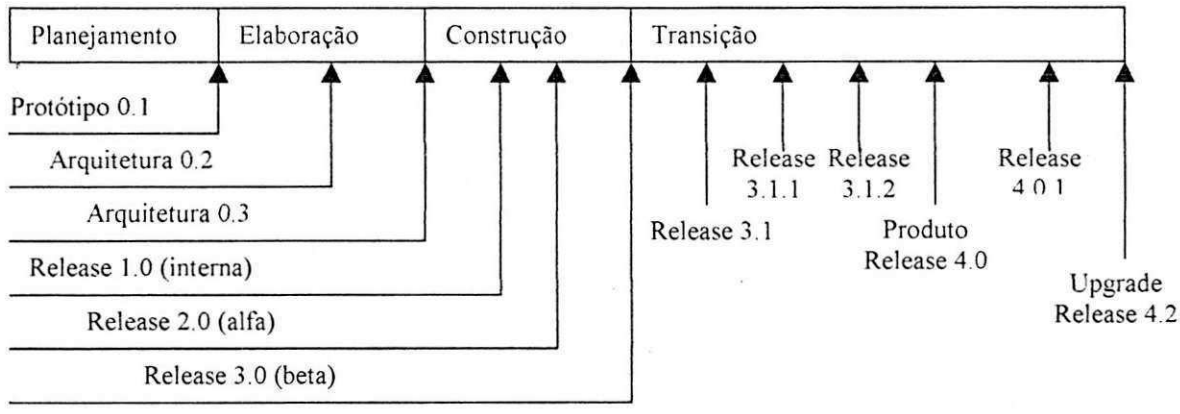
Primeiro, no desenvolvimento incremental e iterativo existem dois tipos básicos de saídas ou entregas:

1. Releases internas
2. Releases externas

Em geral, recomenda-se que as mudanças sejam controladas através de três dimensões. Tais dimensões servem para identificar e diferenciar releases principais, secundárias ou temporárias, rotuladas na forma p.s.t, onde:

- p - número que indica uma release principal (representa uma nova geração do produto);
- s - número que indica uma release secundária (representa melhorias na principal)
- t - número que indica uma release temporária (representa uma versão temporária e ainda interna)

A figura abaixo ilustra um histórico típico de controle de releases desenvolvidas com base no processo incremental e iterativo.



É também de extrema importância a forma como o desenvolvedor classifica as mudanças. Existem cinco categorias básicas:

Tipo 0 → corresponde a falhas críticas encontradas, que devem ser corrigidas antes que qualquer release seja liberada para uso externo (em geral, associada a Use Cases críticos);

Tipo 1 → um defeito ou bug que não impede o uso do sistema. Em geral, são ruídos em Use Cases críticos ou falhas mais sérias em Use Case secundários, com baixa probabilidade de ocorrência;

Tipo 2 → uma mudança que representa uma melhoria no sistema e não apenas uma reação a algum defeito ou bug. Em geral, são responsáveis por melhorias em desempenho, usabilidade, testabilidade ou outro aspecto de qualidade que agregue valor ao produto;

Tipo 3 → uma mudança que é necessária devido ao surgimento de novos requisitos. Pode representar uma nova funcionalidade ou a introdução de aspectos que estejam fora do contexto da visão atual presente no business case.

Tipo 4 → mudanças que não se enquadrem nas outras categorias. Por exemplo, documentação adicional, upgrades de componentes de um terceiro, etc.

Existem alguns formatos padrões que podem ser consultados:

- IEEE - Std 828 (1990)
- MIL-STD-973 (1992)
- DOD-STD-2167A

A documentação recomendada para descrever uma dada release contempla os seguintes aspectos:

1. Contexto
   - Conteúdo básico
   - Métricas
2. Notas
   - Limitações

3. Resultados
   - Formas para acompanhamento de sua evolução
   - Recomendações para a próxima

4. Outros dados
   - Post-mortem sumarizado

Finalmente, é necessário que seja disponibilizado um ambiente para gestão das releases produzidas (internas ou externas).

6.1.4.5.2 Ferramentas de Controle de Versões

A ferramenta de controle de versões deve permitir entre outras coisas:

- Manter históricos de versões de fontes individuais ou de conjuntos de fontes como projetos de software.
- Manter controle de versões de um determinado conjunto de projetos independentes.
- Permitir a automação nas tarefas de liberação de novos releases.
- Permitir manter controle sobre arquivos binários, tais como: bitmaps, dlls, executáveis, entre outros.

A atividade seguinte a Revisão Bibliográfica foi a realização de um levantamento do nível atual de maturidade da empresa no desenvolvimento de produtos de software, envolvendo as fases de planejamento, análise, projeto, implementação, teste e entrega do produto.

6.2 *Estágio Atual da Empresa no Desenvolvimento de Software*

Neste levantamento, foram analisados os itens relativos às fases de planejamento, análise, projeto, implementação, teste e entrega do produto.

6.2.1 Planejamento

Nenhum padrão era seguido. As decisões eram tomadas de forma bastante subjetiva, pois não havia quase nenhuma documentação de projetos passados no qual a mesma pudesse se espelhar.

A estimativa dos custos do projeto também era bastante empírica e no máximo se baseava numa pesquisa junto aos concorrentes.

A estimativa de tempo de duração ou de entrega do produto ao cliente era baseada em projetos passados, mas devido a pouquíssima documentação, só se tinha uma idéia aproximada de quanto tempo teria durando tal projeto. Quando o projeto era totalmente novo e fora do domínio do conhecimento da equipe, este projeto tinha o tempo de duração alongado o máximo possível e tolerado pelo cliente.

O planejamento de recurso necessário não era realizado de forma adequada. Eram sempre dois coordenadores e três programadores: um dos coordenadores era responsável pelo contato com o cliente e o outro pelo projeto do banco de dados. Um dos programadores era responsável pelo projeto das interfaces com o usuário e codificação, um outro era responsável somente pela codificação e por último, um programador era responsável pelos relatórios do sistema. Estas funcionalidades eram exercidas independentemente da quantidade de projetos em andamento. Um membro da equipe poderia trabalhar em vários projetos ao mesmo tempo e exercer funções diferentes em cada um deles sem que existisse nenhum plano de atividade que o orientasse nesta transição.

