O Método COSMIC para Medição de Tamanho Funcional

Versão 3.0.1

Manual de Medição

(Guia de Implementação COSMIC para a ISO/IEC 19761: 2003)

Maio de 2009

Agradecimentos

Autores da Equipe Principal COSMIC Versão 2.0[1] (em ordem alfabética)

Alain Abran, École de technologie supérieure – Université du Québec,
Jean-Marc Desharnais, Software Engineering Laboratory in Applied Metrics - SELAM,
Serge Oligny, Bell Canada,
Denis St-Pierre, DSA Consulting Inc.,
Charles Symons, Software Measurement Services Ltd.

Revisores da Versão 2.0 1998/1999 (em ordem alfabética)		
Moritsugu Araki, JECS Systems Research, Japan	Thomas Fetcke, Germany	Patrice Nolin, Hydro Québec, Canada
Fred Bootsma, Nortel, Canada	Eric Foltin, University of Magdeburg, Germany	Marie O'Neill, Software Management Methods, Ireland
Denis Bourdeau, Bell Canada, Canada	Anna Franco, CRSSM, Canada	Jolijn Onvlee, The Netherlands *
Pierre Bourque, , ÉCole de Technologie supérieure, Canada	Paul Goodman, Software Measurement Services,United Kingdom	Laura Primera, UQAM, Canada
Gunter Guerhen, Bürhen & Partner, Germany	Nihal Kececi, University of Maryland, United States	Paul Radford, Charismatek, Australia
Sylvain Clermont, Hydro Québec, Canada	Robyn Lawrie, Australia	Eberhard Rudolph, Germany
David Déry, CGI, Canada	Ghislain Lévesque, UQAM, Canada	Grant Rule, Software Measurement Services, United Kingdom*
Gilles Desoblins, France	Roberto Meli, Data Processing Organization, Italy	Richard Stutzke, Science Applications Int'l Corporation, United States
Martin D'Souza, Total Metrics, Australia	Pam Morris, Total Metrics, Australia*	Ilionar Sylva, UQAM, Canada
Reiner Dumke, University of Magdeburg, Germany	Risto Nevalainen, Software Technology Transfer Finland, Finland *	Vinh T. Ho, UQAM, Vietnam
Peter Fagg, United Kingdom	Jin Ng, Hmaster, Australia	

* Membros fundadores da Equipe Principal COSMIC, juntamente com os autores do COSMIC-FFP

[1] A Versão 2.0 foi a primeira versão do método COSMIC-FFP disponível para o público, como o método foi inicialmente conhecido

Revisores da Versão 3.0 2006/07 (em ordem alfabética)		
Alain Abran, École de Technologie Supérieure, Université du Québec, Canada	Jean-Marc Desharnais, Software Engineering Lab in Applied Metrics – SELAM, Canada	Arlan Lesterhuis*, Sogeti, The Netherlands
Bernard Londeix, Telmaco, United Kingdom	Roberto Meli, Data Processing Organization, Italy	Pam Morris, Total Metrics, Australia
Serge Oligny, Bell Canada	Marie O'Neill, Software Management Methods, Ireland	Tony Rollo, Software Measurement Services, United Kingdom
Grant Rule, Software Measurement Services, United Kingdom	Luca Santillo, Agile Metrics, Italy	Charles Symons*, United Kingdom
Hannu Toivonen, Nokia Siemens Networks, Finland	Frank Vogelezang, Sogeti, The Netherlands	

* Editores das versões 3.0 e 3.0.1 do método COSMIC

Tradução Brasileira

- Mauricio Aguiar, TI Métricas, Rio de Janeiro, Brasil (01-Mar-2012). Comentários sobre a tradução brasileira devem ser enviados ao tradutor via mauricio@metricas.com.br.

Uma versão de domínio público do Manual de Medição COSMIC e outros relatórios técnicos, inclusive traduções para outros idiomas, pode ser encontrada na Web em www.cosmicon.com.

Controle de Versão

A tabela seguinte fornece a história das versões deste documento

DATA	REVISOR(ES)	Mudanças / Acréscimos
99-03-31	Serge Oligny	Primeiro documento, emitido para comentários por parte dos revisores
99-07-31	Ver Agradecimentos	Revisado, incluindo comentários dos revisores
99-10-06	Ver Agradecimentos	Revisado, incluindo comentários do workshop do IWSM '99[2]
99-10-29	Equipe Principal COSMIC	Revisado, comentários finais antes da publicação da versão "teste de campo" 2.0.
01-05-01	Equipe Principal COSMIC	Revisado para conformidade com a ISO/IEC 14143-1: 1998 + esclarecimentos sobre as regras de medição para a versão 2.1.
03-01-31	Comitê de Práticas de Medição COSMIC	Revisado para conformidade com ISO/IEC FDIS 19761: 2002 + esclarecimentos adicionais sobre as regras de medição para a versão 2.2
07-09-01	Comitê de Práticas de Medição COSMIC	Revisado para esclarecimentos adicionais e acréscimos às regras de medição para a versão 3.0, particularmente na área da fase Estratégia de Medição. O nome do método foi alterado de 'método COSMIC-FFP'para 'método COSMIC'. Na mudança de V2.2 para V3.0, algumas partes do 'Manual de Medição' foram transferidas para outros documentos – ver as Notas Introdutórias abaixo e o Apêndice D
09-05-01	Comitê de Práticas de Medição COSMIC	A Versão 3.0 foi revisada e tornou-se a v3.0.1, a fim de incluir pequenas melhorias editoriais e esclarecimentos, assim como distinguir mais claramente os exemplos. Esta versão também incorpora as mudanças propostas nos Boletins de Atualização do Método 3, 4 e 5. Ver o Apêndice D para detalhes dessas mudanças

[2] Anais do International Workshop on Software Measurement IWSM '99, Lac Supérieur, Québec, Canada, 8-10 de setembro de 1999. Ver http://www.cosmicon.com para detalhes.

Notas Introdutórias

O propósito do método COSMIC é prover um método padrão para a medição do tamanho funcional do software correspondente aos domínios funcionais normalmente denominados software para "aplicações de negócio" (ou 'MIS') e software 'real-time'.

O método COSMIC foi aceito pelo ISO/IEC JTC1 SC7 em dezembro de 2002 como o Padrão Internacional ISO/IEC 19761 'Engenharia de Software – COSMIC-FFP – Um método para a medição funcional de tamanho' (daqui em diante referenciado como 'ISO/IEC 19761').

Para maior clareza, a ISO/IEC 19761 contém as definições normativas e regras fundamentais do método. O propósito do Manual de Medição não é apenas prover tais regras e definições, mas também prover explicações adicionais e muito mais exemplos a fim de ajudar os medidores a entenderem completamente e aplicarem o método. Contudo, conforme mais experiência foi sendo adquirida com o método, considerou-se útil acrescentar mais regras e exemplos e até refinar as definições de alguns conceitos subjacentes. O Common Software Measurement International Consortium (COSMIC) planeja que tais acréscimos e refinamentos serão submetidos à ISO para inclusão na ISO/IEC 19761 na época de sua revisão em 2007/08.

Este 'Manual de Medição' é um dos quatro documentos COSMIC que definem a versão 3.0 do método. Os outros três são:

- 'Visão Geral da Documentação e Glossário de Termos' (O Glossário define todos os termos comuns a todos os documentos COSMIC. Este documento também descreve outros documentos de suporte disponíveis, tais como estudos de caso e guias para domínios específicos.)
- 'Visão Geral do Método'
- 'Tópicos Avançados e Relacionados' (Este documento tratará com maior detalhe a tarefa de garantir a compatibilidade entre as medições de tamanho e incluirá capítulos sobre medições aproximadas e no início do projeto, assim como a convertibilidade de medições que apareceram anteriormente na versão 2.2 do Manual de Medição.)

Aos leitores iniciantes na medição funcional de software ('FSM'), ou que estejam familiarizados com outro método FSM, é fortemente recomendada a leitura do documento 'Visão Geral do Método' antes de ler este Manual de Medição.

Principais mudanças na versão 3.0 do método COSMIC

A mudança de designação na versão do método COSMIC de 2.2 para 3.0 indica que esta versão representa um avanço significativo sobre a versão anterior. As principais mudanças na produção desta versão 3.0 do método COSMIC a partir da versão anterior definida no Manual de Medição versão 2.2 são apresentadas a seguir.

- Para tornar a documentação do método COSMIC mais amigável, a versão 3.0 do método é agora definida em quatro documentos distintos, conforme relacionado acima.
- As propostas dos dois Boletins de Atualização do Método publicados desde a última versão 2.2 do Manual de Medição foram incorporadas. Tais são a MUB 1 'Melhorias Propostas na Definição e Características de uma "Camada" de software' e MUB 2 'Melhorias Propostas na Definição de um "objeto de interesse"'. (A versão 3.0.1 incorpora mais três MUB's – ver o Apêndice D.)
- Uma fase 'Estratégia de Medição' foi definida separadamente como a primeira do processo de medição. A fase de estratégia também foi aperfeiçoada com orientações sobre a consideração do 'nível de granularidade' dos Requisitos Funcionais do Usuário do software a ser medido, para ajudar a garantir que as medições de diferentes pedaços de software sejam comparáveis.
- A experiência tem demonstrado que os conceitos correspondentes aos pontos de vista, na medição, do 'Usuário Final' e do 'Desenvolvedor', introduzidos na versão 2.2 do Manual de

Medição podem ser substituído pelo conceito mais simples de 'usuário funcional'. Este último pode ser definido, de uma maneira não exata, como alguém que envia ou que deve receber dados, segundo os Requisitos Funcionais do Usuário do software a ser medido. Todas as medições de tamanho de um pedaço de software serão então realizadas sobre a funcionalidade provida aos usuários funcionais do mesmo, conforme identificado nos respectivos RFU.

- O nome da unidade de medida do método foi alterado de 'Unidade de tamanho funcional COSMIC' (abreviada como Utfc) para 'Ponto de Função COSMIC' (abreviado como PFC). Esta mudança foi efetuada para facilitar a leitura e pronúncia, trazendo também maior conformidade com os outros métodos baseados em 'Pontos de Função'. Como uma simplificação adicional, o nome do método foi alterado de 'COSMIC-FFP' para 'COSMIC'.
- O glossário foi atualizado e melhorado a fim de facilitar a leitura, tendo sido transferido para o novo documento 'Visão Geral da Documentação e Glossário de Termos'.
- Algum material foi removido, especialmente sobre conceitos de análise de dados. Este material agora faz parte do 'Guia para Dimensionamento de Aplicações de Negócio utilizando COSMIC', pois é específico daquele domínio e não de todo o método COSMIC.
- Muitas melhorias editoriais e acréscimos foram efetuados para aumentar a consistência da terminologia e melhorar o entendimento. Dentre esses, foi feita uma distinção mais consistente entre 'princípios' e 'regras' do método, por meio da adoção de uma convenção do mundo da contabilidade, de que 'as regras existem para ajudar a aplicar princípios e definições'. Tanto os princípios quanto as regras devem ser considerados obrigatórios. Dessa forma, a maioria dos exemplos foi transferida das declarações de princípios e regras para o corpo do texto.

Todas essas mudanças estão resumidas no Apêndice D.

Os leitores familiarizados com a versão 2.2 do Manual de Medição encontrarão a maioria das mudanças da versão 3.0 na fase recentemente destacada, a "Estratégia de Medição".

Consequências das principais mudanças sobre as medições de tamanho existentes

Os princípios originais do método COSMIC permaneceram sem mudanças desde a sua publicação inicial no primeiro rascunho do Manual de Medição em 1999, apesar dos refinamentos e acréscimos necessários para produzir o Padrão Internacional e da produção das versões 2.1, 2.2, 3.0 e desta última versão 3.0.1 do Manual de Medição.

Os tamanhos funcionais medidos de acordo com os princípios e regras das versões 3.0 e 3.0.1 do Manual de Medição podem diferir dos tamanhos medidos com base em versões anteriores, apenas porque as novas regras pretendem ser mais precisas. Assim sendo, os medidores possuem menos espaço para interpretações pessoais do que havia nas versões anteriores. As mudanças na área da Estratégia de Medição e a mudança no nome da unidade de medida resultaram em diferenças triviais ao reportar os resultados, se comparado às versões anteriores.

Explicação para as principais mudanças da versão 2.2 para a versão 3.0 do Manual de Medição

Primeiro precisamos enfatizar que a unidade de medição COSMIC, a 'Utfc' (agora denominada 'PFC'), permanece inalterada desde que foi introduzida na primeira versão pública do Manual de Medição COSMIC-FFP no ano de 1999. Este é o equivalente COSMIC de, por exemplo, uma unidade padrão de comprimento tal como o metro. Entretanto, o tamanho funcional de um pedaço de software pode ser medido de várias maneiras, e às vezes é um problema, para qualquer Método de Medição de Tamanho Funcional (MTF), responder à pergunta: 'qual tamanho deveríamos medir?'

O primeiro problema é sabermos que qualquer pedaço de software provê funcionalidade a vários tipos de 'usuários', onde um usuário é definido na terminologia do padrão ISO/IEC 14143/1 ('Princípios de MTF') basicamente como 'qualquer coisa que interage com o software sendo medido'. Segue-se que o tamanho funcional de um pedaço de software depende de quem ou o quê é(são) definido(s) como o(s) seu(s) usuário(s).

Vamos considerar o exemplo do software aplicativo de um telefone celular. Se aceitarmos a definição de 'usuário' da ISO literalmente, os usuários de um aplicativo de telefone celular poderiam incluir todos os seguintes: um ser humano que pressiona os botões; os dispositivos de hardware (por exemplo: a tela, teclas, etc.) que interagem com o aplicativo; o sistema operacional que dá suporte ao aplicativo; softwares 'pares' distintos que interagem com o aplicativo sendo medido. Todos os quatro tipos de usuário requerem funcionalidades diferentes (daí o tamanho funcional diferir dependendo de quem ou o quê for definido como o usuário). Assim sendo, como poderemos saber, dado um tamanho funcional específico, quem ou o quê são os usuários, isto é, quais funcionalidades foram medidas?

Foi esta questão que inicialmente levou o Comitê de Práticas de Medição COSMIC ('MPC') a introduzir na versão 2.2 do Manual de Medição os conceitos de Ponto de Vista de Medição do 'Usuário Final' e do "Desenvolvedor'. Contudo, a experiência tem mostrado que tais definições, especialmente a de Ponto de Vista de Medição do Desenvolvedor, não eram suficientemente genéricas para auxiliar na definição de todas as necessidades de medição. O MPC concluiu que a abordagem correta e mais genérica é definir o conceito de um 'usuário funcional' e que o tamanho funcional muda dependendo de quem ou o quê seja definido como o usuário funcional. A identificação dos usuários funcionais depende do propósito da medição e os usuários funcionais devem ser normalmente identificáveis nos Requisitos Funcionais do Usuário (RFU) do software a ser medido. A idéia de definir 'Pontos de Vista de Medição' específicos não é mais necessária.

O segundo problema é sabermos que os RFU evoluem conforme o projeto avança e, dependendo de como a medição seja realizada, o tamanho medido pode parecer aumentar. A primeira versão dos RFU para um novo pedaço de software pode ser especificada a um 'nível alto'. Conforme o projeto avança e os requisitos são elaborados em maior detalhe, os RFU são especificados em maior detalhe, ou a um 'nível mais baixo', podendo seu tamanho parecer aumentar. Distinguimos tais níveis de detalhe como 'níveis de granularidade'.

Dessa forma, o problema que deve ser abordado agora é: como podemos estar certos de que duas medições foram feitas no mesmo nível de granularidade? As versões 3.0 e 3.0.1 do Manual de Medição fornecem recomendações sobre este tópico, o que é especialmente importante quando os tamanhos são medidos cedo no ciclo de vida de um projeto, quando os RFU ainda estão evoluindo. O tópico torna-se crítico quando os tamanhos são utilizados para medições de desempenho que devem ser comparadas a partir de fontes diferentes, como acontece em exercícios de benchmarking.

É importante enfatizar que estes novos conceitos tais como 'usuário funcional' e 'nível de granularidade', assim como os processos associados à sua determinação que foram introduzidos na Estratégia de Medição não precisam ser uma exclusividade do método COSMIC. Ao contrário, são aplicáveis a todos os métodos de Medição de Tamanho Funcional (MTF). Como o método COSMIC baseia-se em princípios sólidos de engenharia e é aplicável a uma faixa mais ampla de domínios de software do que os métodos MTF de '1a. geração', o problema de definir 'qual tamanho devemos medir?' foi reconhecido e uma solução foi encontrada.

A maioria dos medidores utilizando COSMIC nos casos em que o propósito da medição é relacionado ao desempenho (por exemplo: estimativas, benchmarking, etc.) não precisarão gastar tempo na identificação de usuários funcionais ou na determinação do nível de granularidade a ser utilizado na medição, pois os mesmos serão bastante óbvios. No caso de medições onde as escolhas não sejam óbvias, existe material recente sobre o assunto na 'Fase da Estratégia de Medição' do Manual de Medição e no capítulo sobre como garantir a comparabilidade das medições de tamanho no documento 'Tópicos Avançados e Relacionados', onde os fatores a serem considerados nesses casos são discutidos em maior profundidade.

Comitê de Práticas de Medição COSMIC

Maio de 2009

Conteúdo

1

INTRODUÇÃO

1.1 Aplicabilidade do Método COSMIC

1.1.1 Domínios aplicáveis

O método COSMIC para medição de tamanho funcional foi projetado para ser aplicável à funcionalidade de software dos seguintes domínios:

- Software aplicativo de negócio normalmente necessário para apoiar a administração de negócios tais como: bancos, seguros, contabilidade, pessoal, compras, distribuição ou manufatura. Tal tipo de software é muitas vezes caracterizado como 'rico em dados', pois é amplamente dominado pela necessidade de gerenciar uma grande quantidade de dados sobre eventos do mundo real.
- Software de tempo-real, com a finalidade de acompanhar ou controlar eventos acontecendo no mundo real. Exemplos seriam: software para ligações telefônicas e troca de mensagens, software embarcado em dispositivos para controlar máquinas tais como eletrodomésticos, elevadores, motores de automóveis e aeronaves, controle de processos e aquisição automática de dados, assim como software contido no sistema operacional de computadores.
- Híbridos dos tipos de software acima, tais como sistemas em tempo-real para reservas de companhias aéreas e hotéis, por exemplo.

1.1.2 Domínio não-aplicável

O método de medição COSMIC ainda não foi projetado para levar em conta a funcionalidade de software intensivo em matemática, isto é, software caracterizado por algoritmos matemáticos complexos, ou outras regras especializadas e complexas, tais como sistemas especialistas, software para simulação, software para auto-aprendizagem, sistemas para predição do tempo, etc., ou que processe variáveis contínuas, tais como sons de áudio ou imagens de vídeo, por exemplo: software para jogos de computador, instrumentos musicais, etc.

No caso de software com tal funcionalidade é possível, no entanto, definir extensões locais ao método de medição COSMIC. O Manual de Medição explica em quais contextos tais extensões locais deveriam ser usadas e fornece exemplos de uma extensão local. Quando utilizadas, tais extensões devem ser reportadas de acordo com as convenções apresentadas no capítulo sobre reporte das medições deste Manual de Medição.

1.1.3 Limitações nos fatores que contribuem para o tamanho funcional

Além do que já foi dito, o método COSMIC para medição do tamanho funcional não tenta medir todos os aspectos possíveis da funcionalidade que poderiam ser considerados com contribuintes do 'tamanho' do software. Por exemplo, nem a influência da 'complexidade' (seja lá como for definida) sobre o tamanho do software, nem a influência do número de atributos de dados por movimentação de dados são capturados por este método de medição (para saber mais sobre isto, consultar o capítulo Fase de Mapeamento deste manual). Conforme descrito na seção 1.1.2, se desejado, tais aspectos do tamanho funcional podem ser considerados em uma extensão local do método COSMIC para medição.

1.1.4 Limitação na medição de pedaços de software muito pequenos

Todos os métodos para a medição funcional de tamanho são baseados nas suposições de um modelo simplificado da funcionalidade do software, que pretende ser razoável 'na média' para o seu domínio de aplicabilidade. Dessa forma, é necessário cuidado ao medir, comparar ou utilizar (por exemplo, em estimativas) tamanhos de pedaços de software muito pequenos, especialmente de alterações muito pequenas em um pedaço de software, quando as suposições 'médias' podem não ser válidas. No caso do método COSMIC, 'muito pequenas' significa 'poucas movimentações de dados'.

1.2 Requisitos Funcionais do Usuário

O método de medição COSMIC envolve a aplicação de um conjunto de modelos, princípios, regras e processos aos Requisitos Funcionais do Usuário (ou 'RFU') de um dado pedaço de software. O resultado é um número, o 'valor de uma quantidade' (conforme definido pela ISO), representando o tamanho funcional do pedaço de software de acordo com o método COSMIC.

Os tamanhos funcionais medidos pelo método COSMIC são projetados para serem independentes de quaisquer decisões sobre implementação contidas nos artefatos operacionais do software a ser medido. A 'funcionalidade' refere-se ao 'processamento da informação que o software deve realizar para seus usuários'.

Mais especificamente, uma declaração de RFU descreve 'o quê' o software deve fazer para os usuários funcionais que são a fonte e o destino pretendido para os dados, de e para o software. Uma declaração de RFU exclui quaisquer requisitos técnicos ou de qualidade que dizem 'como' o software deve executar. (Ver seção 2.2 para a definição formal de RFU) Somente os RFU são levados em conta na medição do tamanho funcional.

1.2.1 Extraindo os requisitos funcionais do usuário dos artefatos de software, na prática

No mundo real do desenvolvimento de software é raro encontrar artefatos de software nos quais os RFU estejam claramente separados dos outros tipos de requisitos e expressos de uma forma adequada para a medição direta sem qualquer tipo de interpretação. Isto significa que geralmente o medidor terá que extrair os RFU conforme fornecidos, ou subentendidos nos artefatos reais do software, antes de mapeá-los para os conceitos dos 'modelos de software' COSMIC.

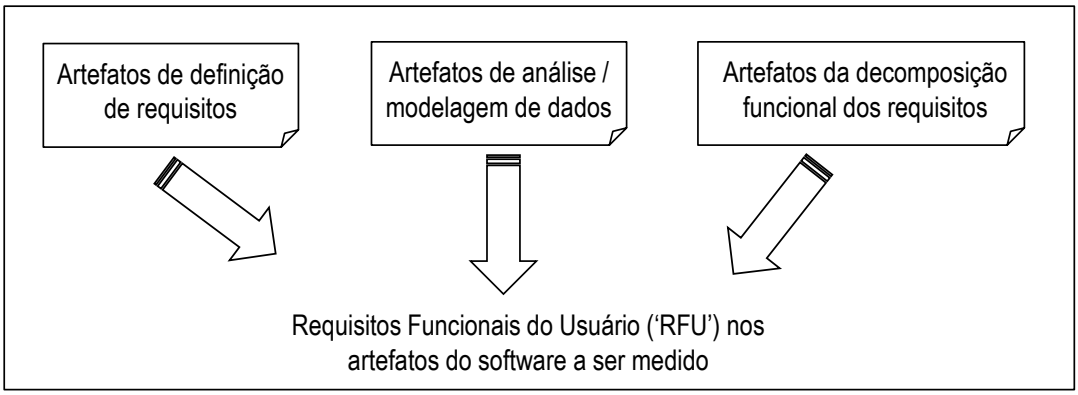


Figura 1.2.1 – Modelo COSMIC dos requisitos funcionais do usuário pré-implementação

Conforme ilustrado na figura 1.2.1, os RFU podem ser derivados dos artefatos de engenharia de software produzidos antes que o software exista. Assim sendo, o tamanho funcional do software pode ser medido antes de sua implementação em um sistema de computador.

É importante notar que os artefatos produzidos antes da implementação do software, por exemplo, durante o levantamento ou análise dos requisitos, podem descrever o software a diferentes 'níveis de granularidade', conforme evoluírem os requisitos – ver a seção 2.4 adiante.

Nota: Os requisitos funcionais do usuário podem ser produzidos antes de serem alocados ao hardware ou software. Como o método COSMIC tem como alvo o dimensionamento dos RFU de um pedaço de software, apenas os RFU alocados ao software são medidos. Entretanto, por princípio, o COSMIC pode ser aplicado aos RFU antes que estejam alocados ao hardware ou software, independentemente da decisão final sobre a alocação. Por exemplo, é direto o dimensionamento da funcionalidade de uma calculadora de bolso utilizando COSMIC, sem qualquer conhecimento de qual hardware ou software (se algum) estará envolvido. Contudo, a afirmação de que o método COSMIC pode ser utilizado para dimensionar RFU alocados a hardware precisa de mais testes na prática, antes de ser considerada totalmente validada sem a necessidade de regras adicionais.

1.2.2 Derivando os requisitos funcionais do usuário a partir de software instalado

Às vezes poderá ser necessário medir algum pedaço de software existente sem que estejam disponíveis artefatos de design ou arquitetura em quantidade suficiente, com os RFU não documentados (por exemplo, no caso de software legado). Nesses casos, ainda será possível derivar os RFU a partir dos artefatos instalados no sistema computacional mesmo após a implementação, conforme ilustrado na figura 1.2.2.

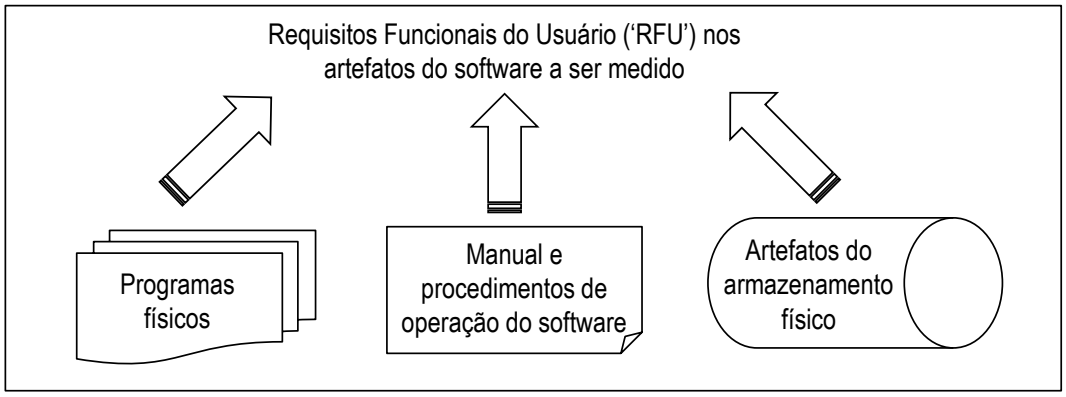


Figura 1.2.2 – Modelo COSMIC dos requisitos funcionais do usuário pós-implementação

1.2.3 Extraindo ou derivando os requisitos funcionais do usuário dos artefatos de software

O processo a ser utilizado e daí o esforço requerido para extrair os RFU dos diferentes tipos de artefatos de engenharia de software, ou derivá-los a partir de software instalado, expressando-os na forma requerida para medição pelo método COSMIC obviamente irão variar; tais processos não podem ser tratados no Manual de Medição. É feita a suposição de que os requisitos funcionais do usuário do software a ser medido existam, ou que possam ser extraídos ou derivados de seus artefatos, à luz do propósito da medição.

O Manual de Medição limita-se então a descrever e definir os conceitos dos modelos de software COSMIC, isto é, os objetivos para a extração ou derivação.[3] Tais conceitos encontram-se em um conjunto de princípios estabelecidos em dois modelos de software COSMIC – o 'Modelo de Contexto de Software' e o 'Modelo Genérico de Software'.

[3] O 'Guia para Dimensionamento de Software Aplicativo de Negócios utilizando COSMIC' fornece orientações para o mapeamento de vários métodos de análise de dados e determinação de requisitos utilizados no domínio da aplicação de negócio para os conceitos COSMIC.

Esses dois modelos são descritos no documento 'Visão Geral do Método'. Os leitores que precisem de um entendimento genérico e de justificativa para os modelos são remetidos àquele documento. Os modelos são copiados abaixo para facilitar e detalhados nos Capítulos 2 e 3 deste Manual de Medição, respectivamente.

1.3 O Modelo de Contexto de Software COSMIC

Um pedaço de software a ser medido com o método COSMIC deve ser cuidadosamente definido (no escopo da medição) e tal definição deve levar em conta, em seu contexto, qualquer outro software e/ou hardware com o qual o mesmo interaja. Este Modelo de Contexto de Software apresenta os princípios e conceitos necessários a esta definição.

Nota – Os termos em negrito na primeira vez que aparecem nas seções 1.3 e 1.4 a seguir possuem significados que podem ser específicos do método COSMIC. Para obter as respectivas definições formais, ver o glossário no documento 'Visão Geral da Documentação e Glossário de Termos'. Todos esses termos são explicados em maior detalhe nos capítulos 2, 3 e 4 a seguir.

PRINCÍPIOS – O Modelo de Contexto de Software COSMIC
a) O Software é delimitado pelo hardware
b) O software é normalmente estruturado em **camadas**
c) Uma camada pode conter um ou mais pedaços de software '**pares**' distintos e qualquer pedaço de software pode ser composto de componentes pares distintos
d) Qualquer pedaço de software a ser medido deverá ser definido por seu **escopo** de medição, o qual deve estar integralmente contido em uma única camada
e) O escopo de um pedaço de software a ser medido depende do **propósito** da medição
f) Os **usuários funcionais** de um pedaço de software devem ser identificados a partir dos requisitos funcionais do usuário do pedaço de software a ser medido, como fontes e/ou destinos pretendidos para os dados
g) Um pedaço de software interage com os seus usuários funcionais por meio de **movimentações de dados** através de uma **fronteira**, e o pedaço de software pode mover dados de e para o **armazenamento persistente** dentro da fronteira
h) Os RFU do software podem ser expressos a diferentes **níveis de granularidade**
i) O nível de granularidade no qual as medições devem ser normalmente efetuadas é o dos **processos funcionais**
j) Se não for possível medir no nível de granularidade dos processos funcionais, nesse caso os RFU do software devem ser medidos através de uma abordagem de aproximação e escalonados para o nível de granularidade dos processos funcionais[4]

Os conceitos do Modelo de Contexto de Software são detalhados no Capítulo 2, 'Estratégia de Medição', deste Manual de Medição.

1.4 O Modelo Genérico de Software

Uma vez interpretados os RFU do software a ser medido em termos do Modelo de Contexto de Software, aplicamos o Modelo Genérico de Software aos RFU, para identificar os componentes da funcionalidade que serão medidos. Este Modelo Genérico de Software assume que os seguintes princípios gerais são verdadeiros para qualquer software que possa ser medido com o método:

[4] Os assuntos 'dimensionamento aproximado' (i.e., estimar um tamanho quando não estiverem disponíveis todos os detalhes necessários à medição de um tamanho exato) e o escalonamento entre diferentes níveis de granularidade são tratados no documento do método COSMIC 'Tópicos Avançados e Relacionados'.

(conforme consta do glossário, qualquer método de medição funcional tem como objetivo identificar 'tipos' e não 'ocorrências' de dados ou funções. No texto abaixo, o termo 'tipo' será omitido ao mencionar os conceitos básicos do COSMIC, a não ser que isso seja essencial para distinguir 'tipos' de ocorrências.)

