UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

MÉTRICAS DE SOFTWARE APLICADAS À METODOLOGIA DE DESENVOLVIMENTO DE SISTEMAS DO STJ

EDENILDO DE OLIVEIRA
NELSON AMOR

ORIENTADOR: RAFAEL TIMÓTEO DE SOUSA JUNIOR
CO-ORIENTADOR: DAYSE DE MELLO BENZI

MONOGRAFIA DE ESPECIALIZAÇÃO EM ENGENHARIA ELÉTRICA
ÁREA DE ENGENHARIA E QUALIDADE DE SOFTWARE

PUBLICAÇÃO: UnB.LabRedes.MFE.17/2008

BRASÍLIA / DF: JULHO/2008

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

MÉTRICAS DE SOFTWARE APLICADAS À METODOLOGIA DE DESENVOLVIMENTO DE SISTEMAS DO STJ

.

EDENILDO DE OLIVEIRA
NELSON AMOR

MONOGRAFIA DE ESPECIALIZAÇÃO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ESPECIALISTA

APROVADA POR:

RAFAEL TIMÓTEO DE SOUSA JUNIOR, Doutor, UnB
(ORIENTADOR)

RICARDO STACIARINI PUTTINI, Doutor, UnB/ENE
(EXAMINADOR INTERNO)

ODACYR LUIZ TIMM JR., Mestre, TECSOFT
(EXAMINADOR EXTERNO)

DAYSE DE MELLO BENZI, Mestre, UnB
(CO-ORIENTADORA)

DATA: BRASÍLIA/DF, 02 DE JULHO DE 2008.

FICHA CATALOGRÁFICA

AMOR, NELSON.; OLIVEIRA, EDENILDO DE.
Métricas de Software Aplicadas à Metodologia de Desenvolvimento de Sistemas do STJ [Distrito Federal] 2008.
xx 83p., 297 mm (ENE/FT/UnB, Especialista, Engenharia Elétrica, 2008).

Monografia de Especialização – Universidade de Brasília, Faculdade de Tecnologia. Departamento de Engenharia Elétrica.

1. Software 2. Métricas
3. Qualidade

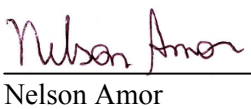
I. ENE/FT/UnB. II. Título (Série).

REFERÊNCIA BIBLIOGRÁFICA

AMOR, NELSON.; OLIVEIRA, EDENILDO DE. (2008). Métricas de Software Aplicadas à Metodologia de Desenvolvimento de Sistemas do STJ. (Monografia de Especialização), Publicação XXX/2008, Departamento de Engenharia Elétrica, Universidade de Brasília , Brasília , DF, (76)p.

CESSÃO DE DIREITOS

NOME DO AUTOR: Nelson Amor, Edenildo de Oliveira.
TÍTULO DA DISSERTAÇÃO: Métricas de Software Aplicadas à Metodologia de Desenvolvimento de Sistemas do STJ.
GRAU/ANO: Especialista/2008.



Nelson Amor
QNL 05 BLOCO B APTO 203
CEP 72.150-612 – Taguatinga – DF - Brasil

Edenildo de Oliveira
QNN 04 CONJUNTO F CASA 4
CEP 72.220-046 – Ceilândia – DF - Brasil

Edenildo dedica essa monografia a sua família pela compreensão e incentivo nas horas em que esteve ausente, por motivo de estudo e pesquisa.

Nelson dedica esta monografia a seus pais por sempre incentivarem a ousadia e ter os pés no chão.

AGRADECIMENTOS

Ao nosso orientador Professor Doutor. Rafael Timóteo de Sousa Júnior, pelo apoio e incentivo da viabilização para o desenvolvimento deste trabalho e para o nosso desenvolvimento como especialista em engenharia de software.

À nossa Professora Mestra Dayse de Mello Benzi, co-orientadora deste trabalho, que nos orientou para um trabalho mais objetivo, incentivou a adoção de métricas para instituições como o STJ e nos deu suporte integral nos momentos mais complicados do trabalho.

Agradecimentos individuais:

Edenildo

Agradeço a todos os professores do curso pelo empenho e paciência, e por terem promovido a troca de conhecimentos necessária para despertar o meu interesse pela área de engenharia de software. Agradeço também a todos os amigos do STJ que sempre ajudaram nas horas de dúvidas, tanto no trabalho quanto em sala de aula.

Nelson

Eu agradeço a oportunidade do STJ ter disponibilizado este esforço de oferecer uma pós-graduação com o intuito de aperfeiçoar o seu corpo de servidores, do qual tenho orgulho de ser um servidor para somar em qualidade e eficiência no emprego do dinheiro público.

RESUMO

O trabalho descrito nesta dissertação objetiva definir métricas aplicáveis ao processo de desenvolvimento de software do STJ – Superior Tribunal de Justiça, bem como apontar soluções para mensuração de seu software legado como mensurar o desenvolvimento de novos softwares.

O resultado prático do uso de métricas deverá ser alcançado em poucos meses após o período de coleta de dados para análise de resultados e definição de parâmetros.

O ponto de equilíbrio do trabalho não foi apenas catalogar inúmeras métricas que poderiam não ter aplicabilidade prática, mas sim definir um conjunto inicial de métricas que podem trazer resultados sólidos para o desenvolvimento de softwares.

ABSTRACT

The work described in this thesis aims define applicable metrics to software development process from STJ – Superior Tribunal de Justiça, as well point solutions to measure their legacy software as well to measure a development of new softwares.

The practical results of the use of metrics should be reached in a few months as time to gather data for results analisys and definition of standards.

The point of balance of the work was not gather several metrics that could has not a practic applicability, but yet define a initial set of metrics that could bring solid results to software development.

ÍNDICE

1. INTRODUÇÃO 1

2. CONSIDERAÇÕES INICIAIS 3

2.1. MEDIDA, MEDIÇÃO E MÉTRICA 3
2.2. QUALIDADE DE SOFTWARE 3
2.3. GERÊNCIA DA QUALIDADE TOTAL 6

3. TAMANHO FUNCIONAL DO SOFTWARE 9

3.1. LINHAS DE CÓDIGO (KLOC – KILO LINES OF CODE) 9
3.2. PONTOS DE FUNÇÃO 11
3.2.1. Benefícios da APF segundo Vasquez [5]: 11
3.2.2. A APF baseia-se em 3 pontos básicos: 13
3.2.2.1. Arquivos: 13
3.2.2.2. Entradas: 13
3.2.2.3. Saídas: 13
3.2.3. O Fator de Ajuste 15
3.2.3.1. Comunicação de Dados 16
3.2.3.2. Processamento Distribuído 16
3.2.3.3. Performance 17
3.2.3.4. Configuração Altamente Utilizada 18
3.2.3.5. Volume das transações 18
3.2.3.6. Entrada de Dados On-Line 19
3.2.3.7. Eficiência do Usuário Final 19
3.2.3.8. Atualização On-Line 20
3.2.3.9. Complexidade de Processamento 21
3.2.3.10. Reusabilidade 22
3.2.3.11. Facilidade de Instalação 23
3.2.3.12. Facilidade de Operação 23
3.2.3.13. Múltiplos Locais 24
3.2.3.14. Facilidade de Mudanças 25
3.2.4. Como classificar as funções do tipo transação 26
3.2.5. Tipos de Contagem 27
3.2.5.1. Contagem Indicativa ou Dedutiva. 28
3.2.5.2. Contagem Estimada ou por Estimativa 28
3.2.5.3. Contagem Detalhada 29
3.2.5.4. Tipo de Aplicação para definir o Tipo de Contagem 29
3.2.6. Adaptações ao estudo da APF 30

4. MÉTRICAS DE QUALIDADE DE SOFTWARE 33

4.1. MÉTRICAS DE QUALIDADE DO PRODUTO 33
4.1.1. Métrica de densidade de Defeitos 34
4.1.2. Perspectiva do cliente 35
4.1.3. Métrica de problemas detectados pelo cliente 35
4.1.4. Métrica de satisfação do cliente 36
4.2. MÉTRICAS DE QUALIDADE DE PROCESSO 37
4.3. MÉTRICAS DE QUALIDADE DE PROJETO 40
4.3.1. Métrica de produtividade de software 40
4.3.2. Métricas OO 41
4.3.2.1. Métricas de Lorenz e as "Rules of Thumb" segundo Kan[2] 41
4.3.2.2. Métricas Chidamber e Kemerer ou Conjunto de Métricas CK segundo Kan [2] 43

5. AS SETE FERRAMENTAS BÁSICAS DE ISHIKAWA. 49

5.1. CHECK-LISTS OU CHECK-SHEET 49
5.2. DIAGRAMA DE PARETO 50
5.3. HISTOGRAMA 51
5.4. RUN-CHARTS 52
5.5. SCATTER DIAGRAM 53

5.6. CONTROL CHART 55
5.7. DIAGRAMA DE CAUSA E EFEITO – FISHBONE 57

6. MÉTRICAS NO RUP 59

6.1. UM CONJUNTO MÍNIMO DE MÉTRICAS 61
6.2. UM CONJUNTO PEQUENO DE MÉTRICAS 62
6.3. UM CONJUNTO COMPLETO DE MÉTRICAS 62
6.3.1. Métricas para o processo 63
6.3.2. Métricas para o produto 63
6.3.2.1. Documentos 63
6.3.2.2. Modelos 64
6.3.3. Métricas para o projeto 65
6.3.4. Métricas para os recursos 66

7. PROPOSTA INICIAL PARA USO NO PROCESSO DE DESENVOLVIMENTO DO STJ 67

7.1. LINHAS DE CÓDIGO 67
7.1.1. Como contar para softwares legados: 67
7.1.2. Como contar para softwares novos 67
7.1.3. Produtos obtidos do KLOC no Tribunal: 68
7.2. USO DE APF NO TRIBUNAL: 68
7.3. MÉTRICA DE SATISFAÇÃO DO CLIENTE 70
7.4. MÉTRICA DE QUALIDADE DO PRODUTO 71
7.5. MÉTRICA DE ORIENTAÇÃO A OBJETOS 71

8. CONCLUSÕES 73

9. BIBLIOGRAFIA 75

10. ANEXO 77

10.1. CONTRATO STJ Nº 067/07 - PROCESSO STJ Nº 8122/2006 77

ÍNDICE DE TABELAS

TABELA 1 - TABELA RÁPIDA DE CLASSIFICAÇÃO DE AÇÕES EM EE, CE E SE..... 27

TABELA 2 - RULES OF THUMB..... 42

TABELA 3 - ANÁLISE DE WMC..... 44

TABELA 4 – DADOS DA ANÁLISE DE CBO DAS CLASSES DE EXEMPLO..... 45

TABELA 5 - DADOS DA ANÁLISE DE RFC DAS CLASSES DE EXEMPLO..... 45

TABELA 6 - DADOS DA ANÁLISE DE LCOM 46

TABELA 7 - MÉTRICAS APLICÁVEIS EM DOCUMENTOS 63

TABELA 8 - MÉTRICAS APLICÁVEIS EM CASOS DE USO 64

ÍNDICE DE FIGURAS

FIGURA 1- INTER-RELACIONAMENTO DOS ATRIBUTOS DE QUALIDADE 4

FIGURA 2 - FASES DO MODELO DO PROCESSO EM CASCATA 5

FIGURA 3 - FASES DO MODELO DO PROCESSO RUP 6

FIGURA 4 - SISTEMA TQM 7

FIGURA 5 - CLASSIFICAÇÃO DO PROCESSO DE CONTAGEM 26

FIGURA 6 - RELACIONAMENTO ENTRE ERRO, FALHA E AVARIA 34

FIGURA 7 - EXEMPLO DE MATRIZ DE APROXIMAÇÃO(ONDE FORAM ENCONTRADOS E ORIGEM DOS ERROS) ONDE I0 A I2 – ITERAÇÕES, UT = UNIT TEST, CT- COMPONENT TEST , ST – SYSTEM TEST 38

FIGURA 8 - REPRESENTAÇÃO DE UM CICLO DE DEFEITOS 38

FIGURA 9 – CLASSES DE EXEMPLO 45

FIGURA 10 – GRAFO DA ANÁLISE DE LCOM 46

FIGURA 11 - EXEMPLO DE *CHECK-SHEET* 49

FIGURA 12 - PARETO DIAGRAM OF DEFECTS BY COMPONENT PROBLEM INDEX 51

FIGURA 13 - EXEMPLOS DE HISTOGRAMAS 52

FIGURA 14 - *RUN CHART* OU GRÁFICO DE LINHA 52

FIGURA 15 - SCATTER DIAGRAM 53

FIGURA 16 - EXEMPLO DE CORRELAÇÃO LINEAR POSITIVA 54

FIGURA 17 - EXEMPLO DE CORRELAÇÃO LINEAR NEGATIVA 54

FIGURA 18 - EXEMPLO DE CORRELAÇÃO NÃO LINEAR 54

FIGURA 19 - MODELO GENÉRICO DE UM GRÁFICO DE CONTROLE (*CONTROL-CHART*) 55

FIGURA 20 - PROCESSO PREVISÍVEL OU ESTÁVEL OU SOB CONTROLE 56

FIGURA 21 - PROCESSO IMPREVISÍVEL OU INSTÁVEL, OU FORA DE CONTROLE 56

FIGURA 22 - GRÁFICO DE CONTROLE DE SHEWHART 57

FIGURA 23 – MODELO BÁSICO DE DIAGRAMA DE CAUSA E EFEITO 58

FIGURA 24 - EXEMPLO DE DIAGRAMA DE CAUSA E EFEITO MAIS COMPLETO. 58

1. INTRODUÇÃO

Atualmente o STJ - Superior Tribunal de Justiça, encontra-se em implantação de uma MDSW (Metodologia de Desenvolvimento de Software).

A fase de gerência de requisitos está definida, é operacional e o STJ possui experiência com utilização da mesma em fábrica externa. A gerência de requisitos atualmente está em processo de realimentação de informações e revisão dos artefatos produzidos, ou seja, o documento de regra de negócios e o documento de requisitos.

O STJ também terceiriza a produção de alguns de seus softwares por fábrica de software externa. Nesse contexto ainda é apenas empregada a métrica de APF (Análise de Pontos de Função) para contagem de pontos de função por estimativa na fase de contratação e a contagem de pontos de função detalhada para finalizar o pagamento do produto.

Não é o intuito desta monografia treinar os analistas e técnicos em APF, mas sim que se catalogue o estado da arte para os conceitos da APF, os níveis de complexidades dos elementos mensuráveis, a determinação do fator de ajuste baseado nas 14 Características Gerais do Sistema (CGS) e aponte como o STJ deverá utilizar a métrica para aferir as diferenças do início para o fim da construção de um software.

Também está fora do escopo desta monografia a técnica COCOMO e COCOMOII, pois os autores identificam a técnica como um tema que poderia estender demais e não ser tão pontualmente aplicável como as métricas de APF e KLOC.

As fases de Projeto, Análise, Construção, Testes e Configuração e Mudança estão programadas para serem iniciadas com o estudo por comissões técnicas formadas por servidores do próprio STJ.

Vale ressaltar também que o STJ já possui uma Arquitetura de Software definida para aplicações Cliente/Servidor e está definindo de uma arquitetura para sistemas WEB.

A definição desta arquitetura para o ambiente cliente/servidor teve como principal fator impulsionador a dificuldade em uniformizar a codificação das diversas aplicações utilizadas no tribunal, juntamente com a recente organização do processo de desenvolvimento, que passou a ser baseado no Processo Unificado. O sucesso da análise e elaboração da arquitetura deve-se graças ao empenho e experiência da equipe técnica formada por analistas e técnicos da própria casa.

Todos os princípios utilizados, como arquitetura em camadas e processo de desenvolvimento iterativo incremental, foram baseados em estudos e análise de casos de uso internos. Outro fator importante que levou à adoção da arquitetura em camadas foi encontrar uma solução inicial na qual se possa futuramente trocar os componentes ou atualizá-los sem grandes transtornos, já que a linguagem de programação utilizada pela maioria das aplicações é a linguagem DELPHI.

As dificuldades já foram aparecendo no início da contratação da fábrica, pois precisávamos encontrar uma forma de se mensurar as solicitações, bem como encontrar uma forma de aferir o que ela poderia nos entregar e cobrar pela construção. Como descrito anteriormente, uma solução inicial adotada para superar esta barreira, foi adoção da APF, para medição funcional dos softwares, fato que incentivou a realização deste trabalho.

Outras dificuldades enfrentadas no início da implantação do processo de desenvolvimento estão sendo analisadas, algumas destas dificuldades devem ser exploradas nesta dissertação, como por exemplo: Análise de dados referentes aos projetos, produtos e processo; Utilização de ferramentas no controle estatístico de qualidade e levantamento de dados para este controle.

Portanto o intuito desta monografia é fazer uma análise do estado da arte do que se utiliza em métricas de software e com isto catalogar e sugerir quais métricas seriam aplicáveis inicialmente no processo definido pelo STJ. Seja para sistemas legados e suas manutenções, bem como para sistemas a serem criados e evoluídos. Métricas aplicáveis tanto para sistemas produzidos pela casa como aos sistemas produzidos por *outsourcing* em fábricas de softwares contratadas.

2. CONSIDERAÇÕES INICIAIS

2.1. MEDIDA, MEDIÇÃO E MÉTRICA

Conforme Galorath e Evans [1] seguem algumas definições para Medida, Medição e Métrica:

- Medida: Valor quantitativo da extensão, quantidade, dimensões, capacidade ou tamanho de algum atributo do processo ou produto de software. (Ex: Quantidade de classes-Chave de um software)
- Medição: Ato de determinar uma medida (Ex: Investigação do número de erros em um módulo)
- Métrica: Medida quantitativa do grau de posse de um atributo dado por parte de um sistema, componente ou processo (Ex: Média de erros detectados na revisão, número de erros encontrados por pessoa).

Portanto as métricas podem ser consideradas como as medidas quantitativas que permitem ter uma visão aprofundada da eficácia do processo de desenvolvimento de software, do software, e dos projetos. O uso de métricas geralmente está associado ao trabalho de engenheiros, arquitetos e analistas de software.

Ainda segundo Galorath e Evans [1] as medidas e métricas servem para obter dados sobre qualidade e produtividade bem como mensurar e elucidar o tamanho de um software ou projeto de software, para que se possa planejar melhor o tempo, esforço e custo do desenvolvimento.

Tais medidas ainda auxiliam muito na avaliação de viabilidade de projetos dentro de uma organização, visto que, se possa planejar e estimar custos e recursos a serem alocados para o desenvolvimento de uma aplicação de software.

2.2. QUALIDADE DE SOFTWARE

A busca por produtos cada vez mais confiáveis e mais aceitos no mercado levou as indústrias a adotarem padrões, regras e processos de desenvolvimento para adequarem os seus produtos às exigências dos consumidores. Estes mesmos conceitos de melhoria da qualidade de produtos industrializados chegaram também a afetar o mercado de desenvolvimento de software.

Com isto surgiu o termo de qualidade de software, que é a aplicação dos modelos de qualidade no processo de desenvolvimento de software.

Uma maneira de se medir a qualidade de um software, portanto, pode ser expressa através de métricas, como por exemplo:

- Taxa de defeitos – Expressa pelo número de defeitos por 1000 linhas de código, por pontos de função ou outra unidade.
- Confiabilidade - Expressa pelo número de falhas por total de horas de operação, tempo médio entre falhas, satisfação do cliente/usuário - Geralmente medida em porcentagem de satisfação ou não satisfação com o produto, etc. A IBM monitora a satisfação do cliente pelo CUPRIMDSO (Capability, Usability, Performance, Reliability, Installability, Maintainability, Documentation/informantion, Service and Overral). A HP baseia-se no FURPS (Functionality, Usability, Reliabiblity, Perfomance and Serviceability). Várias outras corporações utilizam outros métodos para medir a satisfação do cliente, ou seja, cada uma possui um modelo de qualidade e todos estes modelos são baseados nos mesmos atributos de qualidade. Segundo Kan [2] estes atributos de qualidade se inter-relacionam conforme podemos ver na figura abaixo.

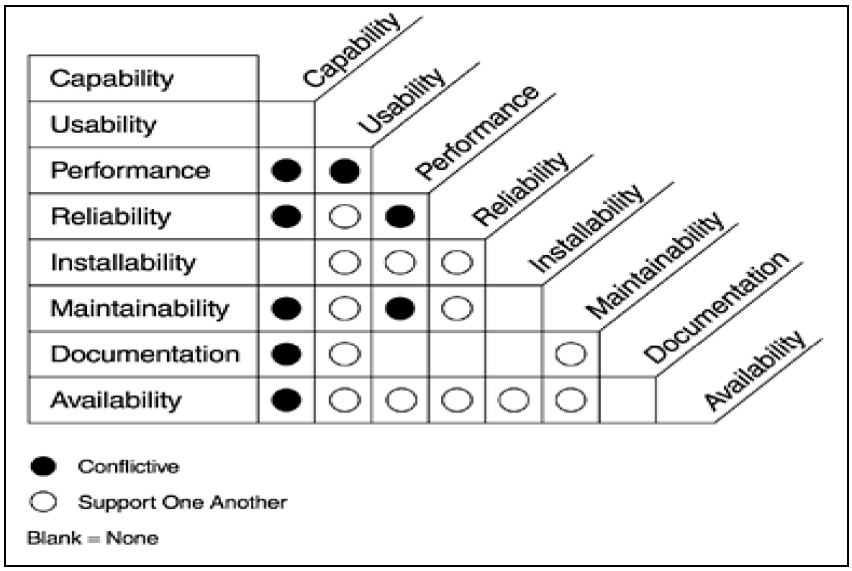


Figura 1- Inter-Relacionamento dos Atributos de Qualidade

Com as visões destes atributos e métodos utilizados, podemos adicionar mais uma informação na definição de qualidade de software: A conformidade dos requisitos do usuário.

“ Não é surpresa que erros em requisitos constituem a maior parte dos erros no desenvolvimento de software. De acordo com Jones (1992) 15% ou mais de todos os defeitos de software são erros em requisitos. Um processo de desenvolvimento que não faz o mapeamento dos requisitos tende a produzir softwares com baixa qualidade.

Para aplicar os conceitos de qualidade ou até um processo de qualidade de software é necessário implantar um processo de desenvolvimento de software que possua estágios, e em cada estágio seriam necessários aplicar e avaliar parâmetros de qualidade que ao final da produção dariam uma visão melhor da qualidade Total do produto. “ por Kan [2]

Focalizando o STJ, ao elaborar uma análise e montar um catálogo inicial de métricas, a serem utilizadas no processo de desenvolvimento verificar-se que está migrando do modelo de processo Cascata para o modelo Iterativo e Incremental baseado no RUP (*Rational Unified Process*). A seguir podem ser vistas as representações das fases dos dois modelos de processo de desenvolvimento.