Não era realizada nenhuma análise dos riscos que poderiam posteriormente afetar de qualquer forma a seqüência do projeto.

6.2.2 Análise

Existe uma preocupação muito grande com relação a análise do sistema. Há consciência de que sem uma análise não se pode concluir um projeto. Assim, o cliente é entrevistado inúmeras vezes até que todas as dúvidas com relação ao escopo do problema estejam sanadas. No entanto, todo o escopo do projeto sempre fica na cabeça do coordenador e na maioria das vezes apenas dois artefatos são gerados: um Documento de Requisitos Funcionais – DRF e um Diagrama de Entidades e Relacionamentos – DER, com um agravante, não existia nenhum mecanismo de controle de atualizações destes artefatos.

6.2.3 Projeto

A divisão quando era realizada, não o era de forma clara ou definida com antecedência. Desta forma, não era definido qualquer cronograma de acompanhamento e controle de progresso do projeto. A importância dessa parte do processo era invariavelmente negligenciada.

6.2.4 Implementação

A implementação era (re)iniciada tão logo já existisse um (novo)protótipo do banco de dados, geralmente, sem uma validação final do cliente. Um programador somente saberia o que fazer no dia seguinte se a tarefa do dia atual não tivesse sido concluída. Aqui detectou-se o maior índice de retrabalho devido a existência de falha na especificação, análise, projeto e até mesmo falha na comunicação entre os membros da equipe e o coordenador do projeto.

Existe a consciência de que todos os métodos deveriam ser comentados, mas não se tinha noção do quanto seria necessário para se fazer entender ou como fazê-lo usando uma linguagem padronizada e não ambígua.

6.2.5 Testes

Não existia um planejamento dos testes o que causava um número muito alto de retornos dos clientes até que o sistema se tornasse progressivamente mais estável. No caso de uma atualização, os teste nunca eram refeitos ou sequer documentados.

Fruto de falhas durante a coleta de requisito (não-funcionais) e de um mal planejamento desses testes, algumas restrições somente eram detectadas meses após a liberação do produto para o cliente, o que agravava mais ainda a situação do projeto.

6.2.6 Entrega do Produto

Não existia nenhuma preocupação com a qualidade da embalagem ou apresentação do produto. Quando o produto estava "pronto" era instalado na máquina do cliente sem nenhum controle de versões . O procedimento era o mesmo tanto na primeira entrega do produto quanto nas entregas das atualizações seguintes.

Por causa dos problemas detectados nessa atividade, principalmente problemas nas fases de análise, projeto e implementação, foi recomendado a realização de uma Refatoração (*refactoring*) no principal produto da empresa, no caso o Panificador V 1.0. Esta Refatoração é parte do Estagio Supervisionado de Rodrigo F. de Albuquerque, de forma que todas as condutas de programação sugeridas no seu estágio serão acatadas sumariamente. A seguir, definiremos o Processo ByteCom de Desenvolvimento de Software.

6.3 *Definição do Processo ByteCom de Desenvolvimento de Software*

É chegada a hora de definirmos o PROCESSO BYTECOM DE DESENVOLVIMENTO DE SOFTWARE – **PROCESSO BYTECOM** . Ou seja, é hora de definirmos as condutas e artefatos que serão gerados em todas as fases do desenvolvimento de software abordadas na seção **6.1.4.2** desse relatório e principalmente levando em consideração as restrições de pessoal e orçamentárias da empresa.

Aliado às condutas de desenvolvimento e ao conjunto de artefatos escolhidos, será definido um conjunto de ferramentas de automação para facilitar a geração e manutenção desses artefatos.

6.3.1 Critérios adotados

Alguns critérios foram adotados na escolha tanto dos artefatos quanto das ferramentas de automação. Estes critérios estão listados nas tabelas abaixo:

Critérios para os Artefatos	Motivo
1. Os artefatos devem ser simples e de fácil entendimento e criação;	• Evitar perda de tempo com o entendimento de sua semântica e a própria criação dificultosa.
2. Os artefatos devem conter o mínimo possível de informação textual;	• Gasta-se muito tempo para se redigir um bom texto; • Gasta-se muito tempo para se manter um bom texto atualizado; • Se o texto escrito não for bom o suficiente, o mesmo pode pecar por ausência de informação, informação imprecisa ou conter ambigüidades.
3. Os artefatos devem expressar o máximo de informação por si só (alto poder de abstração);	• Gasta-se tempo para criar um bom artefato; • Diminui o número de artefatos para serem atualizados e revisado pelo gerente do projeto em questão;

	• Conseqüente economia de tempo;

Todos os artefatos gerados estarão sujeito ao modelo de controle de versão descrito na seção 6.1.4.5 desse relatório, tão logo sejam validados por quem for de direito.

Critérios para as Ferramentas de Automação	**Motivo**
1. Dar suporte a criação dos artefatos do processo de desenvolvimento de software da empresa;	• Óbvio;
2. Simplicidade, facilidade de entendimento e boa documentação;	• Evitar deslocamento de equipes para treinamento fora da empresa; • Redução de custos com treinamento de pessoal;
3. Preço de mercado;	• A ferramenta não pode ser muito cara, preferencialmente freeware, pois a empresa deve adquirir uma ou várias licenças de uso para tal ferramenta; • A empresa deve utilizar apenas ferramentas legalizadas;
4. Prioridade para ferramentas nacionais	• A empresa só comprará no exterior o que não encontrar aqui no Brasil; • Protecionismo;

A seguir, descreveremos as fases do processo, as condutas e os artefatos gerados em todas as fases do **PROCESSO BYTECOM.**

6.3.2 Planejamento

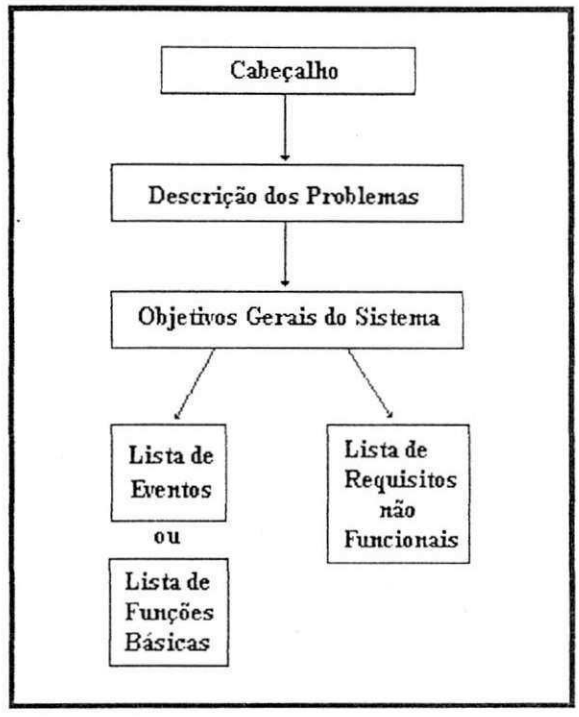
Nessa fase, serão gerados basicamente dois artefatos: o **Documento de Requisitos Básicos – DRB**, e o **Plano de Negócio Simplificado– PNS** que serão descritos a seguir.