PRINCÍPIOS – O Modelo Genérico de Software COSMIC
a) O software recebe dados de **entrada** dos seus usuários funcionais, gerando **saídas** e/ou outros resultados para os usuários funcionais
b) Os requisitos dos usuários funcionais de um pedaço de software a ser medido podem ser mapeados para processos funcionais distintos
c) Cada processo funcional consiste de **subprocessos**
d) Um subprocesso pode ser uma movimentação de dados ou uma **manipulação de dados**
e) Cada processo funcional é disparado por um movimento de dados do tipo **Entry** proveniente de um usuário funcional, o qual informa ao processo funcional que o usuário funcional identificou um **evento**
f) Uma movimentação de dados movimenta um único **grupo de dados**
g) Um grupo de dados consiste de um único conjunto de **atributos de dados** que descreve um único objeto de interesse
h) Há quatro tipos de movimentação de dados. Uma **Entry** movimenta um grupo de dados para dentro do software, a partir de um usuário funcional. Uma **Exit** movimenta um grupo de dados para fora do software, em direção a um usuário funcional. Um **Write** movimenta um grupo de dados do software para o armazenamento persistente. Um **Read** movimenta um grupo de dados do armazenamento persistente para o software
i) Um processo funcional deve incluir pelo menos uma movimentação de dados Entry e uma movimentação de dados Write ou Exit, ou seja, deve incluir no mínimo duas movimentações de dados
j) Como uma aproximação para fins de medição, os subprocessos de manipulação de dados não são medidos separadamente; assume-se que a funcionalidade de qualquer manipulação de dados será considerada na movimentação de dados com a qual a mesma esteja associada

Os conceitos do Modelo Genérico de Software são detalhados no Capítulo 3 – 'Fase de Mapeamento' deste Manual de Medição.

Quando os RFU a serem medidos tiverem sido mapeados para o Modelo Genérico de Software, os mesmos poderão ser medidos utilizando-se o processo da Fase de Medição (Capítulo 4). Os resultados da medição devem ser reportados de acordo com as convenções de 'Reportando a Medição' (Capítulo 5).

1.5 O Processo de Medição COSMIC

O processo geral de medição COSMIC consiste de três fases:

- a Estratégia de Medição, na qual o Modelo de Contexto de Software é aplicado ao software a ser medido (Capítulo 2)
- a Fase de Mapeamento, na qual o Modelo Genérico de Software é aplicado ao software a ser medido (Capítulo 3)
- a Fase de Medição, na qual as medições de tamanho propriamente ditas são obtidas (Capítulo 4)

O resultado da aplicação do processo de medição a um pedaço de software é uma medida de tamanho funcional dos Requisitos Funcionais do Usuário do pedaço de software expressa em 'Pontos de Função COSMIC' (ou 'PFC').

A relação entre essas três fases do método COSMIC é apresentada na Fig. 1.5.

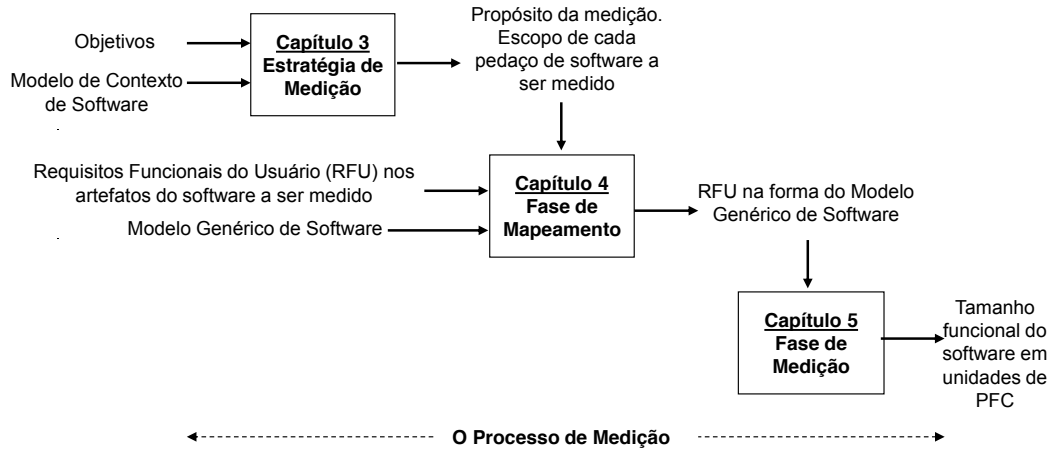


Figura 1.5 – Estrutura do método COSMIC

2

A FASE ESTRATÉGIA DE MEDIÇÃO

Este capítulo aborda os quatro parâmetros-chave da medição de tamanho funcional de software que devem ser considerados antes de iniciar a medição, a saber: o propósito e o escopo da medição, a identificação dos usuários funcionais e o nível de granularidade que deve ser medido. A determinação desses parâmetros ajuda a responder perguntas tais como 'qual tamanho deve ser medido?' ou, para uma medição existente, 'como devemos interpretar esta medição?'

É importante notar que esses quatro parâmetros e os conceitos relacionados não são específicos do método de medição de tamanho funcional (MTF) COSMIC, mas devem ser comuns a todos os métodos MTF. Outros métodos MTF podem não distinguir os diferentes tipos de usuários funcionais, não discutir diferentes níveis de granularidade, etc. É somente a aplicabilidade mais ampla e a flexibilidade do método COSMIC que requerem a consideração desses parâmetros de uma maneira mais cuidadosa do que aquela utilizada por outros métodos MTF.

É muito importante incluir os dados originados nesta fase Estratégia de Medição (conforme seção 5.2) ao registrar o resultado de qualquer medição. A falta da definição e registro consistente destes parâmetros levará a medições que não poderão ser interpretadas de forma confiável, comparadas, ou utilizadas como entrada para processos tais como a estimativa do esforço do projeto.

As quatro seções deste capítulo fornecem as definições formais, princípios e regras, assim como alguns exemplos de cada um dos parâmetros-chave, para ajudar o medidor no processo de determinação de uma Estratégia de Medição, conforme mostrado na figura 2.0 abaixo.

Cada seção fornece uma explicação contextual sobre a razão da importância do parâmetro, utilizando analogias para mostrar por que o parâmetro é comum em outros campos da medição e, dessa forma, mostrar que também deve ser considerado no campo de medição de tamanho funcional de software.

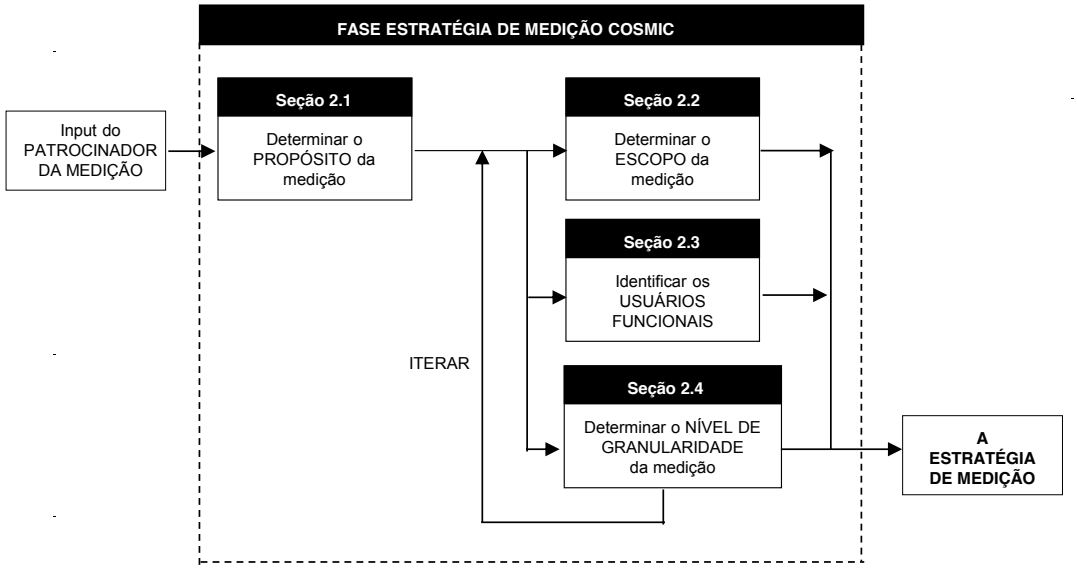


Figura 2.0 - O processo de determinação de uma Estratégia de Medição

2.1 Definindo o propósito da medição

O termo propósito é utilizado com o seu significado normal no idioma.

DEFINIÇÃO – Propósito de uma medição
Uma afirmação que define por que uma medição é requerida e para quê o resultado será utilizado.

2.1.1 O propósito da medição – uma analogia

Há muitas razões para medir o tamanho funcional do software, assim como há muitas razões para medir, por exemplo, as áreas das superfícies de uma casa. Quando o propósito é estimar o custo de um novo desenvolvimento de software, pode ser necessário medir o tamanho funcional do software antes do seu desenvolvimento, assim como pode ser necessário medir as áreas das superfícies de uma casa antes de sua construção. Em um contexto diferente, por exemplo, na comparação de custos reais com estimados, pode ser necessário medir o tamanho funcional do software no momento de sua entrada em produção, da mesma forma que poderia ser útil medir as áreas das superfícies de uma casa depois de construída, para verificar se as dimensões reais estão de acordo com os respectivos planos. O motivo pelo qual uma medida é realizada tem um impacto, embora muitas vezes sutil, *naquilo* que é medido, *sem afetar* a unidade de medida ou os princípios de medição.

No exemplo da casa acima, a medição das áreas de suas superfícies antes da construção baseia-se, obviamente, nos planos de construção. As dimensões requeridas (comprimento e largura) são extraídas dos planos utilizando-se convenções de escalonamento apropriadas, assim como as áreas das superfícies são calculadas de acordo com convenções bem estabelecidas.

Analogamente, a medição do tamanho funcional do software antes do seu desenvolvimento baseia-se nos requisitos funcionais do usuário derivados de seus 'planos', isto é, nos artefatos de software produzidos antes do desenvolvimento. Os requisitos funcionais do usuário são derivados desses artefatos, utilizando-se convenções apropriadas, assim como é identificado o número de dimensões requerido (o número de movimentações de dados), de modo que o tamanho possa ser calculado.

Para prosseguir na analogia da casa, a medição das áreas de suas superfícies após a construção implica em um processo algo diferente. Neste ponto, as dimensões requeridas (comprimento e largura) são extraídas da própria construção, utilizando-se uma ferramenta diferente (fita métrica). No entanto, embora o objeto medido seja diferente (a casa ao invés dos planos), as dimensões, a unidade de medida (incluindo quaisquer convenções de escalonamento) e os princípios de medição permanecem constantes.

Da mesma forma, a medição do tamanho funcional do software depois de colocado em produção implica em um processo de medição um pouco diferente, no qual as dimensões requeridas são extraídas dos vários artefatos do próprio software. Embora a natureza desses artefatos seja diferente, as dimensões, a unidade de medida e os princípios de medição permanecem os mesmos.

Cabe ao medidor, com base no propósito da medição, determinar se o objeto a ser medido é a casa conforme verbalmente descrita por seu proprietário, conforme descrita nos planos, ou a casa construída, e selecionar os artefatos mais apropriados à medição. É claro que os três tamanhos podem ser diferentes. O mesmo raciocínio aplica-se à medição de software.

EXEMPLOS: A seguir, alguns propósitos típicos da medição

- Medir o tamanho dos RFU ao longo de sua evolução, como entrada para um processo de estimativa do esforço de desenvolvimento
- Medir o tamanho das mudanças nos RFU depois que os mesmos foram inicialmente acordados, a fim de gerenciar a variação não controlada no escopo do projeto ('scope creep')
- Medir o tamanho dos RFU do software entregue, como entrada para a medição do desempenho do desenvolvedor

- Medir o tamanho dos RFU do software total entregue, bem como o tamanho dos RFU do software desenvolvido, a fim de obter uma medida de reuso funcional
- Medir o tamanho dos RFU do software existente, como entrada para a medição do desempenho daqueles responsáveis por manter e suportar o software
- Medir o tamanho de algumas mudanças em um sistema de software existente (ou nos respectivos RFU), como uma medida do tamanho do resultado do trabalho de uma equipe de projeto de melhoria
- Medir o tamanho da funcionalidade do software existente fornecida aos usuários funcionais humanos

2.1.2 A importância do propósito

O propósito ajuda o medidor a determinar:

- o escopo a ser medido e, dessa forma, os artefatos que serão necessários para a medição
- os usuários funcionais (conforme será mostrado na seção 2.3, o tamanho funcional muda dependendo de quem ou o quê é definido como o usuário funcional)
- o ponto no tempo, no ciclo de vida do projeto, no qual a medição acontecerá
- a exatidão requerida para a medição, e daí se o método de medição COSMIC deverá ser usado, ou uma versão aproximada e localmente derivada do método (por exemplo, no início do ciclo de vida de um projeto, antes que os RFU estejam completamente detalhados). Estes dois últimos pontos determinarão o nível de granularidade no qual os RFU serão medidos.

2.2 Definindo o escopo da medição

DEFINIÇÃO – Escopo de uma medição

O conjunto de Requisitos Funcionais do Usuário a serem incluídos em um exercício específico de medição funcional de tamanho.

NOTA: Uma distinção deve ser feita entre o 'escopo total', isto é, todo o software que deveria ser medido de acordo com o propósito, e o 'escopo' de qualquer pedaço individual de software dentro do escopo total, cujo tamanho deva ser medido separadamente. Neste Manual de Medição, o termo 'escopo' (ou a expressão 'escopo da medição') relaciona-se a um pedaço individual de software cujo tamanho deve ser medido separadamente.

Os Requisitos Funcionais do Usuário são definidos pela ISO como segue:

DEFINIÇÃO – Requisitos Funcionais do Usuário (RFU)

Um subconjunto dos Requisitos do Usuário. Requisitos que descrevem o que o software deve fazer, em termos de tarefas e serviços.

NOTA: Os Requisitos Funcionais do Usuário dizem respeito, mas não estão limitados a:

- transferência de dados (p.ex., Entrar com dados do cliente; Enviar sinal de controle)
- transformação de dados (p.ex., Calcular juros bancários; Derivar temperatura média)
- armazenamento de dados (p.ex., Armazenar pedido do cliente; Registrar temperatura ambiente ao longo do tempo)
- recuperação de dados (p.ex., Listar empregados atuais; Recuperar posição mais recente da aeronave)

Exemplos de Requisitos do Usuário que não são Requisitos Funcionais do Usuário incluem, mas não estão limitados a:

- restrições de qualidade (p.ex., usabilidade, confiabilidade, eficiência e portabilidade)
- restrições organizacionais (p.ex., locais de operação, hardware-alvo e conformidade com padrões)
- restrições ambientais (p.ex., interoperabilidade, segurança e privacidade)
- restrições de implementação (p.ex., linguagem de desenvolvimento, prazo de entrega)

REGRAS – Escopo
a) O escopo de uma Medição de Tamanho Funcional (MTF) deve ser derivado do propósito da medição.
b) O escopo de uma medição qualquer não deve contemplar mais do que uma camada do software a ser medido

2.2.1 Derivando o escopo a partir do propósito de uma medição

É importante definir o escopo de uma medição antes de iniciar um exercício de medição específico.

Prosseguindo com a analogia de construção de uma casa, se o propósito for a estimativa de custos pode ser necessário medir o tamanho de diversas partes da casa separadamente, por exemplo, fundações, paredes e telhado, já que as mesmas utilizam diferentes métodos de construção. O mesmo é verdade para a estimativa dos custos de desenvolvimento de software.

Se um sistema de software a ser desenvolvido consiste de pedaços que residirão em diferentes camadas da arquitetura do sistema, então o tamanho do software em cada camada deverá ser medido separadamente, i.e., cada pedaço possuirá um escopo separado definido para fins de medição do tamanho. Isto vem do princípio (d) do Modelo de Contexto de Software. (para mais sobre camadas, ver seção 2.2.4 abaixo.)

Similarmente, se o software tiver que ser desenvolvido como um conjunto de componentes pares dentro de uma única camada, cada um utilizando tecnologias diferentes, então será necessário definir um escopo de medição separado para cada componente antes de medir seus tamanhos.

EXEMPLO 1: Se cada componente do software utilizar uma tecnologia diferente e as medições forem utilizadas para estimar o esforço de desenvolvimento, então um escopo de medição separado deverá ser definido para cada componente, pois cada medição de tamanho será associada a uma produtividade de desenvolvimento diferente. (Para saber mais sobre componentes pares, ver seção 2.2.5 adiante)

O propósito também ajuda a determinar o software a ser incluído/excluído do escopo da medição.

EXEMPLO 2: Se o propósito for medir o tamanho funcional de todo o software entregue por uma equipe específica de um projeto, primeiro será necessário definir os requisitos funcionais do usuário de todos os pedaços e componentes entregues pela equipe. Estes poderão incluir os RFU de um pedaço de software que foi utilizado apenas uma vez para converter dados do software que está sendo substituído.

EXEMPLO 3: Se o propósito for medir o tamanho do novo software que está disponível para utilização operacional, este será menor do que o do Exemplo 2, já que os RFU do software utilizado para conversão não serão incluídos no escopo do tamanho medido.

Em resumo, o propósito da medição sempre precisa ser usado para determinar (1) qual software deve ser incluído/ excluído do escopo total e (b) a maneira como o software incluído pode precisar ser dividido em pedaços distintos, cada qual com seu escopo, para que possam ser medidos separadamente.

2.2.2 Tipos genéricos de escopo

EXEMPLOS

- Uma carteira corporativa
- Uma declaração de requisitos contratualmente acordada
- O produto do trabalho *entregue* por uma equipe de projeto (i.e., incluindo o que foi obtido através da exploração dos parâmetros de software existente, pacotes comprados e código reutilizável, qualquer software utilizado para conversão de dados e em seguida descartado, assim como utilitários e software de teste desenvolvido especificamente para este projeto)
- O produto do trabalho *desenvolvido* por uma equipe de projeto (i.e., incluindo qualquer software desenvolvido pela equipe e utilizado para conversão de dados e em seguida descartado, assim como quaisquer utilitários e software de teste desenvolvido especificamente para este projeto, porém *excluindo* toda a funcionalidade obtida pela alteração de parâmetros, exploração de código reutilizável, ou pacotes comprados
- Todo o software em uma camada
- Um pacote de software
- Um aplicativo
- Um componente principal ('par') de um aplicativo
- Uma classe de objetos reutilizável
- Todas as mudanças requeridas para uma nova 'release' de um pedaço de software existente

Na prática, uma declaração de escopo precisa ser explícita ao invés de genérica, i.e., o produto do trabalho desenvolvido pela equipe de projeto 'A', ou aplicativo 'B', ou a carteira da corporação 'C'. A declaração de escopo pode, para fins de clareza, precisar dizer o que foi excluído.

2.2.3 Níveis de decomposição

Notar que alguns dos 'tipos de escopo genéricos' listados acima correspondem a diferentes 'níveis de decomposição' do software, definidos como segue:

DEFINIÇÃO – Nível de decomposição
Qualquer nível resultante da divisão de um pedaço de software em componentes (denominados 'Nível 1', por exemplo), seguida pela divisão dos componentes em subcomponentes ('Nível 2'), então pela divisão dos subcomponentes em subsubcomponentes ('Nível 3'), etc. NOTA 1: Não deve ser confundido com 'nível de granularidade'. NOTA 2: As medições de tamanho dos componentes de um pedaço de software podem ser diretamente comparáveis apenas no caso de componentes pares, isto é, componentes no mesmo nível de decomposição.

EXEMPLO: Diferentes níveis de decomposição, correspondentes a diferentes 'tipos genéricos de escopo' ocorrem quando uma 'carteira de aplicativos' consiste de diversos 'aplicativos', cada um dos quais pode ser composto de 'componentes principais (pares)', cada um dos quais pode por sua vez consistir de 'classes de objetos reutilizáveis'.

A determinação de um escopo de medição pode assim ser mais do que apenas uma questão de decidir quais funcionalidades devem ser incluídas na medição. A decisão também pode envolver a consideração do nível de decomposição de software no qual as medições serão feitas. Esta é uma importante decisão da Estratégia de Medição, dependente do propósito da medição, já que medições em diferentes níveis de decomposição não podem ser facilmente comparadas. Isto ocorre porque, conforme será visto na seção 4.3.1 regra (g), o tamanho de um pedaço de software não pode ser obtido pela simples soma dos tamanhos de seus componentes.

2.2.4 Camadas

Como o escopo de um pedaço de software a ser medido deve ser limitado a uma única camada de software, o processo de definição do escopo pode exigir que o medidor decida primeiro quais são as camadas da arquitetura de software. Nesta seção vamos definir e discutir 'camadas' de software, da forma que esse termo é utilizado no método COSMIC.

As razões para essas definições e regras são apresentadas a seguir:

- O medidor pode estar diante da medição de algum software em um ambiente de software 'legado', o qual evoluiu durante muitos anos sem nunca ter sido projetado em conformidade com uma arquitetura subjacente (trata-se da chamada 'arquitetura espaguete'). O medidor precisará então de orientações sobre como distinguir as camadas de acordo com a terminologia COSMIC
- As expressões 'camada', 'arquitetura em camadas' e 'componente par' não são utilizadas consistentemente na indústria de software. Se o medidor precisar medir algum software descrito como construído segundo uma 'arquitetura em camadas', é aconselhável verificar se as 'camadas' nessa arquitetura estão definidas de uma maneira compatível com o método COSMIC. Para isso, o medidor deverá estabelecer a equivalência entre os objetos arquiteturais específicos do paradigma da 'arquitetura em camadas' e o conceito de camadas definido neste manual

As camadas podem ser identificadas de acordo com as seguintes definições e princípios:

DEFINIÇÃO – Camada

Uma camada é uma partição resultante da divisão funcional de um sistema de software que, juntamente com o hardware, forma um sistema computacional completo, onde:

- as camadas são organizadas segundo uma hierarquia
- há apenas uma camada em cada nível da hierarquia
- há uma dependência hierárquica do tipo 'superior/subordinado' entre os serviços funcionais providos pelo software em quaisquer duas camadas da arquitetura de software que troquem dados diretamente
- os softwares em quaisquer duas camadas da arquitetura de software que troquem dados interpretam apenas parte daqueles dados identicamente

A identificação de camadas é um processo iterativo. As camadas exatas serão refinadas conforme o processo de mapeamento progredir. Uma vez identificada, cada camada candidata deve se conformar aos seguintes princípios e regras:

PRINCÍPIOS – Camada

a) O software em uma camada troca dados com o software em outra camada através dos seus respectivos processos funcionais.

b) A 'dependência hierárquica' entre camadas é tal que o software em qualquer camada pode utilizar os serviços funcionais de qualquer software em qualquer camada abaixo dele na hierarquia. Onde há tais relações de uso, denominamos a camada que utiliza o software abaixo como 'superior' e qualquer camada contendo software utilizado como sua 'subordinada'. O software na camada superior apoia-se nos serviços de software dessas camadas subordinadas para rodar adequadamente; estas últimas apoiam-se por sua vez em suas camadas subordinadas para rodar adequadamente, e assim em toda a hierarquia. Por outro lado, o software em uma camada subordinada, juntamente com o software em quaisquer camadas subordinadas da qual ele depende, pode rodar sem precisar dos serviços de nenhuma camada superior na hierarquia.

c) O software em uma camada não utiliza, necessariamente, todos os serviços funcionais providos pelo software em uma camada subordinada.

d) Os dados que são trocados entre pedaços de software em quaisquer duas

camadas são definidos e interpretados diferentemente nos respectivos RFU dos dois pedaços de software, isto é, os dois pedaços de software reconhecem diferentes atributos de dados e/ou subgrupos de dados sendo trocados. Contudo, devem existir também um ou mais atributos de dados ou subgrupos definidos em comum, para permitir que o software na camada recebedora interprete os dados passados pelo software na camada remetente, de acordo com as necessidades do software recebedor.

Regras – Camada
a) Se o software for concebido utilizando-se uma arquitetura de camadas estabelecida de acordo com o modelo COSMIC, então aquela arquitetura deverá ser utilizada para identificar as camadas para fins de medição
b) No domínio de MIS ou software de negócio, a camada 'topo', i.e., a camada que não é subordinada a nenhuma outra camada, é normalmente referenciada como a camada 'da aplicação'. O software (aplicativo) desta camada apoia-se, em última instância, nos serviços providos pelos softwares de todas as outras camadas para poder executar adequadamente. No domínio de software 'real-time', o software da 'camada topo' é comumente referenciado como sendo um 'sistema', por exemplo, 'sistema de controle de processo', 'sistema de controle de vôo', etc.
c) Não assumir que qualquer software que tenha evoluído sem a consideração de um projeto arquitetural ou estruturação possa ser dividido em camadas de acordo com o modelo COSMIC.

Pacotes de software de serviços funcionais tais como sistemas de gerenciamento de banco de dados, sistemas operacionais ou direcionadores de dispositivos devem ser normalmente considerados como localizados em camadas distintas.

Uma vez identificada, cada camada pode ser registrada como um 'componente' distinto na matriz do Modelo Genérico de Software (apêndice A), com o rótulo correspondente.

EXEMPLO 1: A estrutura física típica de uma arquitetura de software em camadas (utilizando o termo 'camada' conforme definido aqui) é fornecida na figura 2.2.4.1:

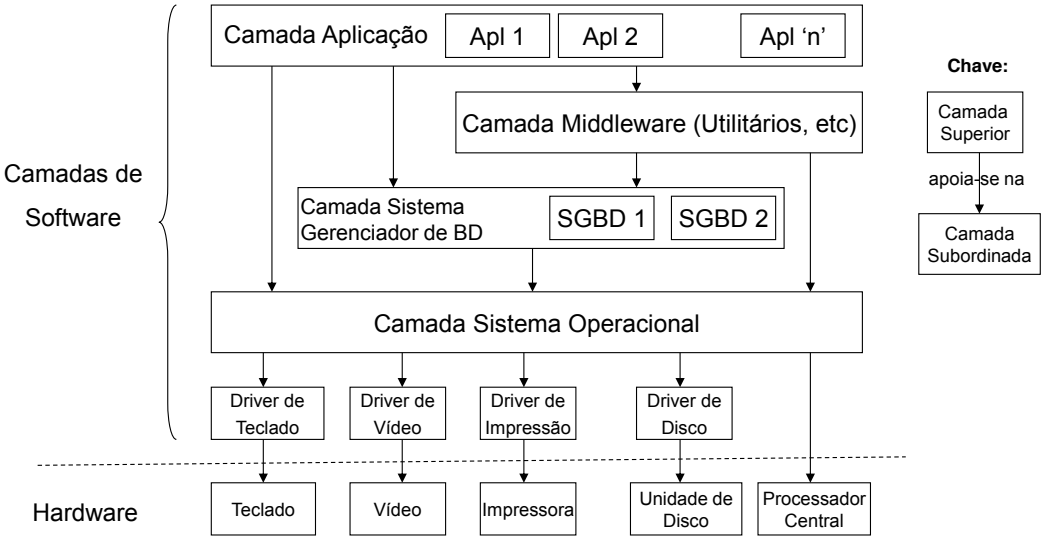


Figura 2.2.4.1 – Arquitetura de software típica em camadas, para um sistema de Negócios/MIS

EXEMPLO 2: A estrutura física típica de uma arquitetura de software em camadas (novamente utilizando o termo 'camada' conforme definido aqui) suportando um pedaço de software 'real-time' embarcado é fornecida na figura 2.2.4.2:

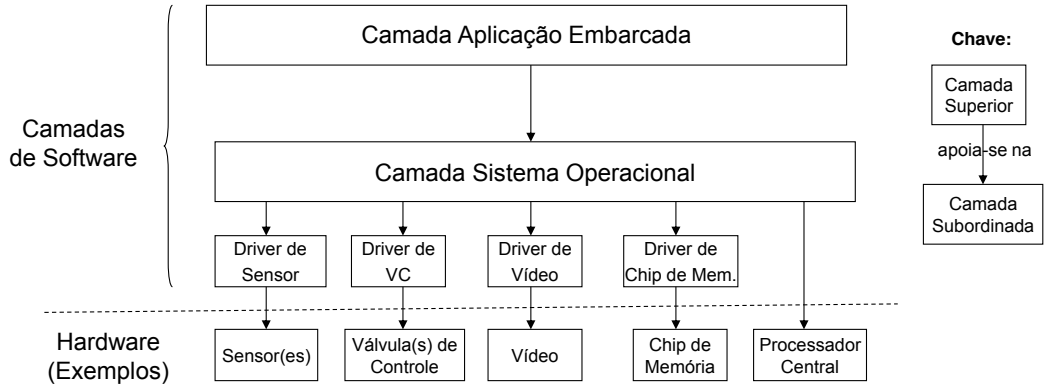


Figura 2.2.4.2 – Arquitetura típica em camadas para um sistema de software 'real-time' embarcado

2.2.5 Componentes pares

Dado o conceito de camadas, 'par' é definido como segue:

DEFINIÇÃO – Par
Dois pedaços de software são pares um do outro se ambos residirem na mesma camada. NOTA. Dois pedaços pares de software não precisam estar no mesmo nível de decomposição.

Componentes pares podem ser identificados de acordo com a seguinte definição e princípios.

DEFINIÇÃO – Componente Par
Um componente de um conjunto de componentes cooperando, todos ao mesmo nível de decomposição, que resulta de se dividir um pedaço de software dentro de uma camada, onde cada componente atende uma porção dos Requisitos Funcionais do Usuário daquele pedaço de software. NOTA: A divisão de um pedaço de software em componentes pares pode ser efetuada em resposta aos requisitos funcionais e não-funcionais do usuário.

Uma vez identificado, cada componente par candidato precisa conformar-se aos seguintes princípios:

PRINCÍPIOS – Componente Par
a) Em um conjunto de componentes pares de um pedaço de software em uma camada não há dependência hierárquica entre os componentes pares da forma que há entre camadas. Os RFU de todos os componentes pares de um pedaço de software em uma camada estão no mesmo 'nível' na hierarquia de camadas. b) Todos os componentes pares de um pedaço de software devem cooperar para que o pedaço de software execute com sucesso. c) Um grupo de dados pode ser diretamente trocado entre dois componentes pares de um pedaço de software, por um processo funcional de um primeiro componente emitindo uma Exit recebida como uma Entry por um processo funcional do segundo componente. Alternativamente, a troca pode acontecer de forma indireta, com um

processo funcional de um primeiro componente tornando um grupo de dados persistente através de um Write, que pode ser recuperado em seguida por um Read de um processo funcional do segundo componente.

Uma vez identificado, cada componente par pode ser registrado como um componente distinto na matriz do Modelo Genérico de Software (apêndice A), com o rótulo correspondente.

EXEMPLO: Quando um aplicativo de negócio é desenvolvido com três componentes principais, especificamente um componente 'interface com o usuário' (ou 'front-end'), um componente de 'regras de negócio' e um componente de 'serviços de dados', os três componentes são componentes pares.