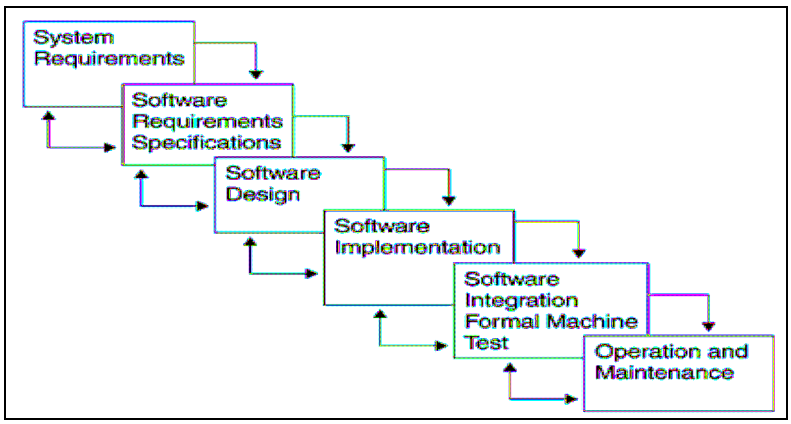


Figura 2 - Fases do Modelo do Processo em Cascata

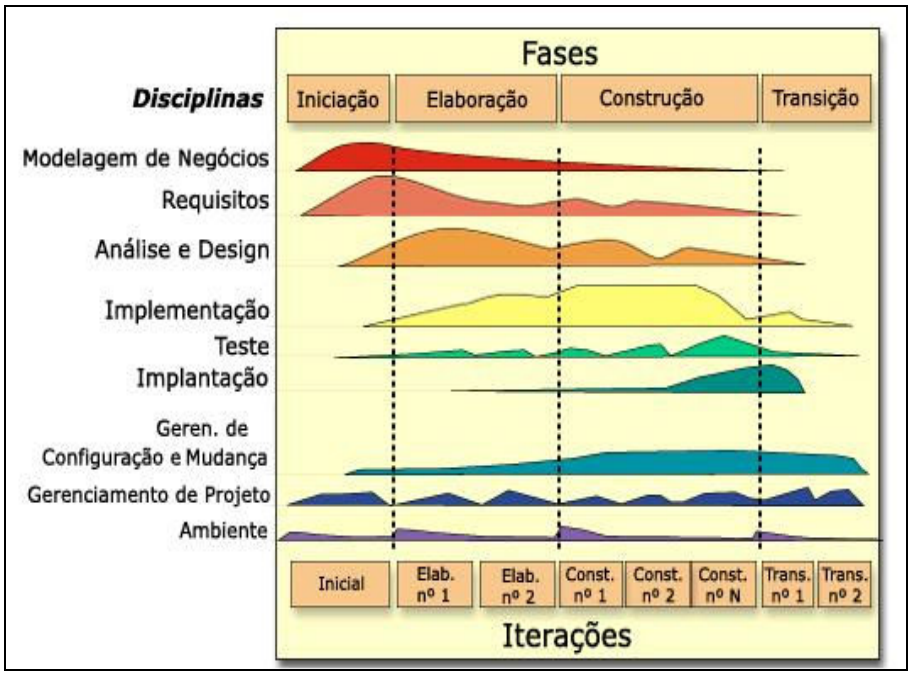


Figura 3 - Fases do Modelo do Processo RUP

As figuras 2 e 3 acima explicitam que os dois processos trabalham com fases nas quais serão extraídas e aplicadas métricas e medições para a criação de indicativos de controle estatístico de qualidade.

2.3. GERÊNCIA DA QUALIDADE TOTAL

O Termo *Total Quality Management* (TQM) foi originalmente utilizado em 1985 pelo *Naval Air Systems Command.* Este termo representa um estilo de gerencia destinado a armazenar casos de sucesso, basicamente ele cria uma cultura na organização em que todos os membros participam da melhoria contínua do processo, produtos e serviços. Vários métodos específicos para implementar a filosofia de qualidade total são encontrados nos trabalhos dos grandes nomes da qualidade (Crosby, Deming, Feigenbaum, Ishikawa, Juran e Gryna).

Apesar de possuir variações em sua implementação, os elementos chave de um sistema TQM podem ser resumidos como mostrado na figura a seguir:

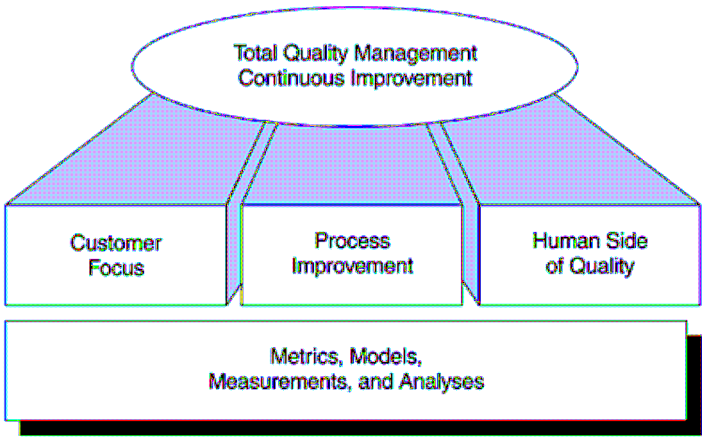


Figura 4 - Sistema TQM

As métricas, assim como pode ser visto na figura 4, permeiam todo o sistema TQM, através delas pode-se avaliar e mensurar etapas e fases de um processo de desenvolvimento de software, bem como também auxiliam na indicação de pontos de alta ou baixa qualidade em um produto, projeto ou processo.

3. TAMANHO FUNCIONAL DO SOFTWARE

3.1. LINHAS DE CÓDIGO (KLOC – KILO LINES OF CODE)

Segundo Kan [2] a métrica de Linhas de Código (LOC) apresentam alguma complexidade. O maior problema vem da ambigüidade da definição operacional e a contagem real. No início da computação, quando a linguagem de programação era o Assembler, onde cada linha física era o mesmo que uma instrução, a definição de LOC era clara. Porém, com a criação de linguagens de programação de alto-nível, foi quebrada essa correspondência de um para um nas linhas de código.

Diferenças entre linhas físicas e sentenças de instruções (ou linhas de código lógica) e diferenças entre linguagens contribuíram para uma grande diferença na contagem de linhas de código.

Ainda com a mesma linguagem de programação, os métodos e algoritmos utilizados por diferentes ferramentas de contagem podem causar diferenças significantes na contagem final.

Jones [3] descreve as seguintes variações da métrica de linha de código:

- Conte apenas linhas executáveis;
- Conte apenas linhas executáveis e definição de dados;
- Conte apenas linhas, definição de dados e comentários;
- Conte apenas linhas executáveis, definição de dados, comentários e controles de tarefas na linguagem;
- Conte apenas linhas que controlem a tela de entrada de dados;
- Conte apenas linhas finalizadas por terminadores lógicos;

Na obra *Software Engineering Metrics and Models* de Conte [4], LOC é definido como: “Uma linha de código é qualquer linha textual de um programa que não é comentário ou linha em branco, não considerando o número de condicionais ou fragmentos de condicionais na linha. Isso inclui especificamente todas as linha contidas em um programa: cabeçalhos, declarações e sentenças executáveis ou não executáveis” (p.35). A definição acima é mais direta, ou seja, toda linha escrita em um programa, exceto comentário ou linha em branco é linha de código mensurável pela métrica LOC segundo Kan [2].

Vale ressaltar que há uma diferença no uso de Linhas de Código para mensurar densidade de defeitos ou produtividade.

O uso de Linhas de Código para mensurar defeitos provê algo mais direto. Pode-se dizer que um módulo do programa possui 2 KLOC (duas mil linhas de código) e que após 2 anos em produção este mesmo módulo obteve taxa de 10 defeitos por KLOC, ou seja, apresentou 20 defeitos neste ciclo de 2 anos de uso.

Já o uso de Linhas de código para mensurar produtividade pode mascarar resultados reais. Uma equipe de software que faz uso de *Frameworks* ou executa a fase de análise com mais critério pode ter uma baixa produtividade por escrever poucas linhas de código, ao passo que uma equipe que não otimiza determinadas rotinas ou tem o estilo de programação mais redundante apresentará uma alta taxa de produtividade por desenvolver aplicativos com um número maior de linhas de código.

Outro perigo do uso da métrica de linha de código é indicar que um software escrito de forma não otimizada é de alto custo ou que um software altamente eficiente no uso dos recursos de máquina como um software simples e barato pelo seu baixo número de linhas de código. Isso porque a métrica de linhas de código não reflete diretamente no esforço das fases de requisito, especificação e análise. A LOC também não é capaz de mensurar o trabalho externo para o uso de padrões de interface, código e projeto, bem como mensurar tarefas derivadas do produto como a criação do manual do usuário.

Um ponto importante para o uso da LOC é o versionamento de um produto, ou seja, algo ligado diretamente à Gerência de Configuração e Mudanças - GCM.

A boa prática do uso de KLOC com GCM é de medir o produto no momento da entrega. Coletar as alterações do produto durante o seu ciclo de vida e durante esse ciclo de vida criar uma base histórica de erros que o mesmo venha a apresentar. Essa densidade de erros por KLOC pode ter uma relação direta com requisitos conflitantes ou imaturidade da equipe de desenvolvimento.

Mas quando ocorre mudança do requisito ou inserção de novas funcionalidades, por projeto de melhoria, é que entra o importante papel da Gerencia de Configuração e Mudanças que deve associar um ID para cada projeto de mudança ou melhoria. A equipe de codificação deve mapear cada alteração/melhoria em porções do código rastreáveis por este ID. A densidade de erros por KLOC pode ser medida no produto total ou diretamente nas mudanças. Muitas empresas optam por rastrear a densidade de erros do produto como um todo e não das partes modificadas. Tudo depende da criticidade do software

3.2. PONTOS DE FUNÇÃO

Segundo Vasquez [5] a Análise de Pontos de Função – APF é uma técnica de medição das funcionalidades fornecidas por um software do ponto de vista de seu usuário. Ponto de função é a unidade de medida desta técnica que tem por objetivo tornar a medição independente da tecnologia utilizada para construção do software. Ou seja, a APF busca medir o que o software faz, e não como ele foi construído.

Uma característica chave da APF é que ela é centrada "no que o usuário enxerga", ou seja, é baseado nas interfaces com o usuário, ou melhor, nas telas, consultas e relatórios obtidos do software.

Essa característica chave torna a APF uma técnica não muito boa para sistemas que tenham alta complexidade embutida em um único botão, ou em sistemas com muito processamento e manipulação automática de dados em *background*. Porém é uma técnica interessante por não ser centrada na tecnologia do momento ou em tecnologias já em desuso. Essa desconexão à tecnologia empregada possibilita que a APF seja usada por qualquer paradigma, o que é diferenciado é o preço ou custo do ponto de função entre as linguagens de programação existentes. Isso é uma vantagem, pois o número de pontos de função de um software é sempre o mesmo para qualquer linguagem que ele venha a ser produzido.

Vale ressaltar que o termo usuário é mais amplo que uma visão de apenas um ser humano interagindo com um software. Usuário é toda pessoa ou agente que interage com a aplicação em questão e demanda por requisitos nessa aplicação. Exemplos de usuários: pessoa física, um outro aplicativo, um hardware, um ator em um caso de uso, gestores do negócio que o software atende.

3.2.1. Benefícios da APF segundo Vasquez [5]:

- Uma ferramenta para determinar o tamanho de um pacote adquirido através da **contagem de todas as funções incluídas**.
- Provê auxílio aos usuários na determinação dos benefícios de um pacote para sua organização, através da contagem **das funções que especificamente correspondem aos seus requisitos**. Ao avaliar o custo do pacote, o tamanho das funções que serão efetivamente utilizadas, a produtividade e o custo da própria equipe, é possível realizar uma análise do tipo *make or buy*.

- Suporta a **análise de produtividade e qualidade**, seja diretamente ou em conjunto com outras métricas como esforço, defeitos e custo. Porém, se o processo de desenvolvimento da organização for caótico (cada projeto é desenvolvido de forma diferente), mesmo que a contagem de pontos de função do projeto e o registro do esforço tenham sido feitos de forma correta, a análise de produtividade entre os projetos será prejudicada.
- Apóia o **gerenciamento de escopo** de projetos. Um desafio de todos os gerentes de projeto é controlar o *scope creep*, ou aumento de seu escopo. Ao realizar as estimativas e medições dos pontos de função do projeto em cada fase do seu ciclo de vida, é possível determinar se os requisitos funcionais cresceram ou diminuíram, e se essa variação corresponde a novos requisitos ou a requisitos já existentes e que foram apenas mais detalhados.
- Complementa o **gerenciamento dos requisitos** ao auxiliar na verificação da solidez e completeza dos requisitos especificados. O processo de contagem de pontos de função favorece uma análise sistemática e estruturada da especificação de requisitos e traz benefícios semelhantes a uma revisão em pares do processo.
- Um meio de **estimar custo e recursos** para o desenvolvimento e manutenção de software. Por meio da realização de uma contagem ou estimativa de pontos de função no início de vida de um projeto de software, é possível determinar seu tamanho funcional. Essa medida pode ser então utilizada como entrada para diversos modelos de estimativa de esforço, prazo e custo.
- Uma ferramenta para **fundamentar a negociação de contratos**. Podem-se utilizar pontos de função para gerar diversos indicadores de níveis de serviço (SLA – *Service Level Agreement*) em contratos de desenvolvimento e manutenção de sistemas. Além disso permite o estabelecimento de contratos a preço unitário – pontos de função – em que a unidade representa um bem tangível para o cliente. Essa modalidade possibilita uma melhor distribuição de riscos entre o cliente e o fornecedor. O capítulo 10 discute essa abordagem em detalhes.
- Um fator de **normalização para comparação de software** ou para comparação da produtividade na utilização de diferentes técnicas. Diversas organizações, como o ISBSG, disponibilizam um repositório de dados de projetos de software que permitem a realização de *benchmarking* com projetos do mercado.

3.2.2. A APF baseia-se em 3 pontos básicos:

3.2.2.1. Arquivos:

- Arquivos Lógicos Internos (ALI): São os arquivos mantidos pela aplicação (inserção, exclusão, atualização). Não se deve pensar impulsivamente que uma tabela de um banco de dados relacional é igual a um ALI na APF de um produto. Pode ocorrer que o agrupamento de duas ou mais tabelas forme um único ALI. É necessário uma análise do profissional que for fazer a contagem que identifique os ALIs de forma correta. Exemplo: Tabelas de dados atualizadas pela aplicação.
- Arquivos de Interface Externa (AIE): São os arquivos não mantidos pela aplicação, mas que são referenciados em consultas e relatórios da aplicação. Também se deve-se ter cuidado na identificação pois não é sempre válida a relação uma tabela do banco um AIE. Exemplo: Tabelas de banco de dados lidas pela aplicação, mas atualizadas por outra aplicação.

3.2.2.2. Entradas:

- Entrada Externa (EE): São ações que resultam em inserção, alteração ou exclusão de dados em um ALI da aplicação. Toda tela de cadastro, alteração ou exclusão de dados são exemplos de EE.

3.2.2.3. Saídas:

- Consulta Externa (CE). São ações que enviam dados ou informações de controle para fora da fronteira da aplicação. Sua função é mostrar informações ao usuário através da recuperação de dados de ALI e/ou AIE Exemplos de CE: Relatório sem totalização ou sem persistência de informações no ALI da aplicação, Geração de um arquivo de texto (.txt) que não contenha cálculo.
- Saída Externa (SE). São ações que enviam dados ou informações de controle para fora da fronteira da aplicação, mas que utilizam lógica de

processamento e não apenas simples recuperação de informações de ALI e/ou AIE. Seu processamento deve conter cálculo, ou criar dados derivados, ou manter registros em ALI(s) ou alterar o comportamento de um sistema. Exemplo: Tela de Login, relatórios de totais de faturamento por fornecedor, relatórios que geram logs consultáveis por um processo de auditoria, geração de um arquivo que contenha cálculo realizado pela aplicação.

Esses três pontos básicos são medidos pela sua complexidade, assim representados:

Para os ALI e AIE:

		Tipo de Dados (TD)		
		<20	20-50	>50
Tipo de Registro (TR)	1	Baixa	Baixa	**Média**
	2-5	Baixa	**Média**	Alta
	>5	**Média**	Alta	Alta

Para as SE e CE

		Tipo de Dados (TD)		
		<6	6-19	>19
Arquivos Referenciados (AR)	<2	Baixa	Baixa	**Média**
	2-3	Baixa	**Média**	Alta
	>3	**Média**	Alta	Alta

Nota: Não há uma SE ou uma EE sem existir pelo menos 1 arquivo referenciado, seja ele um ALI ou um AIE.

Para as EE

		Tipo de Dados (TD)		
		<5	5-15	>15
Arquivos Referenciados (AR)	<2	Baixa	Baixa	Média
	2	Baixa	Média	Alta
	>2	Média	Alta	Alta

Nota: Não há uma EE sem existir pelo menos 1 arquivo referenciado, seja ele um ALI ou um AIE.

Contribuição dos Pontos de Função baseados em sua complexidade e por tipo de arquivo ou ação elementar:

Tipo	Baixa	Média	Alta
ALI	7	10	15
AIE	5	7	10
EE	3	4	6
SE	4	5	7
CE	3	4	6

3.2.3. O Fator de Ajuste

A APF também utiliza um componente chamado **Fator de Ajuste** opcional segundo o padrão ISO/IEC de medição funcional. Sua finalidade é ajustar os pontos de função não-ajustados em +-35% devido a dados coletados em 14 características gerais do sistema (CGS) e seus níveis de influência.

O nível de influência é categorizado em valores de 0 a 5, onde:

0. Nenhuma Influência
1 Influência Mínima
2 Influência Moderada
3 Influência Média
4 Influência Significativa
5 Grande Influência

Segundo Vasquez [5], o fator de ajuste esta ligado aos seguintes pontos ou 14 Características Gerais do Sistema (CGS):

3.2.3.1. Comunicação de Dados

Descreve o nível em que a aplicação comunica-se diretamente com o processador. Os *dados* ou *informações de controle* utilizados pela aplicação são enviados ou recebidos por meio de recursos de comunicação. Terminais conectados localmente à unidade de controle são considerados recursos de comunicação. Protocolo é um conjunto de convenções que permite a transferência ou intercâmbio de informações entre dois sistemas ou dispositivos. Todos os links de comunicação necessitam de algum tipo de protocolo.

Pontue de acordo com as características abaixo

0 – A aplicação é puramente *batch* ou uma estação de trabalho isolada

1 – A aplicação é puramente *batch*, mas possui entrada de dados ou impressão remota.

2 – A aplicação é *batch*, mas possui entrada de dados e impressão remota.

3 – A aplicação possui entrada de dados *on-line*, *front-end* de teleprocessamento para um processamento *batch* ou sistema de consulta.

4 – A aplicação é mais que um *front-end*, mas suporta apenas um tipo de protocolo de comunicação.

5 – A aplicação é mais que um *front-end*, e suporta mais de um tipo de protocolo de comunicação.

Atualmente aplicações típicas de cliente/servidor com duas camadas pontuam em 4. Aplicações de tempo real, servidores de aplicação e *middlewares* pontuam em 5.

3.2.3.2. Processamento Distribuído

Descreve o nível em que a aplicação transfere dados entre seus componentes. Funções ou dados distribuídos dentro da fronteira são características da aplicação:

Pontue de acordo com as características abaixo

0 – A aplicação não participa da transferência de dados ou processamento de funções entre os componentes do sistema

1 – A aplicação prepara dados para processamento pelo usuário final em outro componente do sistema, como planilhas eletrônicas ou banco de dados.

2 – Dados são preparados para transferência, então são processados em outro componente do sistema (não para processamento pelo usuário final).

3 – Processamento distribuído e transferência de dados são feitos *on-line* e em apenas uma direção.

4 – Processamento distribuído e transferência de dados são feitos *on-line* e em ambas direções.

5 – O processamento de funções é executado dinamicamente no componente mais apropriado do sistema.

Atualmente em sistemas *desktop* isolados pontuam com 0. Um sistema de n camadas pontuará com 4. Para pontuar com 5 deveria os componentes serem executados em múltiplos processadores ou servidores, sendo cada um deles selecionado dinamicamente de acordo com a disponibilidade.

3.2.3.3. Performance

Descreve o nível em que considerações sobre tempo de resposta e taxas de transações influenciam o desenvolvimento da aplicação. Os objetivos estabelecidos ou aprovados pelo usuário, em termos de tempo de resposta ou taxa de transações, influenciam (ou influenciará) o projeto, desenvolvimento, instalação e suporte da aplicação.

Pontue de acordo com as características abaixo

0 – O usuário não estabeleceu nenhum requisito especial sobre performance.

1 – Requisitos de performance e projeto foram estabelecidos e revisados, mas nenhuma ação em especial foi tomada.

2 – Tempo de resposta ou taxa de transações são críticos durante as horas de pico. Não é necessário nenhum projeto especial para utilização de CPU. O limite para o processamento é o dia seguinte.

3 – Tempo de resposta ou taxa de transações são críticos durante todas as horas de trabalho. Não foi necessário nenhum projeto especial para a utilização de CPU. O limite de processamento é crítico.

4 – Adicionalmente, requisitos especificados pelo usuário são exigentes o bastante para que tarefas de análise de performance sejam necessárias na fase de projeto.

5 – Adicionalmente, ferramentas de análise de performance devem ser utilizadas nas fases de projeto, desenvolvimento e/ou implementação para que os requisitos de performance do usuário sejam atendidos.

3.2.3.4. Configuração Altamente Utilizada

Descreve o nível em que restrições de recursos computacionais influenciam no desenvolvimento da aplicação. Uma configuração operacional altamente utilizada, necessitando de considerações especiais de projeto, é uma característica da aplicação. Por exemplo, o usuário deseja executar a aplicação em um equipamento já existente ou comprado e que será altamente utilizado.

Pontue de acordo com as características abaixo

0 – Não existem restrições operacionais implícitas ou explícitas nos requisitos.

1 – Existem restrições operacionais, mas são menos restritivas que uma aplicação típica. Não há esforço especial necessário ao atendimento dessas restrições.

2 – Algumas considerações de sincronismo ou segurança são especificadas.

3 – Existem requisitos específicos de processador para uma parte específica da aplicação.

4 – Restrições operacionais explícitas necessitam de um processador dedicado ou utilização pesada do processador central.

5 – Adicionalmente, existem limitações nos componentes distribuídos da aplicação

Em geral, a grande maioria dos sistemas pontuará em 0 ou 1. Aplicações cientificas ou de engenharia com grandes exigências de processamento pontuariam de 3 a 5.

3.2.3.5. Volume das transações

Descreve em que nível o alto volume de transações influencia o projeto, desenvolvimento, instalação e suporte da aplicação.

Pontue de acordo com as características abaixo

0 – Não é antecipado nenhum período de pico de transações.

1 – São antecipados períodos de pico de processamento (por exemplo: mensal, quinzenal, periódico, anual).

2 – Picos de transação semanais são previstos.

3 – Picos de transação diários são previstos.

4 – Altas taxas de transação definidas pelo usuário nos requisitos ou os níveis de serviço acordados (SLA) são altos o bastante para requererem tarefas de análise de performance na fase de projeto.

5 – Adicionalmente, existem requisitos de ferramentas de análise de performance nas fases de projeto, desenvolvimento e/ou instalação.

Essa característica está ligada diretamente com a número 3. Um exemplo de aplicação com picos diários de transação é um sistema de controle de ponto de funcionários.

3.2.3.6. Entrada de Dados On-Line

Descreve em que nível são efetuadas entradas de dados na aplicação por meio de transações interativas.

Pontue de acordo com as características abaixo

0 – Todas as transações são processadas em lote.

1 – De 1% a 7% das transações são entradas on-line.

2 – De 8% a 15% das transações são entradas on-line.

3 – De 16% a 23% das transações são entradas on-line.

4 – De 24% a 30% das transações são entradas on-line.

