

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/341327433>

Quality with Tests and Management

Book · January 2015

CITATIONS

0

READS

56

1 author:



[Alvaro F. Pinheiro](#)

University of Pernambuco

29 PUBLICATIONS 5 CITATIONS

SEE PROFILE

Fundamentos da Engenharia de Software

Qualidade com Testes e Gerência

PINHEIRO, Álvaro Farias
Autor

Fundamentos da Engenharia de Software

Qualidade com Testes e Gerência

Publicação 2015

O autor acredita que todas as informações aqui apresentadas estão corretas e podem ser utilizadas para qualquer fim legal. Entretanto, não existe qualquer garantia explícita ou implícita, de que o uso de tais informações conduzirá sempre ao resultado desejado. Os nomes de sites e empresas, por ventura, mencionados, foram utilizados apenas para ilustrar os exemplos, não tendo vínculo nenhum com o livro, não garantindo a sua existência nem divulgação. Eventuais erratas estarão disponíveis para download no site de publicação.

Algumas imagens utilizadas neste livro foram obtidas a partir da Internet.

Dados da Publicação

Pinheiro, Álvaro Farias

Fundamentos da Engenharia de Software: Qualidade com Testes e Gerência

1. ed. – Volume VI

Recife, 2015

Selo Editorial: Publicação Independente

Bibliografia

ASIN B017EQZ976

ISBN 9781520276960

1. Engenharia de Testes
2. Gestão de Projetos
3. PMBOK

Fundamentos da Engenharia de Software

Qualidade com Testes e Gerência

Publicação Independente
Livro em português com o título
Fundamentos da Engenharia de Software
Qualidade com Testes e Gerência
Volume 6
1ª Edição
Recife - Pernambuco – Brasil
Primeira revisão em Novembro de 2015
Segunda revisão em Junho de 2016

Fundamentos da Engenharia de Software Qualidade com Testes e Gerência

Ao meu filho Alberto Albuquerque de Freitas
Pinheiro que homenageia com o nome a pessoa
íntegra que era o seu avô paterno.

Fundamentos da Engenharia de Software

Qualidade com Testes e Gerência

“Não se consegue controlar o que não se consegue medir”
Tom Demarco

SUMÁRIO

1	Processos de Desenvolvimento.....	1
1.1	Evolução.....	2
1.2	Modelo Cascata.....	4
1.3	Modelo Prototipação.....	5
1.4	Modelo Espiral.....	6
1.5	Modelo Iterativo e Incremental	7
1.6	Metodologia Tradicional.....	8
1.7	Metodologia Ágil	8
1.8	Processo RUP	9
1.9	Processo Scrum	10
1.10	Metodologias-Modelos-Processos-Técnicas	11
1.11	Ferramentas	11
1.12	Sugestões de Leitura.....	11
1.13	Referências Web	12
1.14	Referências Bibliográficas	12
2	Engenharia de Testes.....	14
2.1	Tipos de <i>Bug</i>	15
2.2	Ciclo de Vida de um <i>Bug</i>	15
2.3	Testador	16
2.4	Testes versus Inspeções.....	16
2.5	Técnicas de Encontrar <i>Bugs</i>	17
2.6	Modelos	17
2.6.1	Modelo Cascata	18
2.6.2	Modelo V	18
2.6.3	Modelo Incremental	19
2.6.4	Modelo Codificar e Testar	19
2.6.5	Modelos Ágeis - Test Driven Development (TDD).....	20
2.7	Conceitos em Testes.....	20
2.7.1	Unidade, Componente e Integração.....	21
2.7.2	Test Driver.....	21
2.7.3	Stub de Test.....	21
2.7.4	Processos de Testes.....	21

SUMÁRIO

2.7.5 Estágio de Teste	21
2.7.6 Testes Unitários	22
2.7.7 Testes de Componente	23
2.7.8 Testes de Integração.....	23
2.7.9 Testes de Sistema.....	23
2.7.10 Testes de Aceitação.....	23
2.7.11 Tipos de Testes.....	24
2.7.12 Testes de Carga.....	24
2.7.13 Testes de Configuração	24
2.7.14 Testes de Compatibilidade	24
2.7.15 Testes de Performance	24
2.7.16 Testes de Documentação	24
2.7.17 Testes de Internacionalização.....	25
2.7.18 Testes de Usabilidade.....	25
2.7.19 Testes de Regressão	25
2.7.20 Abordagens de Testes	25
2.7.21 Caso de Testes	25
2.7.22 Ferramentas para Testes	26
2.7.23 Testes Positivos	26
2.7.24 Testes Negativos.....	26
2.8 Sugestões de Leitura.....	26
2.9 Referências Bibliográficas	27
3 Gestão de Projetos	28
3.1 História	29
3.2 Conceitos.....	29
3.3 Portfólio	29
3.4 Programa.....	31
3.5 Projeto	32
3.6 PMI	33
3.7 PMP	34
3.8 Gerenciamento	35

SUMÁRIO

3.9 Planejamento.....	35
3.10 Gerenciar.....	36
3.11 Processo.....	36
3.12 Grupos de Processos	36
3.13 Gerente de Projetos	37
3.14 Interações de Processo	37
3.15 MS-Project.....	37
3.15.1 Gestão de Prazos.....	39
3.15.2 Gestão do Escopo.....	41
3.15.3 Gestão dos Recursos.....	42
3.15.4 Gestão de Custos.....	44
3.15.5 Gestão de Integração	45
3.15.6 Gestão da Comunicação	47
3.15.7 Gestão de Aquisições	48
3.15.8 Gestão de Riscos	48
3.15.9 Gestão de Qualidade	49
3.16 IBM Primavera P6	52
3.17 Ferramentas	58
3.18 Sugestões de Leitura.....	58
3.19 Referências Web	58
3.20 Referências Bibliográficas	59
4 Referências Bibliográficas	60
5 Referências Web	61
6 Biografias.....	62
7 Artefatos de Projetos de <i>Software</i>	67
8 Ferramentas	69
9 Sugestões de Leitura.....	70

ILUSTRAÇÕES

Figura 1.1 Processo.....	1
Figura 1.2 Evolução dos Processos	2
Figura 1.3 Evolução da UML	4
Figura 1.4 Modelo Cascata.....	5
Figura 1.5 Modelo Prototipação.....	6
Figura 1.6 Modelo Espiral	7
Figura 1.7 Modelo Iterativo e Incremental.....	8
Figura 1.8 Metodologia Tradicional.....	8
Figura 1.9 Metodologia Ágil	9
Figura 1.10 Processo RUP	9
Figura 1.11 Processo Scrum	10
Figura 3.1 Portfólio.....	30
Figura 3.2 Programa	32
Figura 3.3 Projeto	32
Figura 3.4 MS-Project Apresentação	38
Figura 3.5 MS-Project Menu de Controle.....	38
Figura 3.6 Menu Suspenso.....	39
Figura 3.7 MS-Project Barra de Ferramentas	39
Figura 3.8 MS-Project Definição do Calendário.....	40
Figura 3.9 MS-Project Definir Início do Projeto	40
Figura 3.10 MS-Project Definir Tarefas.....	41
Figura 3.11 MS-Project Definir Relacionamento	41
Figura 3.12 MS-Project Tipos de Relacionamento	42
Figura 3.13 MS-Project Alocação de Recursos	42
Figura 3.14 MS-Project Alocação de Recursos e Tarefas	43
Figura 3.15 MS-Project Verificar Super Alocação.....	43
Figura 3.16 MS-Project Informações	43
Figura 3.17 MS-Project Tabela de Custos	44
Figura 3.18 MS-Project Definir Linha Base.....	46
Figura 3.19 MS-Project WBS.....	46
Figura 3.20 MS-Project Definir e-mails	47
Figura 3.21 MS-Project Modos de Visualizações	49
Figura 3.22 Primavera Home	55
Figura 3.23 Tela de Abertura	55
Figura 3.24 Primavera Barra de Ferramentas	57
Figura 3.25 Primavera Barra de Opções	57

SIGLAS e ACRÔNIMOS

ANSI - *American National Standards Institute*
IRUP - *IBM Rational Unified Process*
ISO - *International Organization for Standardization*
ITIL - *Information Technology Infrastructure Library*
KM - *Knowledge Management*
PMBOK - *Project Management Body of Knowledge*
PMI - *Project Management Institute*
QSM - *Quantitative Software Management*
SEI - *Software Engineering Institute*
SWEBOK - *Software Engineering Body of Knowledge*

NOTAÇÕES

Este livre adota as seguintes notações, que devem ser observadas para o perfeito entendimento do material aqui produzido.

Notações Técnicas:

<..> - Menor que / Maior que, nos comandos SQL usado para metadados.

<..> - Menor / Maior, no texto indica que é um *hyperlink* (*link* para um endereço web).

[..] - Abre e fecha parênteses, no texto indica utilização de uma sigla.

(..) - Abre e fecha colchetes, nos comandos SQL indica opcionalidade.

| - Barra, para indicar “ou lógico”.

_ - Sublinhado, nos comandos SQL indica o *default* (padrão).

_ - Sublinhado, no texto indica que é um *hyperlink* (*link* para um endereço web).

Negritos - Títulos de capítulos, seções e subseções.

Itálico - Para palavras de língua estrangeira.

SOBRE O AUTOR



Álvaro Farias Pinheiro tem mais de 30 anos de experiência no campo da computação e nestes, mais de 15 anos na área acadêmica, lecionando disciplinas com foco em engenharia de software.

É mestre em Engenharia de Software pelo Centro de Estudos e Sistemas Avançados do Recife (CESAR) com a dissertação *Uso do Balanced Scorecard na Avaliação de Fábricas de Software* utilizadas como Instrumento de Ensino de Tecnologia da Informação Baseado em Problemas Reais. Especialista em Gestão Pública pela Universidade de Pernambuco (UPE) e em

Metodologias de Desenvolvimento pela União Brasileira de Tecnologia. É Bacharel em Sistemas de Informação pela Faculdade Integrada do Recife (FIR), iniciando sua vida acadêmica no Bacharelado em Ciência da Computação da Universidade Católica de Pernambuco (UNICAP).

Iniciou as atividades profissionais em 1986 como Programador na Elógica Microsistemas desenvolvendo para baixas e altas plataformas nas linguagens dBase, Clipper, FoxPro, Cobol e Natural. Como Analista de Sistemas passou pelas instituições: Centro de Prestação de Serviços Técnicos de Pernambuco (CETEPE), Empresa de Fomento da Informática do Estado de Pernambuco (FISEPE), Instituto de Tecnologia em Informática (ITECI), Apply Solutions e está hoje na Agência Estadual de Tecnologia da Informação do Estado de Pernambuco (ATI-PE) por concurso público, trabalhando como Analista em Gestão da Tecnologia da Informação e Comunicação, em exercício na Procuradoria Geral do Estado de Pernambuco (PGE-PE), atuando como Coordenador da Área de Sistemas.

Começou a lecionar em 1992 em cursos técnicos nas empresas ITECI e IBRATEC, e desde 2006 é professor de cursos de graduação e pós-graduação em computação nas instituições de ensino superior da região, a citar, Universidade Federal Rural de Pernambuco (UFRPE), Instituto Federal de Pernambuco (IFPE), Faculdade Estácio do Recife, Faculdade Integrada do Recife (FIR), Faculdade Joaquim Nabuco, Faculdade do Recife, Unibrtec e durante esses anos realizou diversos cursos de extensão como os de Análise de Ponto de Função, Administração de Redes, Linguagens de Programação Natural, Centura, Delphi e CSharp, Banco de Dados Adabas, MySQL, Postgre e Oracle, Análise Estruturada e Orientada a Objetos, entre outros.

Possui experiência na área de Engenharia de *Software* nas disciplinas de Teoria da Computação, Sistemas de Informação e Colaborativos, Desenvolvimento Estrutural e Orientado a Objetos para aplicações Desktop, Web e Mobile,

SOBRE O AUTOR

Processos de Desenvolvimento IRUP, XP e Scrum, Banco de Dados SQLServer e Oracle, e Gestão de Projetos com PMBOK e SWEBOK. Trabalha atualmente na coordenação de projetos e com educação presencial e a distância, ministrando aulas e realizando pesquisas em ambientes virtuais de ensino e linguagens de programação.

<http://www.alvarofpinheiro.eti.br>

<http://br.linkedin.com/in/alvarofpinheiro>

<https://sites.google.com/site/alvarofpinheiro>

<http://www.facebook.com/alvarofpinheiroaulas>

PREFÁCIO

Este livro é o resultado do uso da ferramenta MS Project da Microsoft utilizada na aplicação dos conceitos de gestão de projetos do PMBOK com as premissas da engenharia de testes para aquisição de qualidade nos produtos de software.

O livro aborda os seguintes assuntos:

- **Capítulo 1** *Processos de Desenvolvimento*. Neste capítulo são exibidos os conceitos básicos dos processos de desenvolvimento de *software*, evidenciando a necessidade de qualidade na construção de *software*, conceituando a diferença entre desenvolvimento *Adhoc* e com processo.
- **Capítulo 2** *Engenharia de Testes*. Neste aborda-se a necessidade de qualidade no desenvolvimento de *software*, com o uso da engenharia de testes, introduzindo de forma básica os processos, tipos, técnicas e ferramentas utilizadas nos testes de *software*.
- **Capítulo 3** *Gestão de Projetos*. Neste mostra-se a necessidade de acompanhamento de resultados na gestão de projetos utilizando-se para isso os conceitos básicos de gerenciamento de projeto do *Project Management Institute* (PMI) através da ferramenta de projetos MS-Project da Microsoft.

Finalizando, este livro possui várias subseções que disponibilizam *hyperlink's* para *wiki's* sobre os assuntos abordados, como também links para outros *ebook's* utilizados como referencial bibliográfico para o conteúdo produzido.

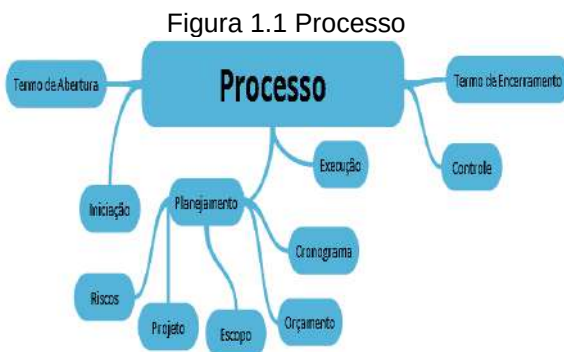
Álvaro Farias Pinheiro

1 Processos de Desenvolvimento

A Engenharia de Software é um segmento da Computação que existe decorrente da necessidade de se construir *software* com qualidade, buscando planejar e organizar o desenvolvimento, e dentro deste campo usar processo de desenvolvimento aliado a gestão de projetos com aplicação de técnicas de engenharia de testes ontem a qualidade desejada na construção de um projeto de software.

Para começar vamos conhecer as diferenças entre os termos metodologia, processo, modelo e técnica, que alguns fazem seu uso como se fossem sinônimos, o que não são.

Metodologia é um conjunto de práticas. Já as práticas são um conjunto de ações estruturadas que podem ser repetidas várias vezes durante um processo de desenvolvimento de *software*. Assim sendo uma metodologia é um conjunto que envolve várias práticas como gerenciamento de projetos, engenharia de requisitos, análise e modelagem, codificação, testes, gerencias como as de configuração, ambiente, risco, qualidade, aquisição, entre várias outras, que vão depender do processo adotado.



Fonte: Próprio Autor

Agora vamos definir Processo e Modelo. Apesar de se confundirem, não se trata da mesma coisa. Podemos dizer que processo de desenvolvimento de *software* é um conjunto de atividades, parcialmente ordenadas, com a finalidade de obter um produto de *software*. Realizando o mais corretamente possível as etapas de uma construção de *software*. Os modelos são as atividades que fazem parte de um processo de engenharia de software. É o primeiro passo para se construir um processo, pois o mesmo deve ser baseado em um modelo. Concluindo as definições, podemos definir Técnica como sendo um m procedimento ou

mesmo um conjunto de procedimentos, que tem como objetivo alcançar um determinado resultado. E segundo os estudiosos do assunto, se caracteriza por ser consciente reflexiva e inventiva.

Em uma análise mais básica, teríamos. Metodologia como um conjunto de técnicas (práticas) que repetidas sempre tendem a chegar ao mesmo resultado esperado. E um processo é a forma de organizar essas técnicas, onde se conhece quem faz como faz, e quando faz utilizado no desenvolvimento de um Projeto. Nos dias atuais, normalmente se classificam as metodologias em tradicionais ou pesadas e ágeis. No próximo tópico vamos falar sobre a evolução dos processos.

1.1 Evolução

Como se pode observar na figura 3.2 entre as décadas de 50 e 80, o que existia era a chamada análise estruturada que posteriormente evoluiu para análise essencial. Sendo a primeira forma organizada de desenvolvimento, focada na qualidade da construção de *software*, naturalmente muitos problemas eram inerentes a essa forma, mas inevitavelmente, era sim, a melhor forma de se construir uma aplicação.

Figura 1.2 Evolução dos Processos



Fonte: Próprio Autor

Mas, o grande problema desse modelo, consistia na grande quantidade de simbologias para as diversas fases de construção. Explicando melhor, um analista fazia fluxogramas para retratar a solução de um problema e uma série de outro diagrama, a exemplificar, diagrama de contexto, macro-diagrama, diagrama de fluxo de dados, e por aí vai.

O grande problema é que para banco de dados, existia outra diagramação, o modelo de entidade relacionamento, e para desenvolvimento, a codificação propriamente dita, não se tinha uma simbologia padrão. Vale salientar, que nesse período, vários estudiosos passaram a desenvolver suas próprias simbologias, criando verdadeiros núcleos, dificultando o entendimento entre os profissionais de escolas diferentes.

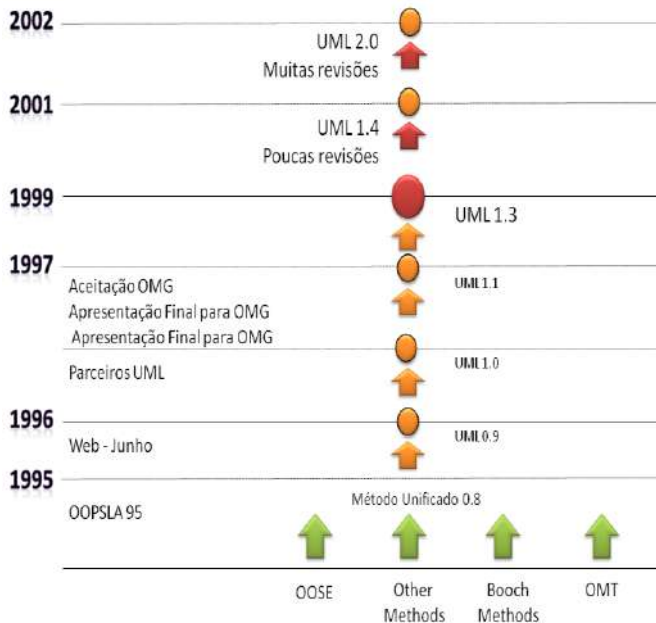
Esse período, onde se encontravam as linguagens de estruturadas e textuais, como COBOL, Fortran, pascal, Clipper e tantas outras, havia uma lacuna muito grande entre os analistas de sistemas e os programadores.