No cálculo de estimativas do projeto de software, a unidade básica que será utilizada é o Ponto por Função – FP, discutido na seção 6.1.1.2.2. Para auxiliar no cálculo do número do FP do sistema foi implementado um aplicativo simples que será descrito no Anexo - A.

6.3.2.1 O Documento de Requisitos Básicos - DRB

- **Descrição :** É um documento que deve conter basicamente a descrição do problema a ser resolvido, o objetivo geral do sistema, uma lista de requisitos funcionais e uma outra lista de requisitos não funcionais.
- **Objetivo :** Determinar o escopo do sistema a ser desenvolvido, suas restrições, o risco sobre o mesmo;
- **Quem cria :** o analista juntamente com o cliente;
- **Quem valida :** o gerente do produto;
- **Quando iniciar :** logo no primeiro contato com o cliente ou quando houver uma percepção dos problemas de mercado que podem ser resolvidos de forma lucrativa pela empresa;

Observe o esquema do documento abaixo:



O item **Cabeçalho** é padrão para todos os outros artefatos e deve conter as seguintes informações de controle:

DOCUMENTO DE REQUISITOS BÁSICOS - DRB		Nº DOCUMENTO:
DATA CRIAÇAO: / /	ULTIMA ALTERAÇAO: / /	VERSAO:
AUTOR:	GERENTE:	
CÓDIGO DO PROJETO:		

O item **Descrição dos Problemas** é um campo específico e deve relatar a problemática encontrada no cliente.

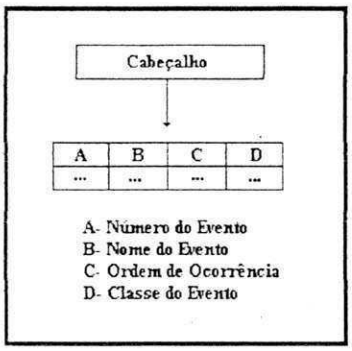
O item **Objetivos Gerais do Sistema** é um campo específico e deve conter resumidamente, os objetivos gerais do sistema em questão.

Os itens **Lista de Eventos** ou **Lista de Funcionalidades Básicas** e **Lista de Requisitos Não Funcionais** são artefatos simples que terão sua estrutura detalhada mais adiante.

➢ Lista de Eventos – LE :

- **Descrição :** Este artefato deve conter uma lista de eventos que descrevem as funcionalidades básicas do sistema classificados pela ordem de ocorrência e classe do evento;
- **Objetivo :** integrar o DRB;
- **Quem cria :** o analista juntamente com o cliente;
- **Quem valida :** o gerente do produto;
- **Quando iniciar :** idem DRB;

Observe o esquema do artefato abaixo:



O campo "Ordem de Ocorrência" do esquema acima representa a ordem de ocorrência do evento no tempo;

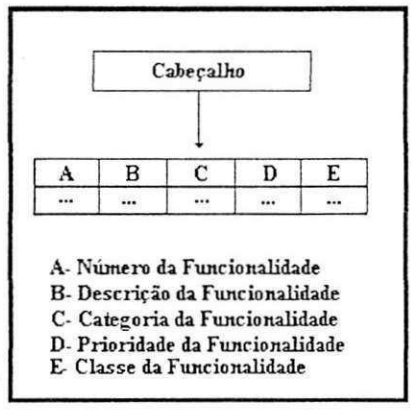
O campo "Classe do Evento" determina se o evento é **crítico** ou **não crítico;**

➢ Lista de Funcionalidades Básicas – LFB

- **Descrição :** Este artefato deve conter uma lista de funcionalidades que descrevem as funcionalidades básicas do sistema classificados pela prioridade, pela categoria e pela classe da funcionalidade;
- **Objetivo :** integrar o DRB;
- **Quem cria :** o analista juntamente com o cliente;

- **Quem valida :** o gerente do produto;
- **Quando iniciar :** idem DRB;

Observe o esquema do artefato abaixo:



O campo "Categoria da Funcionalidade" representa duas: Evidente e Escondida;

O campo "Classe da Funcionalidade" representa duas classes de funcionalidades: Critica e Não-crítica;

Deve-se escolher um entre este dois artefatos. O critério é justamente a complexidade do sistema. Sistemas mais simples, recomenda-se utilizar a LE, enquanto sistemas mais complexos, a LFB.

- Lista de Requisitos Não Funcionais – LRNF:
  - **Descrição :** Este artefato deve conter uma lista de requisitos não funcionais que descrevem requisitos de qualidade, segurança, desempenho, suporte, documentação necessária, facilidade de uso etc;
  - **Objetivo :** integrar o DRB;
  - **Quem cria :** o analista juntamente com o cliente;
  - **Quem valida :** o gerente do produto;
  - **Quando iniciar :** idem DRF;

Este artefato contém um cabeçalho padrão e um campo texto onde serão descritos os requisitos não funcionais do sistema.

6.3.2.2 O Plano de Negócio Simplificado - PNS

- **Descrição :** É um documento que deve conter informações estratégicas como: informações mercadológicas (concorrência, clientes potenciais, parcerias etc.), técnicas (estimativas, cronograma geral etc.) e financeiras (custo de desenvolvimento etc.);
- **Objetivo :** Determinar a viabilidade do projeto;
- **Quem cria :** o gerente do produto, gerente financeiro e o gerente de marketing;
- **Quem valida :** a diretoria;
- **Quando iniciar :** logo que a primeira versão do DRB estiver pronta e validada;

6.3.3 Elaboração

A principal atividade aqui é a priorização das funcionalidades e distribuição dessas entre as iterações a serem desenvolvidas. Esta priorização das funcionalidades é realizada em conjunto com o cliente e validada pelo gerente de produtos.