5 – Mais de 30% das transações são entradas on-line.

Em geral, as aplicações atuais pontuam em 5. Isso é um Fator de Ajuste defasado, pois atualmente 100% das entradas de dados são on-line. Futuramente poderá ser um fator revisto pelo IFPUG.

3.2.3.7. Eficiência do Usuário Final

Descreve em que nível considerações sobre fatores humanos e facilidade de uso pelo usuário final influenciam o desenvolvimento da aplicação. As funções interativas fornecidas pela aplicação enfatizam um projeto para o aumento da eficiência do usuário final. O projeto inclui:

- Auxilio para navegação, como, por exemplo, teclas de função, saltos, menus gerados dinamicamente;
- Menus;
- Ajuda *on-line* e documentação;
- Movimentação automática de cursor;
- Paginação;
- Impressão remota por meio de transações *on-line*;

- Seleção feita por posicionamento de cursor em tela de dados;
- Uso intenso de vídeo reverso, brilho e cores e outros indicadores;
- Documentação impressa das transações;
- Interface de mouse;
- Janelas *pop-up*;
- Utilização de número mínimo de telas para executar uma função do negócio;
- Suporte a dois idiomas (conte como quatro itens);
- Suporte a mais de dois idiomas (conte como seis itens).
-

Pontue de acordo com as características abaixo

0 – Nenhum dos itens anteriores

1 – De um a três dos itens anteriores.

2 – De quatro a cinco dos itens anteriores.

3 – Seis ou mais itens anteriores, mas não existem requisitos específicos do usuário associados à eficiência.

4 – Seis ou mais dos itens anteriores e requisitos explícitos sobre eficiência para o usuário final são fortes o bastante para necessitarem de tarefas de projeto que incluam fatores humanos, como, por exemplo, minimizar o número de toques do teclado, maximizar padrões de campo e uso de modelos.

5 – Seis ou mais dos itens anteriores e requisitos explícitos sobre a eficiência para o usuário final são fortes o bastante para necessitarem do uso de ferramentas e processos especiais para demonstrar que os objetivos foram alcançados.

Aplicações típicas de GUI pontuaram de 3 a 5. Aplicações em interferência de ações do usuário ou *batch* pontuam com 0. Futuramente poderá ser um fator revisto pelo IFPUG, pois são diretrizes defasadas.

3.2.3.8. Atualização On-Line

Descreve em que nível os arquivos lógicos internos da aplicação são atualizados de forma on-line.

Pontue de acordo com as características abaixo

0 – Não há nenhuma atualização *on-line*.

1 – Existe a atualização *on-line* de um a três arquivos. Volume de atualização é pequeno e a recuperação é fácil.

2 – Existe a atualização *on-line* de quatro ou mais arquivos. Volume de atualização é pequeno e a recuperação é fácil.

3 – A atualização da maioria dos arquivos internos é *on-line*.

4 – Adicionalmente, a projeção contra a perda de dados é essencial e foi especialmente projetada e programada no sistema.

5 – Adicionalmente, o alto volume de processamento torna necessária a análise do custo do processamento de recuperação. São incluídos procedimentos altamente automatizados com um mínimo de intervenção do operador.

Em geral, as aplicações atuais pontuam em no mínimo com 3, exceto aplicações *batch* que pontuam com 0. Isso é um Fator de Ajuste defasado. Futuramente poderá ser um fator revisto pelo IFPUG.

3.2.3.9. Complexidade de Processamento

Descreve em que nível o processamento lógico ou matemático influencia o desenvolvimento da aplicação. Os seguintes componentes estão presentes:

- Controle sensível e/ou processamento específico de segurança da aplicação. Exemplo: processamento especial de auditoria
- Processamento lógico extensivo. Exemplo: sistema de gestão de crédito.
- Processamento matemático extensivo. Exemplo: sistema de otimização de corte de tecidos.
- Muito processamento de exceção resultando em transações incompletas que devem ser processadas novamente. Exemplo: transações incompletas em ATM em função de problemas de teleprocessamento, falta de dados ou de edição.
- Processamento complexo para manipular múltiplas possibilidades de entrada e saída, como, por exemplo, multimídia, ou independência de dispositivos. Exemplo: sistema de extrato de conta corrente que emite via terminal de retaguarda, auto-atendimento, *web*, *e-mail*, telefone celular.

Pontue de acordo com as características abaixo

0 – Nenhum dos itens anteriores.

1 – Qualquer um dos itens anteriores.

2 – Quaisquer dois itens anteriores.

3 – Quaisquer três itens anteriores.

4 – Quaisquer quatro itens anteriores.

5 – Todos os cinco itens anteriores.

Como nas outras características gerais do sistema, deve-se avaliar não apenas uma funcionalidade específica, mas o geral da aplicação. O fato de haver uma funcionalidade com grande processamento matemático extensivo, como uma apropriação financeira de encargos ou o levantamento de um saldo devedor, deve ser considerado no contexto do sistema como um todo. Nesse caso, onde uma parte considerável do processamento envolve esse tipo de lógica, deve-se considerar esse componente como presente. Agora, caso seja uma aplicação em que esses processamentos sejam periféricos e constituam uma pequena parte do conjunto total, deve-se ponderar para menos o impacto na aplicação como um todo.

3.2.3.10. Reusabilidade

A aplicação e seu código foram especificamente projetados, desenvolvidos e suportados para serem utilizados em outras aplicações.

Pontue de acordo com as características abaixo

0 – Não há código reutilizável.

1 – Código reutilizável é utilizado na aplicação.

2 – Menos de dez por cento da aplicação levou em consideração as necessidades de mais de um usuário.

3 – Dez por cento ou mais da aplicação levou em consideração as necessidades de mais de um usuário.

4 – A aplicação foi especificamente empacotada e/ou documentada para fácil reutilização. Ela é customizada pelo usuário no nível de código.

5 – A aplicação foi especificamente empacotada e/ou documentada para fácil reutilização. Ela é customizada pelo usuário por meio de manutenção de parâmetros.

Essa CGS é um requisito de qualidade de software, não um requisito funcional. É difícil (senão impossível) ser avaliada para uma aplicação em que não estão disponíveis documentos do projeto ou o seu código-fonte. Além disso, possui considerações técnicas que contrariam o objetivo principal da técnica que é medir o software do ponto de vista do usuário pela funcionalidade fornecida.

3.2.3.11. Facilidade de Instalação

Descreve em que nível a conversão de ambientes preexistentes influencia o desenvolvimento da aplicação. Um plano e/ou ferramentas de conversão e instalação foram fornecidos e testados durante a fase de testes do sistema..

Pontue de acordo com as características abaixo

0 – O usuário não definiu considerações especiais, assim como não é requerido nenhum *setup* para a instalação.

1 – O usuário não definiu considerações especiais, mas é necessário *setup* para a instalação.

2 – Requisitos de instalação e conversão foram definidos pelo usuário, e guias de conversão e instalação foram fornecidas e testadas. Não é considerado importante o impacto da conversão.

3 – Requisitos de instalação e conversão foram definidos pelo usuário, e guias de conversão e instalação foram fornecidas e testadas. É considerado importante o impacto da conversão.

4 – Além do item 2, ferramentas de instalação e conversão automáticas foram fornecidas e testadas.

5 – Além do item 3, ferramentas de instalação e conversão automáticas foram fornecidas e testadas..

Essa CGS é importante em projetos de sistemas que irão substituir aplicações existentes. Pontuando entre 3 a 5 nesse caso.

3.2.3.12. Facilidade de Operação

Descreve em que nível a aplicação atende a alguns aspectos operacionais, como: inicialização, segurança e recuperação. A aplicação minimiza a necessidade de atividades manuais, como montagem de fitas, manipulação de papel e intervenção manual pelo operador.

Pontue de acordo com as características abaixo

0 – Não foi estabelecida pelo usuário outra consideração que não os procedimentos de segurança normais.

1 -4– Um, alguns ou todos os itens seguintes são válidos para a aplicação. Selecione todos aqueles que sejam válidos. Cada item tem um valor de um ponto, a exceção onde seja citado o contrário.

- Procedimentos de inicialização, salvamento e recuperação foram fornecidos, mas é necessária a intervenção do operador.
- Procedimentos de inicialização, salvamento e recuperação foram fornecidos, mas não é necessária a intervenção do operador (conte como dois itens)
- A aplicação minimiza a necessidade de montagem de fitas
- A aplicação minimiza a necessidade de manipulação de papel

5 – Aplicação projetada para operação não assistida. Isto é, não é necessário *nenhuma intervenção do operador* para operar o sistema, que não seja a inicialização e termino da aplicação. A aplicação recupera-se automaticamente de erros.

Típica CGS defasada. A pontuação dos sistemas atuais é 5.

3.2.3.13. Múltiplos Locais

A aplicação foi especificamente projetada, desenvolvida e suportada para instalação em múltiplos locais para múltiplas organizações.

Pontue de acordo com as características abaixo

0 – Os requisitos do usuário não consideram a necessidade de mais de um usuário/local de instalação.

1 – Necessidade de múltiplos locais foi considerada no projeto, e a aplicação foi projetada para operar apenas *nos mesmos* ambientes de hardware e software.

2 – Necessidade de múltiplos locais foi considerada no projeto, e a aplicação foi projetada para operar em apenas ambientes de hardware e de software *similares*.

3 – Necessidade de múltiplos locais foi considerada no projeto, e a aplicação foi projetada para operar em ambientes *diferentes* de hardware e de software.

4 – Adicionalmente nos itens 1 ou 2, plano de suporte e documentação são fornecidos e testados para suportar a aplicação em múltiplos locais.

5 – Adicionalmente ao item 3, plano de suporte e documentação são fornecidos e testados para suportar a aplicação em múltiplos locais.

Exemplos de ambientes de software similares: Windows XP, Windows Vista. Exemplos de ambientes de hardware similares: Pentium Core Duo, Pentium Core 2 Duo.

3.2.3.14. Facilidade de Mudanças

Descreve em que nível a aplicação foi especificamente desenvolvida para facilitar a mudança de sua lógica de processamento ou estrutura de dados.

As seguintes características podem ser válidas para a aplicação:

- São fornecidos mecanismos de consulta flexível, que permitem a manipulação de pedidos simples; por exemplo, lógica de *e/ou* aplicada a apenas um arquivo lógico (conte como um item)
- São fornecidos mecanismos de consulta flexível, que permitem a manipulação de pedidos simples; por exemplo, lógica de *e/ou* aplicada a mais de um arquivo lógico (conte como dois item)
- São fornecidos mecanismos de consulta flexível, que permitem a manipulação de pedidos complexos; por exemplo, lógica de e/ou combinadas em um ou mais arquivos lógicos (conte como três item)
- Dados de controle do negócio são mantidos pelo usuário por meio de processos interativos, mas as alterações só tem efeito no próximo dia útil.
- Dados de controle do negócio são mantidos pelo usuário por meio de processos interativos, e as alterações têm efeito imediato (conte como dois itens).

Pontue de acordo com as características abaixo

0 – Nenhum dos itens anteriores.

1 – Qualquer um dos itens anteriores.

2 – Quaisquer dois itens anteriores.

3 – Quaisquer três itens anteriores.

4 – Quaisquer quatro itens anteriores.

5 – Todos os cinco itens anteriores.

Existem dois tipos de componentes que estão avaliados nesta CGS: mecanismos de consulta flexível e manutenção de dados de controle do sistema. O primeiro reflete consultas em que o próprio usuário monta relatórios a partir dos dados disponíveis no sistema. O segundo é referente à manutenção de parâmetros de forma *on-line* por meio de manutenções de tabelas, por exemplo.

3.2.4. Como classificar as funções do tipo transação

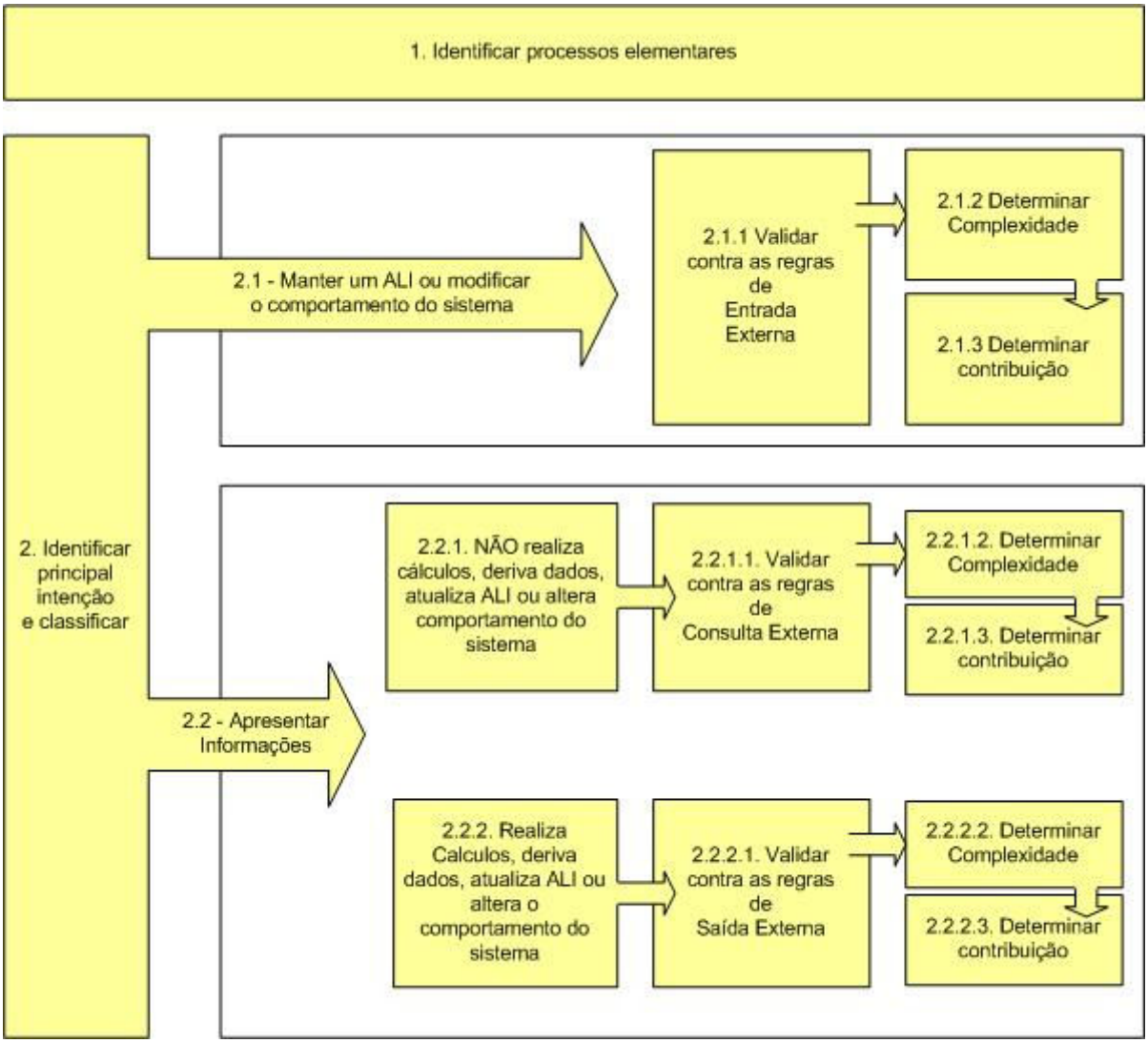


Figura 5 - Classificação do Processo de Contagem

Na figura acima, note que no item 2 (Identificar principal intenção e classificar), a seta há uma divisão. No item 2.1 é guiado para quantificar e qualificar as EE, já no item 2.2 é guiado para as ações que mostram dados e realizam cálculo ou não realizam cálculo. Essa divisão é a principal classificação entre saída externa e consulta externa, respectivamente.

Tabela 1 - Tabela rápida de classificação de ações em EE, CE e SE.

Tipo de Lógica de Processamento	EE	SE	CE
[01] Validações	Pode	Pode	Pode
[02] Cálculos e formulas matemáticas	Pode	Deve*	Não
[03] Conversão em valores equivalentes	Pode	Pode	Pode
[04] Filtro e seleção de dados com base em critérios específicos na comparação de vários conjuntos de dados	Pode	Pode	Pode
[05] Análise de condições para que determinem quais se aplicam	Pode	Pode	Pode
[06] Atualização de pelo menos um ALI	Deve*	Deve*	Não
[07] Referencia a pelo menos um ALI ou AIE	Pode	Pode	Deve
[08] Recuperação de dados ou informações de controle	Pode	Pode	Deve
[09] Criação de dados derivados	Pode	Deve*	Não
[10] Alteração do comportamento do sistema	Deve*	Deve*	Não
[11] Preparação e apresentação de informação para fora da fronteira	Pode	Deve	Deve
[12] Capacidade de aceitar dados ou inf. De controle que entra pela fronteira	Deve	Pode	Pode
[13] Mudança da Ordenação ou organização de um conjunto de dados - apenas diferenças em lógica de ordenação não é suficiente para garantir unicidade de processo elementar	Pode	Pode	Pode

Deve – A transação deve obrigatoriamente executar este tipo de lógica de processamento.

Deve* - A transação deve executar pelo menos uma das lógicas de processamento classificadas como deve*.

Pode – A transação pode executar este tipo de lógica de processamento, mas não é obrigatório.

Não – A transação não pode executar este tipo de lógica de processamento.

3.2.5. Tipos de Contagem

A literatura nos indica três tipos de contagens de pontos de função: A Indicativa, a Estimada e a Detalhada.

3.2.5.1. Contagem Indicativa ou Dedutiva.

Essa técnica, segundo Vasquez [5], considera a possibilidade de contagem de um único componente da análise de pontos de função para a aplicação, geralmente o número de Arquivos Lógicos Internos, derivando o resto da contagem a partir de bases estatísticas.

No trabalho *Early Function Point Counting*, publicado pela NESMA[1], encontra-se descrita uma abordagem para este método. Trata-se da Contagem Indicativa, na qual apenas é necessária a identificação dos Arquivos de Interface Externa (AIE) e Arquivos Lógicos Internos (ALI). Considera-se 35 PF para cada ALI e 15 PF para cada AIE identificado.

Os números 35 e 15 representam as médias de pontos de função identificada para cada um dos tipos de arquivo, considerando projetos que, em média, possuam três Entradas Externas (EE), duas Saídas Externas (SE) e uma Consulta Externa (CE) para cada ALI e uma Saída Externa e uma Consulta Externa para cada AIE.

3.2.5.2. Contagem Estimada ou por Estimativa

Após toda a identificação das funcionalidades do software (EE,SE e CE) e dos arquivos envolvidos (ALI e AIE), utiliza-se a classificação de complexidades do IFPUG e aplica-se a complexidade baixa para cada Arquivo Lógico Interno e Arquivo de Interface Externa e média para cada Entrada Externa, Saída Externa ou Consulta Externa. A estimativa do tamanho é então obtida pela fórmula:

$$E = \#EE \times 4 + \#SE \times 5 + \#CE \times 4 + \#ALI \times 7 + \#AIE \times 5$$

[1] http://www.nesma.nl/english/earlyfpa.htm

3.2.5.3. Contagem Detalhada

A contagem detalhada é na verdade a contagem final de um projeto, pois é analisado tela a tela e relatório a relatório do aplicativo, bem como os arquivos internos e externos do aplicativo.

Na contagem detalhada os arquivos (ALI e AIE), a entrada de dados (EE) e as consultas (SE e CE) são detalhadas em complexidade baixa, média ou alta. E são pontuados de acordo com o valor da complexidade definida pelo IFPUG.

A contagem detalhada é a contagem que realmente autoriza o valor real do aplicativo e normalmente é a contagem que efetivamente remunera contratos de terceirização de produção de software baseado em APF.

3.2.5.4. Tipo de Aplicação para definir o Tipo de Contagem

A aplicação segundo o IFPUG é "uma aplicação é um conjunto coeso de dados e procedimentos automatizados que suportam um objetivo de negócio, podendo consistir de um ou mais componentes, módulos ou subsistemas. Frequentemente, o termo "aplicação" é utilizado como sinônimo para sistema, sistema aplicativo ou sistema de informação".

Os projetos são normalmente classificados em projeto de desenvolvimento e projetos de melhoria.

Projeto de desenvolvimento – o número de pontos de função de um projeto de desenvolvimento mede a funcionalidade fornecida aos usuários finais do software quando da sua primeira instalação. Isso significa que essa contagem também abrange as eventuais funções de conversão de dados necessárias à implantação do sistema, segundo Vasquez [5].

Entenda-se conversão de dados o processo de migração de dados em sistemas pré-existentes.

Projeto de melhoria – O número de pontos de função de um projeto de melhoria mede as funções adicionadas, modificadas ou excluídas do sistema pelo projeto, e também as eventuais funções de conversão de dados, conforme Vasquez [5].

Ainda segundo Vasquez [5], quando o projeto de melhoria é concluído e seus produtos instalados, o número de pontos de função da aplicação deve ser atualizado para refletir as alterações na funcionalidade da aplicação.

3.2.6. Adaptações ao estudo da APF

Muitas empresas que fabricam seu próprio software ou empresas que fornecem software não seguem o modelo de contagem à risca.

Um **projeto de desenvolvimento** segue a seguinte fórmula

DPF = (UPF + CPF) x VAF

Onde:

DPF = Número de pontos de função do projeto de desenvolvimento
UPF = Número de pontos de função não ajustados das funções disponíveis após a instalação.
CPF = Número de pontos de função não ajustados das funções de conversão. Essas funções são disponíveis no momento da instalação da aplicação para converter dados ou fornecer outros requisitos de conversão especificados pelo usuário, como relatório de verificação de conversão. Após a instalação essas funções são descartadas. Deve ficar claro que as CPF não fazem parte do número de pontos de função da aplicação instalada.
VAF = Valor do fator de ajuste

Um **projeto de melhoria** segue a seguinte fórmula

DPF = [(ADD + CHGA + CFP) x VAFA] + (DEL x VAFB)

Onde:

DPF = Número de pontos de função do projeto de melhoria
ADD = Número de pontos de função não ajustados das funções incluídas pelo projeto de melhoria
CHGA = Número de pontos de função não ajustados das funções modificadas. Reflete as funções depois das modificações.

CPF = Número de pontos de função não ajustados adicionados pela conversão.
VAFA = Valor do fator de ajuste da aplicação depois do projeto de melhoria
DEL = Número de pontos de função não ajustados das funções excluídas pelo projeto de melhoria
VAFB = Valor do fator de ajuste da aplicação antes do projeto de melhoria

Após um projeto de melhoria ser realizado, a AFP do Projeto original já desenvolvido deve ser adicionada da DPF do projeto de melhoria. Pois isto é o novo valor de pontos de função do aplicativo. Também deve ser excluído do total da contagem do projeto as funções excluídas.

A fórmula da Contagem Inicial, que representa a aplicação instalada após o seu desenvolvimento e antes de um projeto de melhoria, é assim visualizada:

AFP=ADD x VAF

AFP = Número de pontos de função ajustados da aplicação
ADD = Pontos de função não ajustados das funções instaladas.
VAF = Valor do fator de ajuste da aplicação baseado nas CGS.

Esta formula indica o tamanho da aplicação, quando não há conversão de dados, a AFP da aplicação é igual ao DPF do projeto de desenvolvimento

A fórmula após o Projeto de Melhoria é assim visualizada:

AFP = [(UFPB+ADD+CHGA)-(CHGB+DEL)]xVAFA

Onde:

AFP = Número de pontos de função ajustado da aplicação
UFPB = Pontos de função não ajustados da aplicação antes do projeto de melhoria.
ADD = Pontos de função não ajustados das funções incluídas pelo projeto de melhoria.