É entre as décadas de 80 e 90 que se começa a se tornar comum o uso da orientação a objetos. Esse novo paradigma de desenvolvimento surge no mundo profissional quase em paralelo com os sistemas multitarefa, que viriam a substituir os monotarefas com suas linguagens estruturadas e textuais. Essa nova forma de programar veio com a orientação a eventos e interface gráfica a GUI. Então, definiram-se novos conceitos como classe, objeto, herança, encapsulamento e polimorfismo. Assim, os estudiosos desse tema, começam a construir suas simbologias, correndo o risco de surgirem novos núcleos e escolas independentes que sem dúvida, iriam complicar ainda mais o entendimento dessa árdua tarefa que é construir *software*.

Mas, em meados de 90, A SEI cria um grupo chamado OMG, que tem como responsabilidade evitar que esses núcleos surgissem. Assim, nessa época os principais estudiosos do tema foram convidados para participar desse grupo. Esses nomes são: Grady Booch que manifesta a necessidade de unificar critérios, James Rumbaugh se une a Booch em outubro de 1994 e elaboram a versão 0.8 do *Unified Method* e por fim Ivar Jacobson que completa o trio. Esses três nomes passam a serem considerados os pais da orientação a objetos, definindo os padrões da Linguagem de Modelagem Unificada, que na sua sigla em inglês é conhecida como UML. Passando a ser o modelo aceito como padrão pela OMT e assim pela SEI e por consequência pelo mercado.

Com a simbologia padronizada, evitando aquela chuva de diagrama que variavam de escola para escola, precisava-se agora definir um padrão de desenvolvimento para esse novo paradigma. Mas, padrões já existiam e vamos falar sobre eles.

Figura 1.3 Evolução da UML



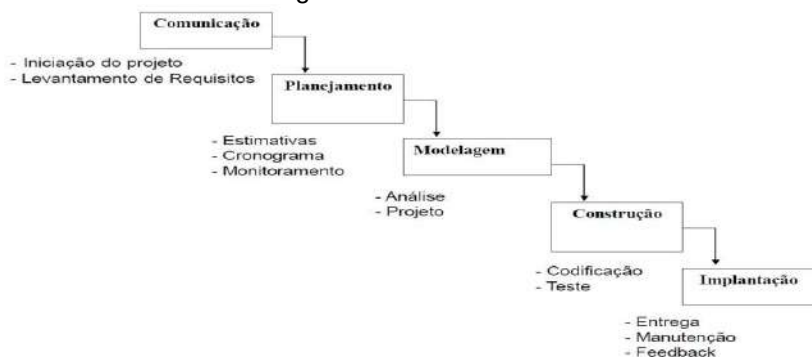
Fonte: Internet

1.2 Modelo Cascata

O modelo cascata é uma forma sequencial de desenvolvimento de *software*, onde o ciclo de vida de um projeto começa pelos requisitos seguindo pelas especificações e modelagem, passando pela codificação e testes, até chegar à implantação e manutenção.

O problema desse modelo é que todas essas etapas são executadas, uma após a outra.

Figura 1.4 Modelo Cascata



Fonte: Internet

Com a observação de que cada etapa de ser exaurida antes de passar para a próxima, ocasionando um grande lapso de tempo entre análise, projeto e implantação. Essa distância entre as fases macros geram inevitavelmente problemas nos requisitos, já que esses podem mudar pela própria natureza do negócio ou pelo problema mais comum, a comunicação entre os envolvidos. Algumas características desse modelo são: Execução sequencial das atividades; Levantamento de todos os requisitos no início do projeto (algo que a experiência mostrou ser muito difícil); Talvez a característica mais negativa desse modelo, seja a de que o usuário é envolvido apenas no início do projeto e na entrega do produto final, aumentando exponencialmente as chances de falha durante o projeto; e as mudanças nos requisitos não são bem vistas, pois gera retrabalho e consequentemente aumento dos custos. Mas, mudança de requisitos é inevitável na construção de qualquer *software*, por mais estável que seja.

1.3 Modelo Prototipação

Uma forma de lidar com a impaciência dos usuários no modelo cascata, foi a criação, ou melhor, a adaptação de uma técnica ao modelo que rendeu a denominação de prototipação e assim, surgiu um novo modelo. Que na verdade era o modelo em cascata, mas servido da técnica de fazer protótipos para que o cliente durante o processo tivesse condições de acompanhar o andamento, visualizando como o *software* ficaria. Mas, o problema desse modelo, era que, sua premissa dizia que o protótipo deveria ser descartável, isto é, apresentado ao cliente, este faria as observações, e no local de desenvolvimento (fábrica de *software*) esse protótipo deveria ser descartado e o desenvolvimento, seria continuado no modelo real. Bom, pura ilusão, pois, o que os desenvolvedores faziam era continuar evoluindo o protótipo que virava o entregável. Além dos problemas já citados acima, esses persistiam, já que a forma de construir o *software* não mudara muito.

Figura 1.5 Modelo Prototipação

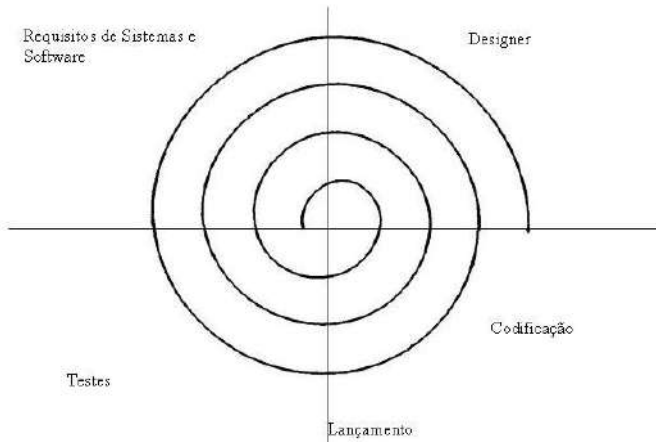
Fonte: Internet

1.4 Modelo Espiral

Até que surge o modelo espiral, isto é, você faz os levantamentos dos requisitos de *software*. Importante, não de todo o projeto, mas do que foi planejado, isto é, em etapas, normalmente do que é mais relevante para o que menos. Porém, dependo muito das necessidades do projeto. Depois desse levantamento, faz-se a diagramação do que foi requisitado, codifica-se, testa-se e implanta-se. Pronto.

Nesse momento o cliente interage novamente com o *software*, fazendo que o lapso de tempo seja bem menor, muito menor. O que ao longo do processo de amadurecimento dos modelos, viu-se ser muito proveitoso para o sistema. Afinal de contas, é para o cliente que se faz o *software*. Nesse momento, existem ajustes a serem combinadas e nova repetição do fluxo se faz, e se fará tantas vezes se faça necessário ou se tenha planejado.

Figura 1.6 Modelo Espiral



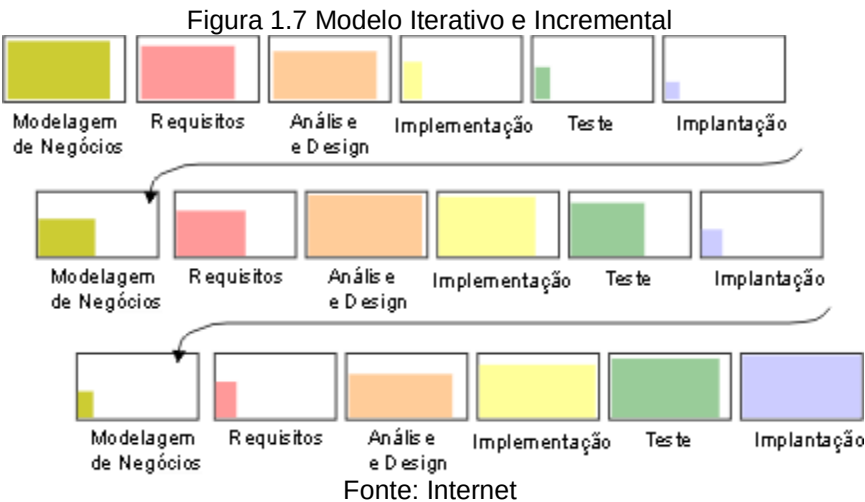
Fonte: Internet

Esse modelo se mostrou muito mais próximo as reais necessidades de um bom desenvolvimento, e dele surgem o que efetivamente ganhou o mercado. Mas, por que esse não se consolidou. Simples, não se fazia ideia que permitir a interação com o cliente entre as fases de desenvolvimento daria tanto problema, assim, foi necessário planejar essa interação, surgindo às iterações.

1.5 Modelo Iterativo e Incremental

O modelo iterativo e incremental consiste em fazer o desenvolvimento do *software* no modelo espiral, mas sabendo que em cada iteração, isto é, nova rodada de fases, talvez se tenha que corrigir falhas das rodadas anteriores, além de implementar as novas funcionalidades da rodada corrente. Simples, direto e objetivo.

Esse modelo se mostrou o mais próximo da realidade do desenvolvimento de projetos de *software*. Se tornando a base para uma grande gama de processos hoje vigentes.



1.6 Metodologia Tradicional

São métodos que enfatizam o processo com todo o formalismo necessário para garantir o controle das atividades de desenvolvimento de *software*, dando muita importância a documentos escritos. As pessoas envolvidas diretamente e indiretamente com o *software* devem utilizar o processo definido.

Figura 1.8 Metodologia Tradicional

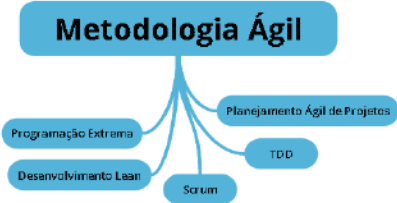


Fonte: Próprio Autor

1.7 Metodologia Ágil

São métodos que enfatizam a comunicação em tempo real entre as pessoas envolvidas, dando menos importância a documentos escritos. As pessoas envolvidas diretamente com o *software* devem estar preferencialmente disponíveis em relação a tempo e local da melhor forma possível.

Figura 1.9 Metodologia Ágil



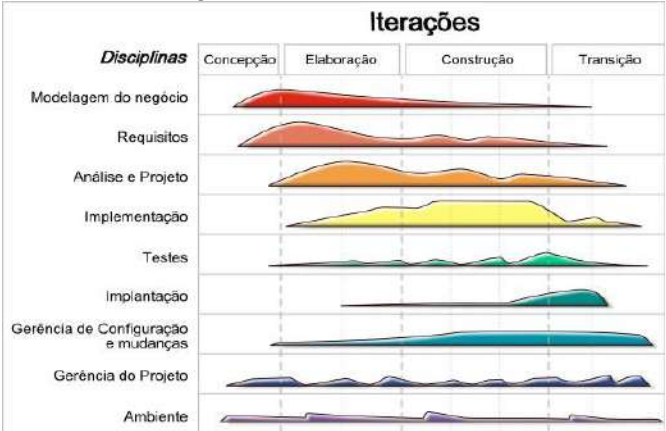
Fonte: Próprio Autor

1.8 Processo RUP

De todos os processos criados o *Rational Unified Process* (RUP), sem dúvida é um dos mais famosos. Esse modelo, criado nos "primórdios" da UML, trás todas as atividades, os papéis, os artefatos, e os fluxos de trabalhos, que se deve ter em um desenvolvimento pesado de *software*. Isto é, ele prevê praticamente tudo que se deve fazer.

E se tornou uma referência de mercado. Sendo "vendido" como um framework, que deve ser adaptado para a realidade do projeto em desenvolvimento. Bom, é aí que começam os problemas. Para projetos de médio e grande porte - excelente. Se existem prazo e custo aceitáveis para a contratação de mão de obra qualificada, e necessidade de controle de toda a produção. Esse é o modelo. Mas, mesmo ele sendo declarado pela seu fabricante Rational, hoje IBM, como "customizável" para projetos de pequeno porte.

Figura 1.10 Processo RUP



Fonte: Internet

O conhecimento desse processo, a experiência do "customizador" e as restrições orçamentárias e de prazo são complicadíssimos para sua adoção.

Mas, por um período de tempo, esse foi o processo utilizado para qualquer empresa de TIC que se deseja construir algo com qualidade. Até que grupos de pensadores começaram a discutir se o uso desses tipos de processos, se justificavam para qualquer tipo de projeto de *software*. Nesse ponto entraremos na nossa próxima aula - as metodologias ágeis, já que os processos que passaram a ser chamados de pesados receberam a denominação de metodologias tradicionais ou mesmo pesadas.

1.9 Processo Scrum

Scrum é um processo ágil que tem como objetivo focar na entrega dos requisitos de maior valor agregado ao cliente no menor intervalo de tempo possível. Permitindo assim que, adequações sejam realizadas no *software* de maneira rápida e contínua, fazendo com que o produto final esteja em conformidade com as necessidades do referido cliente.

Esses intervalos de entregas devem ser de duas a quatro semanas, para que as necessidades do negócio com as suas prioridades sejam entendidas, desenvolvidas, testadas, entregues e usadas pelo cliente.

Figura 1.11 Processo Scrum



Fonte: Internet

Para isso as equipes devem possuir uma auto-organização para poder definir a melhor maneira de realizar as entregas dos requisitos de maior prioridade. Nesses intervalos de duas a quatro semanas todos podem ver o real *software* em produção, decidindo se o mesmo deve ser liberado ou não e isso é denominado de *Sprint*. Resumindo as características dos Scrum são: Equipes que se auto-organizam; O produto evolui em uma série de *Sprints* mensais; Os requisitos são listados em um *Product Backlog*; Não há uma prática específica de engenharia de *software*, pois o Scrum pode ser adequar a todas; Usa regras genéricas na criação de um ambiente ágil para a entrega de projetos; e é uma metodologia ágil.

1.10 Metodologias-Modelos-Processos-Técnicas

Agile Software Development (ASD). Disp.: < http://pt.wikipedia.org/wiki/Agile_Software_Development >
Cleanroom. Disponível em < <http://pt.wikipedia.org/wiki/Cleanroom> >
CRUD. Disponível em < http://pt.wikipedia.org/wiki/Matriz_CRUD >
Crystal. Disponível em < http://en.wikipedia.org/wiki/Crystal_Clear_%28software_development%29 >
Domain-Specific Modelling (DSM). Disponível em < <http://goo.gl/sU1Jfs> >
Driven Design de Domínio (DDD). Disponível em < <http://goo.gl/wDBnjK> >
Dynamic Systems Development Method (DSDM). Disponível em < <http://pt.wikipedia.org/wiki/DSDM> >
eXtreme Programming (XP). Disponível em < <http://goo.gl/691P2> >
Feature Driven Development (FDD). Disp.: < http://pt.wikipedia.org/wiki/Feature_Driven_Development >
IBM Rational Unified Process (IRUP). Disp.: < http://pt.wikipedia.org/wiki/Rational_Unified_Process >
Microsoft Solution Framework (MSF). Disp.: < http://pt.wikipedia.org/wiki/Microsoft_Solution_Framework >
Model Driven Engineering (MDE). Disponível em < <http://goo.gl/iDg3RW> >
Model-Driven Architecture (MDA). Disponível em < <http://goo.gl/bgKPxp> >
Modelo V. Disponível em < http://pt.wikipedia.org/wiki/Modelo_V >
PMBOK. Disponível em < <http://pt.wikipedia.org/wiki/PMBOK> >
Pragmatic Programmer. Disponível em < http://en.wikipedia.org/wiki/The_Pragmatic_Programmer >
PRAXIS. Disponível em < <http://goo.gl/aq09tD> >
Rapid Application Development (RAD). Disp.: < http://pt.wikipedia.org/wiki/Rapid_Application_Development >
Scrum. Disponível em < <http://pt.wikipedia.org/wiki/Scrum> >
Service-Oriented Architecture (SOA). Disponível em < <http://goo.gl/ESXi3U> >
SWEBOK. Disponível em < http://pt.wikipedia.org/wiki/Software_Engineering_Body_of_Knowledge >
Test Driven Development (TDD). Disponível em < <http://www.improveit.com.br/xp/praticas/tdd> >

1.11 Ferramentas

Astah: modelagem UML. Disponível em < <http://astah.change-vision.com/en/index.html> >
Project-open: gerenciador de projetos. Disponível em < <http://www.project-open.org/> >
StarUML: modelagem UML. Disponível em < <http://staruml.sourceforge.net/en/> >

1.12 Sugestões de Leitura

Engenharia de Software e Sistemas de Informação. Disponível em < <http://goo.gl/N9f446> >
Gerenciando Projetos de Desenv. Software com PMI, RUP e UML. Disp.: < <http://goo.gl/GcXUe4> >
Guia prático de engenharia de software. Disponível em < <http://goo.gl/ppe5ff> >

1.13 Referências Web

Análise de Domínio wiki. Disp.: < http://pt.wikipedia.org/wiki/An%C3%A1lise_de_Dom%C3%ADnio >
Análise Essencial wiki. Disponível em < http://pt.wikipedia.org/wiki/An%C3%A1lise_Essencial >
Análise Estruturada wiki. Disponível em < http://pt.wikipedia.org/wiki/An%C3%A1lise_Estruturada >
Análise Orientada a Objetos wiki. Disponível em < <http://goo.gl/N0Ktwf> >
BROD Dicas. Disponível em < <http://www.dicas-l.com.br/brod> >
BROD. Disponível em < <http://www.brod.com.br/> >
ControlChaos. Disponível em < <http://www.controlchaos.com/> >
Engenharia de Software wiki. Disponível em < http://pt.wikipedia.org/wiki/Engenharia_de_software >
IBM RUP. Disponível em < <http://www-01.ibm.com/software/awdtools/rup/> >
Iterativo e Incremental wiki. Disp.: < http://pt.wikipedia.org/wiki/Desenvolvimento_iterativo_e_incremental >
Metodologias de Software wiki. Disp.: < http://pt.wikipedia.org/wiki/Metodologia_%28engenharia_de_software%29 >
Modelo Espiral wiki. Disponível em < http://pt.wikipedia.org/wiki/Modelo_em_espiral >
Modelos Ágeis wiki. Disp.: < http://pt.wikipedia.org/wiki/Desenvolvimento_%C3%A1gil_de_software >
Modelos Ciclo de Vida wiki. Disponível em < http://pt.wikipedia.org/wiki/Modelos_ciclo_de_vida >
RUP. Disp.: < [http://www.wthreex.com/rup/portugues/index.htm%20\(Excelente%20site%20sobre%20RUP\)](http://www.wthreex.com/rup/portugues/index.htm%20(Excelente%20site%20sobre%20RUP)) >
Scrum Alliance. Disponível em < <http://www.scrumalliance.org/> >
Scrum. Disponível em < <http://www.mountangoatsoftware.com/scrum> >
SEI. Disponível em < <http://www.sei.cmu.edu/> >
SWEBOK. Disponível em < <http://www.computer.org/portal/web/swebok> >

1.14 Referências Bibliográficas

PRESSMAN, Roger. Engenharia de Software: uma abordagem profissional. 7ªed. Porto Alegre: Bookman, 2011.
RUMBAUGH, James; JACOBSON, Ivar; BOOCH, Grady. UML: Guia do Usuário. 1ªed. RJ: Campus, 2006.

SOMMERVILLE, Ian. Engenharia de Software. 8ªed. Pearson, 2007.