As iterações devem ter uma duração mínima de duas semanas e máxima de quatro. No entanto, quem determina a complexidade e a duração de cada iteração é a equipe de desenvolvimento, incluindo o gerente de produtos.

Nessa fase, poderão ser gerados novos artefatos, ou apenas um detalhamento de um artefato gerado na fase de planejamento do sistema. Dentre esses artefatos poderemos citar o Modelo Conceitual (primeira versão), o Dicionário de Termos (primeira versão), o Projeto Arquitetural do Sistema (primeira versão)e o Diagrama de Use Cases(versão expandida);

componentes do sistema. Preferencialmente, descrito de maneira não textual, ou com o mínimo possível de texto;

- **Objetivo:** Facilitar o entendimento do problema e a comunicação entre os envolvidos;
- **Quem cria :** o analista;
- **Quem valida :** o gerente do produto;
- **Quando iniciar :** logo que a primeira versão do DRB estiver pronta e validada;

6.3.3.4 Diagrama de Use Cases – DUC

- **Descrição :** é um artefato simples e opcional para sistemas mais simples, e obrigatório para os mais complexos, contudo, este deve citar os atores e as ações de cada um deles no sistema;
- **Objetivo:** Detalhar os requisitos do sistema e facilitar a comunicação entre os envolvidos no processo, principalmente o cliente. Ajudar na geração da documentação do usuário, como ajudas on line e manuais;
- **Quem cria :** o analista e o cliente;
- **Quem valida :** o gerente de produto;
- **Quando iniciar :** logo que a primeira versão do DRB estiver pronta e validada;

6.3.4 Construção

Esta fase é marcada pelos ciclos de desenvolvimento, iterações. Assim, a cada iteração serão realizadas as seguintes atividades :

6.3.4.1 Refinamentos de Artefatos

- Refinamento dos requisitos funcionais (DUC, LE, LFB);
- Refinamento do Modelo Conceitual (DMC ou DER);
- Refinamento do Projeto Arquitetural do Sistema;
- Refinamento do Dicionário de Termos;

6.3.4.2 Projeto de Baixo Nível

Alguns novos artefatos poderão ser incluídos aqui. São estes: Diagrama de Fluxos de Dados, Mini-especificações e o Diagrama de Entidades e Relacionamentos e Diagramas de Classes, Diagrama de Sequencia (opcional), Digrama de Colaboração (opcional);

6.3.4.2.1 Diagrama de Fluxos de Dados – DFD

- **Descrição :** é um artefato simples e obrigatório para sistemas mais simples que relaciona agentes externos, processos, fluxos e depósitos de dados. Não é recomendado para os sistemas mais complexos;
- **Objetivo :** Detalhar os requisitos do sistema e facilitar a comunicação entre os envolvidos no processo, principalmente o cliente. Ajudar na geração da documentação do usuário, como ajudas on line e manuais;
- **Quem cria :** o analista e o cliente;
- **Quem valida :** o gerente de produto;
- **Quando iniciar :** logo que a primeira versão do DRB estiver pronta e validada;

6.3.4.2.2 Mini-especificações – ME

- **Descrição :** é um artefato simples e obrigatório para qualquer sistema. As mini-especificações disponibiliza um algoritmo para cada DFD, DUC e métodos de classes;
- **Objetivo :** Detalhar os requisitos do sistema, facilitar a implementação (codificação) do mesmo e a comunicação entre os envolvidos no processo, principalmente os analistas e programadores;
- **Quem cria :** o analista;
- **Quem valida :** o gerente de produto;
- **Quando iniciar :** logo que uma primeira versão de detalhamento dos requisitos estiver validado;

6.3.4.2.3 Diagrama de Classes – DC

- **Descrição :** é um artefato mais complexo entretanto, mais poderoso e obrigatório para sistemas complexos e grandes;
- **Objetivo :** Permitir encapsulação, reuso, abstração e polimorfismo;
- **Quem cria :** o analista;
- **Quem valida :** o gerente de produto;
- **Quando iniciar :** logo que uma primeira versão de detalhamento dos requisitos baseados em DUC estiver validado;

Os esquemas de bancos de dados serão gerado automaticamente por software de apoio e refinado a cada iteração;

6.3.4.3 Codificação

Consiste da atividade de gerar instruções reconhecíveis por computador. A codificação seguirá os padrões definidos pela empresa, ou seja, padrões definidos no Estágio Supervisionado de Rodrigo F. de Albuquerque. Não se precisa justificar a codificação. Justifica-se entretanto a importância de atacar a codificação do produto apenas *após* a conclusão da etapa de Projeto de Baixo Nível. Codificação é *apenas* codificação. A etapa de codificação contém duas atividades:

- Codificação até o ponto em que o código compila. Normalmente, alguns testes são feitos durante a codificação para exercitar o código, mas tais testes não fazem parte das atividades de testes, já que não são testes formais;
- A Revisão do código é feita por um ou mais programadores indicados pelo gerente do produto.

6.3.4.4 Planos de Testes

Não será feito um planejamento dos casos de teste a serem realizados no sistema. Pois gasta-se bastante tempo planejando testes, então é mais viável torná-los automático utilizando uma ferramenta apropriada para este fim. A documentação gerada pelos testes será o **código de teste** e o **arquivo de log dos testes** contendo a entrada, o valor esperado e o retorno do método. Serão obrigatórios os Testes de Unidade e de Funcionalidades e ainda um Teste de Interface com o Usuário, e todos são de responsabilidade do programador;

O Teste de Interface com o Usuário será baseado numa lista de checagem de itens como:

- Funcionalidades dos botões;
- Ordem de tabulação dos controles;
- Teclas de atalhos para os principais controles;
- Verificação de máscaras de edição;
- Verificação das dicas (hints) sobre cada controle etc.

Será gerado um artefato para este fim.