CHGA = Pontos de função não ajustados das funções alteradas pelo projeto de melhoria depois de seu término.
GHGB = Pontos de função não ajustados das funções alteradas pelo projeto de melhoria antes de seu término.
DEL = Pontos de função não ajustados das funções excluídas pelo projeto de melhoria.
VAFA = Valor do fator de ajuste depois do projeto de melhoria.

Quando um projeto de melhoria é concluído, segundo Vasquez [5], o número de pontos de função da aplicação deve ser atualizado para refletir as modificações na aplicação. Deve-se destacar novamente que mesmo que o projeto da melhoria tenha funções de conversão de dados, elas não fazem parte da aplicação.

Daí se infere que:

1. Nova funcionalidade aumenta o tamanho da aplicação.
2. Mudanças do fator de ajuste pode aumentar, diminuir ou não afetar o tamanho da aplicação
3. Funcionalidade modificada pode aumentar, diminuir ou não alterar o tamanho do software
4. Funcionalidade excluída diminui o tamanho da aplicação.

Mas em análise a essas fórmulas, as mudanças e retirada de funções já existentes, normalmente são feitas por um índice deflator. Este índice deflator serve para evitar que o custo de uma mudança seja igual aos da construção e para lidar com o ônus da exclusão de uma funcionalidade. Normalmente este índice é debatido entre cliente e fornecedor até que se chegue a um índice apropriado para ambos lados. Caso não seja possível o uso do índice deflator, utiliza-se apenas o que a literatura oferece, ou seja, projeto de melhoria com valores integrais.

Pode-se também identificar um índice inflator para casos que a tecnologia não seja algo conhecido pela equipe, ou seja, muito nova.

4. MÉTRICAS DE QUALIDADE DE SOFTWARE

Conforme encontrado na vasta literatura de métricas a utilização destas no processo de desenvolvimento de software, diminuem os defeitos, prazos de entrega, desperdício e custos. Conseqüentemente aumenta a satisfação do cliente, produtividade dos recursos, visibilidade das ações e qualidade do gerenciamento.

Segundo Maia [6] existem duas classificações gerais para métricas, as primitivas e as derivadas, onde as métricas primitivas são aquelas medidas diretamente, mas de pouco significado numa interpretação isolada. As métricas derivadas seriam os atributos mensuráveis, que são obtidos por cálculo a partir de métricas primitivas, além disso, estabelece que as métricas podem ser divididas em: métricas de qualidade, métricas de produtividade, métricas orientadas por função e métricas orientadas a objeto.

Conforme Kan [2], as métricas de software podem ser mais abrangentes e com isto podem ser classificadas em 3 categorias:

- Métricas de produto
- Métricas de processo
- Métricas de projeto

Existem ainda diversas outras classificações para métricas, porém estas duas definições acima foram suficientes para o entendimento da aplicação de métricas no processo de desenvolvimento de softwares.

4.1. MÉTRICAS DE QUALIDADE DO PRODUTO

A definição de qualidade de software consiste em dois níveis: A Qualidade intrínseca do produto e a Satisfação do cliente, onde a qualidade intrínseca do produto usualmente é medida pelo número de *"bugs"* (defeitos funcionais) no software, e o tempo levado até a ocorrência de um *crash*.

As duas métricas que se aplicam melhor na avaliação destas duas situações são: A taxa de densidade de defeitos e o tempo médio de falha (MTTF – *Mean Time to Failure*).

O MTTF é muito utilizado em sistemas críticos, como por exemplo: Sistemas de controle de tráfego aéreo, ou sistemas que envolvam vidas. Um exemplo de caso prático que pode ser citado é a determinação do governo Americano para que o controle de tráfego aéreo não

pudesse ficar indisponível por 3 segundos por ano, procurando desta forma, evitar catástrofes. Já a Métrica de densidade de defeitos, em contraste, é usada em vários sistemas comerciais.

Estas duas métricas estão correlacionadas, porém possuem definições muito diferentes. Uma mede o tempo entre falhas e a outra mede os defeitos relativos ao tamanho do software, que pode ser expresso por SLOC (*Source lines of code*), PF (Pontos de função), etc.

Outro ponto importante é a dificuldade em separar defeitos e falhas nas medidas efetuadas. Para esclarecer melhor esta diferença existem algumas definições de erro, defeito e falha segundo o IEEE/ANSI 982.2 [7]:

- Um erro é um lapso humano que resulta em um software incorreto.
- Uma falha resultante é uma condição acidental que leva uma unidade do sistema a falhar para uma função requerida.
- O defeito ou avaria é uma anomalia do produto.
- Uma falha (*Failure*) ocorre quando uma unidade funcional do software não executa por um período de tempo, por requerer uma função que não executa com os limites especificados.

A representação da relação entre estas definições encontra-se na figura abaixo:

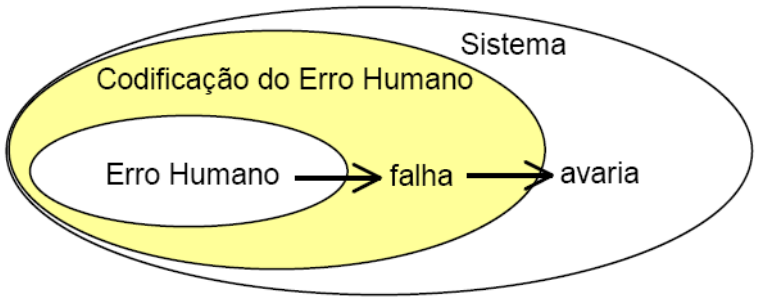


Figura 6 - Relacionamento entre Erro, Falha e Avaria

Na prática pode-se afirmar que não há diferenças entre estes termos, pois em várias organizações eles são bastante utilizados como sinônimos.

4.1.1. Métrica de densidade de Defeitos

A métrica de densidade de defeitos pode ser expressa através de uma taxa onde o numerador é o número de erros encontrados em um software e o denominador é o tamanho do software, usualmente expresso em SLOC ou PF. Juntamente com a taxa é importante lembrar que este valor é válido em um determinado período de tempo (*time frames*).

Esta métrica é muito utilizada por fábricas de software, pois a mesma dá uma visão do que realmente o produto vem a ser, em termos de qualidade, inclusive as fábricas utilizam certos limites de aceitação, e só liberam o produto caso ele tenha uma densidade de defeitos abaixo deste limite. Uma situação ideal seria ter 0 erros em um software e com isto a densidade também seria 0, mas isto é muito difícil de acontecer, pois geralmente o ambiente e as regras de negócio são muito complexos, o que leva a existirem erros, mesmo que não identificados nas fases do processo de desenvolvimento. Desta forma, podem ser aceitos softwares com um determinado nível de erros, desde que estes não impactem diretamente na arquitetura e no negócio em questão.

4.1.2. Perspectiva do cliente

Conforme Kan [2], a densidade de defeitos é uma boa métrica para avaliar a qualidade do desenvolvimento do software numa visão focada na equipe de desenvolvimento. Desta forma vale como boa prática também considerar a perspectiva do cliente como métrica utilizada nas disponibilizações de versões.

Exemplo: Em um processo iterativo e incremental, são disponibilizadas para o cliente várias versões, onde cada uma contém partes de todo o software encomendado. Cada versão vai sendo disponibilizada na medida em que as funcionalidades vão sendo desenvolvidas. Por isso dar-se o nome de processo iterativo e incremental. Suponhamos que na primeira versão (*release*) do software, foram detectados pelo cliente 15 defeitos. Sendo assim, espera-se que na segunda versão o número de defeitos seja bem menor que o da versão anterior, melhorando desta forma a perspectiva do cliente em relação à correção dos erros.

4.1.3. Métrica de problemas detectados pelo cliente

Segundo Kan [2], outra métrica de produto utilizada pela maioria das indústrias de desenvolvimento de software, mede os problemas encontrados pelos clientes quando estão utilizando o produto (*Customer problem métrics*). Quando está montando a métrica de taxa de defeitos o numerador é um número válido de erros, entretanto quando o cliente está avaliando são contados todos os problemas encontrados. Problemas que não são válidos podem ser problemas de usabilidade do produto, de definições, documentações e informações não esclarecidas, defeitos válidos duplicados (Defeitos que não foram gerenciados na hora da coleta). Entretanto todos estes defeitos constituem o universo de defeitos que o sistema pode apresentar segundo a perspectiva do cliente.

Esta métrica pode ser calculada pela seguinte expressão:

PUM = Total problems that customers reported (true defects and non-defect-oriented problems) for a time period
+ Total number of license-months of the software during the period

Onde o número de Licenças-Mês = Número de licenças do software instaladas X Número de meses do período calculado

Desta forma pode-se utilizar o valor da métrica PUM para montar uma relação com KLOC (kilo lines of code) ou com PF (pontos de função), onde o numerador seria o cálculo de todos os problemas reportados (PUM) e o denominador seria a dimensão do software em KLOC ou PF.

4.1.4. Métrica de satisfação do cliente

Kan [2] descreve que a métrica de satisfação do cliente é frequentemente medida pela análise dos dados obtidos do cliente utilizando-se uma escala de 5 pontos:

- Muito satisfeito
- Satisfeito
- Neutro
- Insatisfeito
- Muito insatisfeito

Baseando-se nos dados da escala de 5 pontos, pode-se construir uma diversidade de variações de métricas, como por exemplo:

- Percentual de clientes satisfeitos (Incluindo os completamente satisfeitos)
- Percentual de clientes completamente satisfeitos
- Percentual de clientes insatisfeitos (Incluindo os muito insatisfeitos)
- Percentual de clientes insatisfeitos (Incluindo os neutros e muito insatisfeitos)

Com estes dados é possível medir o índice de satisfação do cliente em relação ao software e com isso avaliar os pontos do software onde é necessário trabalhar para aumentar a satisfação.

4.2. MÉTRICAS DE QUALIDADE DE PROCESSO

As métricas de qualidade de processo são utilizadas para melhorar o desenvolvimento de software e sua manutenção. Estas métricas tornam-se muito importantes em um processo de desenvolvimento, pois medem vários parâmetros nas várias fases do processo.

Kan [2] mostra algumas métricas básicas a nível de processo, como por exemplo, a Efetividade de remoção de defeitos (DRE) durante o desenvolvimento, demonstrada pela equação:

$$\text{DRE} = \frac{\text{Defects removed during a development phase}}{\text{Defects latent in the product}} \times 100\%$$

Onde o denominador é usualmente estimado como: Defeitos removidos durante a fase + defeitos encontrados depois.

Esta métrica pode ser calculada para o processo de desenvolvimento inteiro, para o *Front-end*, antes da integração com o código e para cada fase. Neste caso ela pode ser chamada de *"early defect removal"* e *"phase effectiveness",* quando utilizado para o *"front-end"* e fases específicas respectivamente.

Outro fato importante é que a utilização desta métrica em todas as fases do processo de desenvolvimento faz com que seja possível além de medir a efetividade de remoção de defeitos, também medir a injeção de erros, e com isso poder rastrear as fases que mais injetam erros na aplicação.

Ainda segundo Kan [2], para medir a efetividade de remoção de defeitos apresenta uma nova sugestão, que é a utilização de uma matriz de aproximação, no qual os defeitos são distribuídos e tabulados em termos de origem dos defeitos e fase na qual eles foram encontrados, conforme a figura 7.

Where Found	Defect Origin: Requirements	High-Level Design	Low-Level Design	Code	Unit Test	Component Test	System Test	Field	Total
RQ	—			—					
I0	49	681							730
I1	6	42	681						729
I2	12	28	114	941					1095
UT	21	43	43	223	2				332
CT	20	41	61	261	—	4			387
ST	6	8	24	72	—	—	1		111
Field	8	16	16	40	—	—	—	1	81
Total	122	859	939	1537	2	4	1	1	3465

Figura 7 - Exemplo de matriz de aproximação(Onde foram encontrados e Origem dos erros) onde I0 a I2 – Iterações, UT = Unit Test, CT- Component Test , ST – System Test

Na definição conceitual a contagem de defeitos removidos deve ser igual a contagem de erros detectados, com exceção dos erros de reparos incorretos. Neste último caso o ideal é que se utilize um sistema para fazer o rastreamento deste tipo de erro.

Conforme Kan [2], a fase de testes serve para remover os defeitos, porém nestes reparos podem ocorrer injeções de novos erros decorrentes de reparos incorretos. Na figura abaixo é mostrado o ciclo de defeitos em um passo ou fase no processo de desenvolvimento.

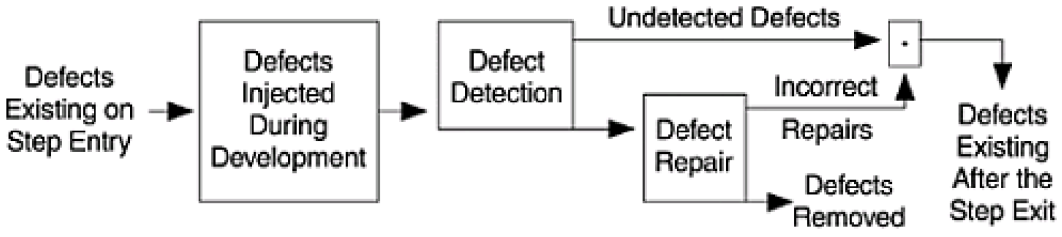


Figura 8 - Representação de um Ciclo de defeitos

$$\frac{\text{Defects removed (at the step)}}{\text{Defects existing on step entry} + \text{Defects injected during development (of the step)}} \times 100\%$$

Além de poder mensurar a quantidade de erros, a efetividade de remoção dos erros e encontrar os erros injetados em cada fase, é necessário se pensar também em encontrar uma

forma de se medir o tempo de resposta entre detecção de defeito e sua correção, desta forma surge mais uma métrica a se utilizar: O tempo médio entre detecção e reparo de erros. Neste caso é necessário um trabalho inicial de coleta e armazenamento das informações referente aos erros encontrados e reparados, para que seja possível posteriormente determinar o valor desta média.

Várias são as métricas que podem ser utilizadas para mensurar e avaliar o processo de desenvolvimento, Kan [2] cita um outro exemplo que é a métrica do índice de manutenção de erros chamada de BMI (*Backlog Management Index*).

$$\text{BMI} = \frac{\text{Number of problems closed during the month}}{\text{Number of problem arrivals during the month}} \times 100\%$$

Além das métricas citadas anteriormente, no IEEE Std 982.1 [8] estão catalogadas várias métricas utilizadas com a finalidade de construir softwares confiáveis. A seguir estão descritas algumas dessas métricas que também podem ser essenciais na implantação inicial de um programa de métricas no STJ.

- Functional or Modular Test Coverage

$$\text{FUNCTIONAL (MODULAR) TEST COVERAGE INDEX} = \frac{FE}{FT}$$

onde:

FE = Número de requisitos funcionais para que todos os casos de testes tenham sido completados satisfatoriamente.

FT = Número total de requisitos funcionais do software

- Test Coverage

$$TC(\%) = \frac{\text{(implemented capabilities)}}{\text{(required capabilities)}} \times \frac{\text{(program primitives tested)}}{\text{(total program primitives)}} \times 100$$

4.3. MÉTRICAS DE QUALIDADE DE PROJETO

Conforme Kan [2], qualidade de software é um subconjunto de métricas de software que se concentram em aspectos do produto, processo e projeto. No geral a qualidade de software está mais associada com as métricas de processo e de produto do que com as métricas de projeto.

As métricas de qualidade do projeto são mais voltadas às características do projeto e sua execução, e geralmente são montadas a partir da combinação das métricas de produto e processo.

Exemplos de possíveis métricas utilizadas em nível de projeto são:

- Número de desenvolvedores para determinado projeto
- Ciclo de vida
- Custo
- Cronograma
- Produtividade

Segundo Santos Filho [9], as métricas de software são usadas a fim de monitorar e avaliar os seguintes aspectos do projeto:

- Progresso em termos de tamanho e complexidade.
- Estabilidade em termos de taxa de mudança na implementação, tamanho ou complexidade.
- Modularidade em termos da extensão da mudança.
- Qualidade em termos do número e tipo de erros.
- Maturidade em termos da freqüência de erros.
- Recursos em termos de recursos despendidos contra os planejados.

4.3.1. Métrica de produtividade de software

A norma IEEE std 1045-1992 [10] descreve os padrões a serem adotados para métricas de produtividade de software. A equação básica para este tipo de medida é:

$$\text{Productivity}_a = \frac{O_a}{E_a}$$

Onde Oa = Tamanho da Saída ou primitiva de saída e Ea = Esforço necessário para produzir a saída Oa, geralmente expresso na unidade homens/hora.

Conforme a norma, as primitivas de saída, pode ser classificada em três categorias: **Linhas de código**, **Pontos de função** e **Documentação**. Como já visto a contagem de linhas de código e Pontos de função servem para mensurar o tamanho de um software, porém neste caso aparece outra importante coisa a se medir, a documentação, pois esta também consegue absorver esforços da equipe de desenvolvimento, portanto ela também deve ser considerada nos cálculos quando necessário.

Assim como as linhas de código, são consideradas na contagem somente as páginas de documentos não brancas, ou seja, somente páginas preenchidas. É determinado ainda que, o valor total deve ser um valor inteiro, e que qualquer tipo de documentação entra nesta contagem desde que seja realmente utilizada. Estes documentos são tipicamente: requisitos, especificações de modelagem e arquitetura, definição de dados, gráficos, manuais do usuário, manuais de referencia, tutoriais, guias de treinamento, manuais de instalação e manutenção e planos de teste.

4.3.2. Métricas OO

O paradigma de orientação a objetos também pode ser avaliado quanto à aderência assim como quanto à avaliação de um padrão mínimo de qualidade.

Existe uma tendência dos programadores que utilizam o modelo estruturado tentarem aplicar um modelo que não é realmente orientado a objetos.

As métricas OO tentam também identificar os possíveis desvios que possam acontecer em um produto que utilize o modelo OO como modelo de desenvolvimento.

Dois modelos são os mais famosos sobre Métricas OO. São eles: As Métricas de Lorenz e Métricas Chidamber e Kemerer.

4.3.2.1. Métricas de Lorenz e as “Rules of Thumb” segundo Kan[2]

As métricas propostas por Lorenz são na verdade uma compilação ou escolha de 11 mais relevantes métricas entre as 30 propostas pelo OOTC (IBM Object Oriented Technology Council) grupo qual, Lorenz participou do estudo.

Para chegar a essas métricas Lorenz participou de inúmeros projetos e ele concebeu o termo “Rules of Thumb” que pode ser traduzido como modelo baseado na sensibilidade e na observação de problemas e soluções encontradas em diversas situações. São elas:

Tabela 2 - Rules of Thumb

Métrica	Rules of Thumb and Comments
1. Tamanho médio de um método (LOC – Linhas de código)	Deve ser menor que 8 LOC para smalltalk e 24 LOC para C++
2. Número médio de métodos por classes	Deve ser menor que 20. Médias maiores indicação muita responsabilidade negocial em muito poucas classes
3. Número médio de instâncias de variáveis por classe	Deve ser menor que 6. Mais instancias de variáveis indicam que uma classe esta executando mais ações que deveria.
4. Nível de aninhamento hierárquico de classe. (Depth of Inherintance Tree, DIT)	Deve ser menor que 6, iniciando do framework de classes ou da classe raiz.
5. Número de Subsistemas/Relacionamento entre Subsistemas	Deve ser menor que o número obtido na métrica 6.
6. Numero Classes/ Relacionamento de classes em cada subsistema	Deve ser relativamente alto. Este item indica que alta coesão de classes em um mesmo sistema. Se uma ou mais classes em um subsistema não interagem com várias outras classes, elas podem ser colocadas em um outro subsistema.
7 Uso da variável instanciada	Se um grupo de métodos em uma classe usa conjuntos diferentes de variáveis de instancias, observe mais de perto se a classe deveria ser dividida em múltiplas classes especializadas por linhas de “serviço”.
8. Número médio de Linhas de comentários (por método)	Deve ser maior que 1.
9. Número de problemas reportados por classe	Deve ser baixo (não há um número definido)
10. Número de vezes que a classe é reusada	Se uma classe não é reusada em diferentes aplicações (especialmente as classes abstratas). Pode ser necessário que ela seja redesenhada.
11. Número de Classes e Métodos Descartados	Deve ocorrer no durante o processo de desenvolvimento. Se não está ocorrendo, uma provável causa é a adoção de um desenvolvimento incremental ao invés de um desenvolvimento iterativo real para desenho e desenvolvimento OO.
Fonte: Kan [2]	

A tabela mostra que as “rules of Thumb” são muito baseadas em experiências do autor no desenvolvimento OO de aplicações. Por exemplo: métrica 8 é uma regra das boas práticas de programação, métrica 9 é um indicador de qualidade, métrica 11 é uma métrica de validação de processo de desenvolvimento OO.

Um ponto importante dessa métrica é que a instituição que a adote, não precisa se preocupar inicialmente com os valores adotados para as métricas 1, 2, 6 e 9. São os projetos, a tecnologia adotada e a pretensão em objetivos a serem alcançados que irão indicar a faixa de valores a serem utilizadas. O interessante é pesquisar os valores que o mercado utiliza estabelecê-los como valores da instituição e depois estabelecer os novos valores a serem utilizados.

4.3.2.2. Métricas Chidamber e Kemerer ou Conjunto de Métricas CK segundo Kan [2]

Em 1994 Chidamber e Kemerer propuseram 6 métricas Orientadas a Objetos para medir desenho e complexidade, métricas essas que comumente são chamadas de conjunto de métricas CK

O conjunto de métricas para projeto orientado a objetos pretende ser uma abordagem compreensível para avaliar as classes no sistema. A maioria delas é calculada por classe, isto é, nenhuma delas avalia o sistema como um todo. Não está claro como estender essas métricas para o sistema todo. Realizar as medidas sobre todas as classes no sistema, geralmente, é inapropriado.

M1: Peso dos Métodos por Classe (WMC – weighted methods per class)

É baseada na intuição de que o número de métodos por classe é uma indicação significativa da complexidade do software.

Considerações posteriores podem ser necessárias para evitar-se dar peso completo, por exemplo, a métodos de receber e ordenar.

O WMC inclui a provisão para a pesagem dos métodos. Seja C um conjunto de classes, cada uma com um número de métodos M1,...Mn. Seja c1,...,cn a complexidade (pesos) das classes (o valor a ser usado para c1 não está definido no papel). Essa é a única métrica do conjunto que é a média de todas as classes do sistema. Nos exemplos, iremos assumir que c1 é igual a 1. Conforme tabela 3.

Tabela 3 - Análise de WMC

Classe	Métodos
Point	3
Rectangle	3
Linklistnode	4
Rectanglelist	3

WMC = 13/4 = 3,25 métodos/classe

Uma análise mais profunda pode-se utilizar a complexidade ciclomática por método. Porém, medir a complexidade ciclomática[2] é difícil para ser implementada porque nem todos os métodos são acessíveis na hierarquia de classes devido à herança.

Além disso, em estudos empíricos, WMC é frequentemente apenas o número de métodos em uma classe, e a média da WMC é o número médio de métodos por classe.