2 Engenharia de Testes

O objetivo dessa disciplina é fazer o entendimento do que é bug e mostrar quais processos foram desenvolvidos para encontrá-los. O que é um *bug*? Várias definições existem em cada empresa: Um *bug* é: um defeito falta, problema, incidente, anomalia, CR, etc. Empresas usualmente perdem um bom tempo discutindo isso, mas normalmente a conclusão é que um *bug* é apenas um *bug*. Uma definição mais formal de *bug* pode ser feita se pensando quando um *bug* ocorre (Patton, 2005).

Um *bug* ocorre quando uma ou mais das opções abaixo for verdadeira:

- O *software* NÃO faz algo que a especificação diz que ele deveria fazer;
- O *software* FAZ algo que a especificação diz que ele NÃO deveria fazer;
- O *software* faz algo que a especificação não menciona;
- O *software* NÃO faz algo que a especificação NÃO menciona, mas deveria mencionar; e
- O *software* é difícil de usar, entender, ou na visão do testador pode ser visto pelo usuário final como não estando correto.

Alguns *bugs* clássicos:

- *Bug* do ano 2000 - Computadores não estavam preparados para o ano 2000;
- Sistema de Telefones de AT&T nos EUA - Toda a rede caiu por causa de um switch case errado; e
- Sistema de Foguetes da NASA e Agência Européia - O foguete explodiu no lançamento, pois os códigos feitos na NASA estavam no sistema métrico inglês e o da agência Européia no sistema internacional.

Qual o custo de um *bug*? O custo de um *bug* está diretamente associado à fase do ciclo de desenvolvimento em que o *bug* é encontrado. Um *bug* encontrado durante a especificação pode custar 1 dólar. O mesmo *bug* encontrado no release pode custar 100 vezes mais.

Por que *bugs* ocorrem? Usualmente a razão de um *bug* ocorrer, depende do tamanho e do tipo do sistema. Problemas de especificação é a principal causa de *bugs*, são mais de 50%. Em Sistemas Pequenos, problemas no código é a principal causa de *bugs*, mais de 75%. Erros durante a construção (*low level design* + *code* + *unit tests* + integração) correspondem por pelo menos 35% dos erros.

Quanto maior o projeto menor o número de erros na construção é maior o número na especificação e quanto maior o *software*, maior o número de erros na arquitetura e design de alto nível.

2.1 Tipos de *Bug*

Vários estudos buscaram definir tipos de *bug* e suas porcentagens de ocorrência. Nenhum dos estudos foi muito conclusivo, mas vários pontos importantes foram levantados. O escopo dos erros é limitado: 85% dos erros podem ser corrigidos modificando apenas 1 classe. Maioria dos erros não está no código, mas no domínio da aplicação e não é problema do código em si. Vários erros são simples erros de digitação: 35% dos erros de código são erros de digitação.

Maiorias dos erros são simples de corrigir, 85% dos erros podem ser corrigidos em menos de 1 hora. Apenas 1% dos erros leva mais que alguns dias para corrigir, 80% do esforço de manutenção é gasto em apenas 20% dos erros. Os erros são concentrados em partes específicas do código, 80% dos erros está em apenas 20% do código, 50% dos erros podem ser encontrados em 5% do código, baseado em Rios (2013) *apud* Capers Jones.

Os testes que buscam identificar quais são essas partes específicas do código podem reduzir em muito o custo de manutenção. Algumas vezes é melhor reescrever as rotinas nos quais os erros estão concentrados.

Podemos classificar o encontrar de *bugs* como:

- ERRO ocorre devido à ação humana, que resulta em um *software* com defeitos, já que um erro pode gerar falhas em um sistema;
- DEFEITO ocorrer devido a problemas de informações, de dados, de instruções incorretas, dentre outros fatores, gerando falha no funcionamento do *software*; e
- FALHA ocorre quando um programa não se comporta conforme o esperado ou apresenta resultados não consistentes.

2.2 Ciclo de Vida de um *Bug*

O ciclo de vida de um *bug* determina os vários estágios pelos quais o *bug* passa durante o desenvolvimento do *software*. Este ciclo de vida precisa ser definido nas organizações. O processo de controle de mudanças é responsável e pode definir o ciclo de vida. O ciclo de vida pode ser bem simples: Aberto: *bug* identificado no teste; Resolvido: *bug* resolvido; e Fechado: solução verificada. O ciclo de vida também pode ser bem complexo: Aberto; Resolvido como não corrigir; Resolvido como corrigido; e Fechado como corrigido. Algumas organizações adotam para o *bug* um ciclo de vida similar ao de desenvolvimento

de um *software*: Inicialdo: *bug* descrito; Análise: *bug* analisado e especificado; Implementação: *bug* implementado; Testes: *bug* testado; e Fechado: *bug* fechado de o teste for correto.

2.3 Testador

O testador está diretamente associado à definição de teste de *software*. O testador encontra o *bug*; Busca encontrá-lo o mais cedo possível; Garante que o *bug* está corrigido; Um *bug* corrigido não necessariamente implementa em mudança no código; Correção de um manual; Correção de treinamento; e o testador deve ser responsável por verificar realmente se o *bug* foi corrigido.

Por que o testador é necessário? Os testes feitos por desenvolvedores tendem a verificar apenas o “caminho feliz”; Normalmente são otimistas e não são sofisticados; Organizações iniciais têm usualmente 5 testes de “caminho feliz” para cada teste de “caminho alternativo”; Organizações maduras usualmente tem o oposto, 5 de caminho alternativo para cada um de caminho feliz; Desenvolvedores normalmente só conseguem enxergar 50-60% dos casos de teste do seu código; O Testador tem um perfil diferente do desenvolvedor: são exploradores; Gostam de encontrar problemas; Criativos no uso do *software*; e possuem visão das diferentes situações em que o *software* pode ser usado.

Por que testes precisam se realizados? Existem várias técnicas de encontrar *bugs*; Nenhuma técnica consegue garantir que todos os *bugs* sejam encontrados; Teste de *software* é uma das possíveis técnicas que podem ser aplicadas. Nenhuma técnica consegue encontrar mais de 85% dos *bugs*; Testes unitários + testes de componente + testes de sistemas atingem no máximo 60% dos *bugs*; Testes precisam ser realizados, mas não são suficientes para garantir que todos os *bugs* sejam encontrados; O ideal é combinar diferentes técnicas para garantir que a maioria dos *bugs* seja encontrada o mais cedo possível.

Algumas organizações chegam a atingir 95% de *bugs* encontrados combinando diferentes técnicas. Vários estudos comparam eficiência de técnicas de encontrar *bugs*. Um grupo de programadores com média de 11 anos de experiência avaliou que um código com 15 erros conhecidos e foram usadas 3 técnicas para correção: Testes contra a especificação; Execução de testes unitários; e de Sistemas. O resultado é que em média apenas 5 erros foram encontrados.

2.4 Testes versus Inspeções

Inspeções dos artefatos buscam validar os artefatos antes que a próxima fase do desenvolvimento seja iniciada; Vários estudos indicam que o custo de encontrar um erro com inspeção é menor que o custo de encontrar com testes

(Kaplan, 1995) A média é de 3,5 horas de esforço para encontrar um erro com inspeção e de 15-25 horas de esforço para encontrar um erro com testes, isso não significa que testes não são necessários e sim que outras técnicas também podem ser aplicadas.

2.5 Técnicas de Encontrar *Bugs*

Seguem alguns exemplos de técnicas para encontrar *bugs*.

Inspeção de *Software*:

- Processo formal de verificação do *software*;
- Pode ser aplicado para praticamente todos os artefatos gerados durante o ciclo de desenvolvimento;
- Tem o custo mais alto de aplicar, mas também é o que produz o melhor resultado;
- Combinação de inspeções na especificação, design e código podem encontrar até 85% dos *bugs* de um *software*; e
- Como o processo é formal, ele é fácil de mensurar e de acompanhar.

Prototipação:

- Ajuda no levantamento e definição dos requisitos;
- Dá uma boa visão para o cliente do que vai ser o *software* final; e
- Reduz bastante as dúvidas de usabilidade e de como as funções devem ser implementadas.

Programação por pares:

- Técnica definida em XP (Extreme Programming);
- 2 desenvolvedores por computador;
- 1 programador digita;
- 1 programador procura por erros;
- Possui várias vantagens;
- Dissemina a cultura da empresa;
- Pode chegar a encontrar a mesma quantidade de problemas da inspeção; e
- É mais aplicável para a fase de codificação.

2.6 Modelos

As atividades de testes são inseridas no contexto das atividades de engenharia de *software*. Todo *software* segue um ciclo de vida. Existem vários modelos de ciclo de vida que podem ser adotados durante o desenvolvimento. O modelo de

ciclo de vida usualmente direciona: Quais testes serão rodados; e Quando os testes serão rodados.

2.6.1 Modelo Cascata

Fases sequências que produzem o *software*; Ao final de cada fase uma revisão interna deve validar se é possível passar a fase seguinte; A princípio só se deve passar a fase seguinte com a anterior finalizada; e Foco principal é garantir que a especificação é bem feita antes de iniciar as fases seguintes.

Vantagens:

- O modelo em cascata é ideal para *software* bem conhecidos;
- Sem grandes surpresas durante a especificação;
- Tecnologia bem conhecida das equipes;
- A fase de testes é bem natural e é executada após toda a codificação; e
- A especificação normalmente é bem feita e detalhada.

Desvantagens:

- Os testes só são executados no final; e
- O custo de corrigir um problema é mais alto.

2.6.2 Modelo V

Variação do modelo em cascata; Associa diferentes tipos de teste de acordo com a fase do ciclo de vida; Testes de aceitação usualmente estão associados aos requisitos; Testes de Integração usualmente estão associados ao design; Testes de componente usualmente estão associados ao código; O modelo V define diferentes estágios de testes para serem executados; Pode ser visto como uma extensão do modelo em cascata. Existe a ênfase na especificação; Cada fase precisa ser finalizada antes do início da fase seguinte; e Também é ideal para *software* com requisitos conhecidos.

Vantagens:

- Identificar os diferentes estágios de teste que validam aspectos do ciclo de vida do *software*;
- Permite um melhor planejamento dos testes que precisam ser executados; e
- Divide os testes em diferentes focos.

Desvantagem:

- Os testes ainda são rodados apenas após o código estar pronto.

2.6.3 Modelo Incremental

Desenvolvimento é dividido em várias iterações; Cada interação pode ser vista como um pequeno modelo cascata; Cada interação passa por todas as fases do ciclo; Em maior ou menor escala requisitos são adicionados ao *software* em cada interação; O modelo incremental busca usar as vantagens do cascata; Foca em ter uma boa especificação, mas sendo realista; Nem toda a especificação pode estar pronta no início do desenvolvimento; As iterações podem ser repetidas até que o *software* esteja estável; Ao final cada interação é necessário determinar os objetivos da próxima interação; Avaliar os riscos; Avaliar os testes; Avaliar os requisitos que já estão implementados; e Planejar a interação seguinte.

Vantagens:

- Os testes são rodados várias vezes ao final de cada interação;
- Os *bugs* podem ser antecipados; e
- Os testadores usualmente devem ser envolvidos no projeto desde o início.

Desvantagens:

- Algumas vezes não fica claro que testes devem ser rodados ao final de cada interação;
- O esforço de planejamento de testes é maior, mas esse esforço normalmente é compensado;
- É necessária uma equipe de testes mais experiente para planejar e executar.

2.6.4 Modelo Codificar e Testar

Parte de uma especificação informal; Ênfase em codificação; Não existe ênfase em outras fases do desenvolvimento; Como a especificação não é clara os testes também não são, mas os testes são executados várias vezes; e é ideal para projetos pequenos e com equipe pequena.

Vantagens:

- Vários ciclos de testes; e
- Custo de correção não é alto.

Desvantagem:

- Planejamento e qualidade dos testes normalmente não são bons.

2.6.5 Modelos Ágeis - Test Driven Development (TDD)

Modelo de Desenvolvimento Orientado a Testes - Muito parecido com o codificar e testar; Foco em fechar várias versões do *software*; Integração contínua; Especificação de requisitos informal, mas o cliente final deve ficar próximo à equipe de desenvolvimento para validar os requisitos; Os testes são mais formais; Devem ser automatizados; Garantir que sempre serão rodados; São executados a cada integração; e Cliente pode participar da validação.

É uma das principais técnicas associadas a modelos ágeis; A ideia é que os testes sejam desenvolvidos antes do código - *Test First Design* (TFD).

Os passos para usar TDD são:

- Escrever um caso de teste;
- Executar um caso de teste;
- Falhando, implementa o código para passar no teste;
- Passando, continua o desenvolvimento e escreve um novo caso de testes; e
- Toda a fase de implementação de um modelo ágil pode ser feita com TDD.

Vantagens:

- Automatização;
- Sai à figura do testador e entra a do desenvolvedor de testes;

Desvantagem:

- É necessário o uso de uma ferramenta para automatização.

2.7 Conceitos em Testes

Seguem alguns conceitos básicos de testes:

- Unidade, Componente e Integração;
- *Test Driver* e *Stub* de Testes;
- Processo de Testes;
- Estágios de Teste;
- Tipos de Teste;
- Testes de Regressão;
- Abordagem de Teste;
- Procedimento de Teste;
- Resultado de Testes;

- Ferramenta de Testes; e
- Testes Positivos e Testes Negativos.

2.7.1 Unidade, Componente e Integração

Um *software* em geral é quebrado em diferentes componentes ou módulos e cada componente é quebrado em unidades (subcomponentes). Estas unidades precisam ser integradas em componentes e os componentes precisam ser integrados no *software* como um todo. Testes precisam ser executados na unidade para garantir que esta funciona separadamente. No componente para garantir que quando as unidades foram integradas, eles continuam funcionando. No sistema como um todo para garantir que os componentes integrados funcionam corretamente.

O escopo de cada um desses testes é bem diferente.

2.7.2 Test Driver

É um componente de *software* que simula as entradas reais do *software*. Por exemplo, um *software* de calculadora recebe números e operações. Um *test driver* para este *software* pode entrar um grande combinação diferente de números e combinações para validar a calculadora.

2.7.3 Stub de Test

É utilizado para testar o *software* de baixo para cima. O *stub* simula resultados que são retornados por um determinado componente que o *software* depende.

2.7.4 Processos de Testes

São sequências de ações, operações que são executadas com o objetivo de encontrar problemas no *software* e encontrá-los o mais cedo possível. Aumentando a percepção de qualidade geral do *software* e garantindo que o usuário final tenha um *software* que atende as suas necessidades.

De forma geral, o processo de teste pode ser separado em 4 grandes ações:

- Planejar - entender o que precisa ser testado e definir como;
- Preparar - selecionar e montar todo o ambiente para execução;
- Executar - executar os testes e coletar o resultado; e
- Avaliar - verificar os resultados e métricas para melhorar os testes.

2.7.5 Estágio de Teste

É usualmente aplicado ao teste que está associado a uma determinada fase do ciclo de desenvolvimento, que pode ser:

- Testes Unitários;

- Testes de Componente;
- Testes de Integração;
- Testes de Sistema; e
- Testes de Aceitação.

2.7.6 Testes Unitários

Executados pelo desenvolvedor para validar a unidade que está sendo implementada. Testes unitários são testes isolados executados sobre uma rotina, classe ou pequena parte de um programa. Esses testes são executados pelo próprio desenvolvedor para garantir que a sua unidade pode ser integrada com o resto do *software*.

São dois os aspectos fundamentais: Os testes são isolados e os testes são implementados e executados pelo próprio desenvolvedor. Testes isolados - *Stubs*, devem ser implementados para suprir alguma parte da funcionalidade ou gerar valores errados para validar a unidade. Os componentes externos a unidade em testes devem ser simulados. Testes executados pelo desenvolvedor - Os testes são normalmente rodados no ambiente de desenvolvimento. Os casos de testes focam apenas na unidade em testes sem se preocupar com a interação com outras unidades.

Testes unitários podem ter a abordagem caixa branca ou caixa preta. Testes caixa branca são bem importantes, pois ajudam a validar todos os fluxos de execução. A formalização de casos de testes unitário é feita em algumas organizações. Principal dificuldade é dá manutenção nos casos de testes . Na maioria das vezes os casos de teste são jogados fora após os testes (especialmente os caixa branca). O resultado do testes unitários algumas vezes também é formalizado. Mas a sua análise não tem tanto sentido em algumas situações.

Automação de testes unitários - É uma das principais formas de garantir que os casos de teste não sejam “jogados fora”. Existem várias ferramentas que focam na automação dos casos de testes: Gerar e/ou implementar os scripts de testes; Executar os scripts de testes; Coletar os resultados da execução; Estas ferramentas rodam os testes sem intervenção do desenvolvedor. Principal vantagem: Facilita a manutenção dos testes e os Os Testes podem ser rodados automaticamente para garantir que as unidades não quebraram. Principal problema: O custo de especificar e implementar os scripts é alto. É necessário avaliar o custo / benefício para cada *software* (ou para cada parte do *software*).

Processo de desenvolvimento orientado a testes unitários - Atualmente existem processos de desenvolvimento orientados a testes unitários - TDD (*Test Driven Development*). O princípio fundamental é desenvolver o teste antes do código. Principal vantagem: Casos de teste estão sempre atualizados; O código

é feito para passar nos testes; O requisito é bem entendido antes de iniciar a implementação; Os testes são pensados com muito cuidado; e Novos casos de testes podem ser facilmente adicionados. Principal desvantagem: Custo do desenvolvimento dos testes;

É necessário avaliar benefício de acordo com o projeto; A equipe precisa ter maturidade para entender a importância dos testes e saber implementar os casos de teste.

Passos para identificar casos de testes unitários chamado de caixa preta:

- Avaliar a interface da rotina a testar;
- Avaliar os parâmetros e seus possíveis valores;
- Avaliar todos os possíveis retornos de acordo com cada possível parâmetros;
- Encontrar os limites dos parâmetros que vão fazer com que o retorno mude;
- Avaliar todos os possíveis erros que podem ser gerados; e
- Encontrar as combinações de parâmetros que devem gerar erro.

Passos para identificar casos de testes unitários caixa branca:

- Avaliar todos os fluxos de execução da rotina;
- Avaliar as funções que são usadas internamente e seus retornos;
- Verificar o comportamento da rotina caso as funções internas retornem valores inesperados;
- Avaliar se todos os possíveis fluxos de execução são tratados; e
- Avaliar se todos os fluxos codificados realmente podem ser atingidos.

2.7.7 Testes de Componente

Executado pelo testador para validar um componente específico do *software*.

2.7.8 Testes de Integração

Executado pelo testador para garantir que vários componentes funcionam corretamente juntos.

2.7.9 Testes de Sistema

Executados pelo testador para garantir que as funcionalidades do *software* estão de acordo com a especificação.

2.7.10 Testes de Aceitação

Executados pelo usuário para garantir que o *software* faz o que foi inicialmente requisitos pelo usuário.

2.7.11 Tipos de Testes

Frequentemente são confundidos com os Estágios de Teste e identificam os diferentes testes que podem ser realizados em cada um dos estágios. Cada tipo pode estar associado a um estágio específico ou pode ser realizado a partir dos testes de componente. Tipos também estão associados com cada domínio de aplicação. Os tipos são:

- Testes de Carga;
- Testes de Configuração;
- Testes de Compatibilidade;
- Testes de Performance;
- Testes de Documentação;
- Testes de Internacionalização;
- Testes de Usabilidade; e
- Testes de Jogabilidade (só aplicável a jogos).