6.3.4.5 Transição para o Produto

Esta etapa envolve atividades de geração de publicações, empacotamento, instalação ,**sugestão de novas funcionalidades** que possam gerar uma nova versão do produto no futuro e uma **lista de problemas/soluções** encontrados na iteração corrente com objetivo de solucionar problemas futuros mais facilmente e em menos tempo.

6.3.5 Implantação

Esta atividade envolve principalmente as publicações a serem realizadas. Entende-se por publicação qualquer documento que deva ser distribuído para o cliente final ou

para uma área da empresa fora do desenvolvimento. Tipicamente, as publicações incluem:

- Manuais do usuário;
- Ajuda on-line;
- Tutoriais para ajudar no uso do produto;
- Documentos de Marketing, tais como Visão Geral do Produto, folhetos, embalagem do produto;
- Documentos internos para uso pelo suporte e/ou setor de documentação: DRB, DRNF, PNS etc.
- Treinamento de usuários;
- Teste Alfa e Teste Beta;

Esta atividade é de responsabilidade do gerente de produto juntamente com o pessoal de Marketing da empresa. No caso da construção de tutoriais, pode ser realocado um programador para auxiliar na tarefa de codificação.

6.3.6 Manutenção e Suporte

A manutenção do produto é de vital importância para retardar o envelhecimento do mesmo. Mas não é fácil desenvolver e saber o que o cliente, mercado, deseja em seguida. A chave para isso está na integração entre o Suporte e o Desenvolvimento.

Esta integração se dá à medida que os clientes informam problemas encontrados no produto, ou sugerem o acréscimo de novas funcionalidades. Quando tais fatos ocorrem, o pessoal do suporte envia o documento de solicitação de alteração do produto (Anexo - B) para o gerente de produtos que dará sequência ao processo conforme descrito na seção 6.1.4.4 desse relatório.

O Estágio Supervisionado de Kalina Ramos Porto teve como produto base o núcleo do sistema de apoio a Comercialização dos Produtos da empresa – SISCOMS, onde

uma das ferramentas visa disponibilizar um ambiente onde o cliente possa solicitar as alterações via Internet servindo assim de ferramenta de apoio à manutenção dos produtos.

O Modelo do Suporte dos Produto da empresa é detalhado no relatório de Estágio Supervisionado de Dilane Diniz de França.

6.4 *Ferramentas de Automação para o Processo ByteCom*

Nessa seção foram eleitas algumas ferramentas de apoio para aplicação do processo na empresa. Estas ferramentas foram escolhidas seguindo os critérios adotados na seção 6.3.1.

6.4.1 TeamSource

O TeamSource é uma ferramenta da Borland/Inprise de controle de *workflow* que ajuda equipes de desenvolvimento de software no gerenciamento de tarefas diárias utilizando um ambiente de desenvolvimento compartilhado. Esta ferramenta foi escolhida por fornecer o melhor custo-benefício para a empresa, apesar de não ter sido encontrada muita documentação na web sobre a mesma.

6.4.2 DUnit

É uma framework de classes utilizadas para realizar testes de unidades e funcionais no ambiente de desenvolvimento Delphi. É distribuída gratuitamente pela Web e foi encontrada bastante documentação a respeito na Web.

6.4.3 Microsoft Project 2000

O Microsoft Project 2000 é uma ferramenta projetada para ajudar a gerenciar uma ampla gama de projetos. Por ser muito genérica, seria subutilizada na empresa. No entanto, pode-se monitorar e rastrear muito facilmente um projeto utilizando essa ferramenta, pois a mesma facilita a criação de gráfico essenciais para tal controle.

Como a mesma não se integra ao ambiente de desenvolvimento da ByteCom, recomenda-se utilizá-la em conjunto com o TeamSource.

6.4.4 Microsoft Office 97/2000

Será utilizada para criar os artefatos no formato ByteCom. Apesar de ser uma ferramenta de natureza genérica, é possível gerar artefatos com qualidade e rapidez sem que a empresa tenha que investir muito alto em ferramentas como o Rational Rose e outros mais.

7 Conclusão e Sugestões

Durante o desenvolvimento desse estágio, pôde-se perceber claramente toda a problemática gerada devida a ausência de um processo de desenvolvimento de software em uma empresa, por mais simples que fosse. De forma que é inconcebível, nos dias de hoje, pensar em uma empresa desenvolvedora de software que não utilize um processo formal de desenvolvimento, com condutas e responsabilidades bem definidas para todas as etapas do processo.

Sabemos que isso não é uma tarefa fácil, pois requer muita perseverança e disciplina por parte de toda a equipe envolvida, desde a concepção até a distribuição do produto para o cliente final. Mas também sabemos que este é o primeiro requisito para se produzir software de qualidade e a empresa que não produz com qualidade está fadada ao esquecimento por parte dos clientes.

Com essa motivação em mente, foi fundamental os conhecimentos adquiridos nas disciplinas da graduação, em especial, Sistemas de Informação I e II, Introdução a Bancos de Dados e principalmente, Engenharia de Software, pois essa foi uma das que mais me dediquei durante o curso e a que mais foi exigida no desenvolvimento do Estágio Supervisionado. Apesar de ter feito uso de todos os conceitos ministrados nessa disciplina, ainda foi necessária muita pesquisa extra sobre novos assuntos que não foram explorados pela mesma durante o seu curso. Dessa forma, recomenda-se uma atualização ou ampliação do conteúdo programático para essa disciplina.

Um fator que dificultou o desenvolvimento do estágio foi o excesso de disciplinas colocadas juntamente com o estágio supervisionado, mas esse fato foi contornado principalmente com muito esforço, dedicação, muito conhecimento e experiência adquirido durante os quase três anos de empresa incubada no POLIGENE, onde durante essa fase, o aluno é exposto a novas situações e problemas aos quais o mesmo só teria a chance de ter contato quando cursasse o Estágio Supervisionado ou quando saísse da universidade. Esse fator foi fundamental para conciliar as obrigações acadêmicas e as obrigações profissionais como sócio da ByteCom Sistemas.