M2: Profundidade da Árvore de Herança (DIT)

A métrica profundidade da árvore de herança (DIT – depth of inherence tree) é exatamente o comprimento máximo de qualquer nó até a raiz da árvore de herança desta classe. A herança pode adicionar complexidade ao software. Essa métrica é calculada para cada classe.

M3: Número de Filhos (NOC)

Não apenas a profundidade, mas também a largura da árvore de herança é significativa. A métrica número de filhos (NOC – number of children) é o número de subclasses imediatamente subordinadas à classe na hierarquia de herança. Essa métrica é calculada para classe.

M4: Acoplamento entre Classes de Objetos (CBO)

No software orientado a objetos, podemos definir acoplamento como o uso de métricas ou atributos em outra classe. Duas classes serão consideradas acopladas quando métodos

[2] A Complexidade Ciclomática é uma métrica de software que proporciona uma medida quantitativa da complexidade lógica de um programa. Quando usado no contexto do método de teste do caminho básico, o valor computado da complexidade ciclomática define o número de caminhos independentes do conjunto básico de um programa e oferece-nos um limite máximo para o número de testes que deve ser executado para garantir que todas as instruções sejam executadas pelo menos uma vez. Um caminho independente é qualquer caminho através do programa que introduza pelo menos um novo conjunto de instruções de processamento ou uma nova condição.

declarados em uma classe usam métodos ou variáveis instanciadas definidas pela classe. O acoplamento é simétrico: se a classe A é acoplada à classe B, então B é acoplada a A. A métrica acoplamento entre classes de objetos (CBO – coupling between object classes) será a contagem do número de classes às quais a classe está acoplada. Essa métrica é calculada para cada classe. Conforme figura 9 com detalhes do cálculo na tabela 4.

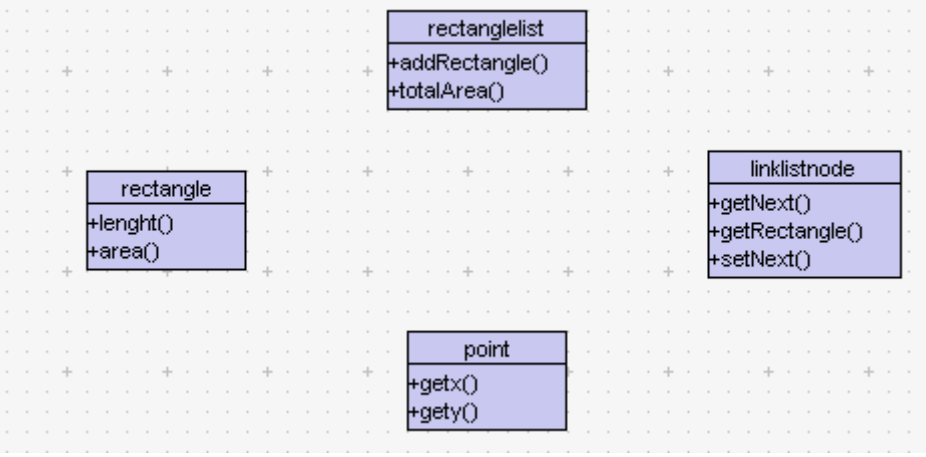


Figura 9 – Classes de exemplo

Tabela 4 – Dados da Análise de CBO das classes de exemplo

Classe	Classes Acopladas	CBO
Point	rectangle	1
Rectangle	point, rectanglelist	2
Linklistnode	rectanglelist	1
Rectanglelist	rectangle, linklistnode	2

M5: Respostas para a Classe (RFC)

O conjunto resposta para a classe, {RS}, é um conjunto de métodos que podem potencialmente ser executados em resposta recebida por um objeto dessa classe (RFC – response for a class). É a união de todos os métodos na classe. Só é contado um em cada nível de chamada. Essa métrica é calculada para cada classe. Conforme tabela 5.

$RFC = |RS|$

Tabela 5 - Dados da análise de RFC das classes de exemplo

Classe	Classe Acoplada	RFC
Point	point, getx, gety	3
Rectangle	rectangle, point, lenght, getx, gety, area	6
Linsklistnode	linkListNode, getNext, getRectangle, setNext	4
Rectanglelist	rectangleList,	8

	addRectangle, rectangle, setNext, totalArea, getRectangle, area, getNext	

M6: Perda de Coesão nos Métodos (LCOM)

Um módulo (ou classe) é coeso se tudo é estritamente relacionado. A métrica de perda de coesão nos métodos (LCOM – lack of cohesion in methods) tenta medir a perda da coesão.

A métrica LCOM pode ser visualizada considerando um grafo bipartido. Um conjunto de nós consiste dos atributos e o outro consiste das funções. Um atributo está ligado a uma função se essa função acessa o atributo. O conjunto de setas é o conjunto Q. Se existem n atributos e m funções, então existem n * m setas possíveis. Assim, o tamanho de P é n * m menos o tamanho de Q.

LCOM = max(|P| - |Q|,0)

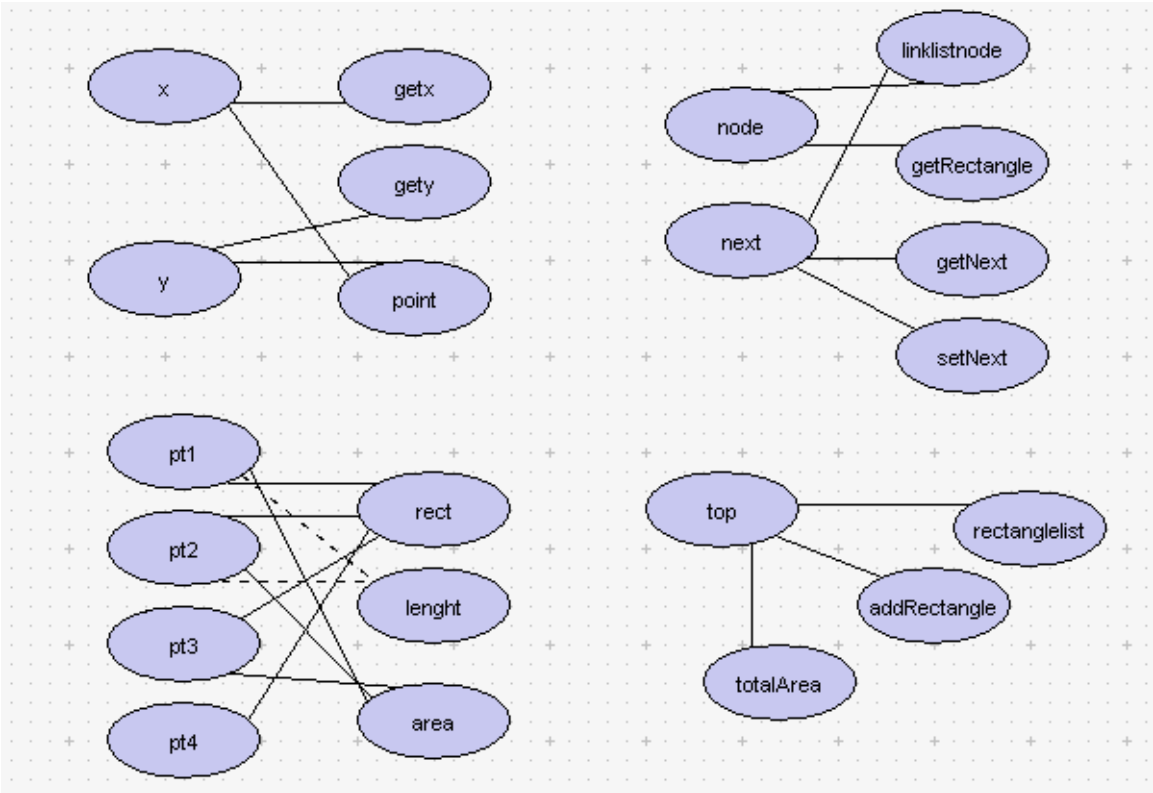


Figura 10 – Grafo da análise de LCOM

Na figura acima, as linhas estão pontilhadas porque dependem dos parâmetros que estão sendo acessados em uma chamada específica.

Tabela 6 - Dados da análise de LCOM

Classe	LCOM

Point	max(0,(6-4)-4)=0
Rectangle	max(0,(12-9)-9)=0
Linklistnode	max(0,(8-5)-5)=0
Rectanglelist	max(0,(3-3)-3)=0

Os estudos nos indicam que as métricas de Chidamber e Kemerer também seguem as linha de Lorenz, os valores médios e as faixas admissíveis são obtidos quando diversos projetos de uma instituição. As métricas de Chidamber e Kemerer são mais complexas de serem aplicadas que as métricas de Lorenz, pois as mesmas analisam muito mais a compreensão do paradigma orientado a objetos das pessoas que desenvolveram o projeto. Conforme a tabela 6.

5. AS SETE FERRAMENTAS BÁSICAS DE ISHIKAWA.

As sete ferramentas básicas de Ishikawa são utilizadas para controle estatístico de qualidade, portanto, podem também serem aplicadas no processo de desenvolvimento de software com o intuito de ajudar na análise de dados coletados no processo de melhoria de qualidade do produto ou no próprio processo de desenvolvimento. Além disso, Ishikawa afirmava que o uso dessas ferramentas resolve aproximadamente 95% dos problemas de qualidade em qualquer tipo de organização.

As sete ferramentas de Ishikawa citadas por Kan [2] são:

- *Check-lists* ou *Check sheets*
- Diagrama de Pareto
- Histogramas
- *Run-charts* (gráfico de linha)
- *Scatter diagram* (gráfico de dispersão)
- *Control Chart* (gráfico de controle)
- Diagrama de causa e efeito – *Fishbone*

5.1. CHECK-LISTS OU CHECK-SHEET

Os *check-lists* e *check-sheets* são formulários com itens para serem checados durante um processo de coleta de informações. A função principal deles é justamente a coleta de informações de forma facilitada para serem posteriormente utilizados e analisados.

VOLTAGEM	CONTAGEM
1,65	
1,64	X
1,63	
1,62	XXX
1,61	X
1,60	XXXXX
1,59	XXXXX XXX
1,58	XXXXX XXXXX XX
1,57	XXXXX XXXXX
1,56	XXXXX X
1,55	XX
1,54	X
1,53	
1,52	XX
1,51	X
1,50	

Figura 11 - Exemplo de *check-sheet*

Segundo Kan [2] o processo de desenvolvimento de software da IBM Rochester consiste de múltiplas fases, por exemplo: Requisitos, arquitetura do sistema, modelagem de alto nível, modelagem de baixo nível, codificação, testes unitários, testes de integração e construção, testes de componentes, testes de sistema e versão inicial do programa. Cada fase possui um conjunto de tarefas que devem ser completadas para se ter uma finalização formal da tarefa. Os *check-lists* e *check-sheets* ajudam os desenvolvedores e programadores a assegurarem que todas as tarefas necessárias foram completadas e que importantes fatores ou características de qualidade de cada tarefa foram cobertos.

Existem vários tipos de *check-lists*, um exemplo deles citado por Kan [2] que é bem utilizado é a lista de erros mais comuns, no qual faz parte do passo 3 do processo de prevenção de erros, processo este que envolve três passos chaves: (1) Análise dos defeitos para traçar as causas, (2) Equipes de ação, para implementar as ações sugeridas, e (3) Reunião para o pontapé inicial (*kickoff*), no qual se coleta os erros mais comuns e realiza o *brain-storm* de idéias, para se iniciar cada fase do desenvolvimento. Outro fato importante citado por Kan [2] é que todos os *check-lists* devem ser revisados conforme vão sendo utilizados e conforme vai aumentando a experiência da equipe na utilização dos mesmos. A facilidade de preenchimento deve ser vital, senão a ferramenta não ajudará na melhoria do processo, outra característica importante que ele deve possuir é a rastreabilidade da informação, ou seja, na detecção de um erro, como por exemplo, o *check-list* deve ter precisão para que se encontre todas as informações necessárias para a sua simulação (Ex: Módulo, data e hora do erro, usuário, máquina e outras informações necessárias para a simulação).

5.2. DIAGRAMA DE PARETO

Um diagrama de Pareto é um gráfico de freqüência de barras em ordem decrescente. A freqüência das barras é usualmente associada com os tipos de problemas. Em 1950 Juran aplicou os princípios deste tipo de diagrama para a identificação de problemas de qualidade, e até hoje ele é utilizado para as mais diversas aplicações de processos de melhoria da qualidade.

Segundo Kan [2], em desenvolvimento de software o eixo X para o diagrama de Pareto é a causa do defeito e o eixo Y é a contagem da ocorrência destes defeitos. O diagrama de Pareto pode identificar as possíveis causas para a maioria dos defeitos, ele indica ainda quais problemas devem ser resolvidos primeiro no processo de remoção de defeitos e melhoria da operação.

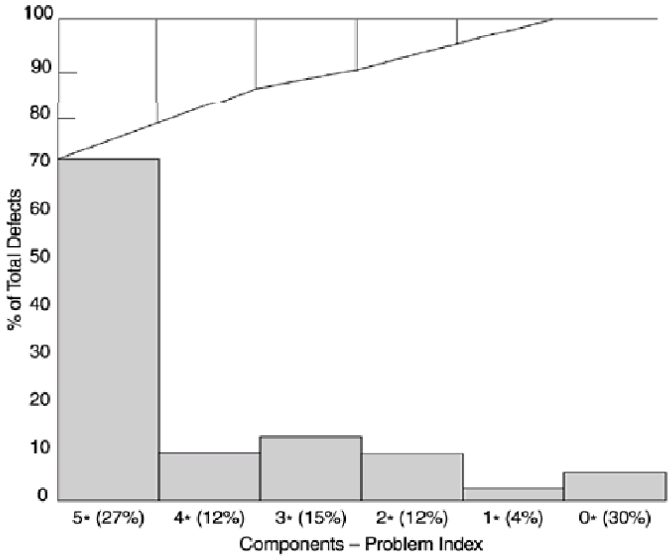


Figura 12 - Pareto Diagram of Defects by Component Problem Index

A análise de Pareto está comumente referenciada no princípio de 80-20 (20% das causas para 80% de defeitos), porém esta relação de Causa-Defeitos nem sempre está nesta distribuição.

Conforme Kan [2], a análise de Pareto ajuda a identificar as áreas que causam a maioria dos problemas, e ele é mais aplicável em qualidade de software porque defeitos de software ou densidade de erros nunca mostram uma distribuição uniforme.

5.3. HISTOGRAMA

Conforme Kan [2] o Histograma é uma representação gráfica de uma amostra ou uma população. O eixo X lista uma unidade de intervalo do parâmetro (Ex: nível de severidade do defeito de software), disposto em ordem crescente da esquerda para direita e o eixo Y contém a contagem das ocorrências.

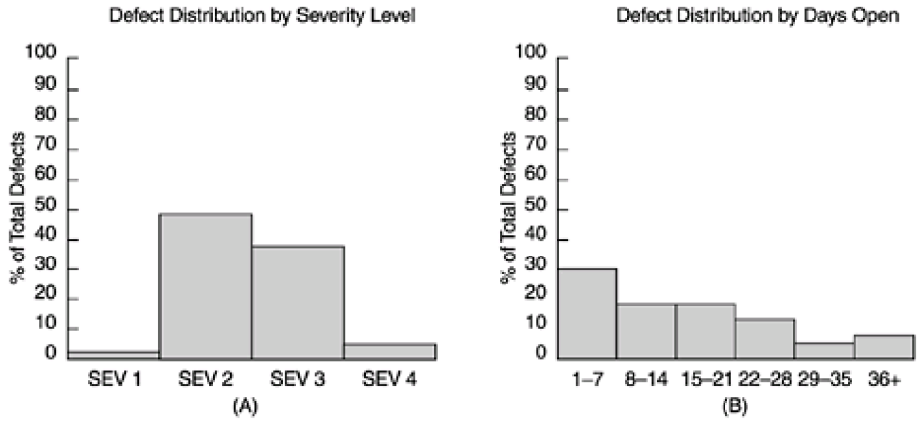


Figura 13 - Exemplos de Histogramas

O propósito do histograma é mostrar a característica da distribuição do parâmetro. (*Central tendency*, *dispersion and skewness*). A sua construção também tem por finalidade identificar anormalidades no processo, e uma de suas vantagens é verificar a existência ou não de simetria do processo em relação à média.

5.4. RUN-CHARTS

O *Run-Chart* mostra a performance do parâmetro de interesse no decorrer do tempo. O eixo X é o tempo e o eixo Y é o valor do parâmetro, o *run-chart* é muito usado para análise de tendência, especialmente se dados históricos são eficazes em comparação com a tendência corrente. Eles são tipicamente utilizados na gerência de projetos de software, e numerosos exemplos podem ser encontrados em livros e artigos de engenharia de software.

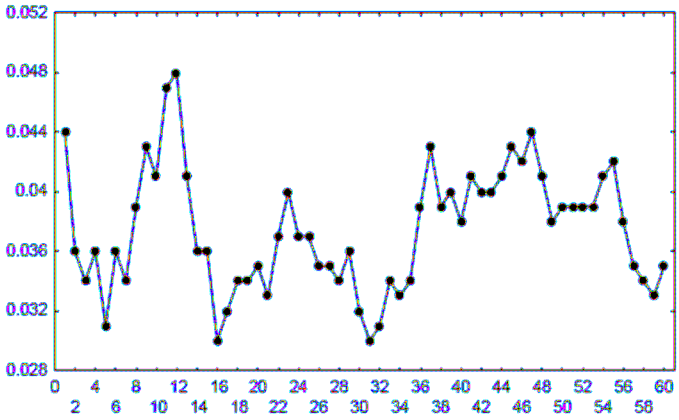


Figura 14 - *Run chart* ou gráfico de linha

Este tipo de gráfico também é conhecido como gráfico de linha, e serve como um termômetro da qualidade em tempo real, inclusive também pode ser comparado com dados armazenados em históricos para se fazer interpretações de perspectivas e projeções futuras.

5.5. SCATTER DIAGRAM

Um *Scatter Diagram* retrata um relacionamento entre dois intervalos variáveis. Em um relacionamento Causa-Efeito o eixo X é a variável independente e o eixo Y é para variável dependente. Cada ponto representa uma observação das duas variáveis. Este diagrama ajuda a produzir decisões baseadas em dados (Ex: Se a ação está planejada na variável X e algum efeito é esperado na variável Y).

O *Scatter Diagram* também é conhecido como gráfico de dispersão e com ele é possível identificar se existe uma tendência de variação conjunta (correlação) entre duas ou mais variáveis.

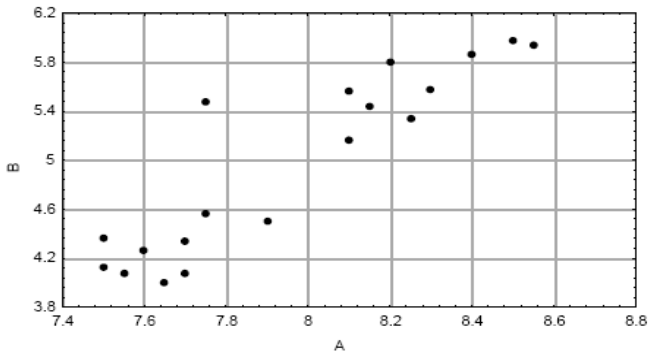


Figura 15 - Scatter Diagram

A análise do *Scatter Diagram* pode resultar em alguns tipos de Tendência/Correlação, que podem ser classificadas como:

- Correlação linear Positiva
- Correlação linear Negativa
- Correlação não linear

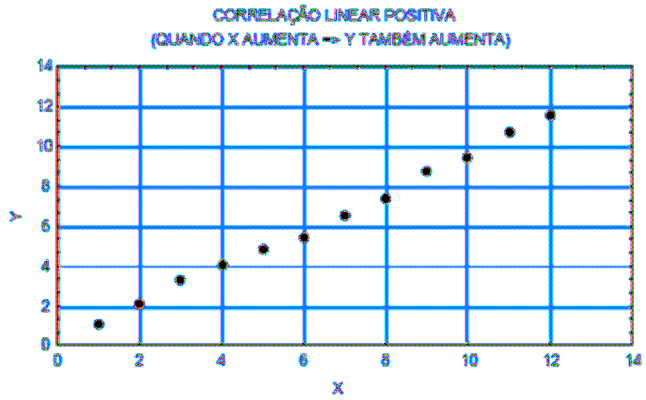


Figura 16 - Exemplo de correlação linear Positiva

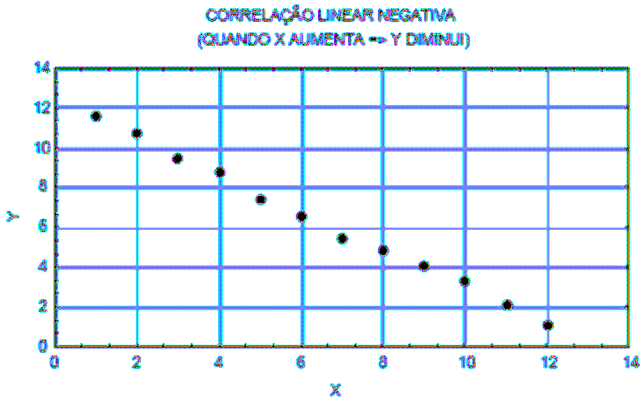


Figura 17 - Exemplo de correlação linear Negativa

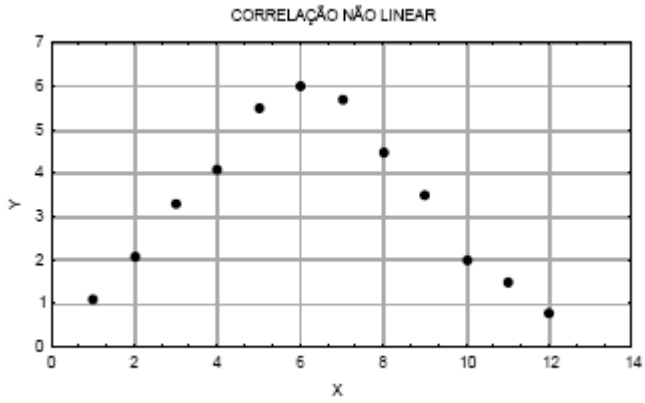


Figura 18 - Exemplo de correlação não linear

5.6. CONTROL CHART

Os gráficos de controle analisam o comportamento do processo de fabricação, permitindo que se possa atuar no processo de forma preventiva efetuando ações corretivas no momento em que ocorrerem desvios e assim permitam manter o processo dentro de condições preestabelecidas. Os gráficos de controle também podem ter um papel importante na aceitação do produto, pois o controle estatístico verifica a estabilidade do processo e a homogeneidade do produto.

Os gráficos de controle exibem três linhas paralelas ao eixo X:

- Linha Central: representa o valor médio do característico de qualidade exigido.
- Linha Superior: representa o limite superior de controle (LSC)
- Linha Inferior: representa o limite inferior de controle (LIC)

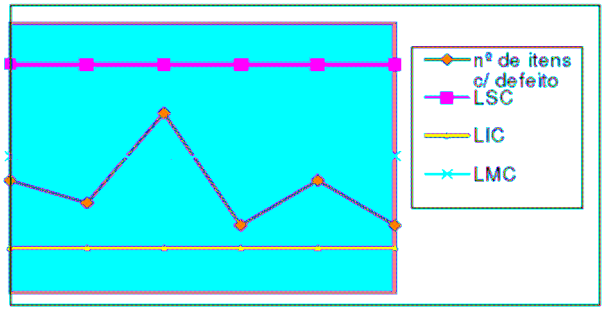


Figura 19 - Modelo Genérico de um gráfico de controle (*Control-Chart*)

Os limites de controle, de um modo geral, são estabelecidos a partir da média ± 3 desvios padrões ($\mu \pm 3$), como o Modelo de Shewhart, segundo Sommer[11]. A faixa entre os limites de controle define a variação aleatória no processo. Se os pontos traçados no gráfico estiverem dentro dos limites de controle e estiverem dispostos de forma aleatória, pode-se dizer que o processo está sob controle estatístico. Caso contrário, se um ou mais pontos estiverem fora dos limites de controle ou estiverem dispostos de forma não aleatória, pode-se dizer que o processo está fora de controle estatístico. Então, indicam uma ou mais causas determináveis de variação, e assim precisa-se identificar os fatores que causam tais variações para que esses pontos sejam eliminados.