2.7.12 Testes de Carga

Foco em garantir um número específico de transações no banco de dados (alto volume de dados e alto número de transações). É aplicável a partir dos testes de componente.

2.7.13 Testes de Configuração

Verifica se o *software* funciona com diferentes configurações de *hardware* e *software*. Está associado aos requisitos não-funcionais / restrições do *software*. Usualmente é aplicável a partir dos testes de integração.

2.7.14 Testes de Compatibilidade

Verifica se um determinado *software* está aderente a um padrão que ele deveria implementar. Usualmente é executado durante o teste de sistema.

2.7.15 Testes de Performance

Garantir que a velocidade de execução do sistema é adequada (tempo para carregar uma página web por exemplo). Associado aos requisitos não funcionais. Pode ser executado a partir dos testes de integração.

2.7.16 Testes de Documentação

Garantir que a documentação do *software* está de acordo com o que foi realmente implementado. Tem mais sentido ser executado durante os testes de sistema.

2.7.17 Testes de Internacionalização

Garantir que o *software* funciona em diferentes línguas e com configurações dependentes de país (moeda, *timezone*, etc.)

2.7.18 Testes de Usabilidade

Verifica se as funções do sistema realmente são “fáceis” de serem realizadas. Deve ser executado durante os testes de sistema.

2.7.19 Testes de Regressão

Execução dos testes que já foram rodados. Considere o seguinte cenário - Durante a execução dos testes de sistema de um *software* encontra-se um erro após a execução de 70% dos testes. Este erro é corrigido e uma nova versão é gerada. Que testes devem ser executados após isso? Os 30% restantes. E também se pode regredir e executar todos os outros testes que já foram executados. Um teste de regressão completo significa que TODOS os testes serão rodados novamente.

2.7.20 Abordagens de Testes

Testes Caixa Preta são aqueles nos quais o testador não vê o “interior” do objeto que está sendo testado. Usualmente executados por um testador que é diferente do desenvolvedor. Avalia todas as diferentes entradas e suas respectivas saídas para garantir que o *software* se comporta como especificado. Associado a estágios de teste como sistema ou componente. Testes Caixa Branca são aqueles nos quais o testador sabe como funciona o “interior” do objeto sendo testado. Usualmente são executados pelo próprio desenvolvedor. São fortemente associados aos testes unitários. Avalia todos os caminhos lógicos do código para garantir que todos os fluxos são executados corretamente. Busca encontrar *bugs*, mas *bugs* no código e não *bugs* na especificação.

Observação das abordagens de testes. A grande maioria do *software* realiza apenas testes caixa preta. Quando testes caixa branca são executados, eles não são formalizados. Não existe procedimento bem definido. Não existe resultado de testes documentado. Caso um procedimento de testes caixa branca seja escrito, existe grande chance dele ser descartado após a execução dos testes.

2.7.21 Caso de Testes

O procedimento de testes está diretamente relacionado ao conceito de caso de teste. Cada caso de teste está associado a um diferente “cenário” a ser testado. Para cada requisito existem diferentes cenários a serem testados. Para validar um caso de testes é necessário:

- Definir o ambiente no qual o teste será realizado;
- Definir a entrada deste caso de teste;
- Definir a saída esperada para cada entrada;
- Definir os passos a serem realizados para executar os testes; e
- O conjunto de passos acima determina o que um procedimento de testes.

Quando cada caso de teste é executado, o seu resultado deve ser coletado. Existem diferentes abordagens para definir o resultado de um caso de teste específico. A mais comum define as seguintes opções:

- Passou - todos os passos do caso de testes foram executados com sucesso para todas as entradas;
- Falhou - nem todos os passos foram executados com sucesso para uma ou mais entradas; e
- Bloqueado - o teste não pode ser executado, pois o seu ambiente não pode ser configurado, ou pode alguma dependência externa.

2.7.22 Ferramentas para Testes

Ferramentas de apoio à execução dos testes; e Ferramentas de apoio ao planejamento e documentação de Testes. Ferramentas de Execução focam na automatização do procedimento de testes através de scripts de testes, na automatização da execução e na automatização do resultado de testes. Ferramentas de Planejamento focam no gerenciamento dos casos de teste e dos ciclos de execução de testes, na atribuição de recursos para execução dos testes e na coleta de métricas.

2.7.23 Testes Positivos

Servem para verificar se o caminho feliz é executado corretamente. Garantem que o *software* funciona minimamente como esperado. Garantem que o *software* não quebra para as suas operações principais. Não levam o *software* aos seus limites para identificar possíveis falhas.

2.7.24 Testes Negativos

Servem para verificar as exceções. Buscam realmente quebrar o *software*. Focam em identificar problemas. Garantem que o *software* funciona em diversas situações diferentes. Observação: Ambos os testes devem ser realizados

2.8 Sugestões de Leitura

Engenharia de Software. Disponível em < <http://goo.gl/p2OV92> >

Teste e Análise de *Software*: Processos, Princípios e Técnicas. Disponível em <
<http://goo.gl/UIKwTi> >

2.9 Referências Bibliográficas

PATTON, Ron. Software Testing. 2ªed.US: Sams Indianapolis, 2005.
PRESSMAN, Roger. Engenharia de Software: uma abordagem profissional.
7ªed. Porto Alegre: Bookman, 2011.
RIOS, Emerson. Teste de Software. 3ªed.Rio de Janeiro: Alta Books, 2013.
SOMMERVILLE, Ian. Engenharia de Software. 8ªed. Pearson, 2007.

3 Gestão de Projetos

Gestão de projetos é uma disciplina que fornece o conhecimento necessário para administrar os recursos utilizados na construção de um projeto de software. Administrar recursos pode ser realizado sem o entendimento de teorias administrativas, fazendo uso da intuição, porém nesses casos o sucesso está condicionado aos fatores sorte, experiência e afinidade. Contar com o fator sorte é um risco não aceitável. Experiência é importante, mas se você não a tem? Afinidade de forma geral, como ter equipes com sinergia, nem sempre é uma realidade. Assim, conhecer os conceitos e as ferramentas que podem auxiliar nessa árdua atividade é importante para se obter sucesso, no que podemos chamar "a arte de administrar". A forma utilizada para expor essa disciplina vai mesclar os conceitos com a operacionalização desses em uma ferramenta, aqui adotada o Project da Microsoft.

Antes de iniciarmos o estudo de gerenciamento de projetos com o uso da ferramenta MS-Project é importante ter alguns conceitos sobre essa disciplina. Começando com o que é um projeto? A definição mais simples e conhecida de projeto é "um conjunto de atividades que representam algo único e que possui início e fim".

Mas existem outros conceitos e níveis de administração de atividades, por exemplo, um conjunto de projetos interligados forma um programa, um conjunto de programas interligados forma um portfólio. Para exemplificar, temos um portfólio de governo que engloba vários programas, como: "Fome Zero"; "Escola para Todos"; "Minha Casa Minha Vida"; etc. Cada programa, a exemplo "Fome Zero", necessita de vários projetos para ser realizado, como: Identificação dos Pontos Necessitados, Captação de Recursos, Logística, etc. Cada projeto para ser concluído terá que realizar várias atividades únicas com prazo, recursos e custos planejados. Essas tarefas podem ser agrupadas em fases, subfases, etc. Em resumo um Portfólio é conjunto de programas; Programa é conjunto de projetos; Projeto é conjunto de fases; Fase é conjunto de atividades; e Atividade é conjunto de subatividades ou tarefa única a ser executada.

Com o passar dos anos várias instituições foram surgindo para criar e padronizar as teorias de administração de projetos, e algumas se consolidaram. Podemos citar o ISO, ITIL, COBIT, PRINCE2, SWEBOK, entre outras, mas a que vem se destacando em várias áreas é o PMI. Porém, independente o processo que define como fazer, quando fazer, quem fazer e onde fazer a gestão de atividades, todas elas servem para definir um corpo de conhecimentos sobre gestão, o que é fundamental, para se ter um local de acesso para se obter um compêndio de boas práticas de gerenciamento.

Esse estudo de gestão de projetos fará uso do corpo de conhecimento de gerenciamento de projetos, mais conhecido na sua sigla em inglês como PMBOK que significa *Project Management Body of Knowledge*. Abordando as suas 9 áreas que são integração, escopo, tempo, custos, qualidade, recursos humanos, comunicações, riscos e aquisições, e como utilizar o MS-Project da Microsoft como ferramenta para aplicar os conhecimentos administrado pelo instituto de gestão de projeto, também mais conhecido como *Project Management Institute (PMI)*.

3.1 História

Nos anos 50 foram criadas as primeiras ferramentas com a finalidade de auxiliar o planejamento e a execução de projetos, nas quais se destacaram o *Program Evaluation and Review Technique (PERT)* e o *Critical Path Method (CPM)*.

O PERT é uma técnica probabilística utilizada para indicar os consumos de recursos a serem utilizados em cada tarefa do projeto. O CPM é uma técnica determinística que atribui duração a cada tarefa determinando quais os recursos humanos e materiais necessários para concluí-las. De certa forma essas duas técnicas se completam, passando com o tempo a serem designadas como PERT/CPM passando-se a utilizar tanto as técnicas probabilísticas e determinísticas de uma e de outra, com o objetivo de identificar o caminho que consome mais tempo entre as atividades que devem ser executadas, sendo conhecido como caminho crítico. Assim, as atividades do caminho crítico são as que ocupam maior tempo de conclusão e se essas atividades atrasarem, todo o projeto atrasará.

3.2 Conceitos

Um projeto pode ser definido como uma sequência de atividades ou eventos com início e fim definidos, dirigidos por pessoas que se destinam a alcançar um dado objetivo dentro de parâmetros de custo, tempo, recursos e qualidade.

MS-Project é um dos mais modernos aplicativos voltados para o gerenciamento de projetos, no qual é possível planejar, especificar, implantar e acompanhar o desenvolvimento de qualquer tipo de projeto. As informações podem estar representadas graficamente ou através de relatórios customizados.

3.3 Portfólio

De forma simplista podemos dizer que um portfólio é um conjunto de programas de projetos ou de outra forma um grande projeto composto de projetos menores, todos interligados. Esse grande projeto (portfólio) tem como objetivo resolver conflitos do tipo: Muitos projetos que estão além da capacidade de

gerenciamento da organização; Projetos que não estão ligados aos objetivos estratégicos, gerando um desbalanceamento entre projetos de melhorias e projetos de inovação; Projetos de desenvolvimento em muito maior número que projetos de pesquisa; Projetos que não refletem os recursos mais importantes da organização; Entre outros tantos.

Para resolver os conflitos acima o PMI indica dois métodos de seleção de projetos de portfólio: Métodos de medição do benefício (abordagem comparativa) e Métodos de restrição otimizados (abordagem matemática).

A necessidade de se utilizar um gerenciamento de portfólio é assegurar que a coleção de projetos escolhidos e completos esteja alinhada com os objetivos da organização.

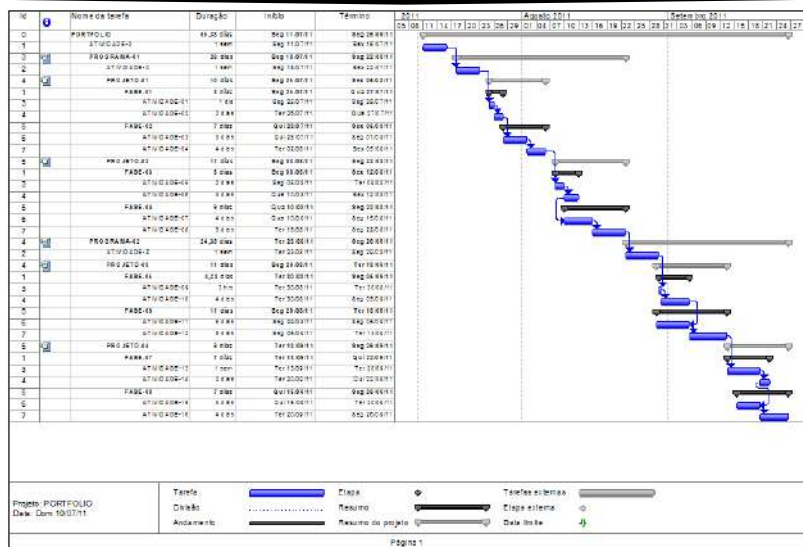
As boas práticas de gestão de portfólio pregadas pelo PMI são:

- Determinar uma estrutura viável para o projeto, que é capaz de atingir os objetivos da organização;
- Balancear o portfólio, para garantir a estrutura do projeto que balanceie melhor longos prazos versus curtos prazos; riscos versus recompensas; pesquisa versus desenvolvimento, etc.;
- Monitorar o planejamento e a execução dos projetos escolhidos;
- Analisar o desempenho do portfólio e maneiras para aprimorá-lo;
- Avaliar novas oportunidades confrontando com o portfólio atual e, comparativamente, levar em conta a capacidade de execução de projetos da organização; e
- Prover informação e recomendações para pessoas com poder de decisão em todos os níveis hierárquicos.

A figura que se segue é a impressão de um portfólio exemplo gerado pela ferramenta MS-Project que consiste da integração de 2 programas e cada programa da integração de 2 projetos.

Portfólio. Disp.: < <http://goo.gl/8aQFWX> >

Figura 3.1 Portfólio



Fonte: Próprio Autor

3.4 Programa

Também de forma simplista podemos dizer que um programa é um conjunto de projetos interligados. Mas uma definição mais acadêmica utilizada pelo PMI é que um programa é um grupo de projetos relacionados e gerenciados de modo coordenado para a obtenção de benefícios estratégicos e controle que não estariam disponíveis se eles fossem gerenciados individualmente.

Outra forma de entender um programa é que ele sempre conterá projetos, mas um projeto necessariamente não precisa pertencer a um programa. Os projetos dentro de um programa são relacionados através do resultado comum ou da capacidade coletiva.

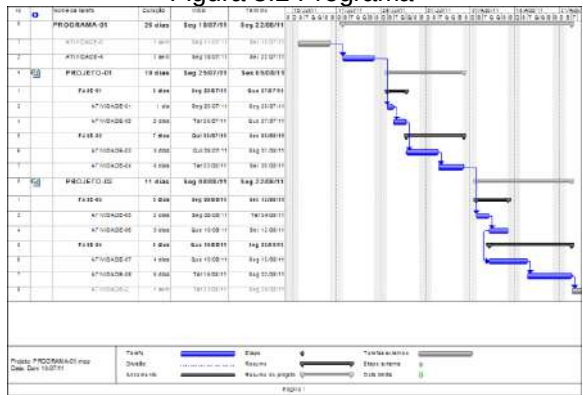
O gerenciamento de programas se concentra nas interdependências do projeto e auxilia a determinar a melhor abordagem para gerenciá-los. As ações relacionadas a essas interdependências podem incluir: Solução de restrições e/ou conflitos de recursos que possam afetar múltiplos projetos no sistema; Alinhamento da orientação estratégica/organizacional que afeta as metas e objetivos do projeto e do programa; e Solução de problemas e gerenciamento de mudanças em uma estrutura de governança compartilhada.

A figura que se segue é um exemplo de um programa gerado na ferramenta MS-Project que possui 2 projetos interligados.

Programa1. Disp.: < <http://goo.gl/mEpHVy> >

Programa2. Disp.: < <http://goo.gl/ei66DV> >

Figura 3.2 Programa



Fonte: Próprio Autor

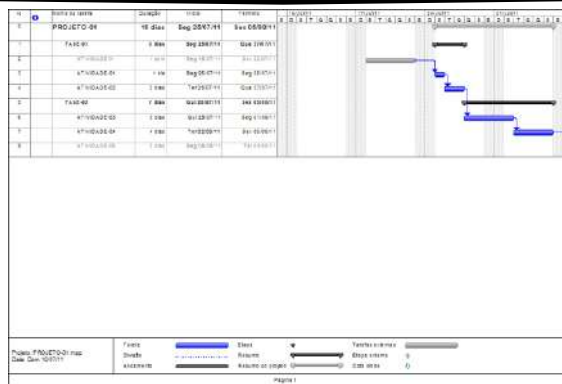
3.5 Projeto

A forma mais simples de definir um projeto, e usando a definição do PMI, é a administração de atividades fazendo uso de conhecimentos, habilidades e técnicas no seu planejamento, relacionando-as para atingir um conjunto de objetivos pré-definidos, num certo prazo, com certo custo e qualidade, através da mobilização de recursos técnicos e humanos.

A figura que se segue é um projeto gerado pela ferramenta MS-Project que consiste do planejamento de 2 fases, em que cada fase possui 2 atividades.

- Projeto1. Disponível: < <http://goo.gl/TfnwNs> >
- Projeto2. Disponível: < <http://goo.gl/36v0nd> >
- Projeto3. Disponível: < <http://goo.gl/O3et32> >
- Projeto4. Disponível: < <http://goo.gl/xl2A01> >

Figura 3.3 Projeto



Fonte: Próprio Autor

3.6 PMI

O *Project Management Institute* (PMI) é uma organização sem fins lucrativos que se destina a criar e manter boas práticas na gestão de portfólios, programas e projetos. Ela foi criada por 5 gerentes de projetos em 1969 na cidade da Filadélfia no estado da Pensilvânia.

Nos dias atuais, o PMI já possui mais de 200.000 associados e é considerado uma das principais organizações de gestão de projetos, com representações em 185 países.

Apesar de inicialmente o PMI ser orientado a projetos de *software*, hoje suas práticas são utilizadas nas mais diversas áreas como: aeroespacial, automotiva, negócios, construção, engenharia, serviços financeiros, farmacêutico, saúde, etc.

A seguir são exibidas as diversas certificações oferecidas pelo instituto:

- *Certified Associate in Project Management* (CAPM), que serve para demonstrar uma base comum do conhecimento e dos termos no campo da gerência de projeto. Ele requer 1500 horas do trabalho em uma equipe de projeto ou 23 horas/aula em gerência de projeto;
- *Project Management Professional* (PMP) contém curso de especialização e experiência, concordando em aderir a um código da conduta profissional, e aprovação para avaliar e medir objetivamente o conhecimento da gerência de projeto. Este certificado deve ser periodicamente renovado, caso contrário implica na perda da certificação; e
- *Program Management Professional* (PgMP), equivalente ao anterior, porém focando na certificação em programa.

O principal produto gerado pelo PMI é o guia chamado de *Project Management Body of Knowledge* (PMBOK) que é um compêndio de boas práticas gerenciais para as 9 áreas que a organização orienta na administração.

O Guia PMBOK serve para oferecer um vocabulário comum na gestão de portfólios, programas e projetos. Permitindo assim a discussão no mesmo nível entre os participantes, disseminando o conhecimento adquirido por todos. Possibilitando assim o intercâmbio eficiente de informações entre os profissionais de gerência de projetos.

O guia é baseado em processos e subprocessos para descrever de forma organizada as atividades de gestão de projetos que se relacionam e interagem. E isso é feito através de artefatos de entradas que podem ser documentos, planos, desenhos, etc.; de ferramentas e técnicas, por exemplo, o MS-Project; e de saídas, como documentos, produtos, etc. A versão 2004 do guia, cita 44 processos agrupados em cinco grupos e nove áreas de conhecimento, que definem o ciclo de vida e da organização de um projeto.