Nota: Dois componentes pares distintos que existem na mesma camada e trocam dados entre si podem fazer isso por meio de uma das sequências alternativas definidas no Princípio (c) para dois componentes pares.

O diagrama seguinte mostra uma situação que ilustra o Exemplo. Todo o software mostrado reside na mesma camada da aplicação. As trocas entre os dois componentes pares da Aplicação X e entre os dois pedaços de software pares (Componente 2 da Aplicação X e Aplicação Y) podem acontecer por meio de uma das sequências alternativas definidas no Princípio (c) para dois componentes pares.

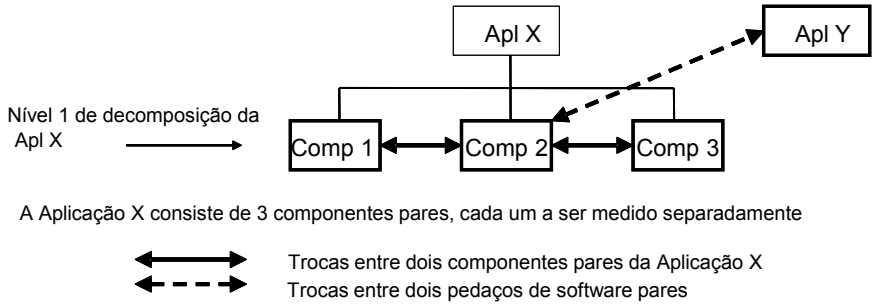


Figura 2.2.5.1 – Relação entre os conceitos de 'par', 'componente' e 'componente par'

2.3 Identificando os usuários funcionais

2.3.1 O tamanho funcional varia com o usuário funcional

Começando com uma analogia, a medição das áreas do chão de um escritório pode ser realizada de acordo com quatro convenções diferentes, como segue. (Nota – o escopo – o escritório específico – é o mesmo para as quatro convenções.)

- O dono do prédio tem que pagar os impostos sobre o escritório. Para o dono do prédio, a área de superfície é a 'metragem quadrada bruta', determinada pelas dimensões externas e dessa forma inclui os saguões públicos, o espaço ocupado pelas paredes, etc.
- O gerente de aquecimento do prédio está interessado na 'metragem quadrada líquida', i.e., as áreas internas, incluindo áreas públicas e o espaço ocupado por elevadores, porém excluindo a espessura das paredes.
- O profissional contratado para a limpeza pelo Inquilino do escritório está interessado na 'metragem líquida-líquida plus', que exclui as áreas públicas, mas inclui corredores utilizados pelo inquilino.
- O gerente de planejamento do escritório está interessado na 'metragem líquida-líquida', i.e., apenas o espaço utilizado para escritórios.

A lição desta analogia é que diferentes tipos de usuários de uma 'coisa' podem 'ver' funcionalidades diferentes e daí medirem tamanhos diferentes para a 'coisa'. No caso do software, diferentes (tipos

de) usuários funcionais podem requerer (via seus RFU) e utilizar funcionalidades diferentes; dessa forma, os tamanhos funcionais irão variar com a escolha dos usuários funcionais.

2.3.2 Usuários funcionais

Um 'usuário' é definido, de fato[5], como 'qualquer coisa que interage com o software sendo medido'. Esta definição é ampla demais para as necessidades do método COSMIC. Para o método COSMIC, a escolha do usuário (ou usuários) é determinada pelos requisitos funcionais do 'usuário' a serem medidos. Este (tipo de) usuário, conhecido como 'usuário funcional', é definido a seguir.

DEFINIÇÃO – Usuário funcional
Um (tipo de) usuário que é uma fonte e/ou um destino pretendido para os dados nos Requisitos Funcionais do Usuário de um pedaço de software.

No método COSMIC é essencial distinguir os usuários funcionais de um pedaço de software que deve ser medido de todos os seus possíveis usuários.

EXEMPLO 1: Consideremos uma aplicação de negócio; seus usuários funcionais normalmente incluiriam seres humanos e outras aplicações pares com os quais a aplicação fizesse interface. Para uma aplicação 'real-time', os usuários funcionais normalmente seriam dispositivos de hardware e outros softwares pares de interface. Os requisitos funcionais do usuário (RFU) de tal software são normalmente expressos de forma que os usuários funcionais sejam as fontes de dados e/ou os destinos pretendidos para os dados, de e para o software, respectivamente.

Contudo, o conjunto total de 'usuários', i.e., incluindo 'qualquer coisa' que interaja com o software, deve incluir o sistema operacional. Porém nunca os RFU de nenhuma aplicação incluiriam o sistema operacional como usuário. Quaisquer restrições que o sistema operacional possa impor a uma aplicação serão comuns a todas as aplicações, sendo normalmente tratadas pelo compilador ou interpretador e invisíveis aos verdadeiros usuários funcionais da aplicação. Na medição de tamanho funcional, um sistema operacional nunca seria considerado um usuário funcional de uma aplicação[6].

Mas nem sempre os usuários funcionais são óbvios.

EXEMPLO 2: Consideremos o software aplicativo de um telefone celular (inicialmente mencionado na Introdução). Embora tenhamos eliminado o sistema operacional do celular como um possível usuário funcional do aplicativo, os 'usuários' ainda poderiam ser (a) seres humanos que pressionam as teclas, ou (b) dispositivos de hardware (p.ex., a tela, teclas, etc.) e aplicações pares que interagem diretamente com o aplicativo do telefone. O usuário humano, por exemplo, verá apenas um subconjunto de toda a funcionalidade que deve ser provida para que o aplicativo do celular funcione. Assim sendo, esses dois tipos de usuários 'verão' funcionalidades diferentes; o tamanho dos RFU para o usuário humano será menor do que os RFU que deverão ser desenvolvidos para fazer funcionar o aplicativo do telefone.[7]

REGRAS – Usuários funcionais
a) Os usuários funcionais de um pedaço de software a ser medido devem ser derivados do propósito da medição
b) Quando o propósito da medição de um pedaço de software estiver relacionado ao esforço para desenvolver ou modificar o pedaço de software, então os usuários funcionais deverão ser aqueles para quem a funcionalidade nova ou modificada deverá ser fornecida.

[5] Ver o glossário para a definição, obtida da ISO/IEC 14143/1:2007

[6] Na realidade, é claro que o oposto é verdadeiro. Os aplicativos são usuários funcionais de um sistema operacional. Os RFU de um sistema operacional são definidos levando-se em conta os aplicativos que o mesmo deverá suportar como seus usuários funcionais.

[7] Toivonen, por exemplo, comparou a funcionalidade de telefones celulares disponíveis para usuários humanos em "Defining measures for memory efficiency of the software in mobile terminals", International Workshop on Software Measurement, Magdeburg, Alemanha, outubro de 2002

Uma vez identificados os usuários funcionais é imediata a identificação da fronteira, já que a mesma situa-se entre o pedaço de software medido e seus usuários funcionais. É ignorado qualquer outro hardware ou software naquele espaço intermediário[8].

DEFINIÇÃO – Fronteira
A fronteira é definida como uma interface conceitual entre o software sendo medido e os seus usuários funcionais. NOTA: A fronteira de um pedaço de software é o limite conceitual entre o referido pedaço e o ambiente no qual o mesmo opera, conforme percebido externamente segundo a perspectiva de seus usuários funcionais. A fronteira permite que o medidor distinga, sem ambiguidade, o que está incluído dentro do software medido daquilo que é parte do ambiente operacional do referido software.

Nota – Esta definição de 'Fronteira' foi obtida da ISO/IEC 14143/1:2007, alterada pela inclusão de 'funcional' para qualificar 'usuário'. A fim de evitar ambiguidade, note-se que a fronteira não deve ser confundida com alguma outra linha que possa ser desenhada ao redor de algum software a ser medido para definir o escopo da medição. A fronteira não é utilizada para definir o escopo de uma medição.

As seguintes regras podem ser úteis para confirmar a situação de uma fronteira candidata:

REGRAS – Fronteira
a) Identificar o(s) usuário(s) funcional(is) que interagem com o software sendo medido. A fronteira reside entre os usuários funcionais e o referido software. b) Por definição, existe uma fronteira entre cada par de camadas identificado, onde o software em uma camada é o usuário funcional do software na outra camada, e este último é o software a ser medido. c) Há uma fronteira entre quaisquer dois pedaços de software, incluindo quaisquer dois componentes que sejam pares um do outro; neste caso cada pedaço de software e/ou cada componente pode ser um usuário funcional de seu par.

A fronteira permite que seja feita uma distinção clara entre qualquer coisa que seja parte do pedaço de software sendo medido (i.e., que esteja do lado da fronteira correspondente ao software) e qualquer coisa que seja parte do ambiente dos usuários funcionais. (i.e., do lado da fronteira correspondente aos usuários funcionais). O armazenamento persistente não é considerado um usuário do software e, dessa forma, está do mesmo lado da fronteira que o software.

2.4 Identificando o nível de granularidade

2.4.1 A necessidade de um nível de granularidade padrão

Quando um projeto está no início da concepção e construção de uma nova casa, os primeiros planos desenhados por um arquiteto estão em um 'nível alto', isto é, mostram um contorno e poucos detalhes. Conforme o projeto avança para a fase de construção, planos mais detalhados ('nível baixo') se fazem necessários.

O mesmo vale para o software. Nos estágios iniciais de um projeto de desenvolvimento de software, os Requisitos Funcionais do Usuário (RFU) são especificados 'a um nível mais alto', isto é, através de um esboço, ou em pequeno detalhe. Conforme o projeto avança os RFU são refinados (p.ex.,

[8] De fato, se o medidor tiver tido que examinar os RFU para identificar as fontes e destinos pretendidos para os dados, a fronteira já terá sido identificada.

através de versões 1, 2, 3, etc.), revelando mais detalhes 'a um nível mais baixo'. Esses diferentes graus de detalhe dos RFU são conhecidos como diferentes 'níveis de granularidade'. (Ver também a seção 2.4.3 para outros termos que podem ser confundidos com o conceito de 'nível de granularidade' conforme definido aqui.)

DEFINIÇÃO – Nível de granularidade
Qualquer nível de expansão da descrição de um pedaço de software distinto (p.ex., uma declaração de seus requisitos, ou uma descrição da estrutura do pedaço de software) tal que a cada nível de expansão adicional a descrição da funcionalidade do pedaço de software está em um nível de detalhe maior e uniforme. NOTA: Os medidores devem estar cientes de que, quando os requisitos evoluem cedo na vida de um projeto de software, a qualquer momento partes da funcionalidade requerida pelo mesmo terão sido documentadas em diferentes níveis de granularidade

As plantas de construção de uma casa são desenhadas segundo escalas padronizadas, sendo fácil traduzir as dimensões medidas em um desenho para outro em uma escala diferente. Por outro lado, não há escalas padronizadas correspondentes aos vários níveis de granularidade para especificação de software, de sorte que pode ser difícil ter certeza que duas declarações de RFU estão no mesmo nível de granularidade. Sem um acordo sobre um nível padrão de granularidade no qual medir (ou para o qual as medições devam ser escalonadas) é impossível saber com certeza se duas medições funcionais podem ser comparadas. Além disso, os medidores podem ter que desenvolver seu próprio método de escalonamento de medidas de um nível de granularidade para outro.

Para ajudar a ilustrar os problemas, consideremos outra analogia. Um conjunto de mapas rodoviários revela os detalhes de uma rede nacional de estradas em três níveis de granularidade.

- O mapa A mostra apenas estradas asfaltadas e estradas principais
- O mapa B mostra estradas asfaltadas, estradas principais e secundárias (assim como em um atlas para motoristas),
- O mapa C mostra todas as estradas com seus nomes (assim como em um conjunto de mapas distritais),

Se o fenômeno dos diferentes níveis de granularidade não fosse reconhecido, ficaria parecendo que esses três mapas revelam diferentes tamanhos da rede nacional de estradas. É claro que, no caso dos mapas rodoviários, todo o mundo entende os diferentes níveis de detalhe mostrados, existindo escalas padronizadas para interpretação do tamanho da rede em qualquer nível. O conceito abstrato de 'nível de granularidade' está por trás das escalas nesses diversos mapas.

Para a medição de software, há apenas um nível de granularidade que pode ser definido de forma não ambígua. Este é o nível de granularidade no qual os processos funcionais distintos foram identificados e as respectivas movimentações de dados definidas. As medições devem ser feitas neste nível, ou escalonadas para este nível sempre que possível[9].

2.4.2 Esclarecimento do 'nível de granularidade'

Antes de prosseguir, é importante garantir que não haja mal-entendido quanto ao significado de 'nível de granularidade' no método COSMIC. Conforme aqui definido, a expansão da *descrição* de um software de um nível de granularidade mais 'alto' para um mais 'baixo' envolve fazer um 'zoom' para dentro e revelar mais detalhes, *sem mudar seu escopo*. Este processo NÃO deve ser confundido com nenhum dos seguintes.

- Fazer 'zoom' para dentro de um artefato descrevendo um software para revelar diferentes subconjuntos da funcionalidade entregue a diferentes usuários, dessa forma provavelmente limitando a funcionalidade a ser medida

[9] O tópico referente ao escalonamento de medições de um nível de granularidade para outro já foi introduzido na v2.2 do Manual de Medição, no capítulo 7 sobre 'Aproximação do Tamanho Funcional no Início Usando COSMIC'. (Agora este tópico será encontrado no documento 'Tópicos Avançados e Relacionados').

- Fazer ‘zoom’ para dentro de um software a fim de revelar seus componentes, subcomponentes, etc. (em diferentes ‘níveis de decomposição’ – ver seção 2.2.2 acima). Tal ‘zoom’ para dentro pode ser requerido se o propósito da medição exigir que o escopo global da medição seja subdividido de acordo com a estrutura física do software
- Evoluir a descrição do software conforme a mesma progride através de seu ciclo de desenvolvimento, i.e., dos requisitos para o projeto lógico, projeto físico, etc. Qualquer que seja o estágio de desenvolvimento do software, estamos interessados apenas nos RFU para fins de medição

O conceito de ‘nível de granularidade’ deve então ser interpretado como aplicável apenas aos Requisitos Funcionais do Usuário de software. Outras maneiras de fazer ‘zoom’ para dentro, ‘quebrar’ o software ou suas diversas descrições também podem ser importantes ao utilizar ou comparar medições de tamanho funcional. Estes pontos serão tratados no documento ‘Tópicos Relacionados e Avançados, v3.0’, no capítulo ‘Garantindo a Comparabilidade das Medições de Tamanho’.

2.4.3 O nível padrão de granularidade

O nível padrão de granularidade para a medição é o ‘nível do processo funcional’, definido como segue[10].

DEFINIÇÃO – Nível de granularidade do processo funcional
Um nível de granularidade da descrição de um pedaço de software no qual os usuários funcionais • são seres humanos distintos, dispositivos de engenharia ou pedaços de software (e não quaisquer grupos desses itens) E • detetam ocorrências distintas de eventos aos quais o pedaço de software deve responder (e nenhum nível no qual grupos de eventos sejam definidos) NOTA 1: Na prática, a documentação do software contendo requisitos funcionais do usuário frequentemente descreve a funcionalidade a vários níveis de granularidade, especialmente quando a documentação ainda está evoluindo. NOTA 2: ‘Grupos desses' (usuários funcionais) poderiam ser, por exemplo, um ‘departamento’ cujos integrantes lidem com diversos tipos de processos funcionais; um ‘painel de controle’ com vários tipos de instrumentos; ou ‘sistemas centrais’. NOTA 3: ‘Grupos de eventos’ poderiam, por exemplo, ser indicados em uma declaração de RFU em um alto nível de granularidade por um fluxo de entrada direcionado a um sistema de software contábil e denominado ‘transações de venda’; ou por um fluxo de entrada para um sistema de software de aviação denominado ‘comandos do piloto’.

Com esta definição, podemos agora definir as seguintes regras e uma recomendação.

REGRAS - Nível de granularidade do processo funcional
a) A medição de tamanho funcional deve ser feita ao nível de granularidade do processo funcional b) Quando é necessária uma medição de tamanho funcional de alguns RFU que ainda não evoluíram até o nível no qual todos os processos funcionais foram identificados e todos os detalhes de suas movimentações de dados definidos, as medições devem ser feitas sobre as funcionalidades que foram definidas e, então, escalonadas para o nível de granularidade dos processos funcionais. (Ver o documento ‘Tópicos Avançados e Relacionados’ para métodos de ‘dimensionamento aproximado’, i.e., para estimar um tamanho funcional no início

[10] A razão para o nome ‘nível de granularidade do processo funcional’ é que este é o nível no qual os processos funcionais e suas movimentações de dados são identificados – ver seção 3.2 para uma discussão mais detalhada dos processos funcionais.

do processo de estabelecimento dos RFU.)

Além das regras, COSMIC *recomenda*[11] que o nível de granularidade dos processos funcionais deva ser o padrão segundo o qual os requisitos funcionais são requeridos e utilizados pelos provedores de serviços de benchmarking e ferramentas de software desenhadas para suportar e utilizar medições de tamanho funcional, p.ex., estimativa do esforço dos projetos.

EXEMPLO: Funcionalidade em diferentes níveis de granularidade

O exemplo – do domínio de software aplicativo de negócios – será de um sistema bem conhecido, para compras via Internet, denominado sistema 'Everest'. (o documento 'Tópicos Avançados e Relacionados' contém um exemplo do domínio de software 'real-time'.). A descrição abaixo foi simplificada para fins desta ilustração dos níveis de granularidade. A descrição também cobre somente a funcionalidade disponível para os usuários clientes do Everest. Exclui todas as funcionalidades que devem estar presentes para que o sistema possa fornecer as mercadorias ao consumidor, tais como aquelas disponíveis para os funcionários do Everest, fornecedores de produtos, anunciantes, fornecedores de serviços de pagamentos, etc.

O escopo da medição é então definido como 'partes do sistema aplicativo Everest acessíveis aos clientes pela Internet'. Assumimos que o propósito da medição é determinar o tamanho funcional da aplicação disponível para os usuários clientes humanos (como 'usuários funcionais'). No nível mais alto, 'nível 1' da aplicação como um todo, uma declaração dos Requisitos Funcionais do Usuário (RFU) do Sistema de Pedidos Everest seria uma simples declaração resumida como a que segue:

"O sistema Everest deve possibilitar que os clientes consultem, selecionem, solicitem, paguem e obtenham a entrega de qualquer item na faixa de produtos do Everest, inclusive produtos disponíveis de outros fornecedores."

Fazendo um 'zoom' para dentro desta declaração de alto nível, descobrimos que no próximo nível mais baixo, 2, o sistema consiste de quatro subsistemas, conforme mostrado na Fig. 2.4.3.1.

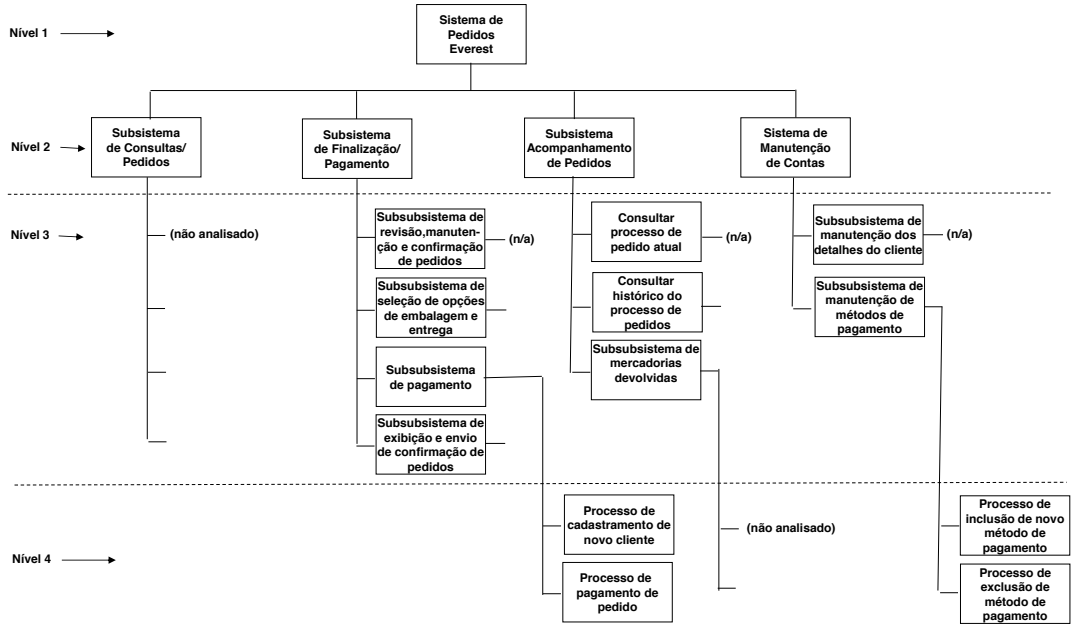


Figura 2.4.3.1 – Análise parcial do Sistema de Pedidos Everest mostrando quatro níveis de análise

[11] A razão pela qual o nível de granularidade dos processos funcionais é 'recomendada' ao invés de ser dada como uma regra é que esta recomendação não se aplica apenas a usuários individuais do método COSMIC, mas às suas redes de fornecedores de serviços e ferramentas que utilizam medições de tamanho. COSMIC pode apenas produzir recomendações para essa comunidade mais ampla.

Os quatro subsistemas são:

- O subsistema de Consulta/Pedido, que permite a um cliente encontrar qualquer produto no banco de dados Everest, assim como seu preço e disponibilidade, além de poder acrescentar qualquer produto selecionado a uma 'cesta' de compra. Este subsistema também promove as vendas, sugerindo ofertas especiais, oferecendo opiniões sobre itens selecionados e possibilitando consultas gerais tais como regras para entrega, etc. É um subsistema bastante complexo, não sendo analisado em maiores detalhes além do nível 2 para o propósito deste exemplo.
- O subsistema de Saída/Pagamento, que possibilita que um cliente confirme o pedido e pague as mercadorias na cesta.
- O subsistema de Acompanhamento de Pedidos, que possibilita que um cliente consulte o estado de um pedido no processo de entrega, mantenha seu pedido (p.ex., mude o endereço da entrega) e devolva mercadorias defeituosas.
- O subsistema de Manutenção de Contas, que possibilita que um cliente mantenha diversos itens de sua conta, tais como: endereço residencial, meios de pagamento, etc.

A figura 2.4.3.1 mostra alguns detalhes que são revelados quando fazemos um 'zoom' para dentro, dois níveis abaixo, nos subsistemas de Saída/Pagamento, Acompanhamento de Pedidos e Manutenção de Contas. Neste processo de 'zoom' é importante notar que

- não mudamos o escopo do sistema aplicativo a ser medido, e
- todos os níveis da descrição do sistema de software Everest mostram as funcionalidades disponíveis para os clientes (como usuários funcionais). Um cliente pode 'ver' as funcionalidades do sistema em todos esses níveis da análise.

A figura 2.4.3.1 também revela que, quando fazemos um 'zoom' na direção dos níveis mais baixos dessa análise dos três subsistemas, encontramos processos funcionais distintos no nível 3 (para duas consultas do subsistema de Acompanhamento de Pedidos) e no nível 4 para outros dois subsistemas. Este exemplo demonstra, portanto, que quando algumas funcionalidades são analisadas segundo uma abordagem de cima para baixo, não se pode assumir que as funcionalidades mostradas em um 'nível' específico de um diagrama sempre corresponderão ao mesmo 'nível de granularidade' conforme definição deste conceito no método COSMIC. (Tal definição requer que em qualquer nível de granularidade as funcionalidades estejam 'em um nível de detalhe comparável'.)

Adicionalmente, outros analistas poderiam desenhar o diagrama diferentemente, mostrando outros grupos de funcionalidades em cada nível do diagrama. Não há uma maneira 'correta' de se analisar a funcionalidade de um sistema tão complexo.[12]

Dadas tais variações que inevitavelmente ocorrem na prática, um medidor deve examinar cuidadosamente os vários níveis de um diagrama de análise para encontrar os processos funcionais que devem ser medidos. Quando isso não é possível na prática, por exemplo, porque a análise ainda não alcançou o nível onde todos os processos funcionais foram revelados, a regra (b) acima deve ser aplicada. Para ilustrar, vamos examinar o caso 'Subsubsistema de manutenção dos detalhes do cliente' (ver figura 2.4.3.1 acima), no ramo do subsistema de Manutenção de Contas.

Para um medidor experiente, a palavra 'manter' quase sempre sugere um grupo de eventos e, assim, um grupo de processos funcionais. Podemos dessa forma, seguramente, assumir que este subsubsistema 'de manutenção' deve compreender três processos funcionais, quais sejam: 'consultar detalhes do cliente', 'atualizar detalhes do cliente' e 'excluir cliente'. (o processo 'criar detalhes do cliente' obviamente também deve existir, mas isto ocorre em outro ramo do sistema, quando um cliente compra mercadorias pela primeira vez. Este último caso está fora do escopo do exemplo simplificado.)

Um medidor experiente será capaz de 'estimar' o tamanho deste subsubsistema em unidades PFC, tomando o número assumido de processos funcionais (três neste caso) e multiplicando este número pelo tamanho médio de um processo funcional. O tamanho médio seria calculado através da calibração em outras partes deste sistema, ou em outros sistemas comparáveis. Exemplos deste processo de calibração são fornecidos no documento 'Tópicos Avançados e Relacionados' sobre o dimensionamento aproximado, bem como outros exemplos de abordagens para o dimensionamento aproximado.

É claro que tais métodos de aproximação possuem limitações. Se aplicarmos tal abordagem a uma declaração de RFU no nível 1 conforme apresentado acima ("*O sistema Everest deve possibilitar que os clientes consultem, selecionem, solicitem, paguem e obtenham a entrega de qualquer item na faixa de produtos do Everest...*"), poderíamos identificar alguns processos funcionais. No entanto, uma análise mais detalhada revelaria que o número real de processos funcionais deve ser muito maior. É por isso que os tamanhos funcionais geralmente parecem crescer conforme mais detalhes dos requisitos

[12] A figura 2.4.3.1 pode nem ser um exemplo de melhor prática, mas é típica de como tais diagramas podem ser desenhados.

são estabelecidos, mesmo sem mudanças no escopo. Tais métodos de aproximação devem então ser utilizados com grande cuidado nos níveis de granularidade altos, quando poucos detalhes estão disponíveis.

O documento 'Tópicos Avançados e Relacionados' fornece um exemplo de dimensionamento de tamanho diferente e a vários níveis de granularidade, neste caso um software que faz parte da arquitetura de software de telecomunicações. Este exemplo é mais complexo do que o exemplo 'Everest', já que os usuários funcionais da parte da arquitetura sendo medida são outros pedaços de software da mesma arquitetura. Este exemplo então também lida com os vários níveis de decomposição do software sendo medido e de seus usuários funcionais.

2.5 Comentários finais sobre a fase estratégia de medição

A consideração cuidadosa dos quatro elementos do processo pertinente à estratégia de medição antes de começar uma medição deve garantir que o tamanho resultante possa ser interpretado adequadamente. Os quatro elementos são:

a) estabelecer o *propósito* da medição
b) definir o *escopo* total do pedaço de software a ser medido e escopo dos pedaços a serem medidos separadamente, considerando as camadas e componentes pares da arquitetura de software
c) estabelecer os *usuários funcionais* do pedaço de software a ser medido
d) estabelecer o nível de granularidade dos artefatos de software a serem medidos e como, se necessário, escalonar os tamanhos para o nível padrão de granularidade dos processos funcionais

Algumas iterações podem ser necessárias em torno dos passos (b), (c) e (d) quando os requisitos estão evoluindo e novos detalhes indicam a necessidade de refinar a definição do escopo.

A grande maioria das medições de tamanho funcional é executada para um propósito de alguma forma relacionado ao esforço de desenvolvimento, p.ex., para medir o desempenho dos desenvolvedores ou estimar projetos. Nessas situações, a definição da estratégia de medição deve ser imediata. O propósito e o escopo são normalmente fáceis de definir, os usuários funcionais são os usuários para quem o desenvolvedor deve prover a funcionalidade e o nível de granularidade requerido para as medições é aquele no qual os usuários funcionais identificam eventos distintos.

Contudo, nem todas as medições seguem este padrão comum, de modo que a estratégia de medição deve ser definida em cada situação.

3

A FASE DE MAPEAMENTO

Este capítulo apresenta as regras e o método para o processo de mapeamento. O método geral para mapear software para o Modelo Genérico de Software COSMIC é resumido na Figura 3.0.

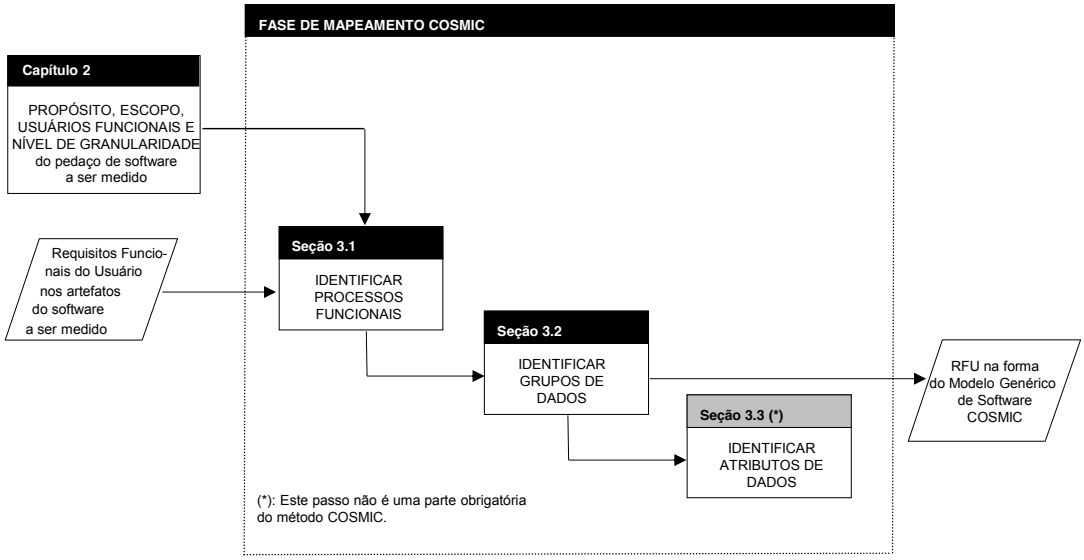


Figura 3.0 – Método geral do processo de mapeamento de software COSMIC

Cada passo neste método é o assunto de uma seção específica (indicada na barra de título do passo na figura 3) onde as definições e regras são apresentadas, acompanhadas de princípios e exemplos.

O método geral delineado na figura 3 acima foi projetado para ser aplicável a uma ampla gama de artefatos de software. Um processo mais sistemático e detalhado proveria regras de mapeamento precisas para uma coleção maior de artefatos altamente específicos, dessa forma diminuindo o nível de ambiguidade na geração do Modelo Genérico de Software COSMIC. Tal processo seria, por definição, altamente dependente da natureza dos artefatos, a qual, por sua vez, depende da metodologia de engenharia de software utilizada em cada organização.

O documento 'Guia para o Dimensionamento de Aplicações de Negócio utilizando COSMIC' fornece orientações sobre o mapeamento de vários métodos de análise de dados e determinação de requisitos para os conceitos do método COSMIC.