Av. Aprígio Veloso , 882. Bodocongó
CEP 58109-907- Campina Grande,PB,
Brasil - Fone/Fax: (0xx83) 310-1438
E-mail: bytecom@bytecom.com.br
http://www.bytecom.com.br

Infelizmente, não foi possível implantar o PROCESSO BYTECOM, pois essa atividade levaria mais tempo que o disponível ao desenvolvimento do processo, pelo fato de ser uma tarefa continuada de validações e aplicações de artefatos e procedimentos, e até mesmo treinamento da equipe envolvida. No entanto, como um dos sócios, posso afirmar que a implantação desse processo é de prioridade máxima para nós que fazemos a ByteCom, pois caso contrário, nossa missão se tornaria impossível de se realizar.

Espera-se que no prazo máximo de um ano, a partir da data de início de implantação, se faça uma avaliação do PROCESSO BYTECOM e verifique-se se os resultados foram satisfatórios, ou se caso contrário, quais as melhorias que poderiam ser implementadas visando uma melhoria continuada desse processo.

8 Referências Bibliográficas

[1] KAN S. H. "Metrics and Models in Software Quality Engineering"; 1995

[2] C. Jones. Applied *Software* Measurement: Assuring Productivity and Quality. Prentice Hall, Second Edition, 1997.

[3] C. Hazan. Metodologia para o Uso de Indicadores na Gerência de Projetos de Desenvolvimento de *Software*. Tese de Mestrado, IME, Maio 1999. Desenvolvimento de Software".; RT032/DE9/OUT98

[4] HAZAN C., "Roteiro para Contagem de Pontos de Função", SERPRO/SUNAT/ATRJO/ATADE , Fevereiro/1999

[5] WEBER, Kival Chaves; MILLET, Paulo Barreira & JÚNIOR, Dorgival Brandão – Qualidade e Produtividade em Sofware: Termo de Referência do Sobprograma Setorial da Qualidade e Produtividade em Software, do Programa Brasileiro da Qualidade e Produtividade – PBQP. Editora Makron Books, Brasília, 1994.

[6] R-CYCLE - Projeto do Núcleo Softex para a Produção de Software Comercial. Procedimentos para o Desenvolvimento, Marketing, Vendas e Suporte. Partes de I a VI, 1995

[7] PRESSMAN, Roger S. - Engenharia de Software. São Paulo : Makron Books, 1995.

[8] LARMAN, Craig Understanding Requiriments in Applying UML and Pattems: An Introduction to Objet-Oriented Analisis and Desing, New Jersey, Prentice-Hall, 1998

[9] GAMA, Erich (e outros) Desing Pattems, Addison-Wesley Pub. Co., 1995

[10] POMPILHO, S. Análise Essencial – Guia Prático de Análise de Sistemas, Rio de Janeiro: Infobook, 1994.

[11] SCHROEDER,M. A Practical Guide to Object-Oriented Metrics, IT Pro November | December, 1999.

[12] Site na Internet:

http://www.sdmagazine.com/articles/2000/0012/0012a/0012a.htm

http://www.xprogramming.com/;

http://www.extremeprogramming.org/;

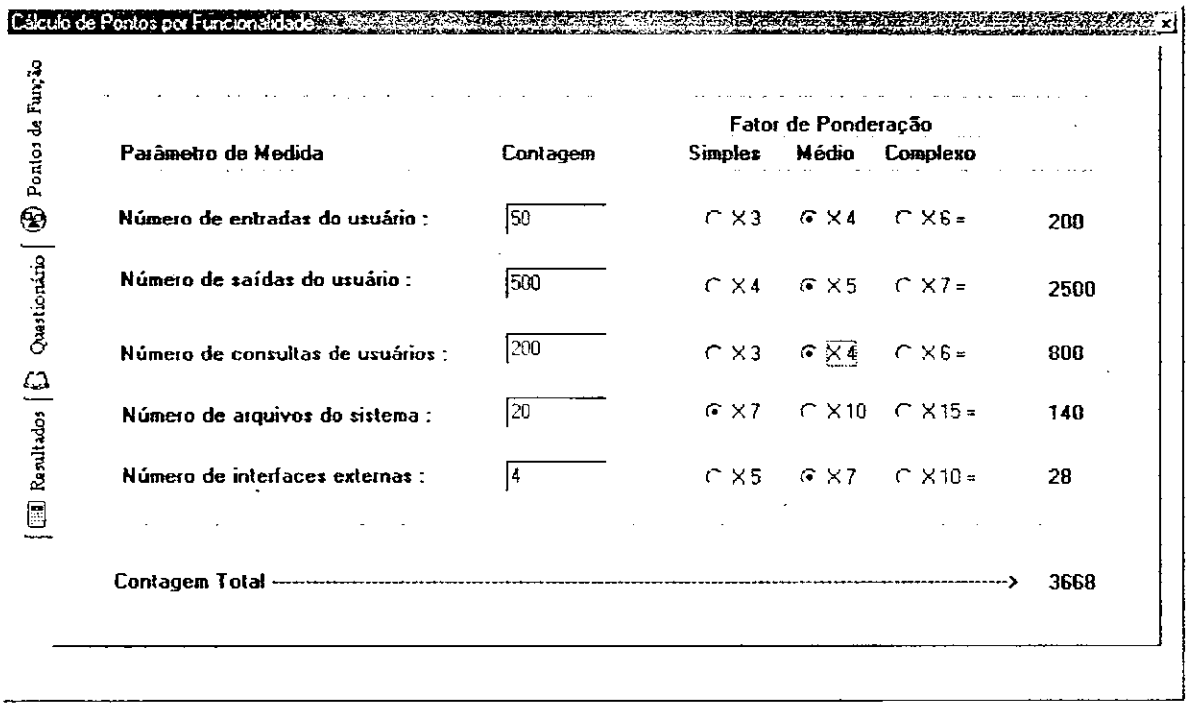
http://www.egroups.com/group/extremeprogramming;

entre outros.

BYTECOM
Av. Aprígio Veloso , 882. Bodocongó
CEP 58109-907- Campina Grande,PB,
Brasil - Fone/Fax: (0xx83) 310-1438
E-mail: bytecom@bytecom.com.br
http://www.bytecom.com.br

ANEXO – A: Ferramenta que auxilia no cálculo do Número de Pontos por Função de uma aplicação.

- Tela de entrada de dados



- **Procedimento:** cada membro da equipe de desenvolvimento, anônimamente, faz suas estimativas baseando-se em suas próprias experiências e algum histórico de projetos passados. Na sequencia, cada membro responde um questionário informando para cada questão o grau de influência da resposta, verifica e anota o número de Pontos por Função na tela de Resultado. Depois que todos tiverem concluído, os resultados são comparados e é feito um tratamento estatístico simples com os resultados de cada membro da equipe de desenvolvimento.