As áreas de conhecimentos de acordo com o PMBOK são:

- Gerenciamento de integração do projeto;
- Gerenciamento do escopo do projeto;
- Gerenciamento de tempo do projeto;
- Gerenciamento de custos do projeto;
- Gerenciamento da qualidade do projeto;
- Gerenciamento de recursos humanos;
- Gerenciamento das comunicações;
- Gerenciamento de riscos do projeto; e
- Gerenciamento de aquisições do projeto.

Os grupos de processos de gerenciamento de projetos são:

- Iniciação;
- Planejamento;
- Execução;
- Monitoramento e controle; e
- Encerramento.

3.7 PMP

O que é um PMP? É uma certificação que significa Project Management Professional ou traduzindo Profissional de Gerência de Projetos, e consiste em um documento emitido pelo PMI atestando os conhecimentos nas boas práticas

de gerenciamento de projetos, além da responsabilidade social e ética profissional ensinadas no PMBOK.

O maior objetivo dessa certificação é capacitar o profissional nas boas práticas de gestão e gerar uma rede de profissionais reconhecidos como excelentes e experientes na gerência de projetos na forma concebida pelo PMI.

Para obter essa certificação é necessário ser aprovado em exame de certificação PMP que é baseado em 200 perguntas de múltipla escolha, além de aderir ao *Code of Professional Conduct*, manter-se atualizado em relação às práticas gerenciais e possuir experiência na área de gestão de no mínimo a comprovação de 4.500h de atividades em projetos, possuir formação de nível superior e essas atividades obrigatoriamente têm que ter ocorrido no período compreendido entre os últimos 8 anos e se distribuir dentro deste período por no mínimo 36 meses ou a comprovação de no mínimo 7.500h de atividades em projetos com formação de nível médio, caso não tenha formação de nível superior e essas atividades obrigatoriamente têm que ter ocorrido no período compreendido entre os últimos 8 anos e se distribuir dentro deste período por no mínimo 36 meses.

3.8 Gerenciamento

Gestão de projetos ou gerenciamento de projetos ou ainda administração de projetos é a aplicação de conhecimentos, habilidades e técnicas na elaboração de atividades relacionadas no cumprimento de objetivos pré-definidos, no prazo, no custo e na qualidade planejados, fazendo uso de recursos humanos e materiais.

3.9 Planejamento

O planejamento do projeto na gestão de projetos é o processo que serve para quantificar o tempo e custo de um projeto, com o intuito de criar um plano de projeto para que o gestor possa utilizar como guia no acompanhamento do progresso dos recursos humanos.

Passos de um planejamento:

- Identificar o principal objetivo do projeto;
- Determinar as condições de finalização do projeto;
- Fazer um inventário das atividades a serem realizadas no projeto;
- Identificar os recursos necessários para execução do projeto;
- Definir as dependências entre tarefas;
- Determinar quais tarefas será impossível de estimar o prazo com precisão;
- Colocar as tarefas sem prazo fora do caminho crítico do projeto;

- Criar um cronograma do projeto; e
- Obter comprometimento da organização na execução do projeto.

3.10 Gerenciar

Ser um gerente de projeto é ser um profissional que raramente participa das atividades diretas do projeto, isto é, "botar a mão na massa", pois sua função é gerenciar o progresso através das variáveis prazos, custo e qualidade, chamada de tríade da gestão, para verificar os desvios nessas.

Assim, podemos dizer que o principal objetivo é evitar e tratar das falhas que venham acontecer e assim minimizá-las ou eliminá-las.

Um gerente de projetos é um maestro que deve entender as necessidades do cliente e planejar um projeto para atendê-lo da melhor forma possível, do começo ao fim, assegurando que tudo que foi planejado será cumprido. Esta tarefa pode ser dividida entre a gerencia de supervisão - o gerente de projetos e a gerencia de execução - o gerente operacional.

3.11 Processo

Um processo é uma série de ações que geram um resultado, nas quais se define quem faz o que se faz como se faz e quando se faz. Os processos se enquadram em duas categorias:

- Processos da gerência de projetos, que se relacionam com a descrição, a organização e a conclusão do trabalho do projeto; e
- Processos orientados ao produto, que se relacionam com a especificação e a criação do produto do projeto, sendo exclusivos a cada produto.

3.12 Grupos de Processos

De acordo com o PMBOK, os processos de gerenciamento de projetos podem ser organizados em 5 grupos:

- Processos de Iniciação, que autoriza o projeto ou a fase;
- Processos de Planejamento, que são os processos iterativos de definição e refinamento de objetivos e seleção dos melhores caminhos para atingir os objetivos;
- Processos de Execução, que determina a execução dos planos do projeto;

- Processos de Monitoramento e Controle, que permite a medição e monitoramento do desempenho do projeto, garantindo que os objetivos do projeto sejam alcançados através do monitoramento e medição regular do progresso, de modo que ações corretivas possam ser tomadas quando necessário; e
- Processos de Fechamento, que consiste da aceitação formal do projeto ou fase para a sua finalização.

3.13 Gerente de Projetos

O gerente de projetos precisa monitorar e comunicar o desempenho do projeto. Os resultados do trabalho que estiverem abaixo de um nível de desempenho aceitável precisam ser ajustados com ações corretivas para que o projeto volte a estar em conformidade com as linhas de base de custo, prazo e escopo. A comunicação do desempenho do projeto é um dos principais elementos para o gerenciamento de projetos bem sucedido.

3.14 Interações de Processo

Cada grupo de processo deve estar ligado pelas suas entradas e saídas, e essas ligações podem ser descritas como as seguintes interações:

- Entradas (Inputs), nos quais os documentos ou itens serão trabalhados pelo processo;
- Ferramentas e técnicas, que serão os mecanismos aplicados aos inputs para criar as saídas; e
- Saídas (Outputs), que são os documentos ou itens que serão o resultado final do processo.

Assim, a saída de um processo geralmente é a entrada para outro. Dentro de cada processo, as ferramentas e técnicas usadas num processo orientam e influenciam a sua saída. Uma saída com falhas pode comprometer a entrada de processos dependentes.

3.15 MS-Project

Microsoft Office Project Standard 2007 se destina a gerentes que não precisam realizar colaboração com outros na criação de projetos;

Microsoft Office Project Professional 2007 se destina a gerentes de projetos que trabalham possuem a necessidade de compartilhar agendas e recursos;

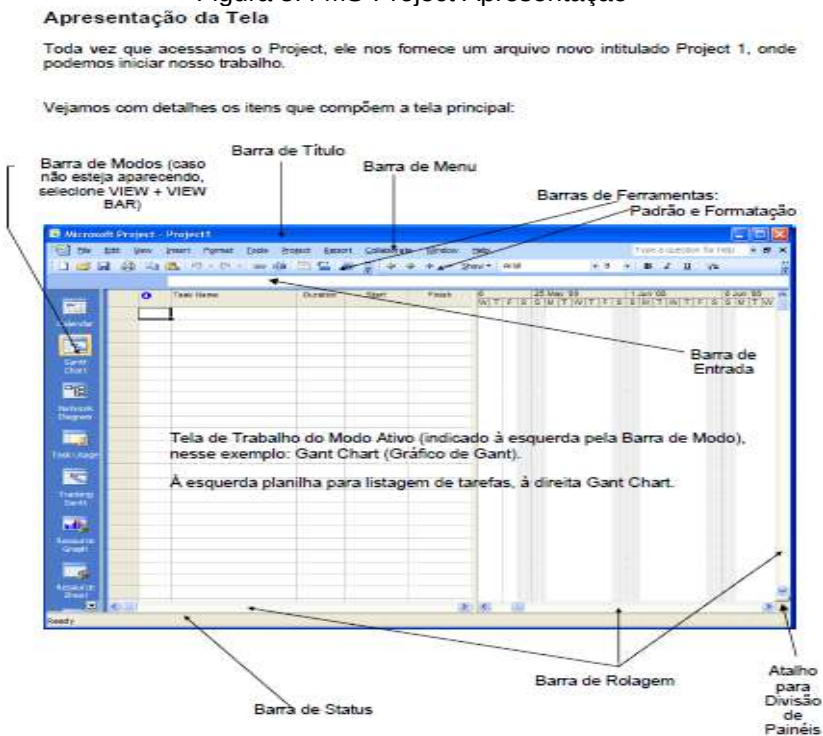
Microsoft Office Project Server 2007 se destina a gerentes que necessitam de acesso a um servidor para criar e compartilha em tempo real seus projetos;

Microsoft Office Project Web Access 2007 se destina a pessoas que não precisam de todas as funções de um cliente desktop como executivos, membros de equipe e administradores;

Microsoft Office Project Portfólio Server 2007 se destina ao gerenciamento de portfólios;

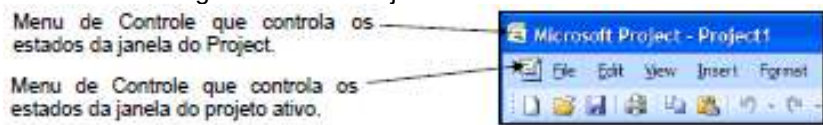
Microsoft Office Enterprise Project Management Solution se destina a usuários que necessitam de uma solução completa de gestão.

Figura 3.4 MS-Project Apresentação



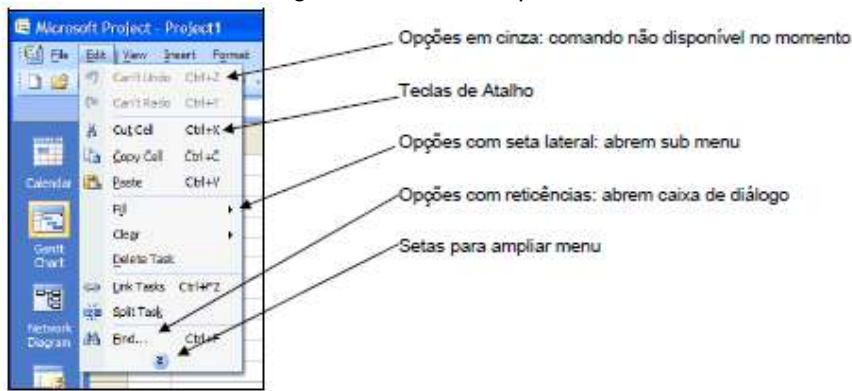
Fonte: Próprio Autor

Figura 3.5 MS-Project Menu de Controle



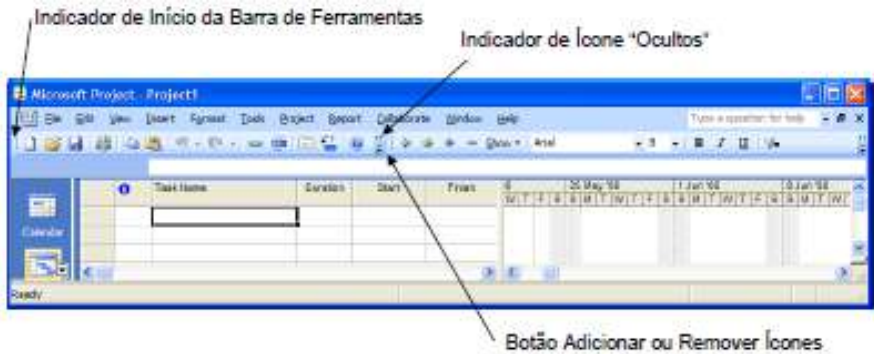
Fonte: Próprio Autor

Figura 3.6 Menu Suspenso



Fonte: Próprio Autor

Figura 3.7 MS-Project Barra de Ferramentas



Fonte: Próprio Autor

3.15.1 Gestão de Prazos

Serve para descrever os prazos para construção dos requisitos permitindo assim que o projeto tenha seu tempo controlado.

3.15.1.1 Artefatos

- Definição da atividade;
- Sequenciamento de atividades;
- Estimativa de recursos da atividade;
- Estimativa de duração da atividade;

- Desenvolvimento do cronograma; e
- Controle do cronograma.

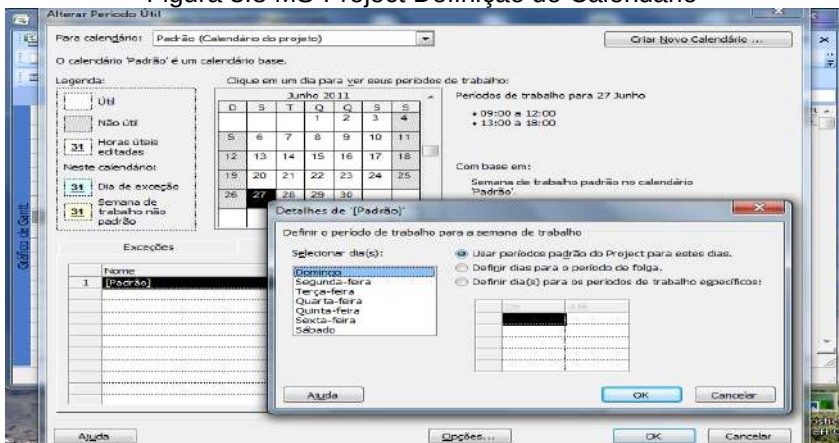
3.15.1.2 Importante

- Definir o calendário; e
- Montar o Cronograma.

3.15.1.3 Wiki's

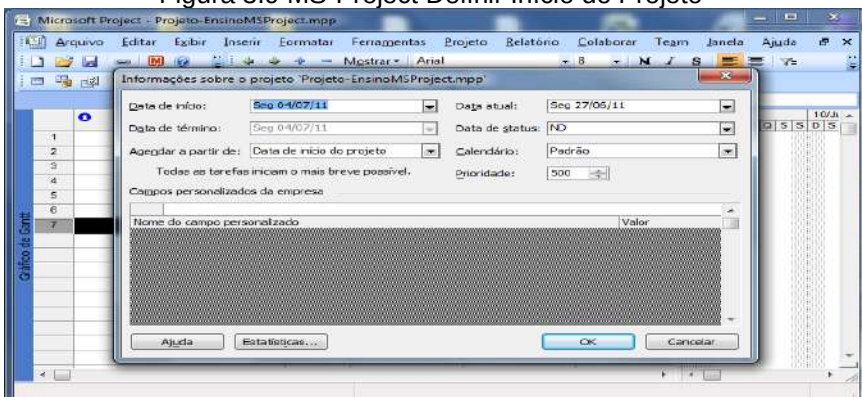
Tempo. Disp.: < <http://goo.gl/fxEEExo> >

Figura 3.8 MS-Project Definição do Calendário



Fonte: Próprio Autor

Figura 3.9 MS-Project Definir Início do Projeto



Fonte: Próprio Autor

3.15.2 Gestão do Escopo

Serve para descrever os requisitos funcionais e não funcionais com o objetivo de fazer somente o que é necessário.

3.15.2.1 Artefatos

- Planejamento do escopo;
- Definição do escopo;
- Verificação do escopo; e
- Controle do escopo.

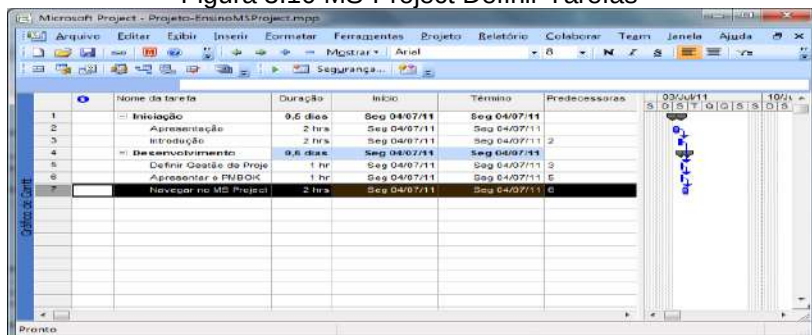
3.15.2.2 Importante

- Definir as tarefas; e
- Definir os *Links* entre as tarefas.

3.15.2.3 Wiki's

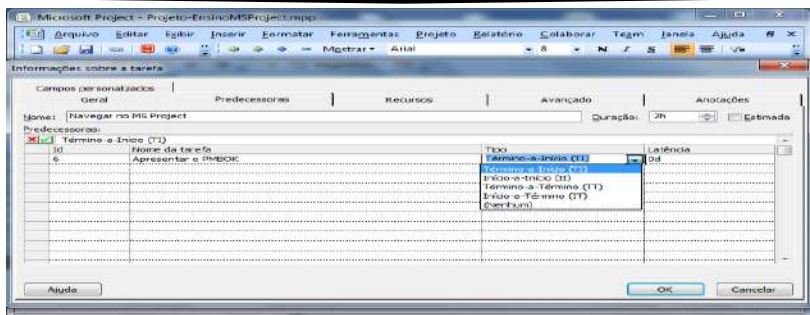
- Escopo. Disp.: < <http://qoo.gl/OTM0cM> >

Figura 3.10 MS-Project Definir Tarefas



Fonte: Próprio Autor

Figura 3.11 MS-Project Definir Relacionamento



Fonte: Próprio Autor

Figura 3.12 MS-Project Tipos de Relacionamento
Tipos de Relacionamento

Tipo de Relacionamento	Descrição	Apresentação no Gráfico de Gant
Finish-to-Start (FS)	A tarefa só começa quando a predecessora termina.	
Start-to-Start (SS)	A tarefa só começa quando a predecessora começa.	
Finish-to-Finish (FF)	A tarefa só termina quando a predecessora termina.	
Start-to-Finish (SF)	A tarefa só termina quando a predecessora começa.	

Fonte: Próprio Autor

3.15.3 Gestão dos Recursos

Server para descrever as pessoas envolvidas no projeto.

3.15.3.1 Artefatos

- Planejamento de recursos humanos;
- Contratar ou mobilizar a equipe do projeto;
- Desenvolver a equipe de projeto; e
- Gerenciar a equipe de projeto.

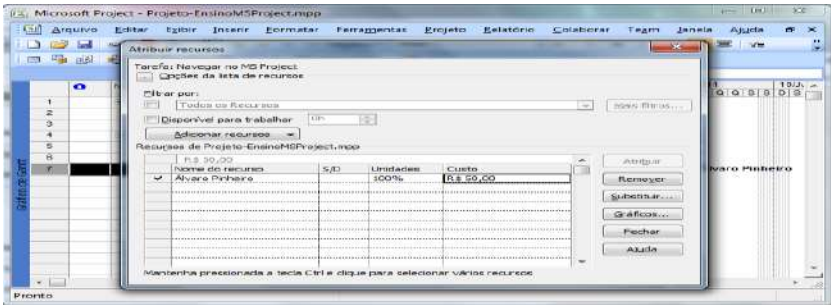
3.15.3.2 Importante

- Planilha de alocação.