3.1 Aplicando o Modelo Genérico de Software

PRINCÍPIO – Aplicando o Modelo Genérico de Software COSMIC
O Modelo Genérico de Software COSMIC deve ser aplicado aos requisitos funcionais do usuário de cada pedaço de software distinto para o qual um escopo de medição distinto tenha sido definido. 'Aplicando o Modelo Genérico de Software COSMIC' diz respeito à identificação do conjunto de eventos disparadores percebidos por cada um dos (tipos de) usuários funcionais identificados nos RFU, seguida pela identificação dos correspondentes processos funcionais, objetos de interesse, grupos de dados e movimentações de dados que devem ser providos como resposta a esses eventos.

As Figuras 3.1.1 e 3.1.2 abaixo ilustram a aplicação do princípio (g) do Modelo de Contexto de Software e dos princípios do Modelo Genérico de Software a um pedaço de software de aplicação de negócio e a um pedaço típico de software 'real-time' embarcado, respectivamente.

Enquanto as Figuras 2.2.3.1 e 2.2.3.2 acima mostraram visões físicas reais das arquiteturas de software em camadas, respectivamente, estas Figuras 3.1.1 e 3.1.2 mostram uma visão lógica dos relacionamentos entre os vários conceitos definidos nos Modelos COSMIC. Nesta visão lógica os usuários funcionais interagem com o software a ser medido através de uma fronteira, via movimentações de dados do tipo Entry e Exit. O software movimenta dados de e para o armazenamento persistente via movimentações de dados do tipo Write e Read, respectivamente. Nestas visões lógicas são ignorados todo o hardware e software necessários para que tais trocas possam acontecer entre o software medido, seus usuários e o armazenamento persistente.

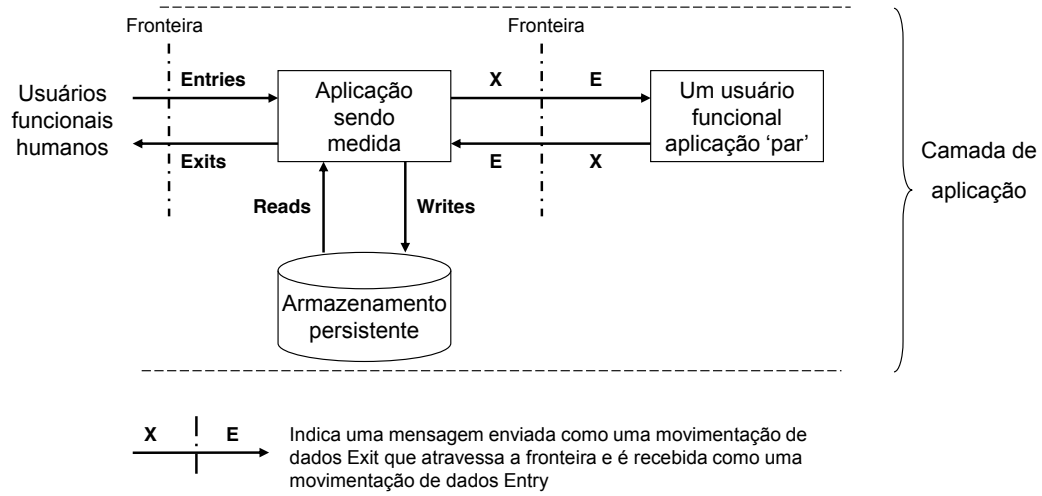


Figura 3.1.1 – Uma aplicação de negócio com seres humanos e outra aplicação 'par' como seus usuários funcionais (visão lógica)

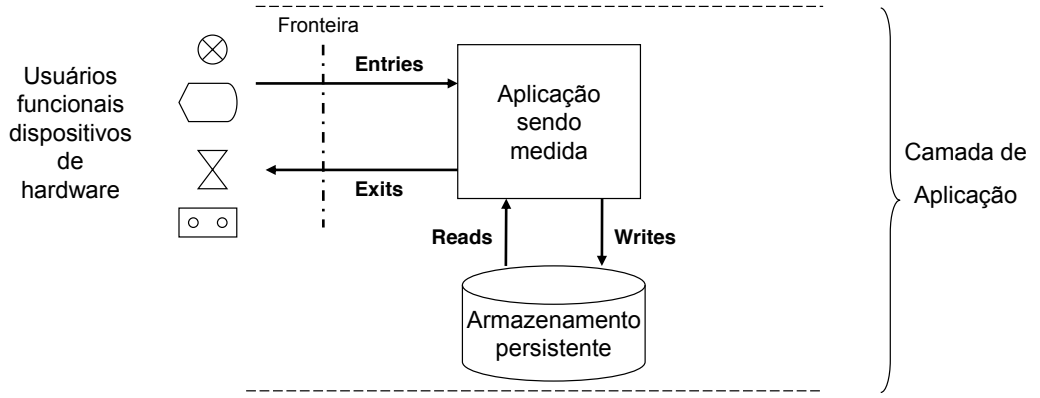


Figura 3.1.2 – Uma aplicação de software 'real-time' embarcado com vários dispositivos de hardware como seus usuários funcionais (visão lógica)

3.2 Identificando processos funcionais

Este passo consiste em identificar o conjunto de processos funcionais do pedaço de software a ser medido, a partir de seus Requisitos Funcionais do Usuário.

3.2.1 Definições

DEFINIÇÃO – Processos funcionais
Um processo funcional é um componente elementar de um conjunto de Requisitos Funcionais do Usuário compreendendo um conjunto de movimentações de dados único, coeso e independentemente executável. É disparado por uma movimentação de dados (uma Entry) de um usuário funcional que informa ao pedaço de software que o usuário funcional identificou um evento disparador. Estará concluído quando tiver executado tudo o que é requerido para ser feito em resposta ao evento disparador. NOTA: Além de informar ao pedaço de software a ocorrência do evento, a 'Entry disparadora' deve incluir dados sobre o objeto de interesse associado ao mesmo.

DEFINIÇÃO – Evento disparador
Um evento (algo que acontece) que faz com que um usuário funcional do pedaço de software inicie ('dispare') um ou mais processos funcionais. Em um conjunto de Requisitos Funcionais do Usuário, cada evento que faz com que um usuário funcional dispare um processo funcional • não pode ser subdividido para aquele conjunto de RFU, e • aconteceu ou não aconteceu. NOTA: O relógio e eventos relativos ao passar do tempo podem ser eventos disparadores.

O relacionamento entre um evento disparador, o usuário funcional e a movimentação de dados Entry que dispara um processo funcional é mostrado na figura 3.2.1 abaixo. A interpretação deste diagrama é: *um evento é percebido por um usuário funcional e o usuário funcional dispara um processo funcional*.

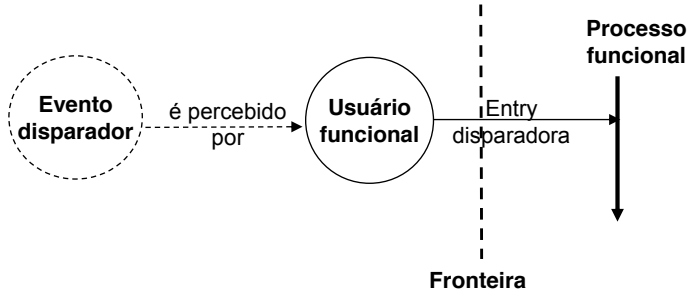


Figura 3.2.1 – Relação entre evento disparador, usuário funcional e processo funcional

A Entry disparadora é normalmente uma mensagem positiva e não ambígua que informa ao software que o usuário funcional identificou um evento disparador. A Entry disparadora muitas vezes também inclui dados sobre um objeto de interesse associado ao evento. Depois de recebida a Entry disparadora, um processo funcional pode ser requerido para receber e processar outras Entries descrevendo outros objetos de interesse.

Se um usuário funcional enviar dados incorretos, p.ex., porque um usuário-sensor está funcionando mal ou um pedido submetido por um ser humano contém erros, cabe geralmente ao processo funcional determinar se o evento realmente ocorreu e/ou se os dados submetidos são realmente válidos e como responder.

3.2.2 A abordagem para a identificação de processos funcionais

A abordagem para a identificação de processos funcionais depende dos artefatos de software disponíveis para o medidor, o que por sua vez depende do ponto no ciclo de vida do software no qual a medição é requerida, bem como nos métodos de análise, projeto e desenvolvimento utilizados. Como estes últimos variam bastante mesmo dentro de um dado domínio de software, está além do escopo deste Manual de Medição prover um ou mais processos genéricos para a identificação de processos funcionais.

O 'Guia para a Medição do Tamanho de Software de Aplicações de Negócio utilizando COSMIC', seção 4.4, fornece mais regras e exemplos sobre como identificar e distinguir processos funcionais no domínio de software de aplicações de negócio.

A recomendação geral mais importante é: quase sempre, é útil tentar identificar os eventos distintos no mundo dos usuários funcionais a que o software deve responder, pois cada um desses eventos (tipo) normalmente dá origem a um (porém às vezes mais de um) processo funcional (tipo). Os eventos podem ser identificados em diagramas de estado e em diagramas de ciclo de vida de entidades, uma vez que cada transição com a qual o software precisa lidar corresponde a um evento.

Usar as seguintes regras para verificar se os processos funcionais candidatos foram corretamente identificados.

REGRAS – Processo funcional
a) Um processo funcional deve ser derivado de pelo menos um Requisito Funcional do Usuário identificável dentro do escopo acordado.
b) Um processo funcional (tipo) deve ser executado quando um evento disparador identificável (tipo) ocorre.
c) Um evento específico (tipo) pode disparar um ou mais processos funcionais (tipos) que executem em paralelo. Um processo funcional específico (tipo) pode ser

disparado por mais de um evento (tipo).

d) Um processo funcional deve compreender pelo menos duas movimentações de dados, uma Entry mais (uma Exit ou um Write).

e) Um processo funcional deve pertencer completamente ao escopo de medição de um pedaço de software em uma e somente uma camada.

f) No contexto de software 'real-time', um processo funcional deve ser considerado concluído quando o mesmo entra em um estado de espera auto-induzido (i.e., o processo funcional já fez tudo o que é requerido como resposta ao evento disparador e espera até receber a próxima Entry disparadora).

g) Um processo funcional (tipo) deve ser identificado mesmo quando seus RFU permitirem que o referido processo funcional ocorra com diferentes subconjuntos de seu número máximo de atributos de dados de entrada, e mesmo que tais variações e/ou diferentes valores de dados de entrada possam dar origem a diferentes caminhos de processamento através do processo funcional.

h) Eventos distintos (tipos) e consequentemente processos funcionais distintos (tipos) devem ser identificados separadamente nos seguintes casos:

- Quando as decisões resultarem em eventos separados e desacoplados no tempo (p.ex., entrar com os dados do pedido hoje e mais tarde confirmar o aceite do pedido, requerendo uma decisão separada, deveria ser considerado como indicação de processos funcionais distintos),
- Quando as responsabilidades pelas atividades estão separadas (p.ex., em um sistema de pessoal onde a responsabilidade pela manutenção dos dados básicos de pessoal e dos dados de pagamento estão separadas, indicando processos funcionais distintos; ou no caso de um pacote de software implementado onde há funcionalidades disponíveis para que um administrador de sistema mantenha os parâmetros do pacote, separadas das funcionalidades disponíveis para os usuários funcionais 'normais'.)

3.2.3 Eventos disparadores e processos funcionais no domínio de aplicações de negócio

a) Os eventos disparadores de uma aplicação 'on-line' de negócio geralmente ocorrem no mundo real do usuário funcional humano da aplicação. O usuário humano transmite a ocorrência do evento para um processo funcional submetendo dados a respeito do evento.

EXEMPLO 1: Em uma empresa, um pedido é recebido (evento disparador) fazendo com que um empregado (usuário funcional) submeta os dados do pedido (Entry disparadora transmitindo dados sobre um objeto de interesse, 'pedido'), como a primeira movimentação de dados do processo funcional 'entrada de pedido'.

b) Às vezes, para uma dada aplicação A pode existir uma aplicação par B que precise enviar/obter dados para/da aplicação A. Neste caso, se a aplicação B dispara um processo funcional quando precisa enviar/obter dados para/da aplicação A, então a aplicação B é um usuário funcional da aplicação A.

EXEMPLO 2: Suponhamos que, na recepção de um pedido no Exemplo (1), seja necessário que a aplicação de pedidos envie os detalhes do cliente a uma aplicação central de cadastramento de clientes, a qual está sendo medida. Agora, a aplicação de pedidos tornou-se um usuário funcional da aplicação central. A aplicação de pedidos percebe o evento de recebimento dos dados do cliente e dispara um processo funcional na aplicação central de cadastramento de clientes para armazenar esses dados, enviando dados sobre o objeto de interesse 'cliente' como uma Entry disparadora para a aplicação central.

c) A princípio não há diferença na análise de um processo funcional se for requerido que o mesmo seja processado em modo 'on-line' ou 'batch'. O processamento noturno é uma decisão sobre a implementação técnica, e 'tudo que é requerido fazer' não terá acontecido até que o processamento noturno tenha sido concluído. Uma cadeia de processamento 'batch' consiste de um ou mais processos funcionais (tipos), devendo cada qual ser analisado do início ao fim, independentemente de quaisquer outros processos funcionais na mesma cadeia. Cada processo funcional tem a sua própria Entry disparadora a ser medida.

EXEMPLO 3: Suponhamos que os pedidos no Exemplo 1 acima sejam submetidos 'on-line', mas sejam armazenados para processamento 'batch' automático noturno. O usuário funcional ainda é o ser humano que submeteu os pedidos 'on-line'; a Entry disparadora ainda é constituída pelos dados do pedido. Há um processo funcional para submissão e processamento dos pedidos.

d) Sinais periódicos de um relógio ('tiques do relógio') podem disparar fisicamente um processo funcional.

EXEMPLO 4: Suponhamos um RFU para um processamento 'batch' de final de ano a fim de reportar o resultado do negócio para o ano, e 'zerar' as posições para o início do ano seguinte. Fisicamente, um tique de final do ano do relógio gerado pelo sistema operacional inicia a cadeia 'batch', consistindo de um ou mais processos funcionais. Os processos funcionais na cadeia que utilizam dados de entrada na cadeia devem ser analisados normalmente (i.e., os dados de entrada para qualquer processo funcional compreendem uma ou mais Entries, a primeira das quais é a Entry disparadora para aquele processo).

Contudo, assumamos que há um processo funcional específico na cadeia que não requer dados de entrada para produzir seu conjunto de relatórios. Fisicamente, o usuário funcional (humano) delegou ao sistema operacional a tarefa de disparar este processo funcional. Como cada processo funcional deve possuir uma Entry disparadora, podemos considerar o tique de fim de ano do relógio que iniciou a cadeia 'batch' como fazendo este papel para este processo. Este processo funcional pode então precisar de vários Reads e Exits para produzir seus relatórios. Logicamente, a análise deste exemplo não mudará se o usuário funcional iniciar a produção de um ou mais relatórios através de um clique do 'mouse' sobre um item de menu, ao invés de delegar a produção do relatório ao sistema operacional através de uma cadeia 'batch'.

e) Um único evento pode disparar um ou mais processos funcionais que executem independentemente.

EXEMPLO 5: Um tique de final de semana do relógio inicia uma produção de relatórios, bem como o processo de revisão da expiração dos limites de tempo em um sistema de 'workflow'.

f) Um único processo funcional pode ser disparado por mais de um tipo de evento disparador.

EXEMPLO 6: Em um sistema bancário, um extrato completo pode ser disparado por processo 'batch' de final de mês, mas também por uma solicitação específica do cliente.

Para diversos exemplos sobre como distinguir eventos disparadores e processos funcionais em cadeias 'batch', ver o 'Guia para a Medição do Tamanho de Software de Aplicações de Negócio utilizando COSMIC', seção 4.6.3.

3.2.4 Eventos disparadores e processos funcionais no domínio de aplicações 'real-time'

a) Um evento disparador é tipicamente identificado por um sensor.

EXEMPLO 1: Quando a temperatura alcança um dado valor (evento disparador), um sensor (usuário funcional) envia um sinal (movimentação de dados por uma Entry disparadora) para ligar uma luz de advertência (processo funcional).

EXEMPLO 2: Uma aeronave militar possui um sensor que identifica o evento 'míssil se aproximando'. O sensor é um usuário funcional do software embarcado que precisa responder à ameaça. Para este software, um evento ocorre somente quando o sensor identifica alguma coisa, e é o sensor (usuário funcional) que dispara o software, enviando a este uma mensagem (Entry disparadora) dizendo, p.ex., 'o sensor 2 identificou um míssil', mais possivelmente uma cadeia de dados sobre a velocidade de aproximação do míssil e suas coordenadas. O objeto de interesse é o míssil.

b) Sinais periódicos de um relógio ('tiques do relógio') podem disparar um processo funcional.

EXEMPLO 3: Em alguns softwares de controle de processo em tempo real um tique (evento disparador) de um relógio (usuário funcional) faz com que o relógio envie um sinal (Entry disparadora), uma mensagem de um 'bit' que diz a um processo funcional que repita seu ciclo de controle normal. O processo funcional então lê vários sensores, recebendo dados sobre objetos de interesse, e executa qualquer ação necessária. Não há outros dados acompanhando o tique do relógio.

c) Um único evento pode disparar um ou mais processos funcionais que executam independentemente e em paralelo.

EXEMPLO 4: Uma condição de emergência identificada em uma usina nuclear pode disparar processos funcionais independentes em diversas partes da usina para baixar as hastes de controle, iniciar resfriamento de emergência, fechar válvulas, tocar alarmes, avisar os operadores, etc.

d) Um único processo funcional pode ser disparado por mais de um tipo de evento disparador.
EXEMPLO 5: A retração das rodas de uma aeronave pode ser disparada pelo sensor de 'peso fora do chão', ou pelo comando do piloto.

3.2.5 Mais sobre processos funcionais distintos

O software distingue os eventos e provê os processos funcionais correspondentes, dependendo apenas de seus RFU. No dimensionamento de software, às vezes pode ser difícil decidir quais são os eventos distintos que o software precisa reconhecer. Este é o caso principalmente quando os RFU originais não estão mais disponíveis e onde, por exemplo, o desenvolvedor pode ter achado econômico combinar diversos requisitos. Pode ser útil à análise examinar a organização dos dados de entrada (ver abaixo) ou examinar os menus de algum software instalado para ajudar a distinguir os eventos distintos aos quais o software deve responder e os processos funcionais correspondentes.

EXEMPLO 1: Quando há um requisito funcional do usuário relativo a créditos fiscais por um filho adicional e, também, relativo a 'créditos fiscais do trabalho' para pessoas de baixa renda, tais são requisitos para que o software responda a dois eventos separados no mundo dos usuários funcionais humanos. Dessa forma devem existir dois processos funcionais, embora um único formulário fiscal possa ter sido usado para capturar os dados de ambos os casos.

EXEMPLO 2: Se a renda em um caso (para uma pessoa) exceder o limite para o crédito fiscal do trabalho, e em outro caso (para uma pessoa diferente) isso não acontecer, tal diferença não dará origem a dois processos funcionais distintos, mas indicará duas condições distintas que serão tratadas no (único) processo funcional.

EXEMPLO 3: Como um exemplo da regra (g), se uma ocorrência de um evento específico disparar a Entry de um grupo de dados compreendendo os atributos de dados A, B e C, e o RFU permitir que outra ocorrência do mesmo evento dispare uma Entry de um grupo de dados que possui valores apenas para os atributos A e B, isto não resulta na identificação de dois processos funcionais (tipos). Apenas uma Entry e um processo funcional (tipo) serão identificados, movimentando e manipulando os atributos de dados A, B e C.

Uma vez identificado, cada processo funcional pode ser registrado em uma linha individual na camada ou componente par apropriado, na matriz do Modelo Genérico de Software (apêndice A), sob o rótulo correspondente.

3.2.6 Os processos funcionais de componentes pares

Quando o propósito de uma medição for medir separadamente o tamanho de cada componente par, um escopo de medição distinto deverá ser definido para cada componente par. Em tal caso, o dimensionamento dos processos funcionais de cada componente seguirá todas as regras normais já descritas.

A partir do processo utilizado em cada medição (... definir o escopo, os usuários funcionais e a fronteira, etc...) segue que se um pedaço de software consistir de dois ou mais componentes pares, não poderá existir nenhuma superposição entre os escopos de medição dos componentes. O escopo de medição de cada componente deverá definir um conjunto completo de processos funcionais. Por exemplo, não poderá existir um processo funcional com parte em um escopo e parte em outro. Da mesma forma, os processos funcionais dentro do escopo de medição de um componente não têm conhecimento dos processos funcionais dentro do escopo de outro componente, embora os dois componentes troquem mensagens.

Os usuários funcionais de cada componente são determinados examinando-se onde ocorrem os eventos que disparam processos funcionais no componente examinado. (Eventos disparadores só podem ocorrer no mundo de um usuário funcional.)

A Figura 4.1.8.2 ilustra os processos funcionais de dois componentes pares e as movimentações de dados que os mesmos trocam.

3.3 Identificando objetos de interesse e grupos de dados

3.3.1 Definições e princípios

Este passo consiste em identificar os grupos de dados referenciados pelo pedaço de software sendo medido. Para identificar grupos de dados, especialmente no domínio de software de aplicações de negócio, geralmente é útil identificar os objetos de interesse e provavelmente também os seus atributos. Os grupos de dados são movimentados através de 'movimentações de dados' identificadas no próximo capítulo.

DEFINIÇÃO – Objeto de interesse
Qualquer 'coisa' que é identificada do ponto de vista dos Requisitos Funcionais do Usuário. Pode ser qualquer coisa física, assim como qualquer objeto conceitual ou parte de um objeto conceitual no mundo do usuário funcional, a respeito do qual o software deve processar e/ou armazenar dados. NOTA: No método COSMIC o termo 'objeto de interesse' é utilizado a fim de evitar termos relacionados a métodos específicos da engenharia de software. O termo não diz respeito a 'objetos' no sentido usado pelos métodos Orientados a Objetos.

DEFINIÇÃO – Grupo de dados
Um grupo de dados é um conjunto de atributos de dados distinto, não vazio, não ordenado e não redundante, onde cada atributo de dado incluído descreve um aspecto complementar do mesmo objeto de interesse.

DEFINIÇÃO – Armazenamento persistente
O armazenamento persistente é o armazenamento que permite a um processo funcional armazenar um grupo de dados além da vida do processo funcional, e/ou de onde um processo funcional pode recuperar um grupo de dados armazenado por outro processo funcional, por uma ocorrência anterior do mesmo processo funcional, ou por algum outro processo. NOTA 1: No modelo COSMIC o armazenamento persistente não é considerado um usuário do software, por residir no mesmo lado da fronteira que o software. NOTA 2: Um exemplo de 'algum outro processo' seria a fabricação de memória 'read-only'.

Uma vez identificado, cada grupo de dados candidato deve obedecer aos seguintes princípios:

PRINCÍPIOS – Grupo de dados
a) Cada grupo de dados identificado deve ser único e distinguível através de sua coleção única de atributos de dados b) Cada grupo de dados deve ser diretamente relacionado a um objeto de interesse nos Requisitos Funcionais do Usuário c) Um grupo de dados deve ser materializado dentro do sistema de computador que suporta o software.

Uma vez identificado, cada grupo de dados pode ser registrado em uma coluna individual da matriz do Modelo Genérico de Software (apêndice A), sob o rótulo correspondente.

3.3.2 Sobre a materialização de um grupo de dados

Na prática, a materialização de um grupo de dados pode assumir várias formas, por exemplo:

a) Como uma estrutura física de registros em um dispositivo persistente (arquivo, tabela de banco de dados, memória ROM, etc.).

b) Como uma estrutura física dentro da memória volátil do computador (estrutura de dados alocada dinamicamente ou através de um bloco de memória pré-alocado).

c) Como a apresentação conjunta de atributos de dados funcionalmente relacionados, em um dispositivo de entrada/saída (tela de vídeo, relatório impresso, tela de um painel de controle, etc.).

d) Como uma mensagem na transmissão entre um dispositivo e um computador, em uma rede, etc.

3.3.3 Sobre a identificação de objetos de interesse e grupos de dados

A definição e os princípios referentes a objetos de interesse e grupos de dados são intencionalmente genéricos para que sejam aplicáveis à faixa de software mais ampla que for possível. Esta característica por vezes faz com que seja difícil aplicar as definições e princípios à medição de um pedaço de software específico. Dessa forma, os exemplos seguintes foram projetados para auxiliar a aplicação dos princípios a casos específicos.

Diante da necessidade de analisar um grupo de atributos de dados que é movimentado para dentro ou para fora de um processo funcional, ou movimentado por um processo funcional para dentro ou para fora do armazenamento persistente, é criticamente importante decidir se todos os atributos transmitem dados sobre um único 'objeto de interesse', pois é este último que determina o número de 'grupos de dados' distintos, conforme definido pelo método COSMIC. Por exemplo, se os atributos de dados a serem submetidos a um processo funcional são atributos de três objetos de interesse distintos, então precisaremos identificar três movimentações de dados do tipo 'Entry' distintas.

Objetos de interesse e grupos de dados no domínio de aplicações de negócio

EXEMPLO 1: No domínio de software de aplicações de negócio, um objeto de interesse poderia ser 'empregado' (físico) ou 'pedido (conceitual), supondo que o software precise armazenar dados sobre empregados ou pedidos. No caso do 'pedido', decorre normalmente do RFU correspondente a pedidos com várias linhas que dois objetos de interesse serão identificados: 'pedido' e 'linha do pedido'. Os grupos de dados correspondentes poderiam ser denominados 'dados do pedido' e 'dados da linha do pedido'.

Grupos de dados são formados sempre que há uma consulta 'ad hoc' que solicita dados sobre alguma 'coisa' a respeito da qual não são mantidos dados no armazenamento persistente, mas que podem ser derivados dos dados mantidos no armazenamento persistente. A movimentação de dados Entry em uma consulta 'ad hoc' (os parâmetros de seleção para derivar os dados requeridos) e a movimentação de dados Exit (contendo os atributos desejados) movimentam grupos de dados sobre uma dessas 'coisas'. Esses são grupos de dados transientes que não sobrevivem à execução do processo funcional. São grupos de dados válidos porque atravessam a fronteira entre o software e o(s) seu(s) usuário(s).

EXEMPLO 2: Seja uma consulta 'ad hoc' contra um banco de dados de pessoal para extrair uma lista de nomes de todos os empregados acima de 35 anos. A Entry movimenta um grupo de dados contendo os parâmetros de seleção. A Exit movimenta um grupo de dados contendo o atributo individual 'nome'; o 'objeto de interesse' (ou 'coisa') é 'todos os empregados acima de 35 anos de idade'. Ao registrar o processo funcional, é importante denominar claramente um grupo de dados transiente em relação ao seu objeto de interesse, ao invés de relacioná-lo ao(s) objeto(s) de interesse a partir do(s) qual(is) o resultado da consulta 'ad hoc' é derivado.

Para uma discussão detalhada sobre os métodos de análise de dados para determinar objetos de interesse e grupos de dados distintos, o leitor é remetido ao 'Guia para a Medição do Tamanho de Software de Aplicações de Negócio usando COSMIC'.

Objetos de interesse e grupos de dados no domínio de software 'real-time'

EXEMPLO 3: Movimentos de dados que são Entries de dispositivos físicos tipicamente contém dados sobre o estado de um único objeto de interesse, por exemplo: se uma válvula está aberta ou fechada, ou indicam um momento no qual os dados contidos na memória volátil de curto prazo estarão válidos ou inválidos, ou se a mesma contém dados que indicam que um evento crítico ocorreu – o que causa uma interrupção. Da mesma forma, um comando Exit para ligar/desligar uma luz de advertência transmite dados sobre um único objeto de interesse.

EXEMPLO 4: Um comutador de mensagens poderia receber um grupo de dados de mensagem como uma Entry e roteá-la sem modificações como uma Exit, de acordo com o RFU desse pedaço de software específico. Os atributos do grupo de dados de mensagem poderiam ser, por exemplo, 'remetente, destinatário, código de rota e conteúdo da mensagem'; o objeto de interesse da mensagem é 'mensagem'.

EXEMPLO 5: Uma estrutura de dados comum, representando objetos de interesse mencionados nos Requisitos Funcionais do Usuário, que pode ser mantida por processos funcionais, e que é acessível à maior parte dos processos funcionais encontrados no software medido.

EXEMPLO 6: Uma estrutura de dados de referência, representando gráficos ou tabelas de valores encontrados nos Requisitos Funcionais do Usuário, que é mantida na memória permanente (memória ROM, por exemplo) e que é acessível à maioria dos processos funcionais encontrados no software medido.

EXEMPLO 7: Arquivos, comumente denominados arquivos 'flat' ('planos'), representando objetos de interesse mencionados nos Requisitos Funcionais do Usuário, que são mantidos em um dispositivo de memória persistente.

3.3.4 Dados ou grupos de dados que não são candidatos a movimentações de dados

Quaisquer dados aparecendo nas telas ou relatórios de entrada ou saída que não sejam relacionados a um objeto de interesse de um usuário funcional não deveriam ser identificados como indicando uma movimentação de dados, e dessa forma não deveriam ser medidos.

EXEMPLO 1: Dados gerais de aplicação tais como cabeçalhos e rodapés (nome da empresa, nome da aplicação, data do sistema, etc.) aparecendo em todas as telas

EXEMPLO 2: Comandos de controle (um conceito definido somente no domínio de aplicações de negócio) que permitem a um usuário funcional controlar o uso do software, em vez de movimentar dados, por exemplo: comandos de paginação, clicar 'OK' para reconhecer uma mensagem de erro, etc. – ver adiante a seção 4.1.10.

O Modelo Genérico de Software COSMIC assume que toda a manipulação de dados dentro de um processo funcional está associada a um de quatro tipos de movimentações de dados – ver seção 4.1.6. Dessa forma, nenhum movimento ou manipulação de dados dentro de um processo funcional pode ser identificado como candidato a alguma movimentação de dados em acréscimo a Entries, Exits, Reads e Writes. (Para exemplos de manipulações e movimentos de dados que podem ser erradamente interpretados como movimentações de dados, ver a seção 4.1.4, princípio (c) para um Read e seção 4.1.5 princípio (d) para um Write).

3.3.5 O usuário funcional como um objeto de interesse

Em muitos sistemas de software 'real-time' simples, tal como o descrito no Exemplo 3 da seção 3.3.3 acima, é impossível distinguir o dispositivo físico – um usuário funcional – do objeto de interesse da movimentação de dados que ele gera ou recebe. Em tais casos é de pouco valor documentar um objeto de interesse como se fosse algo distinto do usuário funcional. O ponto importante é utilizar esses conceitos, quando úteis, para identificar os grupos de dados distintos e assim separar as movimentações de dados.