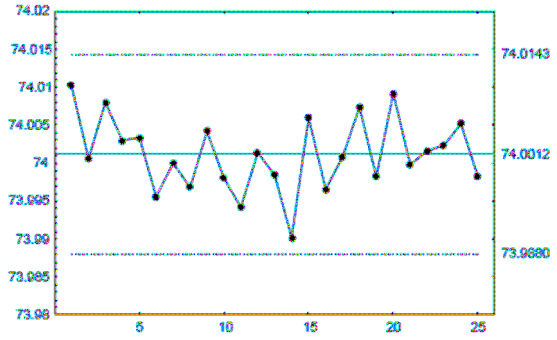


Figura 20 - Processo previsível ou Estável ou sob controle

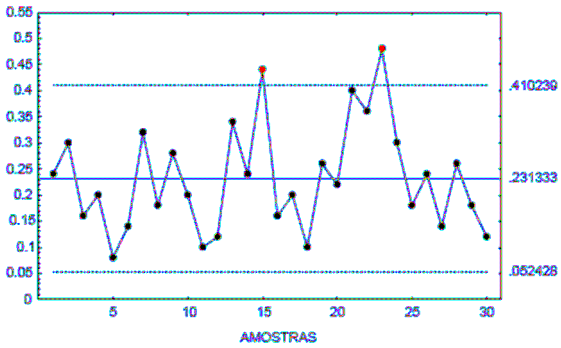


Figura 21 - Processo imprevisível ou Instável, ou fora de controle

O primeiro modelo formal de gráfico de controle foi proposto pelo Dr Walter A. Shewhart em 1931, que atualmente leva o seu nome.

Conforme Russo [12], a compreensão do gráfico de controle pode ser descrita como: Considere X uma estatística amostral que mede uma característica do processo usado para controlar uma linha de produção. Suponha que a média populacional de X seja μ e o desvio padrão populacional seja σ . As seguintes equações são usadas para descrever os três parâmetros que caracterizam os gráficos de controle de Shewhart.

$$LSC = \mu + k\sigma_{\bar{x}}$$

$$LC = \mu$$

$$LIC = \mu - k\sigma_{\bar{x}}$$

Onde LSC é o limite superior de controle, LC é a linha central ou a média do processo, LIS é o limite inferior do processo, e k é a distância dos limites de controle até a linha central, a qual é expressa como um múltiplo do desvio padrão σ . O valor de k mais usado é 3.

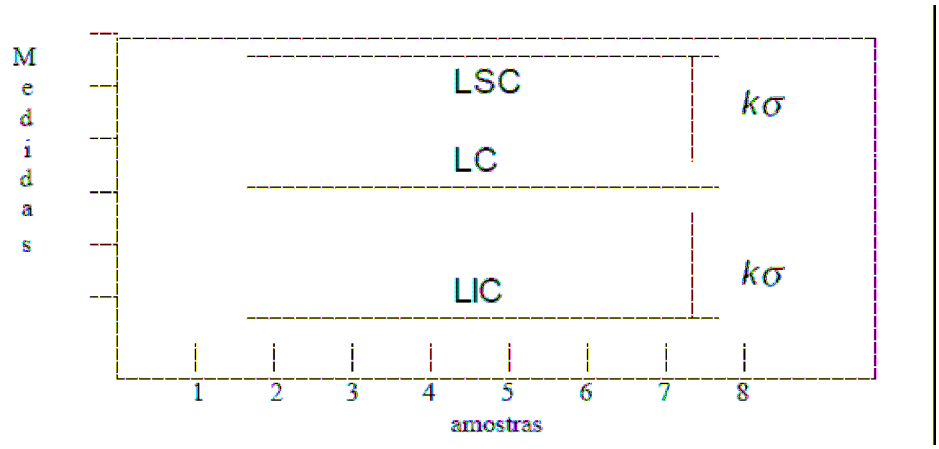


Figura 22 - Gráfico de controle de Shewhart

5.7. DIAGRAMA DE CAUSA E EFEITO – FISHBONE

Também conhecido como "Espinha de Peixe" ou Diagrama de Ishikawa, ajuda a identificar as causas dos problemas. Sua forma é similar à espinha de peixe, onde no eixo principal é colocado o efeito ou o problema que se quer analisar e cada espinha ou ramificação simboliza cada categoria de causas.

Este diagrama foi criado em 1943 por Kaoru Ishikawa e tem por objetivo principal a visualização de um processo, ou seja, o mapeamento entre uma série de fenômenos que se sucedem e que são ligados entre si pelas relações de causa e efeito.

O modelo original sugeria quatro grandes grupos de causas que deveriam ser analisadas. Esses quatro grupos (também conhecidos como quatro M's) eram: materiais, mão-de-obra, métodos e máquinas. Versões mais recentes desse diagrama sugerem a análise orientada por seis grandes grupos de causas: materiais, mão-de-obra, métodos, máquinas, medidas e meio-ambiente conforme Vieira [13]. A figura a seguir representa um modelo geral do diagrama. Dele pode-se compreender o funcionamento da ferramenta:

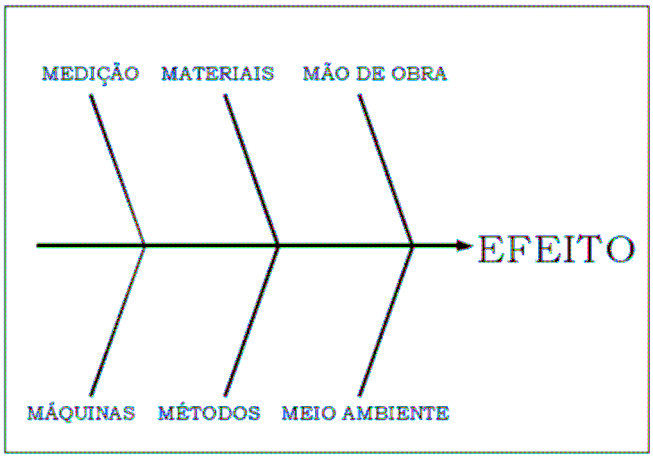


Figura 23 – Modelo básico de Diagrama de Causa e Efeito

Conforme Selner [14], na extremidade direita do eixo central é apresentado o sintoma que representa o problema a ser resolvido, ou o efeito desejado do processo. A esse eixo central estão ligadas as diversas causas que de alguma forma cooperam para que o sintoma ou efeito ocorra, cada causa, por ocasião de sua análise, transforma-se também num eixo central e a esse eixo são ligadas causas menores. Essa iteração pode ocorrer indefinidamente, até que as causas mais elementares sejam identificadas. Conforme exemplo na figura 24.

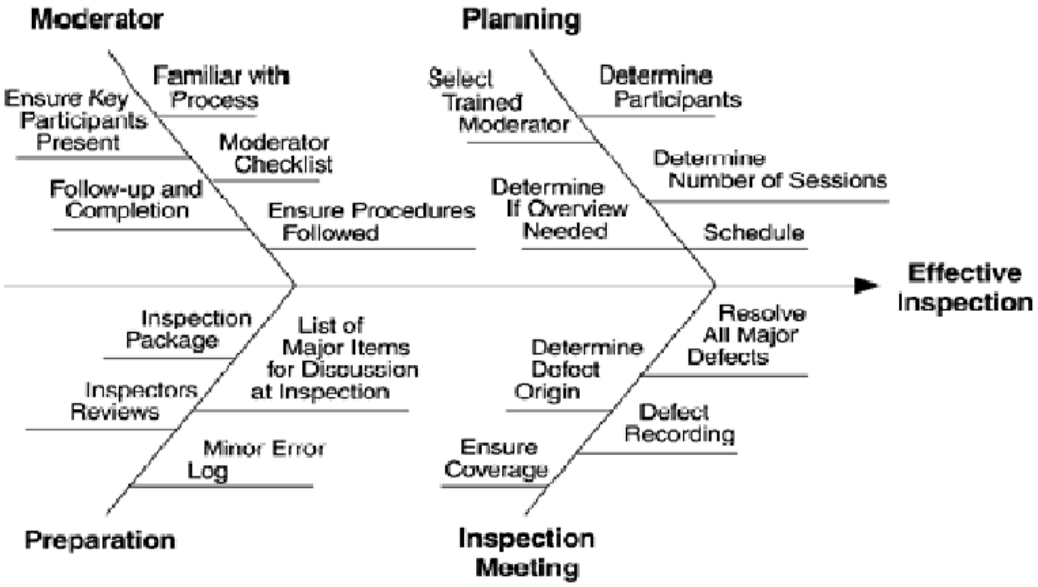


Figura 24 - Exemplo de Diagrama de causa e efeito mais completo.

6. MÉTRICAS NO RUP

Conforme Campos [15], os principais problemas que um Gerente de Projetos de Software pode enfrentar, estão relacionados aos processos de estimativa dos prazos de entrega dos projetos. Influenciando estes prazos, e suas estimativas, tem-se ainda a composição da equipe de projeto e desenvolvimento e o acompanhamento das etapas efetuadas pela mesma. Somam-se a isso, as crescentes mudanças no mercado e as novas tecnologias disponíveis para se projetar e desenvolver softwares.

O Gerente de Projetos de Software torna-se, portanto, fundamental na sobrevivência e crescimento destes projetos e da própria empresa no mercado.

Porém estimar o tempo de projeto, o tempo de desenvolvimento de um software e a data de entrega do mesmo para o cliente é um processo delicado o qual envolve muita sensibilidade do Gerente de Projetos de Software e, principalmente, uma excelente capacidade de estimativa.

É um cenário comum dentro das empresas desenvolvedoras de software. Ocasionando um aumento na responsabilidade dos Gerentes de Projetos de Software, que só podem contar com sua experiência profissional como ferramenta. Mas por maior que seja a experiência do profissional atuando como Gerente de Projetos de Software, ele fatalmente irá deparar-se com projetos de software que ainda não trabalhou, mesmo com algo similar, e pior ainda, iniciar trabalhos com uma equipe, onde ele não conhece os seus membros e nem suas capacidades.

A Engenharia de Software fornece como ferramentas possíveis de auxilio, as métricas de software e métricas para projetos, ferramentas capazes de gerarem dados para serem analisados e utilizados como apoio pelos Gerentes de Projetos de Software.

Na busca por estes dados e informações a engenharia de software demonstrou que é necessário adotar um modelo de processo de desenvolvimento de software e como exemplo podemos citar um dos mais conhecidos atualmente: o RUP (Rational Unified Process).

Tanto o UP (Unified Process) como o RUP são, antes de tudo, processos de desenvolvimento de software, e como tal, seu objetivo é transformar os requisitos do usuário em sistema de software conforme Casalvara [16]. Suas características fundamentais devem ao fato de serem baseados em componentes, ou seja, o software desenvolvido é formado por componentes de software que se comunicam através de interfaces bem definidas. O padrão adotado para representação dos modelos é a Unified Modeling Language (UML).

Conforme Mello Filho [17], apesar de acreditar que medir é sempre melhor que estimar, a prática tradicional adota francamente o feeling e a estimativa, e com isto as métricas verdadeiras somente se concretizarão no final do projeto. Portanto, elas não contribuirão mais ao projeto em questão e desta forma vão contribuir apenas nos próximos projetos. Este problema já não existe no processo iterativo-incremental, pois o mesmo utiliza muito rapidamente toda e qualquer informação de feedback que as métricas podem oferecer.

Seguindo ainda a linha de pensamento de Mello Filho [17], existem métricas nas quais são utilizadas no controle do projeto e que podem auxiliar o gerente de projetos em suas medições, é o que ele chama de métricas de controle. Em seu trabalho ele apresenta sete métricas a serem utilizadas para este fim, e descreve também que ao contrário das métricas de estimativa as métricas de controle devem ser utilizadas periodicamente para que o gerente possa ver o andamento do projeto, e para isto é conveniente que a coleta, computação e apresentação das informações estejam automatizadas. É fundamental ainda a facilidade de obtenção dos valores, pois o período de atenção passa a ser contado em dias ou semanas.

No escopo do RUP, a visão sobre métricas parte do motivo de que medimos para que consigamos ganhar controle sobre o projeto e, portanto para sermos capazes de gerenciá-lo. Secundariamente medimos também para melhor estimar novos projetos. Outro ponto importante sobre o escopo de métricas no RUP é: O que medir? Pois a realização de medidas custa caro e por isso não devemos sair medindo tudo, devemos avaliar as necessidades e pesquisar na organização os seus objetivos e coletar somente as métricas que os satisfaçam. No RUP isto pode ser documentado no Measurement Plan.

A insistência do RUP no planejamento da definição exata dos conceitos, cálculos e procedimentos em torno das métricas dar-se-á necessidade de não fazer más interpretações sobre os dados coletados. Um exemplo que pode ser considerado é o fato de um programador pegar uma determinada rotina que foi mal escrita e depois de refatorá-la esta rotina encolheu possuindo desta forma bem menos linhas de código que antes. Se a medida de produtividade for baseada em linhas de código, logo este programador terá produtividade negativa.

Com a finalidade de corrigir estas interpretações errôneas o RUP direciona para se utilizar as métricas na prática sempre a partir do Plano de métricas que deve ser sempre pré-definido e discutido entre a equipe, aproveitando as experiências de cada um. Vale lembrar também que este plano deve ser feito uma vez para cada projeto e que juntamente com ele devem também ser feitas as coletas das métricas, registrando os resultados no documento de avaliação de status do projeto.

O processo da Rational possui conceitos para a medição tanto do produto, processo, projeto e recursos e possui algumas diretrizes para diversos níveis de aplicação e implantação do plano de métricas:

- Um conjunto mínimo de métricas
- Um conjunto pequeno de métricas
- Um conjunto completo de métricas

Desta forma fica a critério da organização a personalização do grau de aplicação das métricas nos seus projetos.

6.1. UM CONJUNTO MÍNIMO DE MÉTRICAS

Mesmo em pequenos projetos é necessário controlar o andamento para averiguar se será necessário fazer novas estimativas, para possíveis alterações no escopo, em casos em que o projeto não esteja dentro do cronograma e do orçamento. Para auxiliar na resolução deste problema o RUP oferece um conjunto mínimo de métricas focadas no progresso do projeto.

- **Valor Atribuído** – Utilizado para refazer a estimativa do cronograma e do orçamento para o restante do projeto e/ou para identificar a necessidade de mudanças no escopo. É um método muito utilizado para medir o andamento do projeto, onde sua forma de cálculo é o somatório do esforço estimado original de todas as tarefas concluídas. O "Percentual de conclusão" do projeto, portanto pode ser calculado como o Valor atribuído dividido pelo total de esforço estimado original do projeto. A "Produtividade" (ou Índice de desempenho) é o valor atribuído dividido pelo esforço real empregado nas tarefas concluídas.
- **Tendência de Defeitos –** Muito útil para controlar a tendência de defeitos abertos e fechados. Este controle fornece uma indicação aproximada da existência de uma quantidade significativa de trabalho na correção de defeitos, e também a rapidez que se leva na correção. A análise desta métrica não é apresentada de forma detalhada no RUP, porém ela basicamente é feita através de gráficos com a contagem do número de defeitos abertos e corrigidos por um determinado período. Algumas outras considerações são apresentadas, como por exemplo: Ausência de defeitos registrados pode indicar ausência de testes e muito cuidado com o nível de esforço de teste que ocorre ao examinar tendências de defeitos.
- **Tendência de andamento de testes –** É uma outra métrica analisada por gráfico assim como a tendência de defeitos, só que nesta se controla a quantidade de

funcionalidade que já fora integrada ao sistema. Basicamente pode ser representada no gráfico como a freqüência da contagem do número de casos de teste (em andamento e planejados) distribuídos pelas semanas decorridas.

6.2. UM CONJUNTO PEQUENO DE MÉTRICAS

Este conjunto pode ser utilizado em projetos onde não seja suficiente o conjunto mínimo de métricas e desta forma o RUP recomenda se seja seguido para estes projetos um conjunto que pode ser resumido como a seguir:

- **Métricas e métricas primitivas –** Entre as métricas relacionadas para este conjunto no RUP podemos destacar as seguintes: Total de linhas de código estimado (SLOCt), SLOC sob controle de configuração (SLOCc), Defeitos críticos, Defeitos normais, Total de solicitações de mudança e Retrabalho aberto e fechado.
- **Métricas de qualidade para o produto final –** Nesta categoria foram relacionadas várias métricas onde se destacam as seguintes: Taxa de retrabalho, Modularidade, Adaptabilidade, Maturidade e manutenibilidade. Além destas métricas o RUP indica ainda que este conjunto pode resultar outras métricas que sejam bem interessantes.
- **Indicadores de andamento –** Os indicadores citados foram: Estabilidade de retrabalho, Quantidade de retrabalho, Tendência de Modularidade, Tendência de Adaptabilidade e Tendência de Maturidade.

6.3. UM CONJUNTO COMPLETO DE MÉTRICAS

Este conjunto completo significa que o processo de métricas aplicado deverá ser mais abrangente e deste modo deve-se em primeiro lugar investigar o que deve ser necessário para alcançar os objetivos do projeto. Entre estas investigações o RUP destaca a seguinte questão:

- O que deve ser medido?

Em resposta ele destaca os seguintes itens:

- O processo - a seqüência de atividades para se criar o softwares e outros artefatos;
- O produto - os artefatos do processo, como software, documentos e modelos;
- O projeto – a totalidade de recursos, atividades e artefatos do projeto;

- Os recursos - pessoas, métodos, ferramentas, tempo, esforço e orçamento disponíveis para o projeto.

6.3.1. Métricas para o processo

O RUP oferece o conjunto completo de métricas com suas diretrizes para que seja de certa forma, utilizado para corporações que queiram controlar de forma mais completa e abrangente o processo de desenvolvimento de seus projetos.

Nas métricas de processo, por exemplo, são sugeridas a utilização para medição de Esforço, Defeitos (juntamente com as Taxas de detecção e Taxas de correção), rotatividade de pessoal, Duração das atividades, Saída, Uso do ambiente de desenvolvimento de software, entre outras.

6.3.2. Métricas para o produto

O RUP reforça que seus produtos, são basicamente artefatos (documentos, modelos ou elementos do modelo), e portanto, suas métricas de produto são recomendadas para serem aplicadas em todos eles. As características aplicáveis em todos os produtos são:

- Tamanho;
- Esforço;
- Volatilidade;
- Qualidade;
- Abrangência; e
- Rastreabilidade.

6.3.2.1. Documentos

Todos os documentos do RUP podem passar pelo processo de métricas e como exemplos são sugeridos as seguintes métricas para os documentos.

Tabela 7 - Métricas aplicáveis em documentos

Característica	Métricas
Tamanho	Contagem das páginas
Esforço	Unidades de equipe-tempo para a produção

Volatilidade	Quantidade de mudanças e defeitos
Qualidade	Contagem de defeitos
Abrangência	O Julgamento é feito na revisão
Rastreabilidade	O Julgamento é feito na revisão

6.3.2.2. Modelos

As métricas sugeridas pelo RUP, podem ser aplicadas em todos os modelos do RUP, segue abaixo uma lista de modelos que estão descritos com as suas respectivas métricas nas diretrizes do RUP.

- Atributos de requisitos (elemento de modelo)
- Modelo de casos de uso
- Modelo de análise
- Modelo de design
- Modelo de implementação
- Modelo de teste
- Modelo de mudanças (em gerenciamento)

Como exemplo segue uma tabela com algumas métricas aplicáveis no modelo de casos de uso.

Tabela 8 - Métricas aplicáveis em casos de Uso

Característica	Métricas
Tamanho	-Número de Casos de Uso -Números de pacotes de casos de uso -Número de cenários -Número de atores -Tamanho do caso de uso
Esforço	- Unidades de equipe-tempo (com a produção, mudança e correções separadas)
Volatilidade	- Quantidade de defeitos e solicitações de mudanças (Abertos e fechados)
Qualidade	- Complexidade relatada (0 -5, por analogia com o COCOMO(Construct Cost Model) no nível de classe. - Número de defeitos Abertos e fechados por nível de gravidade.
Abrangência	- Casos de uso concluídos (revisados e sob controle de configuração sem defeitos pendentes)/Casos de uso identificados(ou

	número estimado de casos de uso) - Rastreabilidade de requisitos para UC (a partir de Atributos de requisitos)
Rastreabilidade	- Análise (Cenários realizados no modelo de análise/total de cenários) -Design (Cenários realizados no modelo de Design/total de cenários) -Implementação (Cenários realizados no modelo de Implementação/total de cenários) - Teste (Cenários realizados no modelo de Teste- casos de teste/total de cenários)

6.3.3. Métricas para o projeto

Dentre as sugestões do RUP, um projeto precisa ser caracterizado em termos de tipo, tamanho, complexidade e formalidade.

O tipo o tamanho e a complexidade acabam definindo a formalidade do projeto. Já o tamanho do projeto deve ser registrado por custo, esforço, duração tamanho do código a ser desenvolvido e pontos de função a serem liberados. A complexidade do projeto por sua vez pode ser descrita subjetivamente e também pode ser representada em gráficos que mostrem a complexidade técnica.

As métricas derivadas para controle do projeto originam-se portanto das seguintes métricas:

Modularidade = Quebra média NCNB[3] por mudança corretiva ou de aperfeiçoamento no modelo de implementação.

Adaptabilidade = Esforço médio por mudança corretiva ou de aperfeiçoamento no modelo de implementação.

Maturidade = Tempo de teste ativo/ número de mudanças corretivas

Manutenibilidade = Produtividade de manutenção/Produtividade de desenvolvimento= [Correções cumulativas reais/ esforço cumulativo de mudanças corretivas e de aperfeiçoamento] / [número estimado de NCNB na conclusão/ esforço de produção estimado na conclusão]

Estabilidade de retrabalho = Quebra cumulativa – correções cumulativas

[3] NCNB – Conforme encontrado nos documentos do RUP NCNB é o tamanho de código sem comentários e sem espaços vazios.

Quantidade de retrabalho = [Quebra cumulativa – correções cumulativas] / Unidade NCNB testada

O andamento do projeto deve ser relatado a partir do plano do projeto, e seu status é definido usando métricas de conclusão de artefatos, com um peso específico obtido na perspectiva do valor atribuído.

6.3.4. Métricas para os recursos

O RUP especifica que os itens a serem medidos incluem:

- Pessoas (Experiência, habilidade, custo e desempenho)
- Métodos e ferramentas (em termos do efeito sobre a produtividade, a qualidade e o custo)
- Tempo
- Esforço
- Orçamento (recursos utilizados, recursos restantes)

O perfil profissional deve ser registrado ao longo do tempo, mostrando o tipo (Analista, designer, programador, etc), a categoria (que implica no custo) e a equipe para a qual ele foi alocado. Tanto os dados reais quanto os planejados devem ser registrados.