3.15.3.3 Wiki's

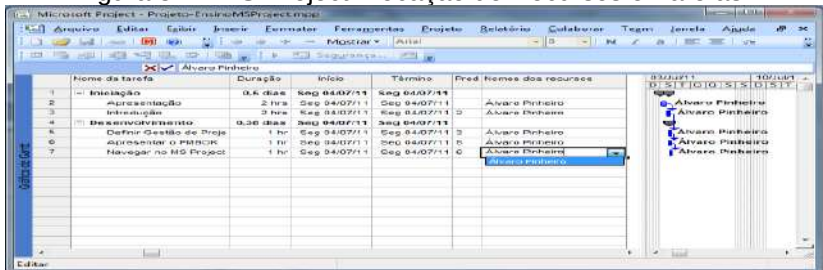
- RH. Disp.: < <http://goo.gl/1z0qhn> >

Figura 3.13 MS-Project Alocação de Recursos



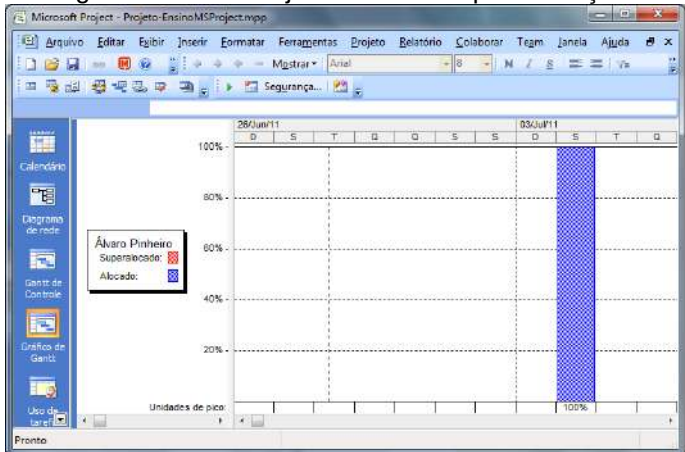
Fonte: Próprio Autor

Figura 3.14 MS-Project Alocação de Recursos e Tarefas



Fonte: Próprio Autor

Figura 3.15 MS-Project Verificar Super Alocação



Fonte: Próprio Autor

Figura 3.16 MS-Project Informações

Vejamos as informações que são armazenadas:

Resource Name ..Nome dado ao recurso.

Type.....- WORK (mão de obra): para identificar pessoas, equipamentos ou as facilidades que são atribuídas a uma tarefa e não consumidas durante a tarefa.
 - MATERIAL: para identificar os recursos consumíveis, tais como o aço, que devem ser usada na realização do projeto
 - COST: para identificar artigos orçamentais, tais como os custos de curso, que não são dependentes da quantidade de trabalho ou da duração de uma tarefa.

Material Label Identificação do usuário.

Initial Um nome abreviado para atribuir recursos em determinados modos.

Group.....Pode-se atribuir os recursos a grupos, para facilitar futuros relatórios.

Max Units.....A quantidade total de unidades de recurso disponível.

Std Rate.....O custo padrão definido pelas unidades de medida da duração.

Ovt Rate O custo da hora extra.

Cost/Use.....O custo por cada vez que o recurso é utilizado (fixo).

Accrue At.....Especifica quando os custos são acumulados.

Base Calendar Informa o calendário básico utilizado pelo recurso.

Code Permite a definição de um código que pode ser utilizado para ordenação, custo por classificação ou reportagem.

Fonte: Próprio Autor

3.15.4 Gestão de Custos

Serve para descrever os processos necessários para que o projeto seja finalizado dentro do orçamento aprovado.

3.15.4.1 Artefatos

- Estimativa de custos;
- Orçamentos; e
- Controle de custos.

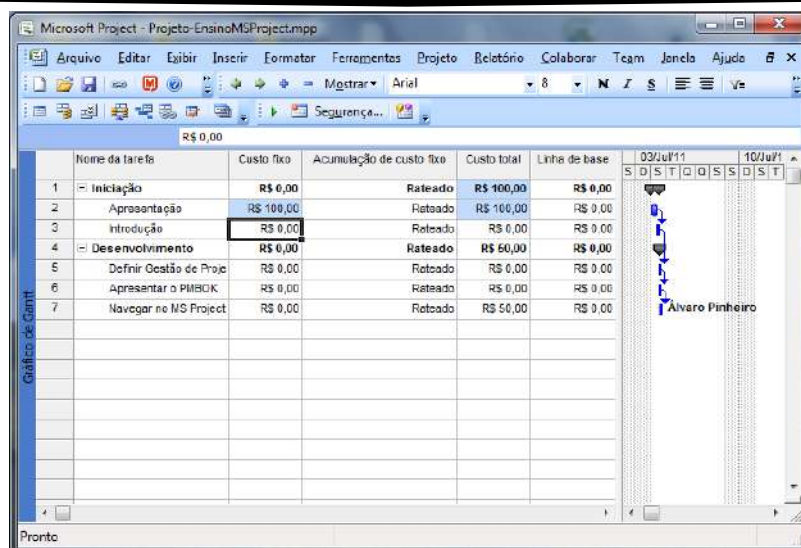
3.15.4.2 Importante

- Custo e custo restante.

3.15.4.3 Wiki's

Custos. Disp.: < <http://qoo.gl/DTsCti> >

Figura 3.17 MS-Project Tabela de Custos



Fonte: Próprio Autor

3.15.5 Gestão de Integração

A Gerência da integração do projeto é o núcleo do gerenciamento de projetos, e é composto dos processos do dia-a-dia com os quais o gerente de projetos conta para garantir que todas as partes do projeto funcionem juntas. É um processo contínuo que a gerente completa para garantir que o projeto prossiga do início ao fim. Gerenciar a integração do projeto é garantir que os componentes do projeto precisam trabalhar juntos e é papel do gerente de projetos fazer que isso aconteça. Exige habilidades em negociação e gerenciamento de conflitos de interesses. Também exige habilidades gerais de gerenciamento, boa comunicação, organização, familiaridade técnica com o produto, etc.

Pode ser dividido em três partes:

- Desenvolvimento do plano do projeto;
- Execução do plano do projeto; e
- Controle de mudanças no projeto.

3.15.5.1 Entradas:

- Declaração do trabalho de projeto;
- Casos de negócio;
- Contrato e SLA.

3.15.5.2 Ferramentas e Técnicas:

- Metodologia de planejamento de projeto;
- Habilidades e conhecimentos dos *stakeholders*;
- Sistemas de informações do gerenciamento de projetos; e
- Gerenciamento do valor do trabalho realizado.

3.15.5.3 Saída:

- Execução do Plano do Projeto.

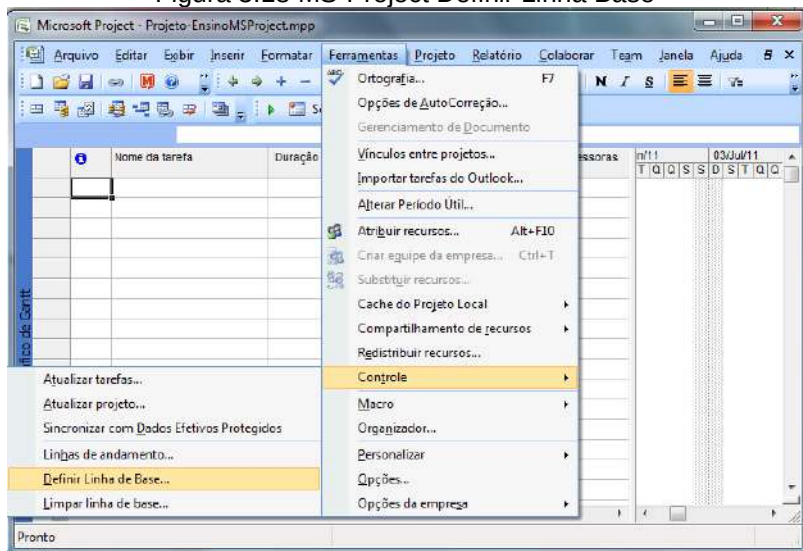
3.15.5.4 Importante

- Definir a Linha de Base; e
- Definir a Estrutura Analítica do Projeto ou Estrutura de Dados em Tópicos (EDT).

3.15.5.5 Wiki's

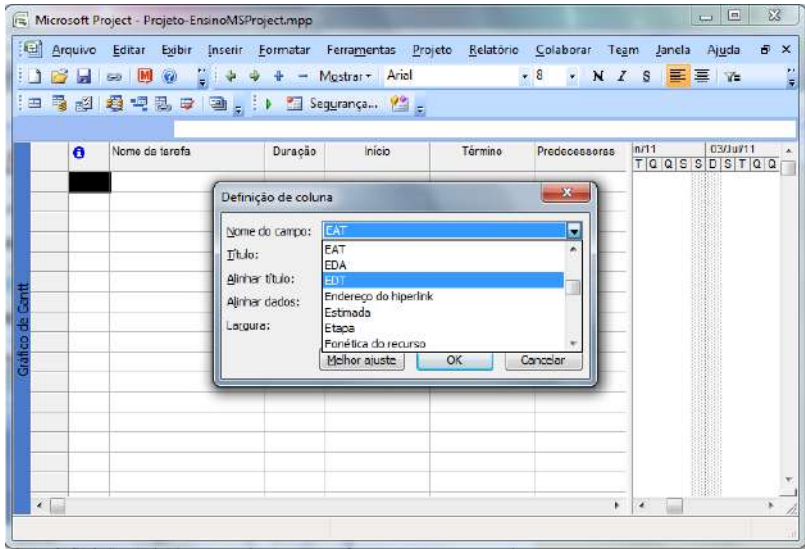
WBS. Disp.: < <http://goo.gl/8iNthq> >

Figura 3.18 MS-Project Definir Linha Base



Fonte: Próprio Autor

Figura 3.19 MS-Project WBS



Fonte: Próprio Autor

3.15.6 Gestão da Comunicação

Server para descrever a forma de comunicação e garantir agilidade na geração, coleção, disseminação, armazenamento e disposição final das informações do projeto.

3.15.6.1 Artefatos

- Planejamento das comunicações;
- Distribuição das informações;
- Relatório de desempenho; e
- Gerenciar as partes interessadas.

3.15.6.2 Importante

Planilha de comunicações.

Figura 3.20 MS-Project Definir e-mails



Fonte: Próprio Autor

3.15.7 Gestão de Aquisições

Serve para descrever as formas de aquisições de bens e serviços de fora da organização do projeto.

3.15.7.1 Artefatos

- Planejar compras e aquisições;
- Planejar contratações;
- Solicitar respostas de fornecedores;
- Selecionar fornecedores;
- Administração de contrato; e
- Encerramento de contrato.

3.15.7.2 Importante

- Planejamento das aquisições.

3.15.7.3 Wiki's

- Aquisições.

3.15.8 Gestão de Riscos

Server para descrever os processos relacionados a identificar, analisar e responder aos riscos do projeto.

3.15.8.1 Artefatos

- Planejamento do gerenciamento de riscos;
- Identificação de riscos;
- Análise qualitativa de riscos;
- Análise quantitativa de riscos;
- Planejamento de respostas a riscos; e
- Monitoramento e controle de riscos.

3.15.8.2 Importante

- Tabela de riscos.

3.15.8.3 Wiki's

- Riscos. Disp.: < <http://goo.gl/dRcFrV> >

3.15.9 Gestão de Qualidade

Serve para descrever os processos requeridos para garantir que o projeto vai satisfazer as necessidades pelas quais ele foi feito.

3.15.9.1 Artefato

- Planejamento da qualidade;
- Realizar a garantia da qualidade; e
- Realizar o controle da qualidade.

3.15.9.2 Importante

- Planilha de qualidade.

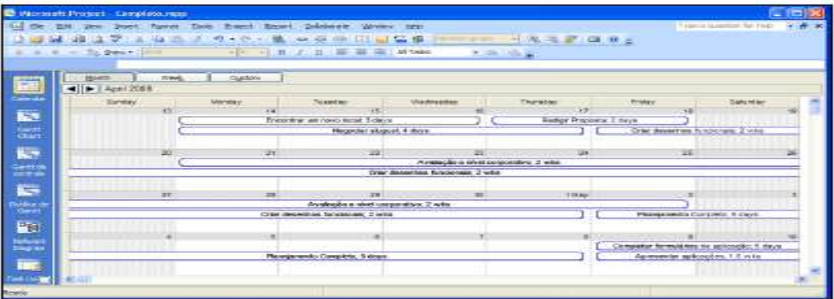
3.15.9.3 Wiki's

- Qualidade. Disp.: < <http://goo.gl/RLDXt> >

Figura 3.21 MS-Project Modos de Visualizações

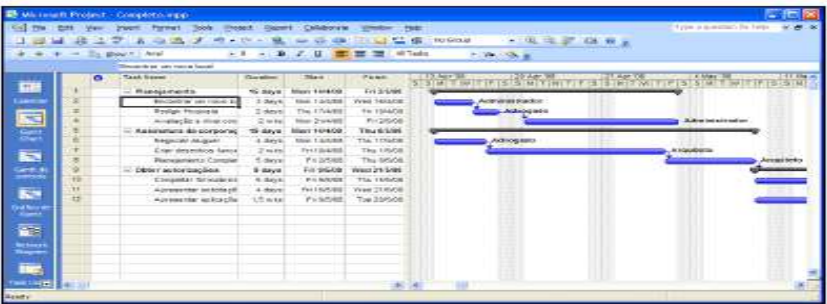
Calendar - Calendário

As tarefas marcadas são exibidas em formato de agenda para representar a sua duração durante as semanas do mês.



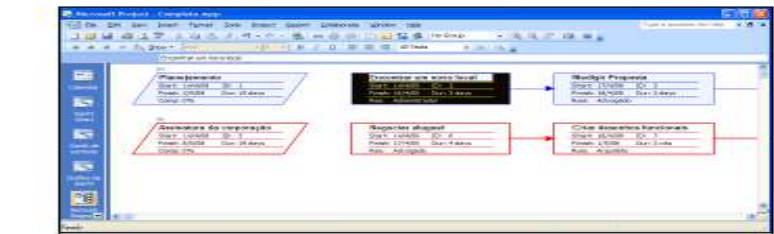
Gant Chart - Gráfico de Gant

O Gráfico de Gant é constituído por uma tabela que relaciona as tarefas e suas informações, e um gráfico que exibe a duração das tarefas e seus relacionamentos.



Network Diagram - Gráfico de Diagrama de Rede

Este fluxograma exibe o projeto com as tarefas em caixas ou nós. As linhas entre os nós representam elos entre as tarefas sucessoras e predecessoras. Cada nó exibe cinco campos de informação pertinente a essa tarefa.



Task Usage – Utilização das Tarefas

Esta tabela mostra a distribuição de custos e de trabalho para cada recurso em um período de tempo específico.



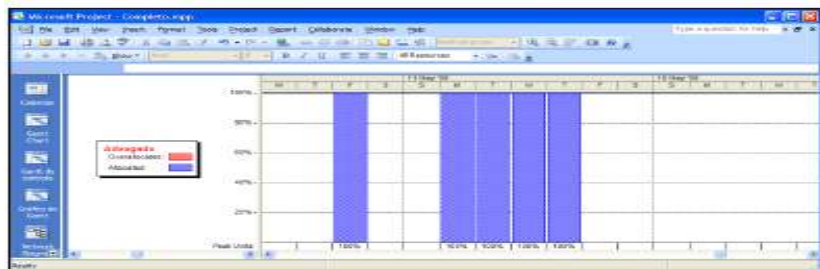
Tracking Gant – Gant de Controle

Este gráfico exibe o acompanhamento do realizado com a linha de base (projetado).



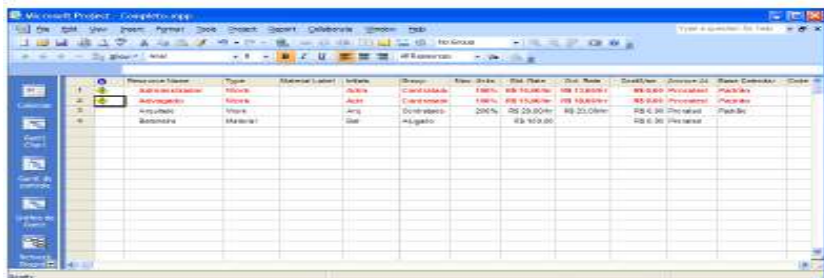
Resource Graph - Gráfico de Recursos

Este gráfico exibe a colocação dos recursos, trabalho ou custo de um recurso durante um período de tempo.



Resource Sheet – Planilha de Recursos

Este formato de planilha mostra a informação sobre cada recurso, tal como: taxa de pagamento, quantidade de pagamento, quantidade de horas de trabalho designadas e custo (inicial e real).



Fonte: Próprio Autor

3.16 IBM Primavera P6

Gestão de projetos é uma disciplina fundamental para conhecer os principais conceitos de gerenciamento. Claro que, administrar recursos pode ser realizado sem o entendimento de teorias administrativas, fazendo uso da intuição, porém nesses casos o sucesso está condicionado aos fatores sorte, experiência e afinidade. Contar com o fator sorte é um risco não aceitável. Experiência é importante, mas se você não a tem? Afinidade de forma geral, como ter equipes com sinergia, nem sempre é uma realidade. Assim, conhecer os conceitos e as ferramentas que podem auxiliar nessa árdua atividade é importante para se obter sucesso, no que podemos chamar "a arte de administrar".