EXEMPLO: Seja um sensor de temperatura 'A' que, quando consultado por um processo funcional, envia ao mesmo a temperatura atual. O usuário funcional é o sensor A; o nome da mensagem Entry poderia ser 'temperatura atual em A'; o objeto de interesse desta mensagem poderia também ser visto como 'sensor A'. Teoricamente, o objeto de interesse não é o 'sensor A', mas sim 'o material ou objeto cuja temperatura está sendo percebida pelo sensor A'. Na prática, contudo, acrescenta pouco valor documentar tais distinções sutis, podendo não valer a pena identificar o objeto de interesse separadamente.

3.4 Identificando atributos de dados (opcional)

Esta seção discute a identificação de atributos de dados referenciados pelo pedaço de software a ser medido. Nesta versão do método de medição, não é obrigatório identificar os atributos de dados. Contudo, pode ser útil analisar e identificar os atributos de dados no processo de distinguir grupos de dados e objetos de interesse. Os atributos de dados também podem ser identificados se for requerida uma subunidade da medida de tamanho, conforme apresentado na seção 4.5, 'Ampliando o método de medição COSMIC'.

3.4.1 Definição

DEFINIÇÃO – Atributo de Dados
Um atributo de dados é a menor parcela de informação, dentro de um grupo de dados identificado, que possui um significado segundo a perspectiva dos Requisitos Funcionais do Usuário do software.

EXEMPLO 1: Atributos de dados no contexto do domínio das aplicações de negócio são, por exemplo, elementos de dados cadastrados no dicionário de dados e atributos de dados que aparecem em um modelo de dados conceitual ou lógico.

EXEMPLO 2: Atributos de dados no contexto do software de aplicação 'real-time' são, por exemplo, atributos de dados de um sinal recebido de um sensor e atributos de dados de uma mensagem em transmissão.

3.4.2 Sobre a associação de atributos de dados e grupos de dados

Na teoria, um grupo de dados pode conter apenas um atributo de dados se isto for suficiente para descrever o objeto de interesse, segundo a perspectiva dos Requisitos Funcionais do Usuário. Na prática, tais casos ocorrem comumente no software de aplicação 'real-time' (p.ex., a Entry transmitindo o tique de um relógio em tempo real); são menos comuns no software de aplicação de negócio.

4

A FASE DE MEDIÇÃO

Este capítulo apresenta as regras e o método para a fase de medição do processo de medição COSMIC. O método geral para a medição de um pedaço de software cujos Requisitos Funcionais do Usuário estejam expressos em termos do Modelo Genérico de Software COSMIC é resumido na Figura 4.0 abaixo.

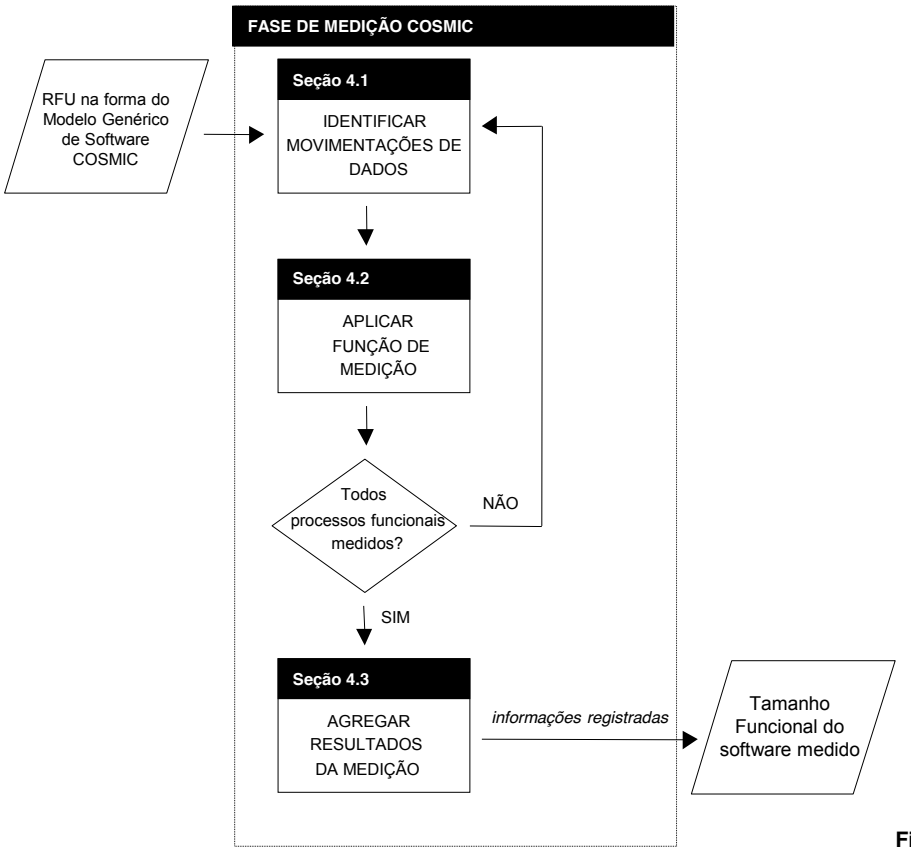


Figura 4.0 – Processo Geral para a Fase de Medição COSMIC

Cada passo neste método é o assunto de uma seção específica neste capítulo, onde as definições e princípios a aplicar são apresentados, juntamente com algumas regras e exemplos.

4.1 Identificando as movimentações de dados

Este passo consiste na identificação das movimentações de dados (Entry, Exit, Read e Write) de cada processo funcional.

4.1.1 Definição dos tipos de movimentações de dados

DEFINIÇÃO – Movimentação de dados
Um componente funcional básico que movimenta um único tipo de grupo de dados. NOTA 1: Há quatro subtipos de tipos de movimentações de dados: Entry, Exit, Read e Write (tipos de). NOTA 2: Para fins de medição, considera-se que cada subtipo de movimentação de dados inclui certas manipulações de dados associadas – ver a seção 4.1.6. NOTA 3: De uma forma mais precisa, trata-se de uma *ocorrência* de uma movimentação de dados, e não de um *tipo* de movimentação de dados, que realmente *movimenta* as *ocorrências* do grupo de dados (não os *tipos*). Este comentário também se aplica às definições de Entry, Exit, Read e Write.

DEFINIÇÃO – Entry (E)
Uma Entry (E) é uma movimentação de dados que movimenta um grupo de dados de um usuário funcional através da fronteira para dentro do processo funcional no qual o mesmo é requerido. NOTA: Considera-se que uma Entry (tipo) inclui certas manipulações de dados associadas (ver seção 4.1.6).

DEFINIÇÃO – Exit (X)
Uma Exit (X) é uma movimentação de dados que movimenta um grupo de dados de um processo funcional através da fronteira na direção do usuário funcional que o requisita. NOTA: Considera-se que uma Exit inclui certas manipulações de dados (ver seção 4.1.6).

DEFINIÇÃO – Read (R)
Uma movimentação de dados que movimenta um grupo de dados do armazenamento persistente para dentro do processo funcional que dele necessita. NOTA: Considera-se que um Read (tipo) inclui certas manipulações de dados associadas (ver seção 4.1.6).

DEFINIÇÃO – Write (W)
Uma movimentação de dados que movimenta um grupo de dados do interior de um processo funcional para o armazenamento persistente. NOTA: Considera-se que um Write (tipo) inclui certas manipulações de dados associadas (ver seção 4.1.6).

A Figura 4.1.1. abaixo ilustra o relacionamento global entre os quatro tipos de movimentações de dados, o processo funcional ao qual pertencem e a fronteira do software medido.

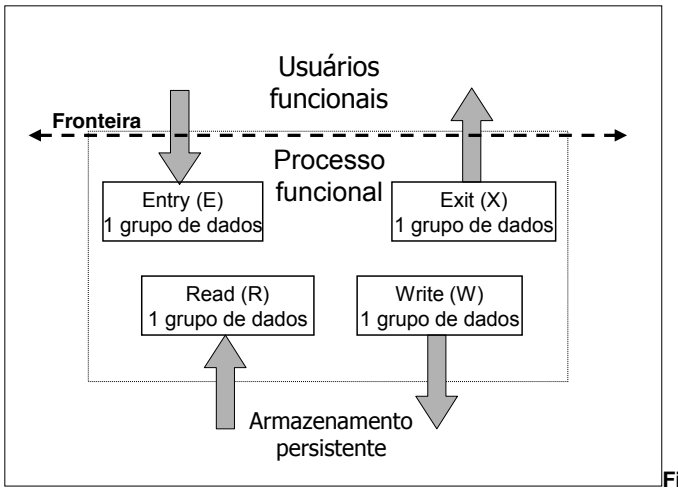


Figura 4.1.1 – Os quatro tipos de movimentações de dados e seu relacionamento com o processo funcional e grupos de dados

4.1.2 Identificando Entries (E)

Uma vez identificada, uma movimentação de dados Entry candidata deve estar em conformidade com os seguintes princípios:

PRINCÍPIOS – Entry (E)
a) Uma Entry deve movimentar um único grupo de dados descrevendo um único objeto de interesse de um usuário funcional através da fronteira e para dentro do processo funcional do qual a Entry faz parte. Se a entrada para um processo funcional compreender mais de um grupo de dados, identificar uma Entry para cada grupo de dados distinto na entrada. (Ver tambem a seção 4.1.7 sobre 'Unicidade nas movimentações de dados'.)
b) Uma Entry não deve enviar dados para fora da fronteira, ler ou gravar dados.
c) Quando um processo funcional precisar obter dados de um usuário funcional e este último não precisar saber quais dados enviar, ou quando o usuário funcional for incapaz de reagir a uma mensagem a ele destinada, identificar uma Entry para o processo funcional pela obtenção dos dados. Nenhuma mensagem do processo funcional ao usuário funcional procurando recuperar os dados deve ser contada como Exit nesses casos. Contudo, quando um processo funcional precisar obter dados de um usuário funcional e o processo funcional precisar fornecer dados ao usuário funcional, dos quais este último precise para atender ao solicitado, contar uma Exit para a solicitação e uma Entry para o retorno dos dados solicitados (ver seção 4.1.9 adiante).

As seguintes regras ajudam a confirmar a situação de uma movimentação de dados Entry candidata:

REGRAS – Entry (E)
a) O grupo de dados de uma Entry disparadora pode consistir de apenas um atributo de dados que simplesmente informe ao software que 'um evento Y ocorreu'. Muitas vezes, principalmente em software de aplicações de negócio, o grupo de dados da Entry disparadora possui diversos atributos que informam ao software que 'um evento Y ocorreu e aqui estão os dados sobre aquele evento específico'.

b) Tiques de relógio que sejam eventos disparadores devem ser sempre externos ao software sendo medido. Assim, por exemplo, um evento tique de relógio ocorrendo a cada 3 segundos deve estar associado a uma Entry movimentando um grupo de dados com um atributo de dados. Notar que não faz diferença se o evento disparador é gerado periodicamente pelo hardware ou por outro pedaço de software fora da fronteira do software sendo medido.

c) A não ser que um processo funcional específico seja necessário, obter a hora a partir do relógio do sistema não deve ser considerado como causa de uma Entry.

d) Se uma ocorrência de um evento específico disparar a Entry de um grupo de dados compreendendo até 'n' atributos de dados de certo objeto de interesse e o RFU permitir que outras ocorrências do mesmo evento possam disparar uma Entry de um grupo de dados, com valores apenas para um subconjunto dos 'n' atributos do objeto de interesse, então uma Entry deverá ser identificada compreendendo todos os 'n' atributos de dados.

EXEMPLO ilustrando a regra (c): quando um processo funcional grava uma 'time stamp', nenhuma Entry é identificada para obter o valor do relógio do sistema.

Uma vez identificada, cada movimentação de dados Entry pode ser registrada marcando-se a célula correspondente da matriz do Modelo Genérico de Software (apêndice A) com um 'E'.

4.1.3 Identificar Exits (X)

Uma vez identificada, uma movimentação de dados Exit candidata deve estar em conformidade com os seguintes princípios:

PRINCÍPIOS – Exit (X)

a) Uma Exit deve movimentar um único grupo de dados, descrevendo um único objeto de interesse do processo funcional do qual a Exit faz parte, através da fronteira para um usuário funcional. Se a saída de um processo funcional compreender mais de um grupo de dados, identificar uma Exit para cada grupo de dados distinto na saída. (Ver também a seção 4.1.7 sobre 'Unicidade nas movimentações de dados'.)

b) Uma Exit não deve enviar dados para dentro da fronteira, ler ou gravar dados.

As seguintes regras podem ser úteis para confirmar a situação de uma movimentação de dados Exit candidata:

REGRAS – Exit (X)

a) Todas as mensagens geradas e enviadas por software sem dados do usuário (p.ex., mensagens de erro) devem ser consideradas como valores de um atributo de um objeto de interesse (o qual poderia ser denominado 'indicação de erro'). Dessa forma, uma única Exit deve ser identificada para representar todas essas ocorrências de mensagens dentro de cada processo funcional onde as mesmas sejam requeridas pelo RFU.

b) Se uma Exit de um processo funcional movimenta um grupo de dados compreendendo até 'n' atributos de dados de certo objeto de interesse e o RFU permite que o processo funcional possua a ocorrência de uma Exit que movimente um grupo de dados com valores apenas para um subconjunto dos 'n' atributos do objeto de interesse, então uma Exit deverá ser identificada, compreendendo todos os 'n' atributos de dados.

Exemplos da regra (a) acima são apresentados a seguir:

EXEMPLO 1: Em um diálogo humano-computador, alguns exemplos de mensagens de erro ocorrendo durante a validação dos dados submetidos poderiam ser: 'erro de formato', 'cliente não encontrado', 'erro: caixa de verificação indicando que os termos e condições foram lidos', 'limite de crédito excedido', etc. Todas essas mensagens de erro devem ser consideradas

ocorrências de uma Exit em cada processo funcional onde tais mensagens ocorram (que poderia ser denominada 'mensagens de erro').

EXEMPLO 2: Mensagens de erro enviadas aos usuários humanos e não geradas pelo software aplicativo devem ser completamente ignoradas na medição da aplicação. Um exemplo de tal tipo de mensagem enviada pelo sistema operacional poderia ser 'A impressora X não está respondendo'.

EXEMPLO 3: Em um diálogo humano-computador, se uma mensagem é enviada em situações de erro, mas contém dados do usuário funcional, então a mesma deveria ser contada como uma Exit no processo funcional onde ocorre. Um exemplo de tal mensagem poderia ser 'Aviso: a quantia que você quer sacar excede o seu limite em $100' (onde $100 é uma variável calculada). Neste exemplo, a Exit contém um grupo de dados relacionado à conta bancária do cliente.

EXEMPLO 4: Em um sistema 'real-time', um processo funcional que verifica periodicamente o funcionamento correto de todos os dispositivos de hardware poderia enviar uma mensagem do tipo 'O sensor X falhou', onde X é uma variável. Tal mensagem deveria ser identificada como uma Exit naquele processo funcional (independentemente do valor de 'X').

EXEMPLO 5: Consideremos os processos funcionais A e B. 'A' pode enviar potencialmente 2 mensagens de confirmação distintas e 5 mensagens de erro aos seus usuários funcionais, enquanto 'B' pode enviar potencialmente 8 mensagens de erro aos seus usuários funcionais. Neste exemplo, uma Exit deve ser identificada no processo funcional 'A' (lidando com 5+2=7 mensagens) e uma Exit distinta deve ser identificada no processo funcional 'B' (lidando com 8 mensagens).

Uma vez identificada, cada movimentação de dados Exit pode ser registrada marcando-se a correspondente célula da matriz do Modelo Genérico de Software (apêndice A) com um 'X'.

4.1.4 Identificando Reads (R)

Uma vez identificada, uma movimentação de dados Read candidata deve estar em conformidade com os seguintes princípios:

PRINCÍPIOS – Read (R)
a) Um Read deve movimentar um único grupo de dados descrevendo um único objeto de interesse do armazenamento persistente para um processo funcional do qual o Read faz parte. Se o processo funcional precisar recuperar mais do que um grupo de dados do armazenamento persistente, identificar um Read para cada grupo de dados distinto recuperado. (Ver também seção 4.1.7 sobre 'Unicidade na movimentação de dados'.)
b) Um Read não deve receber ou enviar dados para fora da fronteira, ou gravar dados.
c) Durante um processo funcional, a movimentação ou manipulação de constantes ou variáveis internas ao processo funcional que possam ser alteradas somente por um programador, o cálculo de resultados intermediários, ou de dados armazenados por um processo funcional como resultado apenas da implementação e não dos RFU não devem ser consideradas movimentações de dados Read.
d) Uma movimentação de dados Read sempre inclui qualquer funcionalidade do tipo 'solicitação de leitura' (assim uma movimentação de dados distinta nunca será contada para uma funcionalidade do tipo 'solicitação de leitura'). Ver também a seção 4.1.9.

Uma vez identificada, cada movimentação de dados Read pode ser registrada marcando-se a célula correspondente da matriz do Modelo Genérico de Software (apêndice A) com um 'R'.

4.1.5 Identificando Writes (W)

Uma vez identificada, uma movimentação de dados Write candidata deve estar em conformidade com os seguintes princípios:

PRINCÍPIOS – Write (W)
a) Um Write deve movimentar um único grupo de dados para o armazenamento persistente, descrevendo um único objeto de interesse do processo funcional do

qual o Write faz parte. Se o processo funcional precisar movimentar mais de um grupo de dados para o armazenamento persistente, identificar um Write para cada grupo de dados distinto que for movimentado para o armazenamento persistente. (Ver também a seção 4.1.7 sobre 'Unicidade nas movimentações de dados'.)

b) Um Write não deve receber ou enviar dados através da fronteira, ou ler dados.

c) Um requisito para excluir um grupo de dados do armazenamento persistente deve ser medido como uma única movimentação de dados Write.

d) No decorrer de um processo funcional, movimentações ou manipulações de dados que não persistirem quando o processo funcional estiver concluído, que atualizem variáveis internas ao processo funcional, ou que produzam resultados intermediários em um cálculo não devem ser consideradas movimentações de dados Write.

Uma vez identificada, cada movimentação de dados Write pode ser registrada marcando-se a célula correspondente da matriz do Modelo Genérico de Software (apêndice A) com um 'W'.

4.1.6 Sobre as manipulaçoes de dados associadas a movimentações de dados

Os subprocessos são, conforme definido no princípio (d) do Modelo Genérico de Software (ver seção 1.4), movimentações de dados ou manipulações de dados. Entretanto, de acordo com uma convenção COSMIC vigente (ver princípio (j) do Modelo Genérico de Software), a existência distinta de subprocessos de manipulação de dados não é reconhecida.

DEFINIÇÃO – Manipulação de dados
Qualquer coisa que aconteça aos dados que não seja a movimentação dos mesmos para dentro ou para fora de um processo funcional, ou entre um processo funcional e o armazenamento persistente.

O seguinte princípio determina como o método COSMIC trata a manipulação de dados.

PRINCÍPIO – Manipulações de dados associadas a movimentações de dados
Todas as manipulações de dados em um processo funcional devem estar associadas aos quatro tipos de movimentações de dados (E, X, R e W). Por convenção, assume-se que as movimentações de dados de um processo funcional também representam as manipulações de dados do processo funcional.

A necessidade de definir os tipos de manipulações de dados associados aos tipos de movimentações de dados surge apenas na medição de *mudanças* no software (ver seção 4.4). Uma mudança requerida geralmente afeta tanto os atributos movimentados quanto a manipulação associada a uma movimentação específica, mas *pode* afetar apenas a manipulação dos dados e não a movimentação dos dados. Tal mudança também precisa ser identificada e medida. Dessa forma, quando há um requisito para mudar uma manipulação de dados em um processo funcional, o medidor precisa identificar a movimentação de dados associada à mudança na manipulação de dados.

Abaixo estão orientações gerais para identificar as manipulações de dados representadas por cada uma das movimentações de dados.

Movimentação de dados Entry

Uma Entry inclui todas as manipulações de dados

- para permitir que um grupo de dados seja submetido e validado por um usuário funcional (p.ex., manipulações de formatação e validação)
- mas não inclui manipulações que envolvam uma outra movimentação de dados, nem manipulações depois que o grupo de dados tenha sido submetido e validado.

EXEMPLO: Uma Entry inclui todas as manipulações, formatações e apresentações em uma tela dos dados submetidos, EXCETO quaisquer Reads que sejam requeridos para validar códigos submetidos ou obter as descrições a eles associadas.

Uma movimentação de dados Entry inclui qualquer funcionalidade do tipo 'solicitação de entrada' exceto quando o processo funcional precisar informar ao usuário funcional quais dados enviar (ver seção 4.1.9 para os dados que devem ser enviados e 4.1.10 para tratamento de uma tela de entrada vazia).

Movimentação de dados Exit

Uma Exit inclui todas as manipulações de dados

- para criar os atributos de dados de um grupo de dados a ser enviado à saída e/ou
- tornar possível que o grupo de dados seja enviado à saída (p.ex., manipulações de formatação e apresentação) e direcionado ao usuário funcional de destino
- mas não inclui nenhuma manipulação de dados que envolva outra movimentação de dados.

EXEMPLO: Uma Exit inclui todo o processamento para formatar e preparar para a impressão alguns atributos de dados, incluindo os campos de títulos que podem ser lidos por seres humanos[13], EXCETO quaisquer Reads ou Entries que possam sejam requeridos para suprir os valores ou descrições de alguns dos atributos de dados impressos.

Movimentação de dados Read

Um Read inclui todo o processamento e/ou cálculo necessários para recuperar um grupo de dados do armazenamento persistente, mas não quaisquer manipulações que envolvam outros tipos de movimentações de dados, nem quaisquer manipulações depois que o Read tenha sido concluído com sucesso.

EXEMPLO: Um Read inclui todos os cálculos matemáticos e processamento lógico requeridos para recuperar um grupo de dados do armazenamento persistente, mas não a manipulação desses atributos depois que o grupo de dados tiver sido obtido.

Um Read também inclui sempre qualquer funcionalidade do tipo 'solicitação para ler' (ver seção 4.1.9).

Movimentação de dados Write

Um Write inclui todo o processamento e/ou cálculo para criar um grupo de dados a ser gravado, mas não inclui quaisquer manipulações que envolvam outros tipos de movimentações de dados, nem quaisquer manipulações depois que o Write tenha sido concluído com sucesso.

EXEMPLO: Um Write inclui todos os cálculos matemáticos e processamento lógico requeridos para criar ou atualizar um grupo de dados a ser gravado ou excluído, EXCETO quaisquer Reads ou Entries que possam ser requeridos para fornecer os valores de quaisquer atributos de dados incluídos no grupo a ser gravado ou excluído.

4.1.7 Unicidade nas movimentações de dados e possíveis exceções[14]

O Modelo Genérico de Software assume que, *normalmente*, em qualquer processo funcional, *todos* os dados descrevendo qualquer objeto de interesse requerido pelo mesmo são submetidos através de um tipo de movimentação de dados Entry, e/ou lidos através de um tipo de movimentação de dados Read, e/ou gravados através de um tipo de movimentação de dados Write, e/ou apresentados através de um tipo de movimentação de dados Exit. O modelo assume, ainda, que todas as

[13] Este exemplo aplica-se à medição de software aplicativo para uso de seres humanos, independentemente do domínio. Obviamente não é aplicável à medição de objetos reutilizáveis que suportem a exibição de cabeçalhos individuais de campos em telas de entrada ou saída.

[14] Esta seção tinha o título 'Des-duplicação de movimentações de dados' na versão 2.2 do Manual de Medição. Considerou-se que o termo 'des-duplicação' não ajudava, assim a terminologia foi modificada.

manipulações de dados resultantes de todos os valores possíveis dos atributos de dados de um grupo de dados que é movimentado são associadas a esta única movimentação de dados.

EXEMPLO ilustrando esta última suposição; Consideremos duas ocorrências de um dado processo funcional (tipo). Suponhamos que na primeira ocorrência os valores de alguns atributos a serem movimentados levam a um subprocesso de manipulação de dados (tipo) 'A' e que em outra ocorrência do mesmo processo funcional os valores dos atributos levam a outro subprocesso de manipulação de dados (tipo) 'B'. Em tais circunstâncias, ambos os subprocessos de manipulação de dados 'A' e 'B' deveriam estar associados à mesma movimentação de dados e, consequentemente, apenas uma movimentação de dados deveria ser normalmente identificada e contada no referido processo funcional.

Podem existir, contudo, circunstâncias *excepcionais* nas quais diferentes tipos de grupos de dados descrevendo um dado objeto de interesse possam precisar (no RFU) ser movimentados em uma movimentação de dados do mesmo tipo (E, R, W, X) no mesmo processo funcional. Alternativamente, e mais uma vez excepcionalmente, o mesmo grupo de dados pode precisar ser movimentado no mesmo tipo de movimentação de dados (E, R, W ou X) no mesmo processo funcional, mas com uma manipulação de dados associada diferente.

As seguintes regras e exemplos cobrem a situação normal, exceções possivelmente válidas e exemplos que podem parecer válidos, mas não o são.

REGRA – Unicidade nas movimentações de dados e possíveis exceções
a) A não ser que os Requisitos Funcionais do Usuário especifiquem de outra forma, todos os atributos de dados descrevendo qualquer objeto de interesse que precisem ser submetidos a um processo funcional, bem como todas as manipulações de dados associadas devem ser identificados e contados como uma Entry (tipo). (Nota: Um processo funcional pode, é claro, precisar tratar diversos tipos de Entry, cada uma movimentando um grupo de dados descrevendo um objeto de interesse (tipo) diferente) A mesma regra equivalente aplica-se a qualquer movimentação de dados Read, Write ou Exit em qualquer processo funcional.
b) Pode ser identificada e contada mais que uma movimentação de dados Entry (tipo), cada uma movimentando um grupo de dados descrevendo o mesmo objeto de interesse (tipo) em um dado processo funcional (tipo), se houver um Requisito Funcional do Usuário para essas diversas Entries. Similarmente, pode ser identificada e contada mais que uma Entry (tipo) movimentando o mesmo grupo de dados (tipo) no mesmo processo funcional, cada uma com uma manipulação de dados diferente associada (tipo) se houver um Requisito Funcional do Usuário para essas diversas Entries. Tais RFU podem surgir quando, em um processo funcional, as diversas Entries tiverem origem em diferentes usuários funcionais que submetam diferentes grupos de dados (cada um descrevendo o mesmo objeto de interesse). A mesma regra equivalente aplica-se a qualquer movimentação de dados Read, Write ou Exit em qualquer processo funcional.
c) Ocorrências repetidas de um tipo de movimentação de dados (i.e., cada ocorrência movimentando o mesmo tipo de grupo de dados com a mesma manipulação de dados) não serão identificadas e contadas mais de uma vez em nenhum processo funcional
d) Se as diversas *ocorrências* de um tipo de movimentação de dados em um processo funcional diferirem quanto à respectiva manipulação de dados porque diferentes valores dos atributos do grupo de dado movimentado conduzem a diferentes caminhos de processamento, o tipo de movimentação de dados não deve ser

identificado e contado mais de uma vez naquele processo.

Os seguintes exemplos ilustram as regras acima.

EXEMPLO 1 para a regra (a): Em qualquer processo funcional, pode-se considerar que todos os Reads de dados descrevendo um objeto de interesse específico retornam todos os atributos requeridos para descrever o referido objeto de interesse (i.e., o 'vetor de estado' completo do objeto de interesse). Daí, normalmente apenas um Read de dados sobre qualquer objeto de interesse será funcionalmente necessário e deve assim ser identificado em qualquer processo funcional.

EXEMPLO 2 para as regras (a) e (b): Seguindo o Exemplo 1, como quaisquer dados lidos devem ter sido feitos persistentes por um comando Write, o caso normal (Regra a) é que seja identificado um Write que movimenta um grupo de dados contendo todos os atributos de um objeto de interesse que deve ser persistido em um dado processo funcional. Excepcionalmente, contudo, podem existir RFU requerendo que um único processo funcional grave dois grupos de dados diferentes descrevendo o mesmo objeto de interesse, por exemplo para uso posterior por dois usuários funcionais diferentes em outros processos funcionais. Um exemplo onde a regra (b) se aplica seria onde um único processo funcional A precisa extrair dois subconjuntos de dados dos arquivos de contas correntes de um banco, para uso posterior por dois programas separados. O primeiro subconjunto é 'detalhes de contas viradas' (que inclui o atributo saldo negativo). O segundo subconjunto é 'detalhes de contas de alto valor' (que possui somente o nome e endereço do correntista, com o objetivo de enviar uma mala direta de marketing). O processo funcional A terá dois Writes, um para cada subconjunto.

EXEMPLO 3 para a regra (b): É possível que exista RFU para que um único processo funcional produza duas ou mais Exits movimentando diferentes grupos de dados descrevendo o mesmo objeto de interesse, destinadas a diferentes usuários funcionais. Por exemplo, quando um novo empregado é admitido em uma empresa, é produzido um relatório para que o empregado confirme que seus dados pessoais são válidos e uma mensagem é enviada à Segurança para autorizar a entrada do empregado no prédio.

EXEMPLO 4 para a regra (c): Suponhamos que um Read é requerido no RFU tal que, na prática, muitas ocorrências sejam recuperadas, como no caso de uma busca em um arquivo. Para fins de dimensionamento, identificar apenas um Read.

EXEMPLO 5 para a regra (c): Suponhamos que em um processo funcional 'real time' o RFU requer que dados sejam submetidos por um usuário funcional, p.ex., um dispositivo de hardware, duas vezes em um intervalo fixo de tempo para medir uma taxa de mudança, ou para verificar se um valor foi alterado durante o processo. Desde que as duas Entries sejam idênticas em termos do grupo de dados movimentado e das manipulações de dados associadas, apenas uma Entry deverá ser identificada. (Consulte a seção 4.1.6 para os tipos de manipulação de dados que são considerados associados a uma Entry.)

EXEMPLO 6 para a regra (c): Ver a seção 4.1.2, Regra (d) para Entries, e a seção 4.1.3, Regra (b) para Exits

EXEMPLO 7 para a regra (d): Suponhamos que é requerido um processo funcional que provê várias opções de manipulação de dados, dependendo dos valores dos atributos de dados de uma Entry. Para fins de dimensionamento, identificar apenas uma Entry.

EXEMPLO 8: Suponhamos que um Read de um grupo de dados é requerido no RFU, mas o desenvolvedor decide implementá-lo com dois comandos para recuperar diferentes subconjuntos de atributos de dados do mesmo objeto de interesse do armazenamento persistente, em diferentes pontos do processo funcional. Identificar apenas um Read.

4.1.8 Quando um processo funcional movimenta dados de ou para o armazenamento persistente

Esta seção explica as movimentações de dados envolvidas quando um processo funcional de um pedaço de software aplicativo precisa movimentar dados de ou para o armazenamento persistente, seja o armazenamento local ou remoto. Os exemplos também mostram como as necessidades de armazenamento da aplicação são tratadas pelos outros softwares que suportam a aplicação em outra camada, tais como o software gerenciador de dispositivo de armazenamento persistente.

Os exemplos ilustram a aplicação do princípio (g) do Modelo de Contexto de Software e os princípios do Modelo Genérico de Software. A chave para o entendimento desses exemplos é que esses princípios devem ser aplicados separadamente a cada pedaço de software a ser medido.