Estes dados são necessários porque os métodos atuais de estimativas requerem a caracterização da experiência e da capacidade da equipe como é o caso do método COCOMO.

7. PROPOSTA INICIAL PARA USO NO PROCESSO DE DESENVOLVIMENTO DO STJ

7.1. LINHAS DE CÓDIGO

7.1.1. Como contar para softwares legados:

- Contar as linhas de código digitado pelo programador (linhas executáveis, definição de procedimentos, definição de funções, definição de classes e de definição de dados)
- Contar as linhas de comentários (somente para softwares com documentação desatualizada)
- Contar as linhas alteradas de arquivos que são incluídas por customizações de componentes em linguagens 4GL. “Exemplo: as linhas incluídas no arquivo”.*DFM”* de um formulário Delphi que não sejam criadas de um formulário vazio.
- Não contar linhas em branco, linhas de um framework definido e reutilizado por diversos softwares.

7.1.2. Como contar para softwares novos

- Contar as linhas de código digitado pelo programador (linhas executáveis, definição de procedimentos, definição de funções, definição de classes e de definição de dados).
- Não Contar as linhas de comentários ou linhas de documentação que possua o estilo javadoc. Pois os mesmos possuem documentação baseada nos artefatos do processo de desenvolvimento de software do STJ.
- Contar as linhas alteradas de arquivos que são incluídas por customizações de componentes em linguagens 4GL. Exemplo: as linhas incluídas no arquivo. DFM de um formulário Delphi que não sejam criadas de um formulário vazio.
- Não contar linhas em branco, linhas de um framework definido e reutilizado por diversos softwares.
- Não contar linhas de facilidades já definidas por uma arquitetura adotada com código reutilizável.

7.1.3. Produtos obtidos do KLOC no Tribunal:

Inicialmente o uso de KLOC deve ser empregado para medir softwares legados e para auxiliar as medições na área de sustentação destes softwares.

Os softwares legados que não possuam documentação ou que não tenham sido feitos com um processo definido ou que a documentação esteja desatualizada, deve-se aplicar a métrica de Linhas de Código para atingir 3 objetivos:

- Objetivo 1 - Medir o parque de legados e identificar o tamanho das aplicações e criar uma baseline de tamanho de software para cada software medido.
- Objetivo 2 – Controlar mensalmente o tamanho do software, gerando um gráfico de barras com a variação do mesmo durante o ano.
- Objetivo 3 - Cruzar os dados de KLOC dos softwares e número defeitos apresentados, para armazenar a densidade de defeito de cada produto, tabulando os valores mensalmente. Consolidando essa informação como uma linha no gráfico do objetivo 2.

Os valores de densidade de defeitos permissíveis que disparem uma possível refatoração do software deve ser avaliado após a mensuração do software e acumulação da média de defeitos por um período de 3 meses, este número mágico de 3 meses é uma meta que o grupo acha interessante para agilizar o processo de definição dos parâmetros de qualidade, porém deve ser avaliado o impacto de uma refatoração baseado no tamanho de cada produto.

Caso em 3 meses não se obtenha valores confiáveis da densidade de erros, pode-se estender este período por mais 3 meses, não prorrogáveis.

A conclusão esperada desses 3 objetivos é classificar e mensurar o patrimônio de código em legados, identificar os legados que precisam ser refatorados seja por alta taxa de defeitos recorrentes ou por alterações de requisitos identificada como erro, e finalmente estabelecer o indicador de densidade de defeitos admissíveis pelo processo de qualidade adotado pelo STJ.

7.2. USO DE APF NO TRIBUNAL:

Aplicável apenas para softwares novos ou em refatoração e softwares cujo escopo e requisitos estão sob controle e a documentação está atualizada. A sua aplicação ocorre de forma com que seja dividida em quatro passos citados a seguir:

Passo 1- Detectar Variação do Tamanho - mensurar o software pelas funcionalidades através da técnica de Análise de Pontos de Função em 2 momentos:

- INICIAL - De forma Estimada para medir o tamanho do software ou funcionalidade ainda não existente.
- FINAL - De forma Detalhada para medir o tamanho real do software a ser colocado em produção.

Registrar a variação de cada produto em indicadores de qualidade que o Tribunal deverá adotar.

VARIAÇÃO DO TAMANHO = ((TAMANHO FINAL/TAMANHO INICIAL) -1)*100

Passo 2 - Tamanho do PF em LOC - derivar uma métrica de Pontos de Função x Linha de Código, ou seja, através do cruzamento do total de PF alcançados por um software ou funcionalidade, contabilizar os valores de LOC do mesmo

Esse valor derivado pode ser utilizado no futuro para auxiliar, de forma indicada, PFxKLOC para manutenção por indicativa de software.

TAMANHO DO PONTO FUNÇÃO = # LOC/ # PF DETALHADO

Passo 3 – Densidade de Erro- derivar uma métrica de Pontos de Função x Erro:

Através do cruzamento do total de PF alcançados por um software ou funcionalidade, depois de colocado em produção, deve ser coletado os erros apresentados.

- Mensalmente, deve ser reportada a densidade de erro por PF.
- Após um período médio de três meses, deve ser obtida a média de erros e assim estabelecer o percentual aceitável de densidade de erros por PF que a instituição deve ter em seus produtos.

DENSIDADE DE ERRO = # ERROS DO MÓDULO / # PF DO MÓDULO

Passo 4 – caso o STJ venha utilizar um *outsourcing* em fábrica externa para sustentação de seus legados, pode ser utilizada a contagem indicativa, aquela baseada apenas nos ALIs e AIEs, para indicar o tamanho do software a ser dada a sustentação externa.

Um fator que possa ajustar o processo de contagem indicada é utilizar as informações obtidas no passo 2 deste roteiro para verificar se realmente a contagem indicada para 35 PF por ALI e 15 PF por AIE é válido para os softwares que o STJ desenvolve.

Após esse passo, quando o software tiver sido absorvido pela fábrica externa, o contrato deverá executar as fases de documentação e identificação com validação dos requisitos, regras de negócio junto ao usuário. E todo o processo de manutenção evolutiva deste projeto deve utilizar os padrões normais para manutenção de softwares que tenham sido desenvolvidos já utilizando a APF para definição do tamanho (contagem estimada prévia e contagem detalhada ao final), além da confecção de todos os artefatos indicados pelo processo do STJ.

Outro ponto importante que o STJ deverá definir é o uso de índices deflatores para manutenções evolutivas e corretivas fora do prazo de garantia dos produtos. E um possível índice inflator para compensar o esforço de fábricas externas em tecnologias que não sejam especificadas em contrato de *outsourcing*.

7.3. MÉTRICA DE SATISFAÇÃO DO CLIENTE

Para verificar a satisfação dos usuários do STJ para com seus softwares desenvolvidos é interessante obter dados sobre:

- O Software
- Requisitos não funcionais declarados como uma exigência do cliente

Os dados obtidos devem utilizar a seguinte escala:

- Muito Satisfeito
- Satisfeito
- Neutro
- Insatisfeito
- Muito Insatisfeito

Os valores percentuais obtidos por este questionário devem ser obtidos e depois publicados como indicadores do desempenho da área de desenvolvimento sob o ponto de vista do usuário.

O cuidado que se deve ter é quando um software entra em declínio na aceitação, pois pode indicar outros dados como alteração de requisitos que o usuário entende como sendo um

erro ou quando a equipe em questão que dá suporte ao sistema está com problemas internos ou sobrecarregados por outros tipos de demandas de prioridade maior.

Vale lembrar que essa métrica também é um marketing da informática com o usuário, quando bem utilizada.

7.4. MÉTRICA DE QUALIDADE DO PRODUTO

O STJ deve evitar o MTTF, pois o mesmo não é adequado para os tipos de softwares criados pelo STJ e os resultados podem ser de pouca utilidade como indicadores de Qualidade.

Já a métrica de densidade de defeitos por KLOC ou PF é muito útil, principalmente quando se verifica se o defeito indicado pelo usuário é realmente um defeito ou falha do software ou se é na verdade um requisito que o usuário modificou, não reportou a equipe de desenvolvimento de software e classifica-o como um defeito.

Densidade de defeitos é interessante para avaliar a qualidade do desenvolvimento de software. Pois, para um software em inúmeras releases, pode ser definido valores cada vez menores de aceitação de erros e esse nível de exigência da própria informática quanto ao versionamento de seus produtos pode melhorar a perspectiva e percepção de qualidade do usuário face ao produto ou solução.

7.5. MÉTRICA DE ORIENTAÇÃO A OBJETOS

Para auxiliar na validação de suas arquiteturas de software aderentes ao modelo de Orientação a Objetos, devem ser utilizadas as métricas de Lorenz catalogadas por Kan [2].

As mesmas são interessantes para obter dados dos softwares produzidos com este paradigma. Todos 11 pontos definidos por essa métrica são úteis como indicadores de qualidade do produto e servem para medir a aderência da própria arquitetura no produto, arquitetura essa que prega o uso de camadas e é aderente ao paradigma OO.

É interessante que o STJ utilize-a como um dos pontos de auditoria do código entregue pela empresa terceirizada que estiver a desenvolver produtos para o STJ.

Os valores obtidos desse conjunto de métricas deve ser armazenado e para os itens 1,2,6 e 9, como dito anteriormente por esse trabalho, durante 3 meses dados devem ser coletados para definir os valores permissíveis para cada um dos itens.

8. CONCLUSÕES

As métricas possíveis de serem aplicadas no Superior Tribunal de Justiça são de uma diversidade muito grande, portanto o estudo da arte realizado neste trabalho serviu como base para que fosse feito uma proposta inicial do que se pode utilizar para iniciar um processo de métricas na coordenadoria de desenvolvimento.

É interessante que a APF seja usada amplamente e que seja passado para o usuário quanto que seria o custo de criação de um produto pelos profissionais da casa. Ou até mesmo que o STJ criasse a idéia de central de custos, para identificar quanto que custa por ano para o Tribunal a disponibilização de recursos de TI em software.

Várias métricas e definições foram encontradas neste estudo e ajudaram bastante no entendimento do problema de se implantar um processo de métricas em um setor de desenvolvimento de software. Outro fato importante que se pode citar é a complexidade das métricas aplicadas em software.

Uma das conclusões a que se foi possível chegar também é que para alcançar um nível de qualidade aceitável é necessário se trabalhar bastante com a área de qualidade e refinar as métricas utilizadas, bem como utilizar meios adequados para a coleta e interpretação dos dados necessários.

Dentre as métricas que mais foram focadas no trabalho estão as métricas de Análise de Ponto por Função – APF e as métricas de linhas de código – KLOC que servem necessariamente para medir o tamanho funcional de um software. Essas duas métricas são consideradas métricas primitivas, pois elas podem ser derivadas gerando outras métricas mais refinadas e específicas a cada caso.

Além disso, o STJ possui um parque de softwares legados muito grande, que foram desenvolvidos sem um processo de desenvolvimento de software definido, fazendo com que seja mais difícil aplicar qualquer processo de métricas. Desta forma em nossa proposta sugeriu-se que sobre os legados em uso no STJ fosse adotada a métrica de KLOC como uma métrica para seus legados que não possuam documentação ou cuja documentação está excessivamente desatualizada, seja devido às mudanças de requisitos ou outros fatores.

No estudo sobre KLOC foi encontrado ainda modelos sugeridos de quais as linhas a serem contabilizadas pelo uso da KLOC.

Quanto aos novos produtos desenvolvido pelo STJ a métrica KLOC também deve ser considerada, pois é uma métrica útil para cruzar com os dados obtidos da APF.

Conclui-se também que a APF também deve ser utilizada como métrica no STJ, e alguns cuidados devem ser tomados para se fazer a contagem estimada antes da construção do projeto e que ao término do projeto seja feita outra contagem detalhada. Desta forma a estimativa pode ser confrontada com a contagem final, obtendo se um *feedback* para que seja possível avaliar e revisar o processo de contagem estimada.

Apesar de terem sido também estudadas as métricas OO as mesmas devem ser utilizadas no STJ apenas aos produtos desenvolvidos sob o paradigma OO. Portanto, a sugestão inicial seria pegar três próximos projetos, que sejam OO e aderidos à arquitetura, e aplicar a coleta de métricas de Lorenz conforme estudado.

Quando este modelo de desenvolvimento OO estiver como padrão de software do STJ, deve-se identificar quais métricas de Chidamber e Kemerer que poderiam auxiliar na otimização de algum produto de missão crítica.

A pretensão de todo este estudo foi montar um conjunto de métricas que sejam aplicáveis ao setor de desenvolvimento de software do STJ, baseando no processo de desenvolvimento do STJ. E com isto dar subsídio às áreas responsáveis pela Qualidade de software da instituição.

9. BIBLIOGRAFIA

[1] GALORATH, Daniel D.; EVANS, Michael W. Software Sizing Estimation, and Risk Management . Auerbach Publications – New York 2006

[2] KAN, STEPEN H. *Metrics and Models in Software Quality Engineering - Second Edition*, september 2002, Addison Wesley

[3] JONES, Capers. *Programming Productivity*, January 1986, McGraw-Hill Companies

[4] CONTE, Samuel Daniel. *Software Engineering Metrics and Models*, march 1986, Benjamin-Cummings Pub Co

[5] VASQUEZ, Carlos Eduardo, et al. *Análise de Pontos de Função*, 2003, Editora Erica.

[6] MAIA, José Ricardo Correia. **Use métricas adequadas – Garanta a qualidade de projeto orientado a objeto**, EUAX gestão de projetos. 2006. Disponível em: http://www.euax.com.br/art.00.index.shtml Acessado em: 05/03/2008

[7] IEEE Std 982.2-1988, IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software

[8] IEEE Std 982.1-1988, IEEE Standard Dictionary of Measures to Produce Reliable Software

[9] SANTOS FILHO, Firminio do. **Métricas e qualidade de software.** São Paulo: Software quality consulting, 2002. Disponível em http://paginas.terra.com.br/informatica/SoftwareQuality/ , Acessado em: 10/03/2008.

[10] IEEE Std 1045-1992, IEEE Standard for Software Productivity Metrics

[11] SOMMER, Willy Arno. Avaliação da qualidade. Apostila da disciplina de Avaliação da Qualidade. Universidade Federal de Santa Catarina, 2000.

[12] RUSSO, Suzana Leitão. GRÁFICOS DE CONTROLE PARA VARIÁVEIS NÃO-CONFORMES AUTOCORRELACIONADAS, FLORIANÓPOLIS, 2002. Disponível em: http://www.qualimetria.ufsc.br/teses_arquivos/suzana.pdf

[13] VIEIRA, Sônia.; WADA, Ronaldo. *As 7 Ferramentas Estatísticas para o Controle da Qualidade. Brasília*. QA&T Consultores Associados Ltda., 1994. 133p.

[14] SELNER, Claudiomir. ANÁLISE DE REQUISITOS PARA SISTEMAS DE INFORMAÇÕES, UTILIZANDO AS FERRAMENTAS DA QUALIDADE E PROCESSOS DE SOFTWARE. Santa Catarina, 1999.

[15] CAMPOS, Fábio Martinho. **Métricas de Software Como Ferramenta de Apoio ao Gerenciamento de Projetos de Software.** Linha de código, 2004. Disponível em: http://www.linhadecodigo.com.br/Artigo.aspx?id=453, Acessado em1704/2008.

[16] CALSAVARA, Alcides et al. **Aderência do RUP à norma NBR ISO/IEC 12207.** Companhia de Informática do Paraná – CELEPAR 2003. Disponível em : http://www.pr.gov.br/batebyte/edicoes/2000/bb104/software.htm. Acessado em: 16/04/2008

[17] MELLO FILHO, Moacyr Cardoso de. Aplicando as **Sete Métricas de Controle de Projeto**. Rational Software White Paper, 2002.

10.ANEXO

10.1. CONTRATO STJ Nº 067/07 - PROCESSO STJ Nº 8122/2006

Prestação de serviços técnicos de informática a serem desenvolvidos sob a modalidade de Fábrica de Software.

Pelo presente instrumento e na melhor forma de direito, as partes abaixo qualificadas têm entre si justo e avençado o objeto a seguir descrito, com fundamento nas disposições do art. 23, II, "c", da Lei nº 8.666/93, do Decreto 1070/94 e mediante as seguintes cláusulas e condições:

CONTRATANTE:

SUPERIOR TRIBUNAL DE JUSTIÇA - STJ, Órgão integrante do Poder Judiciário da União, inscrito no CNPJ/MF sob o nº 00.488.478/0001-02, com sede no SAF Sul, Quadra 06, Lote 01, Brasília-DF, neste ato representado por seu Diretor-Geral, **MIGUEL AUGUSTO FONSECA DE CAMPOS,** brasileiro, inscrito no Cadastro de Pessoas Físicas do Ministério da Fazenda sob o nº 004.881.942-53, portador da Cédula de Identidade nº 782.043, expedida pela SSP/PA, residente e domiciliado nesta Capital.

CONTRATADA:

POLITEC TECNOLOGIA DA INFORMAÇÃO S.A., pessoa jurídica de direito privado, inscrita no CNPJ/MF sob o nº 01.645.738/0001-79, com sede na Avenida República do Líbano, nº 838, Setor Aeroporto, Goiânia/GO, neste ato representada por seu Diretor Comercial, **HÉLIO SANTOS OLIVEIRA**, brasileiro, inscrito no Cadastro de Pessoas Físicas do Ministério da Fazenda sob o nº 076.211.911-04, portador da Cédula de Identidade nº 134.367, expedida pela SSP/DF, e por seu Diretor Regional de Brasília, **WOLNEY MENDES MARTINS,** brasileiro, inscrito no Cadastro de Pessoas Físicas do Ministério da Fazenda sob o nº 184.958.931-34, portador da Cédula de Identidade nº 424.394, expedida pela SSP/DF residentes e domiciliados nesta Capital.

CLÁUSULA PRIMEIRA - DO OBJETO

1.1 – Constitui o objeto do presente contrato a prestação de serviços técnicos de informática a serem desenvolvidos sob a modalidade de Fábrica de Software.

1.2 – As especificações técnicas e detalhamentos constantes do edital de licitação aderem a este contrato e dele fazem parte, independentemente de transcrição.

CLÁUSULA SEGUNDA – DOS SERVIÇOS

2.1 - O serviço de fabricação de software poderá ser aplicado a produção de software novo ou alteração de software já existente, tratada como manutenção.

2.2 - A fabricação de software pela CONTRATADA será dividida em fases, cada uma possuindo subprodutos próprios.

2.3 – A distribuição de fases permeiam todo o processo produtivo, amadurecendo os produtos gerados mediante diversas iterações ao longo do tempo, conforme representação nas seguintes tabelas:

Distribuição das disciplinas por fase				
Disciplina/Fase	Concepção	Elaboração	Construção	Transição
Gerência de Projeto	14,0%	12,0%	10,0%	14,0%
Configuração e mudança	10,0%	8,0%	5,0%	5,0%
Requisitos	38,0%	18,0%	8,0%	4,0%
Análise e Projeto	19,0%	36,0%	16,0%	4,0%
Implementação	8,0%	13,0%	34,0%	19,0%
Testes	8,0%	10,0%	24,0%	24,0%
Implantação	3,0%	3,0%	3,0%	30,0%
Total	100%	100%	100%	100%

Distribuição do esforço por fase					
Fase	Concepção	Elaboração	Construção	Transição	Total
Esforço	5%	20%	65%	10%	100%

Distribuição de esforço por disciplina e fase					
Disciplina/Fase	Concepção	Elaboração	Construção	Transição	Total
Gerência de Projeto	0,70%	2,40%	6,50%	1,40%	11,00%
Configuração e mudança	0,50%	1,60%	3,25%	0,50%	5,85%
Requisitos	1,90%	3,60%	5,20%	0,40%	11,10%
Análise e Projeto	0,95%	7,20%	10,40%	0,40%	18,95%
Implementação	0,40%	2,60%	22,10%	1,90%	27,00%
Testes	0,40%	2,00%	15,60%	2,40%	20,40%
Implantação	0,15%	0,60%	1,95%	3,00%	5,70%
Total	5,00%	20,00%	65,00%	10,00%	100,00%

2.4 – O CONTRATANTE definirá, a seu critério, qual será a concentração do trabalho da CONTRATADA dentre as iterações apresentadas no item 2.3.

2.5 - Os serviços de fabricação de software terão seu valor definido com base na complexidade de execução, mensurada por meio da contagem de pontos de função, utilizando a metodologia descrita na versão 4.2.1 ou mais recente do "Manual de Práticas de Contagens por Pontos de Função" (Function Point Counting Practices Manual), publicado pelo IFPUG (International Function Point Users Group) e disponibilizado no Brasil pelo BFPUG (Brazilian Function Point Users Group).

2.6 - A CONTRATADA entregará ao CONTRATANTE cópia do "Manual de Práticas de Contagens por Pontos de Função" publicado pelo IFPUG, bem como promoverá novo fornecimento sempre que uma nova versão for adotada para definição da complexidade dos serviços de fabricação de software.

2.7 - O CONTRATANTE indicará, a seu exclusivo critério, a adoção de uma nova versão do "Manual de Práticas de Contagens por Pontos de Função" que venha a ser disponibilizada pelo IFPUG.

2.8 - O desenvolvimento dos produtos e a fabricação do software levarão em consideração o ambiente computacional do CONTRATANTE.

2.9 - O CONTRATANTE repassará à execução pela CONTRATADA apenas as fases da fabricação de software que julgar conveniente.

2.10 - A CONTRATADA utilizará e disponibilizará ao CONTRATANTE ferramenta de apoio à contagem de pontos de função referente ao serviço das OS, permitindo:

a) registro dos pontos de função contados pelos especialistas;

b) classificação da complexidade dos elementos identificados na contagem;

c) totalização dos pontos de função;

d) apoio ao cálculo do fator de ajuste e ao cálculo dos pontos de função ajustados;

e) armazenamento do histórico dos pontos de função contados;

f) emissão de relatórios das operações efetuadas.

2.11 - A licença de uso da ferramenta de apoio à contagem de pontos de função fornecida pela CONTRATADA pertencerá ao CONTRATANTE após o término do contrato, de forma a preservar o acesso aos dados históricos acumulados durante a prestação de serviços.

2.12 - A CONTRATADA utilizará e disponibilizará ao CONTRATANTE acesso a ferramenta de gerência de projeto que possibilite o acompanhamento, pela internet, do estágio de execução dos serviços;

CLÁUSULA TERCEIRA – DA EXECUÇÃO

3.1 - As solicitações de fabricação de software serão submetidas à CONTRATADA através de Ordens de Serviço (OS) específicas.

3.2 - Antes da emissão de uma OS, o CONTRATANTE emitirá uma Solicitação de Proposta (SdP) para a CONTRATADA, indicando claramente o serviço desejado.

3.3 - Cada OS solicitará a execução de parte ou da íntegra dos serviços de uma SdP, indicando o número de pontos de função contados pelo CONTRATANTE para o escopo a ser executado, as fases a serem repassadas à CONTRATADA e o prazo para conclusão do serviço, trazendo em anexo os artefatos produzidos em fases anteriores, quando for o caso, e a contagem de pontos de função emitida pela CONTRATADA em resposta à SdP.

3.4 - Nos casos de OS de manutenção, onde as características do software impliquem um escopo de contagem amplo, mas o impacto das alterações seja pequeno, pode ser acordado entre as partes a aplicação de um índice deflator, destinado a minimizar as distorções na determinação da complexidade do serviço.