Início esse estudo exibindo todas as opções do menu da ferramenta com suas respectivas funções:

- File-New: novo arquivo;
- File-Open: abrir arquivo;
- File-Close All: fechar todos os projetos;
- File-Page Setup: configurar página;
- File-Print Setup: configurar impressora;
- File-Print Preview: visualizar impressão;
- File-Print: imprimir;
- File-Project Architect: importar metodologias pré-definidas;
- File-Import: importar dados para o projeto;
- File-Export: exportar dados para o projeto;
- File-Send Project: exporta e envia por e-mail um projeto;
- File-Check in: baixa projeto para torná-lo somente leitura;
- File-Check out: levanta projeto liberando para leitura e escrita;
- File-Select Project Portfolio:
- File-Commit Changes: grava as modificações realizadas;
- File-Refresh Data (F5): atualiza os dados do projeto;
- File-Exit: finalizar o P6;
- Edit-Undo: desfazer a modificação;
- Edit-Cut: recortar dado selecionado;
- Edit-Copy: copiar dado selecionado;
- Edit-Paste: colar dado selecionado;
- Edit-Add: adicionar uma tarefa;
- Edit-Delete: apagar uma tarefa;
- Edit-Dissolve: dissolve os relacionamentos (*Links*) das atividades;
- Edit-Assign: atribui recursos a um objeto selecionado;
- Edit-link Activities: *Linka* atividades;
- Edit-Fill Down: preenche com massa de dados;
- Edit-Select All: seleciona todos os objetos do projeto;

- Edit-Find: procura um dado;
- Edit-Find Next: procura próximo dado;
- Edit-Replace: procura e troca dado;
- Edit-Spell Check: checa ortografia;
- Edit-User Preferences: define as preferências do usuário;
- View-Layout: salva e recupera as aparências definidas;
- View-Columns: esconde, exhibe, personaliza colunas do projeto;
- View-Table Font and Row: configura estilos dos objetos;
- View-Filter By: configura os filtros;
- View-Group and Sort By: agrupa e classifica;
- View-Bars: configura os gráficos;
- View-Show on Top: configura a visualização do topo;
- View-Show on Bottom: configura a visualização de baixo;
- View-Hint Help: configura a exibição da dica dos objetos;
- View-Expand All: expande todos os objetos;
- View-Collapse All: encapsula todos os objetos;
- View-Collapse To: encapsula objeto específico;
- View-Toolbars: configura a barra de ferramentas;
- View-Back: retrocede a visualização;
- View-Forward: avança a visualização;
- View-Home: retorna para a tela padrão;
- Project-Activities: exhibe as atividades;
- Project-Resource Assignments: exhibe recursos atribuídos;
- Project-ABS: exhibe estrutura analítica;
- Project-Assign Baselines: atribui linha de base;
- Project-Maintain Baselines: mantém as linhas de base;
- Project-Expences: realiza cálculos financeiros;
- Project-Work Products and Documents: trabalho com produtos e documentos;
- Project-Thresholds: gatilho definido pelo usuários (referenciando um status);
- Project-Issues: define questões as atividades;
- Project-Risk: gerenciamento de riscos;
- Project-Set Default Project: define projeto como padrão;
- Enterprise-Projects: exhibe os projetos da organização;
- Enterprise-Enterprise Project Estructured: exhibe a estrutura da organização;
- Enterprise-Tracking: exhibe toda a estrutura navegacional da empresa;
- Enterprise-Project Portfolios: exhibe o relacionamento entre os projetos;
- Enterprise-Resources: exhibe os recursos do projeto;
- Enterprise-OBS: monta a estrutura organizacional da empresa;
- Enterprise-Resource Codes: define os códigos dos recursos;
- Enterprise-Project Codes: define os códigos dos projetos;
- Enterprise-Activity Codes: define os códigos das atividades;

- Enterprise-User Defined Fields: define os campos de usuário;
- Enterprise-Calendars: define os calendários dos projetos;
- Enterprise-Resource Shift: realiza troca de recursos;
- Enterprise-Activity Step Templates: monta uma atividade;
- Enterprise-Cost Accounts: contas financeiras;
- Enterprise-Funding Sources: define financiamentos;
- Enterprise-Resource Curves: curvas dos recursos;
- Enterprise-External Applications: aplicações externas;
- Tools-Time Approval: ferramenta de aprovação de tempo;
- Tools-Top Down Estimation: ferramenta de estimativa top down;
- Tools-Reorganize Now: ferramenta de reorganização;
- Tools-Apply Actuals: ferramenta que aplica configurações;
- Tools-Schedule: ferramenta de programa execuções;
- Tools-Level Resources: ferramenta de definição de níveis de recursos;
- Tools-Update Progress: ferramenta de acompanhamento do progresso de atualização;
- Tools-Summarize: ferramenta de configuração de sumários;
- Tools-Job Services: ferramenta de configuração de serviços;
- Tools-Store Period Performance: ferramenta de otimização de períodos de trabalho;
- Tools-Global Change: ferramenta de configurações globais;
- Tools-Monitor Thresholds: ferramenta de monitoração dos gatilhos;
- Tools-Issue Navigator: ferramenta de navegação das questões;
- Tools-Reports: ferramenta de relatórios;
- Tools-Report Wizard: ferramenta de construção de relatórios;
- Tools-Publish: ferramenta de publicação;
- Tools-Check Project Integrity: ferramenta de cheque de integridade;
- Tools-Recalculate Assignment Cost: ferramenta de recalculo;
- Tools-Set Language: ferramenta de definição da linguagem;
- Tools-Claim Digger:
- Admin-Users: Administração dos usuários;
- Admin-Security Profiles: Administração dos perfis de usuários;
- Admin-Admin Preferences: Administração das preferências;
- Admin-Admin Categories: Administração das categorias;
- Admin-Currencies: Administração dos valores monetários;
- Admin-Timesheet Dates: Administração dos planilhas de trabalhos;
- Admin-Financial Periods: Administração dos períodos financeiros.

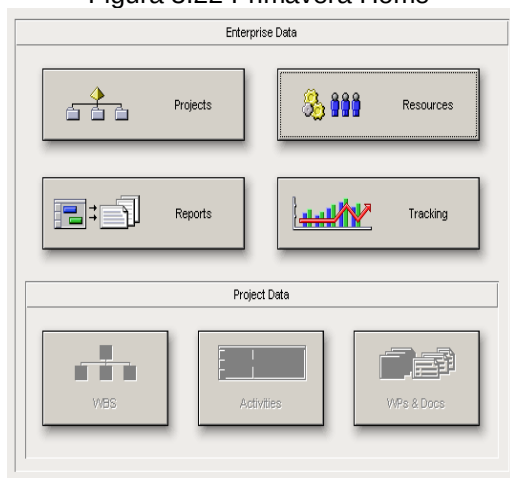
Clicando em *Projects* temos as seguintes informações do projeto: WBS (EAP); *Activities* (Atividades); *Assignments* (Atribuição de Recursos); WPs & Docs (Documentos); *Expenses* (Despesas); *Thresholds* (Gatilhos); *Issues* (Questões Pendentes); *Risks* (Riscos).

Mas antes de iniciar um projeto no P6 é necessário criar a organização da empresa. Assim sendo, vamos começar pelo *Organizational Breakdown Structure* (OBS) ou Estrutura Organizacional ou ainda Organograma.

Depois de estruturado o organograma deve-se criar o *Enterprise Project Structure* (EPS) ou a Estrutura dos Projetos da Empresa. Somente com o OBS e o EPS criados o P6 pode realizar corretamente o controle de múltiplos projetos.

Agora para se criar a estrutura organizacional da empresa antes de tudo deve-se criar os usuários do sistema e antes disso os perfis para que se possa atribuir a esses usuários os seus respectivos direitos e restrições de acesso e *Linkar* os usuários aos seus perfis.

Figura 3.22 Primavera Home



Fonte: Próprio Autor

A estrutura do Primavera *Enterprise* versão 6.0 também chamado de P6 é composta da ferramenta Cliente Servidor conhecida como Primavera Web ou MyPrimavera que possui duas ferramentas. O Enterprise Project Management (EPM) e o *Enterprise Methodology Management*. Esses dois módulos ou aplicações possuem diferentes objetivos e bancos de dados diferentes. O Project Management Database (PMDb) e o *Methodology Management Database* (MMDB). No módulo *Methodology Management* é armazenado a metodologia da organização, os modelos e lições aprendidas. Nestes modelos constarão os layouts e campos personalizados, facilitando o uso da ferramenta e diminuindo o tempo de criação de um novo projeto. O principal objetivo é o uso estruturado dos modelos, para padronização de procedimentos e maior clareza e conhecimento na execução e controle.

Figura 3.23 Tela de Abertura



Fonte: Próprio Autor

O módulo *Project Management* possibilita o planejamento, acompanhamento à execução das atividades e ciclos de vida dos projetos corporativos. O objetivo é criar um procedimento único e uma métrica única para a análise de desempenho. Este módulo deve ser usado pelo Gerente de Projeto para inserir os dados dos projetos. As informações podem ser guardadas em banco de dados único, evitando duplicidade de informações e facilitando o processo. O banco de dados poderá ser Oracle, SQL Server ou o MSSQL.

1. Barra de Títulos: aqui você verá todos os títulos dos projetos abertos com seus IDs e nomes entre parêntesis. O *Current Project* significa que você não está com nenhum projeto aberto no momento.

2. Barra de Menus: aqui são exibidos os menus existentes. Se você não estiver com algum projeto aberto, também não estará com o menu Project disponível.

3. Barra de Navegação: é exibido o nome do diretório do Primavera que você está acessando no momento. Na figura está escrito Home, já que você está na página de entrada do software. Os botões que estão nesta barra, à direita, são os botões de navegação e você poderá avançar ou voltar diretórios (não ações).

4. Barra de Diretórios: esta barra estará presente em todas as telas, caso você mantenha acionado o botão que exibe esta barra. Funciona como um atalho para ir a algum diretório do Primavera, que é estruturado em áreas.

5. Barra de Informações: esta barra há algumas informações sobre os dados abertos de um portfólio, o usuário logado, a data utilizada para o projeto e qual a linha de base do projeto aberto.

6. Barra de atalhos para informações da empresa: atalho para os diretórios de informações da empresa.

7. Barra de atalhos para Informações do projeto: atalho para que você escolha 3 dos 8 diretórios disponíveis para aparecer como atalho de diretórios de informações dos projetos.

- Print - Imprimir;
- Print Preview - Visualizar Impressão;
- Project Table - Tabela de Projetos;
- Gantt Chart - Gráfico de Gantt;
- Chart View - Visão Gráfica;
- Show / Nide Bottom Layout - Mostra/Oculto Layout Inferior;
- Bars - Barras;
- Columns - Colunas;
- Timescale - Escala de Tempo;
- 10.Filters - Filtros;
- 11.Group and Sort - Agrupar e Classificar/Ordenar;
- 12.Zoom in - Aumentar;
- 13.Zoom out - Diminuir; e
- 14.Zoom to best fit - Melhor ajuste de Visualização.

Figura 3.24 Primavera Barra de Ferramentas



Fonte: Próprio Autor

Barra de Opções: acesso ao menu *View* e também na barra de ferramentas. Serve para exibir textos como *Display* e *Layout*.

Abrir um Projeto: No Diretório *Projects*, na Tabela de projetos, selecione o projeto que se deseja abrir. Clique com o botão direito do mouse e aparecerá a lista de opções, selecione *Open Project*. Nesta forma, o padrão é abrir de modo compartilhado (*Shared*). Ao abrir, você será direcionado ao Diretório de atividades (*Activities*).

Figura 3.25 Primavera Barra de Opções



Fonte: Próprio Autor

3.17 Ferramentas

MS-Project. Disponível em < <http://office.microsoft.com/pt-br/> >

Primavera. Disponível em < <http://www.verano.com.br/p6.html> >

3.18 Sugestões de Leitura

Gerenciamento de Projetos: Guia do Profissional. Disponível em < <http://goo.gl/r1WYdD> >

Gerencia de Projetos: Fundamentos. Disponível em < <http://goo.gl/Rnwqll> >

Gerenciamento de Projetos. Disponível em < <http://goo.gl/Kev72Z> >

Experiências em Gestão de Projetos. Disponível em < <http://goo.gl/pjkXC7> >

3.19 Referências Web

Como Instalar o Project Server no Windows Server 2008. Disponível em < <http://goo.gl/qXSL9K> >

Comparações entre software de gerenciamento de projetos. Disponível em < <http://goo.gl/ExGz> >

Control Objectives for Information and related Technology (CobiT) wiki. Disponível em < <http://goo.gl/sidki> >

Gerenciamento de Projetos wiki. Disponível em < <http://goo.gl/osuyX> >

Guide to the Project Management Body of Knowledge (PMBOK Guide) wiki. Disponível em < <http://goo.gl/Oa0fh> >

Information Technology Infrastructure Library (ITIL) wiki. Disponível em < <http://goo.gl/BhTvV> >

Microsoft Office Guides. Disponível em < <http://goo.gl/Be2F2h> >

Microsoft Training wiki. Disponível em < <http://goo.gl/pdtEsm> >

MS-Project 2007 Quick Guide. Disponível em < <http://goo.gl/KbLVLK> >
MS-Project 2007 Step-by-step. Disponível em < <http://goo.gl/JpLnem> >
MS-Project Server wiki. Disponível em < <http://goo.gl/JpLnem> >
MS-Project wiki. Disponível em < <http://goo.gl/5n1AA> >
PMBOK 1996. Disponível em < <http://goo.gl/WD4ULJ> >
PMI Brasil wiki. Disponível em < <http://www.pmi.org.br/> >
PMI Internacional. Disponível em < <http://www.pmi.org/> >
PMI Pernambuco. Disponível em < <http://www.pmi.org.br/v2/html/> >
Project Management Institute (PMI) wiki. Disponível em < http://pt.wikipedia.org/wiki/Project_Management_Institute >
RoadMaps MS-Project. Disponível em < <http://goo.gl/IY9smr> >
Teclas de Atalho Project wiki. Disponível em < <http://goo.gl/mnMwJ0> >
Tecnologias da Internet. Disponível em < <http://goo.gl/16DUwW> >

3.20 Referências Bibliográficas

ABDALA, Elisabeth. Tecnologias da Internet: Casos Práticos Em Empresas. 1ªed. Porto Alegre: Edipucrs, 2003.
CHIAVENATO, Idalberto. Introdução à Teoria Geral da Administração. 7ªed. Rio de Janeiro: Campus, 2004.
COMER, Douglas. Redes de Computadores e Internet. 4ªed. US: Bookman, 2007.
DAYCHOUM, Merhi. 40 Ferramentas e Técnicas de Gerenciamento. 4ªed. Rio de Janeiro: Brasport, 2012.
DEMARCO, Tom; LISTER Timothy. Peopleware Productive Projects and Teams. US: Dorset House Pub, 1999.
JOHNSON, Timothy; CHATFIELD, Carl. Microsoft Office Project 2007: Step by Step. Washington: Microsoft, 2009.
KAPLAN, Robert; NORTON, David. A estratégia em ação: Balanced Scorecard. 1ªed. RJ: Campus, 2003.
KOSCIANSKI, André; SOARES, Michel. Qualidade de Software. 2ªed. São Paulo: Navatec, 2007.
MANZANO, Alves. Guia de Orientação e Desenvolvimento de Sites. 1ªed. São Paulo: Érica, 2008.
MARMEL, Elaine. Microsoft Project 2007: The Bible. Washington: Wiley, 2008.
PMI. A Guide to the Project Management Body of Knowledge: Pmbok Guide. 2011.

4 Referências Bibliográficas

- ABDALLA, Mauricio. O Princípio da Cooperação. 2ªed. Rio de Janeiro: Paulus, 2004.
- CHIAVENATO, Idalberto. Introdução à Teoria Geral da Administração. 7ªed. Rio de Janeiro: Campus, 2004.
- DAYCHOUM, Merhi. 40 Ferramentas e Técnicas de Gerenciamento. 4ªed. Rio de Janeiro: Brasport, 2012.
- DEMARCO, Tom; LISTER Timothy. Peopleware Productive Projects and Teams. US: Dorset House Pub, 1999.
- DRUCKER, Peter. Management: Tasks, Responsibilities, Practices. 1ªed. US: Transaction Publishers, 2007.
- JOHNSON, Timothy; CHATFIELD, Carl. Microsoft Office Project 2007: Step by Step. Washington: Microsoft, 2009.
- KAPLAN, Robert; NORTON, David. A estratégia em ação: Balanced Scorecard. 1ªed. RJ: Campus, 2003.
- KOSCIANSKI, André; SOARES, Michel. Qualidade de Software. 2ªed. São Paulo: Navatec, 2007.
- KRUCHTEN, Philippe. Introdução ao RUP Rational Unified Process. 1ªed. Rio de Janeiro: Ciência Moderna, 2003.
- LOPES, Filomena; *et al.* Desenvolvimento de Sistemas de Informação, Métodos e Técnicas, 2005.
- MARMEL, Elaine. Microsoft Project 2007: The Bible. Washington: Wiley, 2008.
- PATTON, Ron. Software Testing. 2ªed.US: Sams Indianapolis, 2005.
- PMI. A Guide to the Project Management Body of Knowledge: Pmbok Guide. 2011.
- PRESSMAN, Roger. Engenharia de Software: uma abordagem profissional. 7ªed. Porto Alegre: Bookman, 2011.
- RIOS, Emerson. Teste de Software. 3ªed.Rio de Janeiro: Alta Books, 2013.
- SOMMERVILLE, Ian. Engenharia de Software. 8ªed. Pearson, 2007.
- VARIAN, Hal. A Economia da Informação. 13ªed. RJ: Campus, 2003.

5 Referências Web

ANSI site oficial. Disponível em < <http://www.ansi.org/> >
ANSI wiki. Disponível em < <http://goo.gl/ZCrMxk> >
Como Instalar o Project Server no Windows Server 2008. Disponível em < <http://goo.gl/qXSL9K> >
Comparações entre software de gerenciamento de projetos. Disponível em < <http://goo.gl/ExGz> >
Engenharia de Software wiki. Disponível em < http://pt.wikipedia.org/wiki/Engenharia_de_software >
Guia de Teste. Disponível em < <http://goo.gl/hKmKE1> >
Guide to the Project Management Body of Knowledge (PMBOK Guide) wiki. Disponível em < <http://goo.gl/Oa0fh> >
IBM Rational Unified Process. Disponível em < <http://goo.gl/ugl9H7> >
IBM RUP. Disponível em < <http://www-01.ibm.com/software/awdtools/rup/> >
ISO site oficial. Disponível em < <http://www.iso.org/iso/home.html> >
ISO wiki. Disponível em < <http://goo.gl/vjWG> >
Knowledge Management (KM) wiki. Disponível em < http://en.wikipedia.org/wiki/Knowledge_management >
Microsoft Office Guides. Disponível em < <http://goo.gl/Be2F2h> >
Microsoft Training wiki. Disponível em < <http://goo.gl/pdtEsm> >
MS-Project 2007 Quick Guide. Disponível em < <http://goo.gl/KbLVlK> >
MS-Project 2007 Step-by-step. Disponível em < <http://goo.gl/JpLnem> >
MS-Project Server wiki. Disponível em < <http://goo.gl/JpLnem> >
MS-Project wiki. Disponível em < <http://goo.gl/5n1AA> >
PMBOK 1996. Disponível em < <http://goo.gl/WD4ULJ> >
PMI Brasil wiki. Disponível em < <http://www.pmi.org.br/> >
PMI Internacional. Disponível em < <http://www.pmi.org/> >
PMI Pernambuco. Disponível em < <http://www.pmi.org.br/v2/html/> >
Project Management Institute (PMI) wiki. Disponível em < http://pt.wikipedia.org/wiki/Project_Management_Institute >
QSM. Disponível em < <http://goo.gl/3UzXCY> >
Rational Unified Process (RUP). Disponível em < <http://www-01.ibm.com/software/br/rational/> >
RoadMaps MS-Project. Disponível em < <http://goo.gl/IY9smr> >
RUP. Disp.: < [http://www.wthreex.com/rup/portugues/index.htm%20\(Excelente%20site%20sobre%20RUP\)](http://www.wthreex.com/rup/portugues/index.htm%20(Excelente%20site%20sobre%20RUP)) >
SEI. Disponível em < <http://www.sei.cmu.edu/> >
SWEBOK. Disponível em < <http://www.computer.org/portal/web/swebok> >

6 Biografias

Alan Turing nasceu na Inglaterra em 1912, foi matemático, lógico, criptoanalista e cientista da computação. É um dos nomes importantes na Ciência da Computação. Formulou conceitos de algoritmo e computação com a máquina de Turing. Trabalhou no *National Physical Laboratory*. Saber mais. Disponível em < http://en.wikipedia.org/wiki/Alan_Turing >

Alfred Vaino Aho é cientista da computação canadense. Conhecido internacionalmente como co-autor da linguagem de programação AWK, junto com Peter Weinberger e Brian Kernighan, e também pela sua co-autoria no Livro do Dragão como é mais conhecido o livro, *Compilers: Principles, Techniques, and Tools*, escrito em parceria com Ravi Sethi e Jeffrey Ullman. Suas contribuições para área foram às versões iniciais das ferramentas Unix egrep e fgrep. Também possui livros nas áreas de algoritmos e estruturas de dados. Saber mais. Disponível em < http://pt.wikipedia.org/wiki/Alfred_Aho >

Avram Noam Chomsky nasceu nos Estados Unidos na Filadélfia em 1928, é um linguista e filósofo. É professor de Linguística no *Massachusetts Institute of Technology*. É autor de trabalhos fundamentais sobre as propriedades matemáticas das linguagens formais, sendo o seu nome associado à chamada Hierarquia de Chomsky. Seus trabalhos estão relacionados a uma abordagem matemática da linguagem. Saber mais. Disponível em < http://pt.wikipedia.org/wiki/Noam_Chomsky >

Christopher Date formou-se no Reino Unido pela High Wycombe Royal Grammar School em 1958 e recebeu o bacharelado em Matemática pela Universidade de Cambridge em 1962. Iniciou as atividades na área da computação como programador na Leo *Computers Ltd* em Londres onde também começou a lecionar. Em 1967 começou a trabalhar na IBM na Inglaterra também como programador e instrutor de computação, ficando entre os anos de 1969 e 1974 como o principal instrutor da área educacional da IBM. Saber mais. Disponível em < http://en.wikipedia.org/wiki/Christopher_J._Date >

Craig Larman é um canadense nascido em 1958 que trabalha como cientista da computação especializado em desenvolvimento iterativo e incremental, desenvolvimento de *software* ágil, análise orientada a objetos e modelagem ágil. Ele é o autor de vários textos sobre Scrum. Saber mais. Disponível em < http://en.wikipedia.org/wiki/Craig_Larman >

Erich Gamma é um dos quatro autores do livro *Design Patterns: Elements of Reusable Object-Oriented Software*. Ele é também o principal projetista do JUnit e do Eclipse. Nascido em 1961, em Zurique é cientista da computação e co-autor influente de livros didáticos sobre engenharia de *software*. Co-escreveu o JUnit framework de teste de *software* com Kent Beck e liderou o projeto do Eclipse da plataforma *Java Development Tools* (JDT). Também trabalhou na IBM Rational projeto Jazz. Saber mais.