O primeiro exemplo lida com as movimentações de dados de uma consulta em uma aplicação onde o requisito de recuperar dados persistentes é tratado por software local gerenciador de dispositivo em outra camada. O segundo exemplo mostra como as movimentações de dados diferem quando o requisito de recuperar é primeiramente satisfeito por um pedaço de software par da aplicação.

Em termos práticos, os exemplos são aplicáveis quando a tarefa é medir dois pedaços de software que possuem um relacionamento hierárquico (i.e., quando os dois pedaços estão em camadas diferentes) ou possuem um relacionamento cliente-servidor (i.e., quando os dois pedaços são pares um do outro). Os exemplos mostram como as movimentações de dados fisicamente trocadas entre dois pedaços de software a serem medidos são modeladas.

Os exemplos são ilustrados utilizando as convenções dos Diagramas de Sequência de Mensagens. A notação utilizada nesses diagramas é apresentada a seguir:

- Uma seta vertical em negrito apontando para baixo representa um processo funcional.
- As setas horizontais representam movimentações de dados, rotuladas como E, X, R ou W significando Entry, Exit, Read e Write, respectivamente. Entries e Reads são mostradas como setas entrando no processo funcional e Exits e Writes como setas saindo; aparecem na sequência requerida do processo funcional, de cima para baixo.
- Uma linha vertical pontilhada representa uma fronteira.

EXEMPLO 1: Quando um processo funcional precisa movimentar dados de ou para o armazenamento persistente local.

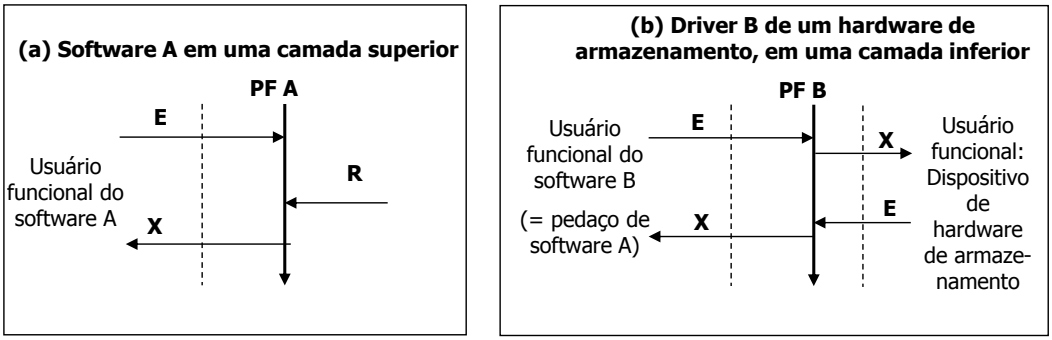
Este exemplo envolve dois pedaços de software, nominalmente um pedaço de software aplicativo 'A' e um pedaço de software distinto 'B' que é o gerenciador de dispositivo para um dispositivo de armazenamento persistente que o software aplicativo utiliza. (Ignoramos a provável presença de um sistema operacional por simplicidade; o sistema operacional transmite solicitações da aplicação para o software gerenciador de dispositivo e retorna os resultados das solicitações.)

O conceito de camadas nos diz que os dois pedaços de software estão em camadas diferentes: a camada de aplicação e a camada de gerenciador de dispositivo. Fisicamente há um relacionamento hierárquico entre os dois pedaços e (ignorando o sistema operacional) uma interface física entre o software nas duas camadas, conforme mostrado, por exemplo, na Fig. 2.2.4.1.

Normalmente, o RFU da aplicação 'A' especificará processos funcionais que incluem a necessidade de Reads e Writes para o armazenamento persistente, mas não como esses Reads e Writes serão tratados por outro software de infraestrutura.

Aplicando-se os modelos COSMIC aos dois pedaços de software, os usuários funcionais do software A na camada de aplicação poderiam ser, por exemplo, usuários humanos, enquanto o usuário funcional do software B na camada de gerenciador é o pedaço de software aplicativo A (ignorando o sistema operacional).

Suponhamos que um processo funcional de consulta 'PF A' do software A na camada de aplicação requer uma movimentação de dados Read. A Fig. 4.1.8.1 (a) mostra o modelo COSMIC da consulta do aplicativo. A recuperação física dos dados requeridos do armazenamento persistente é tratada por um processo funcional 'PF B' do software B na camada de gerenciador de dispositivo. A Figura 4.1.8.1 (b) mostra o modelo para este processo funcional do gerenciador de dispositivo.



Figuras 4.1.8.1 (a) e (b) Solução para um Read emitido pelo software 'A' na camada de aplicação para o software 'B' na camada do gerenciador de dispositivo

A Fig. 4.1.8.1 (a) mostra a consulta disparada por uma Entry, seguida por um Read e então por uma Exit com o resultado da consulta. O PF A não tem conhecimento de onde os dados são recuperados, nem que na prática o Read é delegado ao software de gerenciador de dispositivo.

A Fig. 4.1.8.1 (b) mostra que funcionalmente a solicitação de Read da aplicação A é recebida como uma Entry disparada para o processo funcional FP B, que então recupera os dados solicitados do dispositivo de armazenamento persistente através de um par Exit/Entry e retorna os dados à aplicação como uma Exit. A aplicação A e o dispositivo de hardware de armazenamento persistente são dessa forma usuários funcionais do software gerenciador de dispositivo B.

O aparente descompasso entre o número de movimentações de dados do 'Read' do software na camada de aplicação e o par 'Entry/Exit' do software no gerenciador de dispositivo é devido ao fato de que, por convenção, uma movimentação de dados Read é considerada como incluindo qualquer funcionalidade do tipo 'solicitação de leitura'.

Modelos exatamente análogos seriam aplicáveis se o processo funcional PF A precisasse tornar dados persistentes através de uma movimentação de dados Write. Neste exemplo, a Exit do processo funcional PF B do software gerenciador de dispositivo para a aplicação A conteria qualquer 'código de retorno' ou mensagem de erro.

EXEMPLO 2: Quando um processo funcional precisa obter dados de um pedaço de software par

Neste exemplo, assume-se que os dois pedaços de software pares a serem medidos possuem um relacionamento 'cliente/servidor', i.e., onde um pedaço, o cliente, obtém serviços e/ou dados do outro pedaço, o 'servidor, na mesma camada'. A Fig. 4.1.8.2 mostra um exemplo de tal relacionamento, no qual os dois pedaços são componentes principais (pares) da mesma aplicação. O mesmo relacionamento existiria e o mesmo diagrama seria aplicável se os dois pedaços fossem aplicações distintas, onde uma precisasse obter dados da outra.

Fisicamente, os dois componentes pares poderiam executar em processadores distintos; em tal caso trocariam dados através dos respectivos sistemas operacionais e quaisquer outras camadas intermediárias de seus processadores em uma arquitetura de software como a da Fig. 2.2.4.1. Mas logicamente, aplicando-se os modelos COSMIC, os dois componentes trocam dados através de 'pares Entry/Exit'. O software e o hardware entre os dois são ignorados neste modelo (como também mostrado no lado direito da Fig. 3.1.1).

A Fig. 4.1.8.2 mostra que um processo funcional PF C1 do componente cliente C1 é disparado por uma Entry de seu usuário funcional que compreende, por exemplo, os parâmetros da consulta. O RFU do componente C1 reconhecerá que este componente precisa pedir os dados requeridos ao componente servidor C2, precisando dizer a ele quais dados são necessários.

Para obter os dados requeridos, então, o PF C1 emite uma Exit para o componente C2, contendo os parâmetros da solicitação de consulta. O componente C1 é agora um usuário funcional do componente C2; daí então existe uma fronteira entre os dois componentes. Esta movimentação de dados Exit atravessa a fronteira entre C1 e C2 e assim torna-se a Entry disparadora de um processo funcional PF C2 no componente C2. O processo funcional PF C2 do componente C2 obtém os dados necessários através de um Read e envia os dados de volta a C1 através de uma Exit. O processo funcional PF C1 do componente C1 recebe esta movimentação de dados como uma Entry. O PF C1 então passa os dados adiante como uma Exit para satisfazer a consulta de seu usuário funcional. Esta consulta do Exemplo 2 então requer 7 movimentações de dados para satisfazer a solicitação de consulta na camada de aplicação. Compare-se isto com as 3 movimentações de dados (1 x E, 1 x R e 1 x X) que teriam sido requeridas na camada de aplicação se o componente C1 tivesse sido capaz de recuperar os dados de armazenamento persistente 'local', conforme mostrado na Fig. 4.1.8.1 (a).

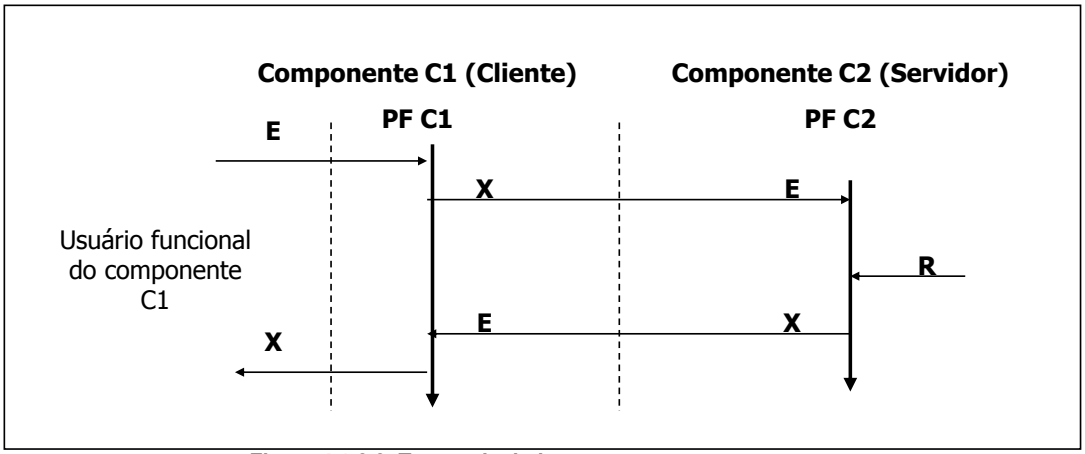


Figura 4.1.8.2 Trocas de dados entre componentes pares

O componente C2 provavelmente utilizará os serviços de algum software gerenciador de dispositivo de armazenamento persistente em uma camada inferior da arquitetura de software para recuperar os dados do hardware, como no Exemplo 1.

Comparando os Exemplos 1 e 2, vemos que no Exemplo 1 os modelos da aplicação A e do gerenciador de dispositivo B não podem ser combinados como no Exemplo 2. Isto acontece porque um Read não atravessa uma fronteira. A Fig. 4.1.8.1 (b) mostra que a aplicação A é um usuário funcional do software gerenciador de dispositivo B. Porém o inverso não é verdadeiro, o que demonstra dessa forma a natureza hierárquica do software em camadas diferentes.

Em contraste, a Fig. 4.1.8.2 pode mostrar os dois componentes em um modelo, porque eles trocam dados como 'pares' ou iguais na mesma camada. O componente C1 é um usuário funcional do componente C2 e vice-versa, e eles compartilham uma fronteira comum.

Note-se que nesses dois exemplos ignoramos, por simplicidade, a geração de uma Exit de mensagem de erro pela aplicação A ou pelo componente C1 (em acréscimo à Exit contendo o resultado da consulta) que poderia resultar em um 'código de retorno' acompanhando a movimentação de dados Read.

4.1.9 Quando um processo funcional requer dados de um usuário funcional

Segundo o princípio (c) para uma Entry (ver seção 4.1.2), se um processo funcional precisar obter dados de um usuário funcional há dois casos. Se o processo funcional não precisar dizer ao usuário funcional quais dados enviar, uma única Entry será suficiente (por objeto de interesse). Se o processo funcional precisar dizer ao usuário funcional quais dados enviar, será necessário um par Exit/Entry. As seguintes regras são aplicáveis:

REGRAS – Quando um processo funcional requer dados de um usuário funcional

a) Um processo funcional deve obter um grupo de dados através uma movimentação de dados Entry de um usuário funcional, quando a Entry *não precisa dizer ao usuário funcional quais dados devem ser enviados*, como em cada um dos seguintes quatro casos:

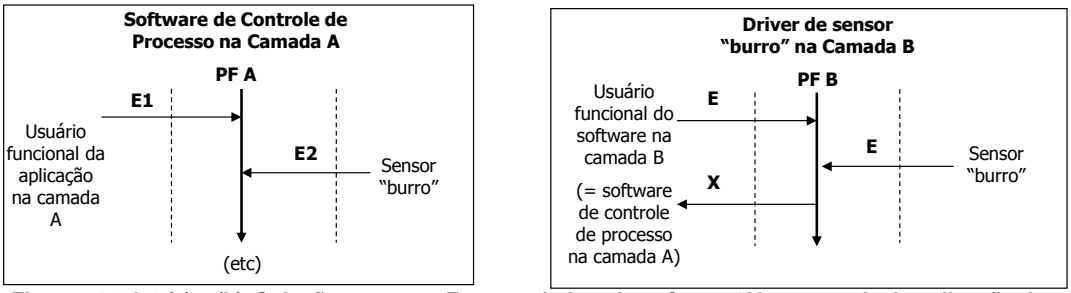
- um usuário funcional envia uma Entry disparadora que inicia o processo funcional;
- um processo funcional, tendo recebido uma Entry disparadora, entra em espera, aguardando a chegada de uma outra Entry enviada pelo usuário funcional (pode ocorrer quando um usuário funcional humano submete dados a um software aplicativo de negócio);
- um processo funcional, tendo iniciado, solicita ao usuário funcional: 'envie-me seus dados agora, se tiver algum' e o usuário funcional envia seus dados;
- um processo funcional, tendo iniciado, inspeciona o estado de um usuário funcional e recupera os dados de que necessita.

Nos últimos dois casos (tipicamente ocorrendo em software 'real-time' fazendo 'polling'), por convenção, nenhuma Exit do processo funcional deverá ser identificada na obtenção dos dados requeridos. O processo funcional precisa apenas enviar uma mensagem de aviso ao usuário funcional e a funcionalidade da mensagem de aviso é considerada parte da Entry. O processo funcional sabe quais dados vai receber. Apenas uma Entry é necessária neste caso.

b) Quando um processo funcional precisa obter os serviços de um usuário funcional (por exemplo, para obter dados) e *o usuário funcional precisa ser informado sobre o que enviar* (normalmente quando o usuário funcional é outro pedaço de software fora do escopo do software sendo medido), um par de movimentações de dados Exit/Entry deve ser identificado. A Exit contém a solicitação dos dados específicos; a Entry contém os dados retornados.

EXEMPLO 1 da regra (a), primeiro e segundo itens: Quando um processo funcional provê uma tela para entrada de dados por um usuário funcional humano, formatada, porém com os campos 'em branco', com a exceção de possíveis valores 'default', como ocorre em uma aplicação de negócio on-line, a provisão da tela 'em branco' não é contada como uma 'Exit'. Somente a tela preenchida é contada como uma ou mais Entries. (Ver também seção 4.1.10.)

EXEMPLO 2 da regra (a), terceiro ou quarto itens: Suponhamos que um processo funcional de um sistema de software de controle de processo em 'real-time' precisa investigar um 'array' de sensores 'burros' idênticos. O processo funcional obtém seus dados através de uma Entry (tipo). (Como os sensores são idênticos, apenas uma Entry (tipo) é identificada e contada, embora existam diversas ocorrências.) Suponhamos ainda que a Entry precise ser passada a um pedaço de software gerenciador de dispositivo em uma camada mais baixa da arquitetura de software, que obtém fisicamente os dados requeridos do sensor, conforme ilustrado na Fig. 2.2.3.2. Os processos funcionais do software de controle de processo e do software gerenciador de dispositivo para os sensores 'burros' seriam conforme mostrado nas Figs. 4.1.9.1 (a) e (b) abaixo.



Figuras 4.1.9.1 (a) e (b) Solução para uma Entry enviada pelo software 'A' na camada da aplicação de controle de processo para o software 'B' na camada do gerenciador de dispositivo do sensor 'burro'

A Fig. 4.1.9.1 (a) mostra que o processo funcional 'PF A' do software de controle de processo é disparado por uma Entry 'E'' de um tique de relógio. O processo funcional então obtém a Entry 'E2' do 'array' de sensores 'burros' para receber as diversas ocorrências de leituras dos sensores. Os sensores 'burros' também são usuários funcionais do software de controle de processo.

A Fig. 4.1.9.1 (b) mostra que o software que gerencia os sensores 'burros' recebe uma Entry (provavelmente através de um sistema operacional) como disparadora de um processo funcional 'PF B'. Este processo funcional obtém uma Entry de seu usuário funcional (tipo), o sensor 'burro' (tipo), para obter os dados dos sensores que são passados de volta para o software de controle de processo como uma Exit. O processo funcional do software de controle de processo continua então o processamento dos dados dos sensores. Novamente, o fato de que há diversas ocorrências deste ciclo de coleta de dados de cada um dos sensores idênticos é irrelevante para o modelo.

O aparente descompasso entre a Entry do software de controle de processo e o par Entry/Exit do software gerenciador de dispositivo é devido à convenção de que uma Entry inclui qualquer funcionalidade de 'solicitação de submissão de dados' neste caso em que o usuário funcional não tem a capacidade de responder a uma mensagem de um processo funcional.

EXEMPLO 3 da regra (b): Suponhamos que um processo funcional envia a um de seus usuários funcionais, tais como um dispositivo de hardware 'inteligente', ou outro pedaço de software par, alguns parâmetros para uma consulta, os parâmetros para um cálculo, ou alguns dados a serem comprimidos. A resposta do usuário funcional é obtida através de um par Exit/Entry, conforme descrito na seção 4.1.8, Exemplo 2.

4.1.10 Comandos de controle

Um 'comando de controle' é uma categoria especial de movimentação de dados que é reconhecida apenas no domínio de aplicação de negócio e que deve ser ignorada na medição de um tamanho funcional. A definição é:

DEFINIÇÃO – Comando de controle
Um comando de controle é um comando que permite a um usuário funcional controlar seu uso do software, mas que não envolve qualquer movimentação de dados sobre um objeto de interesse.

NOTA: O termo 'comando de controle' é utilizado SOMENTE no contexto da medição de software de aplicação de negócio. Neste contexto, um comando de controle não é uma movimentação de dados, porque o comando não movimenta dados sobre um objeto de interesse. Exemplos: comandos de paginação; pressionar uma tecla Tab ou Enter; clicar em 'OK' para confirmar uma ação anterior, etc.

REGRA – Comandos de controle no domínio de aplicações de negócio
No domínio de aplicações de negócio 'comandos de controle' devem ser ignorados, pois os mesmos não envolvem movimentação de dados sobre um objeto de interesse.

EXEMPLOS: Comandos de controle no domínio de aplicações de negócio são as funções que permitem a um usuário funcional controlar a exibição (ou não) de um cabeçalho de subtotais que tenham sido calculados, navegar para cima e para baixo e entre telas físicas, clicar 'OK' para reconhecer uma mensagem de erro, confirmar dados submetidos, etc. Dessa forma, comandos de controle também incluem comandos de menu que permitem ao usuário funcional navegar para um ou mais processos funcionais específicos, mas que não iniciam nenhum processo funcional por si mesmos, assim como comandos que apresentam uma tela vazia para a entrada de dados.

Nota – Fora do domínio de aplicações de negócio, o conceito de 'comando de controle' não possui significado específico e qualquer movimentação de dados ou sinal vindo de um usuário funcional sobre um objeto de interesse deve ser medido.

4.2 Aplicando a função de medição

Este passo consiste em aplicar a função de medição COSMIC a cada uma das movimentações de dados identificadas em cada processo funcional.

DEFINIÇÃO – Função de medição COSMIC
A função de medição COSMIC é uma função matemática que atribui um valor ao seu argumento com base no padrão de medição COSMIC. O argumento da função de medição COSMIC é a movimentação de dados.

DEFINIÇÃO – Padrão de medição COSMIC
O padrão de medição COSMIC, 1 PFC (Ponto de Função COSMIC) é definido como o tamanho de uma movimentação de dados.

De acordo com esta função de medição, cada instância de uma movimentação de dados (Entry, Exit, Read ou Write) que precisa ser incluída, alterada ou excluída e que tiver sido identificada de acordo com a seção 4.1 recebe um tamanho numérico de 1 PFC.

4.3 Agregando os resultados da medição

Este passo consiste em agregar os resultados da função de medição em um único valor do tamanho funcional, aplicando a função de medição a todas as movimentações de dados. Este passo é realizado de acordo com as seguintes regras.

4.3.1 Regras gerais de agregação

REGRAS – Agregando os resultados da medição
a) Para qualquer processo funcional, os tamanhos funcionais das movimentações de

dados individuais devem ser agregados em um único valor de tamanho funcional em unidades de PFC, somando-os aritmeticamente:

Tamanho (processo funcional$_i$) = Σ tamanho(Entries$_i$) + Σ tamanho(Exits$_i$) + Σ tamanho(Reads$_i$) + Σ tamanho(Writes$_i$)

b) Para qualquer processo funcional, o tamanho funcional das alterações em seus Requisitos Funcionais do Usuário deve ser agregado a partir dos tamanhos das movimentações de dados que tiverem sido incluídas, alteradas ou excluídas no processo funcional, produzindo como resultado o tamanho da mudança em unidades de PFC, de acordo com a seguinte fórmula:

Tamanho (Mudança(processo funcional$_i$)) = Σ tamanho (movimentações de dados incluídas $_i$) +

Σ tamanho (movimentações de dados alteradas $_i$) +

Σ tamanho (movimentações de dados excluídas $_i$)

Para mais informações sobre a agregação do tamanho funcional, ver seção 4.3.2. Para a medição de software alterado, ver seção 4.4.

c) O tamanho de um pedaço de software em um escopo definido deve ser obtido pela agregação dos tamanhos dos processos funcionais referentes ao pedaço de software, conforme as regras (e) e (f) abaixo

d) O tamanho de qualquer mudança em um pedaço de software em um escopo definido deve ser obtido agregando-se os tamanhos de todas as mudanças em todos os processos funcionais correspondentes ao pedaço, de acordo com as regras (e) e (f) abaixo.

e) Os tamanhos de pedaços de software ou das mudanças em pedaços de software dentro de camadas podem ser somados somente se forem medidos no mesmo nível de granularidade de processo funcional de seu RFU.

f) Os tamanhos de pedaços de software e/ou de mudanças em tamanhos de pedaços de software dentro de qualquer camada, ou de diferentes camadas devem ser somados somente se isso fizer sentido para o propósito da medição.

g) O tamanho de um pedaço de software não pode ser obtido pela soma dos tamanhos de seus componentes (independentemente de como o item for decomposto) *a não ser que* as contribuições de tamanho das movimentações de dados entre componentes sejam eliminadas.

h) Se o método COSMIC for ampliado localmente (por exemplo, para medir algum aspecto do tamanho não coberto pelo método padrão), então o tamanho medido através da extensão local deverá ser reportado separadamente conforme descrito na seção 5.1 e NÃO poderá ser somado ao tamanho obtido pelo método padrão medido em PFC (ver mais na seção 4.5)

EXEMPLO 1 para as regras (b) e (c): Uma alteração solicitada em um pedaço de software pode ser: 'acrescente um novo processo funcional de tamanho 6 PFC, e em outro processo funcional acrescente uma movimentação de dados, faça modificações em três outras movimentações de dados e exclua duas movimentações de dados'. O tamanho total da mudança solicitada é 6 + 1 + 3 + 2 = 12 PFC.

EXEMPLO 2 para a regra (f): Se várias partes principais de um pedaço de software forem desenvolvidas com a utilização de diversas tecnologias, por diferentes subequipes de projeto, pode não haver utilidade prática na soma de seus tamanhos.

EXEMPLO 3 para a regra (g): Se um pedaço de software

- for primeiro medido 'como um todo', i.e., todo dentro de um escopo
- tiver, em segundo lugar, o tamanho de seus componentes medido separadamente, i.e., cada um com seu próprio escopo,

então o tamanho total obtido pela soma de todos os componentes separadamente (no segundo caso) excederá o tamanho 'medido como um todo' (no primeiro caso) devido à contribuição para o tamanho de todas as movimentações de dados entre componentes. Tais movimentações de dados entre componentes não são visíveis quando o pedaço é medido 'como um todo' e devem ser eliminadas para obter o tamanho 'do todo'. Ver também o exemplo na seção sobre a medição em vários níveis de granularidade em arquiteturas de software puras, no documento 'Tópicos Avançados e Relacionados'.

Deve-se notar que, *dentro* de cada camada identificada, a função de agregação é completamente escalável. Dessa forma um subtotal pode ser gerado para processos funcionais individuais ou para todo o software dentro de uma camada, dependendo do propósito e do escopo de cada exercício de medição e sujeito às regras (d), (e) e (f) acima.

4.3.2 Mais sobre a agregação do tamanho funcional

Em um contexto onde o tamanho funcional deva ser usado como uma variável em um modelo, p. ex. para estimar o esforço, e o software a ser medido possua mais de uma camada ou componente par, a agregação será tipicamente executada por camada ou por componente par, já que os mesmos são frequentemente implementados com tecnologias diferentes.

EXEMPLO 1: Consideremos software onde a camada de aplicação será implementada utilizando uma 3GL e um conjunto de bibliotecas existentes, enquanto uma camada de 'driver' será implementada com linguagem 'assembly'. O esforço unitário associado à construção de cada camada será provavelmente diferente e, consequentemente, uma estimativa de esforço será preparada separadamente para cada camada, com base no respectivo tamanho.

EXEMPLO 2: Se uma equipe de projeto tem que desenvolver uma quantidade de pedaços de software e está interessada em sua produtividade global, a mesma poderá somar as horas de trabalho necessárias ao desenvolvimento de cada pedaço. Similarmente, podem ser somados os tamanhos dos pedaços desenvolvidos se (e somente se) os respectivos tamanhos satisfizerem as regras fornecidas acima.

A razão pela qual os tamanhos dos pedaços de software de diferentes camadas em uma arquitetura em camadas padrão, medidos no mesmo nível de granularidade de processo funcional, podem ser somados é que tal arquitetura possui um conjunto de usuários funcionais coerentemente definidos. Cada camada é um usuário funcional das camadas 'inferiores' que utiliza e qualquer pedaço de software em uma camada pode ser um usuário funcional de qualquer de seus pedaços de software pares. Os requisitos de tal arquitetura impõem que os RFU dos vários pedaços precisam trocar mensagens. É então necessário e razoável que os tamanhos dos vários pedaços sejam somados, sempre de acordo com as regras (d), (e) e (f) acima. Entretanto, em contraste, o tamanho de um pedaço de software pode *não* ser obtido pela soma dos tamanhos de seus objetos componentes reutilizáveis, a não ser que as movimentações de dados entre objetos sejam eliminadas, conforme a regra (f) acima.

A agregação dos resultados da medição por tipo de movimentação de dados pode ser útil para analisar a contribuição de cada tipo para o tamanho total de uma camada e, dessa forma, ajudar a caracterizar a natureza funcional da camada medida.

4.4 Mais sobre a medição do tamanho das mudanças no software

Uma 'mudança funcional' no software existente é interpretada no método COSMIC como 'qualquer combinação de inclusões de novas movimentações de dados, ou de alterações ou exclusões de movimentações de dados existentes'. Os termos 'melhoria' e 'manutenção'[15] são frequentemente utilizados para o que aqui chamamos uma 'mudança funcional'.

A necessidade de uma mudança no software pode ter origem em

[15] Uma convenção normal da medição é que o tamanho funcional de um pedaço de software não muda se o software precisar ser alterado para corrigir um defeito, de maneira a alinhar o software com seu RFU. O tamanho funcional do software não muda se a mudança é para corrigir um defeito no RFU.

- novo RFU (i.e., apenas inclusões na funcionalidade existente), ou
- mudança no RFU (talvez envolvendo inclusões, alterações e exclusões) ou
- necessidade de 'manutenção' para corrigir um defeito

As regras para o dimensionamento de qualquer uma dessas mudanças são as mesmas, mas o medidor é alertado para distinguir as diversas circunstâncias ao realizar medições de desempenho e estimativas.

Quando um pedaço de software é completamente substituído, por exemplo, sendo reescrito, com ou sem ampliar e/ou omitir funcionalidades, o tamanho funcional da mudança é o tamanho do software substituto, medido de acordo com as regras normais para o dimensionamento de novo software. Este caso não será mais considerado nesta seção. O medidor deve estar ciente, contudo, da necessidade de distinguir projetos para desenvolver software completamente novo de projetos para 'redesenvolver' ou 'substituir' software existente, ao realizar medições de performance ou estimativas.

Muitas vezes uma parte obsoleta de uma aplicação é excluída ('desconectada' seria uma descrição melhor) deixando-se o código do programa no lugar e apenas removendo-se o contato com a funcionalidade obsoleta. Quando a funcionalidade da parte obsoleta tem 100 PFC, mas a parte pode ser desconectada alterando-se, digamos, 2 movimentações de dados, 100 e não 2 movimentações de dados devem ser identificadas como o tamanho da mudança funcional. Medimos o tamanho do requisito e não o tamanho implementado.[16]

Notar a diferença entre o tamanho da mudança funcional (discutido aqui) e a mudança no tamanho funcional do software. Geralmente são diferentes. O tamanho do software é considerado na seção 4.4.3.

4.4.1 Modificando funcionalidade

Qualquer movimentação de dados de certo tipo (E, X, R e W) envolve dois tipos de funcionalidade: movimenta um único grupo de dados e tem alguma manipulação de dados associada (para esta última parte, ver seção 4.1.6). Daí, para fins de medição, uma movimentação de dados é considerada como funcionalmente modificada se

- o grupo de dados movimentado e/ou
- sua manipulação de dados associada

são modificados de alguma forma.

Um grupo de dados é modificado se

- novos atributos são acrescentados a este grupo de dados e/ou
- atributos existentes são removidos do grupo de dados e/ou
- um ou mais atributos existentes são alterados, por exemplo, no significado ou no formato (mas não em seus valores)

Uma manipulação de dados é alterada se sua funcionalidade mudou de alguma forma, por exemplo, alterando-se os cálculos, a formatação específica, a apresentação e/ou validação dos dados. 'Apresentação' pode significar, por exemplo, a fonte, cor do fundo, tamanho do campo, número de casas decimais, etc.

[16] Notar que, para fins de estimativa, pode ser aconselhável utilizar uma produtividade diferente para esta parte da mudança funcional, já que desconectar é bem diferente de uma exclusão 'real'. Alternativamente, para fins de estimativa pode ser preferível medir o tamanho que será implementado (2 PFC no exemplo) em vez do tamanho do requisito (100 PFC no exemplo). Se o 'tamanho do projeto' de 2 PFC for medido, isto deve estar claramente documentado e separado da medição do RFU, que requer que a aplicação seja reduzida em tamanho por 100 PFC.

Comandos de controle e dados gerais de aplicação não envolvem movimentações de dados, já que nenhum dado sobre objetos de interesse é movimentado. Dessa forma, modificações em comandos de controle e dados gerais de aplicação não devem ser medidos. Como um exemplo, quando a cor de fundo de todas as telas é alterada, a mudança não deve ser medida. (Ver a seção 4.1.10 para uma explicação sobre comandos de controle e dados geral da aplicação.)