BYTECOM
Av. Aprigio Veloso , 882. Bodocongó
CEP 58109-907- Campina Grande,PB,
Brasil - Fone/Fax: (0xx83) 310-1438
E-mail: bytecom@bytecom.com.br
http://www.bytecom.com.br

Telas do questionário

Cálculo de Pontos por Funcionalidade

Pontos de Função | Questionário | Resultados

Questões	Sem Influência	Incidental	Moderado	Médio	Significativo	Essencial
1) O sistema requer backup e recuperação confiáveis ?	○	○	○	○	○	◉
2) São exigidas comunicações de dados ?	○	○	○	○	◉	○
3) Há funções de processamento distribuídas ?	◉	○	○	○	○	○
4) O desempenho é crítico ?	○	○	○	◉	○	○
5) O sistema funcionará num ambiente operacional intensivamente utilizado ?	○	○	○	○	○	◉
6) O sistema requer entrada dados on line ?	○	○	○	○	◉	○
7) A entrada de dados on line exige que a transação de entrada seja elaborada em múltiplas telas ou operações?	○	○	○	◉	○	○

>>

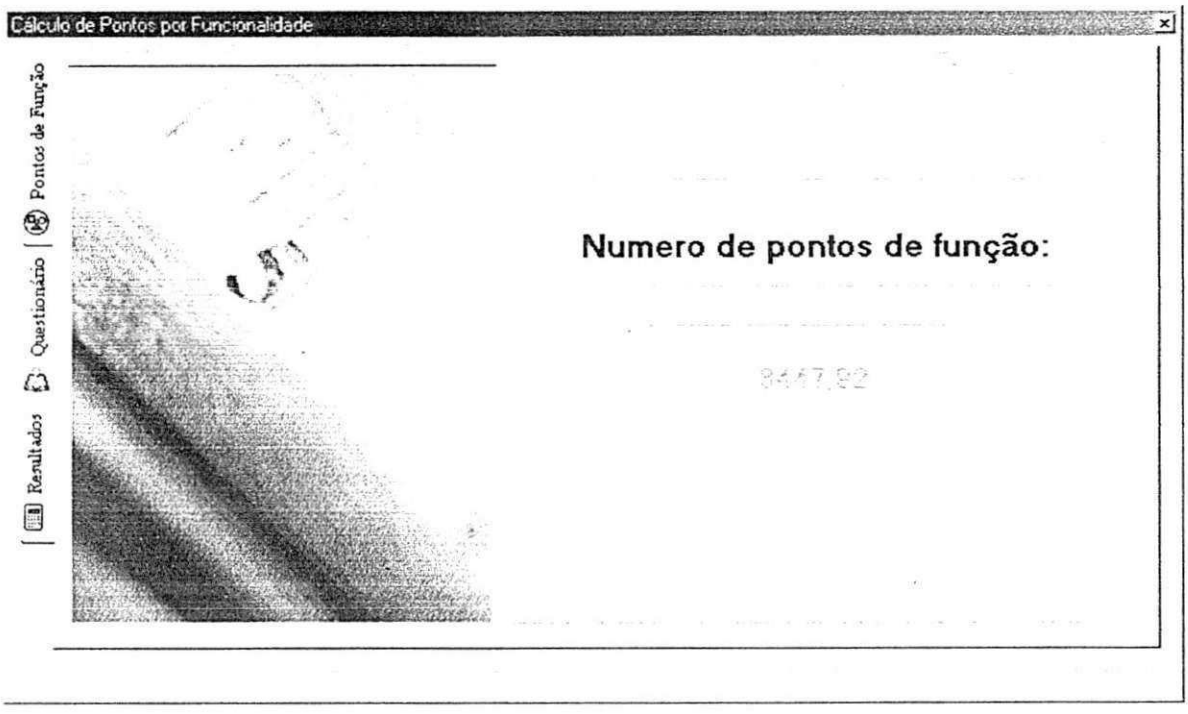
Cálculo de Pontos por Funcionalidade

Pontos de Função | Questionário | Resultados

Questões...	Sem Influência	Incidental	Moderado	Médio	Significativo	Essencial
8) Os arquivos -mestres são atualizados on line ?	○	○	○	○	○	◉
9) A entrada , saída, arquivos ou consultas são complexos ?	○	○	○	○	◉	○
10) O processo interno é complexo ?	○	○	○	○	◉	○
11) O código foi projetado de forma a ser reutilizável ?	○	◉	○	○	○	○
12) A conversão e a instalação estão incluídas no projeto ?	○	○	○	○	○	◉
13) O sistema é projetado para múltiplas instalações em diferentes organizações ?	○	○	○	○	○	◉
14) A aplicação é projetada de forma a facilitar mudanças e uso pelo usuário ?	○	○	○	○	○	◉

<<

➢ **Tela de Resultados**



BYTECOM
Av. Aprígio Veloso , 882. Bodocongó
CEP 58109-907- Campina Grande,PB,
Brasil - Fone/Fax: (0xx83) 310-1438
E-mail: bytecom@bytecom.com.br
http://www.bytecom.com.br

ANEXO – B: FORMULÁRIO DE PEDIDO DE MUDANÇAS

Número do Formulário : ☐

Dados do Cliente

Código do Cliente :_____ **Data de Solicitação :** _____
Nome do Responsável : _____
Rua : _____ **Nº** _____
Bairro : _____ **Fax :** _____
E-mail : _____ **Telefone :** _____

Dados do Produto

Software Utilizado : _____
Versão : _____ **Data de Aquisição :** _____

Sobre o Pedido

Pedido de Correção ☐ **Pedido de Adição de Funcionalidade** ☐

Módulo de Produção ☐ **Módulo Financeiro** ☐
Módulo de Estoque ☐ **Módulo de Vendas** ☐
Módulo de Compras ☐

Solicitações

_____
_____
_____
_____
_____
_____
_____

_____
Assinatura do Requerente

BYTECOM
Av. Aprigio Veloso , 882. Bodocongó
CEP 58109-907- Campina Grande,PB,
Brasil - Fone/Fax: (0xx83) 310-1438
E-mail: bytecom@bytecom.com.br
http://www.bytecom.com.br

ANEXO – C: DECLARAÇÃO DA EMPRESA

DECLARAÇÃO

Declaramos para os devidos fins que o Sr. WILLIAMS ALVES DANTAS, solteiro, residente na rua Treze de Maio, 233 – Edf. Morada Nobre – apt 101, Bairro Centro, Campina Grande – Paraíba, cumpriu estágio supervisionado nesta empresa, na função de Analista, durante os meses de Dezembro 2000 à Abril de 2001, onde desenvolveu as atividades relatadas no Plano de Estágio.