Disponível em < http://en.wikipedia.org/wiki/Erich_Gamma >

Grady Booch nasceu em 1955 nos Estados Unidos, é engenheiro de software e cientista-chefe da IBM Research. Obteve seu grau de bacharel pela *United States Air Force Academy* e título de mestre em engenharia elétrica pela Universidade da Califórnia em Santa Barbara. Saber mais.

Disponível em < http://en.wikipedia.org/wiki/Grady_Booch >

Hal Varian, nascido em 1947, em Ohio, é economista especializado em microeconomia e economia da informação. É professor emérito nas universidades de Berkeley e Haas *School of Business*. Formou-se como bacharel em economia pelo e licenciado em matemática pelo MIT. Possui o doutorado em economia pela *Berkeley University*, e é doutor honorário pelo Instituto de Tecnologia de Karlsruhe, Alemanha. Já lecionou em *Stanford University*, *Oxford University*, *Michigan University*. Nos dias atuais é o economista-chefe da Google e publica regularmente artigos no *New York Times* e *Wall Street Journal*. Saber mais.

Disponível em < http://en.wikipedia.org/wiki/Hal_Varian >

Harvey Deitel tem mais de 40 anos de experiência na área de computação. Graduou-se e fez o mestrado no *Massachusetts Institute of Technology* e o doutorado na *Boston University*. Possuindo mais de 20 anos de experiência no ensino universitário. Possuindo diversas publicações traduzidas em diversos idiomas. Saber mais.

Disponível em http://www.informit.com/authors/author_bio.aspx?ISBN=9780137058426 <

Ian Sommerville, inglês nascido em 1951 é professor titular de engenharia de software na Universidade de St. Andrews na Escócia. Destacado pesquisador no campo da engenharia de sistemas, segurança e informática social. Saber mais.

Disponível em < http://en.wikipedia.org/wiki/Ian_Sommerville_%28academic%29 >

Idalberto Chiavenato, nascido em 1936 em São Paulo, é autor na área de administração de empresas e de recursos humanos. Atualmente, atua como conselheiro no Conselho Regional de Administração do Estado de São Paulo (CRA-SP) e é Presidente do Instituto Chiavenato de Educação. Saber mais. Disponível em < http://pt.wikipedia.org/wiki/Idalberto_Chiaavenato >

Ivar Jacobson é um sueco nascido em 1939, se graduou no Instituto de Tecnologia de Chalmers, onde fez seu mestrado em Engenharia Elétrica e possuindo doutorado pela *Royal Institute of Technology de Estocolmo*. Saber mais. Disponível em < http://en.wikipedia.org/wiki/Ivar_Jacobson >

James Gosling nasceu em Calgary no Canadá em 1955 é conhecido como o pai da linguagem de programação Java. Em 1977, formou Bacharel em Ciência da Computação pela Universidade de Calgary, e em 1983 tornou-se PhD em Ciência da Computação pela Universidade Carnegie Mellon. Enquanto estudava no doutorado, Gosling programou a versão original do editor de texto Emacs para UNIX, e antes de se juntar a Sun Microsystems ele construiu uma versão de UNIX para máquinas multiprocessadas, assim como diversos compiladores e sistemas de mensagens eletrônicas. Trabalhou de 1984 até 2010 na Sun Microsystems, atualmente está na Google. Saber mais. Disponível em < http://pt.wikipedia.org/wiki/James_Gosling >

James Rumbaugh é Bacharel em Física pelo Massachusetts Institute of Technology (MIT), possui mestrado em Astronomia pela Caltech, e é Ph. D. em Ciência da Computação também pelo MIT. Trabalhou na *Digital Equipment Corporation* (DEC), como cientista. Foi um dos líderes na criação da OMT e em 1994 se juntou a *Rational Software*, e junto com Ivar Jacobson e Booch de Grady desenvolveram a *Unified Modeling Language* (UML). Mais tarde, eles fundiram as suas metodologias de desenvolvimento de *software*, OMT, OOSE e Booch em um único processo batizado de *Rational Unified Process* (RUP). Saber mais. Disponível em < http://en.wikipedia.org/wiki/James_Rumbaugh >

John von Neumann nasceu na Hungria em 1903 naturalizado como estadunidense, foi um matemático. Membro do Instituto de Estudos Avançados em Princeton, *New Jersey*, do qual também faziam parte Albert Einstein. Professor na Universidade de Princeton e um dos construtores do ENIAC. Trabalhou no desenvolvimento da Física Nuclear e contribuiu com diversas teorias, a exemplo as dos conjuntos, análise funcional, ergódica, mecânica quântica, ciência da computação, economia, teoria dos jogos, análise numérica, estatística, entre outras. Saber mais.

Disponível em < http://pt.wikipedia.org/wiki/John_von_Neumann >

Kenneth Laudon é professor de Sistemas de Informação na New York University's Stern School of Business. Bacharel em Economia pela *Stanford University* e Ph. D pela *Columbia University*. Autor de vários livros que tratam do tema Sistemas de Informação, Organizações e Sociedade. Saber mais. Disponível em < <http://pages.stern.nyu.edu/~klaudon/> >

Patrick Naughton americano nascido em 1965 é um desenvolvedor de software conhecido como sendo um dos criadores originais da linguagem de programação Java, pois trabalhou na Sun Microsystems no Projeto Verde a origem do Java. Em 1994, saiu da Sun e foi para Starwave de propriedade de Paul Allen para desenvolver um servidor Java. É autor do livro O Manual do Java, Osborne, 1995 e co-autor do *Java: The Complete Reference*, com Herbert Schildt, Osborne, 1996. Em 1998, torna-se vice-presidente executivo da Infoseek. Saber mais.

Paul Deitel é graduado em Tecnologia da Informação e pós-graduação em Administração pela Massachusetts *Institute of Technology*. Trabalha como consultor em diversas organizações internacionais. Saber mais. Disponível em < http://www.informit.com/authors/author_bio.aspx?ISBN=9780137058426 >

Peter Ferdinand Drucker, austríaco nascido 1909 e que faleceu recentemente em 2005, foi um filósofo e economista, que por suas contribuições a comunidade, passou a ser considerado o pai da administração moderna, se tornando um dos pensadores mais citados pela teoria da globalização em economia em geral. Saber mais. Disponível em < http://pt.wikipedia.org/wiki/Peter_Drucker >

Roger Pressman é Bacharel em Engenharia de Software pela University of Connecticut, com Mestrado pela *University of Bridgeport* e PhD pela *University of Connecticut*. Pertence ao conselho editorial da *IEEE Software* e do *The Cutter TI Jornal*. Autor e consultor na área de Engenharia de Software. Trabalha na área de engenharia de software a mais de 30 anos, atuando como administrador, professor e consultor. É presidente da *R.S. Pressman & Associates, Inc.*, empresa de consultoria especializada em práticas de Engenharia de Software. Saber mais. Disponível em < <http://www.cutter.com/meet-our-experts/rpbio.html> >

Stephen Schach é professor emérito do curso de Ciência da Computação e Engenharia de Computação do Departamento de Engenharia Elétrica e Ciência

da Computação da Universidade de Vanderbilt. Possui mestrado em Física pelo *Weizmann Institute of Science* e doutorado em Matemática Aplicada pela *University of Cape Town*. Saber mais.

Disponível em < <http://www.vuse.vanderbilt.edu/~srs/> >

Tim O'Reilly, nasceu em 1954 na Irlanda, é o fundador da O'Reilly Media e estudioso e fomentador de software livre, para muitos é o criador da expressão Web 2.0. É Bacharel em filologia clássica pela Harvard University em 1975. Saber mais.

Disponível em < http://pt.wikipedia.org/wiki/Tim_O'Reilly >

Tom DeMarco, nascido em 1940 na Pensilvânia, é autor, professor e palestrante em engenharia de software e é reconhecido como um dos criadores da Análise Estruturada nos anos 1980. É Bacharel em Engenharia Elétrica pela Universidade de *Cornell University*, Mestre pela *Columbia University* e Doutor pela University of Paris at the Sorbonne. Saber mais.

Disponível em < http://en.wikipedia.org/wiki/Tom_DeMarco >

7 Artefatos de Projetos de Software

Avaliação da Organização de Desenvolvimento. Disponível em < <http://goo.gl/n4Aqyg> > Acesso:Jan/15.

Avaliação da Organização-alvo. Disponível em < <http://goo.gl/n5bKph> >

Avaliação de Iteração. Disponível em < <http://goo.gl/YIRRBv> >

Avaliação de Status. Disponível em < <http://goo.gl/546iYI> >

Caso de Desenvolvimento. Disponível em < <http://goo.gl/xT7kXd> >

Caso de Negócios. Disponível em < <http://goo.gl/9euZmQ> >

Documento de Arquitetura de Negócios. Disponível em < <http://goo.gl/9AuSQa> >

Documento de Arquitetura de Software. Disponível em < <http://goo.gl/BsuHyg> >

Especificação de Caso de Uso de Negócios Nome do Caso de Uso de Negócios. Disp.:< <http://goo.gl/XkILXt> >

Especificação de Caso de Uso Nome do Caso de Uso. Disponível em < <http://goo.gl/8OsCF8> >

Especificação de Realização de Casos de Uso Nome do Caso de Uso de Negócios. Disp.:< <http://goo.gl/PoKB2f> >

Especificação de Realização de Casos de Uso Nome do Caso de Uso. Disponível em < <http://goo.gl/NHNReG> >

Especificação dos Requisitos de Software Para Subsistema ou Recurso. Disponível em < <http://goo.gl/mnOYe5> >

Especificação dos Requisitos de Software. Disponível em < <http://goo.gl/dfnJMy> >

Especificação Suplementar de Negócios. Disponível em < <http://goo.gl/G52y77> >

Especificação Suplementar. Disponível em < <http://goo.gl/aGdgmi> >

Glossário de Negócio. Disponível em < <http://goo.gl/SRqCx8> >

Glossário de Negócios. Disponível em < <http://goo.gl/vjDhRQ> >

Glossário. Disponível em < <http://goo.gl/UvhQ2H> >

Lista de Materiais. Disponível em < <http://goo.gl/A0sTGQ> >

Lista de Riscos. Disponível em < <http://goo.gl/Vvy1js> >

Modelo de Negócio. Disponível em < <http://goo.gl/Q3O6Ya> >

Notas de Release. Disponível em < <http://goo.gl/e0FC0h> >

Plano de Aceitação do Produto. Disponível em < <http://goo.gl/RH4JDj> >

Plano de Desenvolvimento de Software (Projeto Pequeno). Disponível em < <http://goo.gl/J9gsVB> >

Plano de Desenvolvimento de Software. Disponível em < <http://goo.gl/5KRFCJ> >

Plano de Garantia de Qualidade. Disponível em < <http://goo.gl/D1M0Gp> >

Plano de Gerenciamento de Configuração. Disponível em < <http://goo.gl/almYbn> >

Plano de Gerenciamento de Requisitos. Disponível em < <http://goo.gl/1iwQYG> >

Plano de Gerenciamento de Riscos. Disponível em < <http://goo.gl/kvwGyW> >

Plano de Implantação. Disponível em < <http://goo.gl/zK4sDV> >

Plano de Integração do Build. Disponível em < <http://goo.gl/J5eP3T> >

Plano de Iteração. Disponível em < <http://goo.gl/1f4yLQ> >

Plano de Métricas. Disponível em < <http://goo.gl/64uo2r> >

Plano de Negócio. Disponível em < <http://goo.gl/Jfw2VK> >

Plano de Requisitos. Disponível em < <http://goo.gl/sLM5Lu> >

Plano de Resolução de Problemas. Disponível em < <http://goo.gl/pwvd6l> >

Plano de Teste. Disponível em < <http://goo.gl/XM0cNO> >

Proposta Comercial. Disponível em < <http://goo.gl/D117e4> >

Regras de Negócios. Disponível em < <http://goo.gl/Zx5CWY> >

Requisitos de Software. Disponível em < <http://goo.gl/Bi5piJ> >

Solicitações dos Principais Envolvidos. Disponível em < <http://goo.gl/XGUuGR> >

Sumário de Avaliação de Testes. Disponível em < <http://goo.gl/ZVLVb6> >

Termo de Abertura do Projeto. Disponível em < <http://goo.gl/CYnUmT> >

Visão. Disponível em < <http://goo.gl/liXVDn> >

Visão (Projeto Pequeno). Disponível em < <http://goo.gl/x8YMS7> >

Visão do Negócio. Disponível em < <http://goo.gl/wUKpZc> >

8 Ferramentas

MS-Project. Disponível em < <http://office.microsoft.com/pt-br/> >

Primavera. Disponível em < <http://www.verano.com.br/p6.html> >

Project-open: gerenciador de projetos. Disponível em < <http://www.project-open.org/> >

9 Sugestões de Leitura

Engenharia de Software. Disponível em < <http://goo.gl/K55qXq> >

Experiências em Gestão de Projetos. Disponível em < <http://goo.gl/pjkXC7> >

Gerencia de Projetos: Fundamentos. Disponível em < <http://goo.gl/Rnwgll> >

Gerenciamento de Projetos. Disponível em < <http://goo.gl/Kev72Z> >

Gerenciamento de Projetos: Guia do Profissional. Disponível em < <http://goo.gl/r1WYdD> >

Gerenciando Projetos de Desenv. Software com PMI, RUP e UML. Disp.: < <http://goo.gl/GcXUe4> >

Guia prático de engenharia de software. Disponível em < <http://goo.gl/ppe5ff> >

Teste e Análise de Software: Processos, Princípios e Técnicas. Disponível em < <http://goo.gl/UIKwTi> >

Livros da série Fundamentos da Engenharia de Software



Fundamentos da Engenharia de Software: Conceitos Básicos é uma coletânea de

disciplinas que integradas servem para fundamentar o entendimento da construção de projetos de software com qualidade, isto é, baseado em processos maduros e reconhecidos pela comunidade tecnológica. O objetivo deste livro é fornecer ao leitor as bases necessárias para o desenvolvimento de aplicações sejam Desktop, Web ou Mobile. Iniciando a leitura na Teoria da Computação, passando por Processos, Linguagens, Bancos de Dados e finalizando com Sistemas de Informação e Colaboração. Este livro pode ser lido capítulo a capítulo ou somente a disciplina desejada, pois sua elaboração consiste na compilação das disciplinas fundamentais da Engenharia de Software que são independentes, mas ao mesmo tempo se integram objetivando o desenvolvimento de aplicações.



Introdução à Banco de Dados. Neste são abordados os conceitos básicos de bancos de dados e seus

sistemas gerenciadores, mas com o foco na arquitetura relacional, porque ainda hoje o mercado faz uso em larga escala desses bancos de dados, mesmo que o paradigma predominante seja o orientado a objetos e que, já existam a um bom tempo bancos orientados a objeto, até mesmo os bancos objetos-relacionais que são um híbrido entre essas duas arquiteturas, o que predomina ainda é o relacional, assim, este material é focado na linguagem de consulta estruturada para os SGBD-Rs do mercado, com foco na comparação de cinco dos mais utilizados bancos relacionais, os quais são: Oracle, SQLServer, MySQL, SQLBase e Interbase.



Este livro é sobre processos de desenvolvimento de software, evidenciando a necessidade

de qualidade na construção de sistemas, conceituando a diferença entre desenvolvimento Adhoc e com processo. Para isso é realizado a introdução à engenharia de requisitos abordando as técnicas para a elicitación de requisitos que forneçam subsídios necessários para uma construção de software com maior qualidade, enfatizando a necessidade de se aplicar na construção de qualquer sistema as técnicas de análise e modelagem, evidenciando o uso da linguagem da Linguagem de Modelagem Unificada (UML) para diagramar um projeto de software, explicando a necessidade do uso de modelos na construção, entrando com detalhes na análise orientada a objetos, com o objetivo de explorar os seus conceitos de requisitos e modelagem integrados. Este material é finalizado com a introdução à medidas de esforço de desenvolvimento, técnica necessária para responder as perguntas básicas de qualquer desenvolvimento: Qual o prazo e custo? E para responder a essas questões é abordado o uso da métrica análise de ponto de função.



Este livro aborda os sistemas que são classificados como informação, a exemplo, sistemas de apoio a

decisão, sistemas estratégicos, sistemas gerenciais e sistemas transacionais. A produção deste material que compõe o volume 4 da coleção Fundamentos da Engenharia de Software é resultado da compilação das aulas produzidas nas disciplinas que compõem os capítulos deste livro.



A motivação deste livro é exemplificar os conceitos de Padrões de Projetos utilizando a linguagem de programação

o Java, sendo a construção uma compilação das aulas produzidas com o intuito de facilitar o entendimento do assunto abordando os seguintes temas: Paradigma Orientado a Objetos que introduz o leitor nos conceitos do POO; Linguagem de Modelagem Unificada para apresentar a simbologia UML dos conceitos de POO; Linguagem de Programação Java apresentando essa poderosa linguagem de programação orientada a objetos para exemplificar os padrões de projeto; e Padrões de Projetos que neste livro aborda os mais referenciados nas academias, sendo eles o GRASP e GoF.



Este livro é o resultado do uso da ferramenta MS Project da Microsoft utilizada na aplicação dos conceitos de

gestão de projetos do PMBOK com as premissas da engenharia de testes para aquisição de qualidade nos produtos de software.



Este livro aborda basicamente e os conceitos básicos de programação o como autômatos, tipos de

linguagens, princípios dos compiladores, paradigmas de desenvolvimento e lógica de programação.



Este livro introduz nas tecnologias Web abordando os conceitos básicos para desenvolvim ento para

Internet com a apresentação da plataforma Dot Net e exibindo dicas de codificação para a linguagem de marcação ASPX, para a linguagem de script mais utilizada pelos navegadores o JavaScript com exemplos de CSS e principalmente dicas de código para a linguagem de programação CSharp e de banco de dados SQL com foco no SQLServer.