REGRAS – Alterando uma movimentação de dados
a) Se uma movimentação de dados deve ser alterada devido a uma mudança na manipulação de dados a ela associada, e/ou devido a uma mudança no número ou tipo de atributos no grupo de dados movimentado, um PFC alterado deve ser medido, independentemente do número de modificações na movimentação de dados. b) Se um grupo de dados deve ser modificado, as movimentações de dados movimentando o grupo de dados alterado, cuja funcionalidade não for afetada pela alteração no grupo de dados, não devem ser identificadas como movimentações de dados alteradas. NOTA 1: Uma modificação no valor de uma ocorrência, tal como uma alteração em um código membro de um atributo cujos valores são um sistema de codificação não é uma alteração no tipo do atributo. NOTA 2: Uma modificação em qualquer dado, aparecendo nas telas de entrada ou saída, que não seja relacionado a um objeto de interesse para um usuário funcional não deve ser identificada como um PFC alterado (ver seção 3.3.4 para exemplos de tais dados.)

EXEMPLO para as regras (a) e (b): Seja um requisito para incluir ou alterar os atributos de dados de um grupo de dados D_1, tal que após a modificação o mesmo torna-se D_2. No processo funcional 'A' onde esta modificação é requerida, todas as movimentações de dados afetadas pela alteração deveriam ser identificadas e contadas como alteradas. Assim, de acordo com a Regra (a), se o grupo de dados alterado D_2 for tornado persistente e/ou for direcionado à saída no processo funcional A, deve-se identificar uma movimentação de dados Write e/ou Exit respectivamente como modificada. Contudo, é possível que outros processos funcionais leiam (Read) ou submetam (Entry) este mesmo grupo de dados D_2, mas sua funcionalidade não é afetada pela modificação, pois os mesmos não usam os atributos de dados incluídos ou alterados. Esses processos funcionais continuam a processar o grupo de dados movimentado como se ainda fosse D_1. Assim, de acordo com a Regra (b), essas movimentações de dados nos outros processos funcionais não afetadas pela modificação nas movimentações de dados do processo funcional A NÃO devem ser identificadas e contadas como modificadas.

4.4.2 Tamanho do software funcionalmente modificado

Depois da alteração funcional de um pedaço de software, seu novo tamanho total é igual ao tamanho original mais o tamanho funcional de todas as movimentações de dados incluídas, menos o tamanho funcional de todas as movimentações de dados excluídas. As movimentações de dados alteradas não exercem influência sobre o tamanho do pedaço de software, pois existem antes e depois das modificações serem realizadas.

4.5 Ampliando o método de medição COSMIC

4.5.1 Introdução

O método de medição de tamanho funcional COSMIC não pretende medir todos os aspectos do 'tamanho' do software. Assim sendo, o método de medição COSMIC não está projetado atualmente para prover uma maneira padronizada de dar conta do tamanho de certos tipos de Requisitos Funcionais do Usuário, notadamente algoritmos matemáticos complexos ou sequências complexas de regras tais como aquelas encontradas em sistemas especialistas. A influência do número de atributos de dados por movimentação de dados sobre o tamanho do software também não é capturada por este método de medição. A influência dos subprocessos de manipulação de dados

sobre o tamanho é levada em conta através de uma suposição simplificada, que é válida apenas para certos domínios de software, conforme definido na seção 1.1.1 sobre a aplicabilidade do método.

Outros parâmetros tais como 'complexidade' (seja lá como for definida) poderiam ser considerados contribuintes do tamanho funcional. Um debate construtivo sobre este assunto iria exigir primeiro a existência de definições acordadas para os outros elementos, dentro da mal definida noção de 'tamanho', conforme aplicada ao software. Tais definições ainda estão, neste ponto, sujeitas a mais pesquisas e a muito debate.

Não obstante, a medida de tamanho COSMIC é considerada uma boa aproximação para o propósito declarado do método e seu domínio de aplicabilidade. Ainda assim, pode ser que no ambiente local de uma organização utilizando o método de medição COSMIC seja desejável dar conta de tal funcionalidade de uma maneira significativa sob a forma de um padrão local. Por este motivo, o método de medição COSMIC possui uma provisão para extensões locais. Quando tais extensões forem utilizadas, os resultados da medição deverão ser reportados de acordo com a convenção especial apresentada na seção 5.1.

As seções seguintes mostram como ampliar o método com um padrão local.

4.5.2 Extensão local com algoritmos complexos

Se for julgado necessário dar conta de algoritmos complexos, um padrão local pode ser arranjado para esta funcionalidade excepcional. Em qualquer processo funcional onde houver um subprocesso funcional de manipulação de dados anormalmente complexo, o medidor ficará livre para atribuir seus Pontos de Função determinados localmente.

EXEMPLO: Uma extensão local padrão poderia ser: "Em nossa organização, um PF Local é atribuído a algoritmos matemáticos tais como (lista de exemplos localmente significativos e bem compreendidos). Dois PF Locais são atribuídos a (outra lista de exemplos), etc."

4.5.3 Extensão local com subunidades de medição

Quando for requerida maior precisão na medição de movimentações de dados, uma subunidade da medida pode ser definida. Por exemplo, um metro pode ser subdividido em 100 centímetros ou em 1000 milímetros. Por analogia, o movimento de um único atributo de dados poderia ser utilizado como uma subunidade de medida. Medições em uma pequena amostra de software nos testes de campo do COSMIC indicaram que na amostra medida o número médio de atributos por movimentação de dados não variou muito nos quatro tipos de movimentação de dados. Por este motivo e para maior facilidade na medição, a unidade COSMIC de medição, 1 PFC, foi fixada no nível de uma movimentação de dados. Contudo, é necessário cuidado ao comparar os tamanhos medidos em PFC de dois diferentes pedaços de software, se o número médio de atributos de dados por movimentação de dados diferir muito entre os dois pedaços de software.

Qualquer um que deseje refinar o método COSMIC introduzindo uma subunidade de medida está livre para fazê-lo, mas deve deixar claro que as medidas de tamanho resultantes não estarão expressas em Pontos de Função COSMIC padrão.

5

REPORTANDO A MEDIÇÃO

O Modelo Genérico de Software pode ser apresentado em forma de matriz, onde as linhas representam processos funcionais (os quais podem ser agrupados por camadas), as colunas representam grupos de dados e as células contêm os subprocessos identificados (Entry, Exit, Read e Write). Esta representação do Modelo Genérico de Software é apresentada no apêndice A.

Os resultados de medições COSMIC devem ser reportados e arquivados de acordo com as convenções seguintes.

5.1 Rotulando

Ao reportar um tamanho funcional COSMIC, o mesmo deve ser rotulado de acordo com a seguinte convenção, em conformidade com o padrão ISO/IEC 14143-1: 2007:

REGRA – Rótulos das medições COSMIC
Um resultado de medição COSMIC deve ser designado como "***x*** CFP (v.***y***)", onde: • "**x**" representa o valor numérico do tamanho funcional, • "v.**y**" representa a identificação da versão do padrão COSMIC utilizado para obter o valor numérico do tamanho funcional "x". NOTA: Se um método de aproximação local foi utilizado para obter a medição, mas no demais a medição foi realizada utilizando as convenções de uma versão padrão COSMIC, a convenção acima deverá ser usada, mas a utilização do método de aproximação deverá ser anotada em outro lugar – ver seção 5.2.

EXEMPLO: Um resultado obtido utilizando-se as regras deste Manual de Medição é designado como '**x** CFP (v3.0.1)'

Quando forem utilizadas extensões locais, conforme definido na seção 4.5 acima, o resultado da medição deverá ser reportado conforme definido abaixo.

REGRA – Rótulos em extensões locais COSMIC
Um resultado de medição COSMIC utilizando extensões locais deve ser designado como: "***x*** CFP (v. ***y***) + z PF Local", onde: • "**x**" representa o valor numérico obtido agregando-se todos os resultados individuais de medição de acordo com o método padrão COSMIC, versão v.y, • "v.**y**" representa a identificação da versão padrão do método COSMIC utilizada para obter o valor numérico do tamanho funcional "x". • "z" representa o valor numérico obtido agregando-se todos os resultados individuais de medição obtidos através de extensões locais para o método COSMIC.

5.2 Arquivando os resultados de medição COSMIC

Ao arquivar resultados de medições COSMIC, as seguintes informações devem ser mantidas para garantir que o resultado sempre seja interpretável:

REGRA– Reporte de medições COSMIC

Além das próprias medições, registradas conforme em 5.1, os seguintes atributos de cada medição deveriam ser registrados.

a) Identificação do componente de software medido (nome, ID da versão ou ID da configuração).

b) As fontes de informação utilizadas para identificar os RFU usados na medição

c) O domínio do software

d) Uma declaração do propósito da medição

e) Uma descrição do escopo da medição, bem como de sua relação com o escopo global de um conjunto de medições relacionadas, se existir. (Utilizar as categorias de escopo genéricas na seção 2.2)

f) Os usuários funcionais do software

g) O nível de granularidade do RFU e o nível de decomposição do software.

h) O ponto no ciclo de vida do projeto onde a medição foi feita (especialmente se a medição for uma estimativa baseada em RFU incompletos, ou se foi feita com base em funcionalidade realmente entregue).

i) O alvo ou margem de erro imaginada para a medição.

j) Uma indicação se o método padrão de medição COSMIC foi utilizado e/ou uma aproximação local para o método padrão, e/ou se extensões locais foram utilizadas (ver seção 4.5). Utilizar as convenções para rótulos das seções 5.1 ou 5.2.

k) Uma indicação se a medição é de funcionalidade desenvolvida ou entregue (funcionalidade 'desenvolvida' é obtida criando-se novo software; funcionalidade 'entregue' inclui funcionalidade 'desenvolvida' e também inclui funcionalidade obtida por outros meios além da criação de novo software, i.e., incluindo-se todas as formas de reuso de software existente, utilização de parâmetros existentes para incluir ou alterar funcionalidades, etc.).

l) Uma indicação se a medição é de funcionalidade nova ou é o resultado de uma atividade de 'melhoria' (i.e., a soma de funcionalidade incluída, alterada e excluída – ver 4.4).

m) A descrição de uma arquitetura de camadas nas quais a medição é feita, se aplicável.

n) O número de componentes principais, se aplicável, cujos tamanhos foram somados para obter o tamanho total registrado.

o) Uma matriz de medição para cada escopo dentro do escopo global da medição, conforme especificado no apêndice A.

p) O nome do medidor e quaisquer qualificações referentes à certificação COSMIC.

APÊNDICE A - DOCUMENTANDO UMA MEDIÇÃO DE TAMANHO COSMIC

A estrutura abaixo pode ser utilizada como repositório para guardar os resultados de uma medição para cada componente identificado de um escopo global que tenha sido mapeado para o Modelo Genérico de Software. Cada escopo dentro do escopo de medição total terá a sua própria matriz.

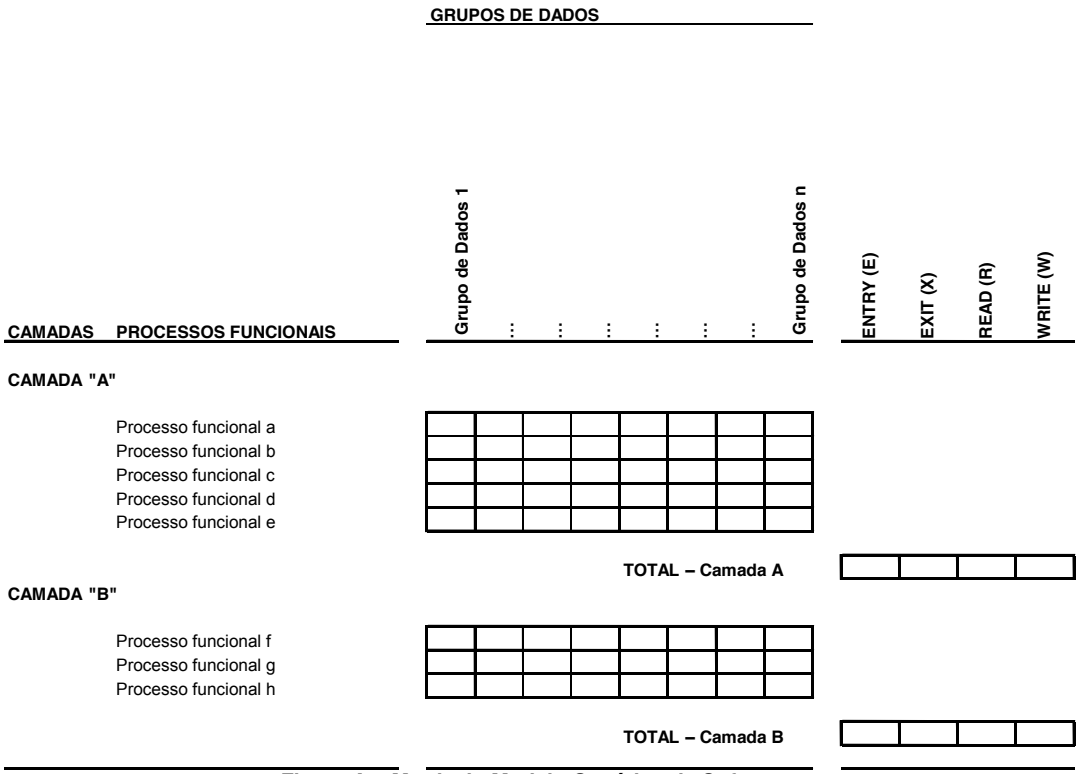


Figura A – Matriz do Modelo Genérico de Software

FASE DE MAPEAMENTO

- Cada grupo de dados identificado é registrado em uma coluna
- Cada processo funcional é registrado em uma linha específica, agrupado por componente identificado.

FASE DE MEDIÇÃO

- Para cada processo funcional identificado, as movimentações de dados identificadas são anotadas na célula correspondente, utilizando-se a seguinte convenção: “E” para Entry, “X” para Exit, “R” para Read e “W” para Write.
- Para cada processo funcional identificado, as movimentações de dados são então somadas por tipo e cada total é registrado na coluna apropriada do lado direito, ao final da matriz.
- O resumo da medição pode ser então calculado e registrado nas células/caixas correspondentes a cada componente, na linha “TOTAL”.

APÊNDICE B – RESUMO DOS PRINCÍPIOS DO MÉTODO COSMIC

Para fins de referência precisa, a tabela abaixo identifica cada princípio encontrado no Método de Medição COSMIC.

ID	DESCRIÇÃO DO PRINCÍPIO
P-01	**O Modelo de Contexto de Software COSMIC** a) O Software é delimitado pelo hardware b) O software é normalmente estruturado em **camadas** c) Uma camada pode conter um ou mais pedaços de software '**pares**' distintos e qualquer pedaço de software pode ser composto de componentes pares distintos d) Qualquer pedaço de software a ser medido deverá ser definido por seu **escopo** de medição, o qual deve estar integralmente contido em uma única camada e) O escopo de um pedaço de software a ser medido depende do **propósito** da medição f) Os **usuários funcionais** de um pedaço de software devem ser identificados a partir dos requisitos funcionais do usuário do pedaço de software a ser medido, como fontes e/ou destinos pretendidos para os dados g) Um pedaço de software interage com os seus usuários funcionais por meio de **movimentações de dados** através de uma **fronteira**, e o pedaço de software pode mover dados de e para o **armazenamento persistente** dentro da fronteira h) Os RFU do software podem ser expressos a diferentes **níveis de granularidade** i) O nível de granularidade ao qual as medições devem ser normalmente efetuadas é o dos **processos funcionais** j) Se não for possível medir ao nível de granularidade dos processos funcionais, nesse caso os RFU do software devem ser medidos através de uma abordagem de aproximação e escalonados para o nível de granularidade dos processos funcionais
P-02	**O Modelo Genérico de Software COSMIC** a) O software recebe dados de **entrada** dos seus usuários funcionais e produz **saídas** e/ou outros resultados para os usuários funcionais b) Os requisitos dos usuários funcionais de um pedaço de software a ser medido podem ser mapeados para processos funcionais únicos c) Cada processo funcional consiste de **subprocessos** d) Um subprocesso pode ser uma movimentação de dados ou uma **manipulação de dados** e) Cada processo funcional é disparado por um movimento de dados do tipo **Entry** proveniente de um usuário funcional, o qual informa ao processo funcional que o usuário funcional identificou um **evento** f) Uma movimentação de dados movimenta um único **grupo de dados** g) Um grupo de dados consiste de um único conjunto de **atributos de dados** que descreve um único objeto de interesse h) Há quatro tipos de movimentação de dados. Uma **Entry** movimenta um grupo de dados para dentro do software, a partir de um usuário funcional. Uma **Exit** movimenta um grupo de dados para fora do software, em direção a um usuário funcional. Um **Write** movimenta um grupo de dados do software para o

ID	DESCRIÇÃO DO PRINCÍPIO
	armazenamento persistente. Um **Read** movimenta um grupo de dados do armazenamento persistente para o software i) Um processo funcional deve incluir pelo menos uma movimentação de dados Entry e uma movimentação de dados Write ou Exit, ou seja, deve incluir um mínimo de duas movimentações de dados j) Como uma aproximação para fins de medição, os subprocessos de manipulação de dados não são medidos separadamente; assume-se que a funcionalidade de qualquer manipulação de dados será considerada na movimentação de dados com a qual a mesma esteja associada
P-03	**O princípio de medição COSMIC** O tamanho funcional de um pedaço de software é diretamente proporcional ao número de suas movimentações de dados.
P-04	**Camada** a) O software em uma camada troca dados com o software em outra camada através dos seus respectivos processos funcionais. b) A 'dependência hierárquica' entre camadas é tal que o software em qualquer camada pode utilizar os serviços funcionais de qualquer software em qualquer camada abaixo dele na hierarquia. Onde há tais relações de uso, denominamos a camada que utiliza o software abaixo como 'superior' e qualquer camada contendo software utilizado como sua 'subordinada'. O software na camada superior apoia-se nos serviços de software dessas camadas subordinadas para rodar adequadamente; estas últimas apoiam-se por sua vez em suas camadas subordinadas para rodar adequadamente, e assim em toda a hierarquia. Por outro lado, o software em uma camada subordinada, juntamente com o software em quaisquer camadas subordinadas da qual ele depende, pode rodar sem precisar dos serviços de nenhuma camada superior na hierarquia. c) O software em uma camada não utiliza, necessariamente, todos os serviços funcionais providos pelo software em uma camada subordinada. d) Os dados que são trocados entre pedaços de software em quaisquer duas camadas são definidos e interpretados diferentemente nos respectivos RFU dos dois pedaços de software, isto é, os dois pedaços de software reconhecem diferentes atributos de dados e/ou subgrupos de dados sendo trocados. Contudo, devem existir também um ou mais atributos de dados ou subgrupos definidos em comum, para permitir que o software na camada recebedora interprete os dados passados pelo software na camada remetente, de acordo com as necessidades do software recebedor.
P-05	**Componente par** a) Em um conjunto de componentes pares de um pedaço de software em uma camada não há dependência hierárquica entre os componentes pares da forma que há entre camadas. Os RFU de todos os componentes pares de um pedaço de software em uma camada estão no mesmo 'nível' na hierarquia de camadas. b) Todos os componentes pares de um pedaço de software devem cooperar para que o pedaço de software execute com sucesso. c) Um grupo de dados pode ser diretamente trocado entre dois componentes pares de um pedaço de software, por um processo funcional de um primeiro componente emitindo uma Exit recebida como uma Entry por um processo funcional do segundo componente. Alternativamente, a troca pode acontecer de forma indireta, com um processo funcional de um primeiro componente tornando um grupo de dados persistente através de um Write, que pode ser recuperado em seguida por um Read de um processo funcional do segundo componente.
P-06	**Aplicando o Modelo Genérico de Software COSMIC** O Modelo Genérico de Software COSMIC deve ser aplicado aos requisitos funcionais do usuário de cada pedaço de software distinto para o qual um escopo de medição distinto

ID	DESCRIÇÃO DO PRINCÍPIO
	tenha sido definido. 'Aplicando o Modelo Genérico de Software COSMIC' diz respeito à identificação do conjunto de eventos disparadores percebidos por cada um dos (tipos de) usuários funcionais identificados nos RFU, seguida pela identificação dos correspondentes processos funcionais, objetos de interesse, grupos de dados e movimentações de dados que devem ser providos como resposta a esses eventos.
P-07	**Grupo de dados** a) Cada grupo de dados identificado deve ser único e distinguível através de sua coleção única de atributos de dados b) Cada grupo de dados deve ser diretamente relacionado a um objeto de interesse nos Requisitos Funcionais do Usuário c) Um grupo de dados deve ser materializado dentro do sistema de computador que suporta o software.
P-08	**Entry (E)** a) Uma Entry deve movimentar um único grupo de dados descrevendo um único objeto de interesse de um usuário funcional através da fronteira e para dentro do processo funcional do qual a Entry faz parte. Se a entrada para um processo funcional compreender mais de um grupo de dados, identificar uma Entry para cada grupo de dados distinto na entrada. (Ver tambem a seção 4.1.7 sobre 'Unicidade nas movimentações de dados'.) b) Uma Entry não deve enviar dados para fora da fronteira, ler ou gravar dados. c) Quando um processo funcional precisar obter dados de um usuário funcional e este último não precisar saber quais dados enviar, ou quando o usuário funcional for incapaz de reagir a uma mensagem a ele destinada, identificar uma Entry para o processo funcional pela obtenção dos dados. Nenhuma mensagem do processo funcional ao usuário funcional procurando recuperar os dados deve ser contada como Exit nesses casos. Contudo, quando um processo funcional precisar obter dados de um usuário funcional e o processo funcional precisar fornecer dados ao usuário funcional, dos quais este último precise para atender ao solicitado, contar uma Exit para a solicitação e uma Entry para o retorno dos dados solicitados (ver seção 4.1.9 adiante).
P-09	**Exit (X)** a) Uma Exit deve movimentar um único grupo de dados descrevendo um único objeto de interesse do processo funcional do qual a Exit faz parte através da fronteira para um usuário funcional. Se a saída de um processo funcional compreender mais de um grupo de dados, identificar uma Exit para cada grupo de dados distinto na saída. (Ver também a seção 4.1.7 sobre 'Unicidade nas movimentações de dados'.) b) Uma Exit não deve enviar dados para dentro da fronteira, ler ou gravar dados.
P-10	**Read (R)** a) Um Read deve movimentar um único grupo de dados descrevendo um único objeto de interesse do armazenamento persistente para um processo funcional do qual o Read faz parte. Se o processo funcional precisar recuperar mais do que um grupo de dados do armazenamento persistente, identificar um Read para cada grupo de dados distinto recuperado. (Ver também seção 4.1.7 sobre 'Unicidade na movimentação de dados'.) b) Um Read não deve receber ou enviar dados para fora da fronteira, ou gravar dados. c) Durante um processo funcional, a movimentação ou manipulação de constantes ou variáveis internas ao processo funcional que possam ser alteradas somente por um programador, o cálculo de resultados intermediários, ou de dados armazenados por um processo funcional como resultado apenas da implementação e não dos RFU não devem ser consideradas movimentações de dados Read.

ID	DESCRIÇÃO DO PRINCÍPIO
	d) Uma movimentação de dados Read sempre inclui qualquer funcionalidade do tipo 'solicitação de leitura' (assim uma movimentação de dados distinta nunca será contada para uma funcionalidade do tipo 'solicitação de leitura'). Ver também a seção 4.1.9.
P-11	**Write (W)** a) Um Write deve movimentar um único grupo de dados para o armazenamento persistente, descrevendo um único objeto de interesse do processo funcional do qual o Write faz parte. Se o processo funcional precisar movimentar mais de um grupo de dados para o armazenamento persistente, identificar um Write para cada grupo de dados distinto que for movimentado para o armazenamento persistente. (Ver também a seção 4.1.7 sobre 'Unicidade nas movimentações de dados'.) b) Um Write não deve receber ou enviar dados através da fronteira, ou ler dados. c) Um requisito para excluir um grupo de dados do armazenamento persistente deve ser medido como uma única movimentação de dados Write. d) No decorrer de um processo funcional, movimentações ou manipulações de dados que não persistirem quando o processo funcional estiver concluído, que atualizem variáveis internas ao processo funcional, ou que produzam resultados intermediários em um cálculo não devem ser consideradas movimentações de dados Write.
P-12	**Manipulações de dados associadas a movimentações de dados** Todas as manipulações de dados em um processo funcional devem estar associadas aos quatro tipos de movimentações de dados (E, X, R e W). Por convenção, assume-se que as movimentações de dados de um processo funcional também representam as manipulações de dados do processo funcional.

APÊNDICE C – RESUMO DAS REGRAS DO MÉTODO COSMIC

A tabela abaixo identifica cada regra encontrada no Método de Medição COSMIC, para fins de referência precisa.

ID	DESCRIÇÃO DA REGRA
R-01	**Escopo** a) O escopo de uma Medição de Tamanho Funcional (MTF) deve ser derivado do propósito da medição. b) O escopo de uma medição qualquer não deve contemplar mais do que uma camada do software a ser medido
R-02	**Camada** a) Se o software for concebido utilizando-se uma arquitetura de camadas estabelecida de acordo com o modelo COSMIC, então aquela arquitetura deveria ser utilizada para identificar as camadas para fins de medição b) No domínio de SIG ou software de negócio, a camada 'topo', i.e., a camada que não é subordinada a nenhuma outra camada, é normalmente referenciada como a camada 'da aplicação'. O software (aplicativo) desta camada apoia-se, em última instância, nos serviços provido pelo software de todas as outras camadas para poder executar adequadamente. No domínio de software 'real-time', o software da 'camada topo' é comumente referenciado como sendo um 'sistema', por exemplo, 'software de sistema de controle de processo', 'software de sistema de controle de vôo', etc. c) Não assumir que qualquer software que tenha evoluído sem a consideração de um projeto arquitetural ou estruturação possa ser particionado em camadas de acordo com o modelo COSMIC.
R-03	**Usuários Funcionais** a) Os usuários funcionais de um pedaço de software a ser medido devem ser derivados do propósito da medição b) Quando o propósito da medição de um pedaço de software estiver relacionado ao esforço para desenvolver ou modificar o pedaço de software, então os usuários funcionais deverão ser aqueles para quem a funcionalidade nova ou modificada deverá ser fornecida.
R-04	**Fronteira** a) Identificar o(s) usuário(s) funcional(is) que interagem com o software sendo medido. A fronteira reside entre os usuários funcionais e o referido software. b) Por definição, existe uma fronteira entre cada par de camadas identificado, onde o software em uma camada é o usuário funcional do software na outra camada, e este último é o software a ser medido. c) Há uma fronteira entre quaisquer dois pedaços de software, incluindo quaisquer dois componentes que sejam pares um do outro; neste caso cada pedaço de software e/ou cada componente pode ser um usuário funcional de seu par.
R-05	**Nível de granularidade do processo funcional** a) A medição de tamanho funcional deve ser feita ao nível de granularidade do processo funcional b) Quando é necessária uma medição de tamanho funcional de alguns RFU que

ID	DESCRIÇÃO DA REGRA
	ainda não evoluíram até o nível no qual todos os processos funcionais foram identificados e todos os detalhes de suas movimentações de dados definidos, as medições devem ser feitas sobre as funcionalidades que foram definidas e, então, escalonadas para o nível de granularidade dos processos funcionais. (Ver o documento 'Tópicos Avançados e Relacionados' para métodos de 'dimensionamento aproximado', i.e., para estimar um tamanho funcional no início do processo de estabelecimento dos RFU.)
R-06	**Processo Funcional** a) Um processo funcional deve ser derivado de pelo menos um Requisito Funcional do Usuário identificável dentro do escopo acordado. b) Um processo funcional (tipo) deve ser executado quando um evento disparador identificável (tipo) ocorre. c) Um evento específico (tipo) pode disparar um ou mais processos funcionais (tipos) que executem em paralelo. Um processo funcional específico (tipo) pode ser disparado por mais de um evento (tipo). d) Um processo funcional deve compreender pelo menos duas movimentações de dados, uma Entry mais (uma Exit ou um Write). e) Um processo funcional deve pertencer completamente ao escopo de medição de um pedaço de software em uma e somente uma camada. f) No contexto de software 'real-time', um processo funcional deve ser considerado concluído quando o mesmo entra em um estado de espera auto-induzido (i.e., o processo funcional já fez tudo o que é requerido como resposta ao evento disparador e espera até receber a próxima Entry disparadora). g) Um processo funcional (tipo) deve ser identificado mesmo quando seus RFU permitirem que o referido processo funcional ocorra com diferentes subconjuntos de seu número máximo de atributos de dados de entrada, e mesmo que tais variações e/ou diferentes valores de dados de entrada possam dar origem a diferentes caminhos de processamento através do processo funcional. h) Eventos distintos (tipos) e consequentemente processos funcionais distintos (tipos) devem ser identificados separadamente nos seguintes casos: • Quando as decisões resultarem em eventos separados e desacoplados no tempo (p.ex., entrar com os dados do pedido hoje e mais tarde confirmar o aceite do pedido, requerendo uma decisão separada, deveria ser considerado como indicação de processos funcionais distintos), • Quando as responsabilidades pelas atividades estão separadas (p.ex., em um sistema de pessoal onde a responsabilidade pela manutenção dos dados básicos de pessoal e dos dados de pagamento estão separadas, indicando processos funcionais distintos; ou no caso de um pacote de software implementado onde há funcionalidades disponíveis para que um administrador de sistema mantenha os parâmetros do pacote, separadas das funcionalidades disponíveis para os usuários funcionais 'normais'.)
R-07	**Entry (E)** a) O grupo de dados de uma Entry disparadora pode consistir de apenas um atributo de dados que simplesmente informe o software que 'um evento Y ocorreu'. Muitas vezes, principalmente em software de aplicações de negócio, o grupo de dados da Entry disparadora possui diversos atributos que informam o software que 'um evento Y ocorreu e aqui estão os dados sobre aquele evento específico'. b) Tiques de relógio que sejam eventos disparadores devem ser sempre externos ao software sendo medido. Assim, por exemplo, um evento tique de relógio ocorrendo a cada 3 segundos deve estar associado a uma Entry movimentando um grupo de dados com um atributo de dados. Notar que não faz diferença se o