3.5 - A quantidade de pontos de função e o prazo de execução definidos numa OS podem ser redimensionados, com a possibilidade de exclusão de tarefas não realizadas, inclusão de novas tarefas ou ajustes referentes a complexidade não identificada no momento da emissão da SdP, desde que as partes estejam de comum acordo. Neste caso, será necessária a abertura de uma nova OS em aditamento à anterior.

3.6 - A CONTRATADA designará profissional com certificação Project Management Professional – PMP e, no mínimo, 2 (dois) anos de experiência em gerência de projetos de software, para gerenciar a execução de cada OS com prazo de execução superior a 30 dias corridos, o qual manterá e assinará os documentos de controle indicados na metodologia de gerência de projetos do CONTRATANTE.

3.7 - O CONTRATANTE se reserva o direito de alterar a metodologia de gerência de projetos a qualquer tempo, encaminhando a nova metodologia à CONTRATADA, que passará a adotá-la para as novas OS.

3.8 - A CONTRATADA fará comunicação formal e por escrito do cumprimento das OS emitidas pelo CONTRATANTE, entregando, juntamente com o comunicado, todos os produtos requeridos para as fases de produção de software abrangidas pela OS.

3.9 - As inconformidades encontradas nos produtos entregues serão comunicadas formalmente e por escrito à CONTRATADA.

3.10 - Caso uma inconformidade impeça o prosseguimento da homologação dos produtos entregues, esse fato será indicado no comunicado de inconformidade e a contagem de prazo será interrompida até que impedimento seja resolvido.

CLÁUSULA QUARTA – DA GARANTIA

4.1 - Os softwares fabricados pela CONTRATADA terão garantia contra defeitos por 180 (cento e oitenta) dias corridos, a contar da data do aceite, dentro dos quais a CONTRATADA corrigirá os defeitos identificados sem custos para o CONTRATANTE.

4.2 - São considerados defeitos as falhas provocadas pela operação normal do produto e os comportamentos que estejam em desacordo com os requisitos estabelecidos ou com as especificações do software.

CLÁUSULA QUINTA – DOS PRAZOS

5.1 - A CONTRATADA terá 5 (cinco) dias úteis para responder a uma SdP, estimando a complexidade do serviço solicitado com base na contagem de pontos de função, a qual será formulada e assinada por profissional com certificação Certified Function Points Specialist – CFPS.

5.2 - A CONTRATADA terá 15 (quinze) dias corridos para apresentar questionamento formal e por escrito quanto ao prazo de execução ou ao número de pontos de função contados pelo CONTRATANTE para uma OS, não sendo interrompida a contagem de prazo de entrega dos serviços durante o lapso temporal para questionamento ou mesmo na ocorrência de um questionamento por parte da CONTRATADA.

5.3 - O redimensionamento tratado no item 3.5 só poderá ser pactuado com antecedência de, no mínimo, 15 (quinze) dias corridos antes da data de conclusão das atividades, ficando esse prazo estabelecido em 5 (cinco) dias úteis para as Ordens de Serviço com prazo igual ou inferior a 30 (trinta) dias corridos.

5.4 - O CONTRATANTE terá o prazo de 15 dias úteis, de acordo com seu calendário oficial, para emitir aceite definitivo dos produtos entregues.

5.5 - Caso haja substituição de produtos a menos de 5 (cinco) dias úteis do final do prazo de homologação, o prazo restante passará a ser de 5 (cinco) dias úteis, sendo a contagem de prazo reiniciada e a CONTRATADA comunicada apenas caso o produto substituído comprometa a avaliação já realizada.

5.6 - Nas OS com prazo de execução superior a 60 dias, a entrega dos produtos poderá ser efetuada de forma parcelada, sendo o aceite emitido com relação a cada lote de produtos entregue, desde que essa forma de entrega tenha sido indicada originalmente na OS.

5.7 - O prazo para correção de defeito de software será estabelecido com base na severidade do incidente:

a) Severidade ALTA: Defeito que impeça a utilização do software ou de funcionalidade indispensável a este, comprometendo de forma crítica uma atividade de negócio do CONTRATANTE, tendo a CONTRATADA 1 (um) dia útil, a contar da comunicação da falha pelo CONTRATANTE, para sanar o problema ou executar ação paliativa que coloque o incidente em severidade média;

b) Severidade MÉDIA: Defeito que comprometa a utilização do software ou de parte deste, prejudicando a produtividade de uma atividade de negócio do CONTRATANTE de forma claramente observável, tendo a CONTRATADA 5 (cinco) dias úteis, a contar da comunicação da falha pelo CONTRATANTE, para sanar o problema ou executar ação paliativa que coloque o incidente em severidade baixa;

c) Severidade BAIXA: Defeito do software que não comprometa significativamente uma atividade de negócio do CONTRATANTE, tendo a CONTRATADA 20 (vinte) dias corridos, a contar da comunicação da falha pelo CONTRATANTE, para sanar o problema.

5.8 - Caso a solução apresentada pela CONTRATADA não repare o defeito, nem surta o efeito paliativo a que se propôs, a CONTRATANTE fará a comunicação do não aceite da solução e a contagem do tempo de correção será retomada do ponto e severidade em que foi interrompida.

5.9 - A identificação e a comunicação formal de defeito de software deverão ser feitas dentro do prazo de garantia, devendo a correção ser realizada ainda que a conclusão do serviço extrapole o prazo de garantia.

CLÁUSULA SEXTA - DO DIREITO DE PROPRIEDADE E CONFIDENCIALIDADE

6.1 - O CONTRATANTE terá o direito de propriedade sobre todos os produtos desenvolvidos pela CONTRATADA por meio de Ordens de Serviços decorrentes deste contrato, sendo vedada qualquer comercialização por parte da CONTRATADA.

6.2 - A CONTRATADA obriga-se a tratar como "segredos comerciais e confidenciais" quaisquer informações, dados, processos, fórmulas, códigos, entre outros, obtidos em conseqüência ou por necessidade da execução de OS, utilizando-os apenas para as finalidades previstas no contrato, não podendo revelá-los ou facilitar sua revelação a terceiros.

CLÁUSULA SÉTIMA - DA RELAÇÃO EMPREGATÍCIA E DOS ENCARGOS SOCIAIS

7.1 - As partes desde já ajustam que não existirá para o CONTRATANTE solidariedade quanto ao cumprimento das obrigações trabalhistas e previdenciárias para com os empregados da CONTRATADA, cabendo a esta assumir, de forma exclusiva, todos os ônus advindos da relação empregatícia.

CLÁUSULA OITAVA - DO PREÇO E DA REPACTUAÇÃO

8.1 – As partes estipulam que os preços dos pontos de função para cada fase da prestação de serviços da fabricação de um software são os discriminados a seguir:

	Concepção	Elaboração	Construção	Transição	Total
Preço do Ponto de Função (R$)	22,44	89,75	291,69	44,88	448,75

8.2 - O custo total de uma OS será determinado pelo número de pontos de função da OS, multiplicado pelo valor do ponto de função estabelecido para a fase e pelo índice deflator, quando houver, efetuando-se a soma desse resultado para quantas forem as fases abrangidas pela OS.

8.3 - Os preços ajustados são finais e definitivos, neles estando inclusos todos os encargos que a CONTRATADA experimentará no cumprimento das obrigações assumidas.

8.4 – Os preços descritos no item 8.1 poderão ser repactuados, mediante negociação entre as partes, observado o interregno mínimo de 1 (um) ano, a contar da data do orçamento a que a proposta se referir, cabendo à CONTRATADA apresentar, junto à solicitação, a devida justificativa e demonstração analítica da variação dos componentes de custo do contrato, de acordo com planilha de custos e formação de preços, acordo, convenção ou dissídio coletivo de trabalho ou equivalente, com a comprovação de registro na Delegacia Regional do Trabalho, entre outros, visando à análise e aprovação pelo CONTRATANTE.

8.5 – Para os fins previstos no item anterior, considera-se como data do orçamento a que a proposta se referir, a data do início da vigência do acordo, convenção ou dissídio coletivo de trabalho ou equivalente, que estipular o salário vigente à época da apresentação da proposta.

8.6 – Ocorrendo a primeira repactuação, as subseqüentes só poderão ocorrer obedecendo ao prazo mínimo de 1 (um) ano, a contar do início dos efeitos da última repactuação.

8.7 – Por ocasião da repactuação, poderão ser contemplados todos os componentes de custo do contrato que tenham sofrido variação, desde que haja uma demonstração analítica devidamente justificada e comprovada.

8.8 – Não é admitida a inclusão, por ocasião da repactuação, de qualquer item de custo não previsto nos componentes apresentados originariamente.

CLÁUSULA NONA - DO VALOR DO CONTRATO E DA DOTAÇÃO ORÇAMENTÁRIA

9.1 – As partes ajustam que o valor do presente contrato fica estimado em R$ 3.231.000,00 (três milhões, duzentos e trinta e um reais), conforme descrito a seguir:

Quantidade estimada de Pontos de Função	Valor Unitário (R$)	Valor Total (R$)
7.200	448.75	3.231.000,00

9.2 - As despesas com a execução deste contrato serão atendidas com os recursos consignados ao CONTRATANTE no Orçamento Geral da União e suplementações a ele incorporadas.

9.3 - Foi emitida em 28/05/2007 a Nota de Empenho Estimativo nº 2007NE001201, no valor de R$ 1.884.750,00 (um milhão, oitocentos e oitenta e quatro mil, setecentos e cinqüenta reais), à conta da seguinte dotação orçamentária: Programa de Trabalho 02.126.0568.1H24.0001 e Natureza da Despesa 3.3.90.39.

9.4 - Observado o disposto nos §§ 1º e 2º do artigo 65 da Lei nº 8.666/93, poderá o CONTRATANTE, durante a vigência do contrato e mediante aditivo próprio, efetuar acréscimos ou supressões no presente ajuste, respeitada a natureza do objeto contratual.

CLÁUSULA DÉCIMA - DA GARANTIA CONTRATUAL

10.1 - Para o fiel cumprimento das obrigações ora assumidas, a CONTRATADA entregará ao CONTRATANTE, no prazo máximo de 20 (vinte) dias úteis contados da data da assinatura do contrato, garantia

no valor de R$ 161.550,00 (cento e sessenta e um mil, quinhentos e cinqüenta reais), nos termos do artigo 56, § 1º, incisos I, II e III, da Lei nº 8.666/93.

10.2 – Ao CONTRATANTE é reservado o direito de somente liberar a garantia de que trata o item anterior, no prazo de 30 (trinta) dias corridos, contado do término da vigência deste contrato, caso haja adimplemento total de seu objeto.

10.3 – O CONTRATANTE poderá descontar da garantia os valores que a CONTRATADA passe a lhe dever em virtude da ocorrência de qualquer das situações expressamente previstas neste contrato.

10.4 – Caso o valor da garantia venha a ser utilizado em pagamento de qualquer obrigação, desde que atribuída à CONTRATADA, esta se obriga a efetuar a respectiva reposição no prazo máximo de 48 (quarenta e oito) horas, a contar da data do recebimento da comunicação por parte do CONTRATANTE.

10.5 – A garantia a que se refere o item 10.1 corresponderá a 5% (cinco por cento) do valor atribuído ao contrato, sendo atualizada nas mesmas condições deste.

10.6 - No caso da CONTRATADA optar pela caução em dinheiro, esta deverá ser feita na Caixa Econômica Federal, conforme Decreto-Lei nº 1.737, de 21/12/1979.

CLÁUSULA DÉCIMA PRIMEIRA – DO FATURAMENTO E DO PAGAMENTO

11.1 – Os pagamentos serão devidos mensalmente, em contraprestação aos serviços efetivamente prestados, e serão efetuados mediante crédito em conta-corrente da CONTRATADA, produzindo o depósito bancário correspondente, os efeitos jurídicos de quitação da prestação devida.

11.2 – Para os fins previstos no item anterior, a CONTRATADA deverá apresentar, no primeiro dia útil de cada mês, documento de cobrança referente aos serviços prestados no mês imediatamente anterior, bem como informará o banco, a agência e o número da conta-corrente em que o crédito deverá ser efetuado.

11.3 – O documento de cobrança deverá ser apresentado acompanhado de:

a) Demonstrativo de Contabilização dos esforços realizados nas Ordens de Serviços encerradas e devidamente aceitas no mês anterior ao do faturamento;

b) Demonstrativo de Contabilização dos esforços realizados no mês anterior ao de faturamento para Ordens de Serviços aceitas parcialmente.

11.4 - As notas fiscais de cobrança, bem como os documentos que devem acompanhá-las, serão entregues pela CONTRATADA no Protocolo Administrativo do CONTRATANTE, situado no Setor de Administração Federal Sul, Quadra 06, Lote 01, Bloco "B", Edifício dos Plenários, Térreo, Sala B-003, Brasília/DF.

11.5 - Caberá ao gestor do contrato atestar os documentos de cobrança e encaminhá-los à Coordenadoria de Orçamento e Finanças do CONTRATANTE para fins de pagamento.

11.6 – Obedecida a ordem de exigibilidade dos créditos, de acordo com o artigo 5º da Lei nº 8.666/93, o CONTRATANTE efetuará o pagamento dos serviços, no prazo de até 10 (dez) dias úteis contados do primeiro dia útil subseqüente ao da atestação do documento de cobrança.

11.7 - O inadimplemento do pagamento na data aprazada, conforme disposto no item 11.6, desde que motivado pelo CONTRATANTE, acarretará a correção monetária do valor devido, calculada *pro rata tempore*, até a data do efetivo pagamento, com base no último percentual divulgado do IGP/DI-FGV, a ser cobrada na fatura do mês subseqüente.

11.8 – No caso de cobrança referente a serviços que porventura deixaram de ser faturados na época devida, os valores a serem cobrados serão os vigentes na data em que ocorreram.

11.9 - Caso o objeto contratado seja faturado em desacordo com as disposições previstas neste contrato ou sem a observância das formalidades legais pertinentes, a CONTRATADA deverá emitir e apresentar novo documento de cobrança.

11.10 – Ocorrendo a hipótese prevista no item 11.9 o prazo para o pagamento do novo documento de cobrança obedecerá a regra estabelecida no item 11.6.

11.11 - O atraso no pagamento decorrente das circunstâncias descritas no item anterior, não exime a CONTRATADA de promover o pagamento dos profissionais nas datas regulamentares.

11.12 - Por ocasião da assinatura do contrato, bem como da realização de cada pagamento, a CONTRATADA deverá estar em situação regular junto ao INSS, ao FGTS e à Fazenda Nacional.

CLÁUSULA DÉCIMA SEGUNDA - DA VIGÊNCIA

12.1 – O prazo de vigência do presente contrato é de 12 (doze) meses, contados a partir da data de assinatura, podendo ser prorrogado por iguais e sucessivos períodos, até o limite de 60 (sessenta) meses.

12.2 – A prorrogação da vigência do presente contrato em exercícios subseqüentes ficará condicionada à avaliação da qualidade dos serviços prestados, à comprovação da compatibilidade dos preços conforme o mercado, bem como à existência, em cada ano, de dotação orçamentária para suportar as despesas dele decorrentes.

CLÁUSULA DÉCIMA TERCEIRA – DAS RESPONSABILIDADES DAS PARTES

13.1 - Além das demais obrigações expressamente previstas neste contrato e de outras decorrentes da natureza do ajuste, deverá a CONTRATADA:

a) indicar um representante e um eventual substituto, que deverá estar disponível nas dependências do CONTRATANTE, nos dias úteis, no horário de 9h às 19h, e acessível através de contato telefônico em qualquer outro horário, com vistas a:

I) acompanhar a execução das ordens de serviço em vigor;

II) assegurar-se de que as determinações do CONTRATANTE sejam disseminadas junto à CONTRATADA, com vistas à alocação dos profissionais necessários para execução das Ordens de Serviço;

III) informar ao CONTRATANTE sobre problemas que possam impedir o bom andamento dos serviços;

IV) elaborar documentos (relatórios gerenciais e outros) referentes ao acompanhamento da execução das Ordens de Serviço;

V) executar os procedimentos administrativos referentes aos recursos alocados para prestação dos serviços contratados.

b) formalizar a indicação do representante junto ao CONTRATANTE e contar com a anuência deste;

c) atender às instruções do CONTRATANTE quanto à execução e horários de realização dos serviços, permanência e circulação de pessoas nos prédios do Tribunal;

d) responsabilizar-se pelos materiais, produtos, ferramentas, instrumentos e equipamentos disponibilizados para a execução dos serviços, não cabendo ao CONTRATANTE qualquer responsabilidade por perdas decorrentes de roubo, furto ou outros fatos que possam vir a ocorrer;

e) cumprir as normas relacionadas com a segurança e higiene no trabalho;

f) promover o afastamento, no prazo máximo de 24 (vinte e quatro) horas após o recebimento de notificação, de qualquer dos seus empregados que não corresponder aos critérios de confiança ou que perturbe a ação da equipe de fiscalização do CONTRATANTE;

g) apresentar seus empregados com pontualidade, de acordo com os horários fixados pelo CONTRATANTE, para fins da execução dos serviços contratados;

h) implantar adequadamente o planejamento, a execução e a supervisão permanente dos serviços, de forma a obter uma operação correta e eficaz, realizando os serviços de forma meticulosa e constante, mantendo sempre em perfeita ordem todas as dependências do CONTRATANTE;

i) prestar os serviços dentro dos parâmetros e rotinas estabelecidos neste contrato, com observância às recomendações aceitas pela boa técnica, normas e legislação, bem como observar conduta adequada na utilização dos materiais, equipamentos, ferramentas e utensílios;

j) responsabilizar-se pela limpeza e conservação dos ambientes onde desempenhe seus serviços;

k) comunicar à unidade do CONTRATANTE responsável pela fiscalização do contrato, por escrito, qualquer anormalidade de que tenha conhecimento na execução do mesmo;

l) responsabilizar-se por danos causados ao patrimônio do CONTRATANTE, ou de terceiros, ocasionados por seus empregados, em virtude de dolo ou culpa, durante a execução do objeto contratado;

m) submeter seus empregados aos regulamentos de segurança e disciplina instituídos pelo CONTRATANTE, durante o tempo de permanência em suas dependências;

n) manter, durante todo o período de vigência do ajuste, todas as condições que ensejaram sua contratação;

13.2 - Além das demais obrigações previstas neste contrato e de outras decorrente da natureza do ajuste, deverá o CONTRATANTE:

a) emitir Ordens de Serviço para a execução do objeto da presente contratação;

b) efetuar o pagamento para os valores de ponto de função e de horas efetivamente utilizados;

c) assegurar o acesso dos empregados da CONTRATADA, quando devidamente identificados, aos locais em que devam executar suas tarefas;

d) prestar as informações e esclarecimentos necessários que os empregados da CONTRATADA encarregados da execução dos serviços venham a solicitar para o desenvolvimento dos trabalhos;

e) disponibilizar à CONTRATADA local adequado para a guarda de materiais e equipamentos utilizados durante a execução dos serviços;

f) fiscalizar o cumprimento das cláusulas e condições estabelecidas neste contrato.

13.3 - Poderá o CONTRATANTE, a qualquer tempo, exigir da CONTRATADA a comprovação das condições referidas na alínea "n" do item 13.1.

CLÁUSULA DÉCIMA QUARTA - DAS PENALIDADES

14.1 - Nos termos do art. 86, da Lei nº 8.666/93, fica a CONTRATADA, em caso de atraso injustificado na execução do ajuste, sujeita à multa moratória de 0,3% (três décimos por cento) ao dia, limitada a 30 (trinta) dias, calculada sobre o valor da parcela inadimplida.

14.2 – Na hipótese do item anterior, decorrido o lapso de 30 (trinta) dias, a Unidade Gestora do CONTRATANTE deverá manifestar-se sobre o interesse na continuidade da execução do contrato.

14.3 – Não havendo mais interesse do CONTRATANTE na execução do contrato, total ou parcialmente, em razão do descumprimento, por parte da CONTRATADA, de qualquer das condições avençadas, fica

estipulada a multa compensatória de 20% (vinte por cento) sobre o valor pactuado, nos termos do inciso II, do artigo 87, da Lei nº 8.666/93.

14.4 – O disposto nos itens anteriores não prejudicará a aplicação de outras penalidades a que esteja sujeita a CONTRATADA, nos termos dos artigos 87 e 88 da Lei nº 8.666/93.

14.5 – O valor da multa aplicada, após regular procedimento administrativo, será descontado dos pagamentos eventualmente devidos pelo CONTRATANTE ou ainda, se for o caso, cobrado judicialmente.

14.6 - Excepcionalmente, *"ad cautelam"*, o CONTRATANTE poderá efetuar a retenção do valor presumido da multa, antes da instauração do regular procedimento administrativo.

14.7 – As penalidades previstas nesta cláusula poderão ser relevadas ou atenuadas pela autoridade competente aplicando-se o Princípio da Proporcionalidade, em razão de circunstâncias fundamentadas em fatos reais e comprovados.

CLÁUSULA DÉCIMA QUINTA – DA RESCISÃO

15.1 - O presente contrato poderá ser rescindido nas hipóteses estabelecidas pelos artigos 77 a 79 da Lei. 8.666/93, o que a CONTRATADA declara conhecer.

15.2 – Reserva-se ao CONTRATANTE o direito de rescindir unilateralmente este contrato ocorrendo qualquer hipótese de cisão, fusão ou incorporação que possa prejudicar a execução do objeto contratado.

15.3 - Na hipótese de a rescisão ocorrer por culpa da CONTRATADA, fica o CONTRATANTE autorizado a reter, até o limite dos prejuízos experimentados, os créditos a que aquela tenha direito.

CLÁUSULA DÉCIMA SEXTA – DAS DISPOSIÇÕES FINAIS

16.1 - A presente contratação foi precedida da Licitação nº 01/07, na modalidade Concorrência, com fundamento nas disposições do art. 23, II, "c", da Lei nº 8.666/93, do Decreto 1070/94, na autorização constante no Processo STJ nº 8122/2006 e nas condições da proposta apresentada pela CONTRATADA em 30/03/2007, razão pela qual ficam fazendo parte integrante deste ajuste.

16.2 - Os casos omissos serão resolvidos com base nas disposições constantes da Lei nº 8.666/93, dos princípios de Direito Público, e, subsidiariamente, outras leis que se prestem a suprir eventuais lacunas legais.

16.3 – A Secretaria de Tecnologia da Informação do CONTRATANTE será responsável pelo acompanhamento e fiscalização da execução do presente ajuste, devendo proceder ao registro de eventuais ocorrências e adotar as providências necessárias ao seu fiel cumprimento.

16.4 - De conformidade com o disposto no parágrafo único, do artigo 61, da Lei nº 8.666/93, o presente ajuste será publicado no Diário Oficial da União, na forma de extrato.

16.5 - Para dirimir as questões oriundas do presente contrato, fica eleito o foro de Brasília-DF.

E, para firmeza e como prova de assim haverem ajustado, os representantes das partes assinam o presente CONTRATO em 02 (duas) vias.

Brasília-DF, 30 de maio de 2007.

MIGUEL AUGUSTO FONSECA DE CAMPOS
Diretor-Geral
Superior Tribunal de Justiça

HÉLIO SANTOS OLIVEIRA
Diretor Comercial
Politec Tecnologia da Informação S.A.

WOLNEY MENDES MARTINS
Diretor Regional de Brasília
Politec Tecnologia da Informação S.A.