Campina Grande, 15 de Maio de 2001

BYTECOM SISTEMAS LTDA.
Robert Kalley Cavalcanti de Menezes
Supervisor Técnico

Av. Aprígio Veloso , 882. Bodocongó
CEP 58109-907- Campina Grande,PB,
Brasil - Fone/Fax: (0xx83) 310-1438
E-mail: bytecom@bytecom.com.br
http://www.bytecom.com.br

ANEXO – D: PLANO DE ESTÁGIO

1 Ambiente do Estágio

1.1 Dados da Empresa

Razão Social: ByteCom Sistemas Ltda.

CNPJ: 03.688.196/0001-47

Endereço: Av. Aprígio Veloso,882 - Bodocongó, Campina Grande, Paraíba

Tele/Fax: (83)310-1438

E-mail: bytecom@bytecom.com.br

URL: http://www.bytecom.com.br

1.2 Nicho de mercado

A ByteCom Sistemas é uma empresa encubada no CENTRO SOFTEX GENESIS de Campina Grande - POLIGENE que atual na informatização de indústrias/empresa do setor de Alimentação. Os principais clientes da ByteCom Sistemas são panificadoras, confeitarias, restaurantes, fast foods e pizzarias.

1.3 Estado da Informática na Empresa

A empresa atualmente utiliza a estrutura do laboratório de prototipagem de software do Centro Softex Genesis de Campina Grande – Poligene. No total são 08 (oito) computadores IBM com processadores Pentium Celeron 333 MHz, com 32 e 64 MB de memória RAM e um Servidor Netfinity 3500 também da IBM. Todos os computadores possuem acesso à Internet 24 horas por dia .

A equipe de desenvolvimento é formada basicamente por alunos da graduação do curso de Ciências da Computação e do curso de Desenho Industrial da UFPB, Campus II. O supervisor é Robert Kalley Menezes, coordenador do Centro Softex Genesis de

Campina Grande – Poligene e professor do Departamento de Sistemas e Computação – DSC.

1.4 Dados do(a) Estagiário(a)

Nome : Williams Alves Dantas

Matrícula : 9611039

Endereço : R. Treze de Maio, 233, Apt 101 – Centro, Campina Grande –PB

Telefone : (83)972-3082

E-mail : williams.dantas@lcc.ufpb.br

williamsad@globo.com

2 Supervisão

2.1 Supervisor técnico

Nome: Roberto Kalley Cavalcanti de Menezes

Naturalidade: Campina Grande - PB

E-mail: kmenezes@dsc.ufpb.br

Endereço: Av. Manoel Alves de Oliveira, 1095 Cidade Universitária

58.100-000 - Campina Grande - Pb

2.2 Supervisor acadêmico

Nome: Francilene Procópio Garcia

Naturalidade: Campina Grande - PB

E-mail: garcia@dsc.ufpb.br

Endereço: Av. Aprígio Veloso, 882 Cidade Universitária

58.109-970 - Campina Grande - Pb

3 Resumo do Problema Objeto do Estágio

A ByteCom Sistemas não possui um processo formal de desenvolvimento e gestão de projetos de software que se adapte às constantes mudanças nos requisitos de mercado e na própria empresa (*turn over* alto).

4 Proposta de Solução

4.1 Objetivo Geral

Dotar a empresa Bytecom de uma abordagem flexível de desenvolvimento para processos de software que possa ser adotada pelo seu time em situações de mudanças de requisitos, em novos projetos e no suporte ao cliente, de forma a atender as demandas do mercado.

4.2 Objetivo Específico

Elaborar e implantar um processo de desenvolvimento de software, orientado ao desenvolvimento de sistemas de informação, que apresente condutas gerenciais para planejamento, análise, projeto, implementação, testes e disponibilização de soluções junto aos clientes; (PROCESSO DE DESENVOLVIMENTO DE SOFTWARE DA BYTECOM)

Av. Aprigio Veloso , 882. Bodocongó
CEP 58109-907- Campina Grande,PB,
Brasil - Fone/Fax: (0xx83) 310-1438
E-mail: bytecom@bytecom.com.br
http://www.bytecom.com.br

5 Atividades a Serem Desenvolvidas e Cronograma

5.1 Atividades

PROCESSO DE DESENVOLVIMENTO DE SOFTWARE DA BYTECOM	
Descrição da Atividade	**Duração**
1. Revisão bibliográfica (processos iterativos com ênfase em sistemas de informação)	40 horas
2. Levantamento do nível atual de maturidade da empresa no desenvolvimento de produtos de software, envolvendo as fases de planejamento, análise, projeto, implementação, teste e entrega do produto	40 horas
3. Elaboração de um "processo Bytecom", centrado na abordagem iterativa, capaz de se adaptar às mudanças nos requisitos do cliente, contemplando: condutas para estimar tempo e custos, templates de controle, planos de testes, e gestão de versão	120 horas
4. Geração de documentos básicos para mapeamento de atividades chaves durante o processo de desenvolvimento - formato ByteCom	40 horas
5. Eleição de ferramentas de apoio para aplicação do processo na empresa	20 horas
6. Aplicação do "processo Bytecom" em sistemas existentes para validação/refinamento de atividades chaves sugeridas	80 horas
7. Relatório final de resultados	20 horas

Total = 360 horas

5.2 Cronograma de Atividades Proposto

Número da Atividade	Duração (horas)								
	40	80	120	160	200	240	280	320	360
1									
2									
3									
4									
5									
6									
7									

5.3 Cronograma de Atividades Realizado

Número da Atividade	Duração (horas)								
	40	80	120	160	200	240	280	320	360
1									
2									
3									
4									
5									
6									
7									