ID	DESCRIÇÃO DA REGRA
	evento disparador é gerado periodicamente pelo hardware ou por outro pedaço de software fora da fronteira do software sendo medido. c) A não ser que um processo funcional específico seja necessário, obter a hora a partir do relógio do sistema não deve ser considerado como causa de uma Entry. d) Se uma ocorrência de um evento específico dispara a Entry de um grupo de dados compreendendo até 'n' atributos de dados de certo objeto de interesse e o RFU permitir que outras ocorrências do mesmo evento possam disparar uma Entry de um grupo de dados com valores apenas para um subconjunto dos 'n' atributos do objeto de interesse, então uma Entry deverá ser identificada compreendendo todos os 'n' atributos de dados.
R-08	**Exit (X)** a) Todas as mensagens geradas e enviadas por software sem dados do usuário (p.ex., mensagens de erro) devem ser consideradas como valores de um atributo de um objeto de interesse (o qual poderia ser denominado 'indicação de erro'). Dessa forma, uma única Exit deve ser identificada para representar todas essas ocorrências de mensagens dentro de cada processo funcional onde as mesmas sejam requeridas pelo RFU. b) Se uma Exit de um processo funcional movimenta um grupo de dados compreendendo até 'n' atributos de dados de certo objeto de interesse e o RFU permite que o processo funcional possua uma ocorrência de uma Exit que movimente um grupo de dados com valores apenas para um subconjunto dos 'n' atributos do objeto de interesse, então uma Exit deverá ser identificada, compreendendo todos os 'n' atributos de dados.
R-09	**Unicidade nas movimentações de dados e possíveis exceções** a) A não ser que os Requisitos Funcionais do Usuário especifiquem de outra forma, todos os atributos de dados descrevendo qualquer objeto de interesse que precisem ser submetidos a um processo funcional, bem como todas as manipulações de dados associadas devem ser identificados e contados como uma Entry (tipo). (Nota: Um processo funcional pode, é claro, precisar tratar diversos tipos de Entry, cada uma movimentando um grupo de dados descrevendo um objeto de interesse (tipo) diferente.) A mesma regra equivalente aplica-se a qualquer movimentação de dados Read, Write ou Exit em qualquer processo funcional. b) Pode ser identificada e contada mais que uma movimentação de dados Entry (tipo), cada uma movimentando um grupo de dados descrevendo o mesmo objeto de interesse (tipo) em um dado processo funcional (tipo), se houver um Requisito Funcional do Usuário para essas diversas Entries. Similarmente, pode ser identificada e contada mais que uma Entry (tipo) movimentando o mesmo grupo de dados (tipo) no mesmo processo funcional, cada uma com uma manipulação de dados diferente associada (tipo) se houver um Requisito Funcional do Usuário para essas diversas Entries. Tais RFU podem surgir quando, em um processo funcional, as diversas Entries tiverem origem em diferentes usuários funcionais que submetam diferentes grupos de dados (cada um descrevendo o mesmo objeto de interesse). A mesma regra equivalente aplica-se a qualquer movimentação de dados Read, Write ou Exit em qualquer processo funcional. c) Ocorrências repetidas de um tipo de movimentação de dados (i.e., cada ocorrência movimentando o mesmo tipo de grupo de dados com a mesma

ID	DESCRIÇÃO DA REGRA
	manipulação de dados) não serão identificadas e contadas mais de uma vez em nenhum processo funcional d) Se as diversas *ocorrências* de um tipo de movimentação de dados em um processo funcional diferirem quanto à respectiva manipulação de dados porque diferentes valores dos atributos do grupo de dado movimentado conduzem a diferentes caminhos de processamento, o tipo de movimentação de dados não deve ser identificado e contado mais de uma vez naquele processo.
R-10	**Quando um processo funcional requer dados de um usuário funcional** a) Um processo funcional deve obter um grupo de dados através uma movimentação de dados Entry de um usuário funcional, quando a Entry *não precisa dizer ao usuário funcional quais dados devem ser enviados*, como em cada um dos seguintes quatro casos: • um usuário funcional envia uma Entry disparadora que inicia o processo funcional; • um processo funcional, tendo recebido uma Entry disparadora, entra em espera, aguardando a chegada de uma outra Entry enviada pelo usuário funcional (pode ocorrer quando um usuário funcional humano submete dados a um software aplicativo de negócio); • um processo funcional, tendo iniciado, solicita ao usuário funcional: 'envie-me seus dados agora, se tiver algum' e o usuário funcional envia seus dados; • um processo funcional, tendo iniciado, inspeciona o estado de um usuário funcional e recupera os dados de que necessita. Nos últimos dois casos (tipicamente ocorrendo em software 'real-time' fazendo 'polling'), por convenção nenhuma Exit do processo funcional deve ser identificada na obtenção dos dados requeridos. O processo funcional precisa apenas de enviar uma mensagem de aviso ao usuário funcional e a funcionalidade da mensagem de aviso é considerada parte da Entry. O processo funcional sabe quais dados vai receber. Apenas uma Entry é necessária neste caso. b) Quando um processo funcional precisa obter os serviços de um usuário funcional (por exemplo, para obter dados) e o usuário funcional precisa ser informado sobre o que enviar (normalmente quando o usuário funcional é outro pedaço de software fora do escopo do software sendo medido), um par de movimentações de dados Exit/Entry deve ser identificado. A Exit contém a solicitação dos dados específicos; a Entry contém os dados retornados.
R-11	**Comandos de controle no domínio de aplicações de negócio** No domínio de aplicações de negócio 'comandos de controle' devem ser ignorados, pois os mesmos não envolvem movimentação de dados sobre um objeto de interesse.
R-12	**Agregação de resultados da medição** a) Para qualquer processo funcional, os tamanhos funcionais das movimentações de dados individuais devem ser agregados em um único valor de tamanho funcional em unidades de PFC, somando-os aritmeticamente: Tamanho (processo funcional$_i$) = Σ tamanho(Entries$_i$) + Σ tamanho(Exits$_i$) + Σ tamanho(Reads$_i$) + Σ tamanho(Writes$_i$) b) Para qualquer processo funcional, o tamanho funcional das alterações em seus Requisitos Funcionais do Usuário deve ser agregado a partir dos tamanhos das movimentações de dados que tiverem sido incluídas, alteradas ou excluídas no processo funcional, produzindo como resultado o tamanho da mudança em unidades de PFC, de acordo com a seguinte fórmula: Tamanho (Mudança(processo funcional$_i$)) = Σ tamanho (movimentações de

ID	DESCRIÇÃO DA REGRA
	dados incluídas $_i$) + Σ tamanho (movimentações de dados alteradas $_i$) + Σ tamanho (movimentações de dados excluídas $_i$) Para mais informações sobre a agregação do tamanho funcional, ver seção 4.3.2. Para a medição de software alterado, ver seção 4.4. c) O tamanho de um pedaço de software em um escopo definido deve ser obtido pela agregação dos tamanhos dos processos funcionais referentes ao pedaço de software, conforme as regras (e) e (f) abaixo d) O tamanho de qualquer mudança em um pedaço de software dentro de um escopo definido deve ser obtido agregando-se os tamanhos de todas as mudanças em todos os processos funcionais correspondentes ao pedaço, de acordo com as regras (e) e (f) abaixo. e) Os tamanhos de pedaços de software ou das mudanças em pedaços de software dentro de camadas podem ser somados somente se forem medidos no mesmo nível de granularidade de processo funcional de seu RFU. f) Os tamanhos de pedaços de software e/ou de mudanças em tamanhos de pedaços de software dentro de qualquer camada, ou de diferentes camadas devem ser somados somente se isso fizer sentido para o propósito da medição. g) O tamanho de um pedaço de software não pode ser obtido pela soma dos tamanhos de seus componentes (independentemente de como o item for decomposto) *a não ser que* as contribuições de tamanho das movimentações de dados entre componentes sejam eliminadas. h) Se o método COSMIC for ampliado localmente (por exemplo, para medir algum aspecto do tamanho não coberto pelo método padrão), então o tamanho medido através da extensão local deverá ser reportado separadamente conforme descrito na seção 5.1 e NÃO poderá ser somado ao tamanho obtido pelo método padrão medido em PFC (ver mais na seção 4.5)
R-13	**Alterando uma movimentação de dados** a) Se uma movimentação de dados deve ser alterada devido a uma mudança na manipulação de dados a ela associada, e/ou devido a uma mudança no número ou tipo de atributos no grupo de dados movimentado, um PFC alterado deve ser medido, independentemente do número de modificações na movimentação de dados. b) Se um grupo de dados deve ser modificado, as movimentações de dados movimentando o grupo de dados alterado cuja funcionalidade não for afetada pela alteração no grupo de dados não devem ser identificadas como movimentações de dados alteradas. NOTA 1: Uma modificação no valor de uma ocorrência, tal como uma alteração em um código membro de um atributo cujos valores são um sistema de codificação não é uma alteração no tipo do atributo. NOTA 2: Uma modificação em qualquer dado aparecendo nas telas de entrada ou saída que não seja relacionado a um objeto de interesse para um usuário funcional não deve ser identificada como um PFC alterado (ver seção 3.3.4 para exemplos de tais dados.)
R-14	**Rótulos das medições COSMIC** Um resultado de medição COSMIC deve ser designado como "***x*** CFP (v.***y***)", onde: • "**x**" representa o valor numérico do tamanho funcional,

ID	DESCRIÇÃO DA REGRA
	• "v.**y**" representa a identificação da versão do padrão COSMIC utilizado para obter o valor numérico do tamanho funcional "x". NOTA: Se um método de aproximação local foi utilizado para obter a medição, mas no demais a medição foi realizada utilizando as convenções de uma versão padrão COSMIC, a convenção acima deverá ser usada, mas a utilização do método de aproximação deverá ser anotada em outro lugar – ver seção 5.2.
R-15	**Rótulos em extensões locais COSMIC** Um resultado de medição COSMIC utilizando extensões locais deve ser designado como: "***x*** CFP (v. ***y***) + z PF Local", onde: • "**x**" representa o valor numérico obtido agregando-se todos os resultados individuais de medição de acordo com o método padrão COSMIC, versão v.y, • "v.**y**" representa a identificação da versão padrão do método COSMIC utilizada para obter o valor numérico do tamanho funcional "x". • "z" representa o valor numérico obtido agregando-se todos os resultados individuais de medição obtidos através de extensões locais para o método COSMIC.
R-16	**Reporte de medições COSMIC** Além das próprias medições, registradas conforme em 5.1, os seguintes atributos de cada medição deveriam ser registrados. a) Identificação do componente de software medido (nome, ID da versão ou ID da configuração). b) As fontes de informação utilizadas para identificar os RFU usados na medição c) O domínio do software d) Uma declaração do propósito da medição. e) Uma descrição do escopo da medição, bem como de sua relação com o escopo global de um conjunto de medições relacionadas, se existir. (Utilizar as categorias de escopo genéricas na seção 2.2) f) Os usuários funcionais do software g) O nível de granularidade do RFU e o nível de decomposição do software. h) O ponto no ciclo de vida do projeto onde a medição foi feita (especialmente se a medição for uma estimativa baseada em RFU incompletos, ou se foi feita com base em funcionalidade realmente entregue). i) O alvo ou margem de erro imaginada para a medição. j) Uma indicação se o método padrão de medição COSMIC foi utilizado e/ou uma aproximação local para o método padrão, e/ou extensões locais foram utilizadas (ver seção 4.5). Utilizar as convenções para rótulos das seções 5.1 ou 5.2. k) Uma indicação se a medição é de funcionalidade desenvolvida ou entregue (funcionalidade 'desenvolvida' é obtida criando-se novo software; funcionalidade 'entregue' inclui funcionalidade 'desenvolvida' e também inclui funcionalidade obtida por outros meios além da criação de novo software, i.e., incluindo-se todas as formas de reuso de software existente, utilização de parâmetros existentes para incluir ou alterar funcionalidades, etc.). l) Uma indicação se a medição é de funcionalidade nova ou é o resultado de uma atividade de 'melhoria' (i.e., a soma de funcionalidade incluída, alterada e excluída – ver 4.4). m) A descrição de uma arquitetura de camadas nas quais a medição é feita, se aplicável. n) O número de componentes principais, se aplicável, cujos tamanhos foram somados para obter o tamanho total registrado. o) Uma matriz de medição para cada escopo dentro do escopo global da medição, conforme especificado no apêndice A. p) O nome do medidor e quaisquer qualificações referentes à certificação COSMIC.

APÊNDICE D - HISTÓRIA DAS VERSÕES DO MÉTODO COSMIC

Este Apêndice contém um resumo das principais mudanças efetuadas na derivação da versão 3.0 e desta versão 3.0.1 do método de medição de tamanho funcional COSMIC a partir da versão 2.2. A Versão 2.2 do método foi completamente descrita no 'Manual de Medição v2.2' (abreviado 'MM'), mas da versão v3.0 em diante a documentação do método é agora distribuída em quatro documentos, apenas um dos quais é denominado 'Manual de Medição'.

O propósito deste Apêndice é permitir a um leitor familiarizado com o MM v2.2 acompanhar as mudanças feitas nas versões 3.0 e 3.0.1 e, onde necessário, entender sua justificativa. (Para as mudanças efetuadas na derivação da versão 2.2 a partir da versão 2.1, ver a versão 2.2 do Manual de Medição, Apêndice E.)

Da versão 2.2 para a versão 3.0

Na tabela abaixo, mostrando as principais mudanças efetuadas na derivação da v3.0 a partir da v2.2, são referenciados dois 'Boletins de Atualização do Método' (ou 'MUB's' – 'Method Update Bulletins'). Um MUB é publicado pelo COSMIC para propor melhorias no método, no intervalo de tempo entre publicações mais importantes sobre a definição do método. Os dois MUBs são:

- MUB 1 "Melhorias propostas para a definição e características de uma 'Camada' de software, publicado em maio de 2003.

- MUB 2 "Melhoria proposta para a definição de um 'objeto de interesse', publicado em março de 2005.

No processo de atualização do método da v2.2 para a v3.0, foi feito um esforço para racionalizar as orientações em 'princípios' e 'regras' e separar todos os exemplos dos princípios e regras. Mudanças como essas e várias melhorias editoriais não foram descritas a seguir.

Ref V2.2	Ref V3.0	Mudança
Reestruturação do Método COSMIC		
2.2, 2.7, 3.1, 3.2	1.5, Capítulo 2	Uma fase 'Estratégia de Medição' foi destacada como a primeira fase do que é agora um método composto por três fases. A fase Estratégia de Medição agora inclui a consideração de 'camadas', 'fronteiras' e 'usuários (funcionais)', que eram considerados parte da fase de Mapeamento no MM v2.2
Reestruturação do Manual de Medição		
		Durante a produção da v3.0 do Método COSMIC, o MM V2.2 foi dividido em quatro documentos para facilitar o uso • 'Método COSMIC v3.0: Visão Geral da Documentação e Glossário de Termos' (novo) • 'Método COSMIC v3.0: Visão Geral do Método' (Capítulos 1 e 2 do MM v2.2) • 'Método COSMIC v3.0: Manual de Medição' (Capítulos 3, 4 e 5 e os Apêndices do MM v2.2 • 'Método COSMIC v3.0: Tópicos Avançados e Relacionados'

		(Capítulos 6 e 7 do MM v2.2) As referências a artigos de suporte e de pesquisa foram removidas do MM. Estão agora disponíveis em www.cosmicon.com. As únicas referências que permaneceram nesses quatro documentos estão em notas de pé de página
Capítulo 4	Capítulo 4	O capítulo sobre a Fase de Medição foi consideravelmente reestruturado para fornecer uma exposição mais lógica
Apêndice D	--	Este Apêndice sobre 'Mais informações sobre camadas de software' foi removido, por ser incompatível com o MUB 1 e agregar pouco valor. Ver também as notas abaixo sobre as mudanças ligadas ao MUB 1
Mudanças de nomes e terminologia		
Geral		O nome do método 'COSMIC-FFP' foi simplificado para método 'COSMIC'
5.1	4.2	O nome da unidade de medida, 'medida de tamanho funcional COSMIC' (abreviada 'Cfsu') foi alterada para 'Ponto de Função COSMIC' (abreviada 'PFC'). Ver as Notas Introdutórias neste MM v3.0 para uma explicação desta mudança
4.1	4.1.7	A regra 'Desduplicação de movimentação de dados' foi renomeada regra da 'Unicidade na movimentação de dados'
Conceitos novos, substituídos e removidos		
2.7	2.3	Os conceitos de 'abstração', 'ponto de vista', 'Ponto de Vista da Medição', 'Ponto de Vista do Usuário Final na Medição' e 'Ponto de Vista do Desenvolvedor na Medição' foram removidos. Foram substituídos pelo conceito mais genérico de que o tamanho funcional de um pedaço de software a ser medido depende da funcionalidade disponível para o(s) 'usuário(s) final(is)' do software. Tais devem ser identificáveis nos Requisitos Funcionais do Usuário para o software a ser medido. A definição do(s) 'usuário(s) funcional(is)' é portanto um pré-requisito para a definição de qual tamanho deve ser medido ou para a interpretação de uma medida de tamanho existente. Ver as Notas Introdutórias a este MM v3.0 para uma explicação mais detalhada desta mudança
3.2	2.3	O conceito de 'usuário' (conforme definido na ISO/IEC 14143/1) foi substituído pelo conceito de 'usuário funcional' que é um conceito mais restrito. Ver as Notas Introdutórias a este MM v3.0 para uma explicação mais detalhada desta mudança. No item 2.3.2, dois exemplos foram introduzidos de quando o tamanho funcional varia com o tipo de usuário funcional definido no RFU, para um telefone celular e para um pacote de software de aplicação de negócio.
--	2.2.2	O 'nível de decomposição' do software a ser medido foi introduzido na discussão do escopo da medição
--	2.4	O 'nível de granularidade' dos requisitos funcionais do usuário do software a ser medido foi introduzido na discussão da estratégia de medição. Um exemplo abrangente é fornecido. Ver as Notas Introdutórias deste MM v3.0 para uma explicação mais detalhada para esta mudança
--	2.4.3	Um 'nível de granularidade do processo funcional' foi definido, assim como foram fornecidas regras e uma recomendação a respeito
3.4	4.2	O conceito de 'persistência de um grupo de dados' incluindo três níveis de persistência, nominalmente 'transiente', 'curto' e 'indefinido' foram removidos por terem sido demonstrados desnecessários. O conceito de 'armazenamento persistente' foi introduzido
4.1	--	O conceito de 'desduplicação' foi removido por ter sido demonstrado desnecessário
Definições, princípios e regras melhorados e refinados		
2.3	2.2	A definição de 'Requisitos Funcionais do Usuário' foi alterada para manter conformidade com a edição de 2007 da ISO/IEC 14143/1

2.4.1	1.3	O 'Modelo de Contexto de Software' foi ampliado e refinado como uma declaração de princípios
2.4.2	1.4	O 'Modelo Genérico de Software' foi ampliado e refinado como uma declaração de princípios
2.4, 3.1	2.2.3, 2.2.4, 3.1	A definição de 'camada' foi atualizada para levar em conta o MUB 1. As Figuras 2.4.1.1, 2.4.1.2 e 2.4.1.3 do MM v2.2 que foram usadas para ilustrar a interação dos usuários com o software em 'camadas' foram substituídas na v3.0 pelas Figuras 2.2.3.1 e 2.2.3.2 ilustrando arquiteturas em camadas típicas e as Figuras 3.1.1 e 3.1.2 ilustrando a interação lógica dos usuários funcionais com o software em camadas. O objetido desta mudança é distinguir mais claramente a visão *física* das arquiteturas de software em camadas típicas da visão *lógica* de um usuário funcional interagindo com um pedaço de software a ser medido de acordo com o modelo COSMIC
--	3.2.4	O conceito de 'componente par' foi definido e seus princípios foram estabelecidos levando em conta o MUB 1
2.5	4.2, 4.3	As 'Características' do processo de medição descritas na v2.2 foram reescritas como um conjunto de princípios e regras na v3.0
2.6	2.4	O material sobre 'dimensionamento no início da vida de um projeto: escalonamento da medição' foi tratado parcialmente no MM v3.0 seção 2.4 sob 'Nível de Granularidade' e está detalhado no documento 'Método COSMIC v3.0: Tópicos Avançados e Relacionados'
2.7	2.2	Uma nota foi acrescentada à definição de 'escopo' para distinguir o 'escopo global' de um exercício de medição (o qual pode incluir diversos pedaços distintos de software cujo tamanho precise ser medido) do 'escopo' de uma medição individual de tamanho.
3.2	4.1.8	Três Exemplos que ilustram a interação dos usuários através de uma fronteira com o software em diferentes camadas e de diferentes 'pontos de vista da medição' foram mudados de lugar e reescritos tendo em vista a remoção dos 'pontos de vista da medição'. Ver 4.1.8 (do MM v3.0) abaixo
--	3.1	Um princípio foi acrescentado a respeito da aplicação do Modelo Genérico de Software ao software a ser medido
3.3	3.2.1	A definição de 'processo funcional' foi refinada para levar em conta a introdução do conceito de 'usuário funcional' que substitui 'ator' nesta definição
3.3	3.2.1	A definição de 'evento disparador' foi refinada
--	3.2.1	A relação entre um evento disparador, um usuário funcional, uma Entry disparadora e um processo funcional foi clarificada na Figura 3.1.1
3.1	3.2.2	Os princípios e regras para um 'processo funcional' foram fundidos em um conjunto de regras revisadas
--	3.2.3, 3.2.4, 3.2.5	Muitos exemplos de processos funcionais e de como distinguí-los foram introduzidos
--	3.3.1	A definição de um 'objeto de interesse' foi introduzida, alinhada com o MUB 2
3.4	3.3.3	Exemplos da identificação de objetos de interesse e grupos de dados foram separados das regras para grupos de dados. Algum material de domínios específicos sobre convenções da análise de dados de entidades-relacionamentos foi transferido para o 'Guia para dimensionamento de Software de Aplicação de Negócio v1.0'.
--	3.3.4	Foram incluídas orientações sobre 'dados ou grupos de dados que não são candidatos a movimentações de dados'
--	3.3.5	Foram incluídas orientações sobre quando, tipicamente em software 'real-time', não deve valer a pena distinguir um 'usuário funcional' de um 'objeto de interesse' sobre o qual dados são movimentados
3.5	3.4	A discussão de 'atributos de dados' foi reduzida e simplificada já que a

		consideração de atributos de dados não é uma parte obrigatória do método
4.1	4.1.1	A definição de 'movimentação de dados' foi racionalizada
4.1	4.1.7	Foram esclarecidas as regras sobre 'desduplicação de movimentações de dados' (agora renomeadas como regras sobre 'unicidade na movimentação de dados e possíveis exceções') com a inclusão de vários exemplos
4.1	--	As regras para domínios específicos sobre as movimentações de dados Read e Write em Processos Funcionais de 'Atualização' foram transferidas para o 'Guia para o Dimensionamento de Software de Aplicações de Negócio v1.0'.
4.1	4.1.8	As 'Regras para correspondência de dados através de fronteiras' (que no MM v2.2 eram aplicáveis no 'Ponto de Vista do Desenvolvedor na Medição', agora eliminado) foram excluídas, mas os conceitos foram combinados com os Casos anteriormente fornecidos no MM v2.2 seção 3.2, para produzir uma nova seção sobre 'quando um processo funcional movimenta dados de ou para o armazenamento persistente'
4.1	4.1.6	Foi definida a 'Manipulação de dados' e incluído um princípio sobre 'manipulação de dados associada à movimentação de dados'. As orientações sobre a manipulação de dados associada aos diversos tipos de movimentações de dados foram expandidas
4.1.1	4.1.2	Os princípios e regras para uma Entry foram racionalizados. Um novo princípio foi acrescentado sobre funcionalidade 'solicitação para submeter'.
4.1.2	4.1.3	Os princípios e regras sobre uma Exit foram racionalizados, incluindo a remoção da referência ao 'ponto de vista do usuário final na medição'
4.1.3	4.1.4	Os princípios para um Read foram racionalizados. Um novo princípio foi acrescentado sobre funcionalidade 'solicitação de leitura'
4.1.4	4.1.5	Os princípios para um Write foram racionalizados
--	4.1.9	Foram incluídas novas regras sobre 'quando um processo funcional requer dados de um usuário funcional'
--	4.1.10	Uma definição e uma regra foram introduzidas para o conceito de um 'comando de controle' que é válido somente no domínio de software de aplicação de negócio
4.1.5	4.5	A discussão de 'extensões locais para o método' foi expandida
4.3	4.3	Os princípios de 'agregação de resultados da medição' foram alterados para 'regras' e expandidos para cobrir as regras para a obtenção do tamanho de um pedaço de software através da soma dos tamanhos dos seus componentes. O MM v2.2 referia-se a essas regras somente no contexto do 'ponto de vista do desenvolvedor'. Tal restrição foi removida
--	4.4	Uma nova seção sobre 'medição do tamanho de mudanças no software' e novas regras foram acrescentadas
5.1	5.1	As regras para rotular os resultados da medição foram alteradas para reconhecer a mudança da unidade de medida de 'Cfsu' para 'PFC'
5.2	5.2	As regras sobre 'reporte da medição' foram expandidas para listar mais itens
Ap. B	Ap. B	Atualizado para os Princípios da v3.0
Ap. C	Ap. C	Atualizado para as Regras da v3.0

Da versão 3.0 para a versão 3.0.1

As mudanças mais importantes na derivação da versão 3.0.1 a partir da versão 3.0 ocorreram onde foi desejável melhorar a escrita de algumas definições, princípios e regras, bem como algumas partes do texto. Com a exceção de um erro, as melhorias foram todas feitas por razão de clareza. A maior parte das mudanças foi publicada em três Boletins de Atualização do Método antes da versão 3.0.1.

- MUB 3: ‘Correção de um erro na Fig. 4.1.8.1 (b) do Manual de Medição do Método COSMIC v3.0’, publicado em junho de 2008
- MUB 4: ‘Clarificação do princípio e regras para funcionalidade ‘Solicitação de submissão’ no ‘Manual de Medição do Método COSMIC v3.0’, publicado em junho de 2008
- MUB 5: ‘Melhorias propostas para (a) Definições de ‘Nível de Decomposição’ e de ‘Componente Par’, e (b) Princípio (c) para um ‘Componente Par’, publicado em fevereiro de 2009

Diversas melhorias editoriais também foram efetuadas. Estas envolvem, principalmente, separar os Exemplos mais claramente do texto principal, com um tipo diferente de fonte e mais subseções.

Resumo das principais mudanças:

Ref V3.0.1	**Mudança**
2.2.3	A definição de ‘Nível de Decomposição’ foi alterada para maior clareza (MUB 5).
2.2.4	Na definição de ‘Camada’ o termo ‘arquitetura de software’ foi substituído por ‘sistema de software’, pois o termo ‘arquitetura’ implica que o software já está particionado. Regra (c) removida. Esta era uma regra geral de projeto de software e não específica de camadas e medição. A referência ao ‘Apêndice D’ na v3.0 estava incorreta.
2.2.5	A definição de ‘Par’ foi inserida e a definição de ‘Componente Par’ modificada para maior clareza. O princípio (c) para ‘Componente Par’ foi alterado para um princípio que é mais relevante para MFT. A Figura 2.2.5.1 foi incluída para esclarecer o relacionamento entre componentes pares e pedaços de software pares. (MUB 5).
2.3.2	A regra (c) para Fronteira foi alterada para maior clareza. (Consequência de MUB 5).
2.4.3	Na Figura 2.4.3.1, nas duas caixas do lado inferior direito, ‘método de pagamento’ foi alterado para ‘meios de pagamento’, pois este é um melhor termo para cheque, cartão de crédito, etc.
3.2.2	A regra (e) para um processo funcional foi restringida pelas palavras em itálico acrescentadas. Agora se lê: ‘Um processo funcional deve pertencer inteiramente ao *escopo de medição de um pedaço de* software em uma e apenas uma camada’. Esta restrição já existia no ‘Guia para Dimensionamento de Software de Aplicação de Negócio’ v1.1, mas é válida para software de qualquer domínio.
3.2.6	Acrescentada nova seção, com título ‘Os processos funcionais de componentes pares’. Este texto foi, em grande parte, obtido do ‘Guia para o Dimensionamento de Software Aplicativo de Negócio’ v1.1, mas é válido para software de qualquer domínio. Isto tem relação com a mudança na Regra (e) em 3.2.2.
4.1.2	O princípio (c) para uma Entry foi alterado para esclarecer como levar em conta uma funcionalidade ‘Solicitação de submissão’ (MUB 4).
4.1.7	A sentença de abertura desta seção sobre ‘Unicidade dos dados e possíveis exceções’ foi alterada para esclarecer o significado pretendido e torná-lo consistente com a Regra (a). O exemplo 2 também foi extensamente modificado para maior clareza.
4.1.8	A Figura 4.1.8.1 (b) foi alterada para corrigir um erro sobre como um software gerenciador de dispositivo interage com o hardware (MUB 3).
4.1.9	As regras para ‘Quando um processo funcional requer dados de um usuário funcional’ e os exemplos relacionados foram alterados para maior clareza (MUB 4).

APÊNDICE E - PROCEDIMENTO COSMIC PARA SOLICITAÇÃO DE MUDANÇA E COMENTÁRIOS

O Comitê de Práticas de Medição COSMIC ('COSMIC Measurement Practices Committee – MPC') está bastante disposto a receber 'feedback', comentários e, se necessário, solicitações de mudança para o método COSMIC. Este apêndice estabelece a forma de comunicação com o COSMIC MPC.

Todas as comunicações com o COSMIC MPC devem ser enviadas via e-mail para o seguinte endereço:

mpc-chair@cosmicon.com

Comentários e 'feedback' gerais e informais

Comentários informais e/ou 'feedback' sobre a documentação COSMIC, tais como quaisquer dificuldades no entendimento ou na aplicação do método COSMIC, sugestões para melhoria geral, etc., devem ser enviadas via e-mail para o endereço acima.

As mensagens serão registradas e normalmente reconhecidas dentro de duas semanas do recebimento. O MPC não garante que iniciará qualquer ação em função desses comentários gerais.

Soicitações formais de mudança

Quando o leitor da documentação COSMIC acredita que há um erro no texto, necessidade de esclarecimento, ou que alguma parte do texto precisa ser melhorada, uma solicitação formal de mudança ('change request' – CR) pode ser submetida.

As CRs formais serão registradas e reconhecidas dentro de duas semanas do respectivo recebimento. Cada CR receberá um número de série e circulará entre os membros do COSMIC MPC, um grupo mundial de especialistas no método COSMIC. Seu ciclo normal de revisão é de no mínimo um mês e pode levar mais se a CR for de difícil resolução.

O resultado da revisão poderá indicar que a CR será aceita, rejeitada, ou 'suspensa aguardando discussão adicional' (neste último caso, por exemplo, se houver uma dependência em relação a outra CR), e o resultado será comunicado ao autor da submissão assim que possível.

Uma CR formal será aceita somente se vier com todas as informações relacionadas a seguir.

- Nome, posição e organização da pessoa que submeteu a CR
- Informações de contato da pessoa que submeteu a CR
- Data da submissão
- Declaração geral do propósito da CR (por exemplo: 'necessidade de melhorar o texto…')
- Texto que necessita de mudança, substituição ou exclusão (ou uma referência clara ao mesmo)
- Texto proposto adicional ou substituto
- Explicação completa sobre por que a mudança é necessária

Um formulário para submissão de CR está disponível no sítio www.cosmicon.com.

A decisão do COSMIC MPC sobre o resultado da revisão de uma CR e, se aceita, em qual versão da documentação COSMIC a CR será aplicada, é final.

Questões sobre a aplicação do método COSMIC

O COSMIC MPC lamenta ser incapaz de responder questões relativas ao uso e aplicação do método COSMIC. Existem organizações comerciais que podem oferecer treinamento e consultoria, ou suporte de ferramentas para o método. Por gentileza consulte o sítio www.cosmicon.com para maiores detalhes.